Lecture 17: Unsupervised Learning

Alan Ritter

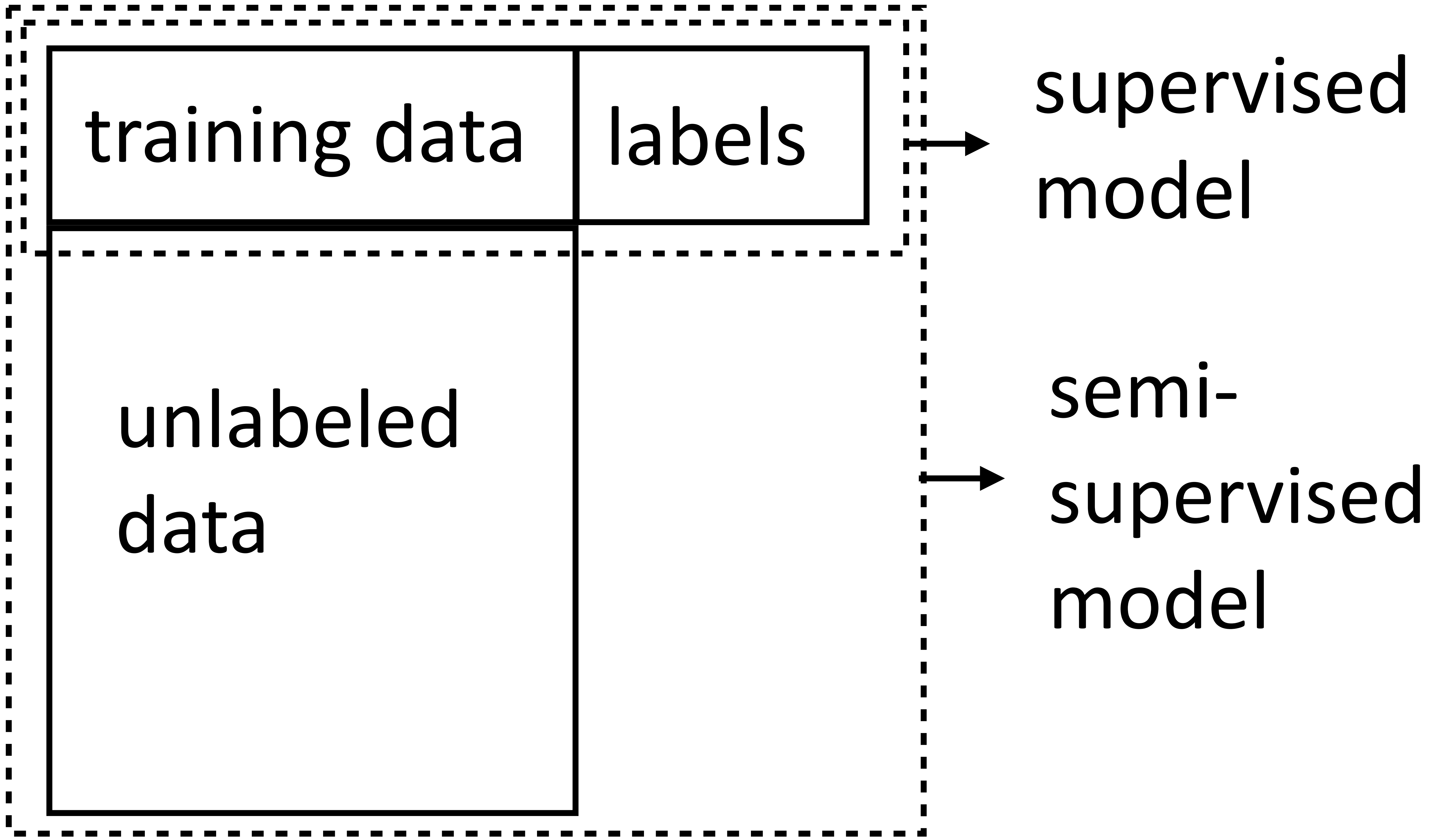
(many slides from Greg Durrett)

Administrivia

- Wei Xu will present on Friday
- No Class on December 4
- Final Project Presentations are during the final exam time scheduled on December 12

What data do we learn from?

- Supervised settings:
  - Tagging: POS, NER
  - Parsing: constituency, dependency, semantic parsing
  - IE, MT, QA, …
- Semi-supervised models
  - Word embeddings / word clusters (helpful for nearly all tasks)
  - Language models for machine translation
  - Learn linguistic structure from unlabeled data and use it?



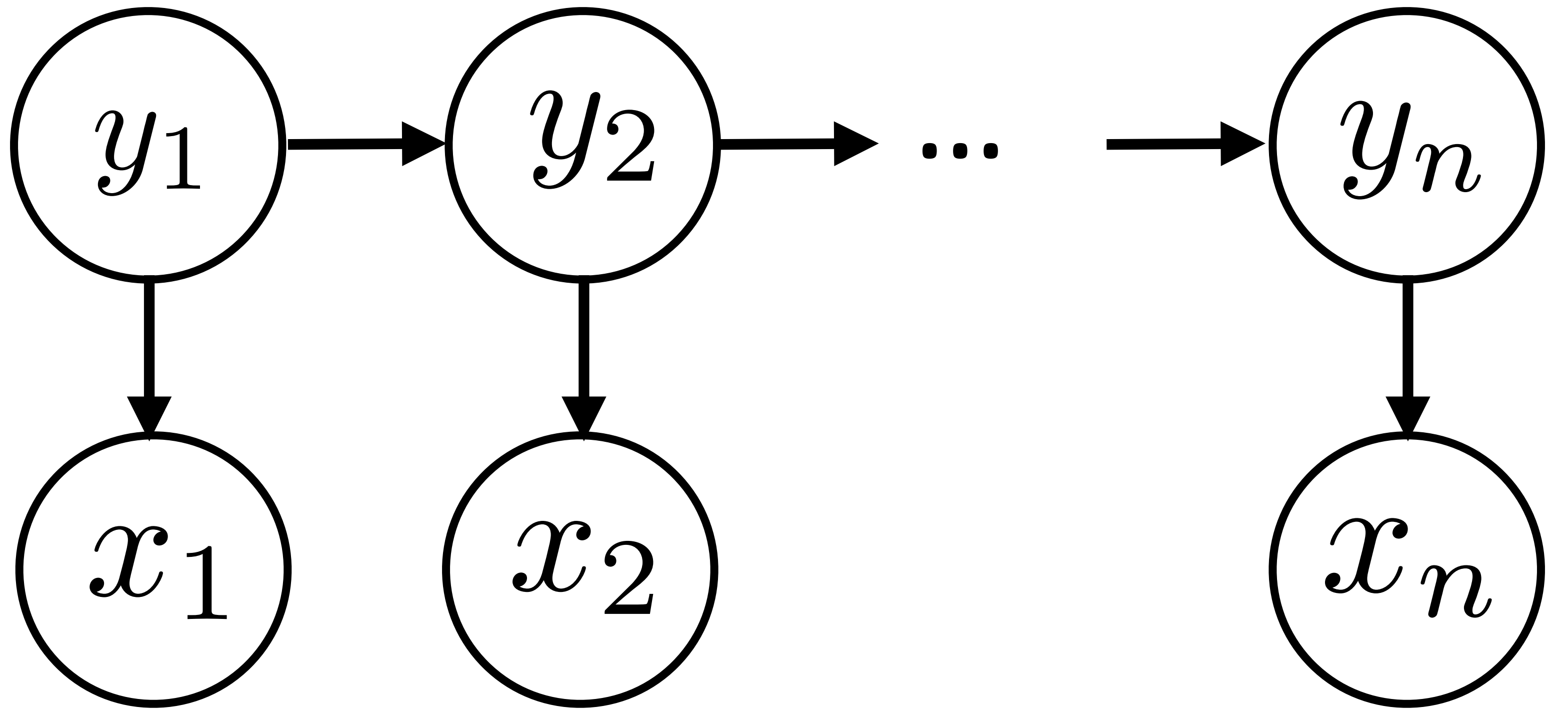
This Lecture

- Discrete linguistic structure from generative models: unsupervised POS induction
  - Expectation maximization for learning HMMs
- Continuous structure with generative models: variational autoencoders
- Continuous structure with "discriminative" models: transfer learning

EM for HMMs

Recall: Hidden Markov Models

- Input $\mathbf{x} = (x_1, ..., x_n)$ Output $\mathbf{y} = (y_1, ..., y_n)$



$$P(\mathbf{y}, \mathbf{x}) = \underbrace{P(y_1)}_{\text{Initial distribution}} \underbrace{\prod_{i=2}^{n} P(y_i|y_{i-1})}_{\text{Transition probabilities}} \underbrace{\prod_{i=1}^{n} P(x_i|y_i)}_{\text{Emission probabilities}}$$

- Observation (*x*) depends only on current state (*y*)
- Multinomials: tag x tag transitions, tag x word emissions
- P(*x*|*y*) is a distribution over all words in the vocabulary — not a distribution over features (but could be!)

Unsupervised Learning

- Can we induce linguistic structure? Thought experiment…

a b a c c c c

b a c c c

- What's a two-state HMM that could produce this?
- What if I show you this sequence?

a a b c c a a

- What did you do? Use current model parameters + data to refine your model. This is what EM will do

Part-of-Speech Induction

- Input $\mathbf{x} = (x_1, ..., x_n)$ Output $\mathbf{y} = (y_1, ..., y_n)$

- Assume we don’t have access to labeled examples — how can we learn a POS tagger?

- Key idea: optimize $P(\mathbf{x}) = \sum_{\mathbf{y}} P(\mathbf{y}, \mathbf{x}) \longleftarrow$ Generative model explains the data ***x***; the right HMM makes it look likely

- Optimizing marginal log-likelihood with no labels **y**:

$$\mathcal{L}(\mathbf{x}_{1,\ldots,D}) = \sum_{i=1}^{D} \log \sum_{\mathbf{y}} P(\mathbf{y}, \mathbf{x}_i)$$

  - non-convex optimization problem

Part-of-Speech Induction

- Input $\mathbf{x} = (x_1, ..., x_n)$ Output $\mathbf{y} = (y_1, ..., y_n)$
- Optimizing marginal log-likelihood with no labels **y**:

$$\mathcal{L}(\mathbf{x}_{1,\ldots,D}) = \sum_{i=1}^{D} \log \sum_{\mathbf{y}} P(\mathbf{y}, \mathbf{x}_i)$$

- Can't use a discriminative model; $\sum_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}) = 1$, doesn't model **x**
- What's the point of this? Model has inductive bias and so should learn some useful latent structure ***y*** (clustering effect)
- EM is just one procedure for optimizing this kind of objective

Expectation Maximization

$$\log \sum_{\mathbf{y}} P(\mathbf{x}, \mathbf{y} | \theta)$$

- Condition on parameters θ

$$= \log \sum_{\mathbf{y}} q(\mathbf{y}) \frac{P(\mathbf{x}, \mathbf{y} | \theta)}{q(\mathbf{y})}$$

- Variational approximation *q* — this is a trick we'll return to later!

$$\geq \sum_{\mathbf{y}} q(\mathbf{y}) \log \frac{P(\mathbf{x}, \mathbf{y} | \theta)}{q(\mathbf{y})}$$

- Jensen's inequality (uses concavity of log)

$$= \mathbb{E}_{q(\mathbf{y})} \log P(\mathbf{x}, \mathbf{y} | \theta) + \mathrm{Entropy}[q(\mathbf{y})]$$

- Can optimize this lower-bound on log likelihood instead of log-likelihood

Adapted from Leon Gu

Expectation Maximization

$$\log \sum_{\mathbf{y}} P(\mathbf{x}, \mathbf{y} | \theta) \geq \mathbb{E}_{q(\mathbf{y})} \log P(\mathbf{x}, \mathbf{y} | \theta) + \mathrm{Entropy}[q(\mathbf{y})]$$

- If $q(\mathbf{y}) = P(\mathbf{y} | \mathbf{x}, \theta)$, this bound ends up being tight
- Expectation-maximization: alternating maximization of the lower bound over *q* and θ
  - Current timestep = *t*, have parameters θ^{t-1}
  - E-step: maximize w.r.t. *q*; that is, $q^t = P(\mathbf{y} | \mathbf{x}, \theta^{t-1})$
  - M-step: maximize w.r.t. θ; that is, $\theta^t = \mathrm{argmax}_\theta \mathbb{E}_{q^t} \log P(\mathbf{x}, \mathbf{y} | \theta)$

Adapted from Leon Gu

EM for HMMs

- Expectation-maximization: alternating maximization
  - E-step: maximize w.r.t. *q*; that is, $q^t = P(\mathbf{y}|\mathbf{x}, \theta^{t-1})$
  - M-step: maximize w.r.t. θ; that is, $\theta^t = \mathrm{argmax}_\theta \mathbb{E}_{q^t} \log P(\mathbf{x}, \mathbf{y}|\theta)$

- E-step: for an HMM: run forward-backward with the given parameters

- Compute $P(y_i = s|\mathbf{x}, \theta^{t-1})$, $P(y_i = s_1, y_{i+1} = s_2|\mathbf{x}, \theta^{t-1})$

  tag marginals at each position

  tag pair marginals at each position

- M-step: set parameters to optimize the crazy argmax term

M-Step

- Recall how we maximized log P(**x**,**y**): read counts off data

DT	NN	→	count(DT, the) = 1	→	P(the|DT) = 1
the	*dog*		count(DT, dog) = 0		P(dog|DT) = 0
			count(NN, the) = 0		P(the|NN) = 0
			count(NN, dog) = 1		P(dog|NN) = 1

- Same procedure, but maximizing P(**x**,**y**) in expectation under *q* means that *q* specifies *fractional counts*

	the	*dog*	→	count(DT, the) = 0.9	→	P(the|DT) = 0.75
q	**DT: 0.9**	DT: 0.3		count(DT, dog) = 0.3		P(dog|DT) = 0.25
	NN: 0.1	**NN: 0.7**		count(NN, the) = 0.1		P(the|NN) = 0.125
				count(NN, dog) = 0.7		P(dog|NN) = 0.875

M-Step

- Same for transition probabilities

q

DT—NN: 0.6
DT—DT: 0.1
NN—DT: 0.2
NN—NN: 0.1

the *dog*

$\longrightarrow$

P(DT|DT) = 1/7
P(NN|DT) = 6/7
P(DT|NN) = 2/3
P(NN|NN) = 1/3

How does EM learn things?

- Initialize (M-step 0):
  - Emissions

P(the|DT) = **0.9**	P(the|NN) = 0.05
P(dog|DT) = 0.05	P(dog|NN) = **0.9**
P(marsupial|DT) = 0.05	P(marsupial|NN) = 0.05

  - Transition probabilities: uniform
- E-step 1: (all values are approximate)

DT: 0.95	DT: 0.05	**DT: 0.95**	$\boxed{\text{DT: 0.5}}$
NN: 0.05	**NN: 0.95**	NN: 0.05	$\boxed{\text{NN: 0.5}}$
the	*dog*	*the*	*marsupial*

- uniform

How does EM learn things?

- E-step 1:

DT: 0.95	DT: 0.05	**DT: 0.95**	$\boxed{\text{DT: 0.5}}$
NN: 0.05	**NN: 0.95**	NN: 0.05	$\boxed{\text{NN: 0.5}}$
the	*dog*	*the*	*marsupial*

- M-step 1:
  - Emissions aren't so different
  - Transition probabilities (approx): P(NN|DT) = 3/4, P(DT|DT) = 1/4

How does EM learn things?

- E-step 2:

DT: 0.95	DT: 0.05	**DT: 0.95**	$\boxed{\text{DT: 0.30}}$
NN: 0.05	**NN: 0.95**	NN: 0.05	$\boxed{\textbf{NN: 0.70}}$
the	*dog*	*the*	*marsupial*

- M-step 1:
  - Emissions aren't so different
  - Transition probabilities (approx): P(NN|DT) = 3/4, P(DT|DT) = 1/4

How does EM learn things?

- E-step 2:

DT: 0.95	DT: 0.05	**DT: 0.95**	DT: 0.30
NN: 0.05	**NN: 0.95**	NN: 0.05	**NN: 0.70**
the	*dog*	*the*	*marsupial*

- M-step 2:
  - Emission P(marsupial|NN) > P(marsupial|DT)
  - Remember to tag marsupial as NN in the future!
  - Context constrained what we learned! That’s how data helped us

How does EM learn things?

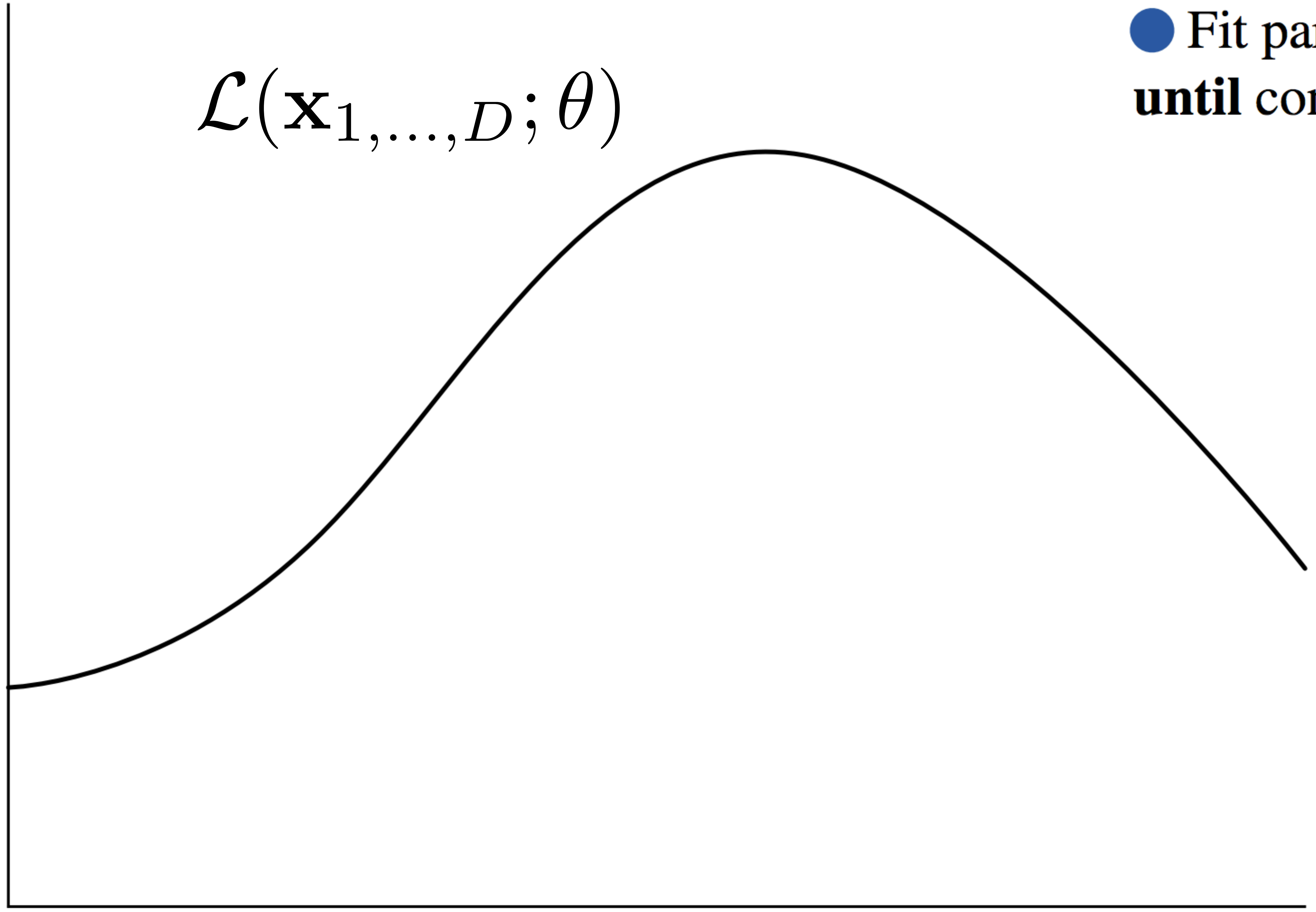
- Can think of q as a kind of "fractional annotation"
- E-step: compute annotations (posterior under current model)
- M-step: supervised learning with those fractional annotations
- Initialize with some reasonable weights, alternate E and M until convergence

EM's Lower Bound

$$\mathcal{L}(\mathbf{x}_{1,\ldots,D}) = \sum_{i=1}^{D} \log \sum_{\mathbf{y}} P(\mathbf{y}, \mathbf{x}_i)$$

Initialize probabilities $\boldsymbol{\theta}$
repeat
- Compute expected counts $\mathbf{e}$
- Fit parameters $\boldsymbol{\theta}$

until convergence



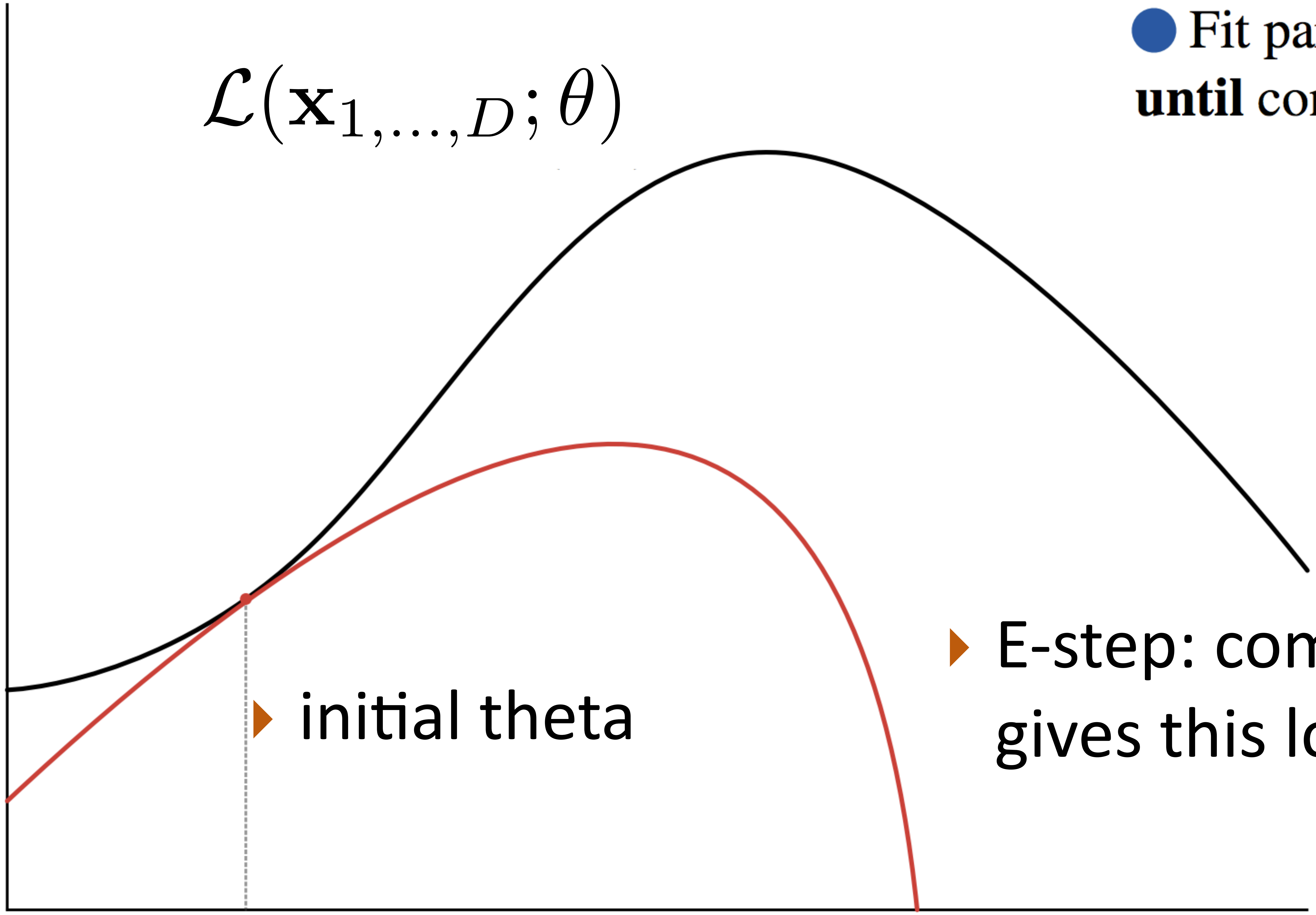
slide credit: Taylor Berg-Kirkpatrick

EM's Lower Bound

$$\mathcal{L}(\mathbf{x}_{1,\ldots,D}) = \sum_{i=1}^{D} \log \sum_{\mathbf{y}} P(\mathbf{y}, \mathbf{x}_i)$$

Initialize probabilities $\boldsymbol{\theta}$
repeat
- Compute expected counts $\mathbf{e}$
- Fit parameters $\boldsymbol{\theta}$

until convergence



- E-step: compute *q* which gives this lower bound

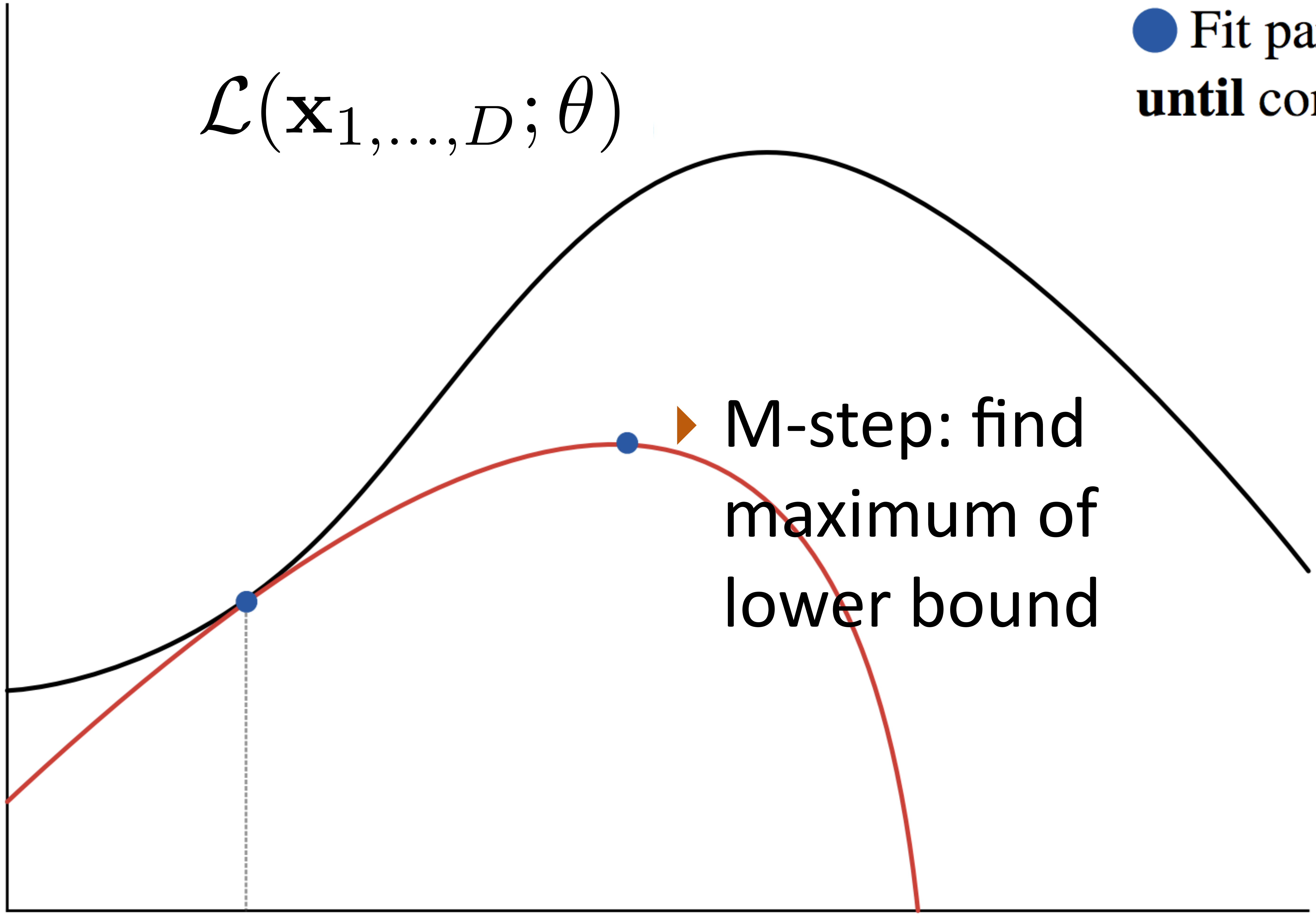
slide credit: Taylor Berg-Kirkpatrick

EM's Lower Bound

$$\mathcal{L}(\mathbf{x}_{1,\ldots,D}) = \sum_{i=1}^{D} \log \sum_{\mathbf{y}} P(\mathbf{y}, \mathbf{x}_i)$$

Initialize probabilities $\boldsymbol{\theta}$
repeat
- Compute expected counts $\mathbf{e}$
- Fit parameters $\boldsymbol{\theta}$

until convergence



- M-step: find maximum of lower bound

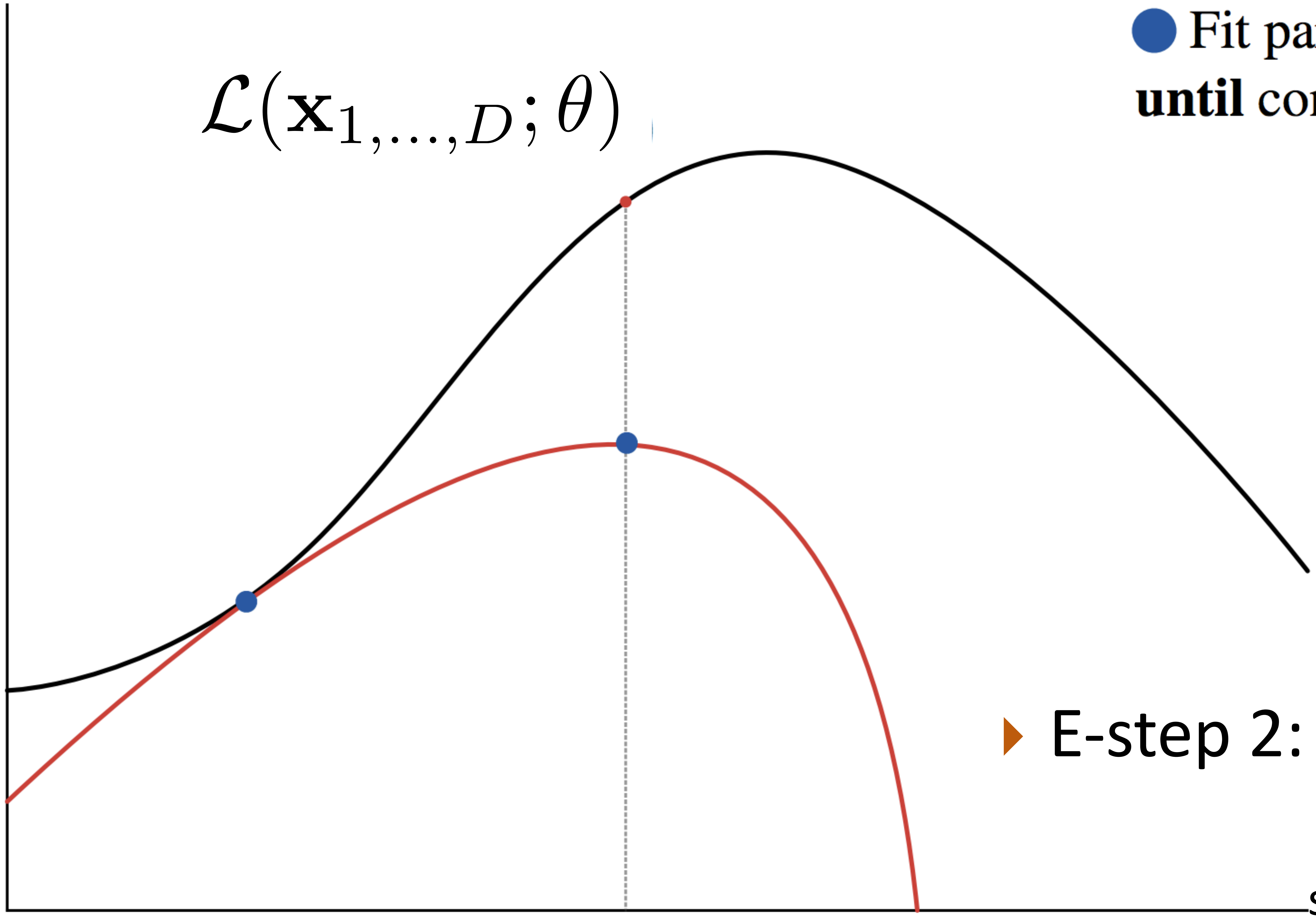
slide credit: Taylor Berg-Kirkpatrick

EM's Lower Bound

$$\mathcal{L}(\mathbf{x}_{1,\ldots,D}) = \sum_{i=1}^{D} \log \sum_{\mathbf{y}} P(\mathbf{y}, \mathbf{x}_i)$$

Initialize probabilities $\boldsymbol{\theta}$
repeat
- Compute expected counts $\mathbf{e}$
- Fit parameters $\boldsymbol{\theta}$

until convergence



- E-step 2: re-estimate *q*

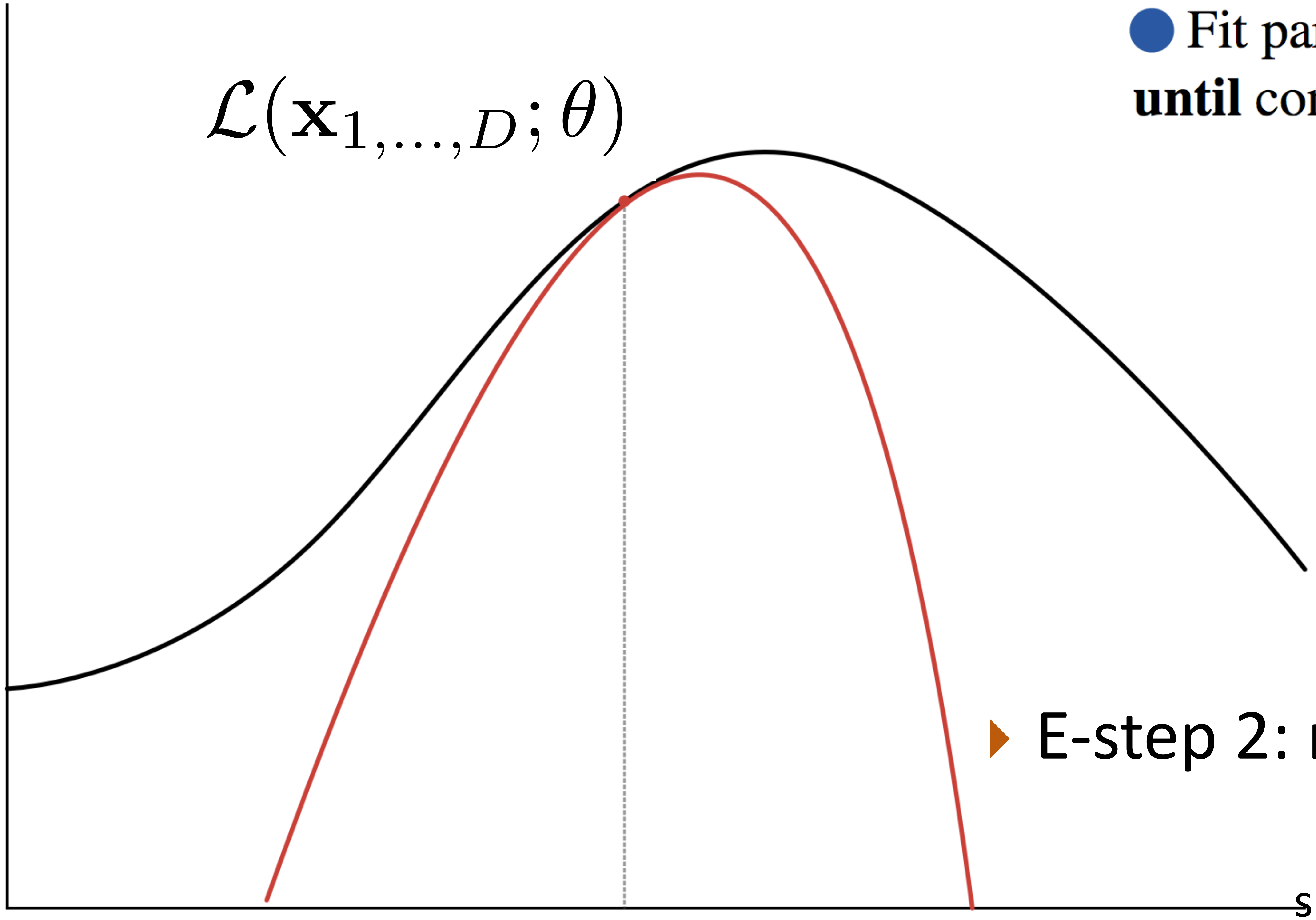
slide credit: Taylor Berg-Kirkpatrick

EM's Lower Bound

$$\mathcal{L}(\mathbf{x}_{1,\ldots,D}) = \sum_{i=1}^{D} \log \sum_{\mathbf{y}} P(\mathbf{y}, \mathbf{x}_i)$$

Initialize probabilities $\boldsymbol{\theta}$
repeat
- Compute expected counts $\mathbf{e}$
- Fit parameters $\boldsymbol{\theta}$

until convergence



- E-step 2: re-estimate *q*

slide credit: Taylor Berg-Kirkpatrick

EM's Lower Bound

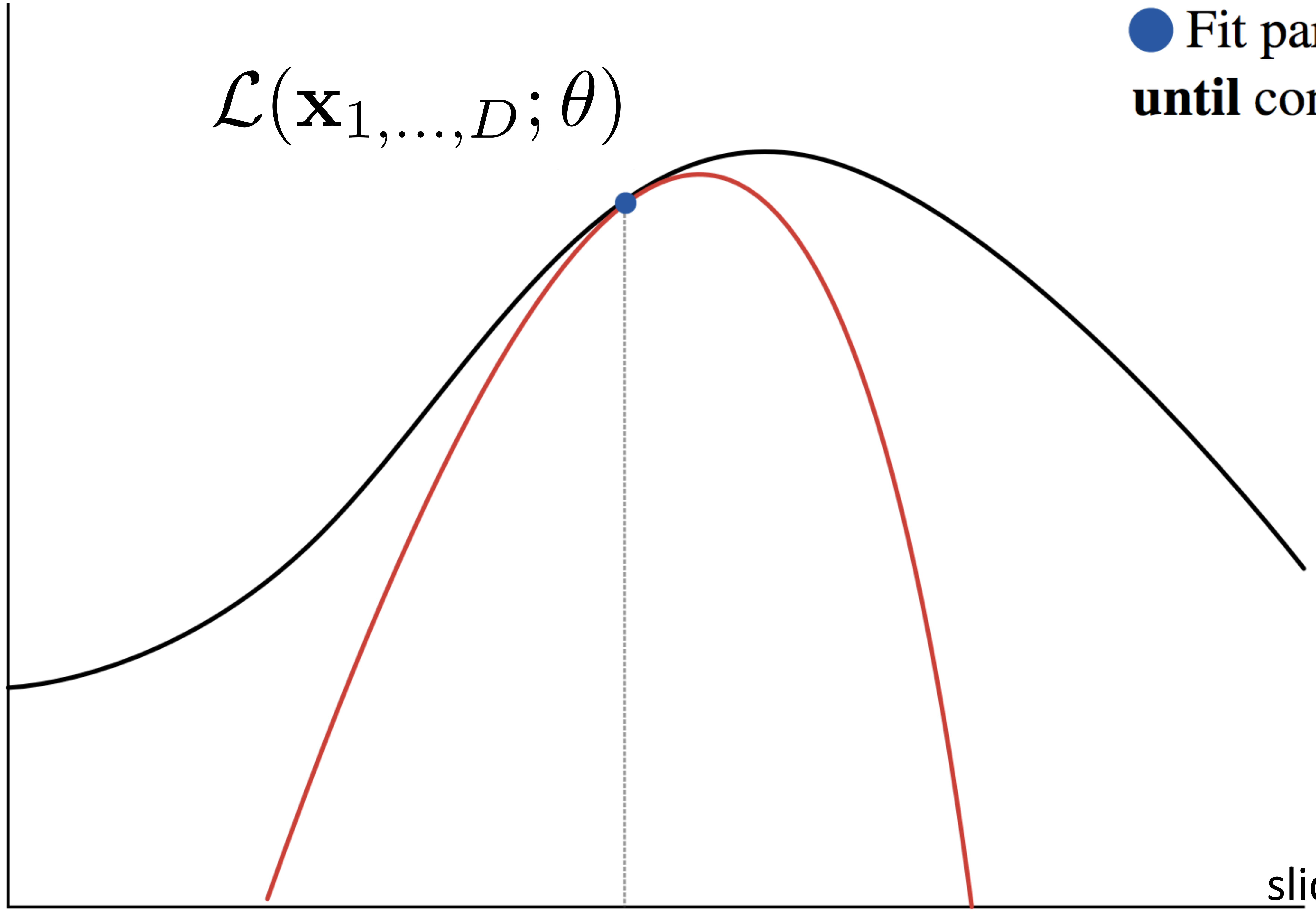
$$\mathcal{L}(\mathbf{x}_{1,\ldots,D}) = \sum_{i=1}^{D} \log \sum_{\mathbf{y}} P(\mathbf{y}, \mathbf{x}_i)$$

Initialize probabilities $\boldsymbol{\theta}$

repeat

- Compute expected counts $\mathbf{e}$
- Fit parameters $\boldsymbol{\theta}$

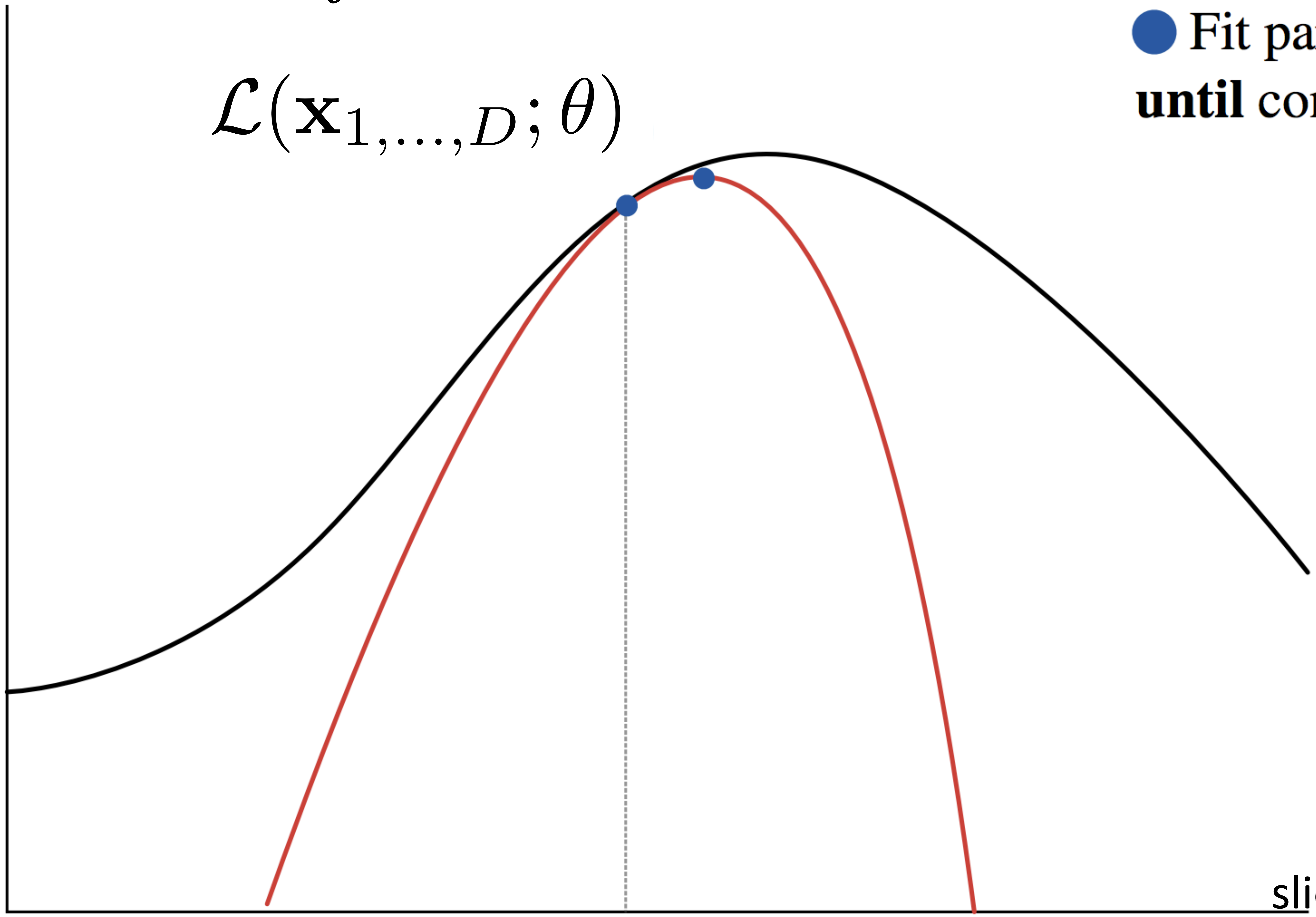
until convergence



slide credit: Taylor Berg-Kirkpatrick

EM's Lower Bound

$$\mathcal{L}(\mathbf{x}_{1,\ldots,D}) = \sum_{i=1}^{D} \log \sum_{\mathbf{y}} P(\mathbf{y}, \mathbf{x}_i)$$



Initialize probabilities $\boldsymbol{\theta}$
repeat
- Compute expected counts $\mathbf{e}$
- Fit parameters $\boldsymbol{\theta}$

until convergence

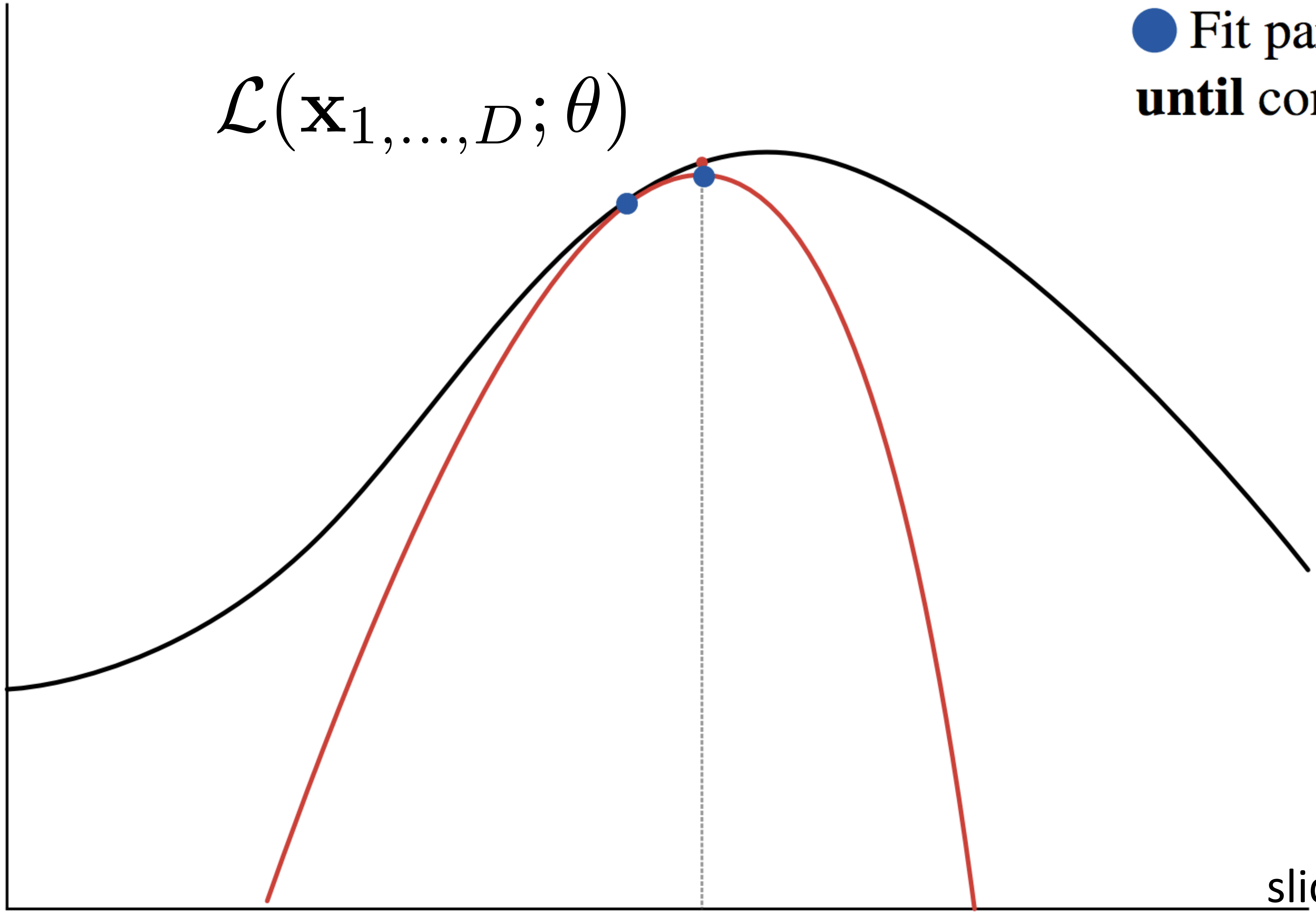
slide credit: Taylor Berg-Kirkpatrick

EM's Lower Bound

$$\mathcal{L}(\mathbf{x}_{1,\ldots,D}) = \sum_{i=1}^{D} \log \sum_{\mathbf{y}} P(\mathbf{y}, \mathbf{x}_i)$$

Initialize probabilities $\boldsymbol{\theta}$
repeat
- Compute expected counts $\mathbf{e}$
- Fit parameters $\boldsymbol{\theta}$

until convergence



slide credit: Taylor Berg-Kirkpatrick

Part-of-speech Induction

- Merialdo (1994): you have a whitelist of tags for each word
- Learn parameters on *k* examples to start, use those to initialize EM, run on 1 million words of unlabeled data
- Tag dictionary + data should get us started in the right direction…

Part-of-speech Induction

	Number of tagged sentences used for the initial model						
	0	100	2000	5000	10000	20000	all
Iter	Correct tags (% words) after ML on 1M words						
0	77.0	90.0	95.4	96.2	96.6	96.9	97.0
1	80.5	92.6	95.8	96.3	96.6	96.7	96.8
2	81.8	93.0	95.7	96.1	96.3	96.4	96.4
3	83.0	93.1	95.4	95.8	96.1	96.2	96.2
4	84.0	93.0	95.2	95.5	95.8	96.0	96.0
5	84.8	92.9	95.1	95.4	95.6	95.8	95.8
6	85.3	92.8	94.9	95.2	95.5	95.6	95.7
7	85.8	92.8	94.7	95.1	95.3	95.5	95.5
8	86.1	92.7	94.6	95.0	95.2	95.4	95.4
9	86.3	92.6	94.5	94.9	95.1	95.3	95.3
10	86.6	92.6	94.4	94.8	95.0	95.2	95.2

- Small amounts of data > large amounts of unlabeled data
- Running EM *hurts* performance once you have labeled data

Merialdo (1994)

Two Hours of Annotation

Human Annotations	0. No EM			1. EM only			2. With LP		
Initial data	T	K	U	T	K	U	T	K	U
KIN tokens A	**72**	90	58	55	82	32	71	86	58
KIN types A				63	77	32	78	83	69
MLG tokens B	74	89	49	68	87	39	74	89	49
MLG types B				71	87	46	72	81	57
ENG tokens A	63	83	38	62	83	37	**72**	85	55
ENG types A				66	76	37	75	81	56
ENG tokens B	70	87	44	70	87	43	**78**	90	60
ENG types B				69	83	38	75	82	61

- Kinyarwanda and Malagasy (two actual low-resource languages)
- Label propagation (technique for using dictionary labels) helps a lot, with data that was collected in two hours

Garrette and Baldridge (2013)

Variational Autoencoders

Continuous Latent Variables

- For discrete latent variables ***y***, we optimized: $P(\mathbf{x}) = \sum_{\mathbf{y}} P(\mathbf{y}, \mathbf{x})$
- What if we want to use continuous latent variables?

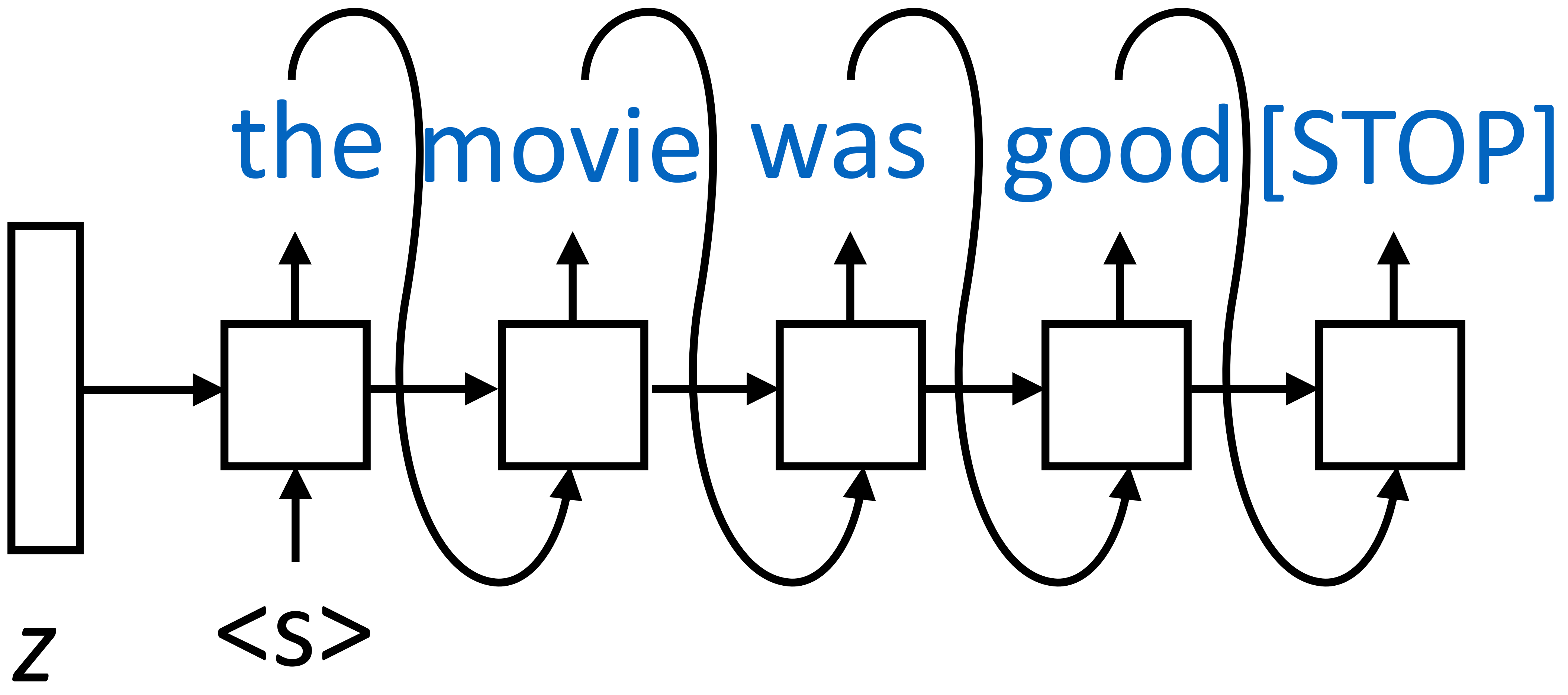
$$P(z, \mathbf{x}) = P(z)P(\mathbf{x}|z)$$

$$P(\mathbf{x}) = \int P(z)P(\mathbf{x}|z)\partial z$$

- Can use EM here when P(*z*) and P(***x***|z) are Gaussians
- What if we want P(***x***|*z*) to be something more complicated, like an LSTM with *z* as the initial state?

Deep Generative Models

$$P(z, \mathbf{x}) = P(z)P(\mathbf{x}|z)$$



- *z* is a latent variable which should control the generation of the sentence, maybe capture something about its topic

Deep Generative Models

$$\log \int_z P(\mathbf{x}, z|\theta) = \log \int_z q(z) \frac{P(\mathbf{x}, z|\theta)}{q(z)} \geq \int_z q(z) \log \frac{P(\mathbf{x}, z|\theta)}{q(z)}$$

Jensen

$$= \mathbb{E}_{q(z|\mathbf{x})}[-\log q(z|\mathbf{x}) + \log P(\mathbf{x}, z|\theta)]$$

$$= \mathbb{E}_{q(z|\mathbf{x})}[\log P(\mathbf{x}|z, \theta)] - \mathrm{KL}(q(z|\mathbf{x}) \| P(z))]$$

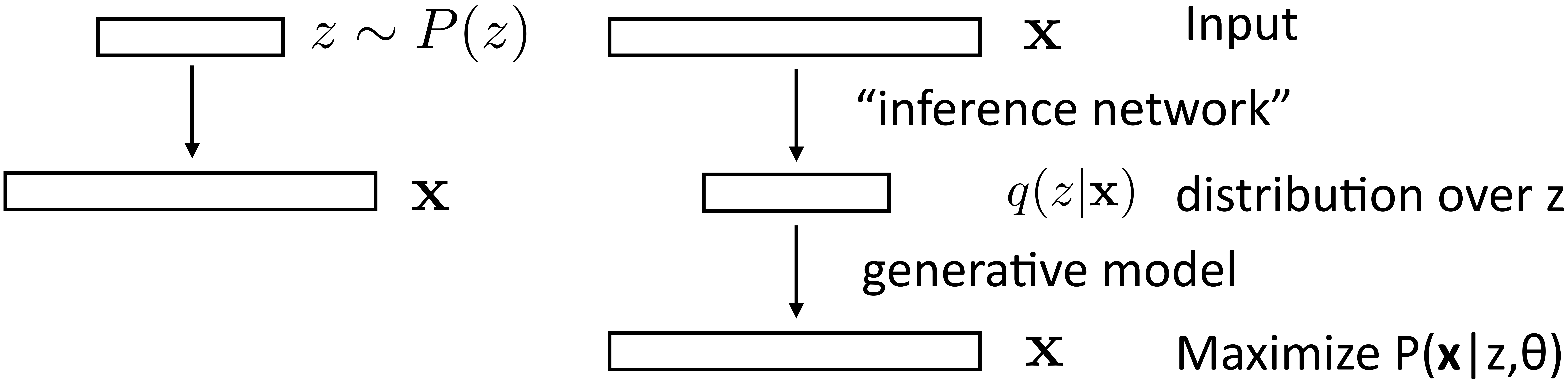
"make the data likely under q" "make q close to the prior"
(discriminative)

- KL divergence: distance metric over distributions (more dissimilar <=> higher KL)

Variational Autoencoders

$$\mathbb{E}_{q(z|\mathbf{x})}[\log P(\mathbf{x}|z,\theta)] - \mathrm{KL}(q(z|\mathbf{x}) \| P(z))$$

Generative model (test): Autoencoder (training):



Miao et al. (2015)

Training VAEs

$$\mathbb{E}_{q(z|\mathbf{x})}[\log P(\mathbf{x}|z,\theta)] - \mathrm{KL}(q(z|\mathbf{x}) \| P(z))$$

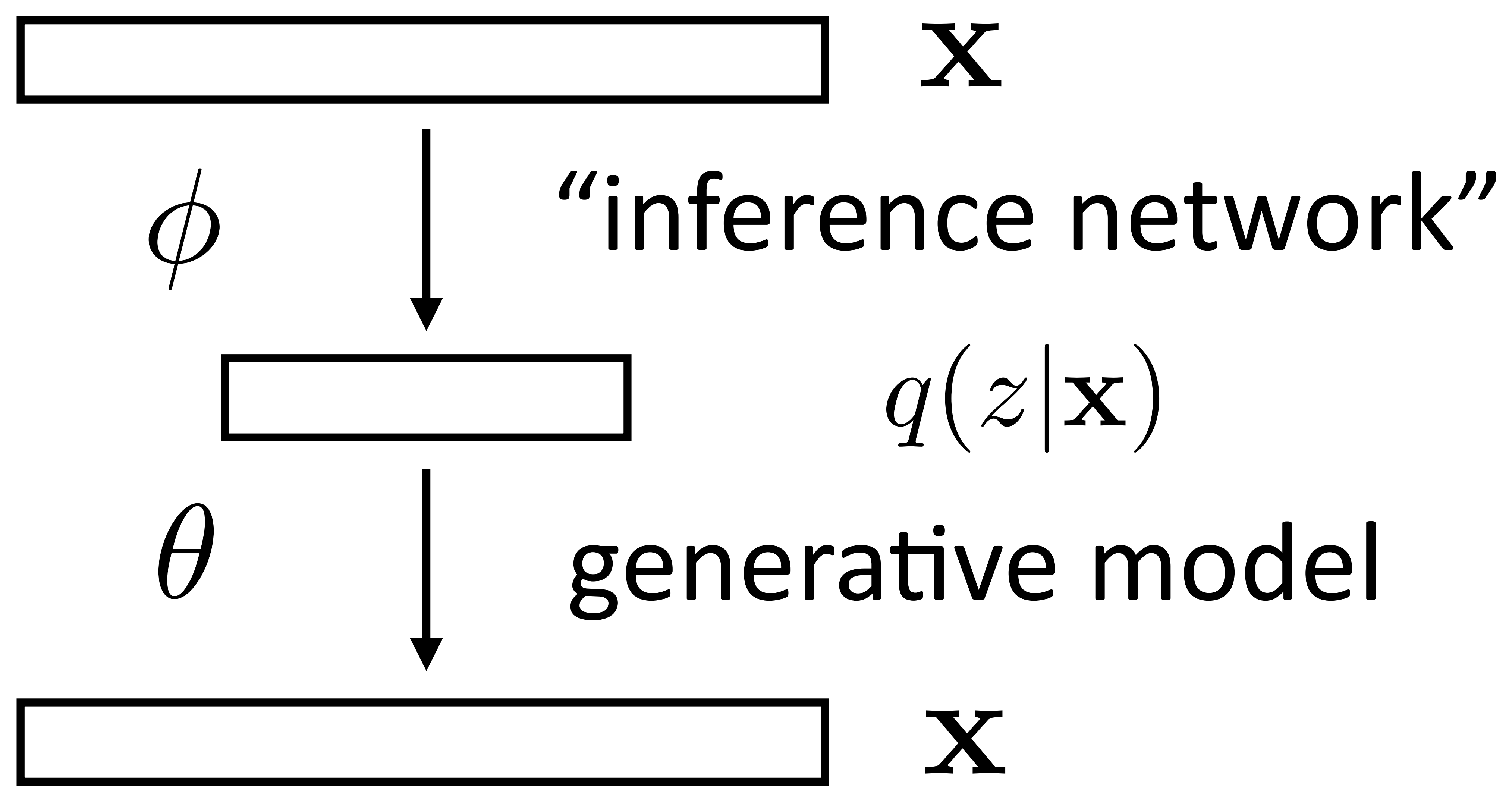
- Choose q to be Gaussian with parameters that are computed from **x**

$$q = N(\mu(\mathbf{x}), \mathrm{diag}(\sigma^2(\mathbf{x})))$$

- mu and sigma are computed from an LSTM over **x**, call their parameters ϕ

- How to handle the expectation? Sampling

Autoencoder (training):



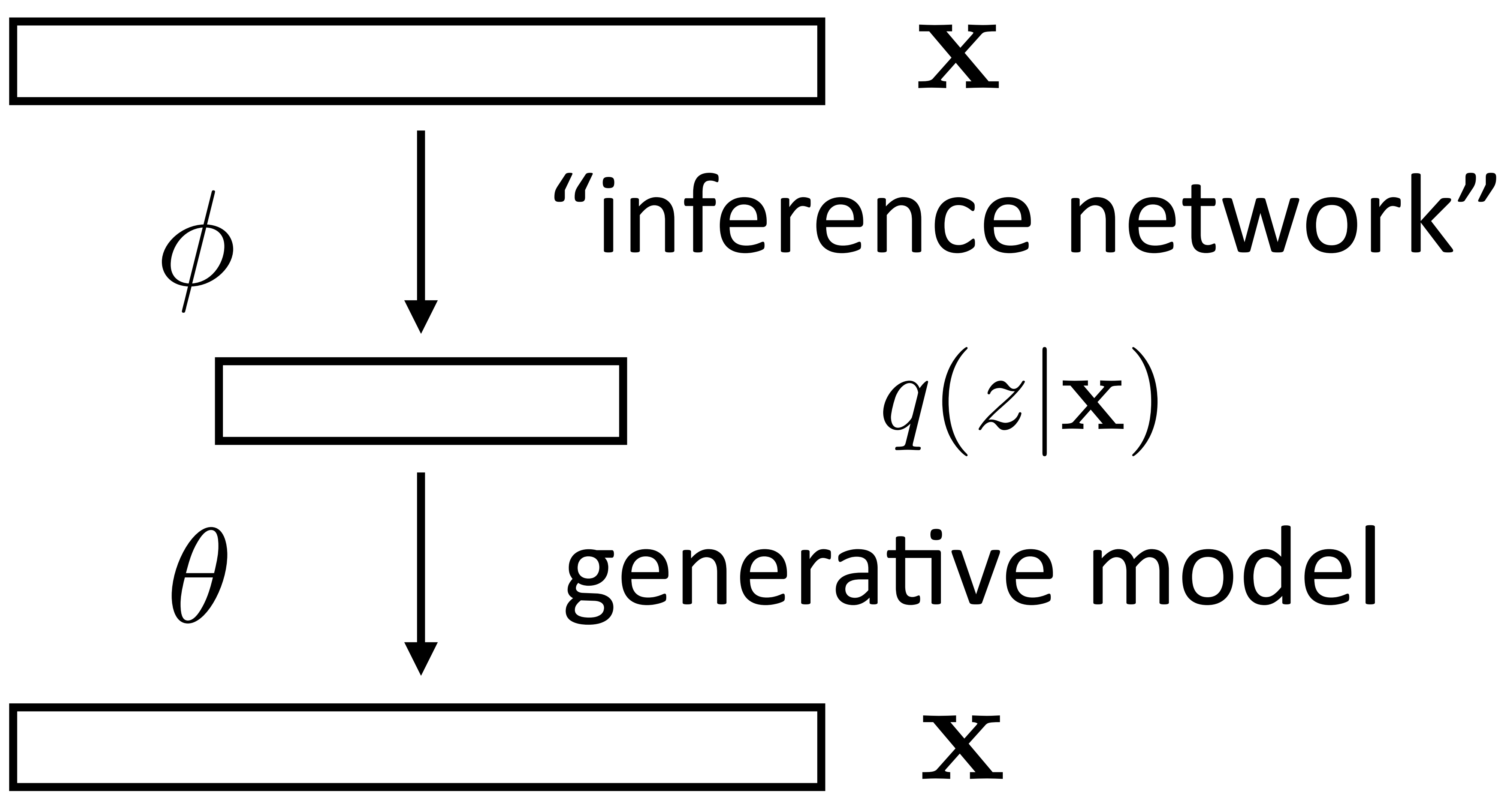
Miao et al. (2015)

Training VAEs

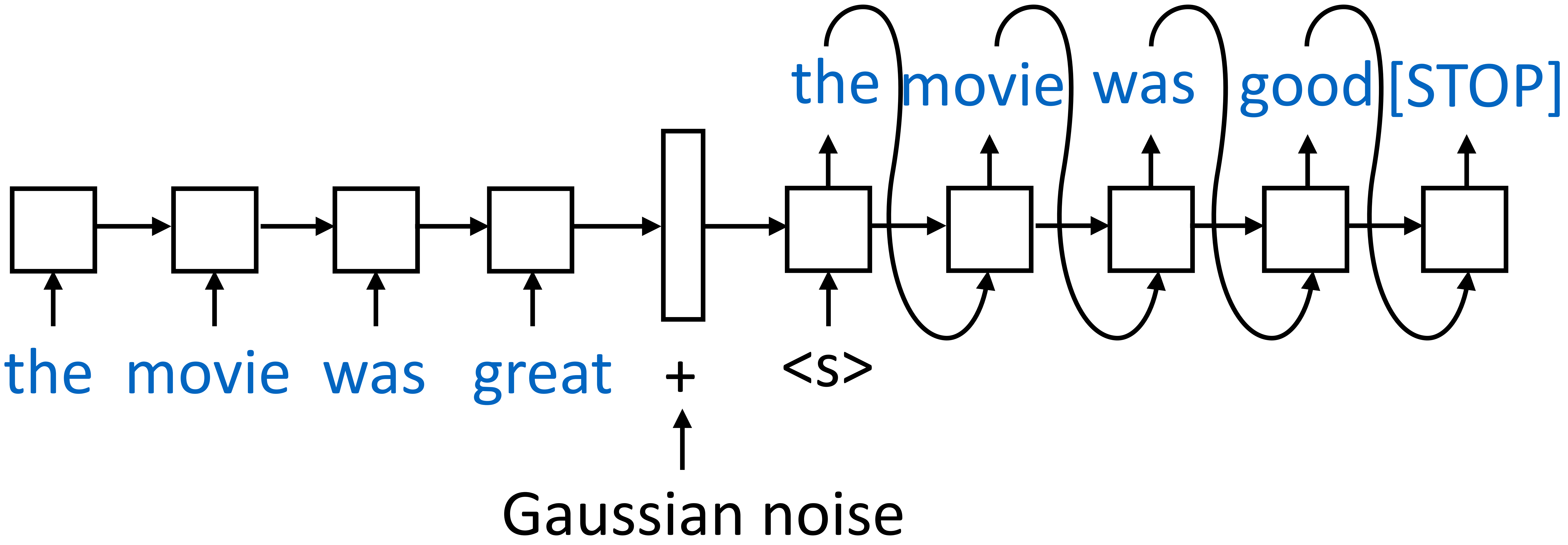
For each example **x**

- Compute *q* (run forward pass to compute mu and sigma)
- For some number of samples
  - Sample z ~ *q*
  - Compute P(**x**|z) and compute loss
  - Backpropagate to update phi, theta

Autoencoder (training):



Autoencoders

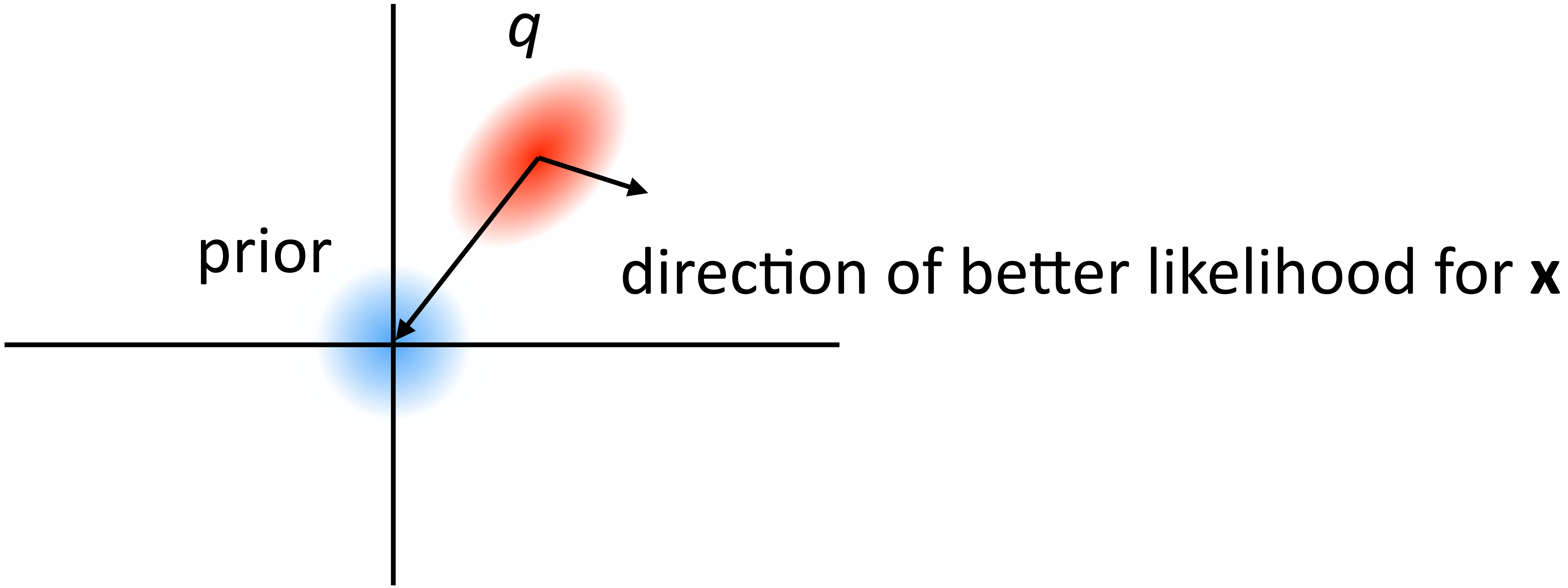


- Another interpretation: train an autoencoder and add Gaussian noise
- Same computation graph as VAE, add KL divergence term to make the objective the same
- Inference network (q) is the encoder and generator is the decoder

Visualization

$$\mathbb{E}_{q(z|\mathbf{x})}[\log P(\mathbf{x}|z,\theta)] + \mathrm{KL}(q(z|\mathbf{x})\|P(z))$$

- What does gradient encourage latent space to do?



What do VAEs do?

- Let us encode a sentence and generate similar sentences:

INPUT	**we looked out at the setting sun .**	**i went to the kitchen .**	**how are you doing ?**
MEAN	*they were laughing at the same time .*	*i went to the kitchen .*	*what are you doing ?*
SAMP. 1	*ill see you in the early morning .*	*i went to my apartment .*	*" are you sure ?*
SAMP. 2	*i looked up at the blue sky .*	*i looked around the room .*	*what are you doing ?*
SAMP. 3	*it was down on the dance floor .*	*i turned back to the table .*	*what are you doing ?*

- Style transfer: also condition on sentiment, change sentiment

Positive	great indoor mall .
⇒ ARAE	no smoking mall .
⇒ Cross-AE	terrible outdoor urine .
Positive	it has a great atmosphere , with wonderful service .
⇒ ARAE	it has no taste , with a complete jerk .
⇒ Cross-AE	it has a great horrible food and run out service .

- ...or use the latent representations for semi-supervised learning

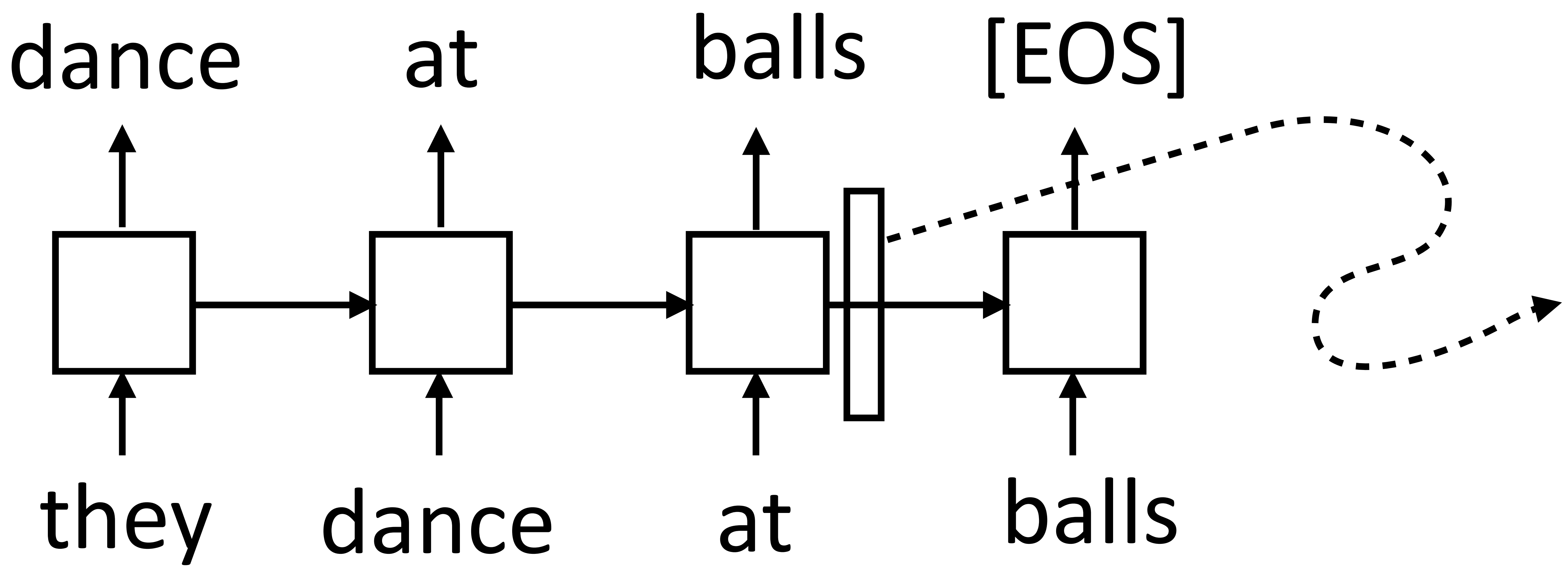
Bowman et al. (2016), Zhao et al. (2017)

BERT

Goals of Unsupervised Learning

- We want to use unlabeled data, but EM “requires” generative models. Are models like this really necessary?
- word2vec: predict nearby word given context. This wasn’t generative, but the supervision is free…
- Language modeling is a “more contextualized” form of word2vec

ELMo



learn a linear classifier on top of this vector to get a POS tagger with 97.3% accuracy (~SOTA)

$$P(x_i | x_1, \ldots, x_{i-1}) = \mathrm{LSTM}(x_1, \ldots, x_{i-1})$$

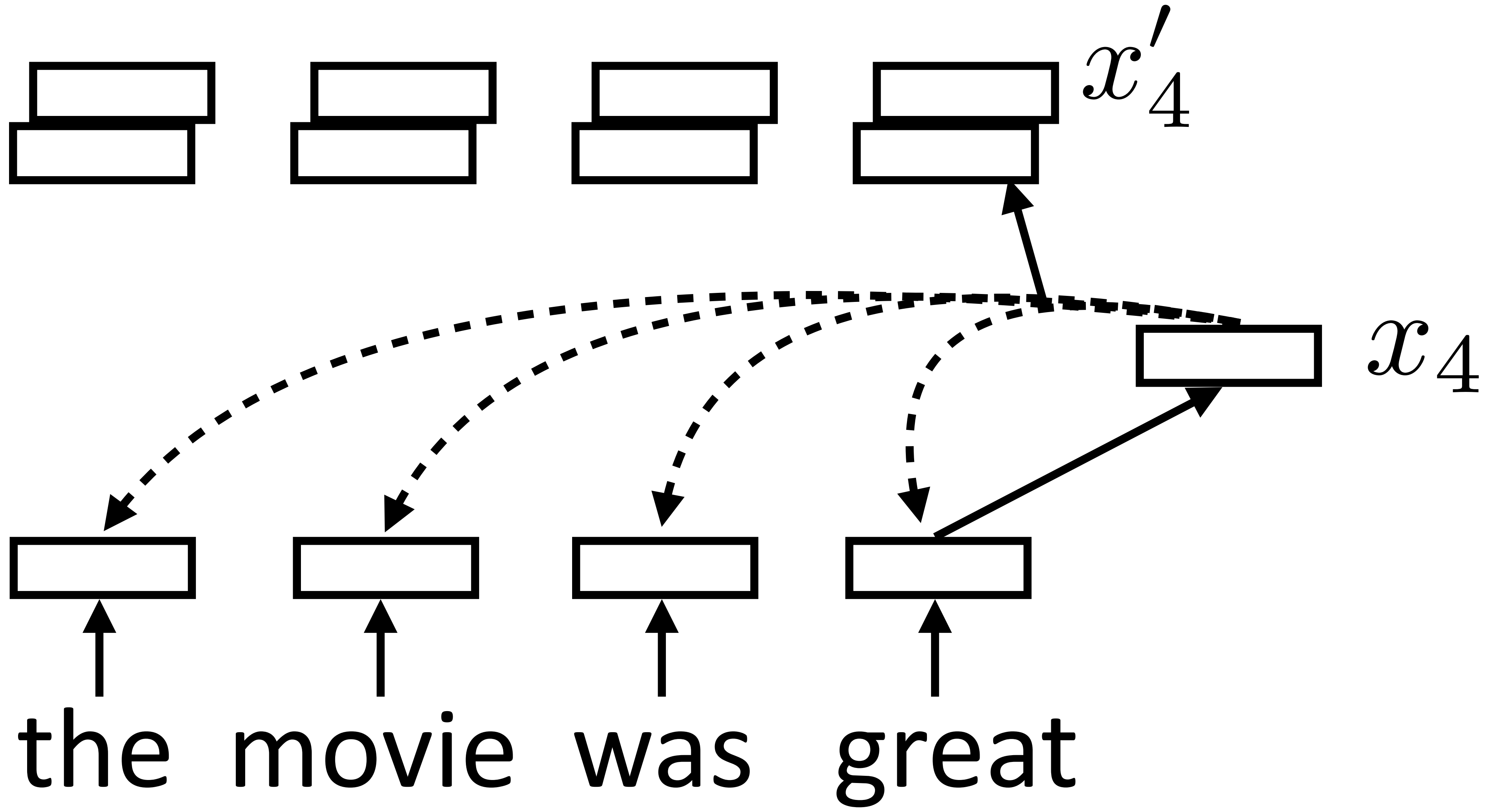
- Generative model of the data!
- Train one model in each direction on 1B words, use the LSTM hidden states as context-aware token representations

Recall: Self-Attention

- Each word forms a “query” which then computes attention over each word

$$\alpha_{i,j} = \mathrm{softmax}(x_i^\top x_j)$$ scalar

$$x'_i = \sum_{j=1}^{n} \alpha_{i,j} x_j$$ vector = sum of scalar * vector

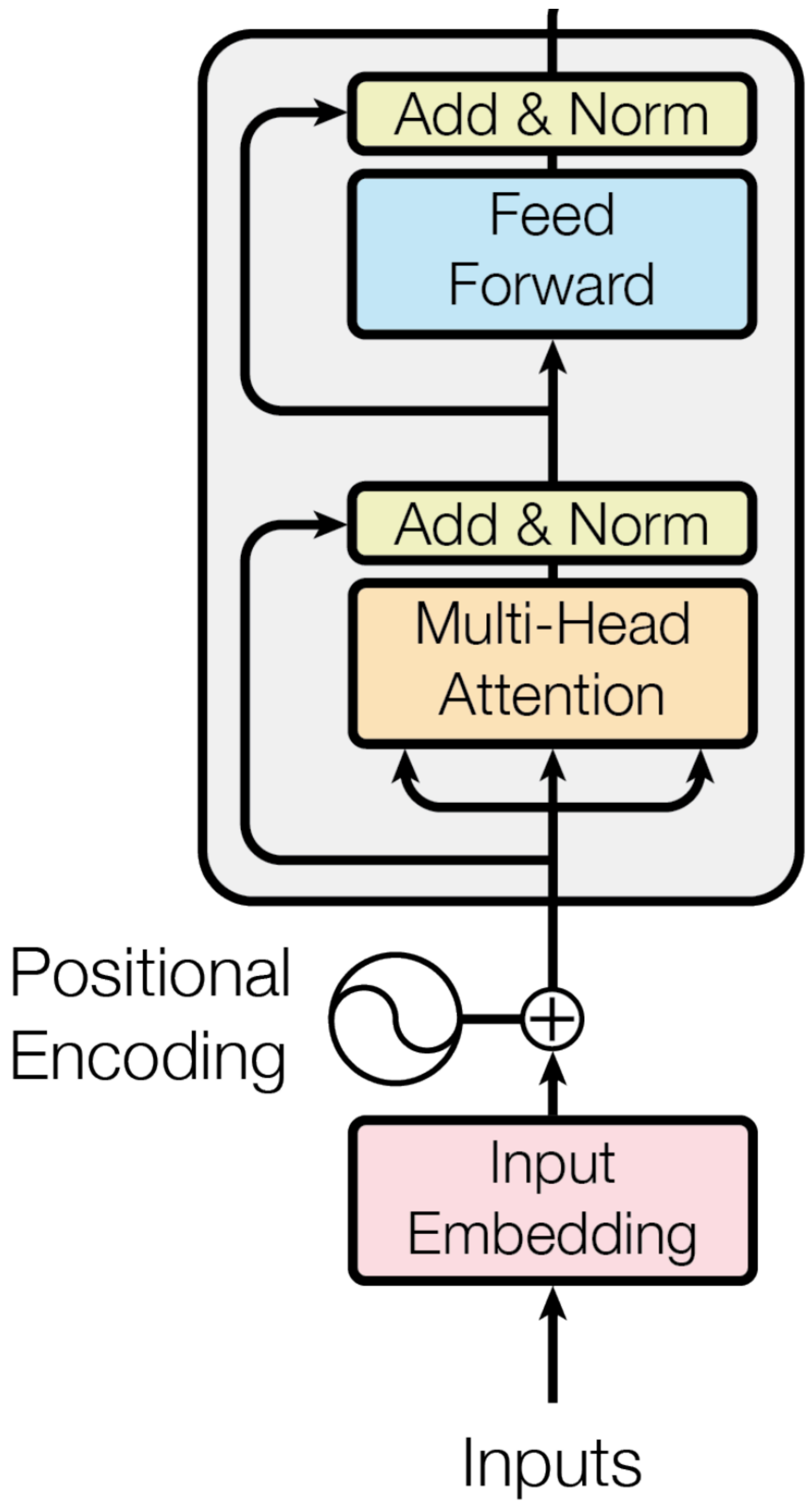


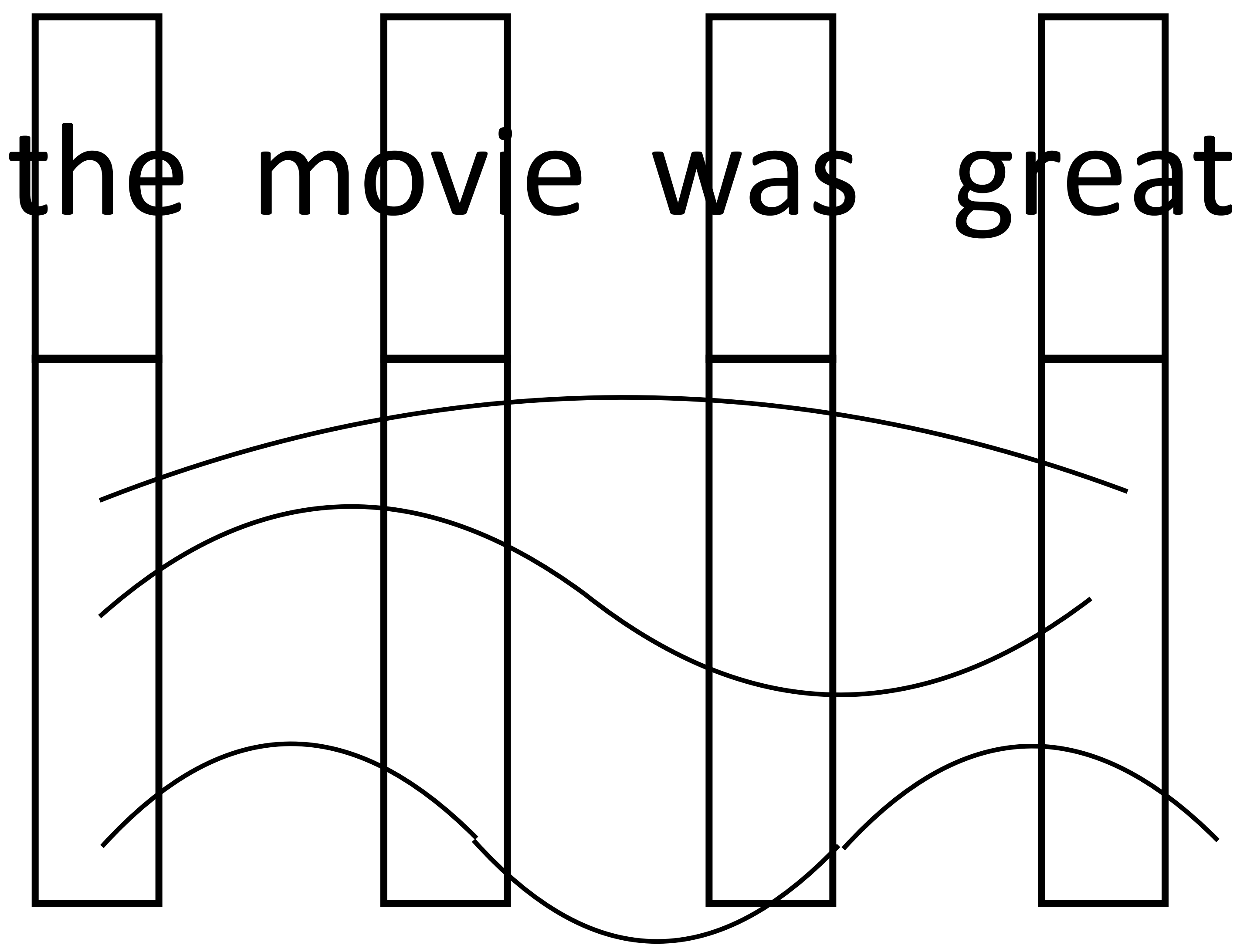
- Multiple “heads” analogous to different convolutional filters. Use parameters W_k and V_k to get different attention values + transform vectors

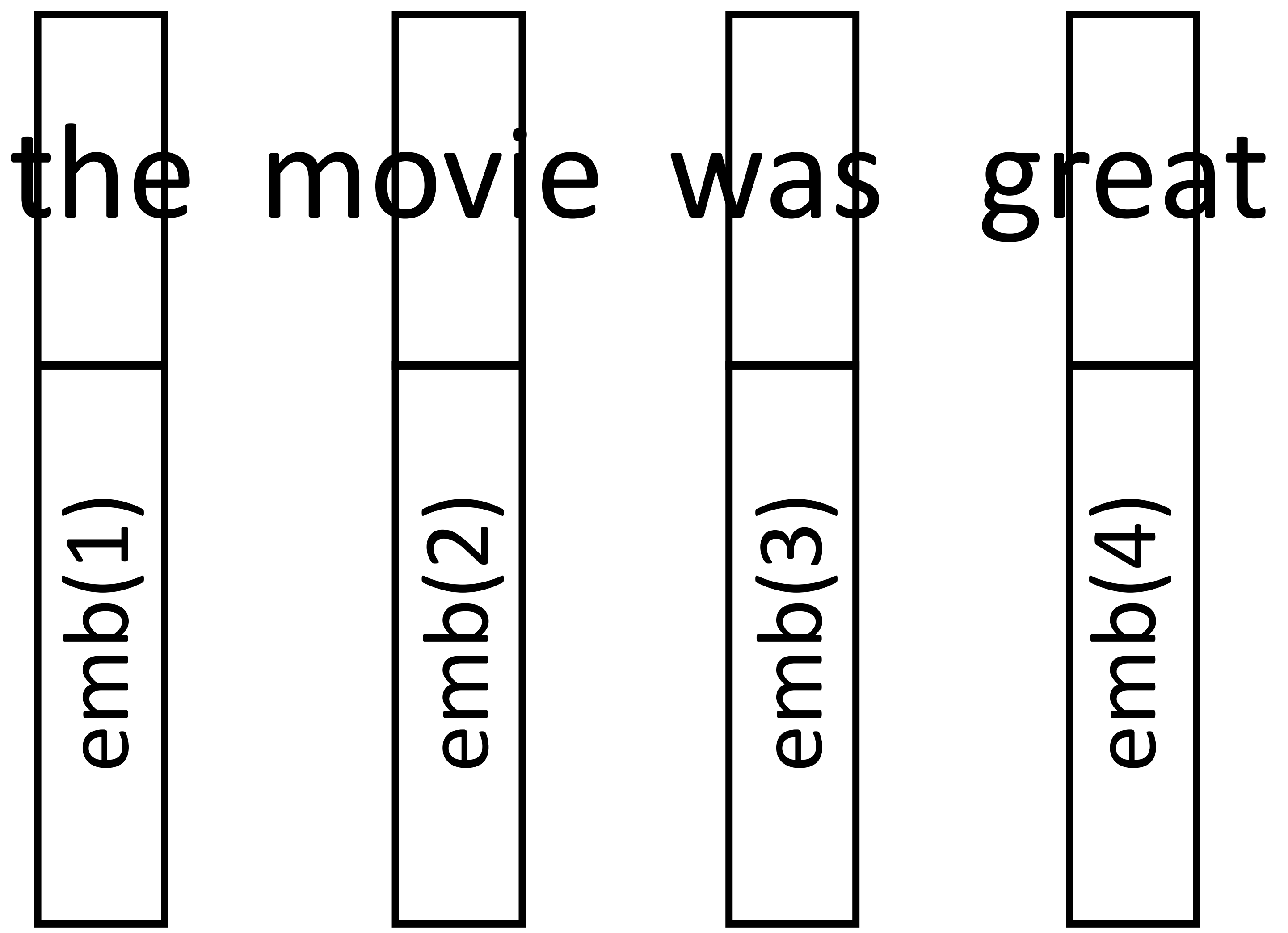
$$\alpha_{k,i,j} = \mathrm{softmax}(x_i^\top W_k x_j) \quad x'_{k,i} = \sum_{j=1}^{n} \alpha_{k,i,j} V_k x_j$$

Vaswani et al. (2017)

Recall: Transformers







- Augment word embedding with position embeddings, each dim is a sine/cosine wave of a different frequency. Closer points = higher dot products
- Works essentially as well as just encoding position as a one-hot vector

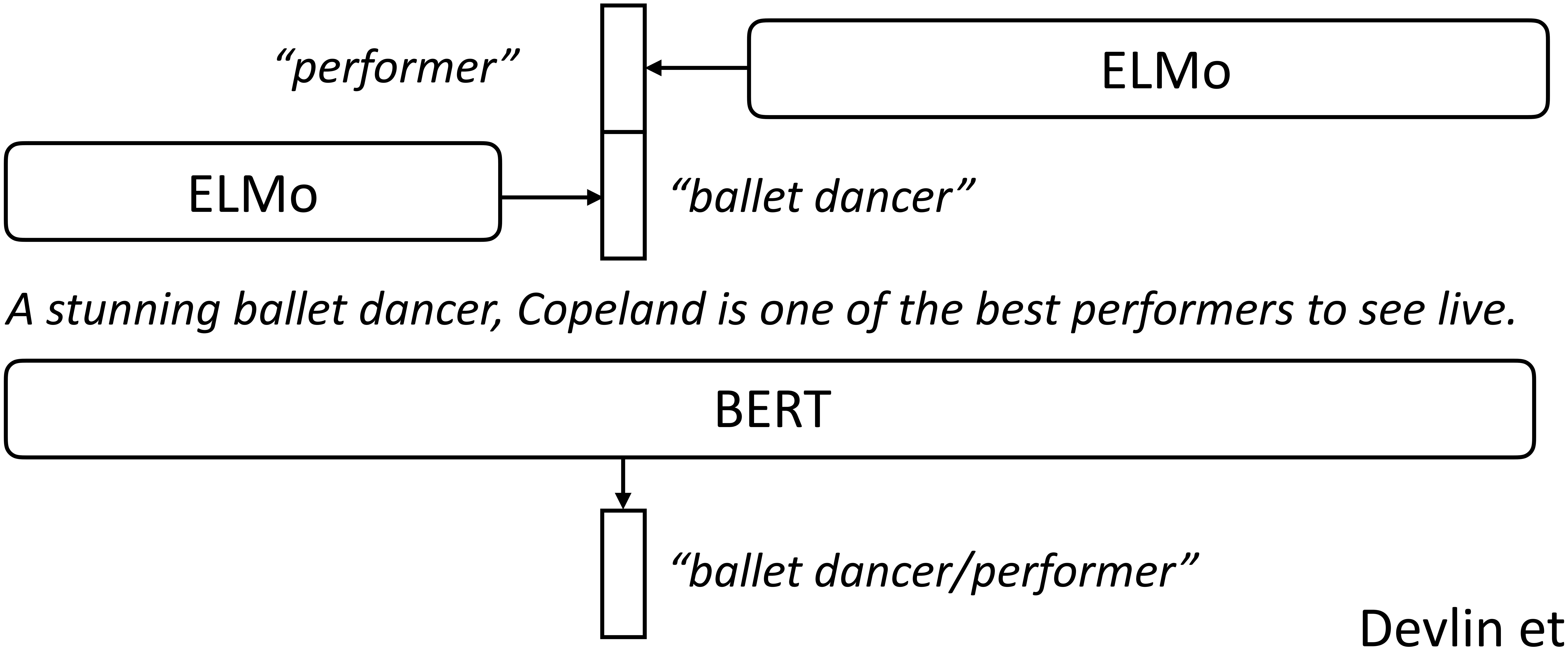
Vaswani et al. (2017)

BERT

- AI2 made ELMo in spring 2018, GPT was released in summer 2018, BERT came out October 2018
- Three major changes compared to ELMo:
  - Transformers instead of LSTMs (transformers in GPT as well)
  - Bidirectional <=> Masked LM objective instead of standard LM
  - Fine-tune instead of freeze at test time

BERT

- ELMo is a unidirectional model (as is GPT): we can concatenate two unidirectional models, but is this the right thing to do?
- ELMo reprs look at each direction in isolation; BERT looks at them jointly

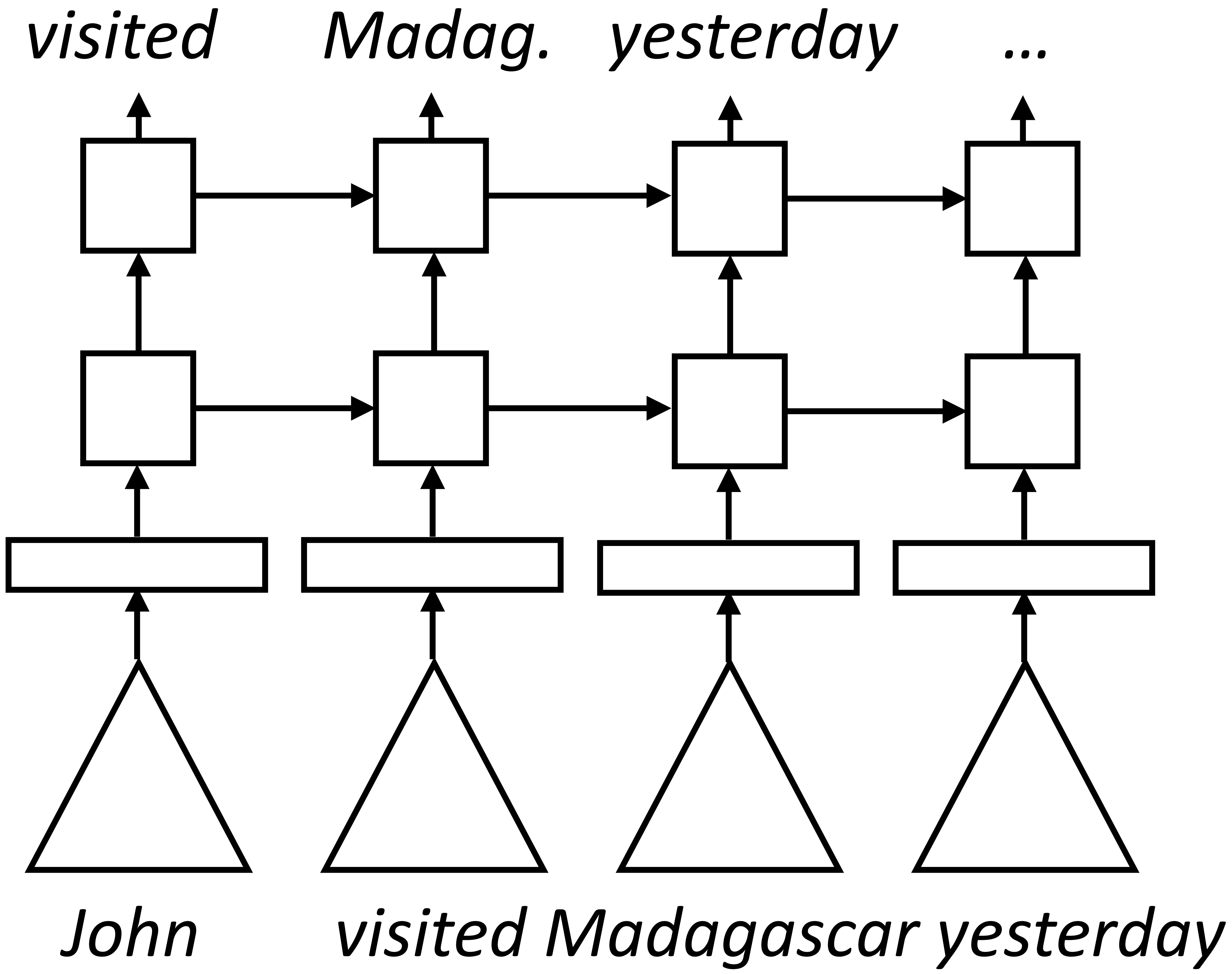


Devlin et al. (2019)

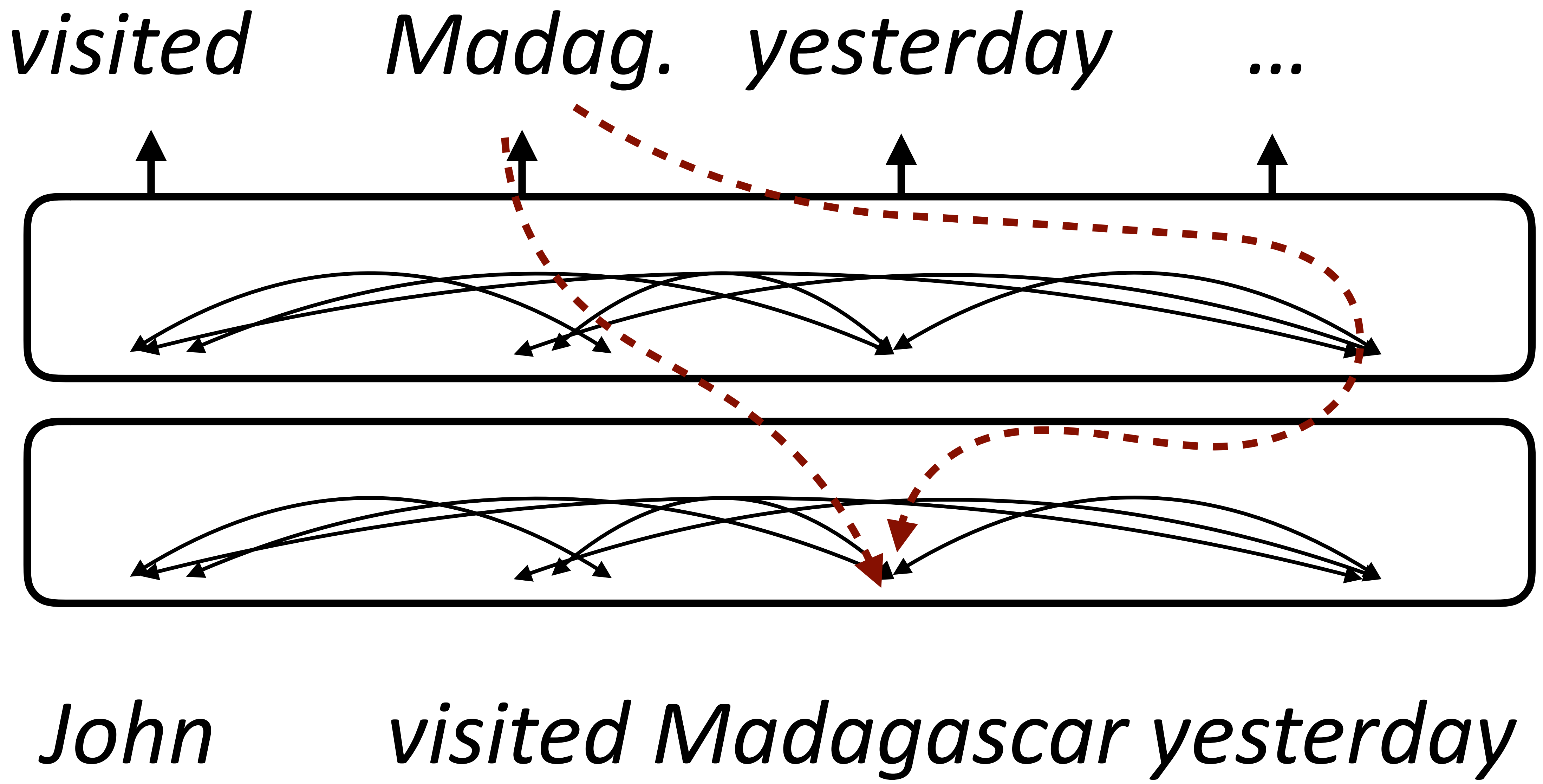
BERT

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

ELMo (Language Modeling)



BERT

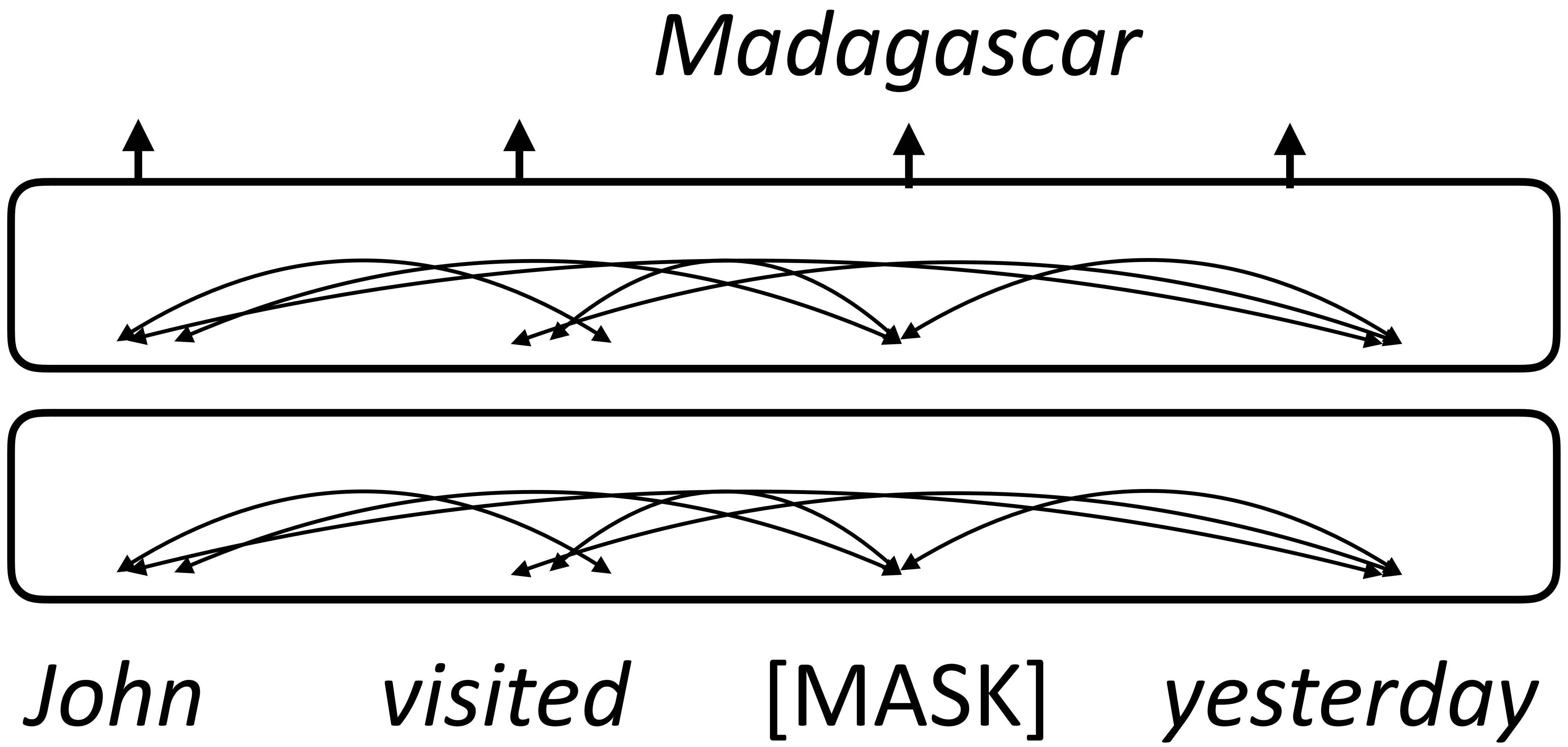


- Transformer LMs have to be “one-sided” (only attend to previous tokens), not what we want

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



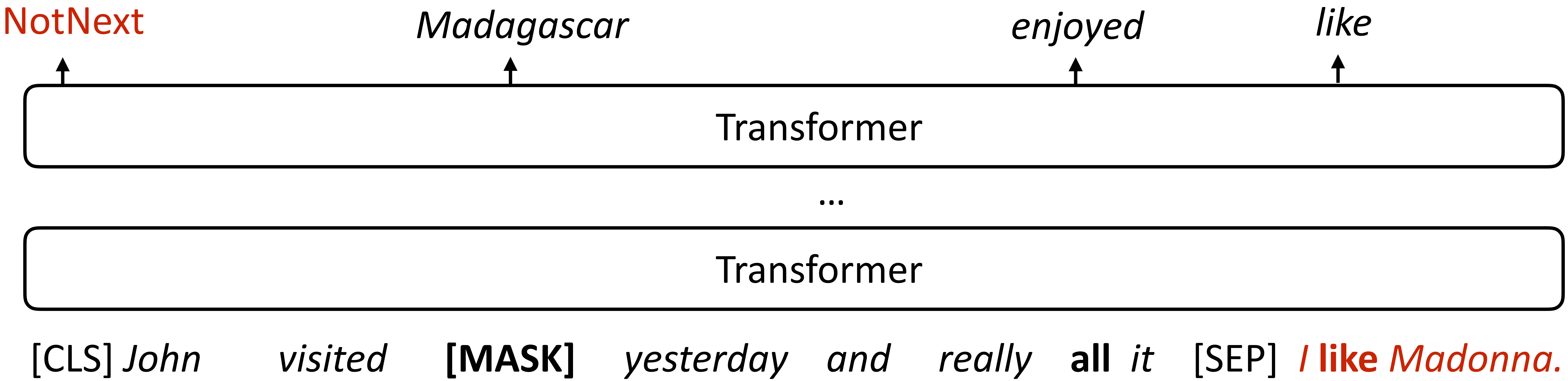
John *visited* **of** *yesterday*

John *visited* Madagascar *yesterday*

Devlin et al. (2019)

Next "Sentence" Prediction

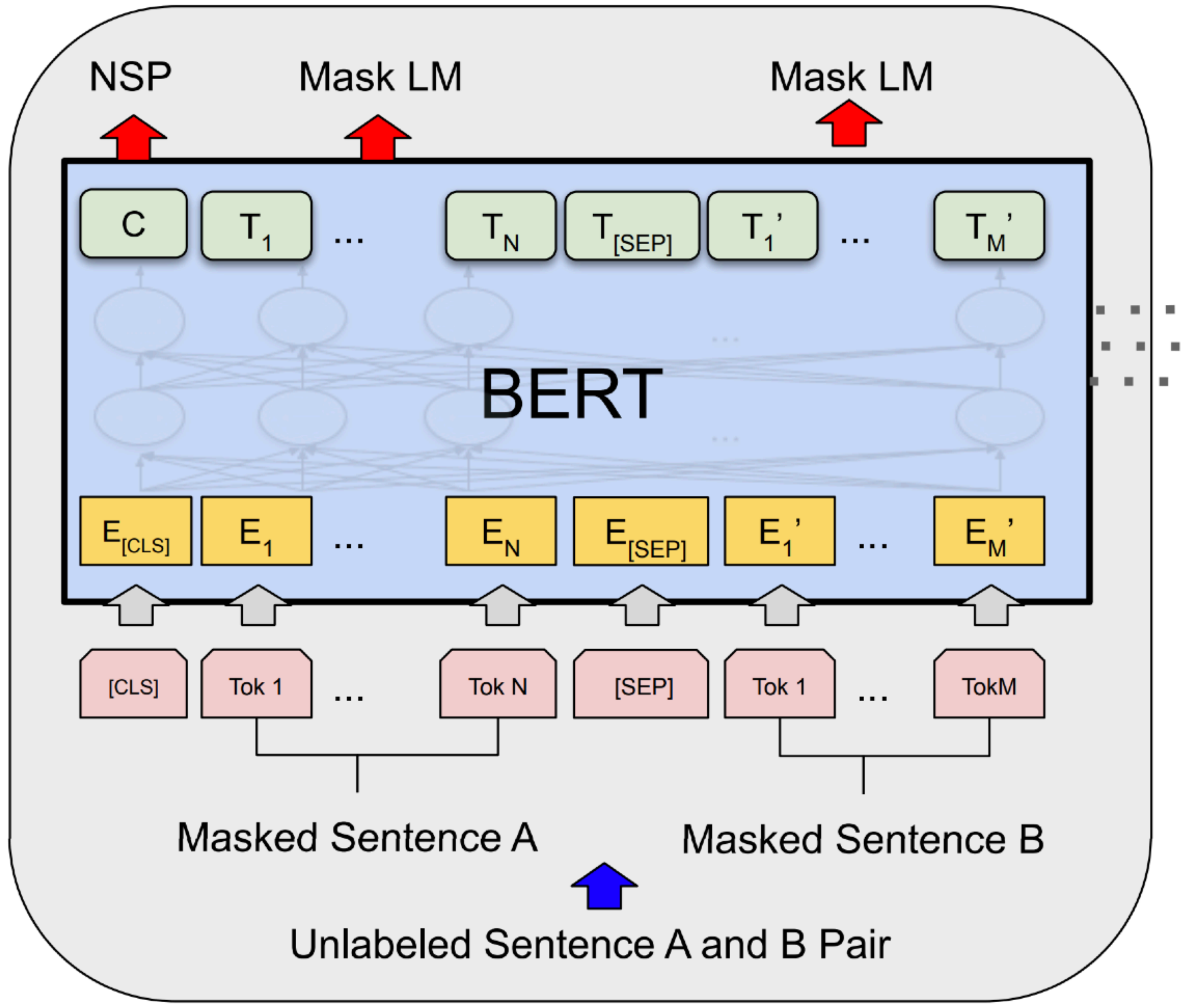
- Input: [CLS] Text chunk 1 [SEP] Text chunk 2
- 50% of the time, take the true next chunk of text, 50% of the time take a random other chunk. Predict whether the next chunk is the "true" next
- BERT objective: masked LM + next sentence prediction

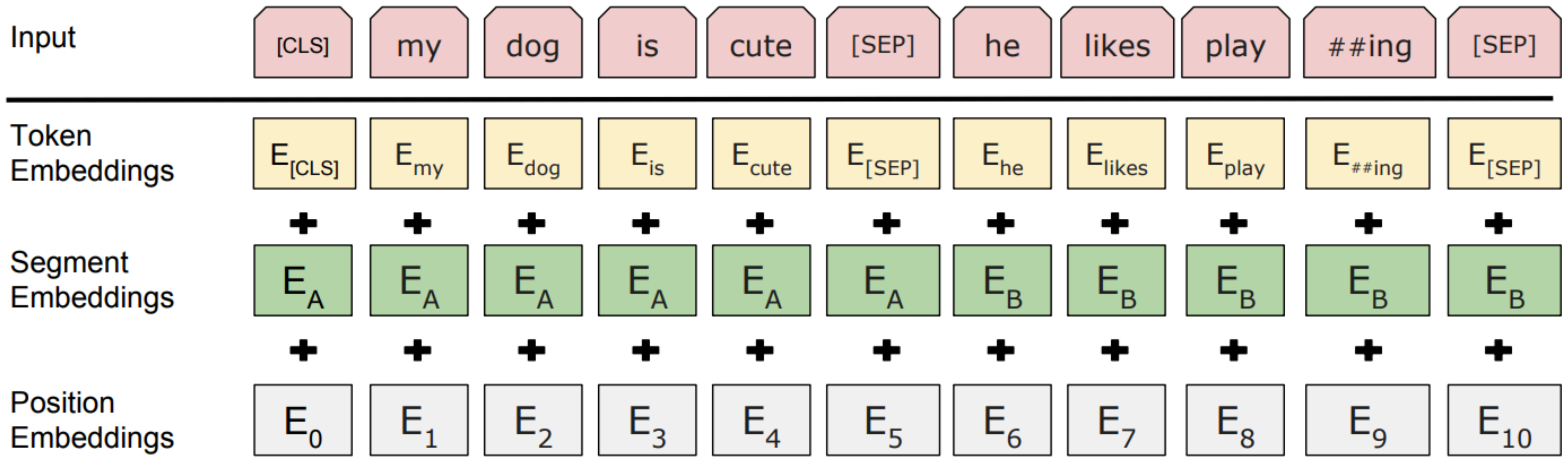


Devlin et al. (2019)

BERT Architecture

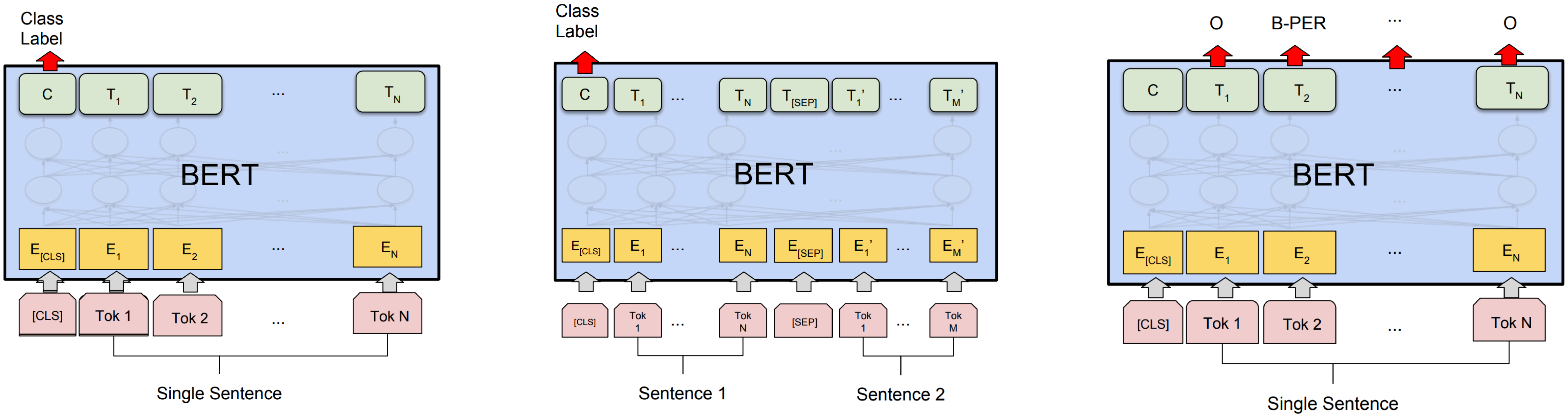
- BERT Base: 12 layers, 768-dim per wordpiece token, 12 heads. Total params = 110M
- BERT Large: 24 layers, 1024-dim per wordpiece token, 16 heads. Total params = 340M
- Positional embeddings and segment embeddings, 30k word pieces
- This is the model that gets **pre-trained** on a large corpus





Devlin et al. (2019)

What can BERT do?



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

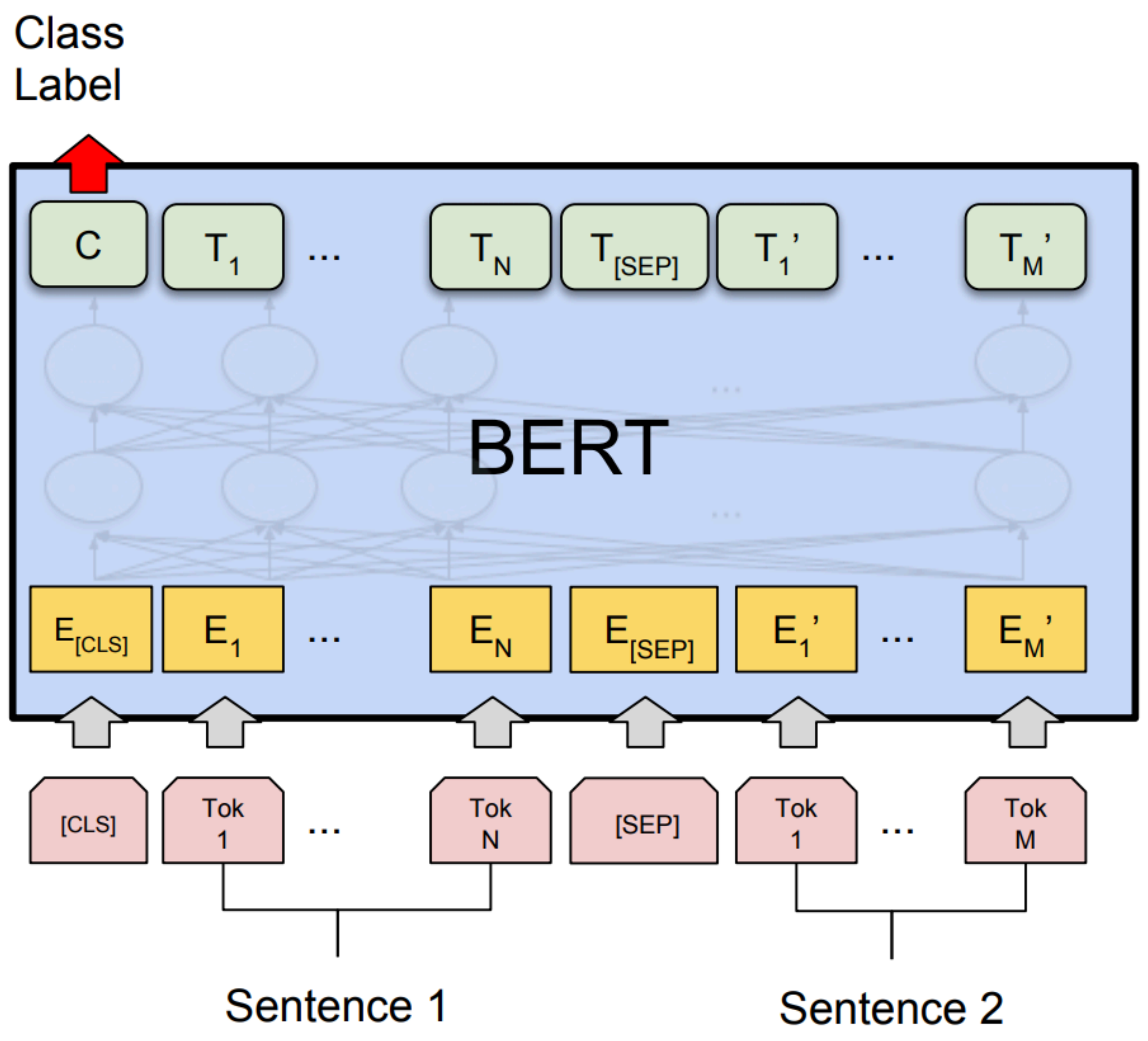
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

- CLS token is used to provide classification decisions
- Sentence pair tasks (entailment): feed both sentences into BERT
- BERT can also do tagging by predicting tags at each word piece

Devlin et al. (2019)

What can BERT do?



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

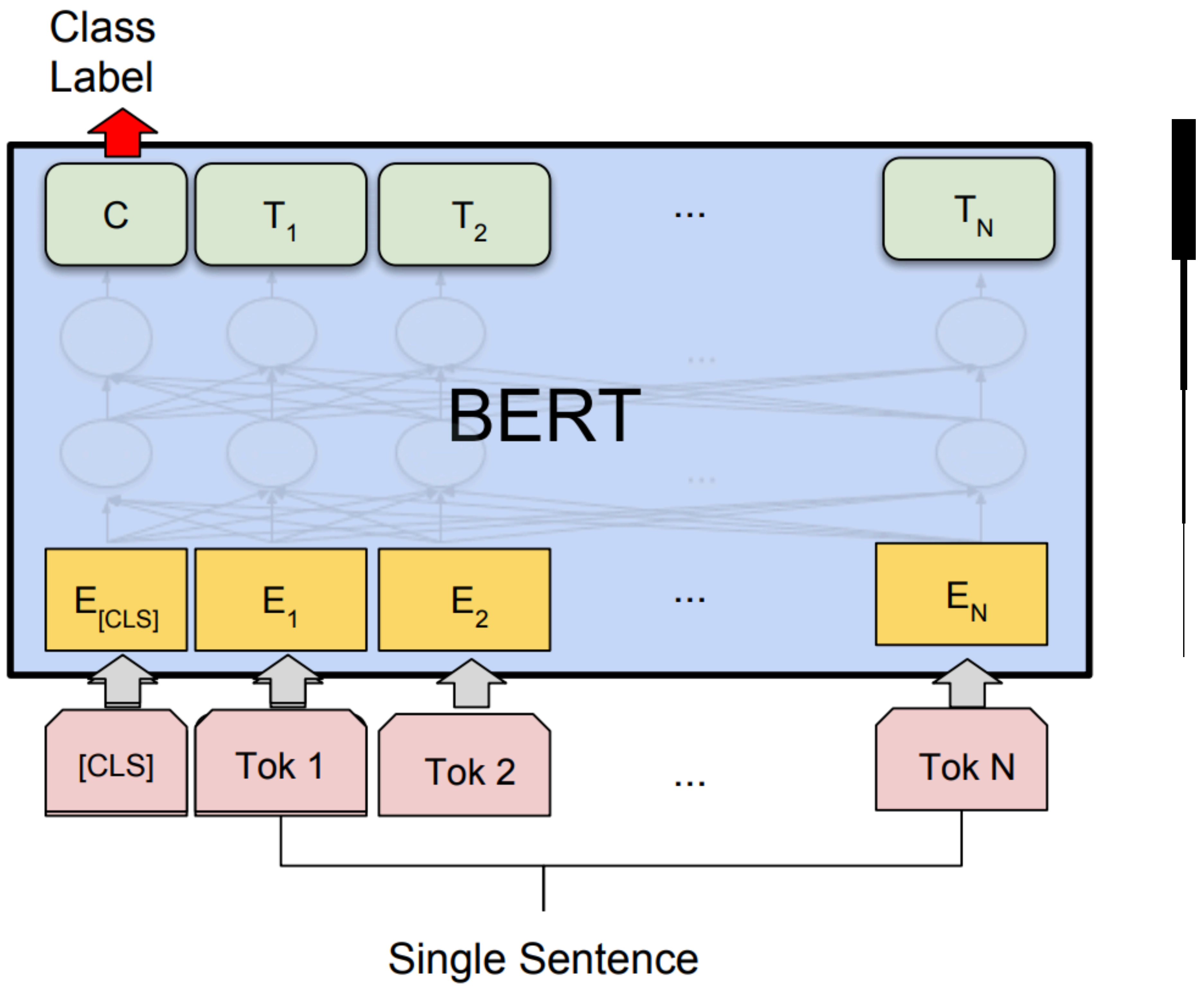
- How does BERT model this sentence pair stuff?
- Transformers can capture interactions between the two sentences, even though the NSP objective doesn't really cause this to happen

What can BERT NOT do?

- BERT **cannot** generate text (at least not in an obvious way)
  - Not an autoregressive model, can do weird things like stick a [MASK] at the end of a string, fill in the mask, and repeat
- Masked language models are intended to be used primarily for “analysis” tasks

Fine-tuning BERT

- Fine-tune for 1-3 epochs, batch size 2-32, learning rate 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

Fine-tuning BERT

Pretraining	Adaptation	NER CoNLL 2003	SA SST-2	Nat. lang. inference MNLI	Nat. lang. inference SICK-E	Semantic textual similarity SICK-R	Semantic textual similarity MRPC	Semantic textual similarity STS-B
Skip-thoughts	❄️	-	81.8	62.9	-	86.6	75.8	71.8
ELMo	❄️	91.7	**91.8**	**79.6**	**86.3**	**86.1**	**76.0**	**75.9**
	🔥	**91.9**	91.2	76.4	83.3	83.3	74.7	75.5
	Δ=🔥-❄️	0.2	-0.6	-3.2	-3.3	-2.8	-1.3	-0.4
BERT-base	❄️	92.2	93.0	**84.6**	84.8	86.4	78.1	82.9
	🔥	**92.4**	**93.5**	**84.6**	**85.8**	**88.7**	**84.8**	**87.1**
	Δ=🔥-❄️	0.2	0.5	0.0	1.0	2.3	6.7	4.2

- BERT is typically better if the whole network is fine-tuned, unlike ELMo

Peters, Ruder, Smith (2019)

Evaluation: GLUE

Corpus	|Train|	|Test|	Task	Metrics	Domain
			Single-Sentence Tasks		
CoLA	8.5k	**1k**	acceptability	Matthews corr.	misc.
SST-2	67k	1.8k	sentiment	acc.	movie reviews
			Similarity and Paraphrase Tasks		
MRPC	3.7k	1.7k	paraphrase	acc./F1	news
STS-B	7k	1.4k	sentence similarity	Pearson/Spearman corr.	misc.
QQP	364k	**391k**	paraphrase	acc./F1	social QA questions
			Inference Tasks		
MNLI	393k	**20k**	NLI	matched acc./mismatched acc.	misc.
QNLI	105k	5.4k	QA/NLI	acc.	Wikipedia
RTE	2.5k	3k	NLI	acc.	news, Wikipedia
WNLI	634	**146**	coreference/NLI	acc.	fiction books

Wang et al. (2019)

Results

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
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)

RoBERTa

- “Robustly optimized BERT”
- 160GB of data instead of 16 GB
- Dynamic masking: standard BERT uses the same MASK scheme for every epoch, RoBERTa recomputes them

Model	**data**	**bsz**	**steps**	**SQuAD** (v1.1/2.0)	**MNLI-m**	**SST-2**
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7

- New training + more data = better performance

Liu et al. (2019)

GPT/GPT2

OpenAI GPT/GPT2

- "ELMo with transformers" (works better than ELMo)
- Train a single unidirectional transformer LM on long contexts
- GPT2: trained on 40GB of text collected from upvoted links from reddit
- 1.5B parameters — by far the largest of these models trained as of March 2019

Parameters	Layers	d_{model}
117M	12	768
345M	24	1024
762M	36	1280
1542M	48	1600

- Because it's a language model, we can **generate** from it

Radford et al. (2019)

OpenAI GPT2

SYSTEM PROMPT (HUMAN-WRITTEN)

Miley Cyrus was caught shoplifting from Abercrombie and Fitch on Hollywood Boulevard today.

MODEL COMPLETION (MACHINE-WRITTEN, SECOND TRY)

The 19-year-old singer was caught on camera being escorted out of the store by security guards.

The singer was wearing a black hoodie with the label ‘Blurred Lines’ on the front and ‘Fashion Police’ on the back.

Scroll down for video

Shoplifting: Miley Cyrus was caught shoplifting from Abercrombie and Fitch on Hollywood Boulevard today (pictured)

The singer was wearing a black hoodie with the label ‘Blurred Lines’ on the front and ‘Fashion Police’ on the back

The singer was also wearing a pair of black-rimmed glasses, a black jacket, black jeans and black sandals.

She was carrying a pair of black and white striped gloves and a small black bag.

slide credit: OpenAI

Open Questions

1) How novel is the stuff being generated? (Is it just doing nearest neighbors on a large corpus?)

2) How do we understand and distill what is learned in this model?

3) How do we harness these priors for conditional generation tasks (summarization, generate a report of a basketball game, etc.)

4) Is this technology dangerous? (OpenAI has only released 774M param model, not 1.5B yet)

Pre-Training Cost (with Google/AWS)

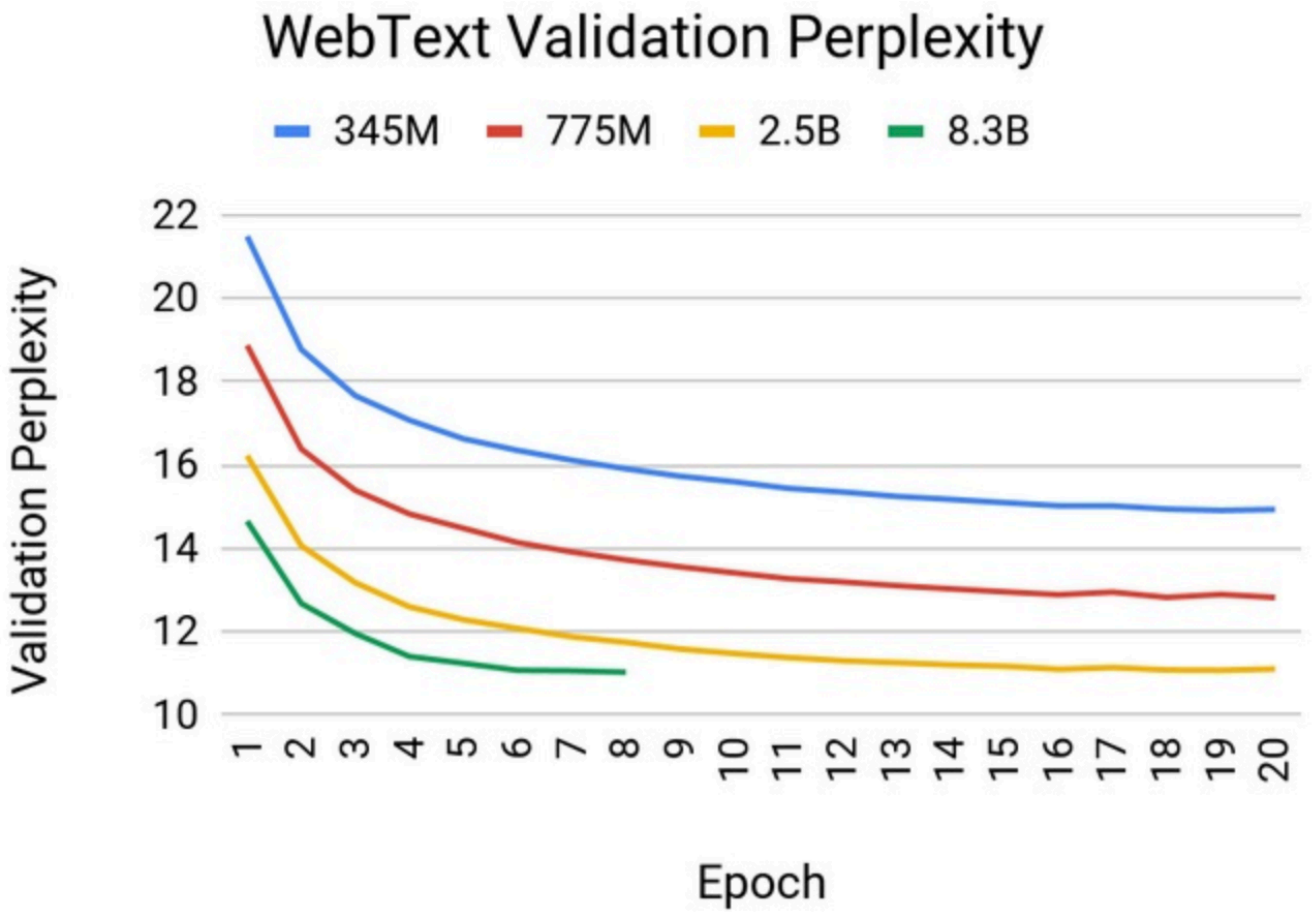
- BERT: Base \$500, Large \$7000
- Grover-MEGA: \$25,000
- XLNet (BERT variant): \$30,000 — \$60,000 (unclear)
- This is for a single pre-training run…developing new pre-training techniques may require many runs
- *Fine-tuning* these models can typically be done with a single GPU (but may take 1-3 days for medium-sized datasets)

https://syncedreview.com/2019/06/27/the-staggering-cost-of-training-sota-ai-models/

Pushing the Limits

- NVIDIA: trained 8.3B parameter GPT model (5.6x the size of GPT-2)

- Arguable these models are still underfit: larger models still get better held-out perplexities



NVIDIA blog (Narasimhan, August 2019)

Google T5

Number of tokens	Repeats	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Full dataset	0	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
2^{29}	64	**82.87**	**19.19**	**80.97**	**72.03**	**26.83**	**39.74**	**27.63**
2^{27}	256	82.62	**19.20**	79.78	69.97	**27.02**	**39.71**	27.33
2^{25}	1,024	79.55	18.57	76.27	64.76	26.38	39.56	26.80
2^{23}	4,096	76.34	18.33	70.92	59.29	26.37	38.84	25.81

- Colossal Cleaned Common Crawl: 750 GB of text

- We still haven't hit the limit of bigger data being useful

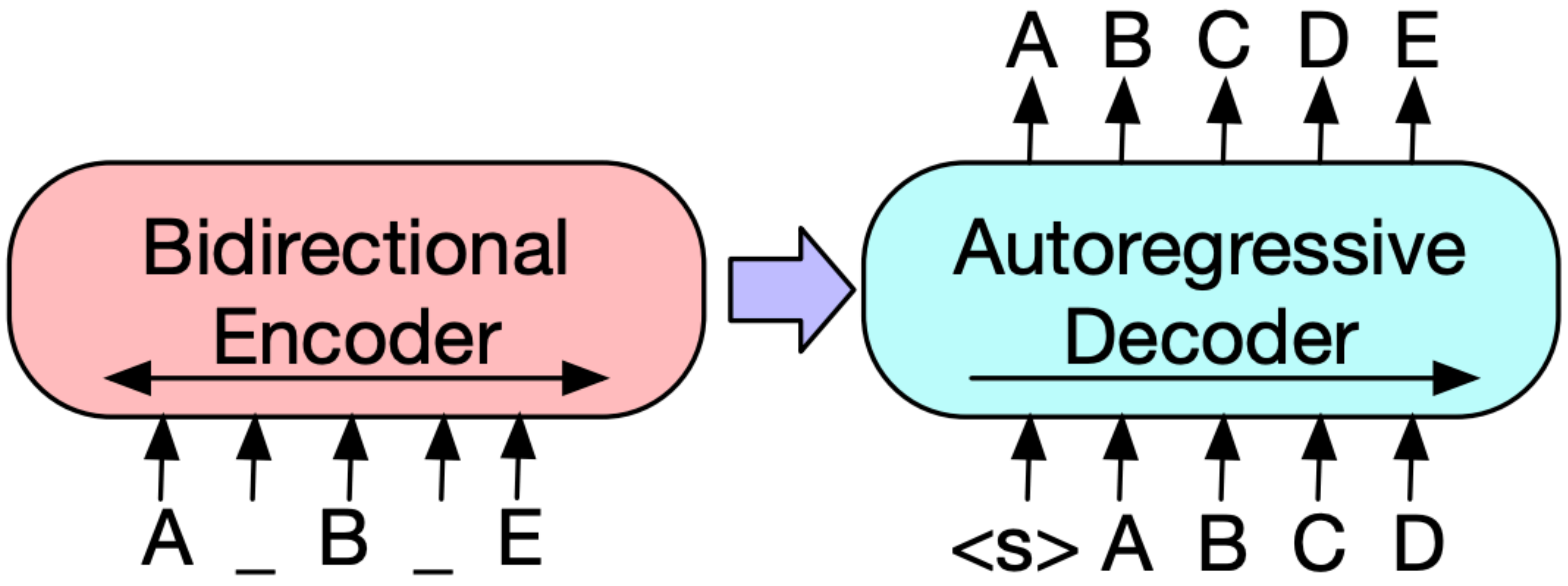
Raffel et al. (October 23, 2019)

BART

- Sequence-to-sequence BERT variant: permute/make/delete tokens, then predict full sequence autoregressively

- For downstream tasks: feed document into both encoder + decoder, use decoder hidden state as output

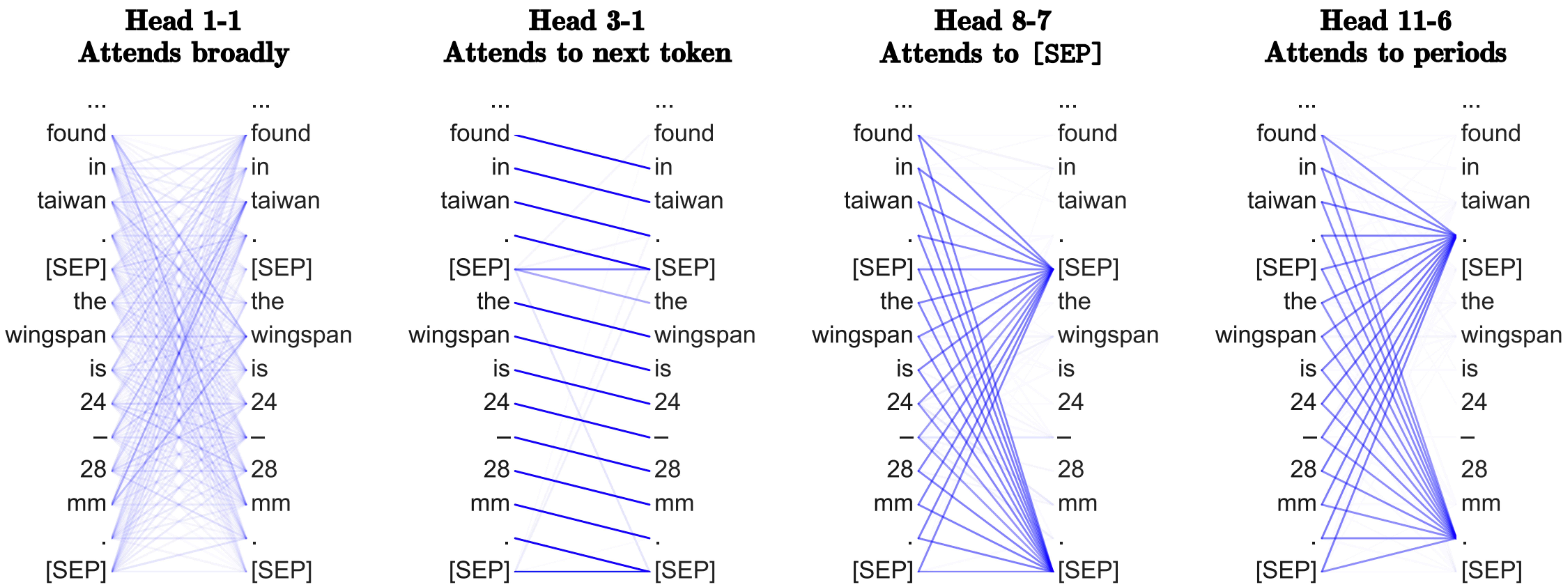
- Good results on dialogue, summarization tasks



Lewis et al. (October 30, 2019)

Analysis

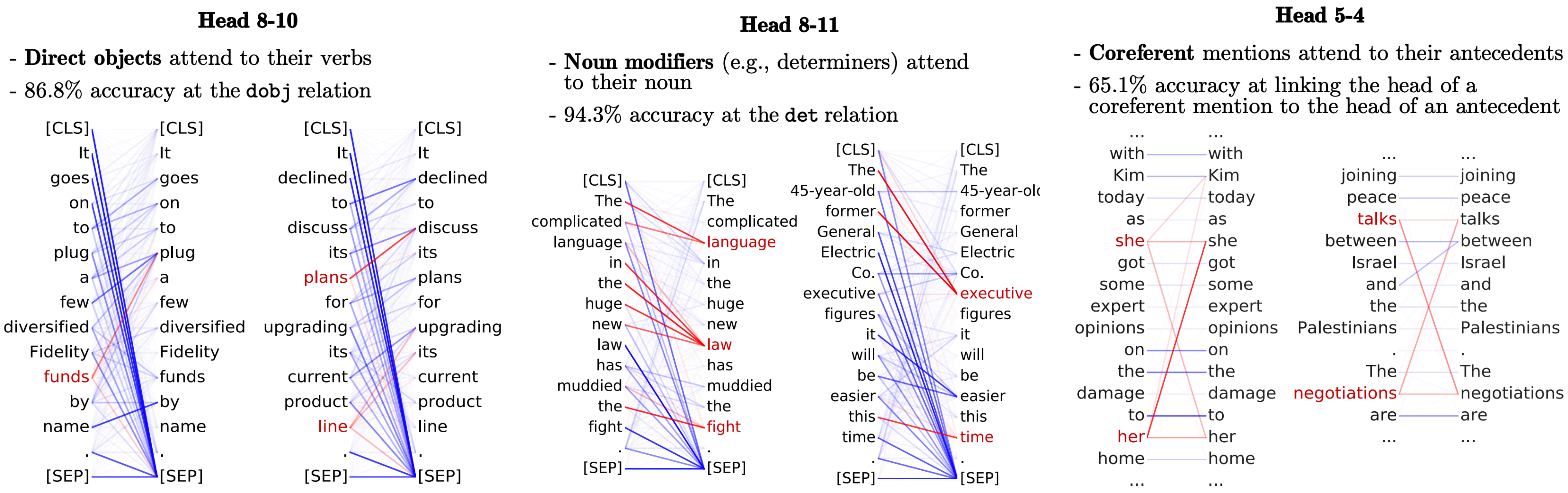
What does BERT learn?



- Heads on transformers learn interesting and diverse things: content heads (attend based on content), positional heads (based on position), etc.

Clark et al. (2019)

What does BERT learn?



- Still way worse than what supervised systems can do, but interesting that this is learned organically

Clark et al. (2019)

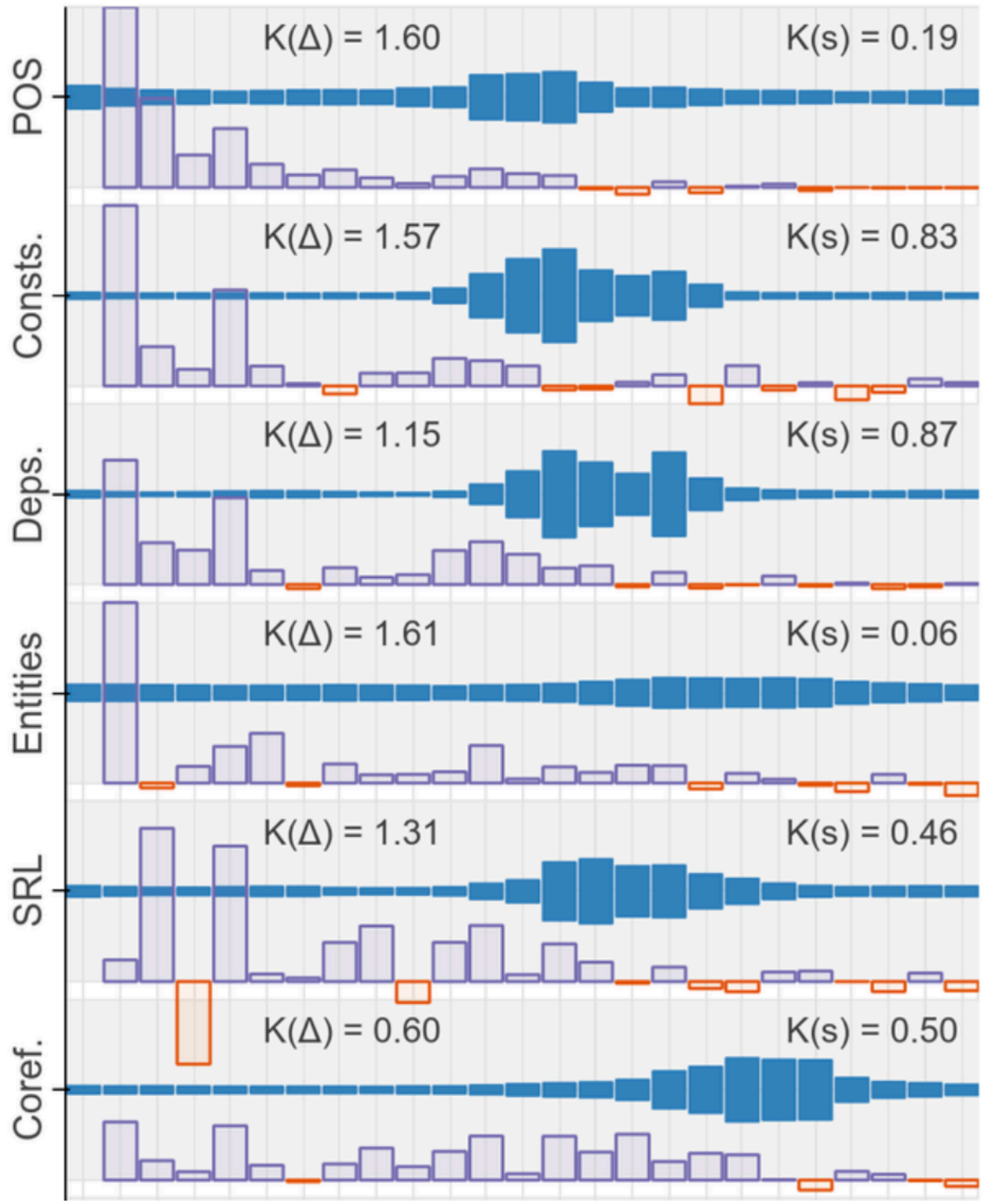
Probing BERT

- Try to predict POS, etc. from each layer. Learn mixing weights

$$\mathbf{h}_{i,\tau} = \gamma_\tau \sum_{\ell=0}^{L} s_\tau^{(\ell)} \mathbf{h}_i^{(\ell)}$$

  representation of wordpiece *i* for task τ

- Plot shows *s* weights (blue) and performance deltas when an additional layer is incorporated (purple)
- BERT "rediscovers the classical NLP pipeline": first syntactic tasks then semantic ones

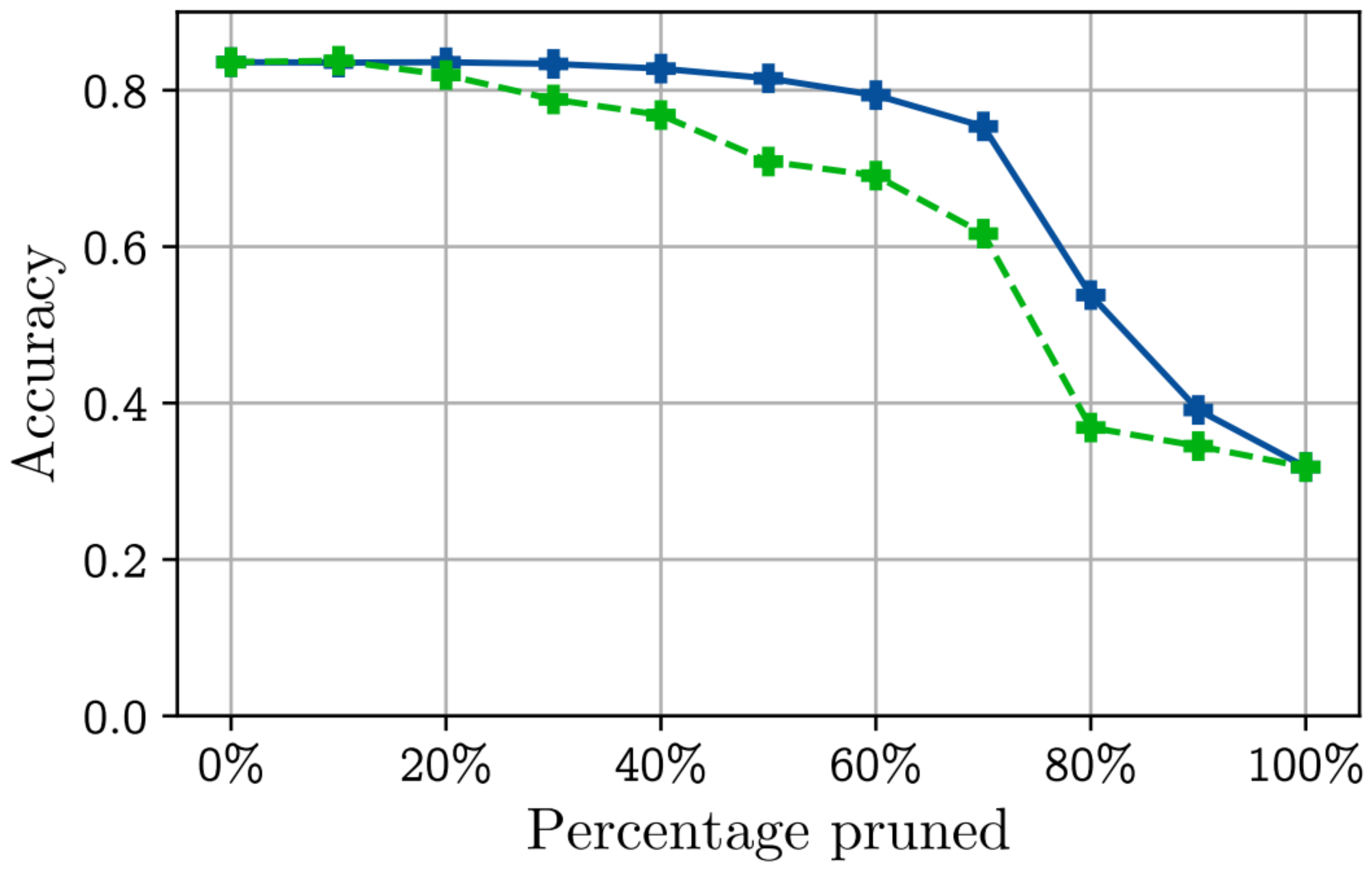


Tenney et al. (2019)

Compressing BERT

- Remove 60+% of BERT's heads 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).

Michel et al. (2019)

Open Questions

- BERT-based systems are state-of-the-art for nearly every major text analysis task
- These techniques are here to stay, unclear what form will win out
- Role of academia vs. industry: no major pretrained model has come purely from academia
- Cost/carbon footprint: a single model costs $10,000+ to train (though this cost should come down)