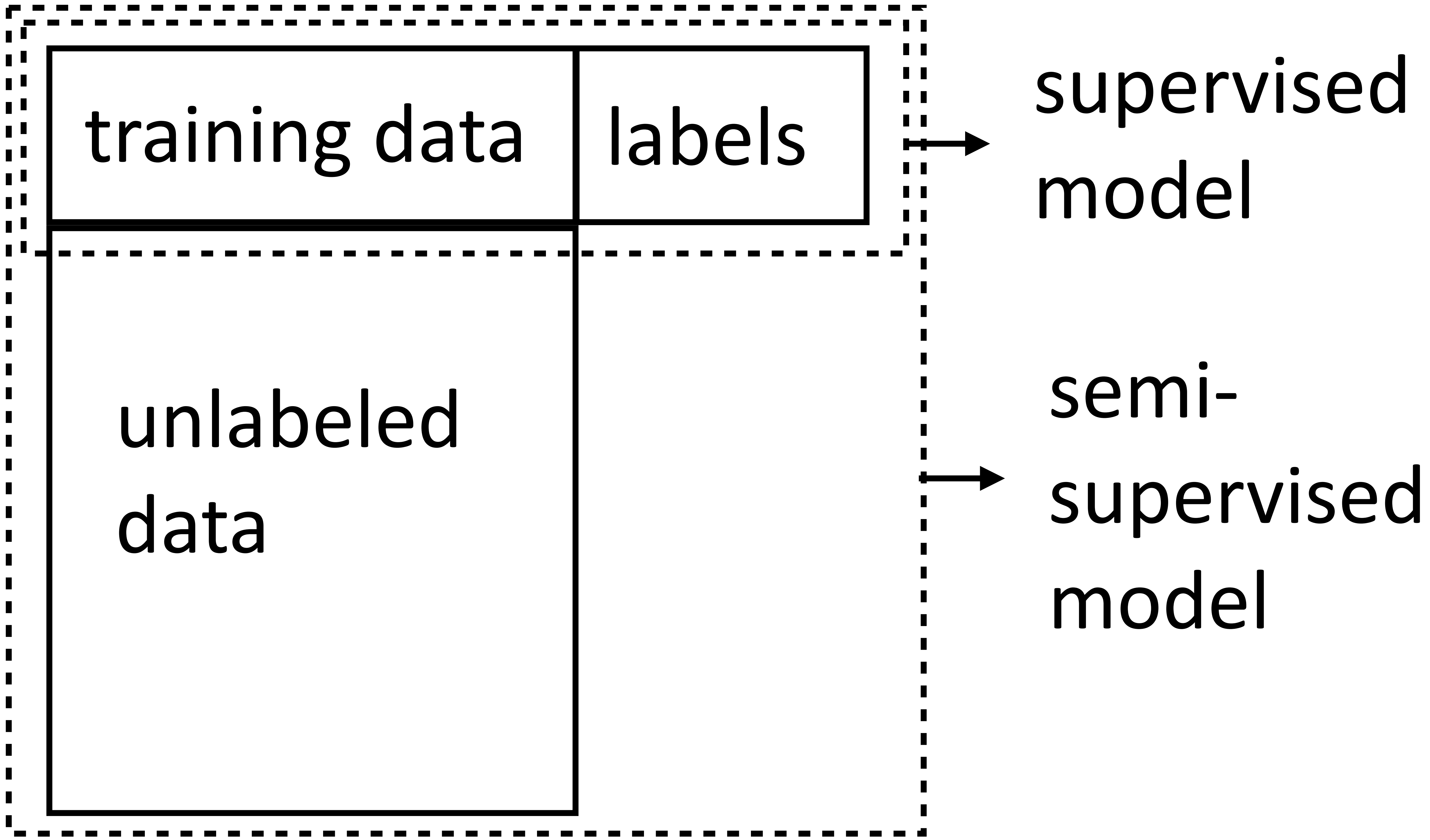
CS388: Natural Language Processing
Lecture 23: Unsupervised Learning

Greg Durrett

Some slides adapted from Leon Gu (CMU), Taylor Berg-Kirkpatrick (CMU)

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?



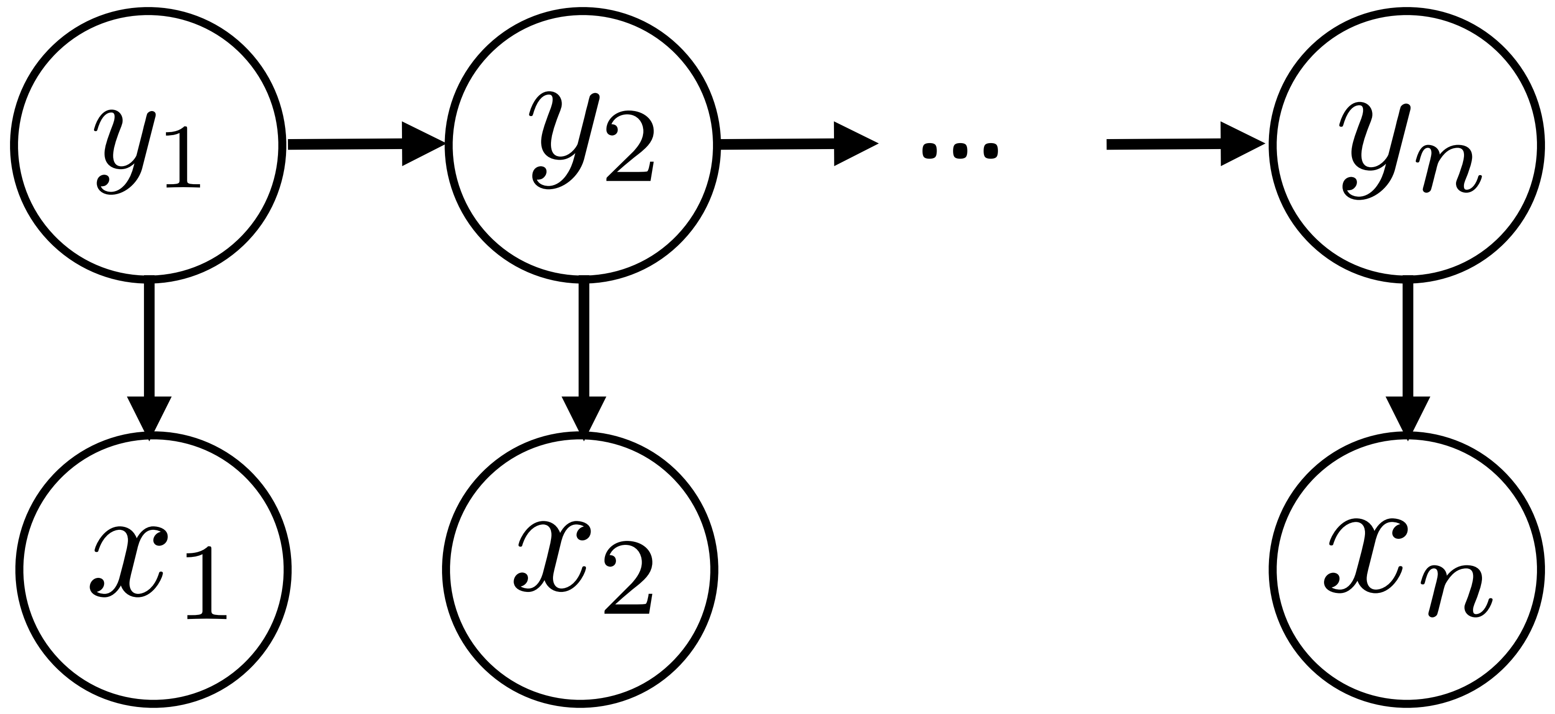
This Lecture

- 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 $P(\mathbf{x}) = \sum_{\mathbf{y}} P(\mathbf{y}, \mathbf{x})$ ← 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

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

- 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, $\theta^t = \text{argmax}_\theta \mathbb{E}_{q^t} \log P(\mathbf{x}, \mathbf{y}|\theta)$

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, $\theta^t = \text{argmax}_\theta \mathbb{E}_{q^t} \log P(\mathbf{x}, \mathbf{y}|\theta)$
- 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*

q	*the*	*dog*
	DT: 0.9	DT: 0.3
	NN: 0.1	**NN: 0.7**

→

count(DT, the) = 0.9
count(DT, dog) = 0.3
count(NN, the) = 0.1
count(NN, dog) = 0.7

→

P(the|DT) = 0.75
P(dog|DT) = 0.25
P(the|NN) = 0.125
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*

→

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**	DT: 0.5
NN: 0.05	**NN: 0.95**	NN: 0.05	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**	DT: 0.5
NN: 0.05	**NN: 0.95**	NN: 0.05	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**	DT: 0.30
NN: 0.05	**NN: 0.95**	NN: 0.05	**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?

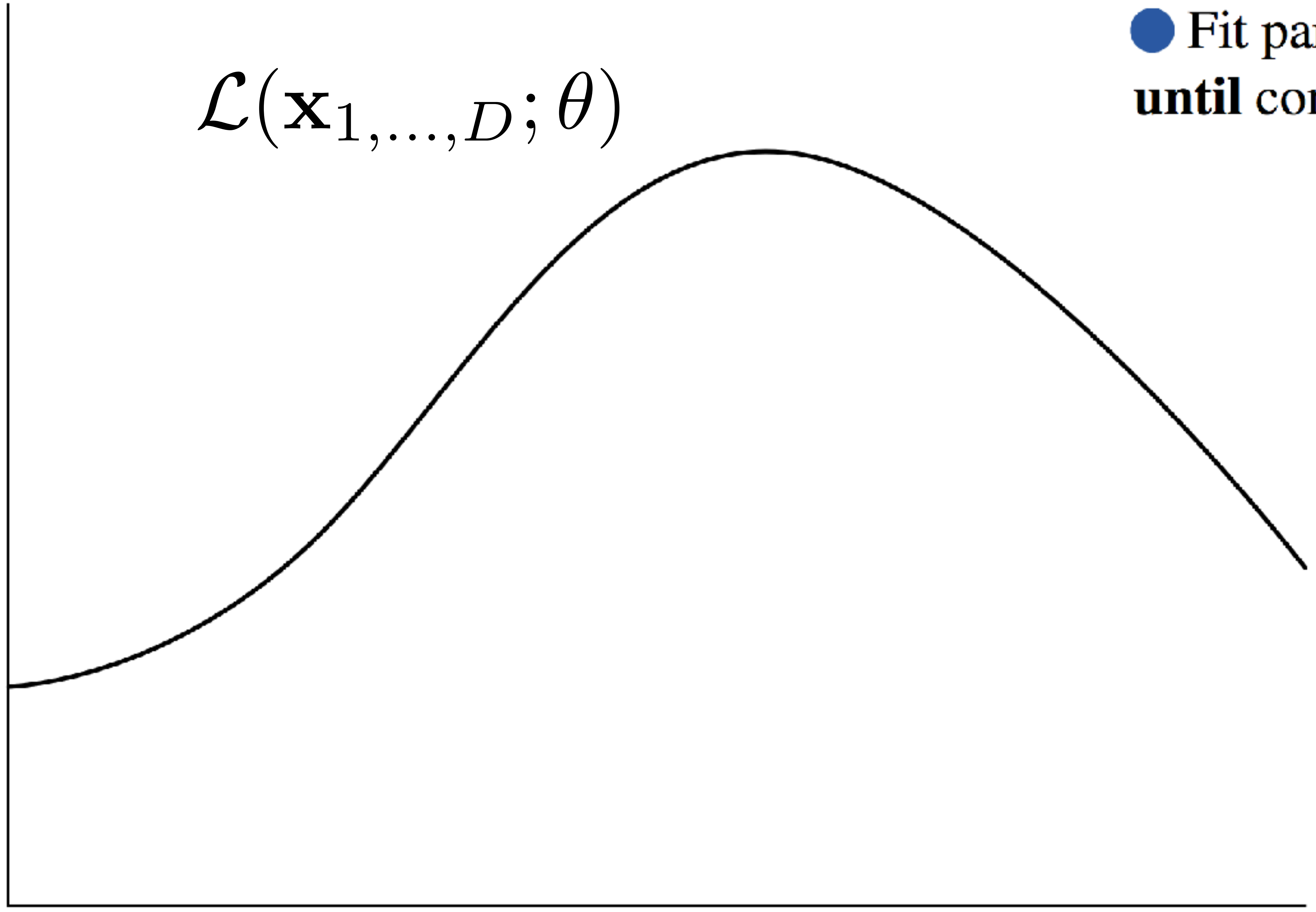
- Can think of *q* as a kind of “fractional annotation”
- E-step: compute annotations (posterior under current model)
- M-step: supervised learning with those fractional annotations
- Initialize with some reasonable weights, alternate E and M until convergence

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



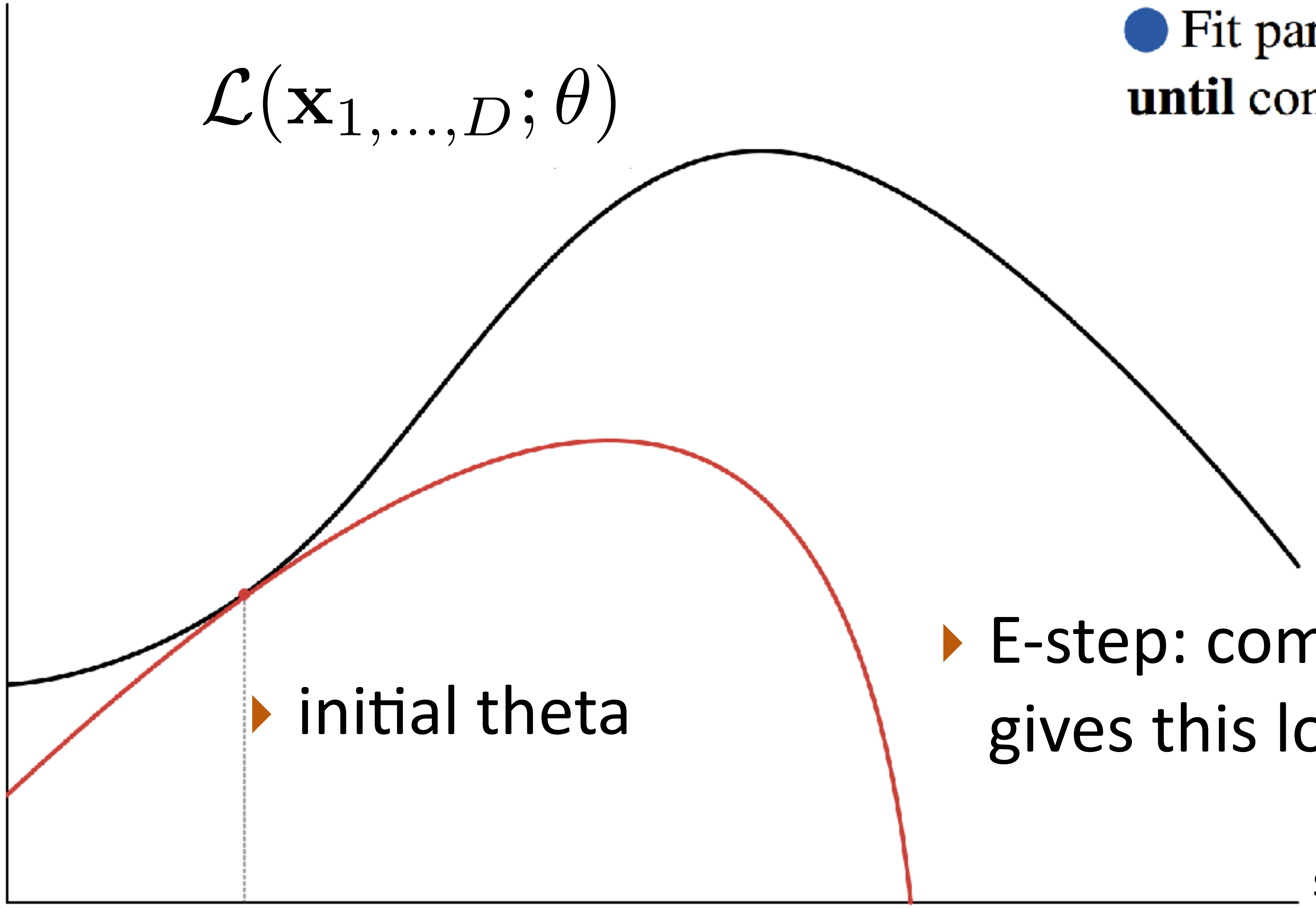
slide credit: Taylor Berg-Kirkpatrick

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



- initial theta
- E-step: compute q which gives this lower bound

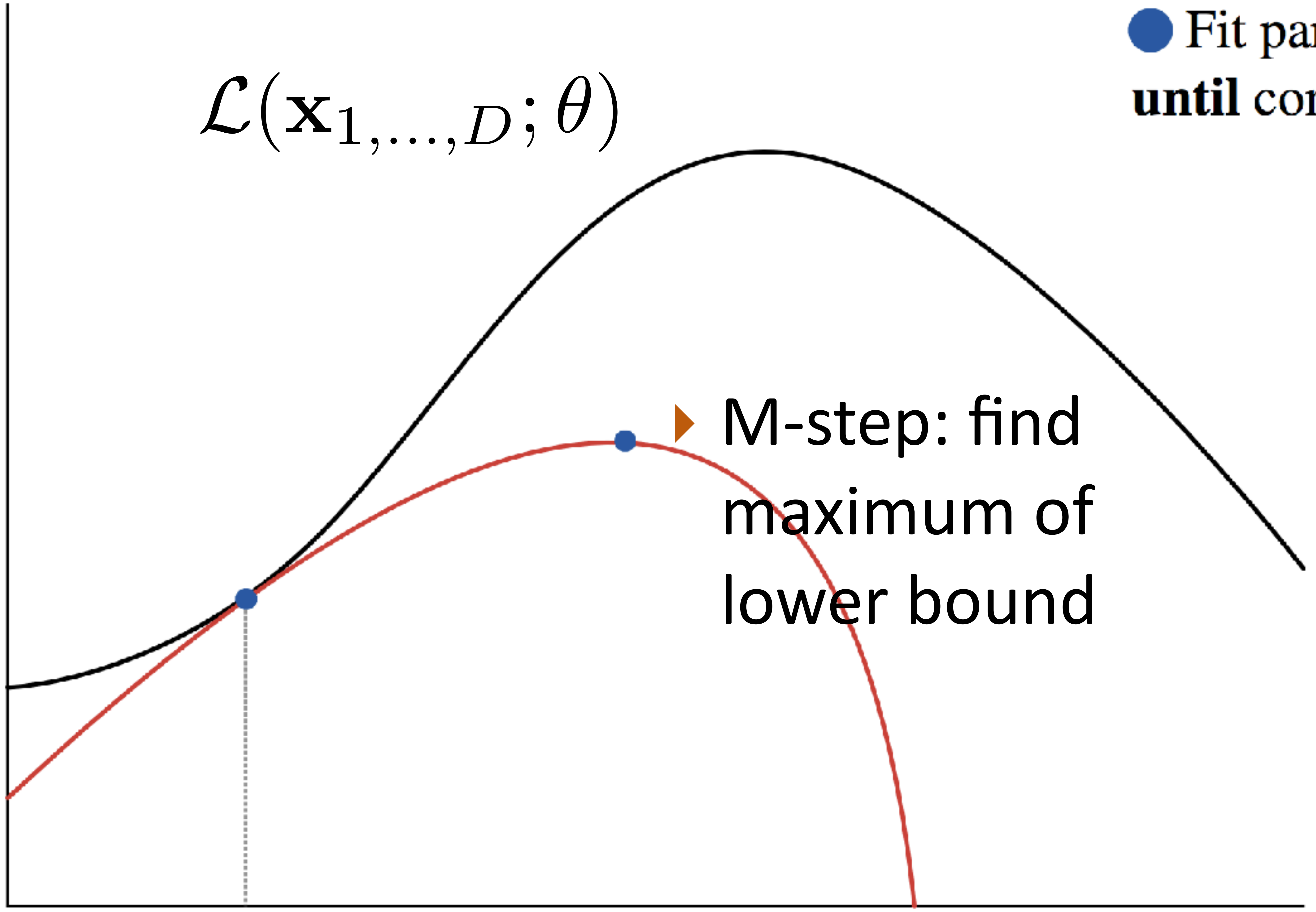
slide credit: Taylor Berg-Kirkpatrick

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

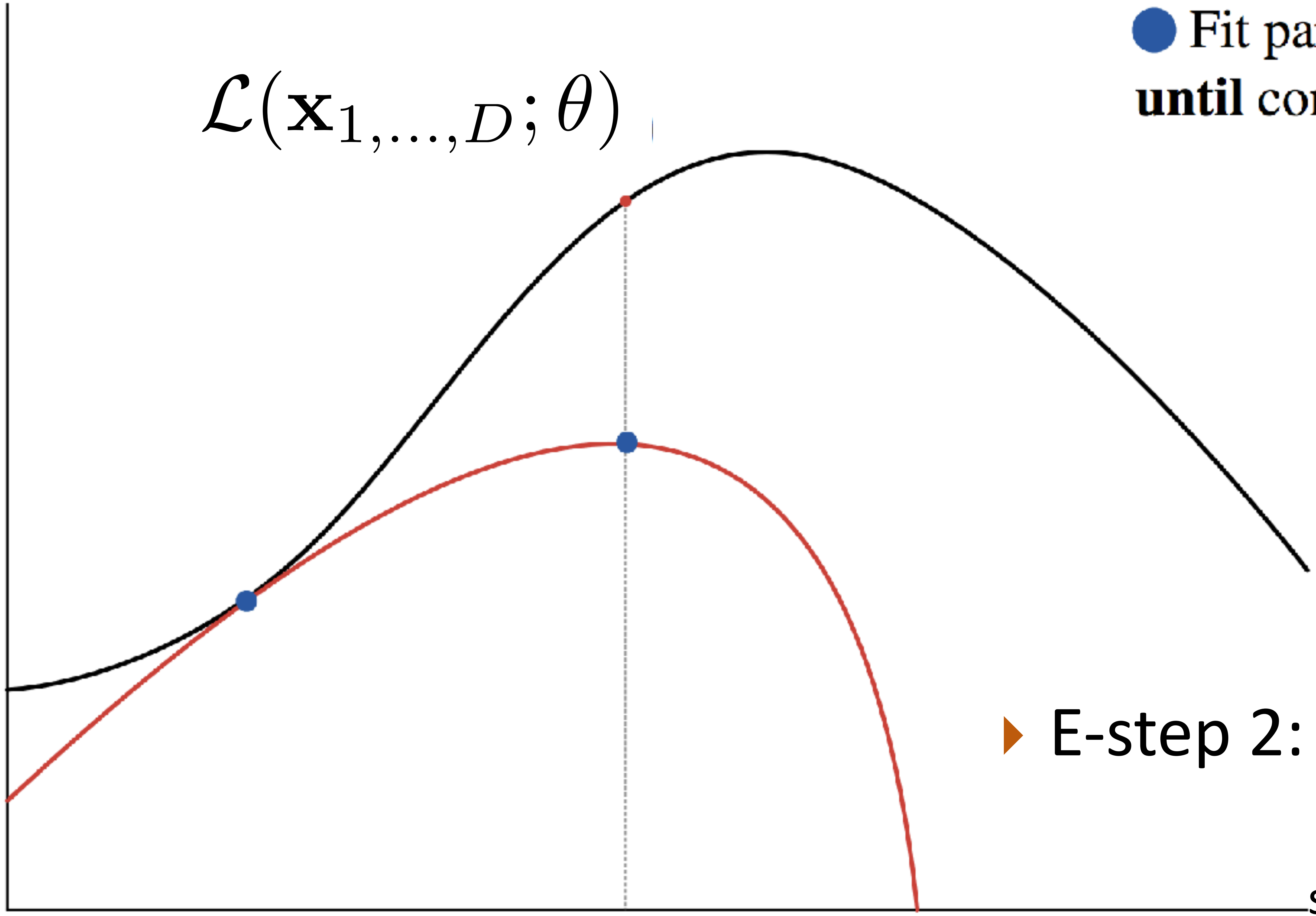


slide credit: Taylor Berg-Kirkpatrick

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

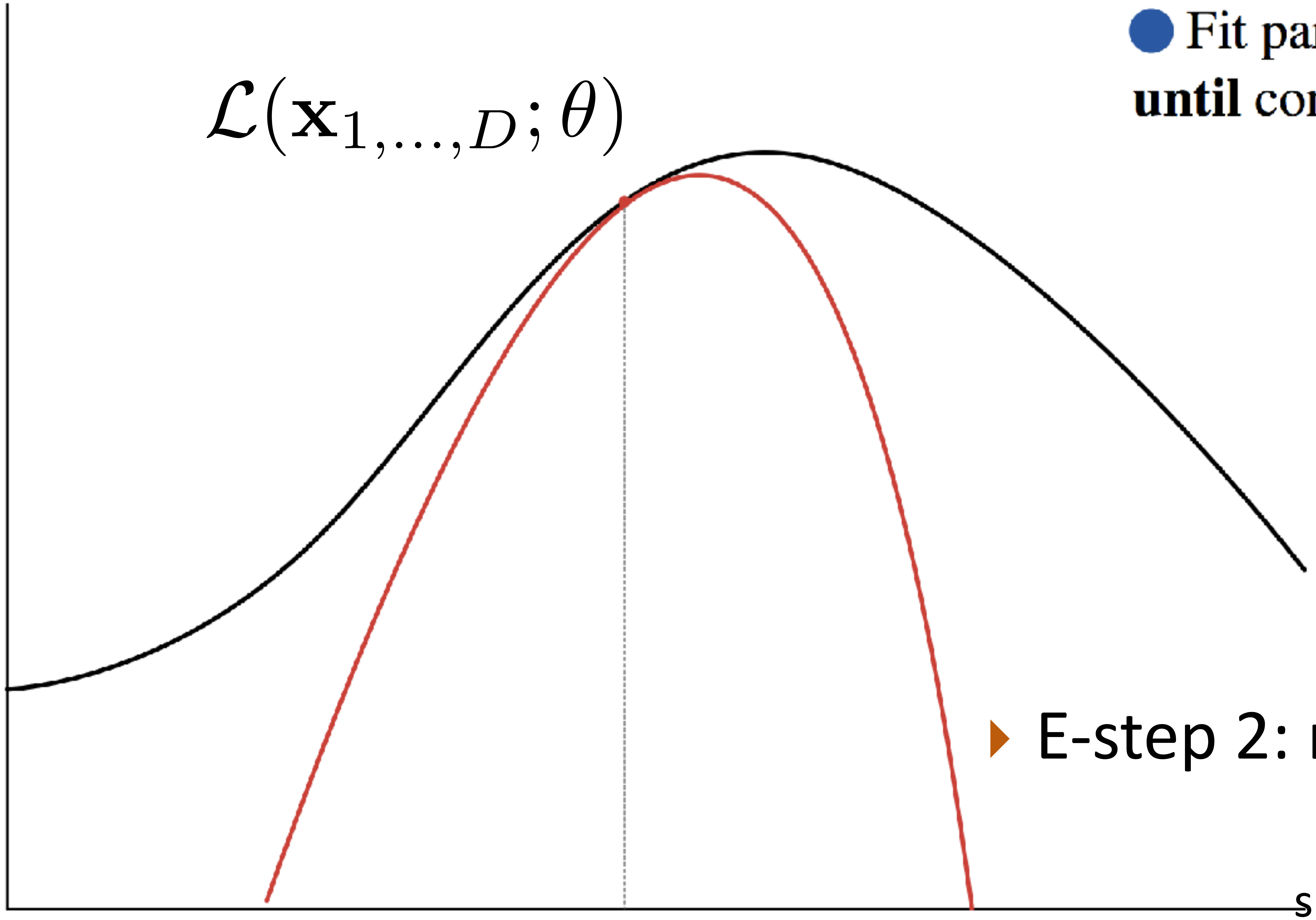


- E-step 2: re-estimate *q*

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*

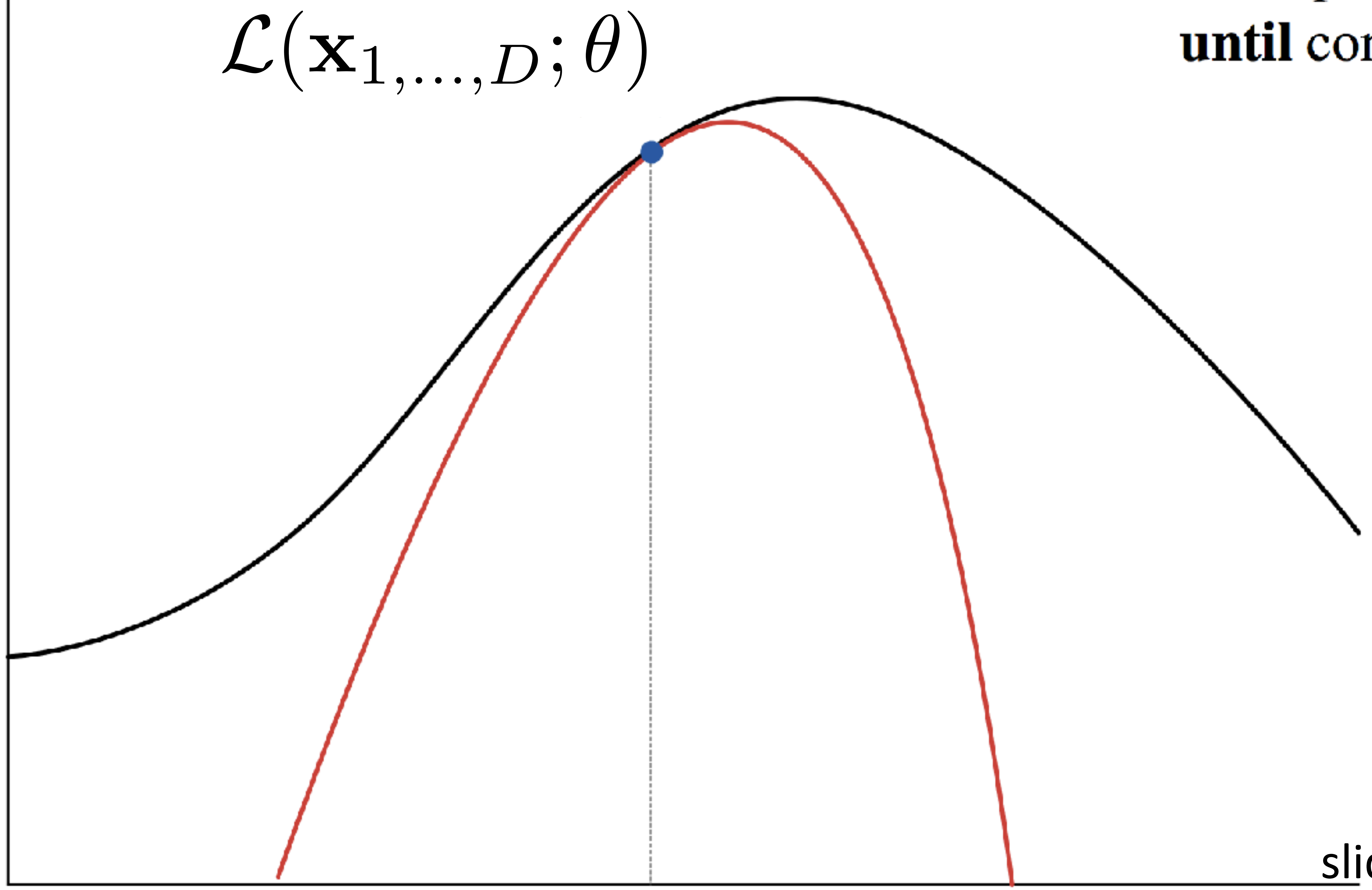
slide credit: Taylor Berg-Kirkpatrick

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 $\boldsymbol{\theta}$

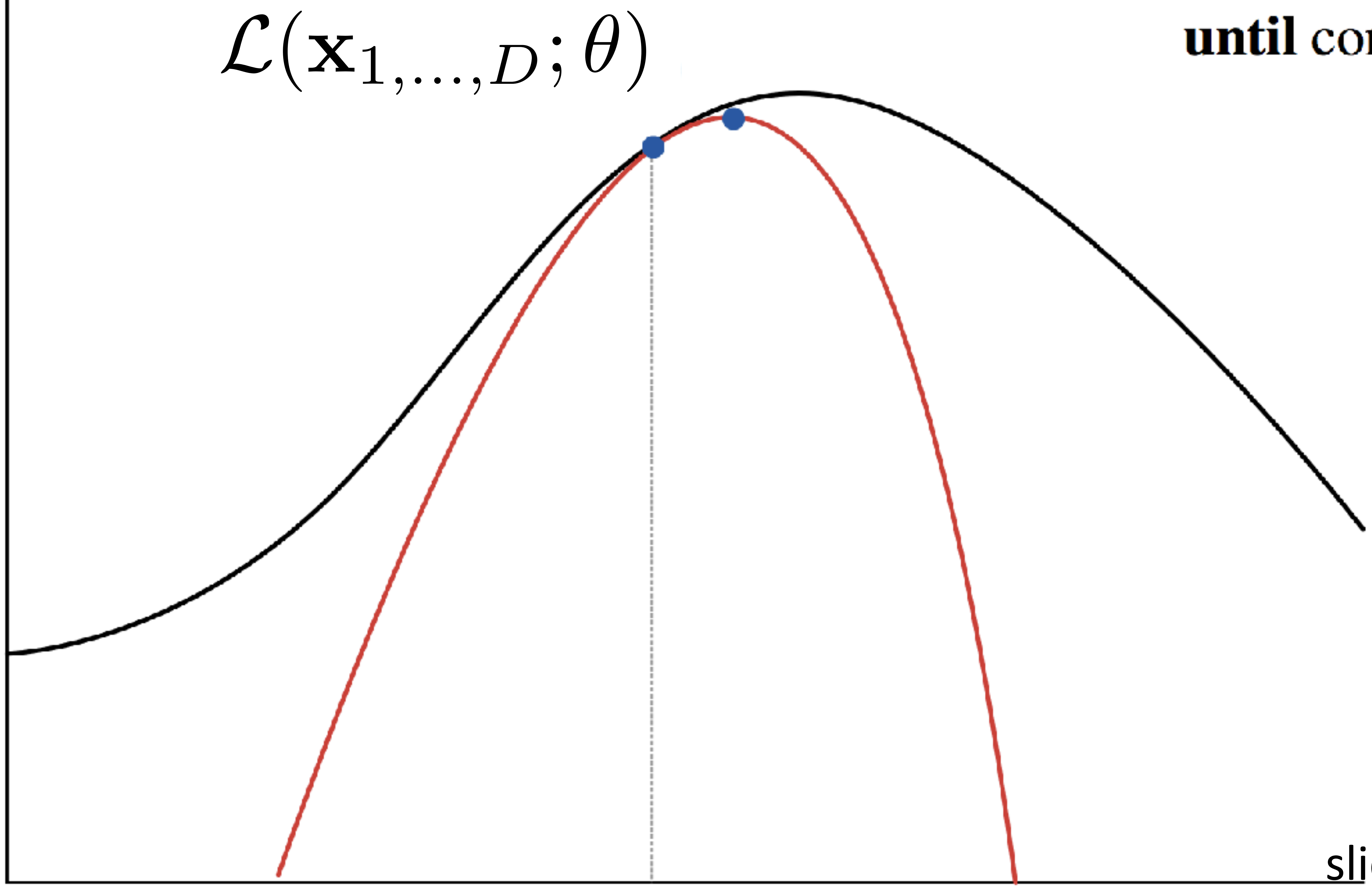
until convergence



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

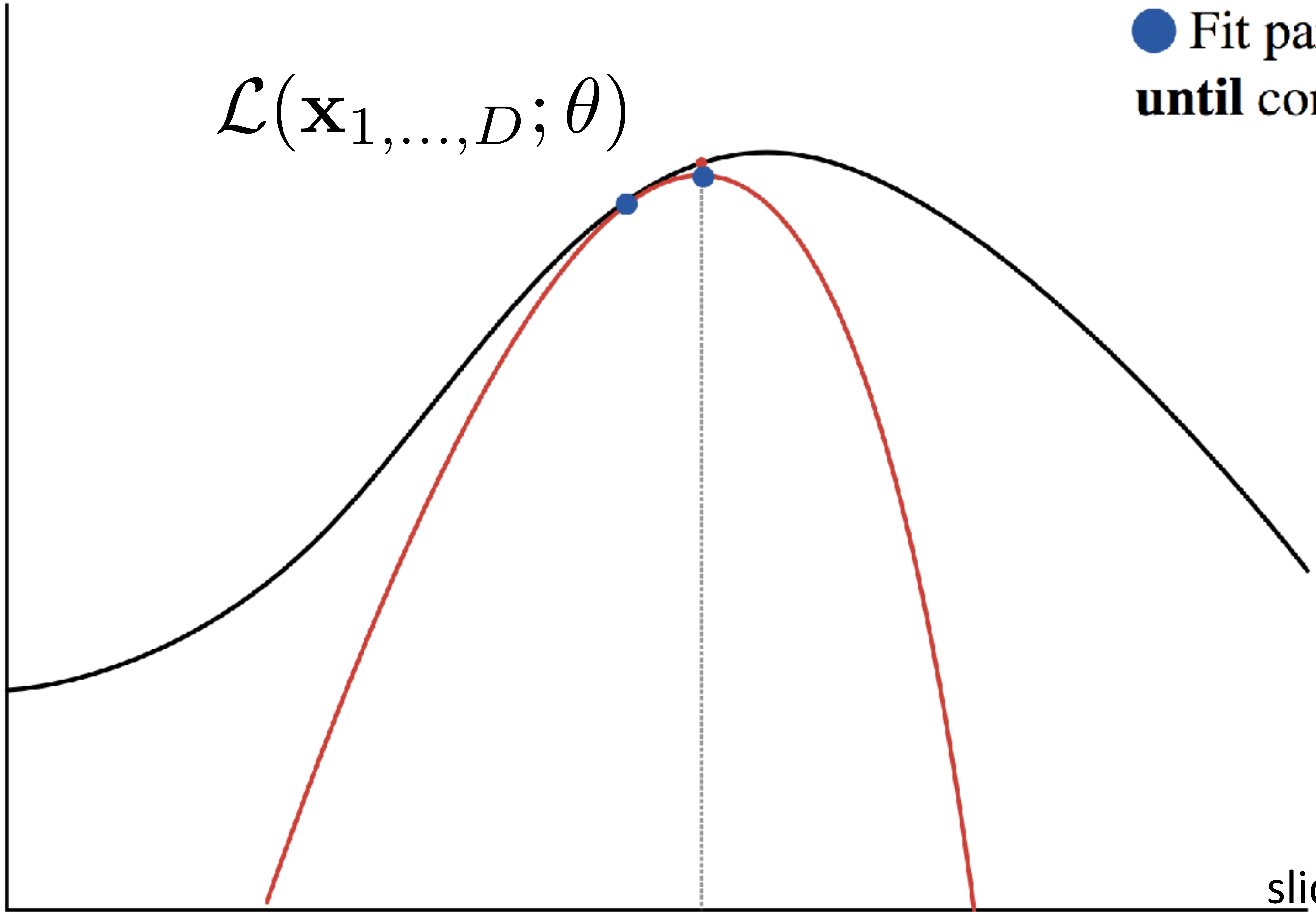
- Compute expected counts $\mathbf{e}$
- Fit parameters $\boldsymbol{\theta}$

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
- (red) Compute expected counts $\mathbf{e}$
- (blue) 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

Iter	Number of tagged sentences used for the initial model						
	0	100	2000	5000	10000	20000	all
	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?

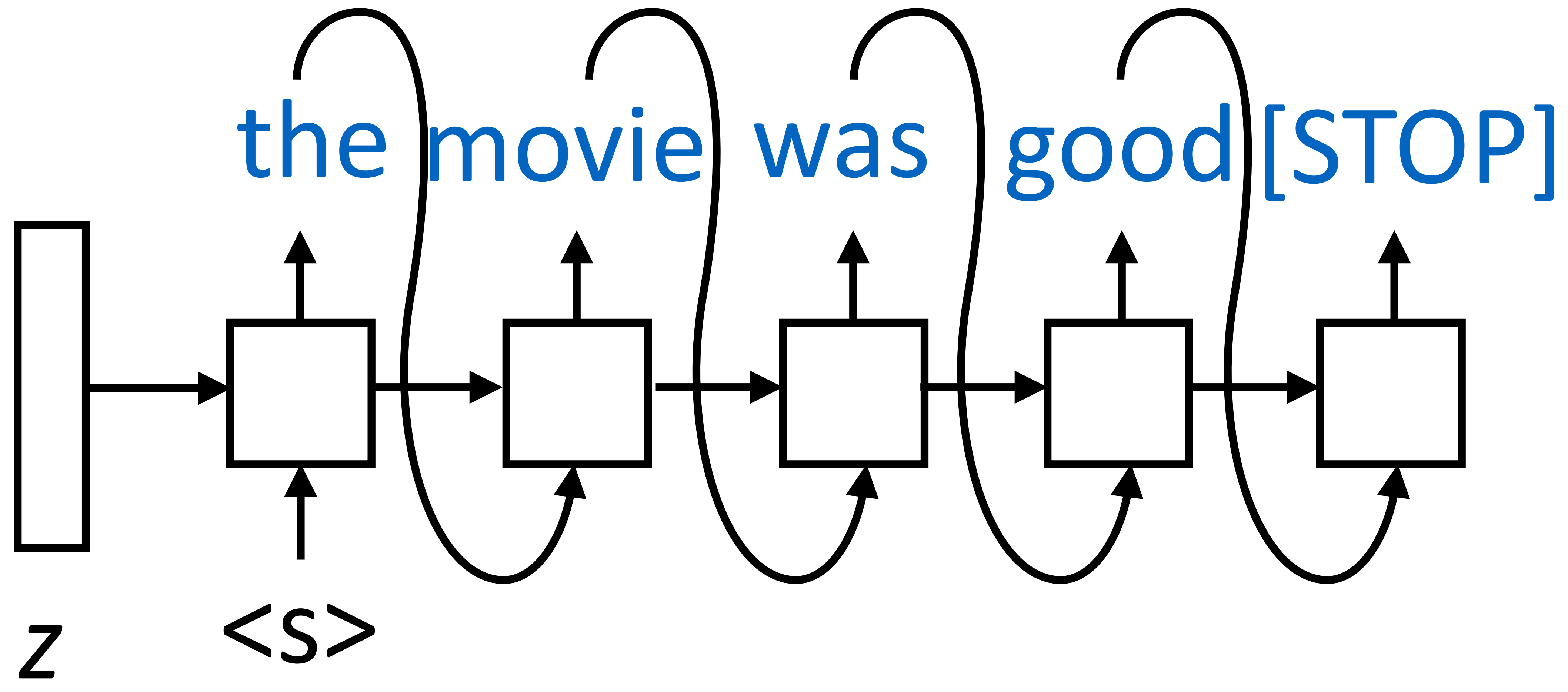
$$P(z, \mathbf{x}) = P(z)P(\mathbf{x}|z)$$

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

- 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"

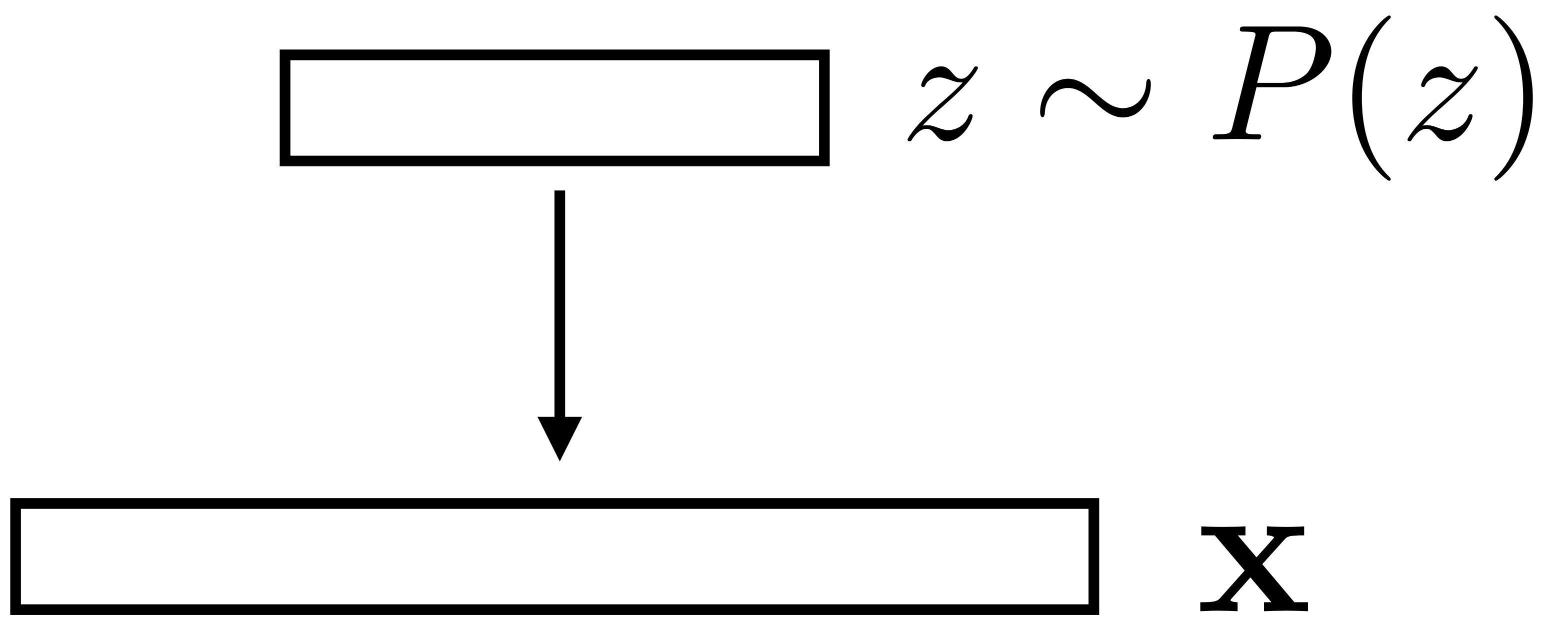
(discriminative)

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

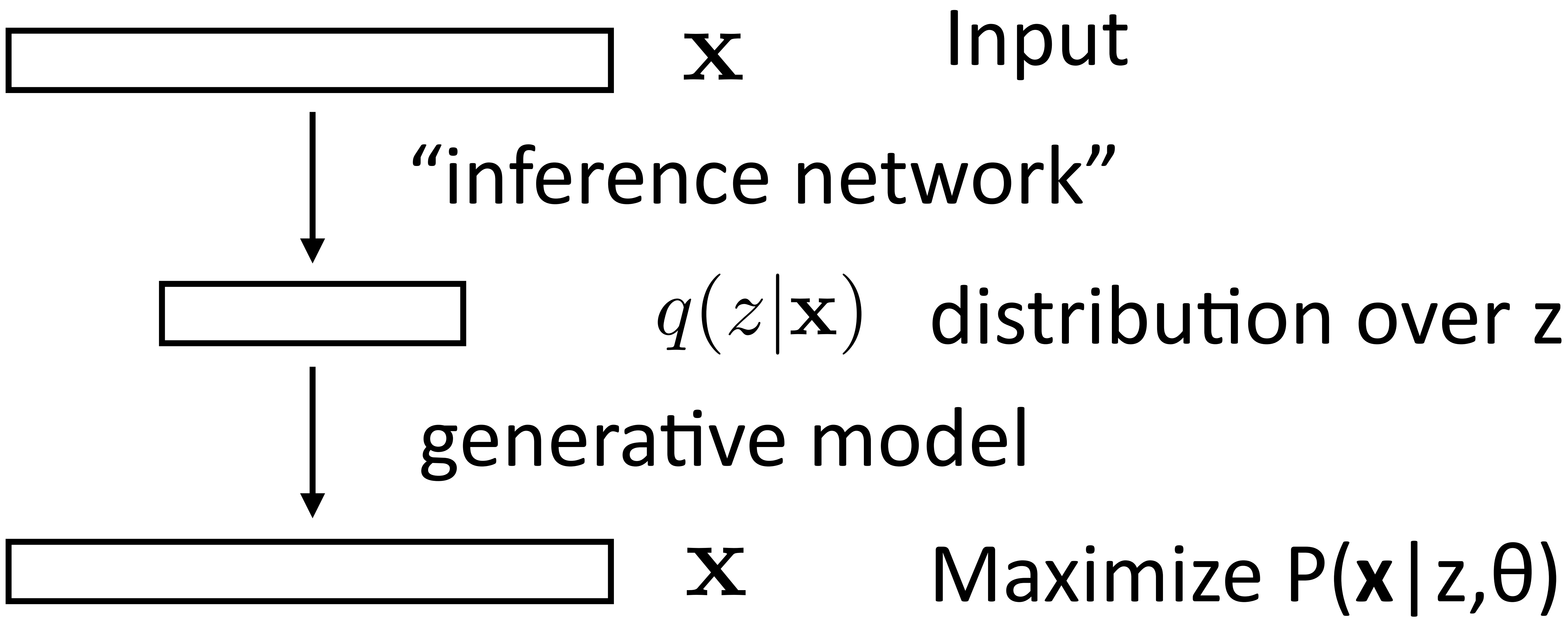
Variational Autoencoders

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

Generative model (test):



Autoencoder (training):



Miao et al. (2015)

Training VAEs

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

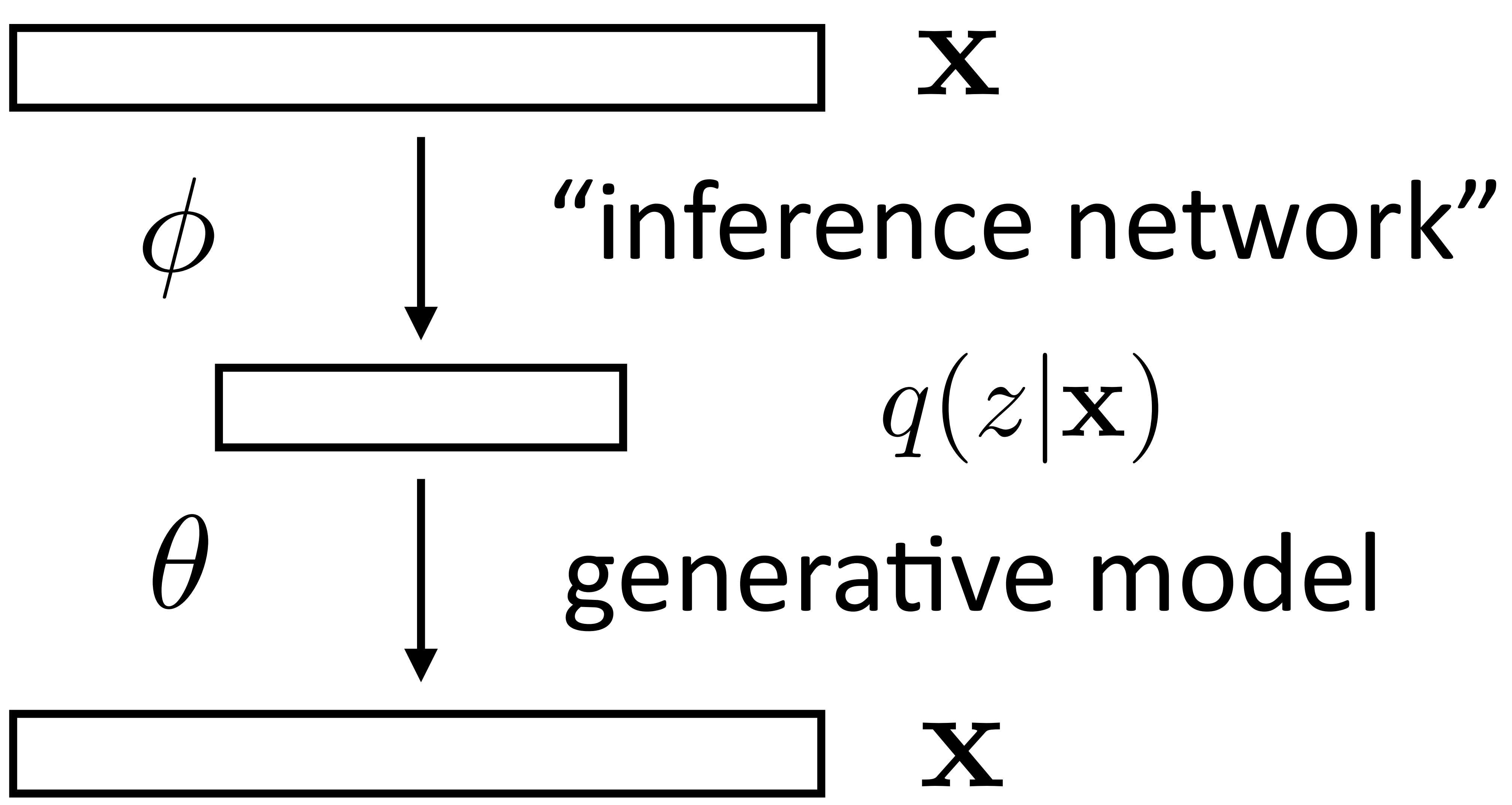
- Choose q to be Gaussian with parameters that are computed from **x**

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

- 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)

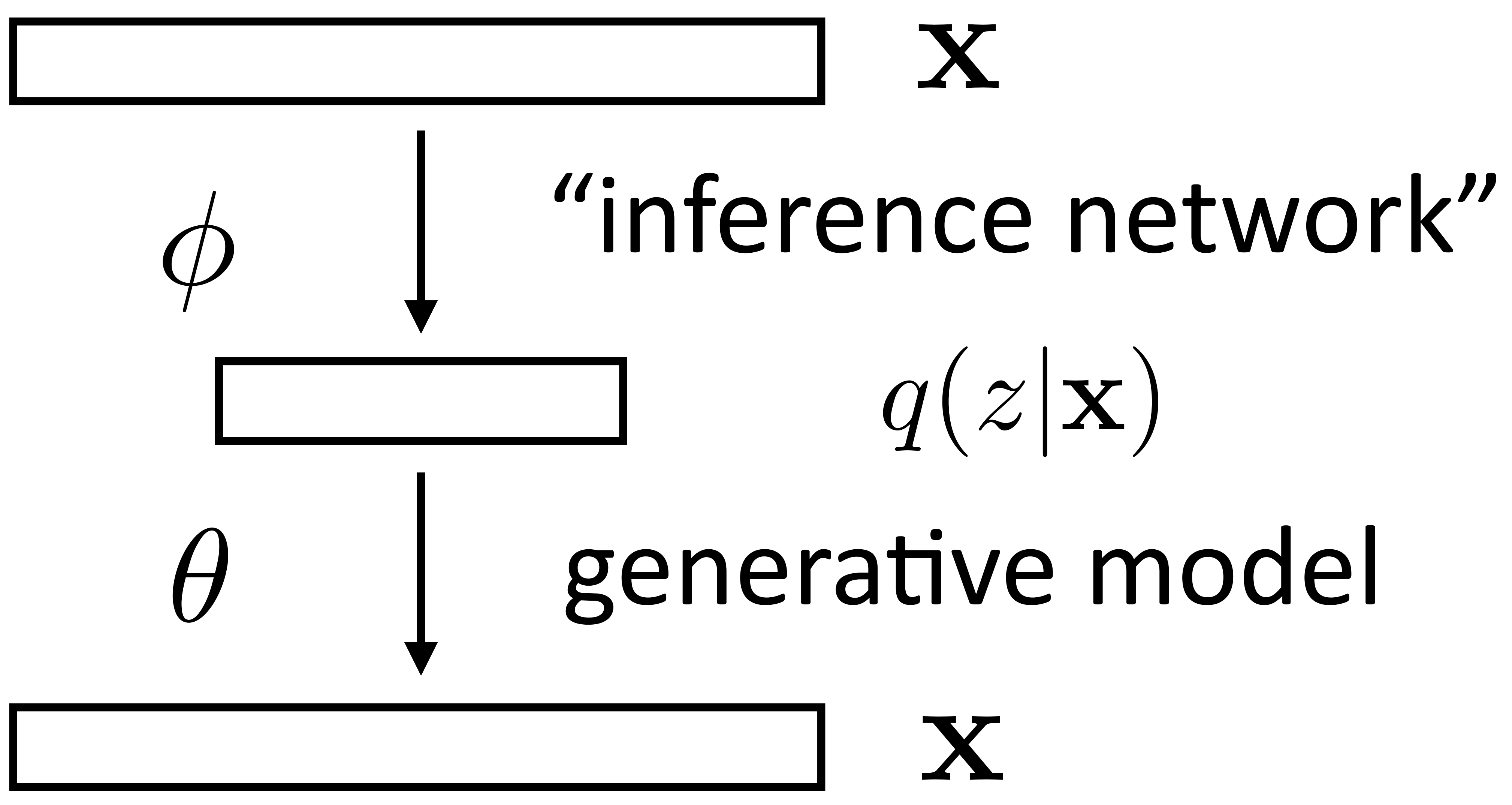
For some number of samples

Sample z ~ *q*

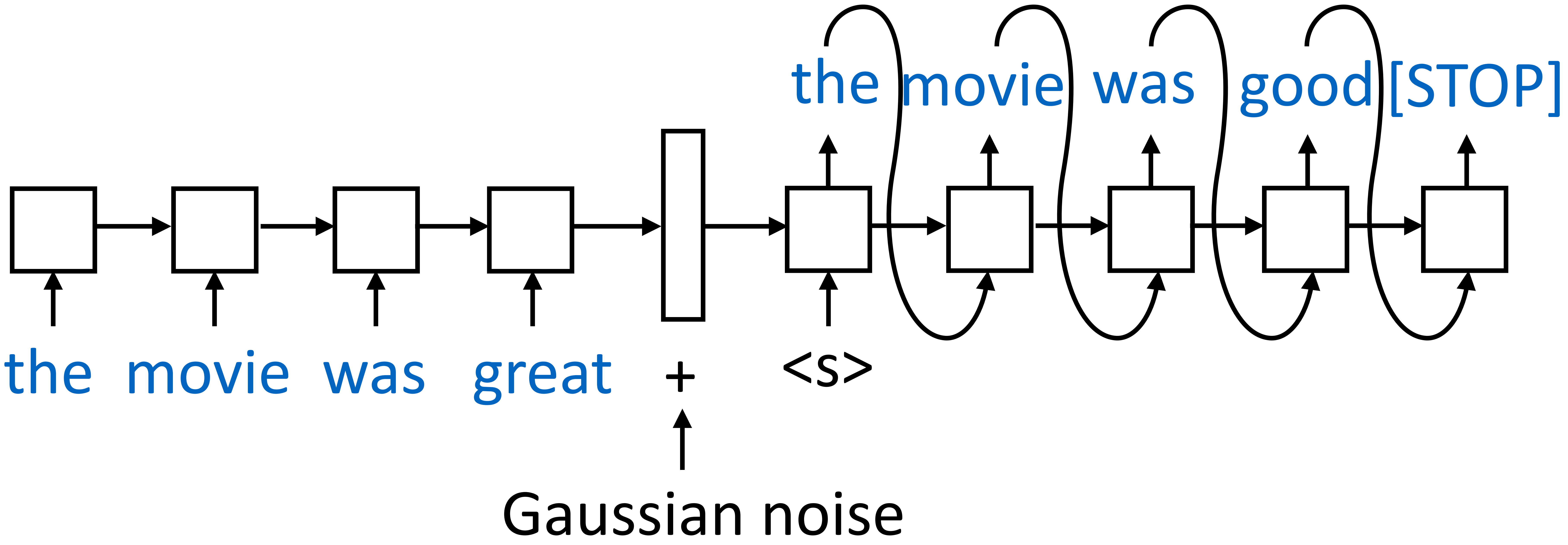
Compute P(**x**|z) and compute loss

Backpropagate to update phi, theta

Autoencoder (training):



Autoencoders

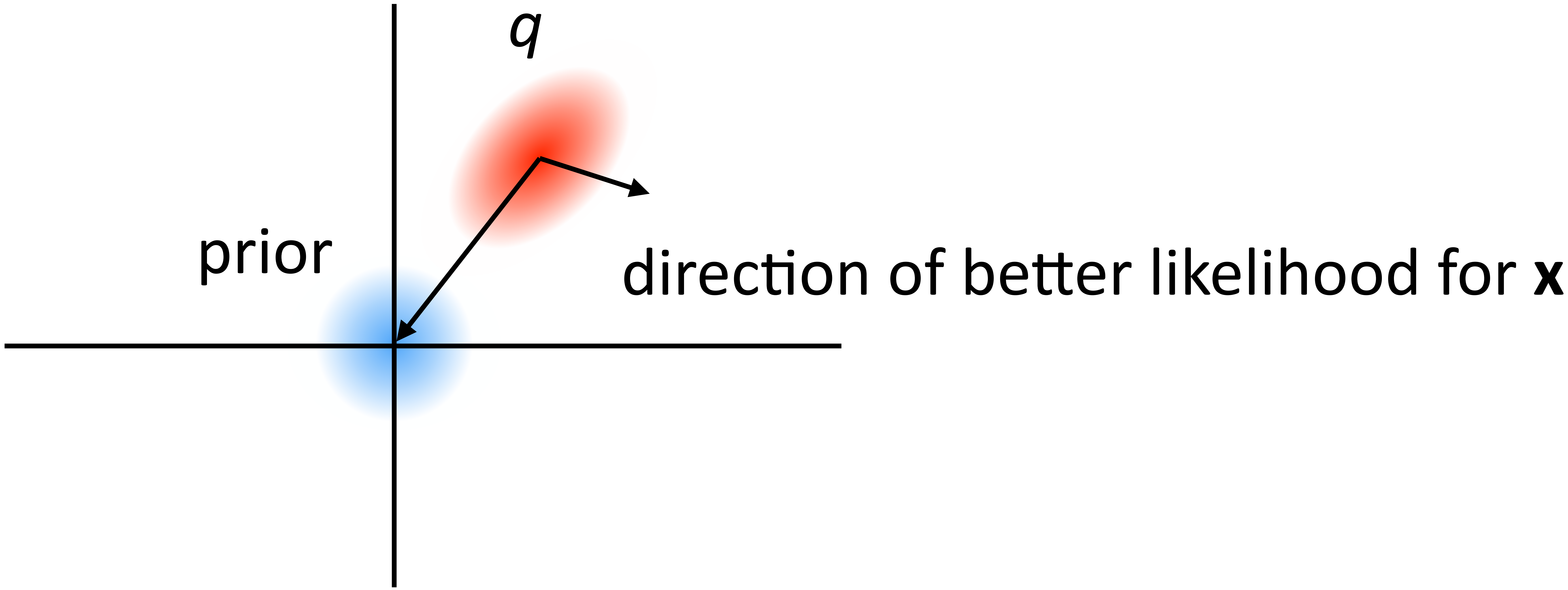


- Another interpretation: train an autoencoder and add Gaussian noise
- Same computation graph as VAE, add KL divergence term to make the objective the same
- Inference network (q) is the encoder and generator is the decoder

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)

Self-Supervision / Transfer Learning

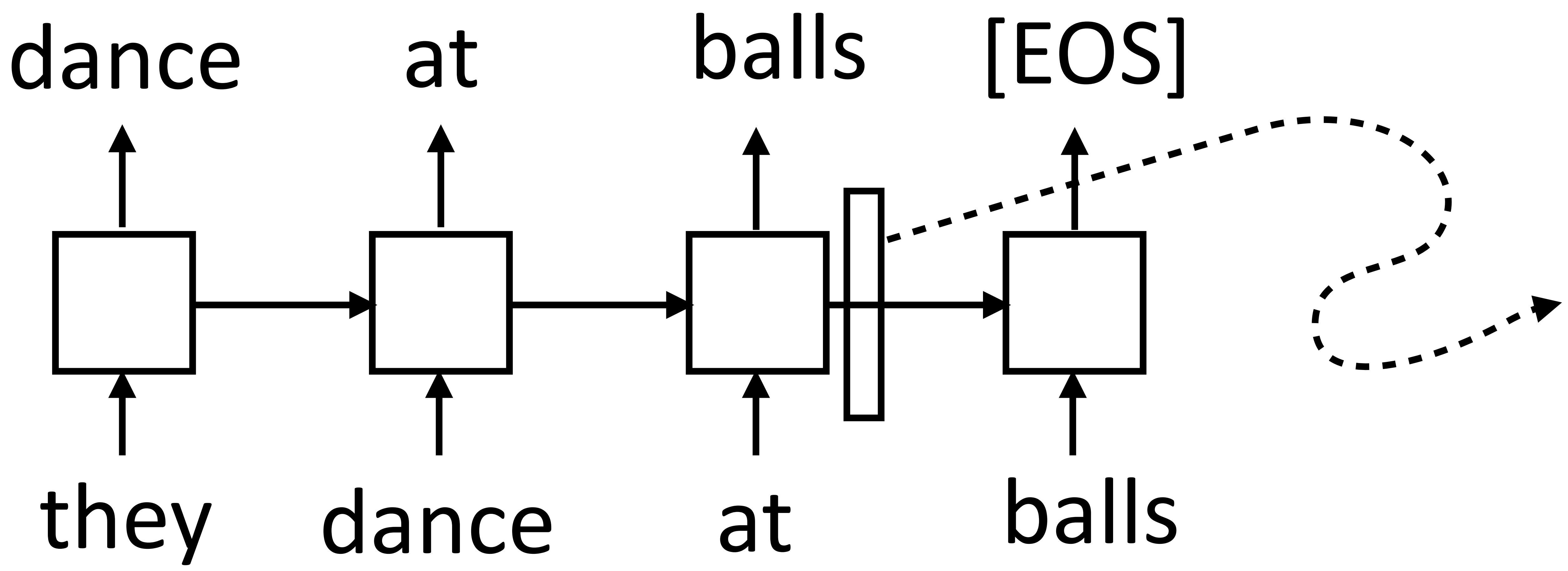
Goals of Unsupervised Learning

- 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

BERT

- Text "infilling" task: replace 15% of tokens with something else and try to predict the original
  - 80% of the time: MASK; 10%: random word; 10%: keep same

I went to the ***store*** and bought ***a*** gallon of ***milk*** . My ***favorite*** kind is 2% .

↑

Transformer (12-24 layers)

↑

I went to the **MASK** and bought **MASK** gallon of ***dog*** . My **MASK** kind is 2% .

- Also generate "fake" sentence pairs and try to predict real from fake

I went to the **MASK** and bought **MASK** gallon of ***dog*** . ***I love karaoke!***

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**

- Dramatic gains on a range of sentence pair / single sentence tasks: paraphrase identification, entailment, sentiment, textual similarity, …
- Not a generative model! But learns really effective representations…

Unsupervised Learning

- Discrete linguistic structure with generative models: unsupervised POS induction
  - These models are hard to learn in an unsupervised way and too impoverished to really be all that useful
- Continuous structure with generative models: variational autoencoders
  - Useful, but also hard to learn in practice
- Continuous structure with "discriminative" models
  - ELMo / BERT seem extremely useful

Takeaways

- EM sort of works for POS induction
- VAE can learn sentence representations
- Language modeling or text infilling as pretraining seems best — arguably not "unsupervised" but the annotation is free
- Using unlabeled data effectively seems like one of the most important directions in NLP right now
- Next time: Jessy Li guest lecture on discourse