Bridging the gap between Bayesian models and deep learning

Dmitry P. Vetrov
Head of Bayesian methods research group
http://bayesgroup.ru
Faculty of Computer Sciences
Higher School of Economics

Outline

- Bayesian methods in statistics
- Bayesian framework in machine learning
- Latent variable models
- Case study: Adaptive Skip-gram (AdaGram)
- Deep learning meets Bayesian models
- Some projects from our group

Bayesian methods research group

Founded in 2007. Currently consists of 8 students, 5 PhD students, 1 researcher and 1 associate professor.

p(B|A)yesgroup.ru

Bayesian framework

Conditional and marginal distributions

Just to remind...

- Conditional distribution

$$\texttt{Conditional} = \frac{\texttt{Joint}}{\texttt{Marginal}}, \quad p(x|y) = \frac{p(x,y)}{p(y)}$$

- Product rule: Any joint distribution can be expressed as a product of one-dimensional conditional distributions

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

- Sum rule: Any marginal distribution can be obtained from the joint distribution by **intergrating out** unnessesary variables

$$p(y) = \int p(x,y)dx = \int p(y|x)p(x)dx = \mathbb{E}_x p(y|x)$$

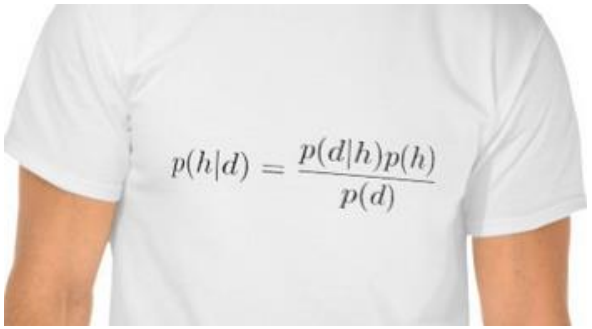
Bayesian Framework

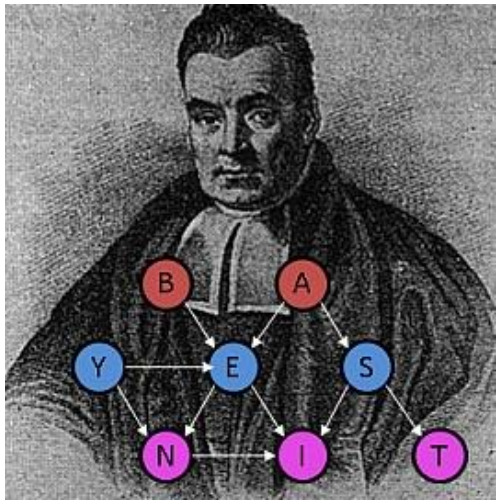
- Treats everything as random variables
- Encodes ignorance in terms of distributions
- Makes use of **Bayes Theorem**

$$\texttt{Posterior} = \frac{\texttt{Likelihood} \times \texttt{Prior}}{\texttt{Evidence}}, \quad p(\theta|y) = \frac{p(y|\theta)p(\theta)}{\int p(y|\theta)p(\theta)d\theta}$$

- Possible to compute the estimate for arbitrary **unknown** variable (U) given **observed** data (O) and not having any knowledge about **latent** variables (L) from the joint distribution $p(U, O, L)$:

$$p(U|O) = \frac{\int p(U, O, L)dL}{\int p(U, O, L)dLdU}$$





Frequentist vs. Bayesian frameworks

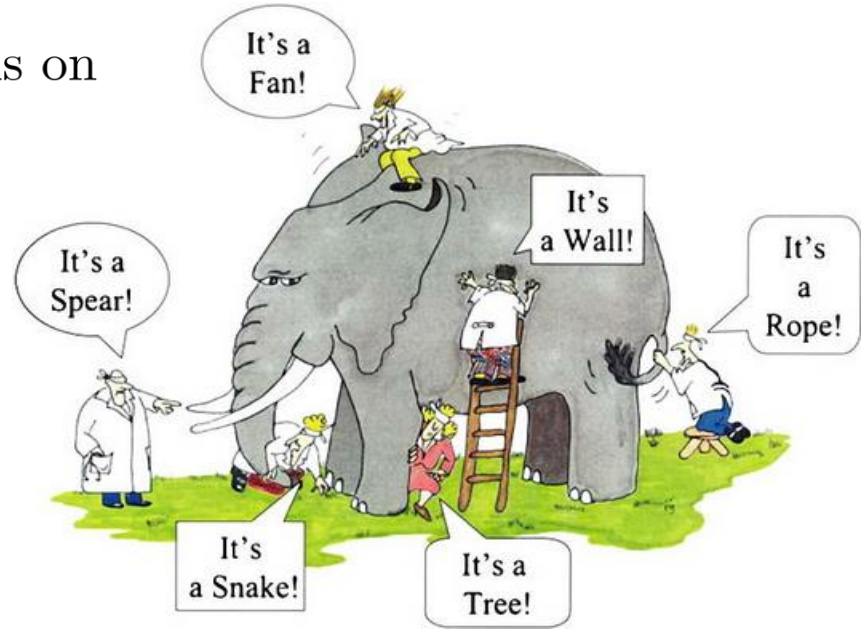
	Frequentist	Bayesian
Randomness	Objective indefiniteness	Subjective ignorance
Variables	Random and Deterministic	Everything is random
Inference	Maximal likelihood	Bayes theorem
Estimates	ML-estimates	Posterior or MAP-estimates
Applicability	$n \gg 1$	$\forall n$

Bayesian framework

- Encodes ignorance in terms of distributions
- Makes use of **Bayes Theorem**

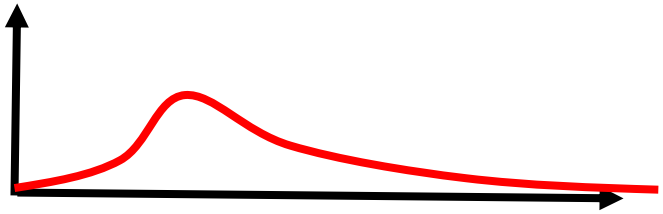
$$\texttt{Posterior} = \frac{\texttt{Likelihood} \times \texttt{Prior}}{\texttt{Evidence}}, \quad p(\theta|X) = \frac{p(X|\theta)p(\theta)}{\int p(X|\theta)p(\theta)d\theta}$$

- Posteriors may serve as new priors, i.e. may combine multiple models!
- **BigData:** we can process data streams on an update-and-forget basis
- Support distributed processing



Bayesian inference

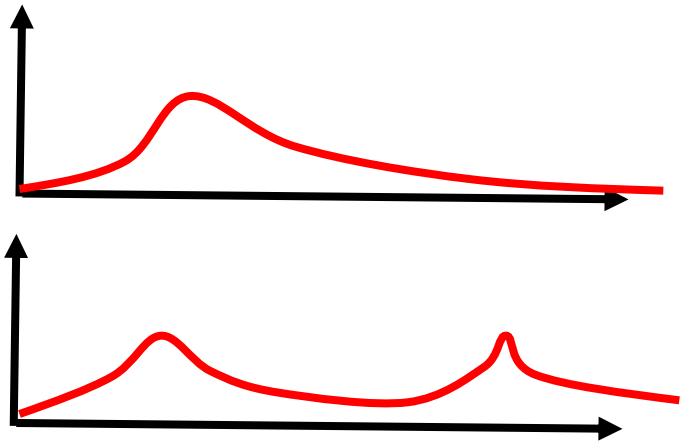
- Consider blind wisdomers who try to estimate the mass of an elephant using their tactile measurements.
- They start with common knowledge about animals typical masses $p(\theta)$



Bayesian inference

- Consider blind wisdomers who try to estimate the mass of an elephant using their tactile measurements.
- They start with common knowledge about animals typical masses $p(\theta)$
- The first wisdomer touches a tail

$$p(\theta|x_1) = \frac{p_1(x_1|\theta)p(\theta)}{\int p_1(x_1|\theta)p(\theta)d\theta}$$



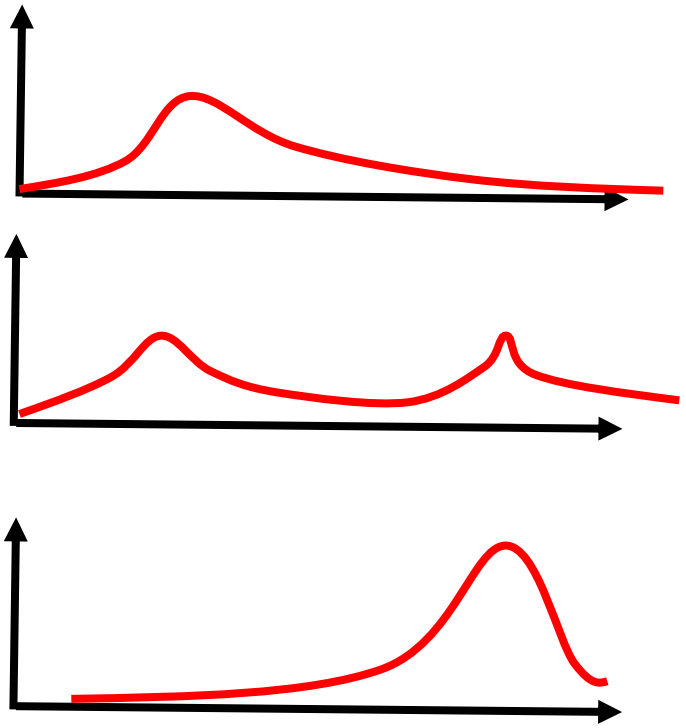
Bayesian inference

- Consider blind wisdomers who try to estimate the mass of an elephant using their tactile measurements.
- They start with common knowledge about animals typical masses $p(\theta)$
- The first wisdomer touches a tail

$$p(\theta|x_1) = \frac{p_1(x_1|\theta)p(\theta)}{\int p_1(x_1|\theta)p(\theta)d\theta}$$

- The second wisdomer touches a leg and uses $p(\theta|x_1)$ as **his new prior**

$$p(\theta|x_1, x_2) = \frac{p_2(x_2|\theta)p(\theta|x_1)}{\int p_2(x_2|\theta)p(\theta|x_1)d\theta}$$



Bayesian inference

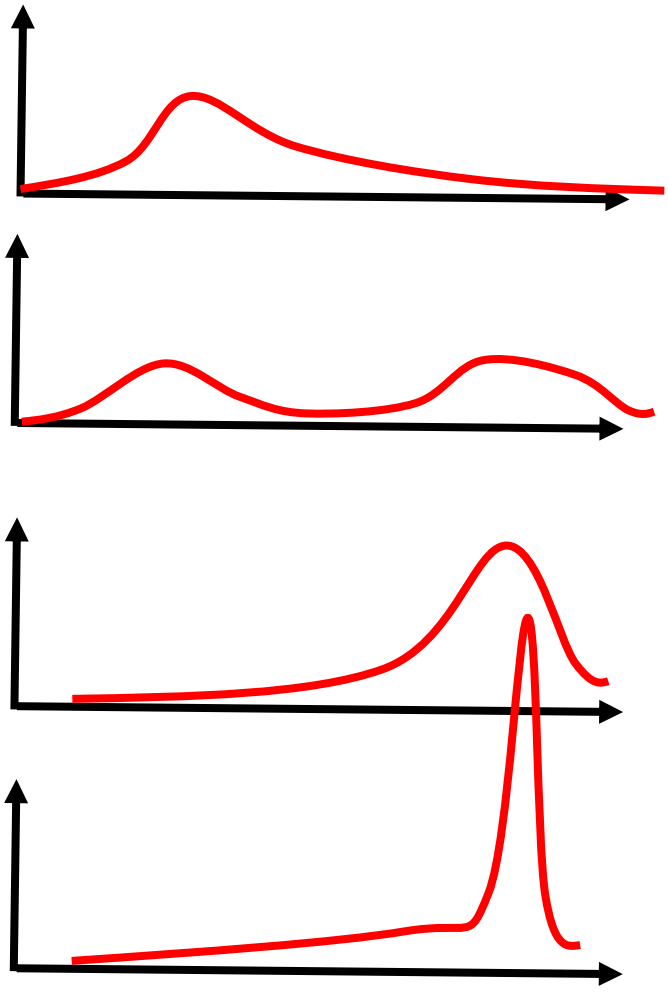
- Consider blind wisdomers who try to estimate the mass of an elephant using their tactile measurements.
- They start with common knowledge about animals typical masses $p(\theta)$
- The first wisdomer touches a tail

$$p(\theta|x_1) = \frac{p_1(x_1|\theta)p(\theta)}{\int p_1(x_1|\theta)p(\theta)d\theta}$$

- The second wisdomer touches a leg and uses $p(\theta|x_1)$ as **his new prior**

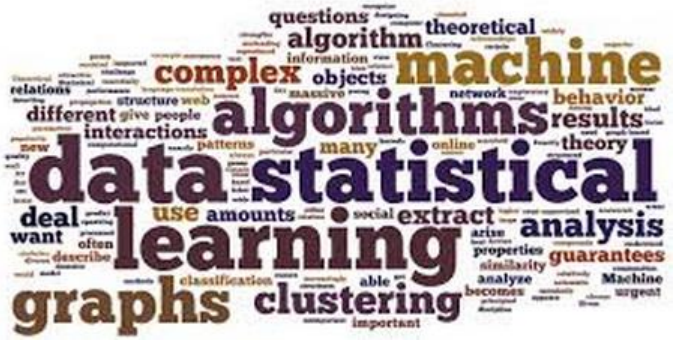
$$p(\theta|x_1, x_2) = \frac{p_2(x_2|\theta)p(\theta|x_1)}{\int p_2(x_2|\theta)p(\theta|x_1)d\theta}$$

- ...
- At the end they form sharp distribution $p(\theta|x_1, \ldots, x_m)$



What is machine learning?

- ML tries to find regularities within the data
- Data is a set of objects (users, images, signals, RNAs, chemical compounds, credit histories, etc.)
- Each object is described by a set of observed variables X and a set of hidden (latent) variables T
- It is assumed that the values of hidden variables are hard to get and we have only limited number of objects with known hidden variables, so-called training set (X_{tr}, T_{tr})
- The goal is to find the way of predicting the hidden variables for a new object given the values of observed variables by adjusting the weights W of decision rule.



Machine learning from Bayesian point of view

Stage	Observed	Hidden	Goal
Learning	(X_{tr}, T_{tr})	W	$p(W\|X_{tr}, T_{tr}) = \frac{p(T_{tr}\|X_{tr},W)p(W)}{\int p(T_{tr}\|X_{tr},W)p(W)dW}$
Testing	X_{test}	T_{test}	$p(T_{test}\|X_{test}, X_{tr}, T_{tr}) = \int p(T_{test}\|X_{test}, W)p(W\|X_{tr}, T_{tr})dW$

We start with a **probabilistic model** $p(T, W|X) = p(T|X, W)p(W)$

- Instead of single algorithm we learn an **ensemble** $p(W|X_{tr}, T_{tr})$
- We use weighted voting at decision-making step
- Conceptually the scheme does not involve any kind of optimization

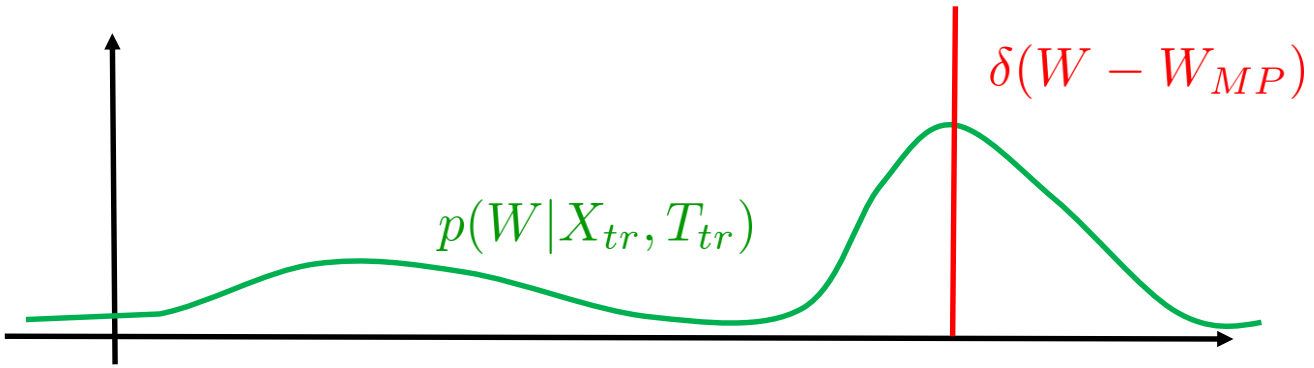
Poor man's Bayes

- Simplified probabilistic modeling
- Approximate posteior $p(W|X_{tr}, T_{tr})$ with a delta function $\delta(W - W_{MP})$
- Corresponds to point estimate of W:

$$W_{MP} = \arg\max p(W|X_{tr}, T_{tr}) = \arg\max p(T_{tr}|X_{tr}|W)p(W)$$

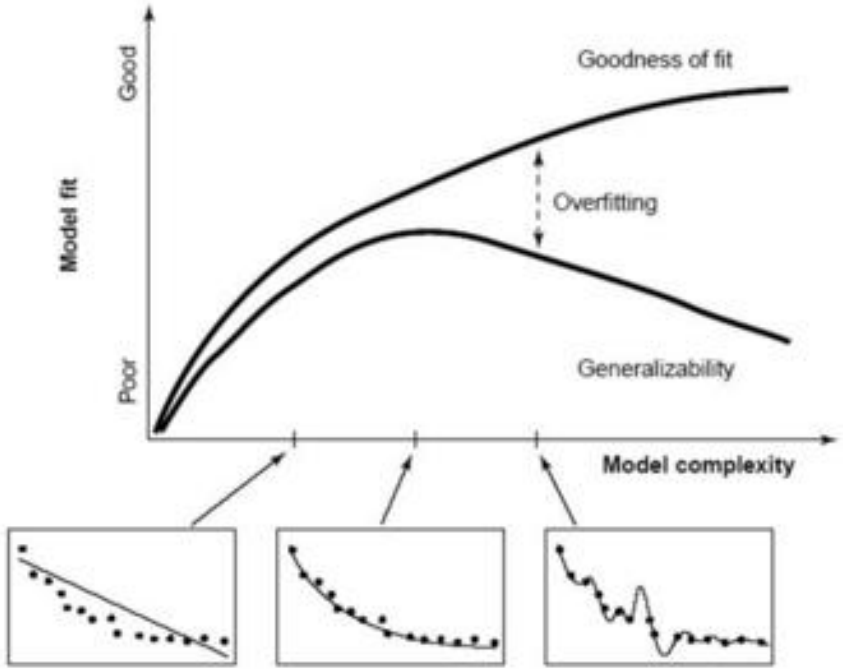
- Inference is more simple

$$p(T|X, X_{tr}, T_{tr}) = \int p(T|X, W)p(W|X_{tr}, T_{tr})dW \approx p(T|X, W_{MP})$$



Model complexity trade-off

- ML is always a trade-off between **underfitting** and **overfitting**
- Underfitting: good generalization, bad fit. We have NOT found all hidden regularities within the data
- Overfitting: bad generalization, good fit. We have found false regularities that are always present in training data because of its limited size



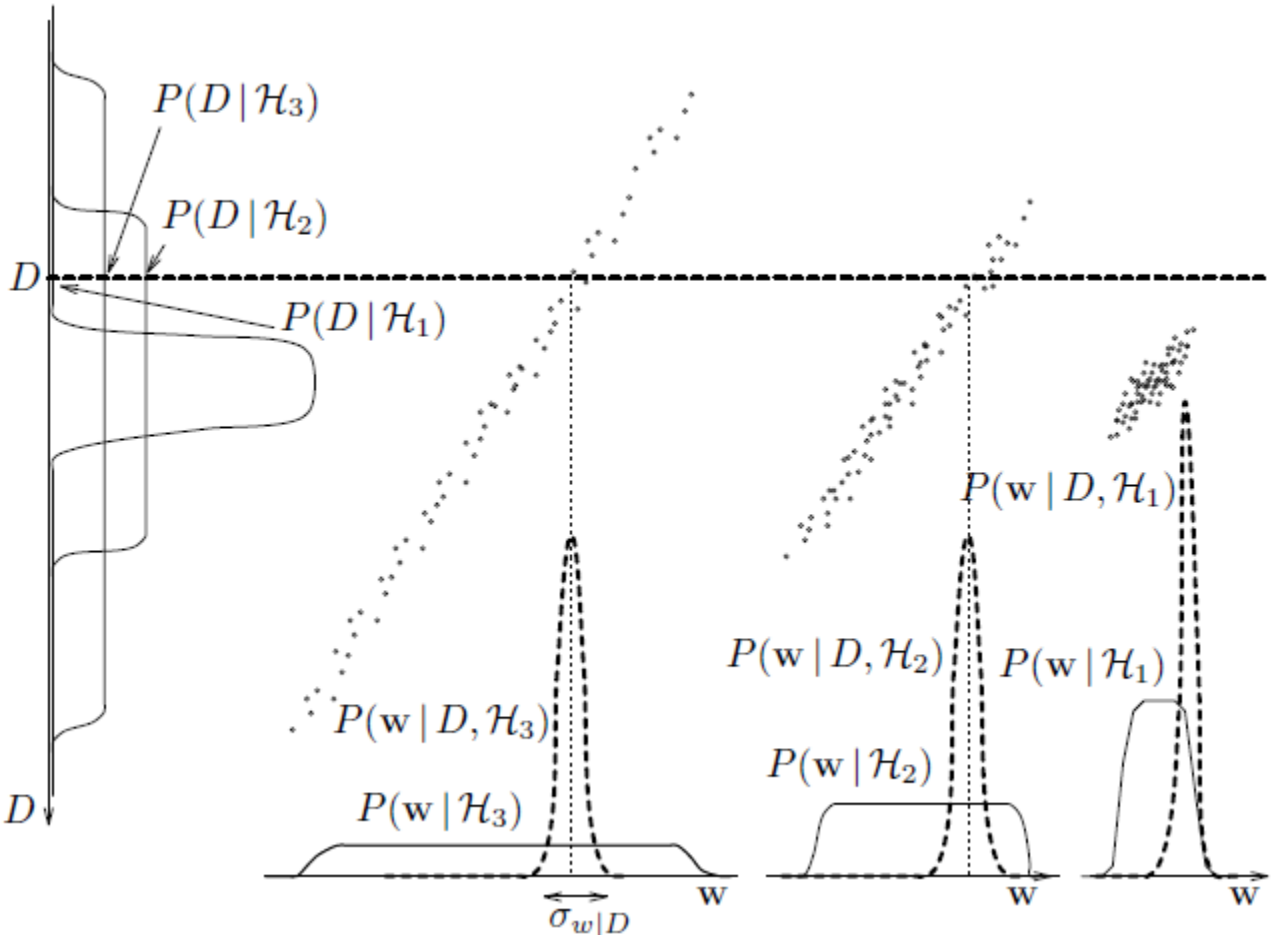
Bayesian model selection

- Suppose we have two concurrent probabilistic models $p_1(T, W|X)$ and $p_2(T, W|X)$. Which one is better for our particular problem (X_{tr}, T_{tr})?
- According to **maximal evidence** principle we should select the model with largest meginal likelihood (evidence)

$$j_* = \arg\max_j p_j(T_{tr}|X_{tr})$$

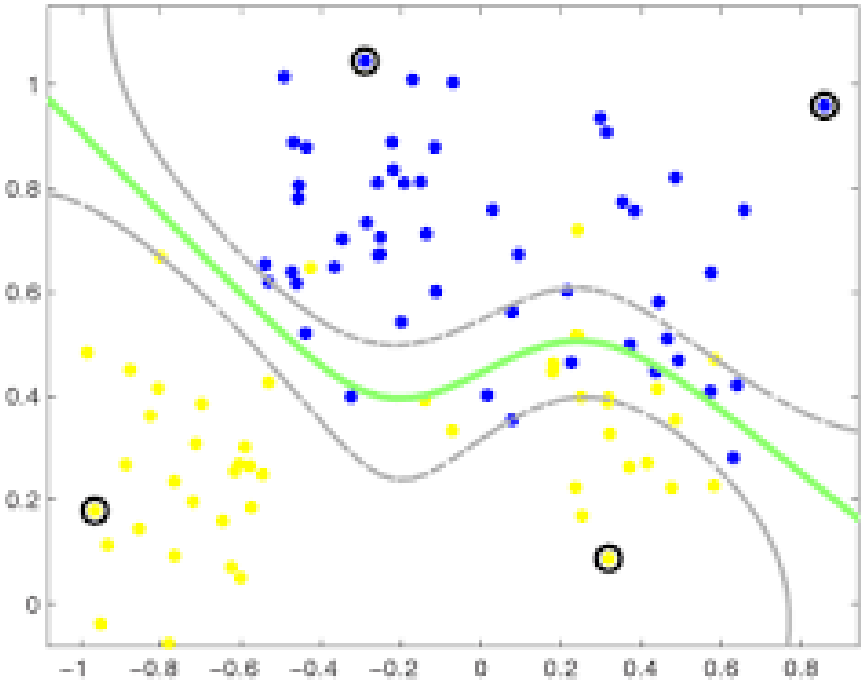
- This is mathematical formalization of **Occam's razor**

Geometric illustration



Advantages of Bayesian model selection

- Allows to select among continuum number of models
- No need for computationally expensive cross-validation
- Automatic relevance determination effect
  - selects relevant features and/or objects
  - sets proper number of clusters
  - finds best degree in polynomial regression
  - potentially may select neural network's architecture



Latent variable modeling

Exponential class of distributions

- Distribution $p(x|\theta)$ belongs to exponential class if it can be expressed as follows

$$p(x|\theta) = \frac{f(x)}{g(\theta)} \exp\left(\theta^T u(x)\right),$$

  where $f(x) \geq 0$, $g(\theta) > 0$

- Function $g(\theta)$ ensures that right-hand expression is a distribution

$$g(\theta) = \int f(x) \exp\left(\theta^T u(x)\right) dx$$

- Functions $u(x)$ are **sufficient statistics** whose values contain all information that can be extracted from sample about distribution
- Function $f(x)$ can be **arbitrary** non-negative function

Log-concavity of exponential class

- Consider derivative of $\log g(\theta)$

$$\frac{\partial \log g(\theta)}{\partial \theta_j} =$$

Log-concavity of exponential class

- Consider derivative of $\log g(\theta)$

$$\frac{\partial \log g(\theta)}{\partial \theta_j} = \frac{1}{g(\theta)} \frac{\partial g(\theta)}{\partial \theta_j} =$$

Log-concavity of exponential class

- Consider derivative of $\log g(\theta)$

$$\frac{\partial \log g(\theta)}{\partial \theta_j} = \frac{1}{g(\theta)} \frac{\partial g(\theta)}{\partial \theta_j} = \frac{1}{g(\theta)} \frac{\partial}{\partial \theta_j} \int f(x) \exp(\theta^T u(x)) dx =$$

Log-concavity of exponential class

- Consider derivative of $\log g(\theta)$

$$\frac{\partial \log g(\theta)}{\partial \theta_j} = \frac{1}{g(\theta)} \frac{\partial g(\theta)}{\partial \theta_j} = \frac{1}{g(\theta)} \frac{\partial}{\partial \theta_j} \int f(x) \exp(\theta^T u(x)) dx =$$

$$\frac{1}{g(\theta)} \int f(x) \exp(\theta^T u(x)) u_j(x) dx$$

Log-concavity of exponential class

- Consider derivative of $\log g(\theta)$

$$\frac{\partial \log g(\theta)}{\partial \theta_j} = \frac{1}{g(\theta)} \frac{\partial g(\theta)}{\partial \theta_j} = \frac{1}{g(\theta)} \frac{\partial}{\partial \theta_j} \int f(x) \exp(\theta^T u(x)) dx =$$

$$\frac{1}{g(\theta)} \int f(x) \exp(\theta^T u(x)) u_j(x) dx = \int p(x|\theta) u_j(x) dx = \mathbb{E}_x u_j(x)$$

Log-concavity of exponential class

- Consider derivative of $\log g(\theta)$

$$\frac{\partial \log g(\theta)}{\partial \theta_j} = \frac{1}{g(\theta)} \frac{\partial g(\theta)}{\partial \theta_j} = \frac{1}{g(\theta)} \frac{\partial}{\partial \theta_j} \int f(x) \exp(\theta^T u(x)) dx =$$

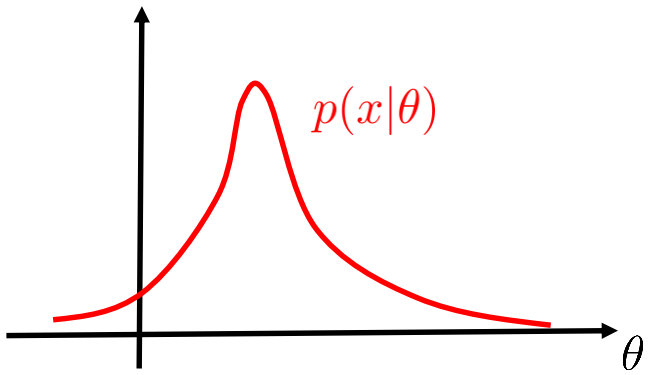
$$\frac{1}{g(\theta)} \int f(x) \exp(\theta^T u(x)) u_j(x) dx = \int p(x|\theta) u_j(x) dx = \mathbb{E}_x u_j(x)$$

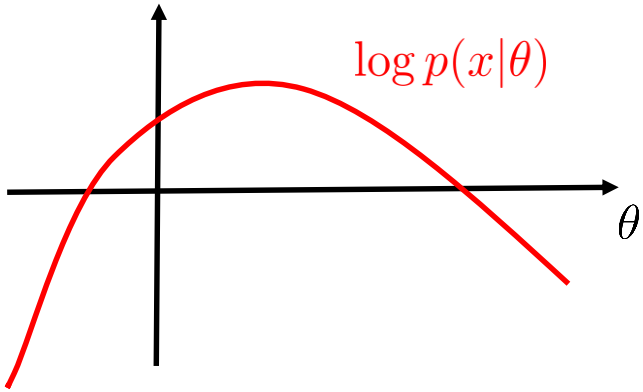
- Analogously $\frac{\partial^2 \log g(\theta)}{\partial \theta_i \partial \theta_j} = \mathrm{Cov}(u_i(x), u_j(x))$
- Thus $\log g(\theta)$ is convex function, consequently

$$\log p(x|\theta) = \theta^T u(x) - \log g(\theta) + \log f(x)$$

is concave function of θ!

Log-concavity of exponential class





- For log-concave distributions maximum likelihood estimation can be done in an efficient manner
- All discrete distributions and many continuous (Gaussian, Laplace, Gamma, Dirichlet, Wishart, Beta, Chi-squared, etc.) belong to exponential class

Example: Gaussian distribution

- Standard form of 1-dimensional Gaussian

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

- Natural form

$$p(x|\theta) = \frac{1}{\sqrt{-\frac{\pi}{\theta_1}} \exp\left(-\frac{\theta_2^2}{4\theta_1}\right)} \exp(\theta_1 x^2 + \theta_2 x),$$

  where $\theta_1 = -\frac{1}{2\sigma^2}$ and $\theta_2 = \frac{\mu}{\sigma^2}$

- Hence x and x^2 are sufficient statistics and

$$g(\theta) = \sqrt{-\frac{\pi}{\theta_1}} \exp\left(-\frac{\theta_2^2}{4\theta_1}\right)$$

- Note that there is one-to-one correspondence between (θ_1, θ_2) and (μ, σ)

Latent variable modeling: example

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

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

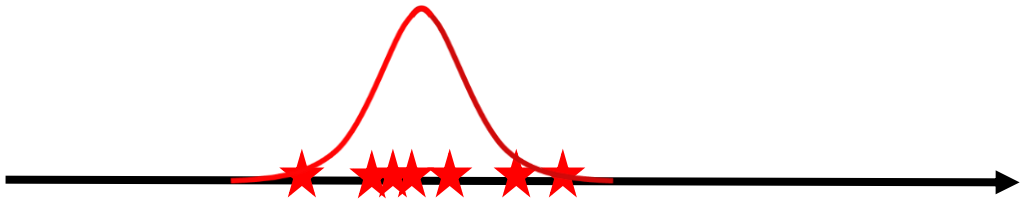
- We need to estimate its parameters μ and σ

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



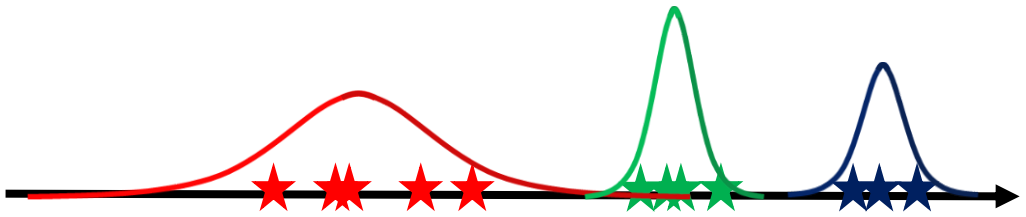
- Solution is simple: we estimate sample mean and variance

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

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



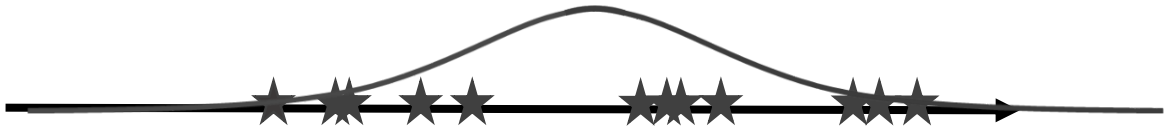
- The problem is as easy if we know what objects were generated from each gaussian

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

$$\log p(X|\theta) = \int q(Z) \log p(X|\theta) dZ$$

Mixture of gaussians

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

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

Mixture of gaussians

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

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

Mixture of gaussians

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

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

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

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

Mixture of gaussians

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

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

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

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

Always non-negative!

Variational lower bound

Mixture of gaussians

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

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

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

Always non-negative!

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

Variational lower bound

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

Mixture of gaussians

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

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

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

Always non-negative!

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

Variational lower bound

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

- 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

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

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

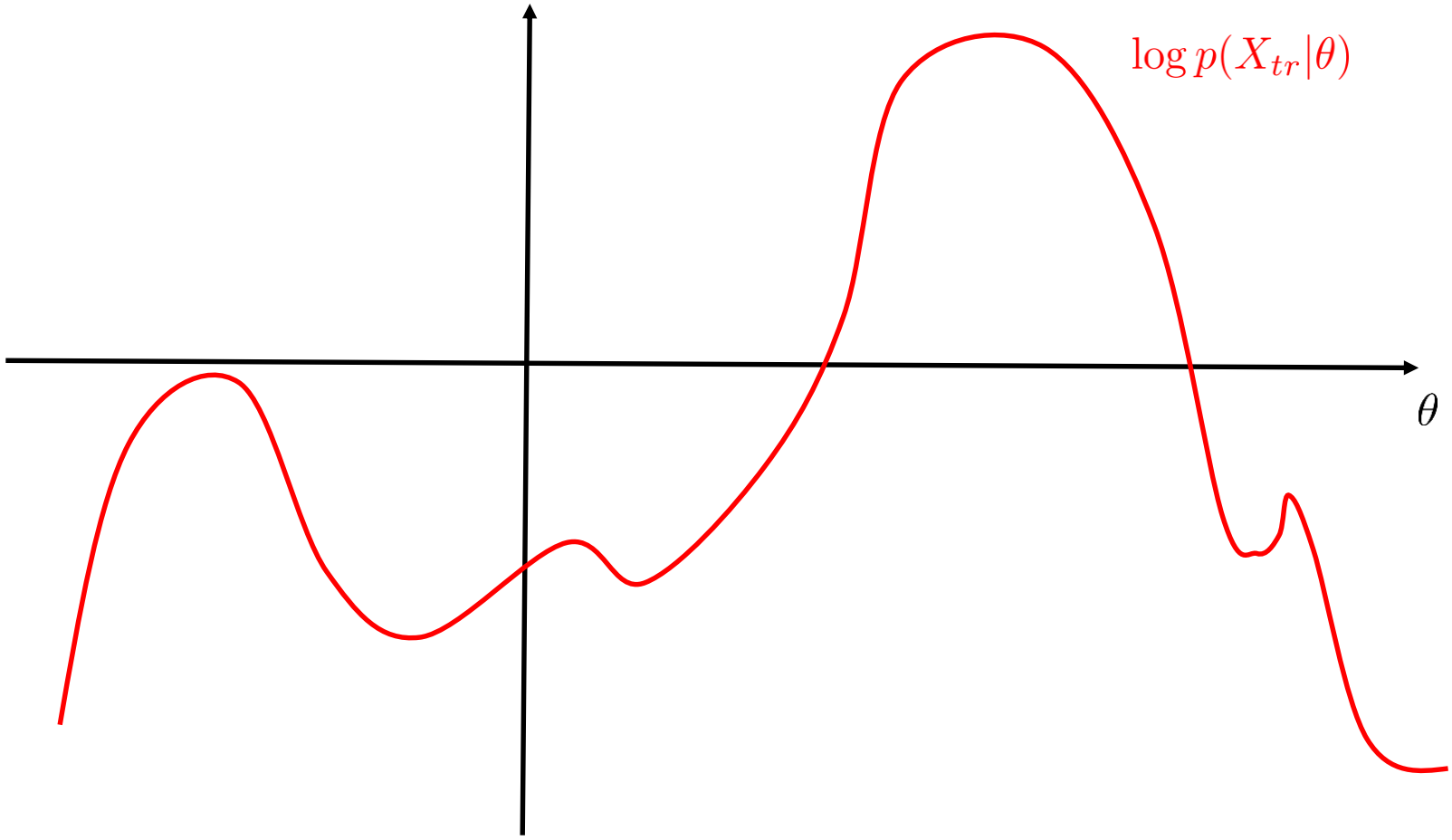
- M-step: solve

$$\theta_* = \arg\max_\theta \mathcal{L}(q, \theta) = \arg\max_\theta \mathbb{E} \log p(X, Z|\theta),$$

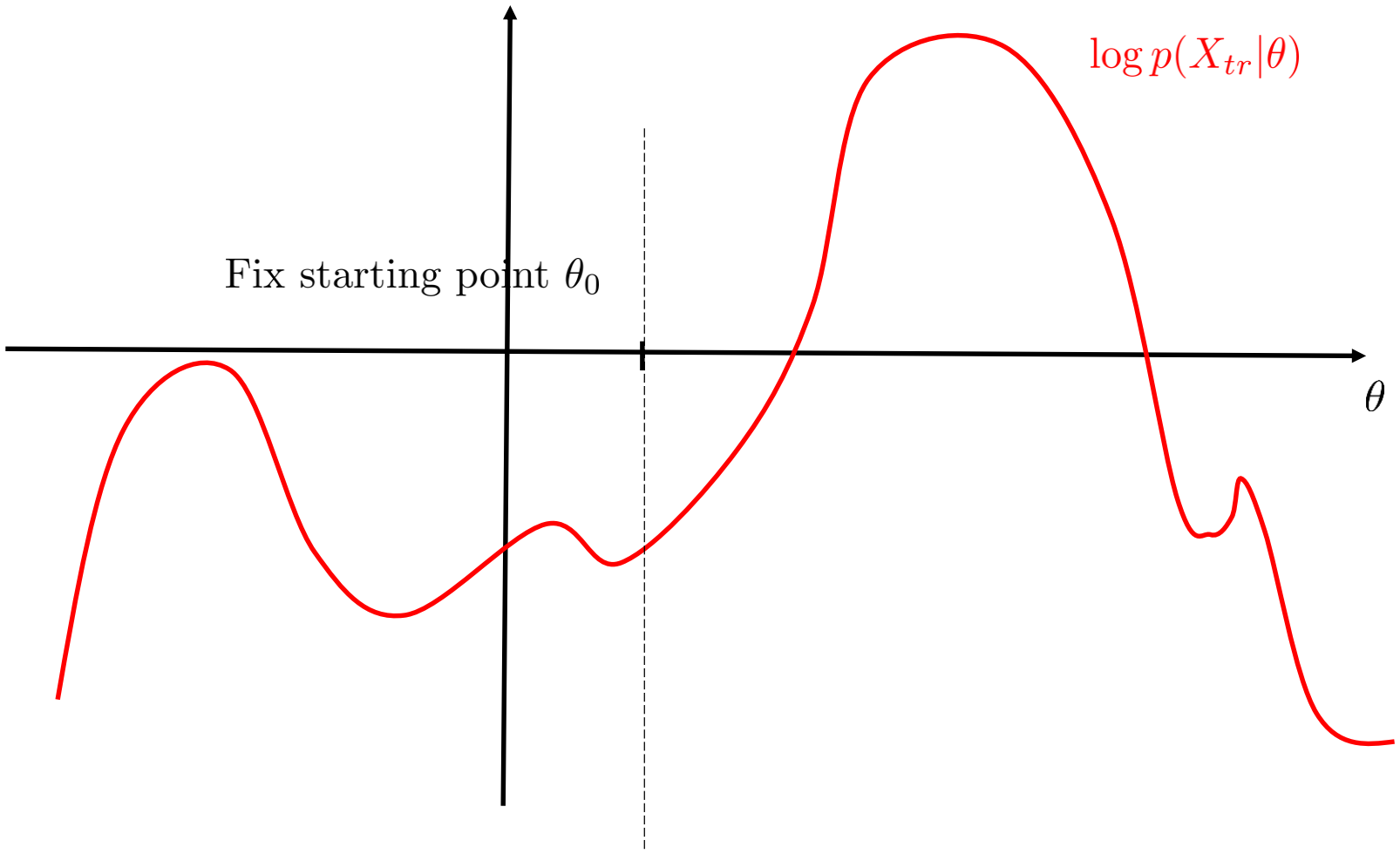
  set $\theta_0 = \theta_*$ and go to E-step until convergence

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

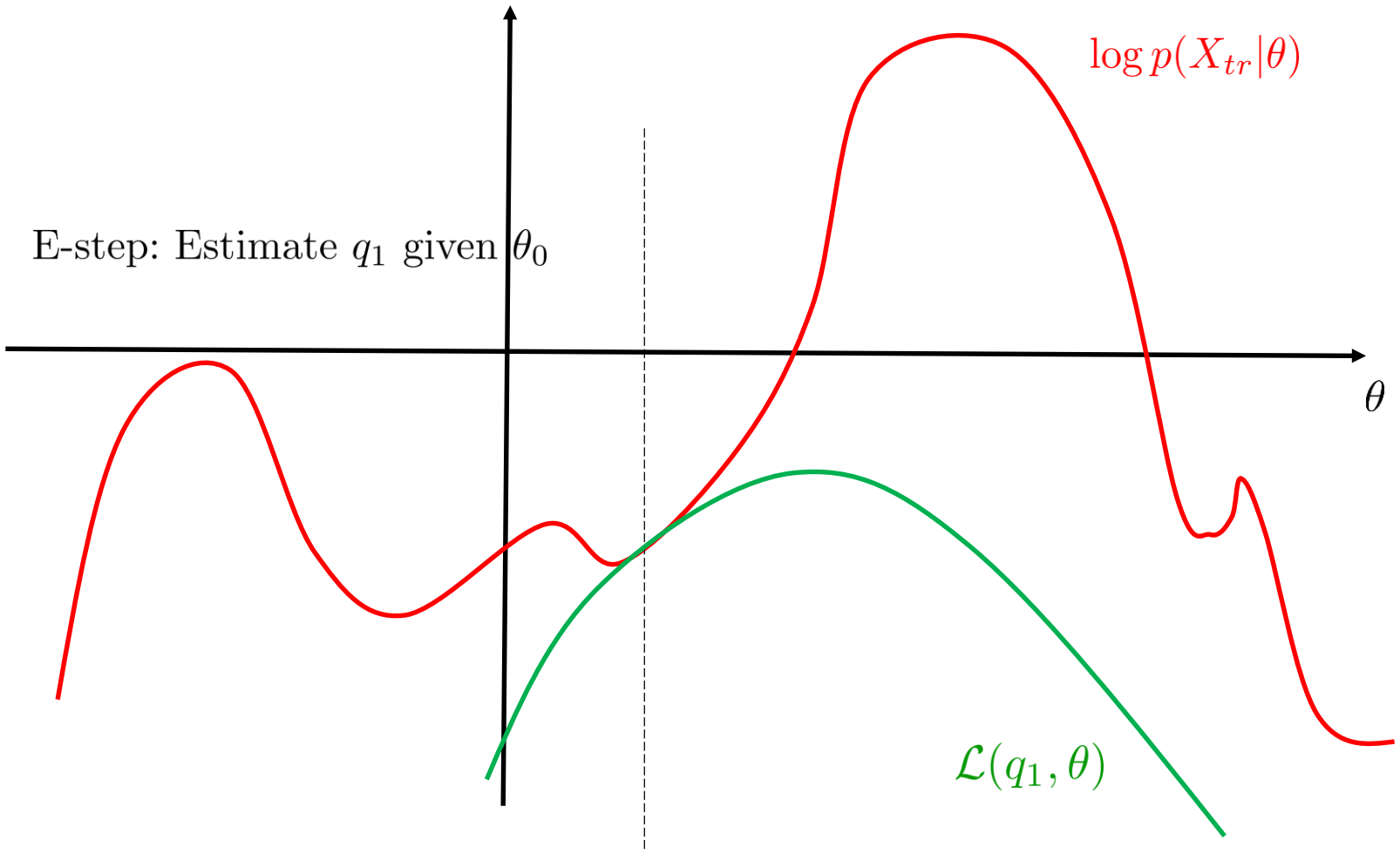
EM-algorithm



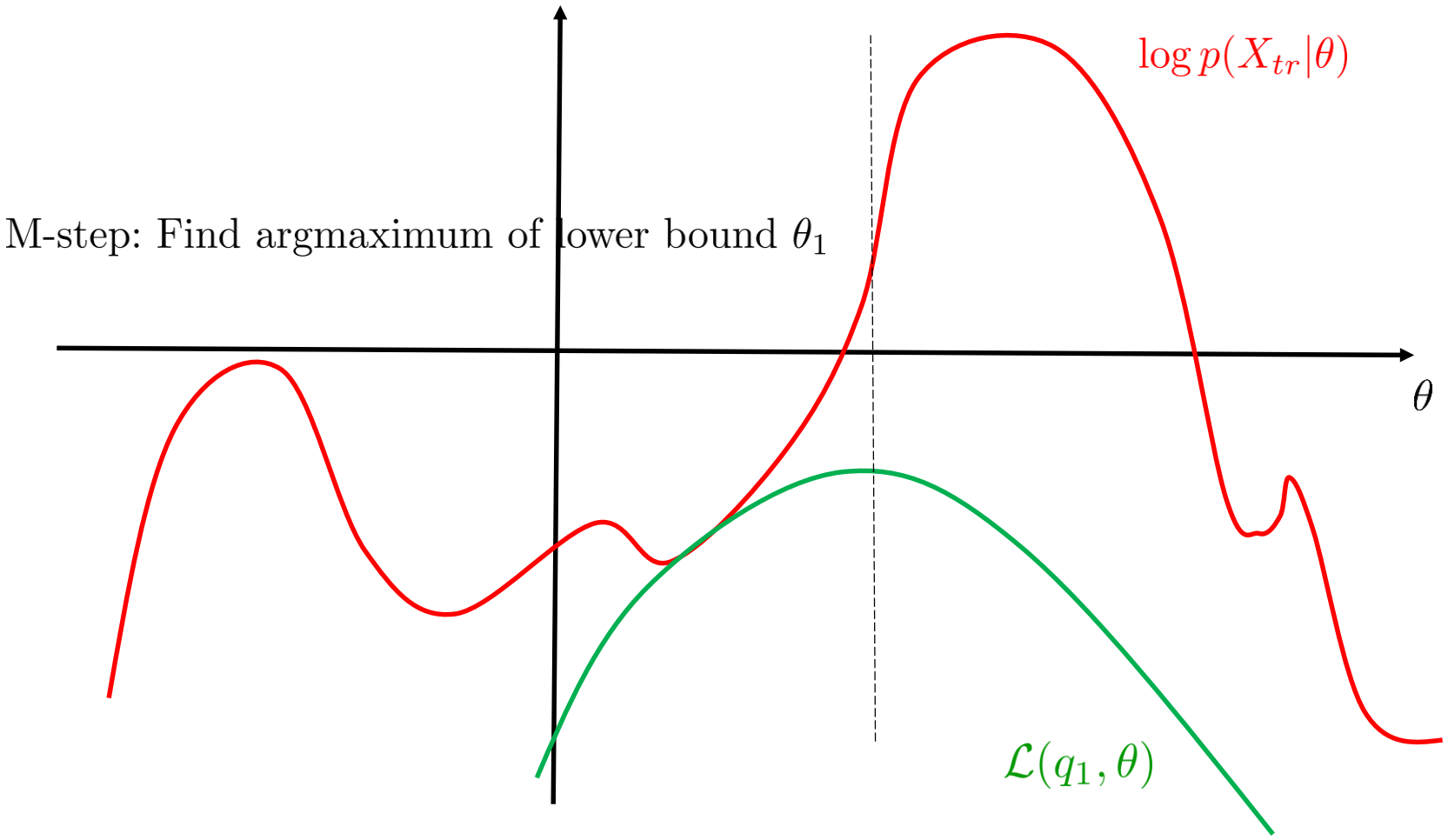
EM-algorithm



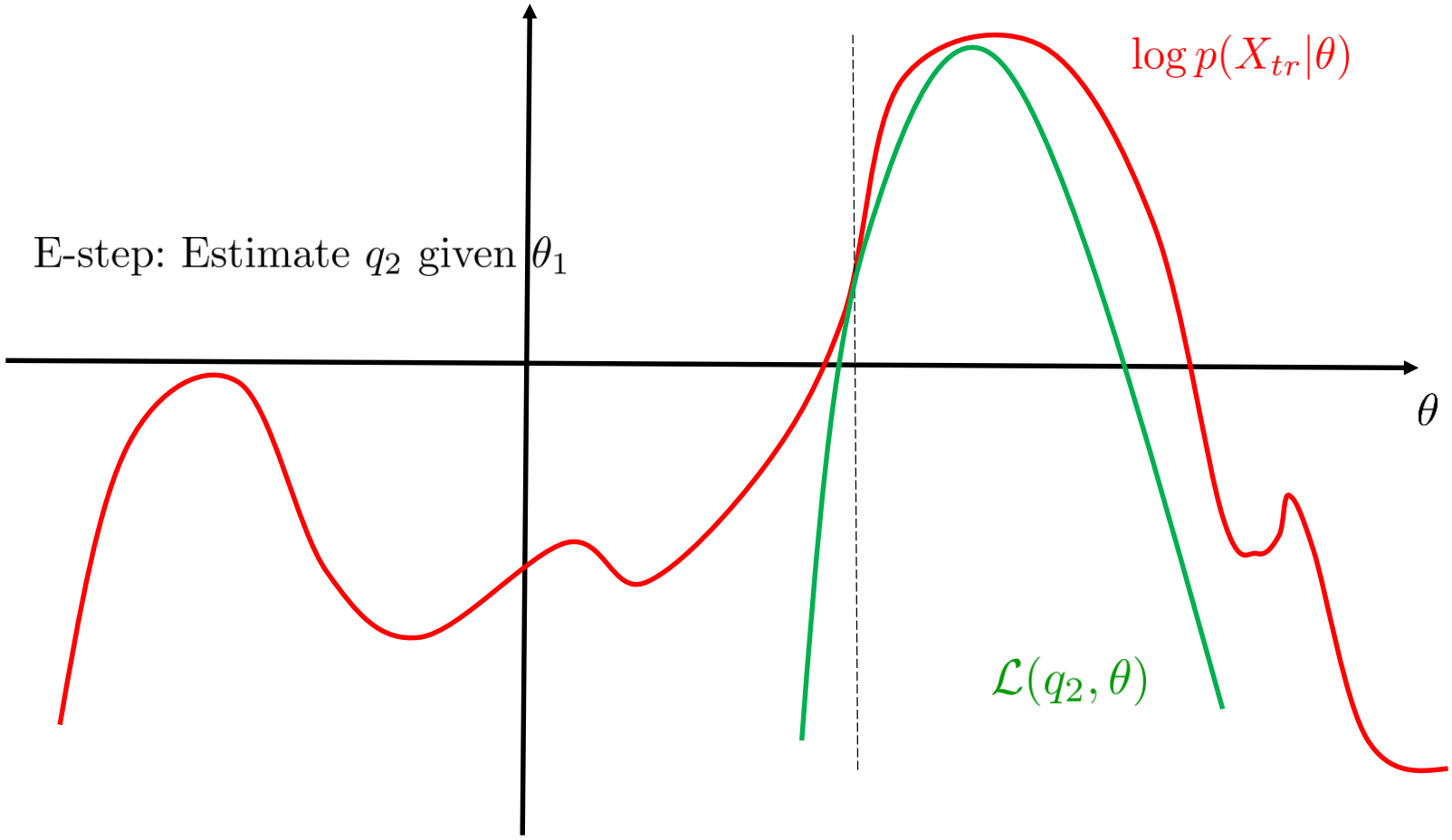
EM-algorithm



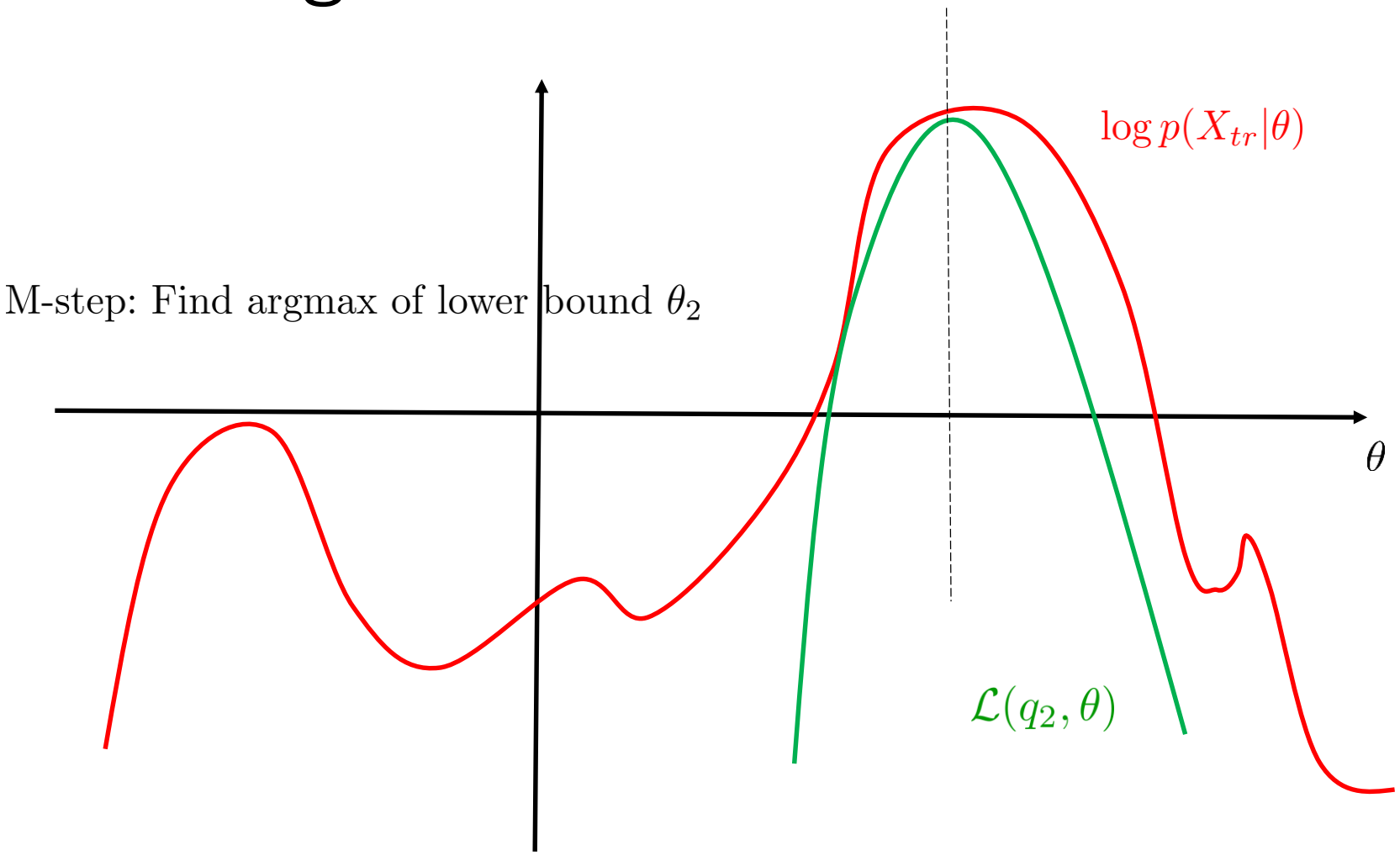
EM-algorithm



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
- We may search for the best $q(Z)$ in paramertic family from the exponential class of distributions. Then E-step is simply multi-dimensional convex optimization problem
- Suitable for ML-estimation for the distributions that do not belong to the exponetial class of distributions, i.e. cannot be represented as

$$p(X|\theta) = \frac{h(X)}{g(\theta)} \exp\left(\theta^T u(X)\right)$$

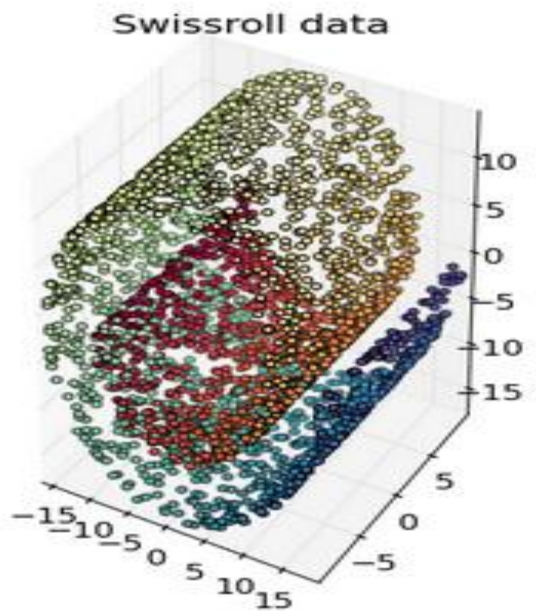
- Basic idea: add latent variables until $p(X, Z|\theta)$ belongs to the exponential class

Continuous latent variables

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

$$p(x|\theta) = \int p(x, z|\theta)dz = \int p(x|z, \theta)p(z|\theta)dz$$

- They are more tricky to perform inference
- Need to check conjugacy property in order to perform E-step explicitly
- Typically used for dimension reduction

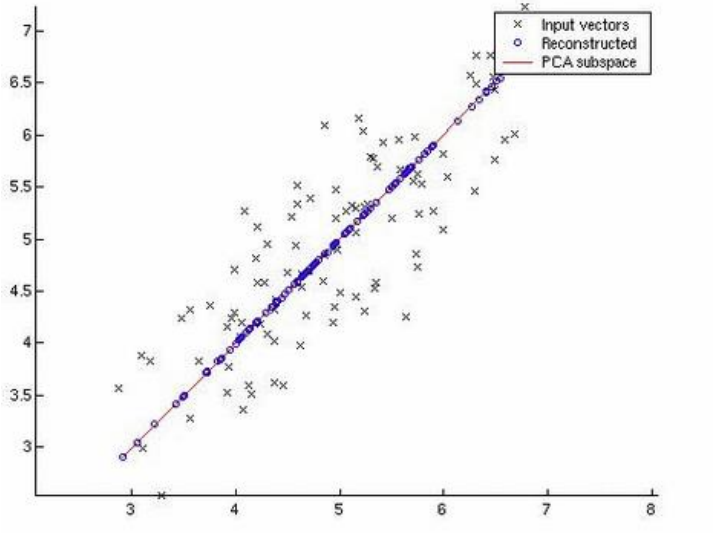


Example: PCA model

- Consider $x \in \mathbb{R}^D$, $z \in \mathbb{R}^d$, such that $D \gg d$
- Joint distribution

$$p(X, Z|\theta) = \prod_{i=1}^{n} p(x_i|z_i, \theta)p(z_i|\theta) = \prod_{i=1}^{n} \mathcal{N}(x_i|Vz_i, \sigma^2 I)\mathcal{N}(z_i|0, I)$$

- θ consists of $D \times d$ matrix V and scalar σ
- Can use EM-algorithm to find $\arg\max_\theta p(X_{tr}|\theta)$



Advantages of EM PCA

In PCA the explicit equation for θ can be obtained analytically. Then why use EM?..

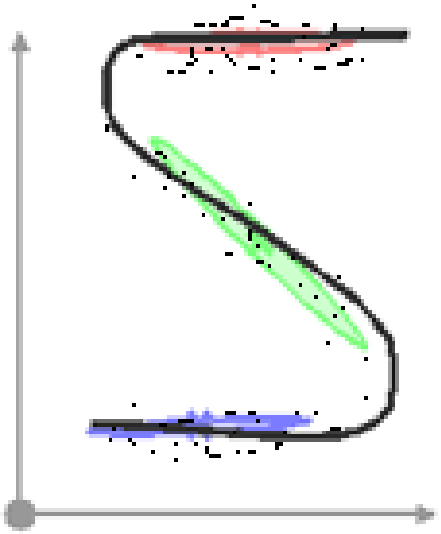
- EM updates have complexity $O(nDd)$ instead of $O(nD^2)$ in analytic solution
- Can process missing parts in X and present parts in Z
- Can determine d if $p(\theta)$ is established
- Can be extended to more general models such as mixture of PCA

Mixture of PCA

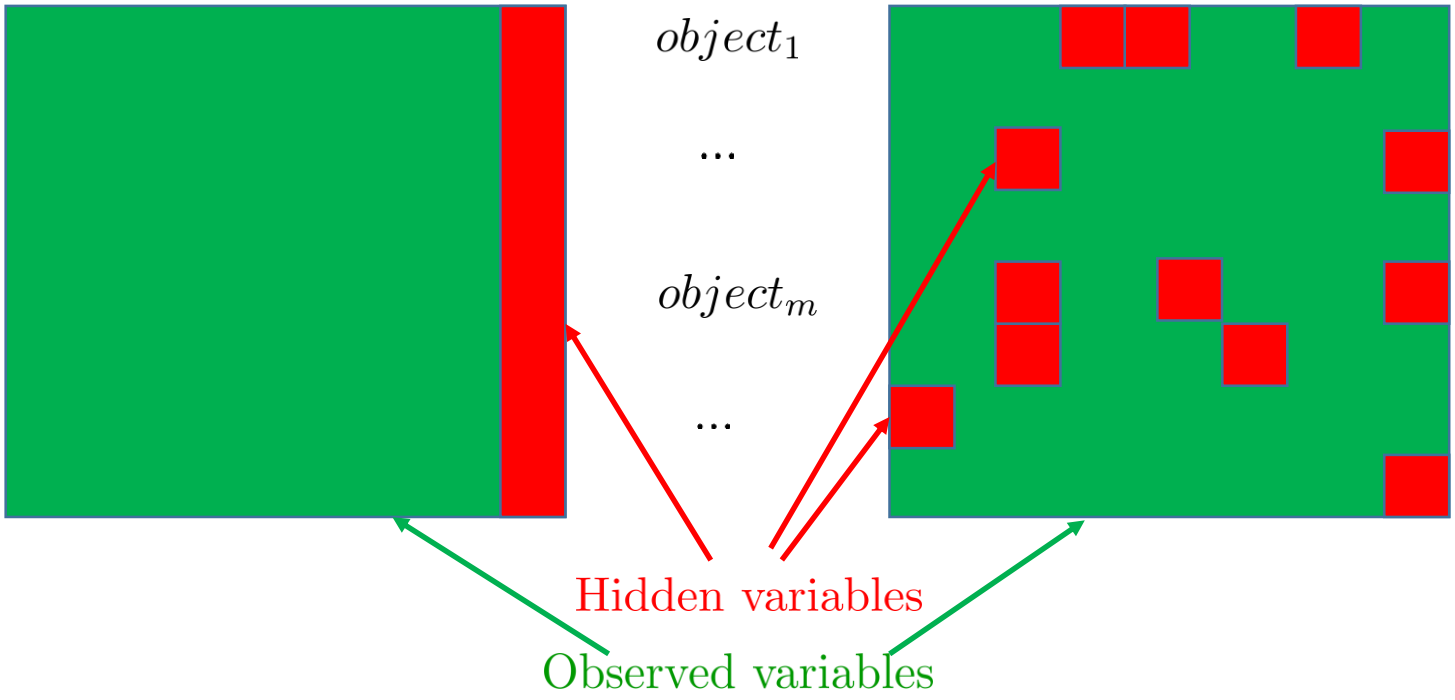
- Two types of latent variables: discrete $t \in \{1, \ldots, K\}$ and continuous $z \in \mathbb{R}^d$
- Joint distribution

$$p(X, Z, T|\theta) = \prod_{i=1}^{n} p(x_i|t_i, z_i, \theta)p(t_i|\theta)p(z_i|\theta) = \prod_{i=1}^{n} \mathcal{N}(x_i|V_{t_i} z_i, \sigma_{t_i}^2 I)\mathcal{N}(z_i|0, I)\pi_{t_i}$$

- θ consists of matrices $\{V_k\}$, scalars $\{\sigma_k\}$, and vector of probabilities θ such that $p(t_i = k) = \pi_k$
- Can be used for non-linear dimension reduction



General nature of EM-framework



- EM algorithm allows processing arbitrary missing data
- May deal with both discrete and continuous variables
- Always converges
- Allows multiple extensions

Extending E-step

- E-step requires conjugate distributions to be performed analytically
- Otherwise normalization constant cannot be computed

$$p(Z|X_{tr},\theta) = \frac{p(X_{tr}|Z,\theta)p(Z|\theta)}{\int p(X_{tr}|Z,\theta)p(Z|\theta)dZ}$$

The most difficult part

- The integration is tractable in case of so-called **conjugate** distributions
- Distribution $p(y)$ and $p(x|y)$ are conjugate iff $p(y|x)$ belongs to the same parametric family as $p(y)$, i.e.

$$p(y) \in \mathcal{A}(\alpha), \quad p(x|y) \in \mathcal{B}(y), \quad \text{then} \quad p(y|x) \in \mathcal{A}(\alpha')$$

Examples of conjugate distributions

Likelihood $p(x\|y)$	y	Conjugate prior $p(y)$
Gaussian	μ	Gaussian
Gaussian	σ^{-2}	Gamma
Gaussian	(μ, σ^{-2})	Gaussian-Gamma
Multivariate Gaussian	Σ^{-1}	Wishart
Bernoulli	p	Beta
Multinomial	$(p_1, \ldots, p_m)$	Dirichlet
Poisson	λ	Gamma
Uniform	θ	Pareto

If the likelihood and the prior are not conjugate the posterior $p(Z|X_{tr}, \theta)$ cannot be computed analytically

Variational Bayes

- Recall that

$$p(Z|X_{tr},\theta) = \arg\max_q \mathcal{L}(q,\theta) = \arg\min_q KL(q(Z)||p(Z|X_{tr},\theta)),$$

  where extremum is taken with respect to **all possible distributions** $q(Z)$

- What if we limit ourselves with more restricted set of distributions?..

Variational Bayes

- Recall that

$$p(Z|X_{tr},\theta) = \arg\max_q \mathcal{L}(q,\theta) = \arg\min_q KL(q(Z)||p(Z|X_{tr},\theta)),$$

  where extremum is taken with respect to **all possible distributions** $q(Z)$

- What if we limit ourselves with more restricted set of distributions?..

Key idea:
Inference becomes optimization

Variational Bayes

- We approximate posterior $p(Z|X_{tr}, \theta)$ by optimizing

$$\mathcal{L}(q, \theta) = \int q(Z) \log \frac{p(X_{tr}, Z|\theta)}{q(Z)} dZ$$

  in some restricted set of distributions $q(Z)$

- Note that all elements inside intergral are computable
- Maximization of $\mathcal{L}(q, \theta)$ w.r.t. $q(Z)$ corresponds to minimization of KL-divergence between $q(Z)$ and $p(Z|X_{tr}, \theta)$

Mean-field and parametric approximations

- Mean-field approximation: We impose factorization constraints

$$q(Z) = \prod_k q(z_k)$$

  In case of so-called **conditional conjugacy** iterative update equations can be derived for $\log q(z_k)$

- Parametric approximation: We require $q(Z)$ to belong to some parametric family

$$q(Z) = q(Z|\phi)$$

  Then we obtain parametric optimization problem

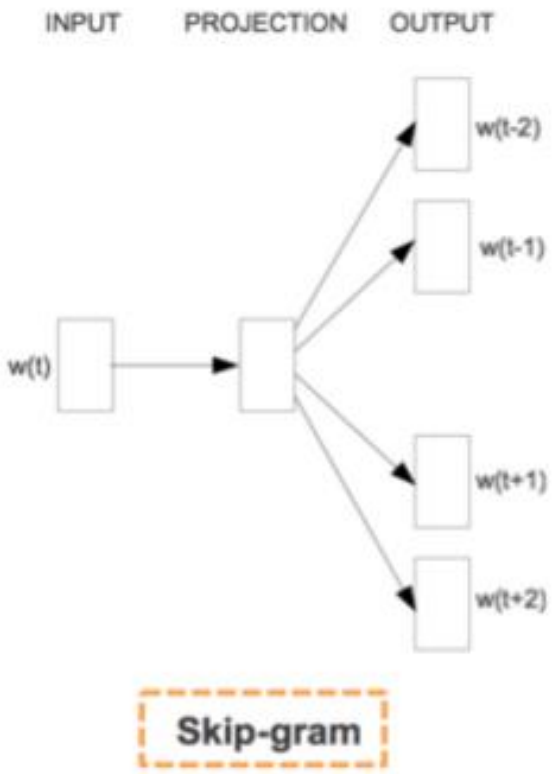
$$\mathcal{L}(q, \theta) = \mathcal{L}(\phi, \theta) \to \max_{\phi}$$

Case study: AdaGram

I guess we should make a coffee-break before we proceed ☺

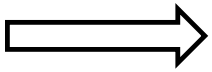
Word2vec model (Mikolov2013)

- 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)
…	…

$$p(y|x) = \frac{\exp\left(In(x)^T Out(y)\right)}{\sum_{y'} \exp\left(In(x)^T Out(y')\right)}$$

$$p(Y|X) \rightarrow \max_{\{In,Out\}}$$

This is how it should work in ideal case. The problem is with denominator which ensures normalization. It requires $O(V)$ to compute it for each x

Hierarchical soft-max

- Let us construct binary Huffman tree for our dictionary
- Each word y to be predicted corresponds to a leaf in the tree
- Denote $Path(y)$ the sequence of internal nodes from root to leaf y
- Denote $d_{c,y}$ the direction of further path from c to y:

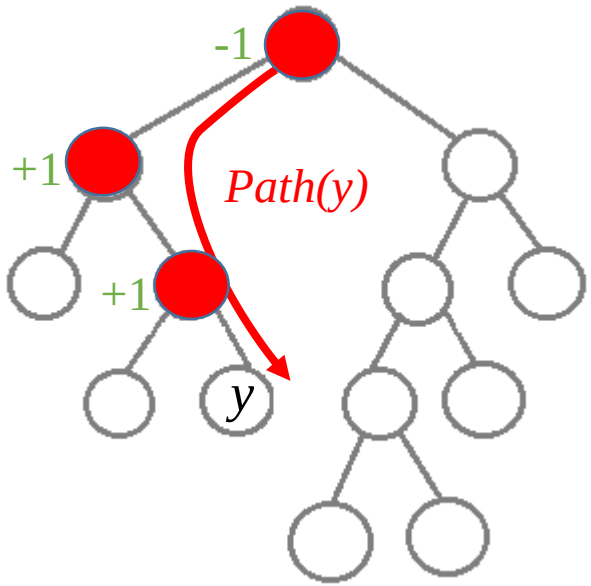
$$d_{c,y} = \begin{cases} +1 & y \text{ is in right subtree} \\ -1 & y \text{ is in left subtree} \end{cases}$$

- Then

$$p(y|x) = \prod_{c \in Path(y)} \sigma\left(d_{c,y} In(x)^T Out(c)\right),$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$

- Reduce complexity from $O(V)$ to $O(\log V)$



Semantic properties of representations

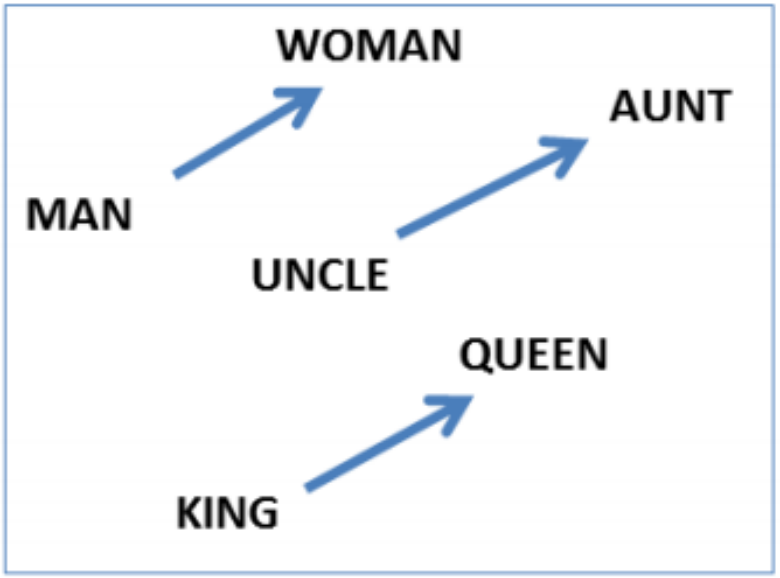
- Most known property of word2vec model: algebraic operations on vectors correspond to semantic operations on senses:

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

  Thousands of examples!

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

- Can it be useful for Q&A models?

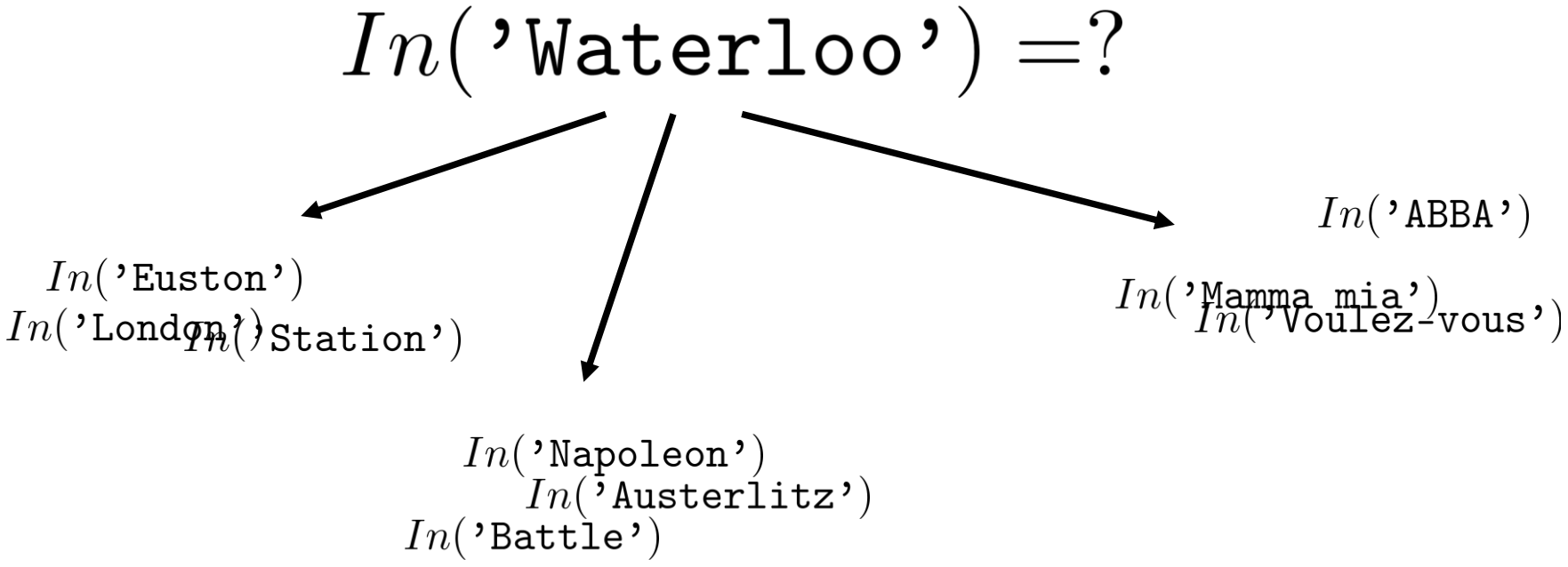


Word ambiguity

- Suppose we want to answer the question
  `When was the Battle of Waterloo?`
- Well... It depends on whether the following holds true:

$$In(\texttt{'Waterloo'}) - In(\texttt{'Battle'}) + In(\texttt{'Date'}) \approx In(\texttt{'1815'})$$

- Even if we succeed we will not be able to answer any questions about the song or the railway station



Word2vec summary

Pros

- Learns untrivial and abstract concepts
- Extremely computationally effective (less than an hour of training on the whole Wikipedia using SGD)
- Usable not only for the words (sentences, abstracts, graphs, etc.)

Cons

- Unique representation for each word regardless of the meaning of the particular word occurence
- Dependant on the choice of a tree in hierarchical soft-max

Multi-sense extension of skip-gram

- For simplicity assume we know the number of meanings for each word
- Define the latent variable z_i that indicates meaning of particular word occurence x_i
- Let us search for vector representations of meanings rather than words $In(x_i, z_i)$
- Now it is easy to define the probability of y_i given the context word and its meaning:

$$p(y_i|x_i, z_i) = \prod_{c \in Path(y_i)} \sigma\left(d_{c,y_i} In(x_i, z_i)^T Out(c)\right),$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$

Multi-sense extension of skip-gram

- We have defined $p(y_i|x_i, z_i)$. To finish model we need to set $p(z_i|x_i)$ that is prior probability of particular meaning for a given word
- In case of absense of any knowledge we may just set it to uniform distribution

$$p(z_i = k|x_i) = \frac{1}{K(x_i)},$$

  where $K(x_i)$ is total number of meanings for word x_i
- Now we have complete discriminative model

$$p(y_i, z_i|x_i) = p(y_i|x_i, z_i)p(z_i|x_i)$$

- If we knew z_i this would be just standard skip-gram model with additional context words
- Since we do not know it we can now use EM-algorithm that will both estimate our parameters $\{In(x, z), Out(c)\}$ and the probabilities of meanings of x_i given its neighbour: $p(z_i|x_i, y_i)$

Naïve EM algorithm

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

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

We can do this in explicit manner assuming the number of meanings is reasonably small

```
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

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

  We can do this in explicit manner assuming the number of meanings is reasonably small

- M-step: Optimize w.r.t. $\{In(x, z), Out(c)\}$

$$\mathbb{E} \log p(Y|Z, X)p(Z|X) \to \max_{\{In,Out\}}$$

  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

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

  We can do this in explicit manner assuming the number of meanings is reasonably small

- M-step: Optimize w.r.t. $\{In(x, z), Out(c)\}$

$$\mathbb{E} \log p(Y|Z, X)p(Z|X) \to \max_{\{In, Out\}}$$

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

- Seems computationally efficient?.. NO!

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

Stochastic optimization

- Extremely efficient technique for large-scale optimization of $f(x)$
- Uses unbiased estimates $g(x)$ instead of true gradients $\nabla f(x)$
- (Robbins, Monro, 1951) If $f(x)$ is differentiable, $\mathbb{E}g(x) = \nabla f(x)$, $\forall x$, and $\sum_k \alpha_k = +\infty$, $\sum_k \alpha_k^2 < +\infty$, $\alpha_k > 0$ then

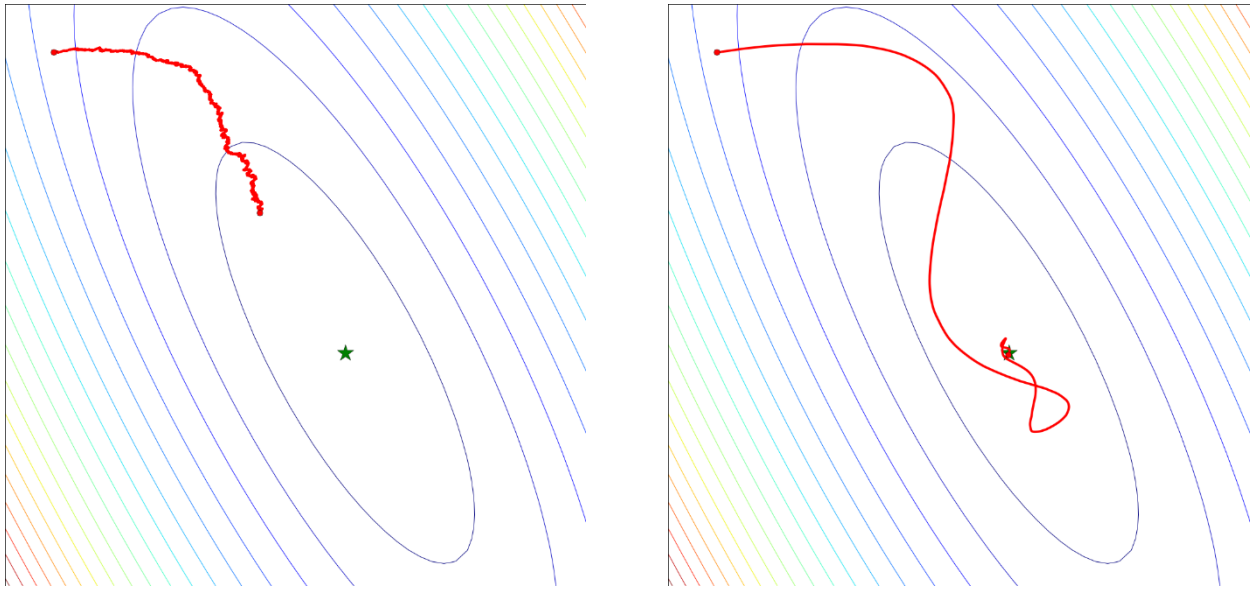
$$x_{k+1} = x_k + \alpha_k g(x_k)$$

  converges to stationary point of $f(x)$

- Convergence is sublinear (very slow!) and slows down with the increase of $\mathbb{D}g(x)$

Advanced techniques

- Modern stochastic optimization methods (SAG, Adam, SFO, IN2, etc.) use either momentum, memory, or unbiased estimates of Hessian to speed up the convergence
- Variance reduction techniques (controled variates, reparametrization, etc.) are also crucial
- Linear and in cases even superlinear convergence



Stochastic gradients

Function	Stochastic gradient
$f(x) = \sum_{i=1}^{N} f_i(x)$	$\nabla f_i(x)$
$f(x) = \mathbb{E}_y h(x, y) = \int p(y) h(x, y) dy$	$\frac{\partial}{\partial x} h(x, y_0), \quad y_0 \sim p(y)$
$f(x) = \mathbb{E}_{y\|x} h(x, y) = \int p(y\|x) h(x, y) dy$	$\frac{\partial}{\partial x} h(x, y_0) + h(x, y_0) \frac{\partial}{\partial x} \log p(y_0\|x), \quad y_0 \sim p(y\|x)$

Last example has extremely large variance!
Variance reduction is needed

Large scale EM

- Remember our scheme
- E-step: For **each** training object estimate the distribution on latent variable

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

  We can do this in explicit manner assuming the number of meanings is reasonably small

- M-step: Optimize w.r.t. $\{In(x, z), Out(c)\}$

$$\mathbb{E} \log p(Y|Z, X)p(Z|X) \to \max_{\{In, Out\}}$$

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

Large scale EM

- Remember our scheme
- E-step: For **each** training object estimate the distribution on latent variable

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

  We can do this in explicit manner assuming the number of meanings is reasonably small

- M-step: Optimize w.r.t. $\{In(x, z), Out(c)\}$

$$\mathbb{E} \log p(Y|Z, X)p(Z|X) \to \max_{\{In, Out\}}$$

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

- What if on M-step we try to make a single step towards stochastic gradient of $\mathbb{E} \log p(Y|Z, X)p(Z|X)$?

Large-scale EM

- Consider the gradient of $\mathbb{E} \log p(Y|Z,X)p(Z|X)$ in detail

$$\nabla \mathbb{E}_Z \log p(Y|Z,X)p(Z|X)$$

Large-scale EM

- Consider the gradient of $\mathbb{E} \log p(Y|Z,X)p(Z|X)$ in detail

$$\nabla \mathbb{E}_Z \log p(Y|Z,X)p(Z|X) \;=\; \nabla \mathbb{E}_Z \sum_{i=1}^{n} (\log p(y_i|z_i,x_i) + \log p(z_i|x_i))$$

Large-scale EM

- Consider the gradient of $\mathbb{E} \log p(Y|Z,X)p(Z|X)$ in detail

$$\nabla \mathbb{E}_Z \log p(Y|Z,X)p(Z|X) = \nabla \mathbb{E}_Z \sum_{i=1}^{n} (\log p(y_i|z_i,x_i) + \log p(z_i|x_i)) =$$

$$\sum_{i=1}^{n} \mathbb{E}_{z_i} (\nabla \log p(y_i|z_i,x_i) + \nabla \log p(z_i|x_i))$$

Large-scale EM

- Consider the gradient of $\mathbb{E} \log p(Y|Z,X)p(Z|X)$ in detail

$$\nabla \mathbb{E}_Z \log p(Y|Z,X)p(Z|X) = \nabla \mathbb{E}_Z \sum_{i=1}^{n} (\log p(y_i|z_i,x_i) + \log p(z_i|x_i)) =$$

$$\sum_{i=1}^{n} \mathbb{E}_{z_i} (\nabla \log p(y_i|z_i,x_i) + \nabla \log p(z_i|x_i)) = \sum_{i=1}^{n} \mathbb{E}_{z_i} (\nabla \log p(y_i|z_i,x_i))$$

Does not depend on *{In, Out}*

Large-scale EM

- Consider the gradient of $\mathbb{E} \log p(Y|Z,X)p(Z|X)$ in detail

$$\nabla \mathbb{E}_Z \log p(Y|Z,X)p(Z|X) = \nabla \mathbb{E}_Z \sum_{i=1}^{n} (\log p(y_i|z_i,x_i) + \log p(z_i|x_i)) =$$

$$\sum_{i=1}^{n} \mathbb{E}_{z_i} (\nabla \log p(y_i|z_i,x_i) + \nabla \log p(z_i|x_i)) = \sum_{i=1}^{n} \mathbb{E}_{z_i} (\nabla \log p(y_i|z_i,x_i))$$

Does not depend on *{In, Out}*

- Its unbiased estimate is simply

$$\mathbb{E}_{z_i} \nabla \log p(y_i|z_i,x_i) = \sum_{j=1}^{K(x_i)} p(z_i = k|y_i,x_i) \nabla \log p(y_i|k,x_i)$$

Large-scale EM

- Consider the gradient of $\mathbb{E} \log p(Y|Z,X)p(Z|X)$ in detail

$$\nabla \mathbb{E}_Z \log p(Y|Z,X)p(Z|X) = \nabla \mathbb{E}_Z \sum_{i=1}^{n} (\log p(y_i|z_i,x_i) + \log p(z_i|x_i)) =$$

$$\sum_{i=1}^{n} \mathbb{E}_{z_i} (\nabla \log p(y_i|z_i,x_i) + \nabla \log p(z_i|x_i)) = \sum_{i=1}^{n} \mathbb{E}_{z_i} (\nabla \log p(y_i|z_i,x_i))$$

Does not depend on *{In, Out}*

- Its unbiased estimate is simply

We know from E-step

$$\mathbb{E}_{z_i} \nabla \log p(y_i|z_i,x_i) = \sum_{j=1}^{K(x_i)} p(z_i = k|y_i,x_i) \nabla \log p(y_i|k,x_i)$$

Large-scale EM

- Consider the gradient of $\mathbb{E} \log p(Y|Z,X)p(Z|X)$ in detail

$$\nabla \mathbb{E}_Z \log p(Y|Z,X)p(Z|X) = \nabla \mathbb{E}_Z \sum_{i=1}^{n} (\log p(y_i|z_i,x_i) + \log p(z_i|x_i)) =$$

$$\sum_{i=1}^{n} \mathbb{E}_{z_i} (\nabla \log p(y_i|z_i,x_i) + \nabla \log p(z_i|x_i)) = \sum_{i=1}^{n} \mathbb{E}_{z_i} (\nabla \log p(y_i|z_i,x_i))$$

Does not depend on *{In, Out}*

- Its unbiased estimate is simply

We know from E-step

$$\mathbb{E}_{z_i} \nabla \log p(y_i|z_i,x_i) = \sum_{j=1}^{K(x_i)} p(z_i = k|y_i,x_i) \nabla \log p(y_i|k,x_i)$$

- But to compute it we only need to know $p(z_i|y_i,x_i)$ for single training instance!

Sketch of the final algorithm

- Build Huffman tree for the dictionary
- Fix initial approximation for each $\theta = \{In(x, z), Out(c)\}$
- Do one pass through training data
  - Compute the probabilities of meanings for x_i

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

  - Make one step towards stochastic gradient:

$$\theta_{new} = \theta_{old} + \alpha_i \sum_{k=1}^{K(x_i)} p(z_i = k|x_i, y_i)\nabla_\theta \log p(y_i|x_i, k)$$

What was not covered in this talk

- Each word occurence is present $2C$ times in training set and of course the corresponding x_i should have the same meaning
- We may use so-called non-parametric Bayesian inference to automatically define the number of meanings for each word
- To do this we need to set a special prior on $p(z_i|x_i)$ using so-called **Chinese restaurant process**
- To obtain tractable approximations for $p(z_i|x_i, y_i)$ we'll need to use Stochastic variational inference (Hoffman, 2013) which is similar to large-scale EM described above

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"

Downloads

- Code and documentation available

https://github.com/sbos/AdaGram.jl

- Trained models available

https://yadi.sk/d/W4FtSjA5o3jUL

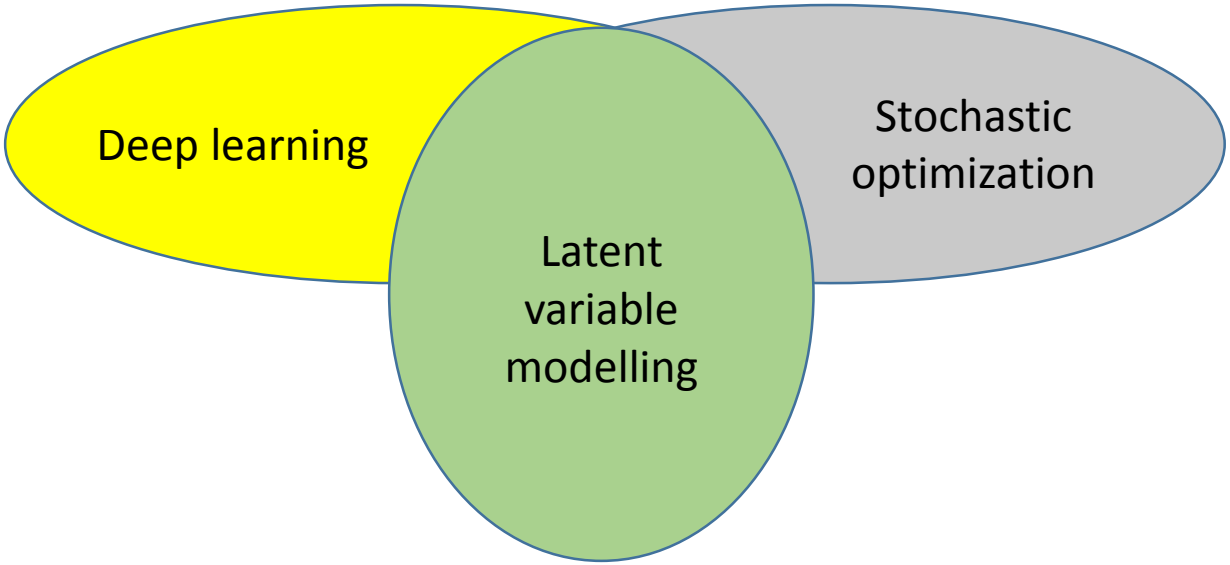
- Paper available

S. Bartunov, D. Kondrashkin, A. Osokin, D. Vetrov. Breaking Sticks and Ambiguities with Adaptive Skip-gram. In *AISTATS 2016*

http://arxiv.org/abs/1502.07257

Conclusion

- Latent variable modelling allows to uncover deeper dependencies in the data that are not obvious even in the training data
- Using LVM we may use weakly-annotated data and learn from multiple sources
- Stochastic optimization allows us to train LVM almost as fast as standard models



Deep Bayes

Variational auto-encoder

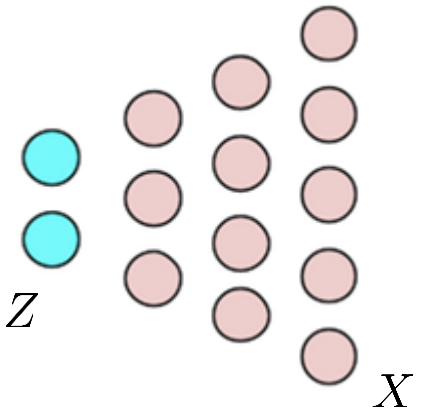
- VAE (Kingma14) is used for building non-linear low-dimensional representation Z of high-dimensional data X
- Generative model $p(X, Z|\theta)$ is established
- To perform inference

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

one needs to know **normalization constant**

$$\int p(X|Z,\theta)p(Z|\theta)dZ$$

- Prior over Z is simple $p(Z) = \mathcal{N}(Z|0, I)$ and likelihood $p(X|Z,\theta)$ is modeled by deep neural network with weights θ



Diederik P Kingma, Max Welling. Auto-Encoding Variational Bayes. arXiv:1312.6114

Variational auto-encoder

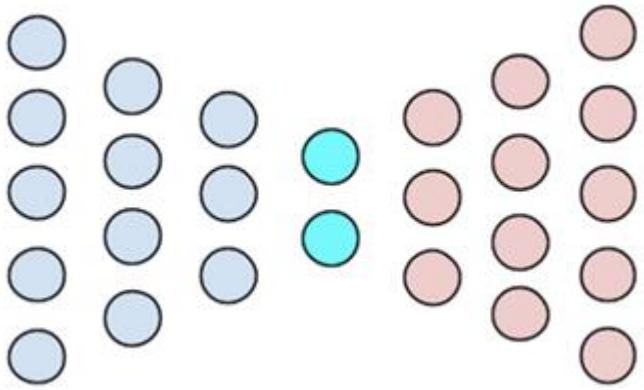
- Variational methods allow to **convert inference to optimization**
- Instead of computing

$$\int p(X|Z,\theta)p(Z|\theta)dZ$$

  we optimize varitional lower bound

$$\mathcal{L}(\phi,\theta) = \int q(Z|X,\phi)\log\frac{p(X,Z|\theta)}{q(Z|X,\phi)}dZ \to \max_{\phi}$$

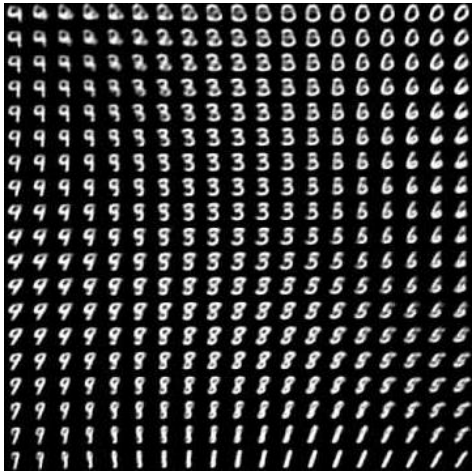
- Both $p(X|Z,\theta)$ and $q(Z|X,\phi)$ are modeled using deep neural networks



Parameterized by ϕ Parameterized by θ

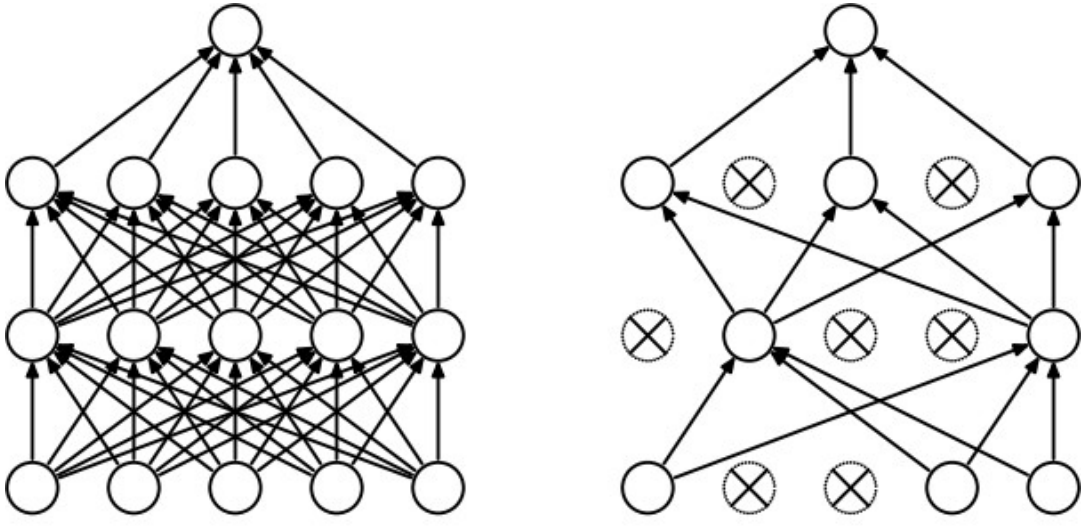
Advantages of VAE

- Better representation learning in terms of marginal likelihood $p(X|\theta)$ and visual feeling
- New basic model for multiple extensions
  - importance reweighted auto-encoders (Burda15)
  - adversarial auto-encoders (Makhzani15)
  - auto-encoders with normalizing flows (Rezende15)
  - conceptual compressors (Gregor16)
  - Structured VAE (Johnson16)
- Opens way for more accurate variational lower bounds



Dropout

- Proposed by Geoffrey Hinton's group (Srivastava14)
- Nullifies the outputs of randomly selected neurons at each iteration of training
- Purely heuristic procedure for preventing overfitting
- But it works!



(a) Standard Neural Net

(b) After applying dropout.

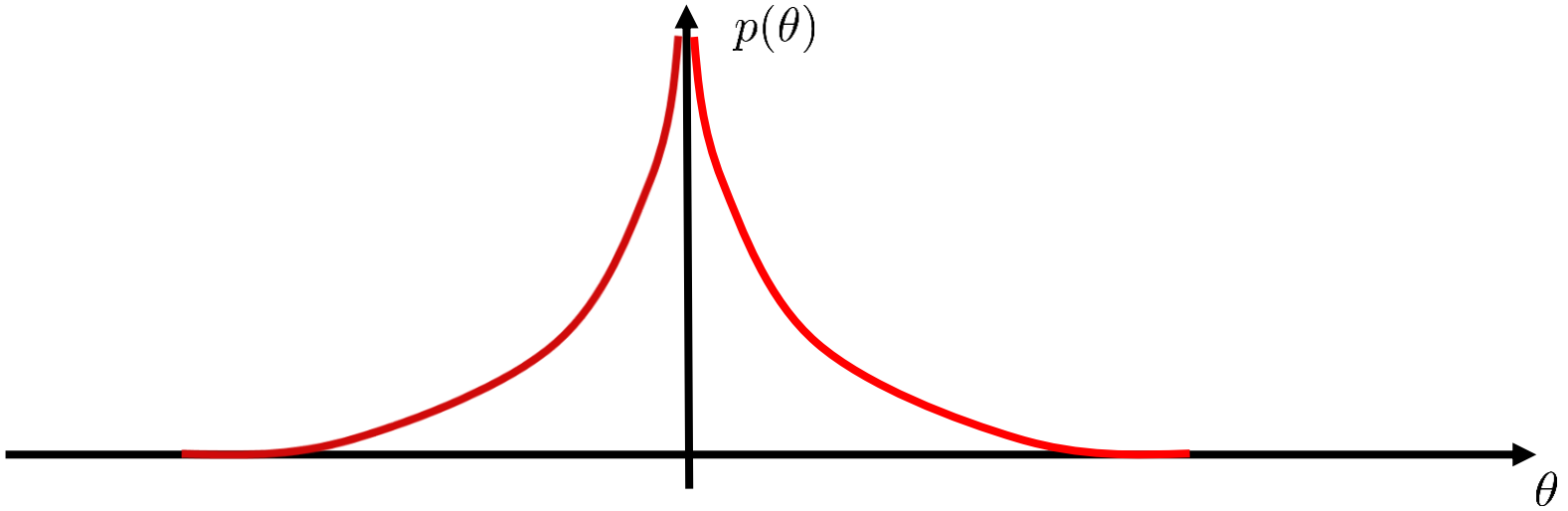
Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. JMLR, 2014.

Variational Dropout

- In December 2015 new techniques of variational dropout was suggested (Kingma15)
- It was shown that dropout correponds to Bayesian inference with special improper prior over the weights θ

$$p(\theta) \propto \frac{1}{|\theta|}$$

- This is so-called **scale-invariant prior** which penalizes the precision of θ



Diederik P. Kingma, Tim Salimans, Max Welling. Variational Dropout and the Local Reparameterization Trick. arXiv:1506.02557

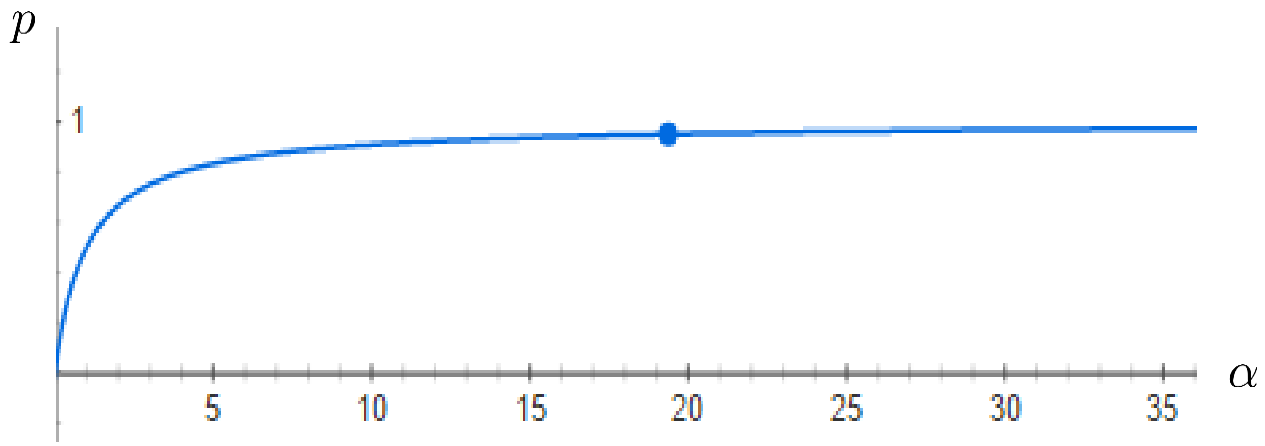
Variational Dropout

- We approximate posterior as gaussian distribution

$$q(\theta) = \prod_{i,j} q(\theta_{ij}) = \prod_{ij} \mathcal{N}(\theta_{ij}|\mu_{ij}, \alpha\mu_{ij}^2)$$

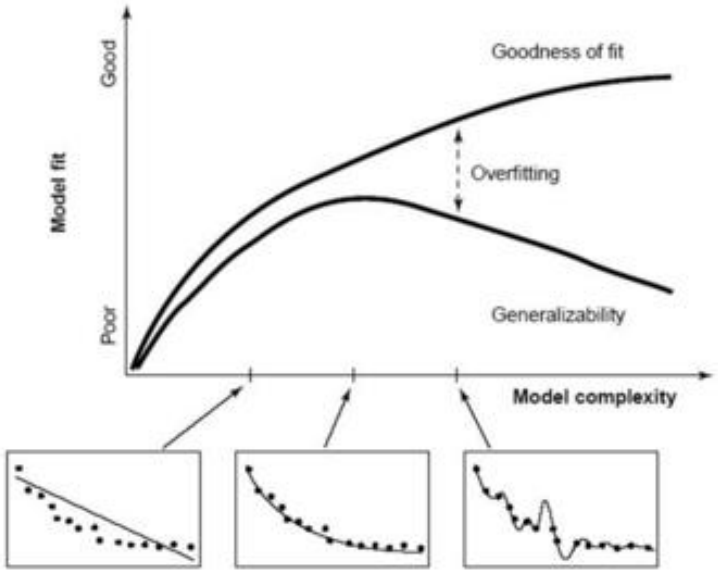
- Optimization of variational lower bound w.r.t. all μ_{ij} given α fixed correponds to standard droupout learning with dropout rate

$$p = \frac{\alpha}{1+\alpha}$$



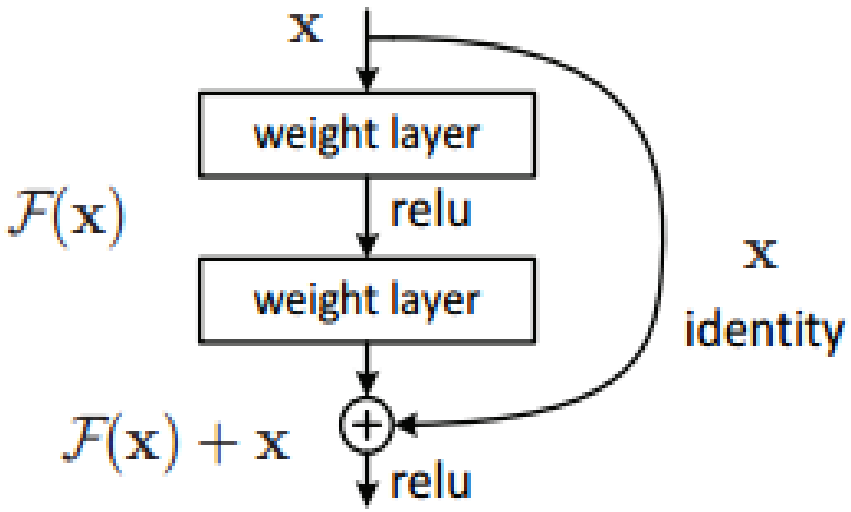
Advantages of variational dropout

- Now we understand that dropout is simply Bayesian regularizer on the weights of neural network
- We may play with other regularizers obtaining alternatives to dropout
- We may adjust droput rates individually for each neuron since we have Bayesian critrion that can be optimized w.r.t. α_{ij}
- By doing this we may perform ARD and adjust the proper structure of NN by performing Bayesian **model selection**



Residual nets

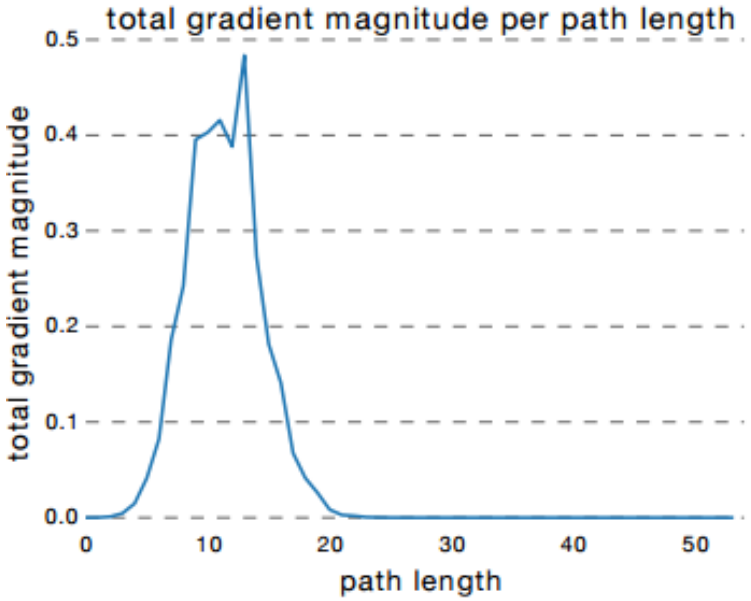
- Current state-of-the-art deep network for image classification with super-human performance (He15)
- Up to one thousand of layers
- Smart tricks to make it work
  - Skip-connections
  - Batch-normalization
  - Weight initialization
  - Stochastic depth



Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. arXiv:1512.03385

ResNet as ensembles

- Recently it was claimed that ResNets should be treated as exponentially large ensembles of networks (Veit16)
- ResNet with n layers is actually as ensemble of 2^n smaller networks
- The most relevant are networks with 15-17 layers of depth
- We could use Bayesian framework to optimize the structure of ensemble which is fixed in resnet



Andreas Veit, Michael Wilber, Serge Belongie. Residual Networks are Exponential Ensembles of Relatively Shallow Networks. arXiv:1605.06431

Summary

- **Variational auto-encoder.** Uses neural networks for building complex probabilistic model and perform approximate yet efficient inference. This dramatically extends the area of application of Bayesian models
- **Variational dropout.** Uses Bayesian regularization for preventing overfitting. We may now do model selection for deep neural networks by adjusting their architecture automatically
- Much more to be done...

Group’s other projects

- TensorNet
- Perforation of convolutions
- Superlinear stochastic optimization

Tensor Train decomposition

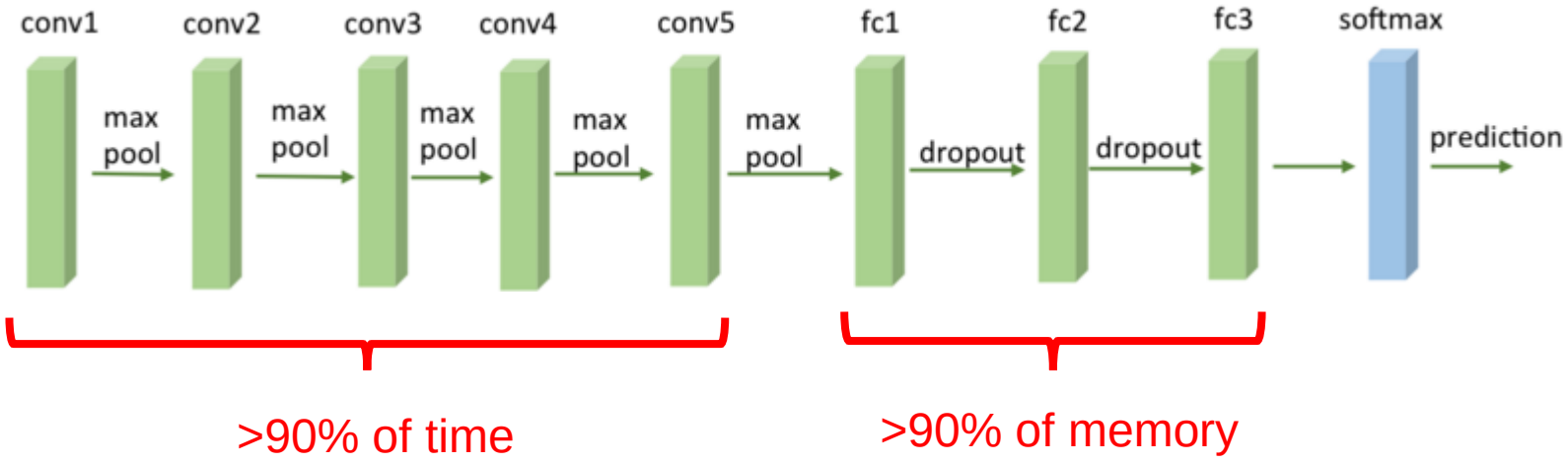
- Tensor is a multi-dimensional array
- The amount of elements in tensor grows up exponentially
- Tensor decompositions provide special format for keeping its elements in a compact form
- We may perform operations on tensors directly in this form
- Tensor train is one of the most promising formats:

$$A[i_1, \ldots, i_n] \approx G_1[i_1] \ldots G_n[i_n], \text{ where } G_k[i_k] \in \mathbb{R}^{r_{k-1} \times r_k}$$

- We may think of it as a multi-dimensional generalization of SVD

TensorNet

- Modern deep neural networks are extremely memory demanding
- Up to 99% of memory are required for keeping fully-connected layers due to the need of keeping weight matrices in memory
- Memory limits the depth and width of neural networks



TT-layer

- We represent weight matrix as tensor (multi-dimensional array)

$$W[i, j] = W[i_1, \ldots, i_d, j_1, \ldots, j_d] = G^1[i_1, j_1] \ldots G^d[i_d, j_d]$$

- Extremely compact form for keeping matrix elements by exploiting strong redundancy within the network
- We may propagate the signal through TT-layer **without** unrolling it back to matrix

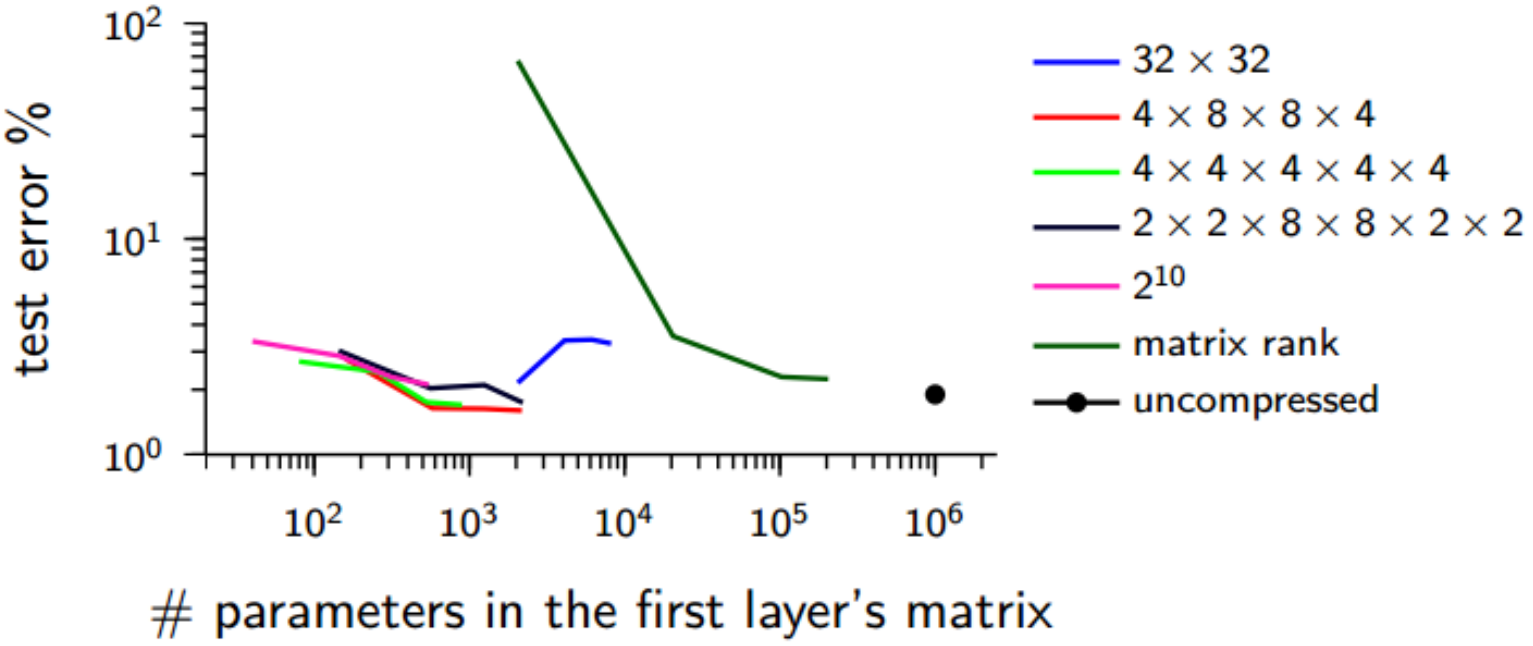
$$y = Wx = \sum_j W[i, j]x[j] = \sum_{j_1, \ldots, j_d} G^1[i_1, j_1] \ldots G^d[i_d, j_d]x[j_1, \ldots, j_d]$$

  All we need to do is to reshape matrices and vectors into 2d and d-dimensional arrays respectively

- The TT-cores $G^k[i_k, j_k]$ can be trained via back-propagation

Compression rates

- TT-layers allow to achieve impressive compression rates without significant acuracy drop
- The preformance depends on the choice of TT-ranks, i.e. the dimensionality of TT-cores $G^k[i_k, j_k]$



VGG-16 compression

Architecture	Matrices compr.	Network compr.	Error
FC FC FC	1	1	30.9
TT4 FC FC	50 972	3.9	31.2
TT2 FC FC	194 622	3.9	31.5
TT1 FC FC	713 614	3.9	33.3
TT4 TT4 FC	37 732	7.4	32.2
LR50 FC FC	70	3.7	36.7

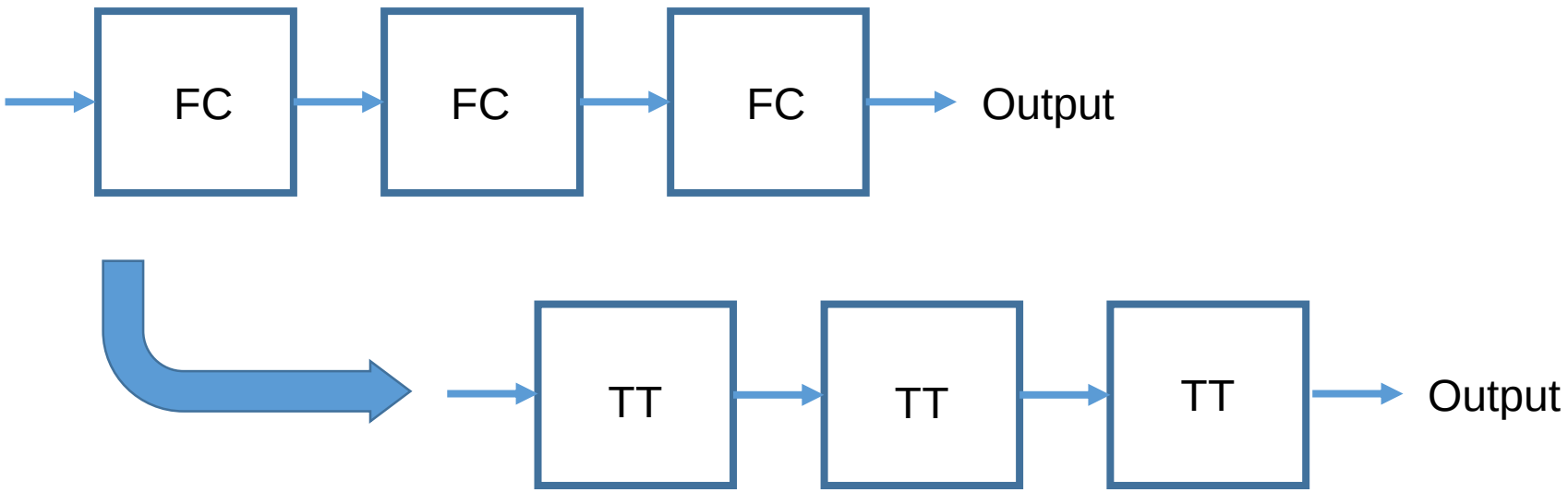
A 3-layered network, FC stands for the traditional layer; TT stands for the TT-layer with all the TT-ranks equal "□"; LR stands for the rank decomposition with the rank equal "□".

TT2 FC FC: the number of the parameters in the matrix W of the largest fully-connected layer is compressed by a factor of 194622 (from 25088×4096 parameters to 528).

A. Novikov, D. Podoprikhin, A. Osokin, D. Vetrov. Tensorizing Neural Networks. In Advances in Neural Information Processing Systems 28 (NIPS). 2015.

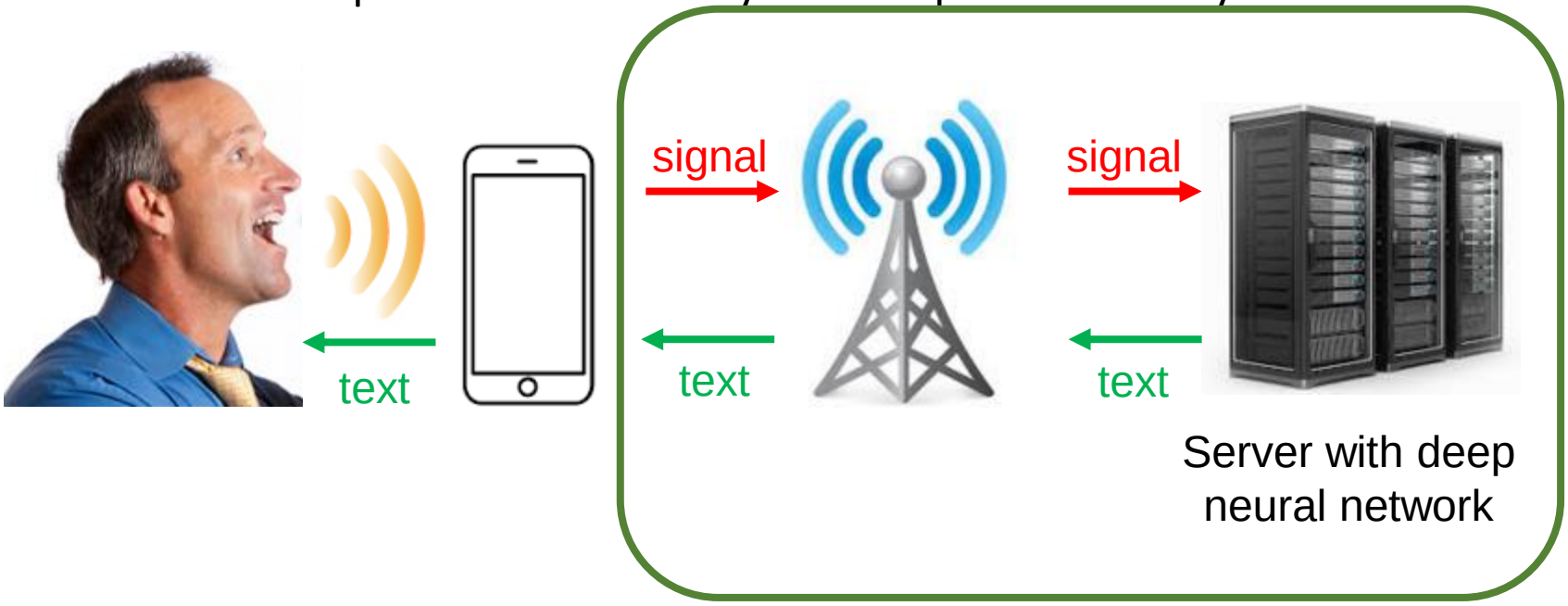
Work in progress

- Currently working on using several TT-layers instead of standard fully-connected ones
- Gradients become unstable
- Recently proposed batch-normalization (Ioffe15) techniques helps
- We are developing methods for projection of gradient to low-rank magnifold to speed up training and make it more robust



Areas of application

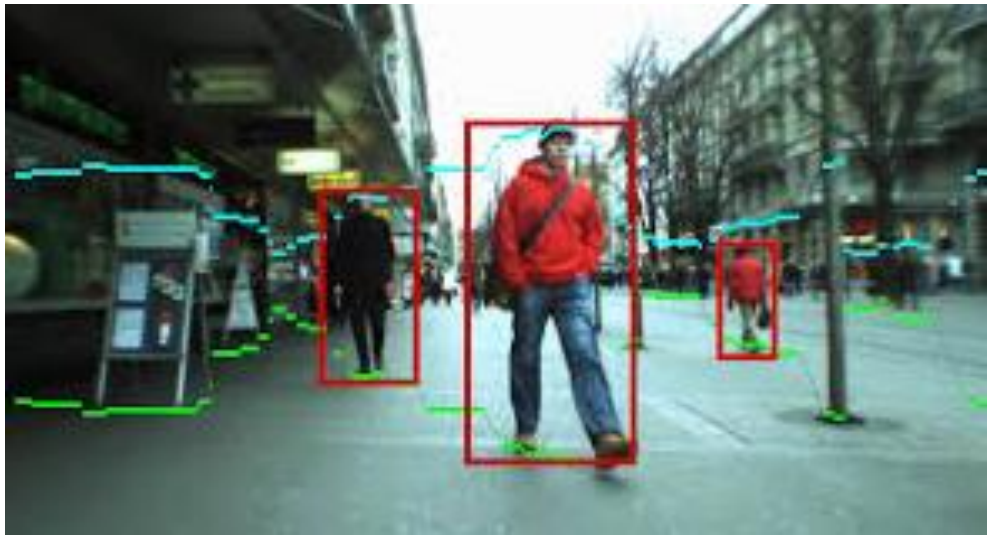
- Implementation of current state-of-art architectures in devices with limited memory such as mobile devices, TV-sets, etc.
- No need in transmitting signal to server
- Increase of depth of current FC layers to improve accuracy



Becomes optional rather than necessary

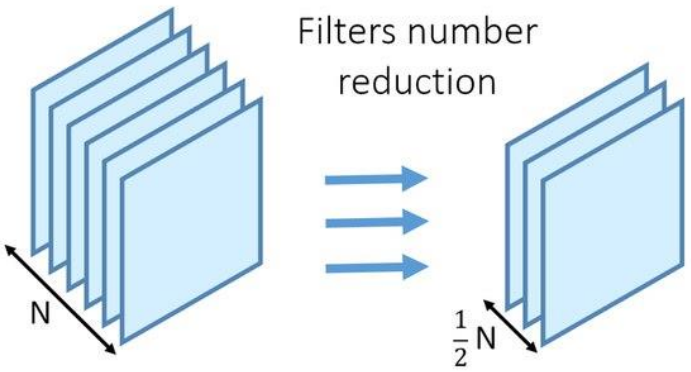
Accelerating CNN

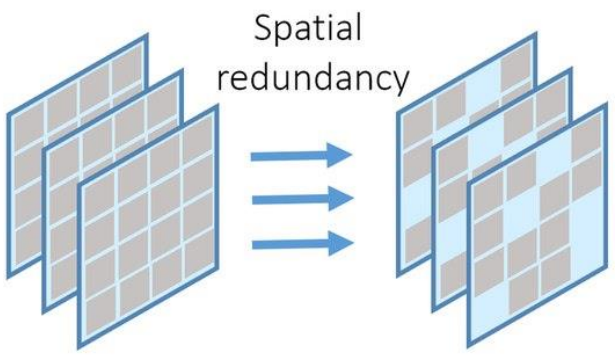
- Modern state-of-the-art deep neural networks require billions of computations for one forward pass
- The convolution operation used in convolutional neural networks is especially computationally expensive
- This makes it problematic to use CNNs in real-time e.g. for video processing



Key ideas

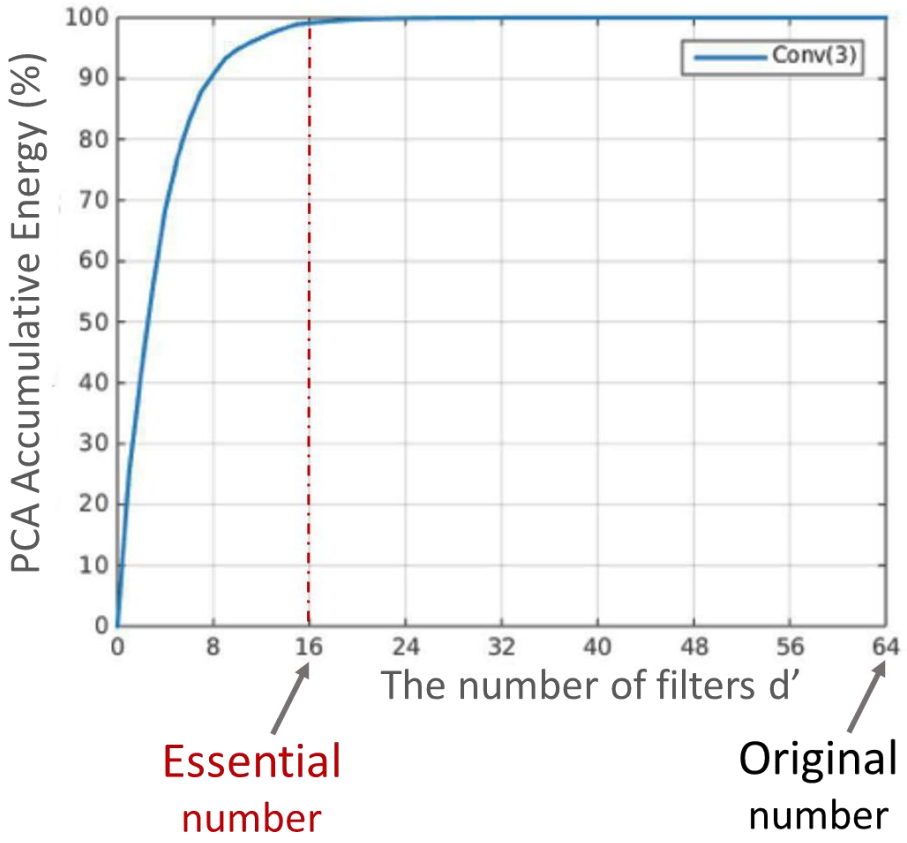
- Deep ConvNets contain a lot of redundant information which slows down the network
- Optimizing the network can accelerate it and conserve memory





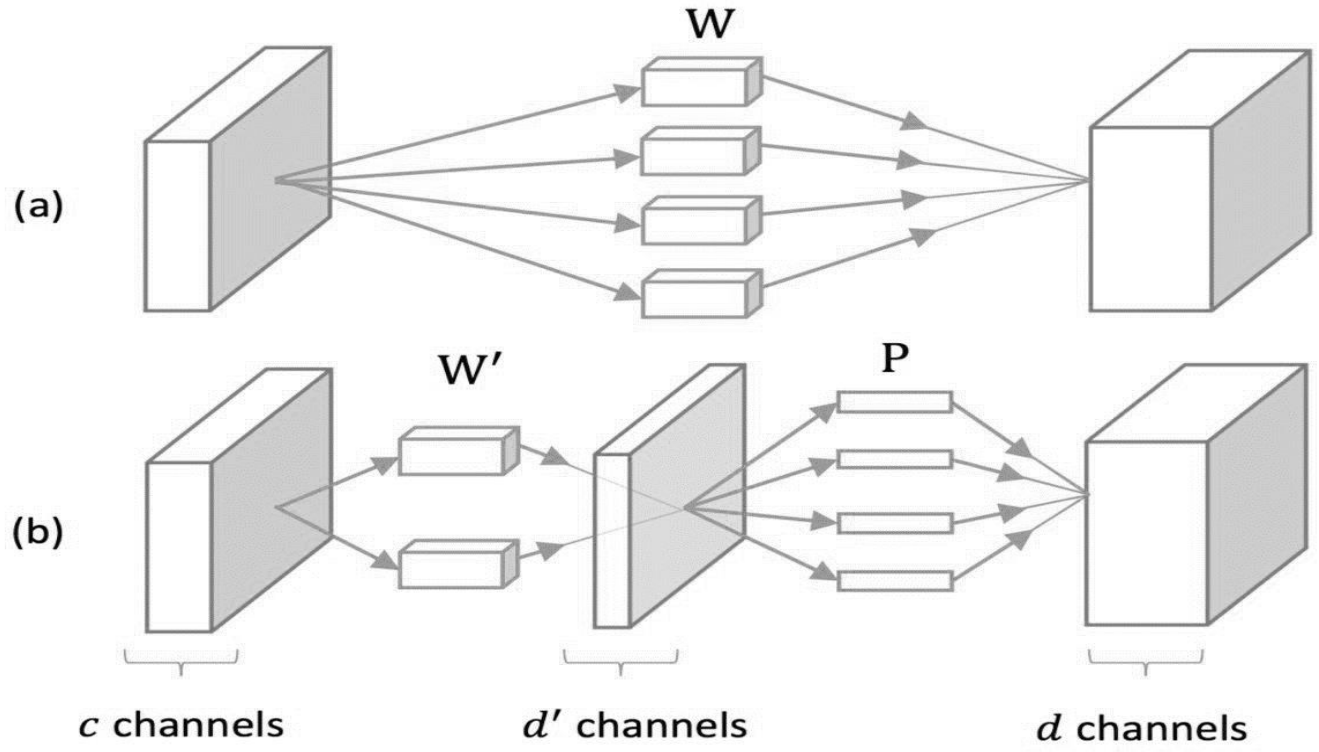
Filter redundancy

PCA Energy shows how much information can be extracted from d most important filters.



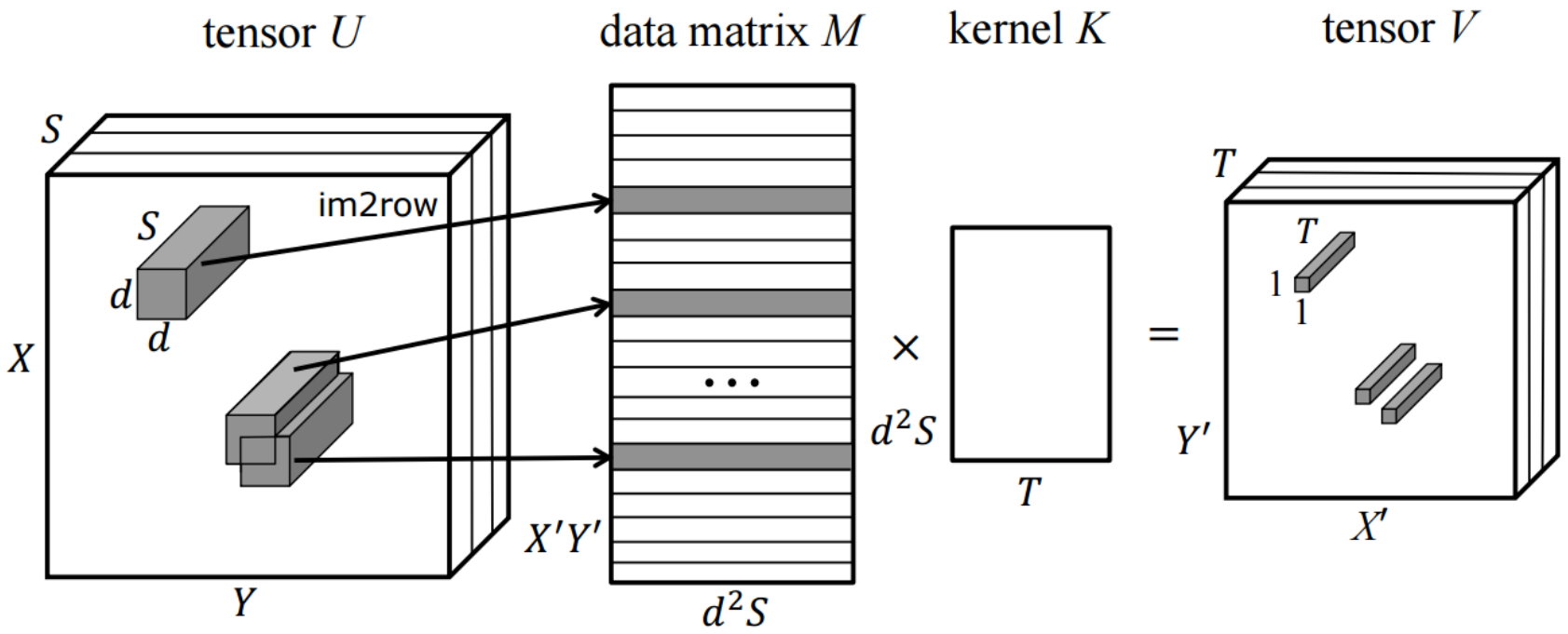
It is enough to use only ¼ filtersfor most conv. layers.

Method 1



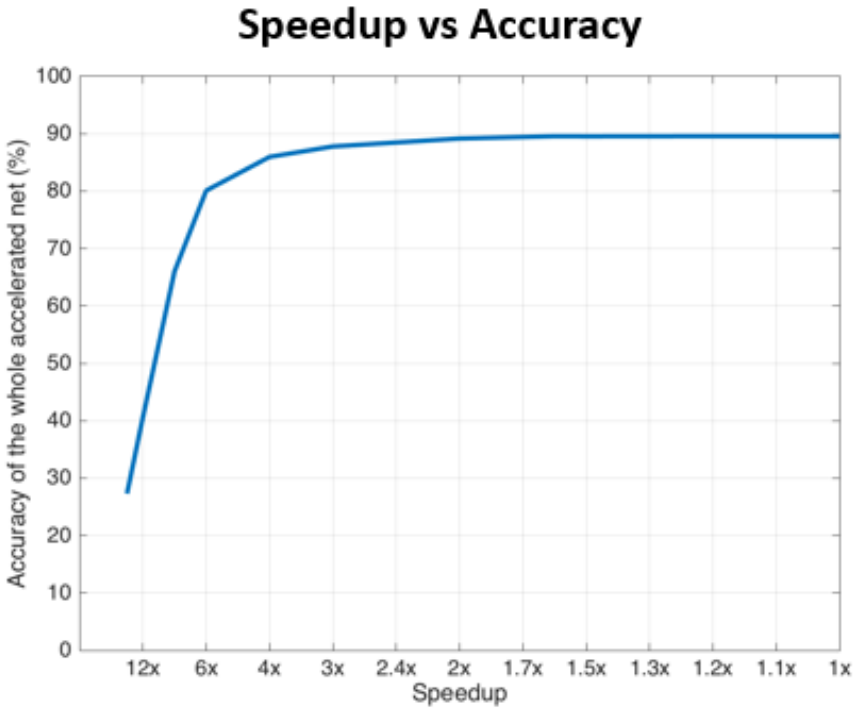
Key Idea: Reduce the number of filters and substitute original layer with 2 lighter ones (Zhang15)

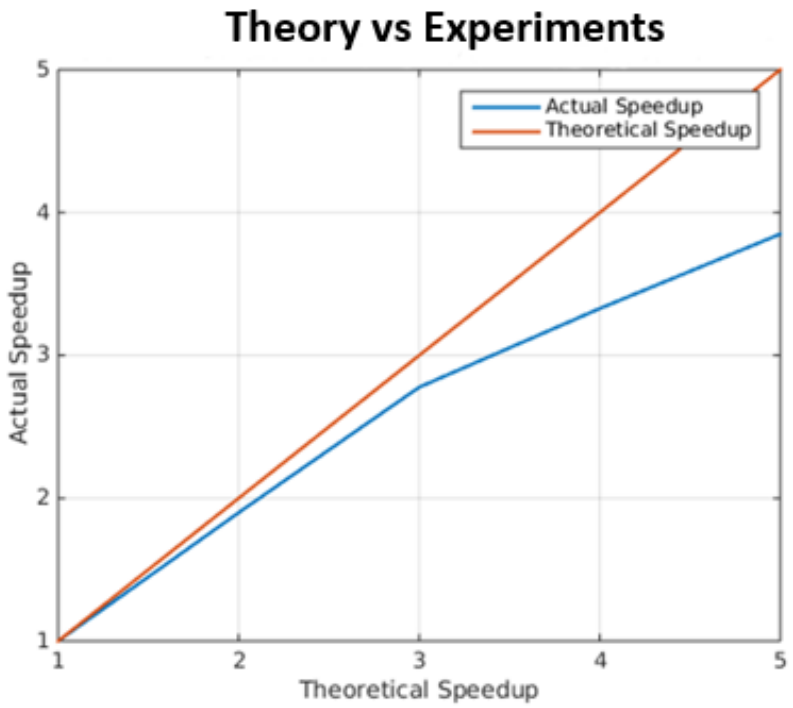
Method 2



Key Idea: Perform perforation and compute only subset of convolutions while interpolating the remaining ones

Trade-off between speed and accuracy





With significant acceleration of convolutional layers fully-connected layers start slowing down the network

Acceleration of VGG-16

Original Error 10.1%

4x speedup	CPU speedup	GPU speedup	Error increase
Method 1	4.0x	2.8x	+5.5%
Method 2	3.8x	2.9x	+3.8%

2x speedup: 1.7x less memory, 1.1% increase of top-5 error

Models can be ensembled with increase in speedup and preservation in the error rate.

After combination and tuning: 4x faster and 2.5x lighter VGG-16 with negligible drop in accuracy

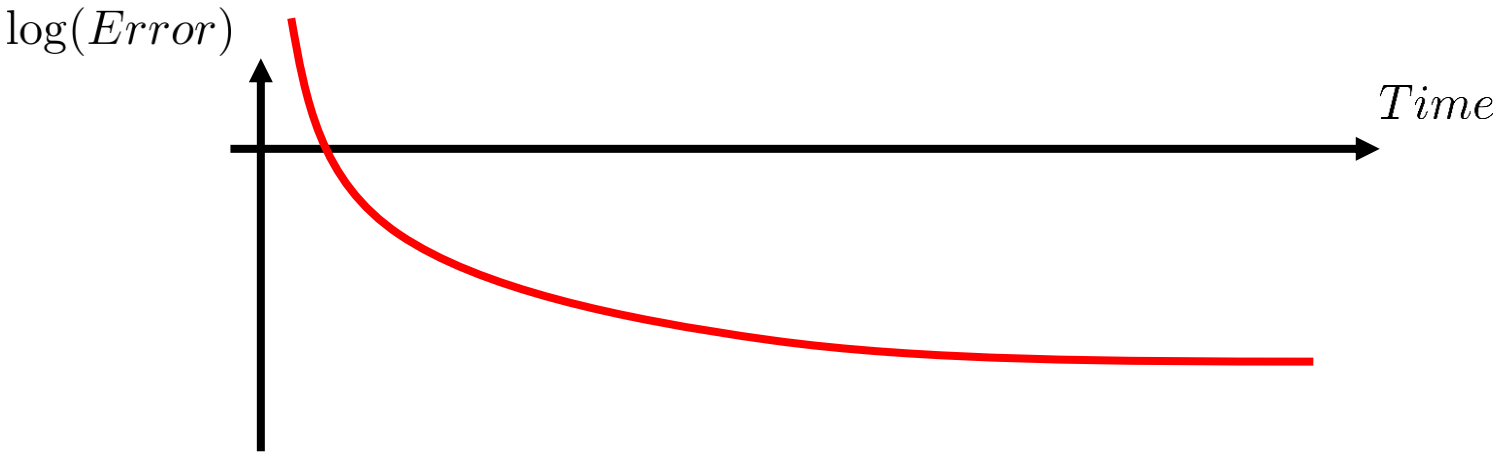
M. Figurnov, D. Vetrov, P. Kohli. PerforatedCNNs: Acceleration through Elimination of Redundant Convolutions. International Conference on Learning Representations (ICLR) 2016

Areas of application

- Mobile devices
- Real-time augmented reality
- Self-driving cars

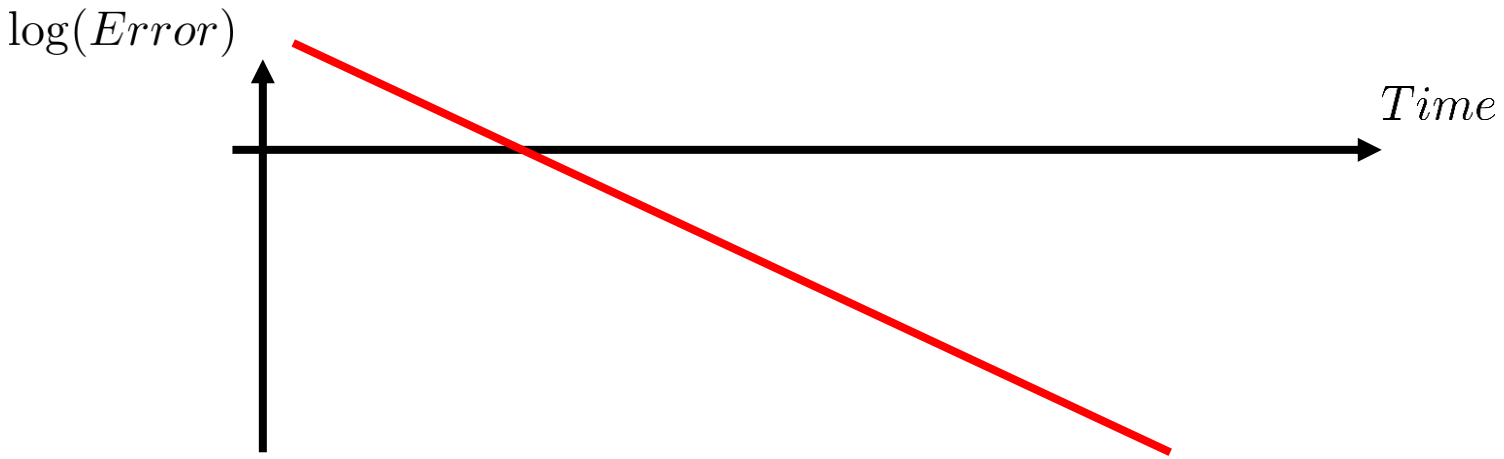
Stochastic gradient

- Stochastic gradient descend (SGD) has revolutionized large-scale machine learning
- Modern deep neural networks are trained using SGD and its variations
- However SGD has **sublinear** convergence
- This means it is able to give only very rough estimate for local extremum during reasonable time



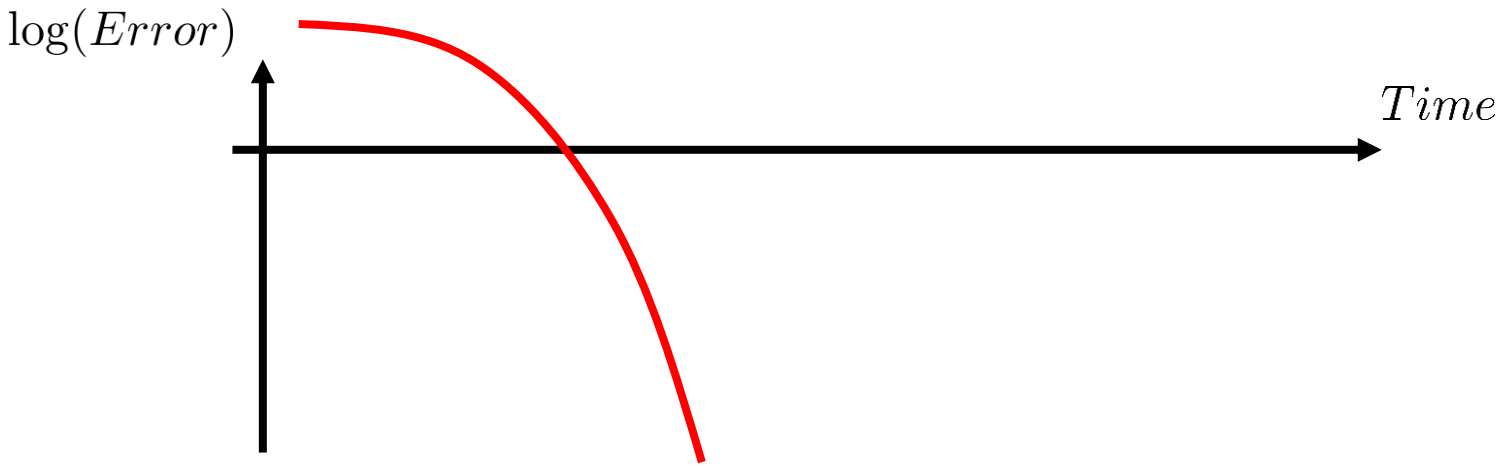
Recent modifications

- Recently proposed **incremental** methods such as SAG or SVRG have **linear** convergence
- The complexity of their iteration does not depend on the size of dataset like in SGD
- The increase in speed is achieved due to larger memory requirements



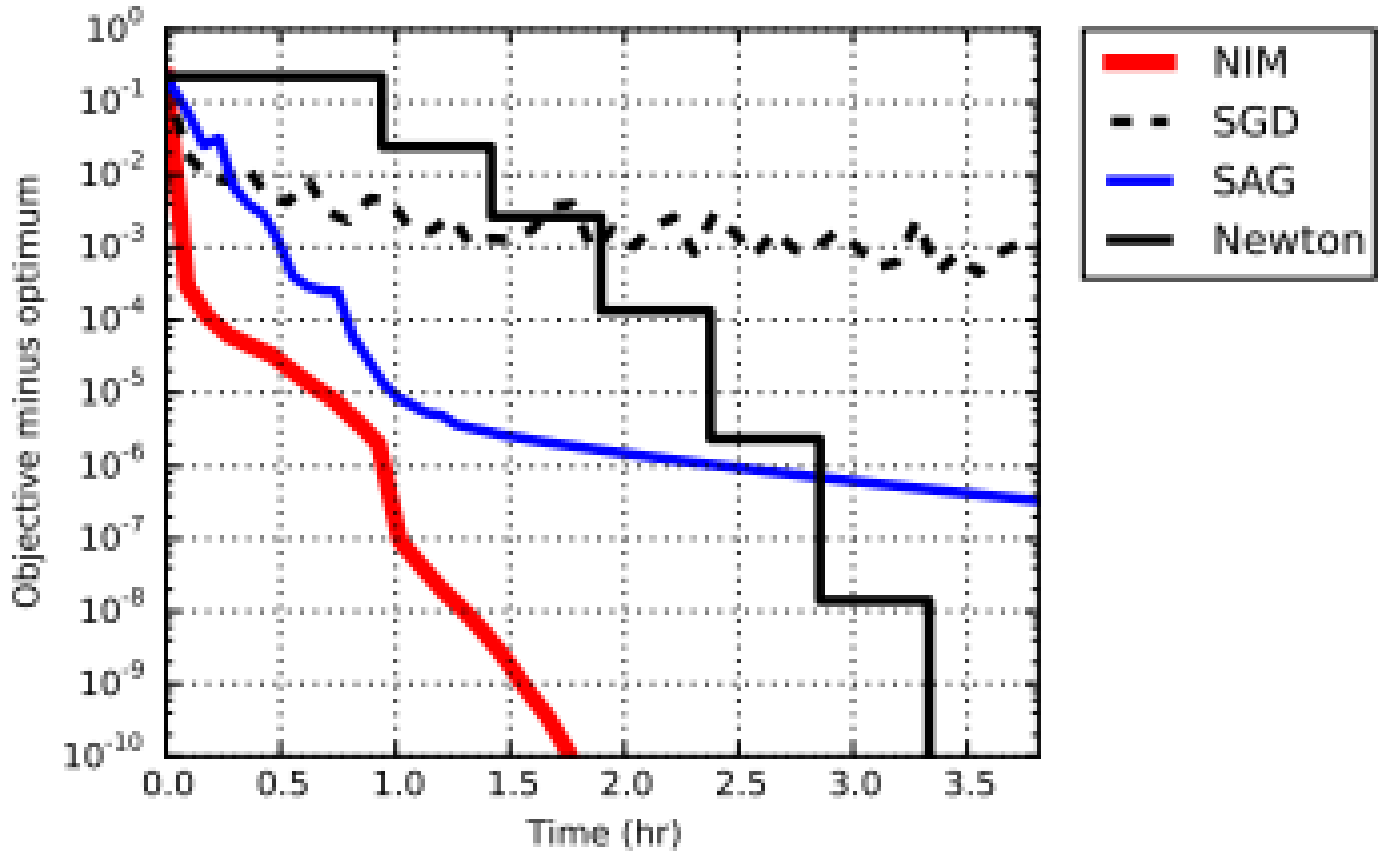
Incremental Newton

- We are developing incremental 2^{nd} order methods which use stochastic estimates for the Hessian matrix
- This method is the first incremental algorithm with **superlinear** convergence speed
- Potentially applicable to small neural networks



Incremental Newton

Training ℓ_2-regularized logistic regression on dataset with $n = 8000000$ of objects and $d = 784$ features



A. Rodomanov, D. Kropotov. A Superlinearly-Convergent Proximal Newton-type Method for the Optimization of Finite Sums. International Conference on Machine Learning (ICML), 2016.

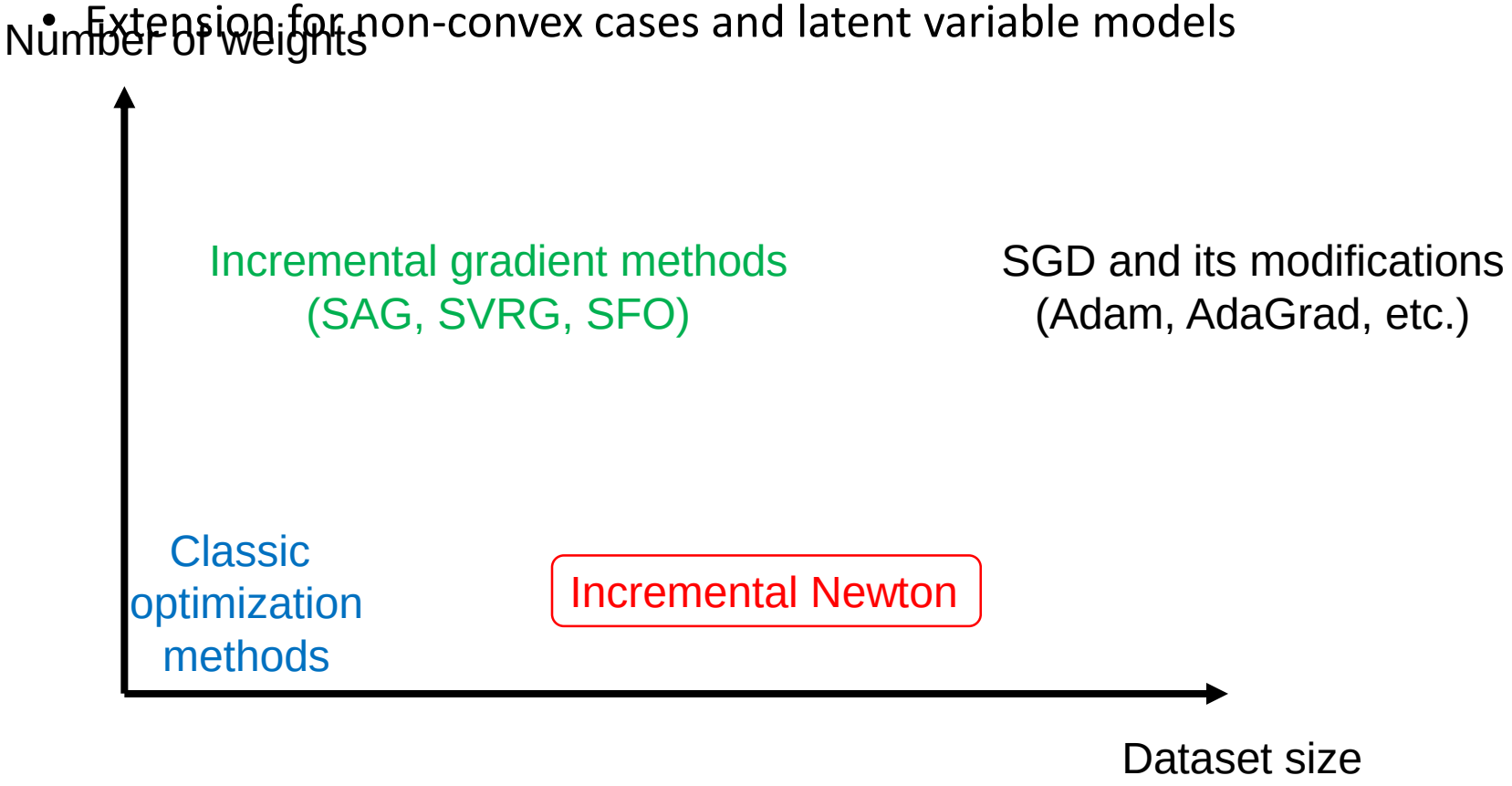
Incremental Newton

L2-reg	*alpha (n=500 000, d=500)*				*mnist8m (n=8 100 000, d=784)*			
Res	NIM	SAG	Newton	LBFGS	NIM	SAG	Newton	LBFGS
10^{-1}	1.91s	**1.36s**	1.6m	4.01s	57.68s	**34.91s**	47.8m	1.1m
10^{-2}	13.37s	**6.72s**	2.6m	17.68s	**1.6m**	2.1m	1.4h	5.2m
10^{-3}	28.56s	**17.73s**	3.0m	37.70s	**3.2m**	3.9m	-	22.9m
10^{-4}	36.65s	**36.04s**	3.4m	58.35s	16.7m	**7.1m**	-	1.6h
10^{-5}	**46.66s**	1.0m	3.6m	1.4m	**26.7m**	1.0h	-	-
10^{-6}	**53.92s**	1.5m	4.0m	1.9m	**33.5m**	-	-	-
10^{-7}	**57.63s**	2.0m	4.0m	2.4m	**40.1m**	-	-	-
10^{-8}	**1.0m**	2.7m	4.1m	2.8m	**46.0m**	-	-	-
10^{-9}	**1.1m**	3.5m	4.3m	3.2m	**49.6m**	-	-	-
10^{-10}	**1.2m**	4.3m	4.7m	3.4m	**53.3m**	-	-	-

ℓ_2 regularized logistic regression

Areas of application

- Faster training algorithms
- Better optimum localization
- Extension for non-convex cases and latent variable models



References

- A. Novikov, D. Podoprikhin, A. Osokin, D. Vetrov. Tensorizing Neural Networks. In Advances in Neural Information Processing Systems 28 (NIPS). 2015.
- S. Bartunov, D. Kondrashkin, A. Osokin, D. Vetrov. Breaking Sticks and Ambiguities with Adaptive Skip-gram. International Conference on Artificial Intelligence and Statistics (AISTATS), 2016
- M. Figurnov, D. Vetrov, P. Kohli. PerforatedCNNs: Acceleration through Elimination of Redundant Convolutions. International Conference on Learning Representations (ICLR) 2016
- A. Rodomanov, D. Kropotov. A Superlinearly-Convergent Proximal Newton-type Method for the Optimization of Finite Sums. International Conference on Machine Learning (ICML), 2016.