Latent variable models and EM-algorithm

Dmitry Vetrov
Research professor at HSE
Head of ML lab in SAIC-Moscow

Deep|Bayes

Outline

- Mixtures of distributions
- EM-algorithm
- Discrete and continuous latent variables
- Case study: AdaGram

Latent variable modeling: example

- Consider the following problem
- We have a set of points generated from a Gaussian

$$x_i \sim \mathcal{N}(x_i|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

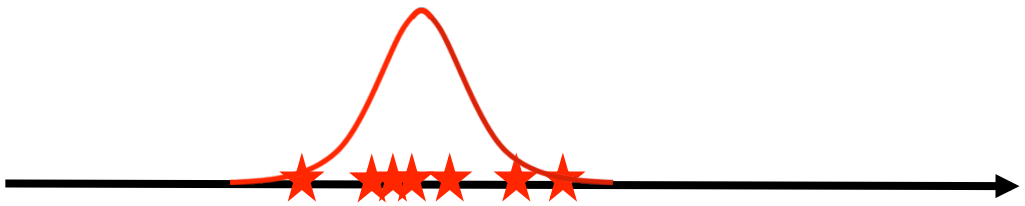
- We need to estimate its parameters μ and σ

Latent variable modeling: example

- Consider the following problem
- We have a set of points generated from a Gaussian

$$x_i \sim \mathcal{N}(x_i|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

- We need to estimate its parameters μ and σ



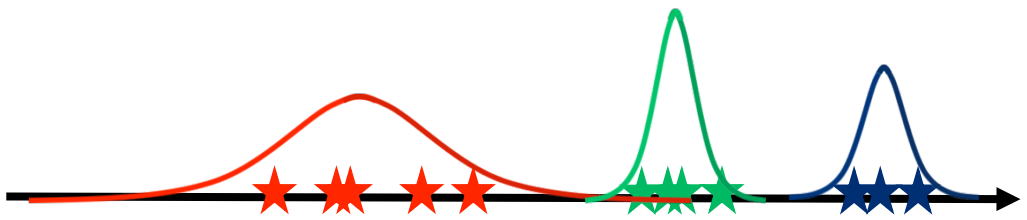
- Solution is simple: we estimate sample mean and variance

Latent variable modeling: example

- Now suppose we're given several sets of points from different gaussians
- We need to estimate the parameters of those gaussians and their weights

Latent variable modeling: example

- Now suppose we're given several sets of points from different gaussians
- We need to estimate the parameters of those gaussians and their weights



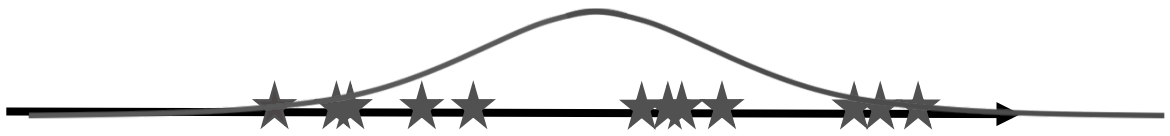
- The problem is as easy if we know what objects were generated from each gaussian

Latent variable modeling: example

- Now what if we do not know what objects were generated by each gaussian
- Of course we could still try to use a single gaussian model…

Latent variable modeling: example

- Now what if we do not know what objects were generated by each gaussian
- Of course we could still try to use a single gaussian model…
- … but there is a better way: latent variable model!



Mixture of gaussians

- For each object x_i we establish additional latent variable z_i which denotes the index of gaussian from which i-th object was generated
- Then our model is

$$p(X, Z|\theta) = \prod_{i=1}^{n} p(x_i, z_i|\theta) = \{\text{Product rule}\} = \prod_{i=1}^{n} p(x_i|z_i, \theta)p(z_i|\theta) = \prod_{i=1}^{n} \pi_{z_i}\mathcal{N}(x_i|\mu_{z_i}, \sigma^2_{z_i})$$

- Here $\pi_j = p(z_i = j)$ are prior probability of j-th gaussian and $\theta = \{\mu_j, \sigma_j, \pi_j\}_{j=1}^{K}$ are the parameters to be estimated
- If we know both X and Z we obtain explicit ML-solution:

$$\theta_{ML} = \arg\max_{\theta} p(X, Z|\theta) = \arg\max_{\theta} \log p(X, Z|\theta)$$

Mixture of gaussians

- What if we do not know Z? Then we need to maximize w.r.t. θ the log of incomplete likelihood

$$\log p(X|\theta)$$

Mixture of gaussians

- What if we do not know Z? Then we need to maximize w.r.t. θ the log of incomplete likelihood

$$\log p(X|\theta) = \int q(Z) \log \frac{p(X, Z|\theta)}{q(Z)} dZ + \int q(Z) \log \frac{q(Z)}{p(Z|X, \theta)} dZ$$

Mixture of gaussians

- What if we do not know Z? Then we need to maximize w.r.t. θ the log of incomplete likelihood

$$\log p(X|\theta) = \boxed{\int q(Z) \log \frac{p(X, Z|\theta)}{q(Z)} dZ} + \boxed{\int q(Z) \log \frac{q(Z)}{p(Z|X, \theta)} dZ} =$$

Always non-negative!

Variational lower bound (ELBO)

$$\mathcal{L}(q, \theta) + KL(q||p) \geq \mathcal{L}(q, \theta)$$

Mixture of gaussians

- What if we do not know Z? Then we need to maximize w.r.t. θ the log of incomplete likelihood

$$\log p(X|\theta) = \boxed{\int q(Z) \log \frac{p(X, Z|\theta)}{q(Z)} dZ} + \boxed{\int q(Z) \log \frac{q(Z)}{p(Z|X, \theta)} dZ} =$$

Always non-negative!

Variational lower bound (ELBO)

$$\mathcal{L}(q, \theta) + KL(q||p) \geq \mathcal{L}(q, \theta)$$

- Instead of optimizing $\log p(X|\theta)$ we optimize variational lower bound $\mathcal{L}(q, \theta)$ w.r.t. both θ and $q(Z)$

- The block-coordinate algorithm is known as EM-algorithm

Variational lower bound

Definition. Function $g(\xi, x)$ is called variational lower bound for function $f(x)$ iff

- For all ξ for all x it follows $f(x) \geq g(\xi, x)$
- For any x_0 there exists $\xi(x_0)$ such that $f(x_0) = g(\xi(x_0), x_0)$

If we managed to find such variational lower bound then instead of solving

$$f(x) \to \max_x$$

we may iteratively perform block-coordinate updates of $g(\xi, x)$

$$x_n = \arg\max_x g(\xi_{n-1}, x), \quad \xi_n = \xi(x_n) = \arg\max_\xi g(\xi, x_n)$$

EM algorithm

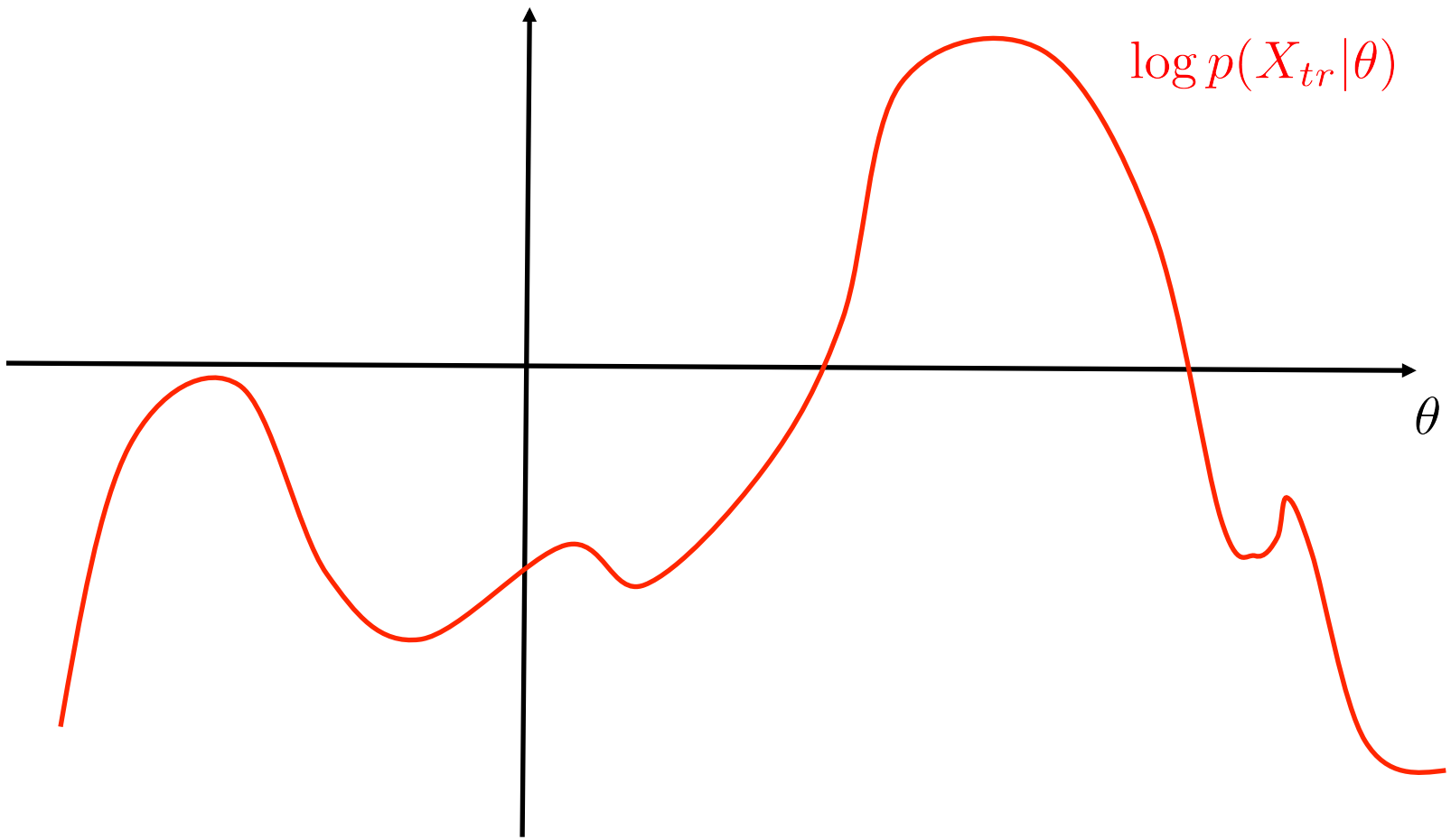
- To solve
$$\mathcal{L}(q,\theta) = \int q(Z) \log \frac{p(X,Z|\theta)}{q(Z)} dZ \to \max_{q,\theta}$$
we start from initial point θ_0 and iteratively repeat

- E-step: find
$$q(Z) = \arg\max_q \mathcal{L}(q,\theta_0) = \arg\min_q KL(q||p) = p(Z|X,\theta_0)$$

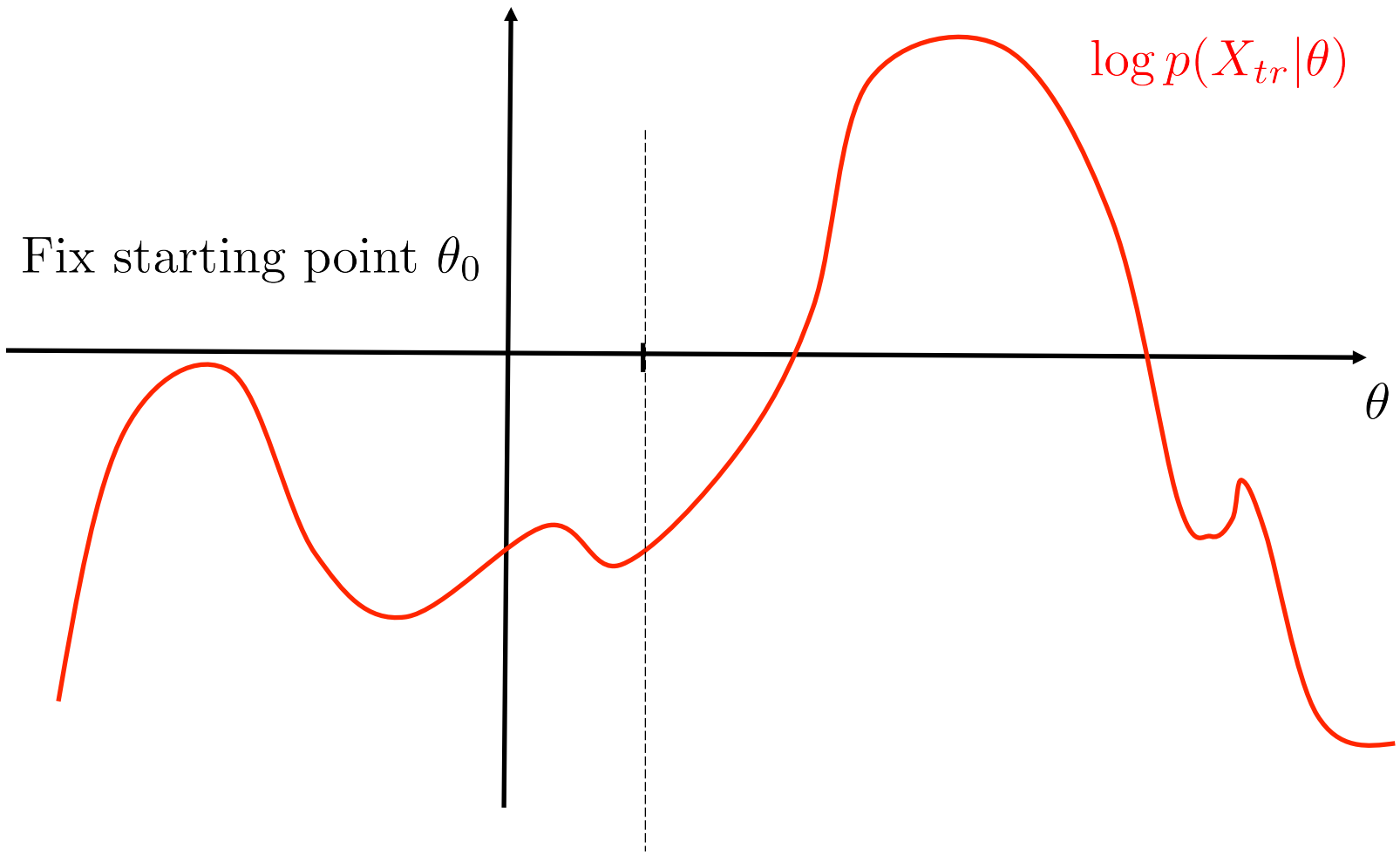
- M-step: solve
$$\theta_* = \arg\max_\theta \mathcal{L}(q,\theta) = \arg\max_\theta \mathbb{E}_Z \log p(X,Z|\theta),$$
set $\theta_0 = \theta_*$ and go to E-step until convergence

- The EM algorithm monotonically increases the lower bound and converges to stationary point of $\log p(X|\theta)$

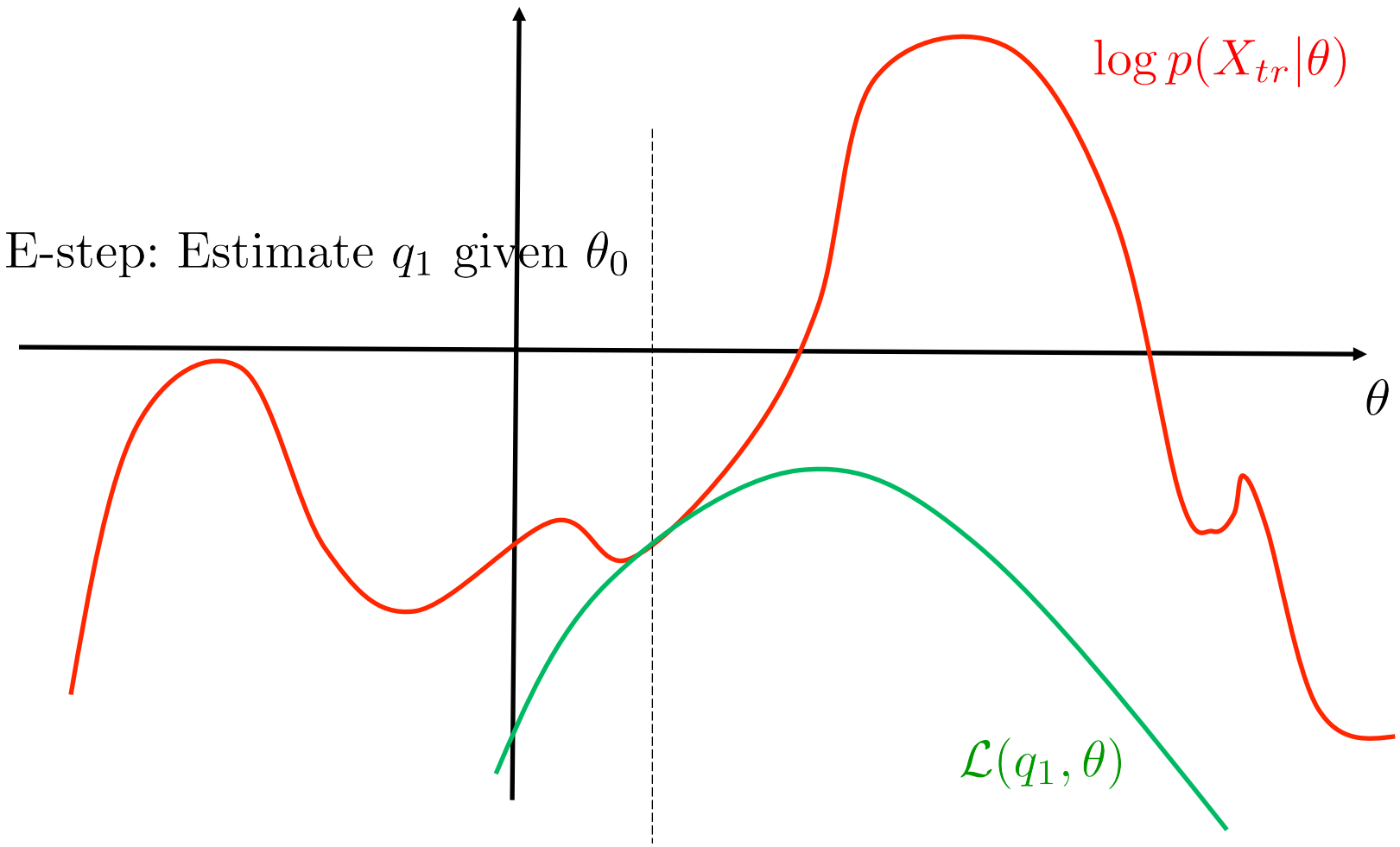
EM-algorithm



EM-algorithm

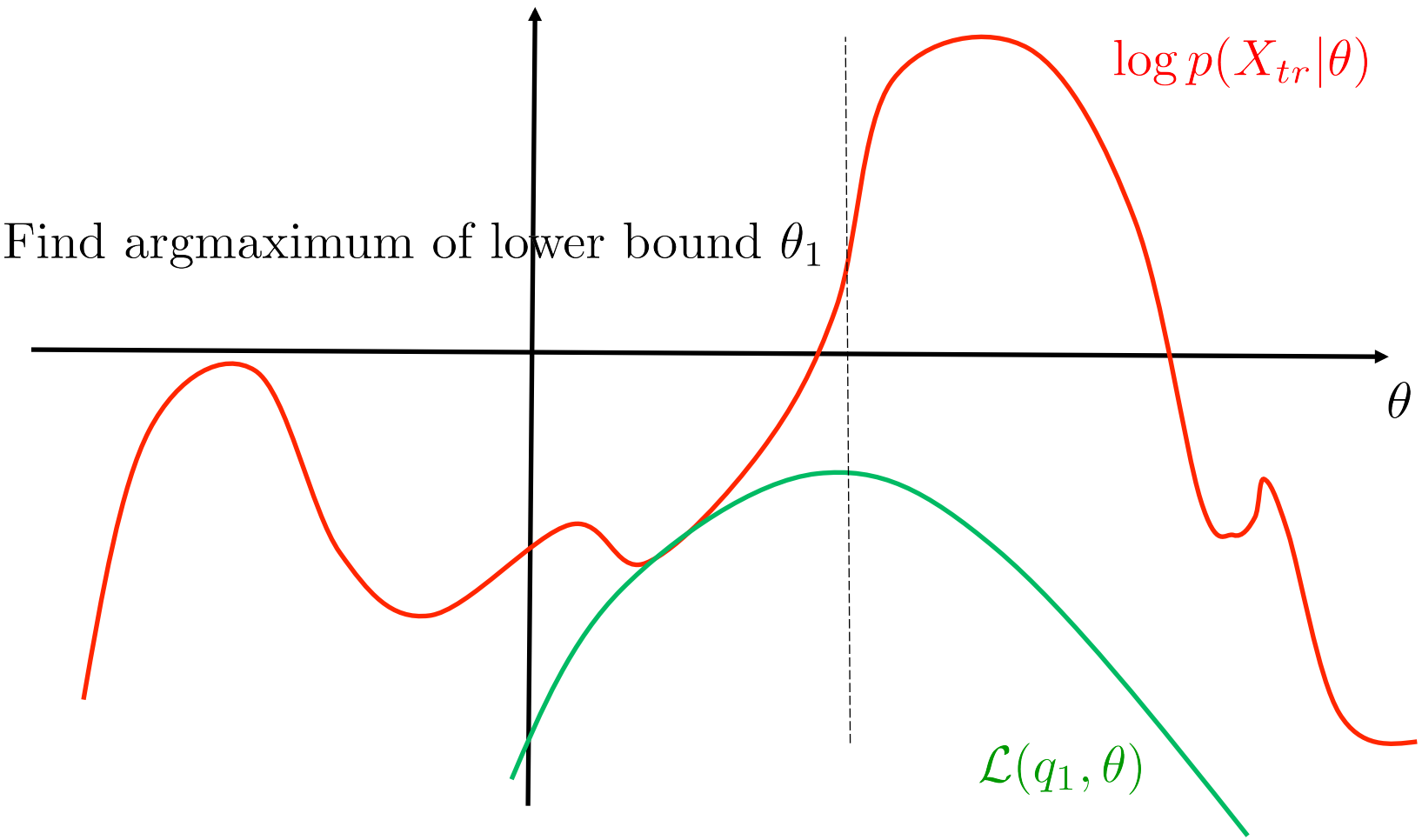


EM-algorithm

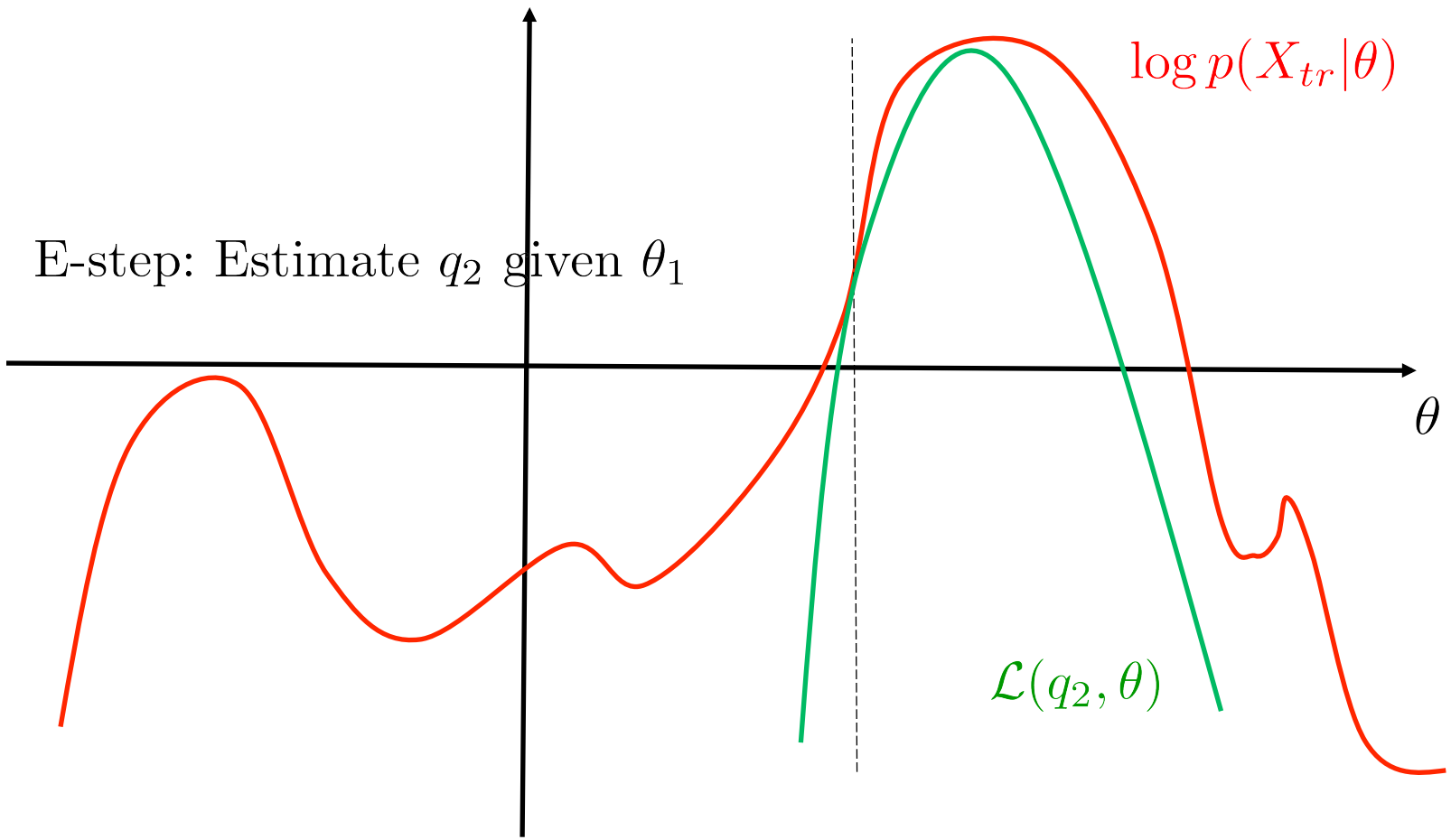


EM-algorithm

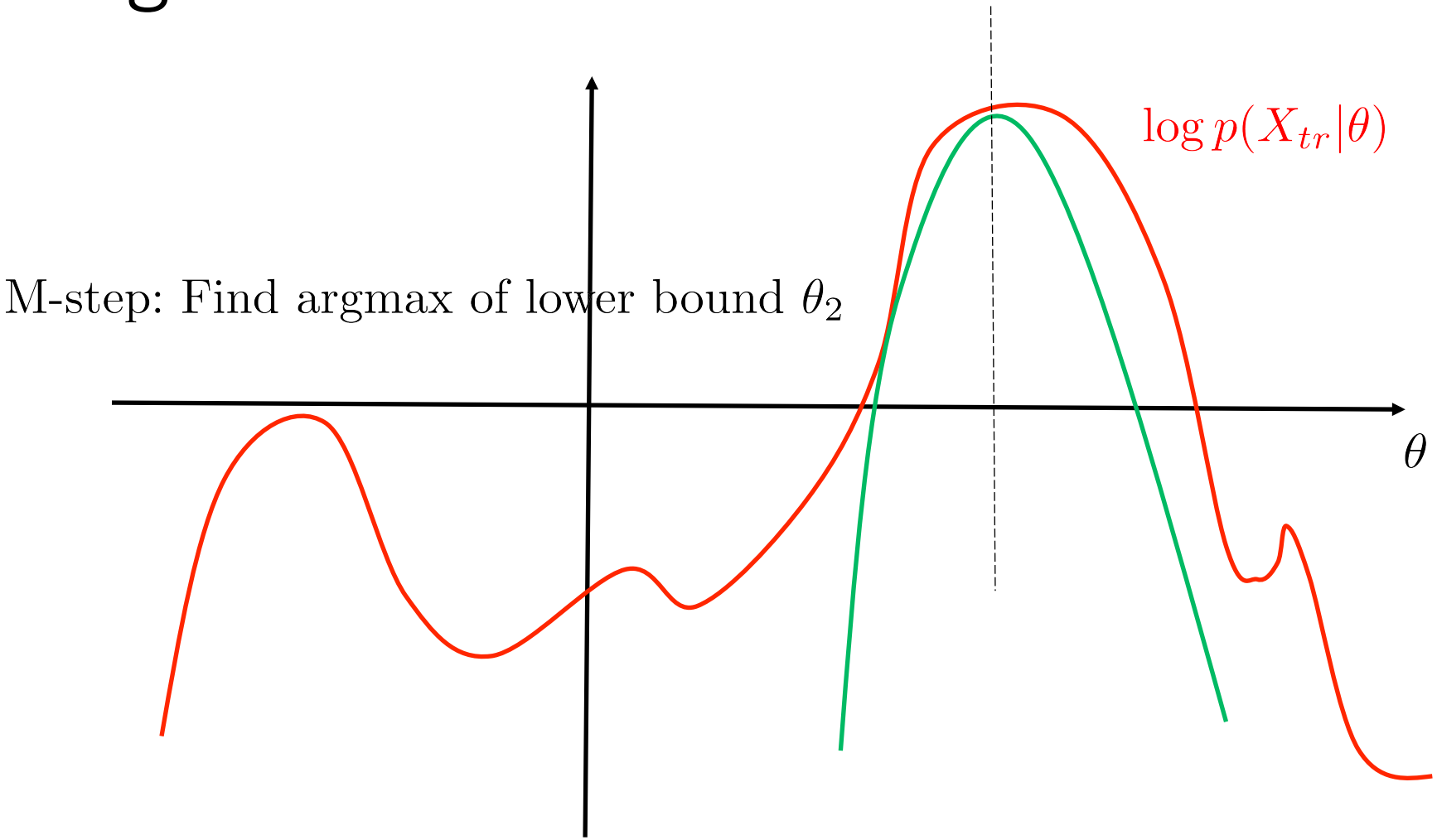
M-step: Find argmaximum of lower bound θ_1



EM-algorithm



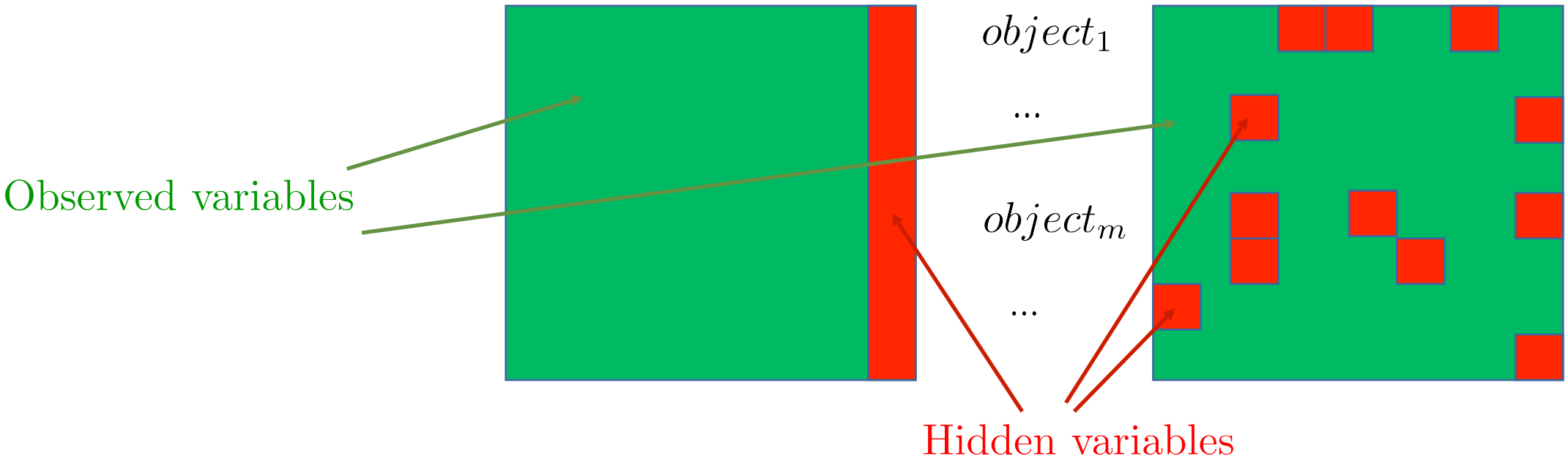
EM-algorithm



Benefits of EM algorithm

- In many cases (e.g. for the mixture of gaussians) E-step and M-steps can be performed in closed form
- Allows to build more complicated models of data using mixtures of simple distributions
- If true posterior $p(Z|X, \theta)$ is intractable we may search for the closest $q(Z)$ among tractable distributions by solving optimization problem
- Allows to process missing data by treating them as latent variables

General nature of EM-framework

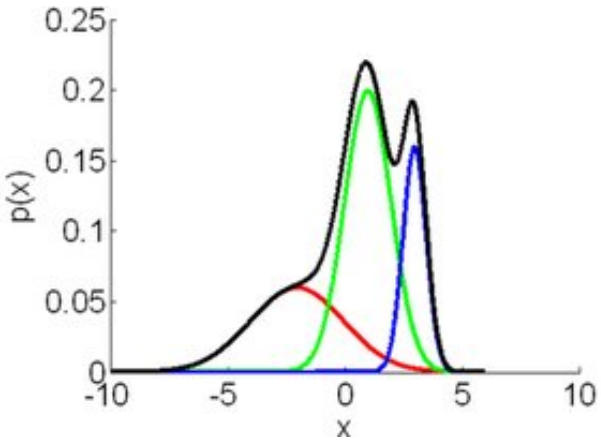


- EM algorithm allows to fill in arbitrary gaps in data
- May deal with both discrete and continuous variables
- Always converges
- Allows multiple extensions

Categorical latent variables

- Assume all $z_i \in \{1, \ldots, K\}$ then marginal

$$p(x_i|\theta) = \sum_{k=1}^{K} p(x_i|k, \theta) p(z_i = k|\theta)$$



is simply a finite mixture of distributions

- If K is not exponentially large E-step can be performed in closed form

$$q(z_i = k) = p(z_i = k|x_i, \theta) = \frac{p(x_i|k, \theta) p(z_i = k|\theta)}{\sum_{l=1}^{K} p(x_i|l, \theta) p(z_i = l|\theta)}$$

- M-step is simply a sum of finite terms

$$\mathbb{E}_Z \log p(X, Z|\theta) = \sum_{i=1}^{n} \mathbb{E}_{z_i} \log p(x_i, z_i|\theta) = \sum_{i=1}^{n} \sum_{k=1}^{K} q(z_i = k) \log p(x_i, k|\theta)$$

Continuous latent variables

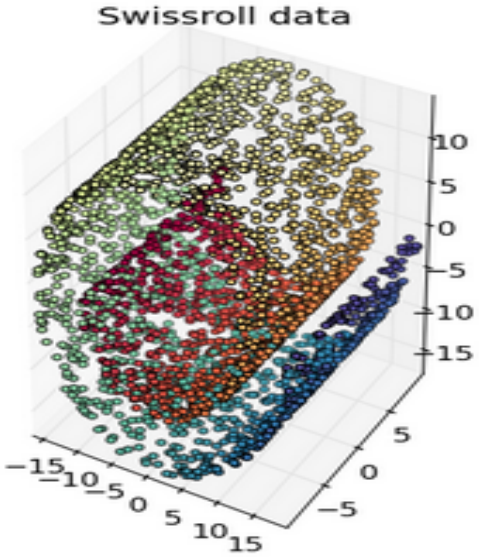
- Continuous varuables can be regarded as a mixture of a continuum of distributions

$$p(x_i|\theta) = \int p(x_i, z_i|\theta)dz_i = \int p(x_i|z_i, \theta)p(z_i|\theta)dz_i$$

- E-step can be done in closed form only in case of **conjugate distributions**, otherwise the true posterior is intractable

$$q(z_i) = p(z_i|x_i, \theta) = \frac{p(x_i|z_i, \theta)p(z_i|\theta)}{\int p(x_i|z_i, \theta)p(z_i|\theta)dz_i}$$

- Typically continuous latent variables are used for dimension reduction also known as **representation learning**



Conceptual idea of EM algorithm

- Build a probabilistic parametric model of data
- Include additional (latent variables) until model becomes simple enough, e.g. belongs to exponential class
- Treat all missing values in data as latent variables
- When fitting the model to data (e.g. using maximal likelihood estimation) run EM
- Estimate a distribution on latent variables
- Maximize the expectation w.r.t. latent variables of joint log-likelihood w.r.t. parameters

Difficult cases

- Each object has multi-dimensional discrete latent variable $\Rightarrow$ exponentially large sums
- Object has both discrete and continuous latent variables (e.g. mixture of low-dimensional manifolds) $\Rightarrow$ mixed discrete-continuous distributions over latent variables
- Continuous latent variables come from non-conjugate priors $\Rightarrow$ intractable multi-dimensional intergrals

Difficult cases

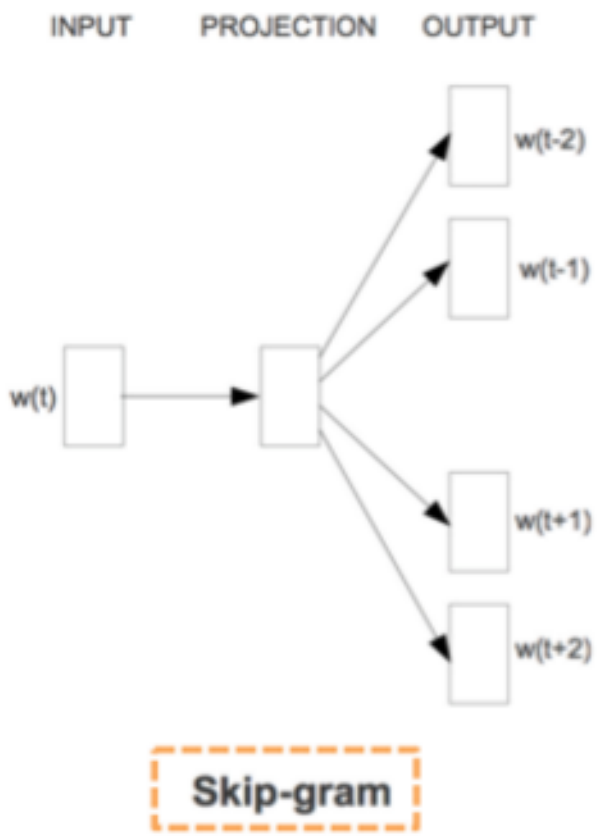
- Each object has multi-dimensional discrete latent variable $\Rightarrow$ exponentially large sums
- Object has both discrete and continuous latent variables (e.g. mixture of low-dimensional manifolds) $\Rightarrow$ mixed discrete-continuous distributions over latent variables
- Continuous latent variables come from non-conjugate priors $\Rightarrow$ intractable multi-dimensional intergrals

Way out: Variational Bayes

Scalable EM-algorithm

Word2vec model (Mikolov2013)

- Designed for word prediction according to its context
- Transforms words to points in 255-dimensional vector space



Mathematical formulation

... w(t-C) ... w(t) ... w(t+C) ...

X	Y
w(t)	*w(t-C)*
w(t)	*w(t-C+1)*
w(t)	...
w(t)	*w(t+C-1)*
w(t)	*w(t+C)*
w(t+1)	*w(t+1-C)*
...	...

The likelihood function depends on **vector representation** of x

$$p(y|x,\theta) = l(V(x), y, \theta),$$

The parameters and vector representations are tuned by maximal likelihood

$$p(Y|X,\theta) \to \max_{\theta, V(.)}$$

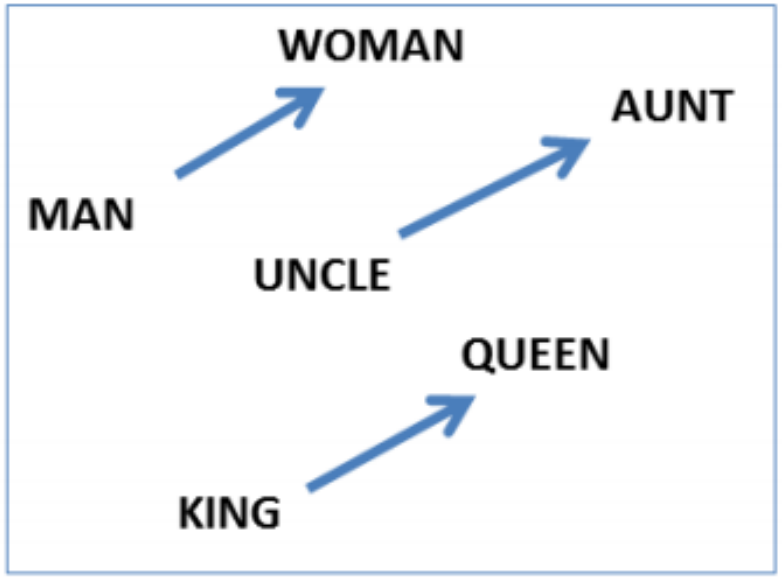
Semantic properties of representations

- Most known property of word2vec model: algebraic operations on vectors correspond to semantic operations on senses:

$$V(\texttt{'Paris'}) - V(\texttt{'France'}) + V(\texttt{'Russia'}) \approx V(\texttt{'Moscow'})$$

  Thousands of examples!

- Word2vec seems to capture notions of gender, geogaphy, number, and many other attributes



Word ambiguity

- Many words have several meanings
- Suppose we see a sentence `The troops met at Waterloo`
- What can we expect from $V(\texttt{'Waterloo'})$?
- But even if we succeed we will not be able to capture semantics of other possible meanings

$$V(\texttt{'Waterloo'}) = ?$$

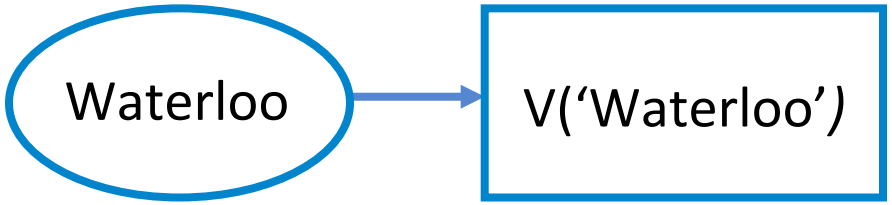
$V(\texttt{'Euston'})$
$V(\texttt{'London'})$
$V(\texttt{'Station'})$

$V(\texttt{'Napoleon'})$
$V(\texttt{'Austerlitz'})$
$V(\texttt{'Battle'})$

$V(\texttt{'ABBA'})$
$V(\texttt{'Mamma mia'})$
$V(\texttt{'Voulez-vous'})$

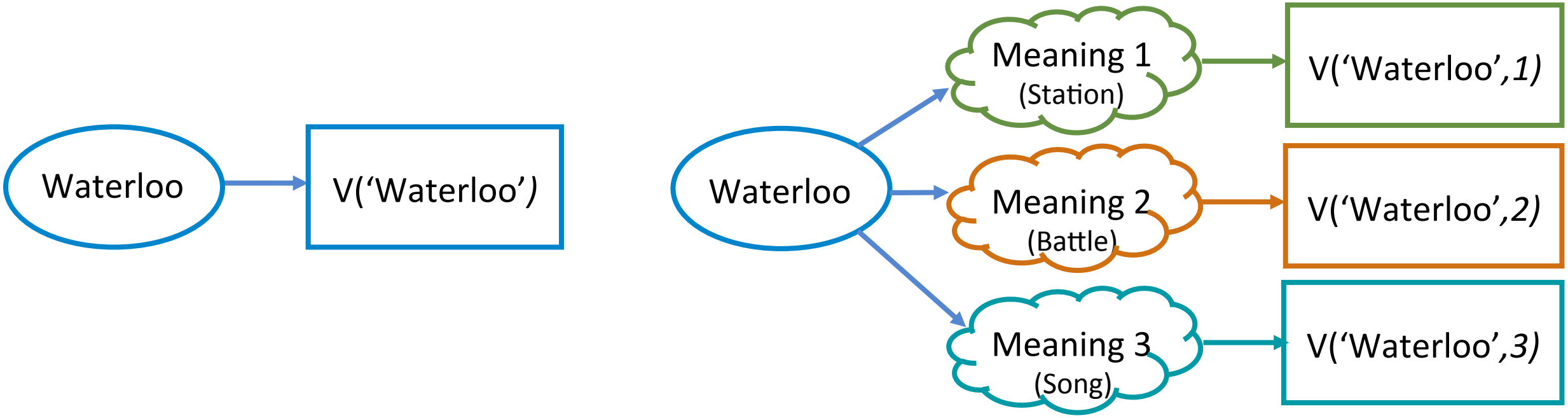
Way out: latent variable model

- Let us associate vector representation $V(.)$ not with a word but with its meaning
- We could expect much higher prediction quality
- What is even better we can perform word desambiguation by comparing what word meaning gave the largest likelihood



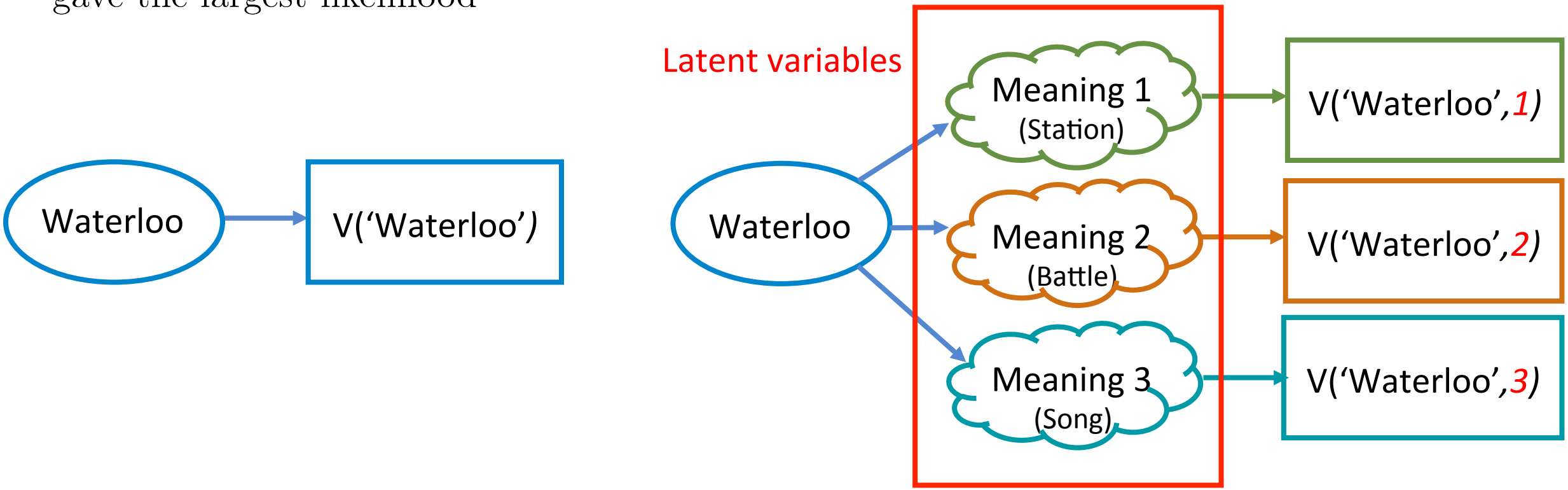
Way out: latent variable model

- Let us associate vector representation $V(.)$ not with a word but with its meaning
- We could expect much higher prediction quality
- What is even better we can perform word desambiguation by comparing what word meaning gave the largest likelihood



Way out: latent variable model

- Let us associate vector representation $V(.)$ not with a word but with its meaning
- We could expect much higher prediction quality
- What is even better we can perform word desambiguation by comparing what word meaning gave the largest likelihood



Multi-sense extension of Ada-Gram

- Denote the index of a particular meaning for a particular word occurence as $z \in \{1, 2, \ldots\}$
- The model takes form

$$p(y|x, z, \theta) = l(V(x, z), y, \theta)$$

- We could train it as before

$$p(Y|X, Z, \theta) \to \max_{V(.,.),\theta}$$

but **we do not know Z at the training stage**

Multi-sense extension of Ada-Gram

- Denote the index of a particular meaning for a particular word occurence as $z \in \{1, 2, \ldots\}$
- The model takes form

$$p(y|x, z, \theta) = l(V(x, z), y, \theta)$$

- We could train it as before

$$p(Y|X, Z, \theta) \to \max_{V(.,.),\theta}$$

but **we do not know Z at the training stage**

- By establishing prior distribution over Z (e.g. uniform among different word meanings) we build latent variable model

$$p(y, z|x, \theta) = p(y|x, z, \theta)p(z|x)$$

- Now we can use EM-algorithm to train it $p(Y|X, \theta) \to \max_{V(.,.),\theta}$

Naïve EM algorithm

- E-step: For each training object estimate the distribution on latent variable

$$p(z_i = k|x_i, y_i, \theta) = \frac{p(y_i|x_i, k, \theta)p(z_i = k|x_i)}{\sum_{l=1}^{K(x_i)} p(y_i|x_i, l, \theta)p(z_i = l|x_i)}$$

We can do this in explicit manner assuming the number of meanings $K(x_i)$ is reasonably small for all x_i

```
Our train arrived to Waterloo at 2pm

                 ⎧ Station  0.76
Waterloo - ?     ⎨ Battle   0.21
                 ⎩ Song     0.03
```

Naïve EM algorithm

- E-step: For each training object estimate the distribution on latent variable

$$p(z_i = k|x_i, y_i, \theta) = \frac{p(y_i|x_i, k, \theta)p(z_i = k|x_i)}{\sum_{l=1}^{K(x_i)} p(y_i|x_i, l, \theta)p(z_i = l|x_i)}$$

  We can do this in explicit manner assuming the number of meanings $K(x_i)$ is reasonably small for all x_i

- M-step: Optimize w.r.t. θ and $V(.,.)$

$$\mathbb{E}_Z \log p(Y|Z, X, \theta)p(Z|X) \to \max_{\theta, V(.,.)}$$

  Equivalent to training standard skip-gram with increased number of context words

- Seems computationally efficient?..

Naïve EM algorithm

- E-step: For each training object estimate the distribution on latent variable

$$p(z_i = k|x_i, y_i, \theta) = \frac{p(y_i|x_i, k, \theta)p(z_i = k|x_i)}{\sum_{l=1}^{K(x_i)} p(y_i|x_i, l, \theta)p(z_i = l|x_i)}$$

  We can do this in explicit manner assuming the number of meanings $K(x_i)$ is reasonably small for all x_i

- M-step: Optimize w.r.t. θ and $V(.,.)$

$$\mathbb{E}_Z \log p(Y|Z, X, \theta)p(Z|X) \to \max_{\theta, V(.,.)}$$

  Equivalent to training standard skip-gram with increased number of context words

- We'll need to recompute $p(z|x, y, \theta)$ for **each** object (In Wikipedia2012 there is about 10^9 of words) to make just **single** iteration of EM

What was not covered in this talk

- We need scalable version of EM-algorithm
- Each word occurence is present $2C$ times in training set and of course the corresponding x_i should have the same meaning
- To automatically define the proper number of different meanings we need to set a special prior on $p(z_i|x_i)$ using so-called **Chinese restaurant process**
- Come to Sergey's talk :)

Experiments: Multiple meanings

Closest words to "platform"		
fwd	stabling	software
sedan	turnback	ios
fastback	pebblemix	freeware
chrysler	citybound	netfront
hatchback	metcard	linux
notchback	underpass	microsoft
rivieraoldsmobile	sidings	browser
liftback	tram	desktop
superoldsmobile	cityrail	interface
sheetmetal	trams	newlib

Closest words to "sound"	
puget	sequencer
sounds	multitrack
island	synths
shoals	audiophile
inlet	stereo
bay	sampler
hydrophone	sequencers
quoddy	headphones
shore	reverb
buoyage	multitracks

Computer is now able to assign different semantic representations to different occurrences of same word depending on the context

Experiments: word disambiguation

- We run AdaGram with $\alpha = 0.2$
- 5 meanings for 'Waterloo' were found
- Let us try to make disambiguation

Experiments: word disambiguation

- We run AdaGram with $\alpha = 0.2$
- 5 meanings for 'Waterloo' were found
- Let us try to make disambiguation

```
Who won the Battle of Waterloo?
```

Experiments: word disambiguation

- We run AdaGram with $\alpha = 0.2$
- 5 meanings for 'Waterloo' were found
- Let us try to make disambiguation

`Who won the Battle of Waterloo?`

Probabilities of meanings
0.0000098
0.997716
0.0000309
0.00207717
0.00016605

Experiments: word disambiguation

- We run AdaGram with $\alpha = 0.2$
- 5 meanings for 'Waterloo' were found
- Let us try to make disambiguation

`Who won the Battle of Waterloo?`

Probabilities of meanings
0.0000098
0.997716
0.0000309
0.00207717
0.00016605

Closest words:
"sheriffmuir"
"agincourt"
"austerlitz"
"jena-auerstedt"
"malplaquet"
"königgrätz"
"mollwitz"
"albuera"
"toba-fushimi"
"hastenbeck"

Experiments: word disambiguation

- We run AdaGram with $\alpha = 0.2$
- 5 meanings for 'Waterloo' were found
- Let us try to make disambiguation

```
Our train has departed from Waterloo at 1100pm
```

Experiments: word disambiguation

- We run AdaGram with $\alpha = 0.2$
- 5 meanings for 'Waterloo' were found
- Let us try to make disambiguation

`Our train has departed from Waterloo at 1100pm`

Probabilities of meanings
0.948032
0.00427984
0.000470485
0.0422029
0.0050148

Experiments: word disambiguation

- We run AdaGram with $\alpha = 0.2$
- 5 meanings for 'Waterloo' were found
- Let us try to make disambiguation

`Our train has departed from Waterloo at 1100pm`

Probabilities of meanings
0.948032
0.00427984
0.000470485
0.0422029
0.0050148

Closest words:
"paddington"
"euston"
"victoria"
"liverpool"
"moorgate"
"via"
"london"
"street"
"central"
"bridge"

Resume: LVM

- Can fill in missing data
- Can reveal the structure in data (manifolds, clusters)
- Can find hidden information in training data
- Can be used for constructing more flexible models of data with better predictive abilities
- Scalability issues??