Mathematics of Big Data

Dmitry P. Vetrov

Head of Bayesian Methods research group, http://bayesgroup.ru

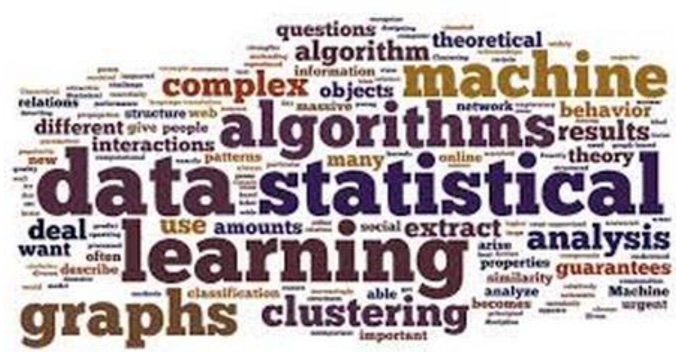
Associate professor in Skoltech, HSE

Outline

- Methods for large-scale data processing
  - Bayesian inference
  - Latent variable models
  - Stochastic optimization
  - Deep learning
  - Tensor decompositions
- Adagram model
  - Word2vec skipgram
  - Multi-sense extension
  - Scalable learning algorithm
  - Experiments

What is machine learning?

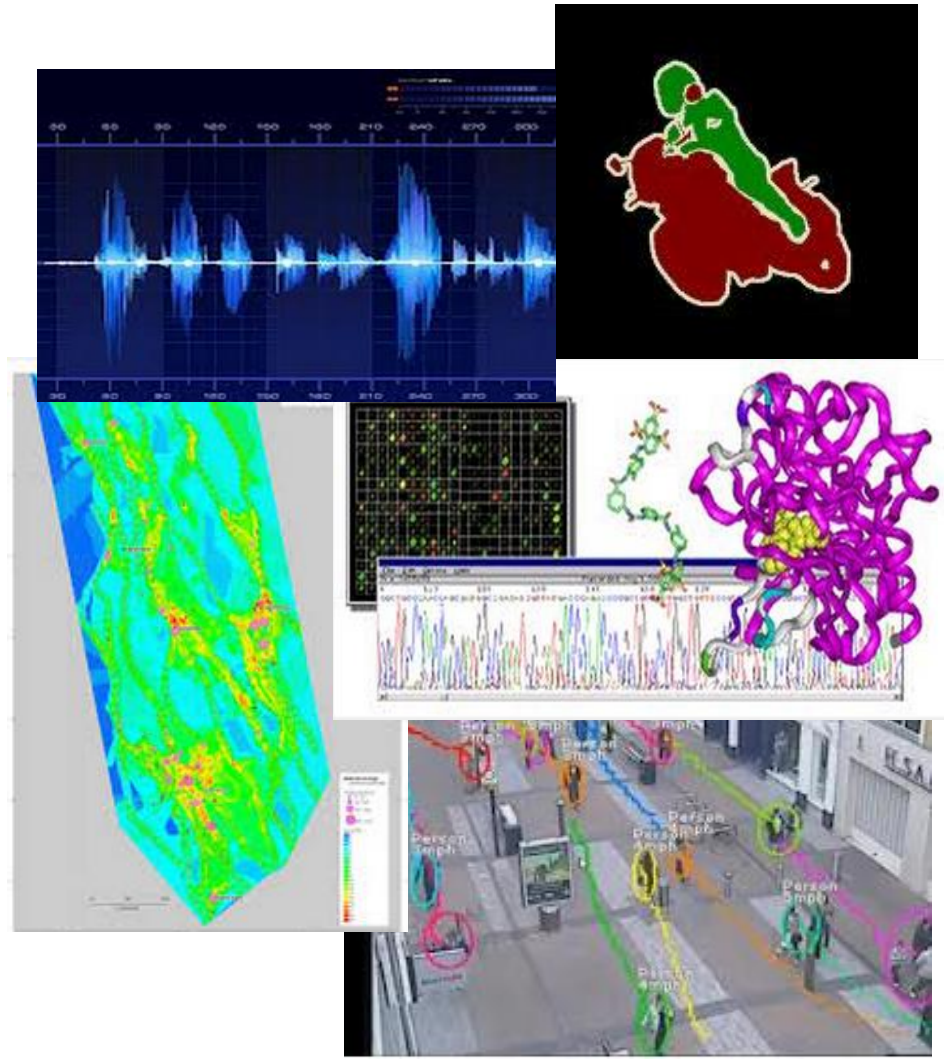
- ML tries to find regularities within the data
- Data is a set of objects (users, images, signals, RNAs, chemical compounds, credit histories, etc.)
- Each object is described by a set of observed variables X and a set of hidden (latent) variables T
- It is assumed that the values of hidden variables are hard to get and we have only limited number of objects with known hidden variables, so-called training set
- 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

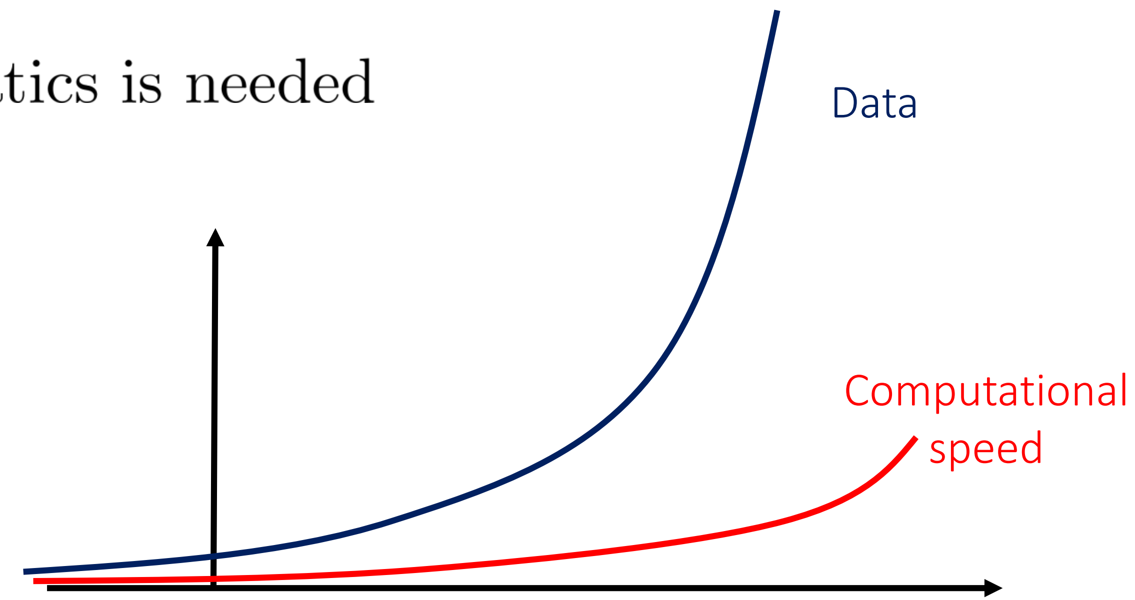
With the spread of information technologies ML has been used in more and more domains

- Computer vision
- Speech recognition
- Credit scoring
- Mineral depostis search
- Bioinformatics
- Web-search
- Recommender services
- Behaviour analysis
- Social studies
- etc.



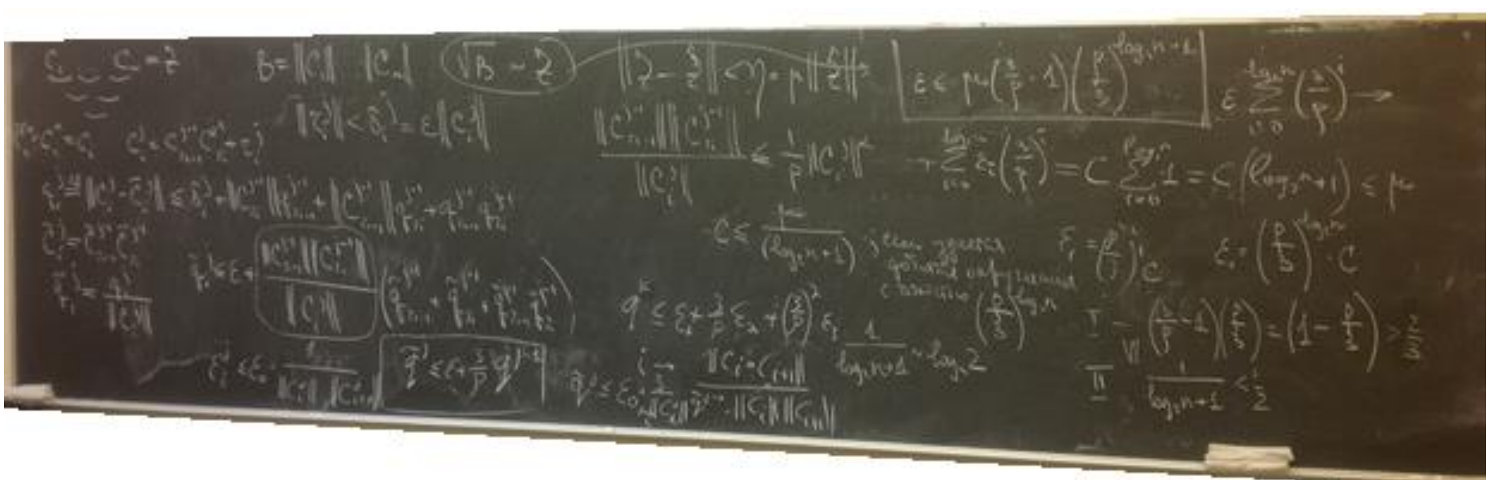
Entering the Age of Big Data

- The amount of data available for analysis grows several orders faster than the computational resources
- Difficult even to keep it not saying about processing
- Old methods simply do not work
- New mathematics is needed



First Steps towards Mathematics of Big Data

- Bayesian Inference & Graphical Models (Koller09)
- Latent Variable Modeling (Bishop06)
- Deep Learning (Bengio14)
- Tensor Calculus & Decomposition Techniques (No good book published yet)
- Stochastic Optimization (No good book published yet)

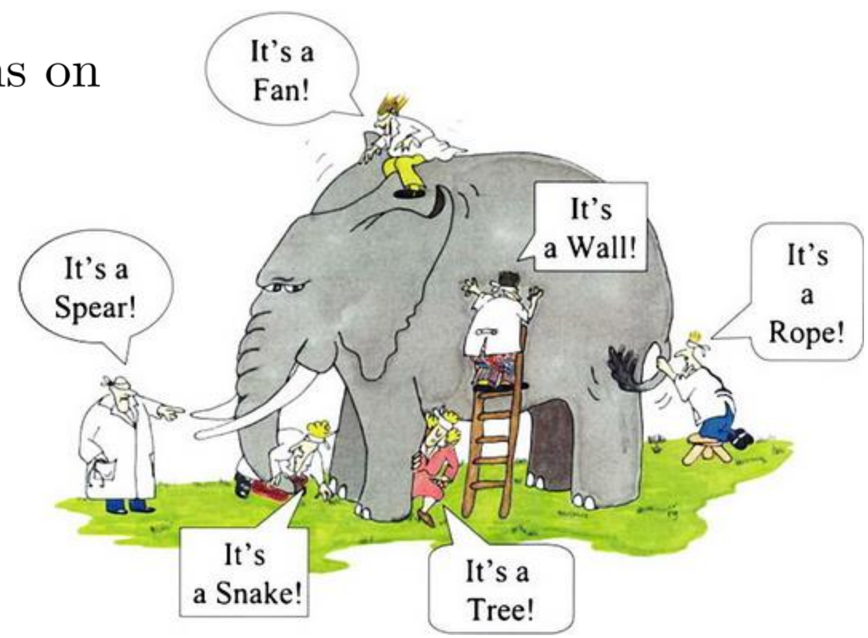


Bayesian framework

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

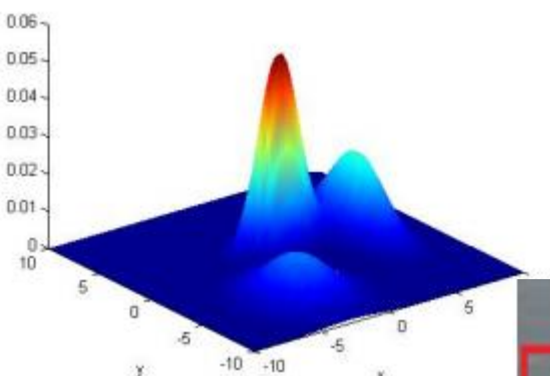
$$\texttt{Posterior} = \frac{\texttt{Likelihood} \times \texttt{Prior}}{\texttt{Evidence}}, \quad p(\theta|X) = \frac{p(X|\theta)p(\theta)}{\int p(X|\theta)p(\theta)d\theta}$$

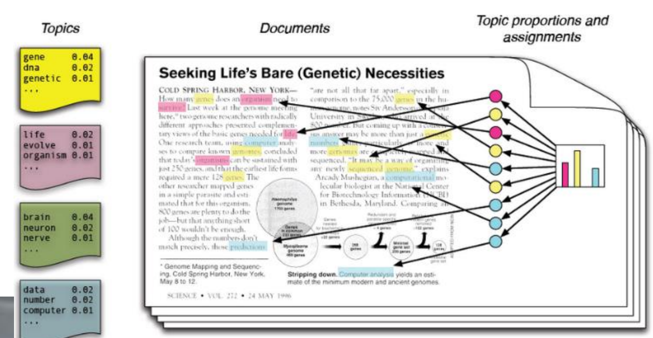
- Posteriors may serve as new priors, i.e. may combine multiple models!
- **BigData:** we can process data streams on an update-and-forget basis
- Support distributed processing



Incomplete data

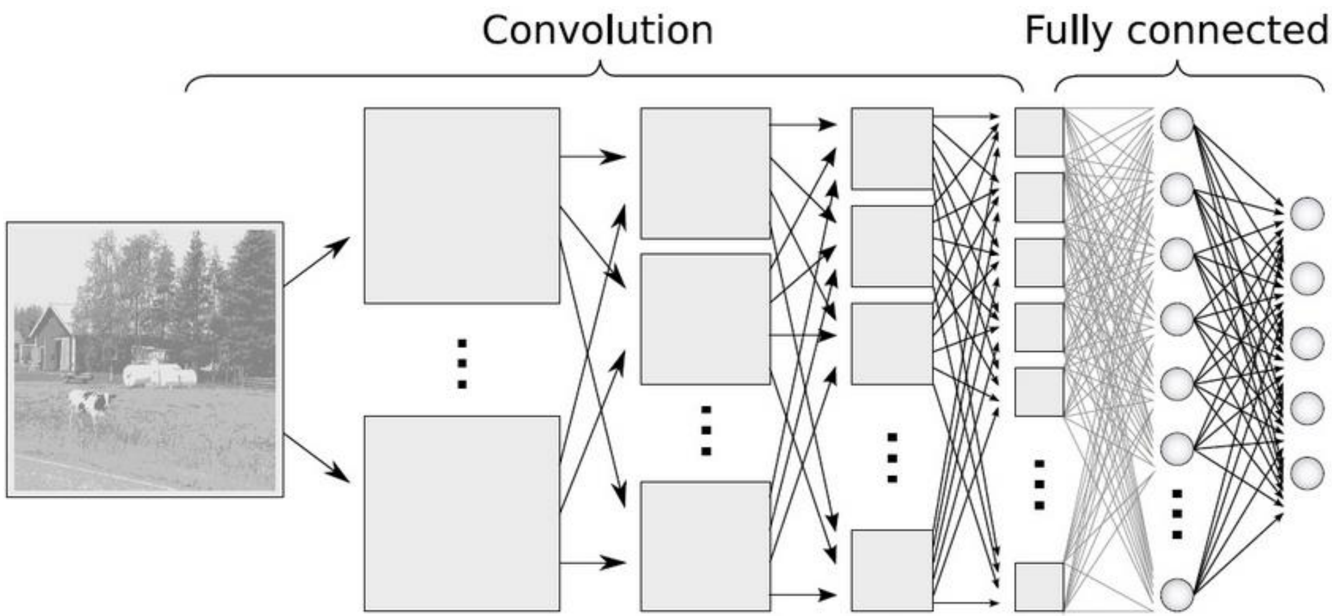
- It is often the case when for training objects we know only the subset of their possible values of hidden variables
- This is an example of so-called **weakly-labeled data**
- Need to build ML models with **latent variables**
- With huge datasets learning from incomplete data is almost as effective





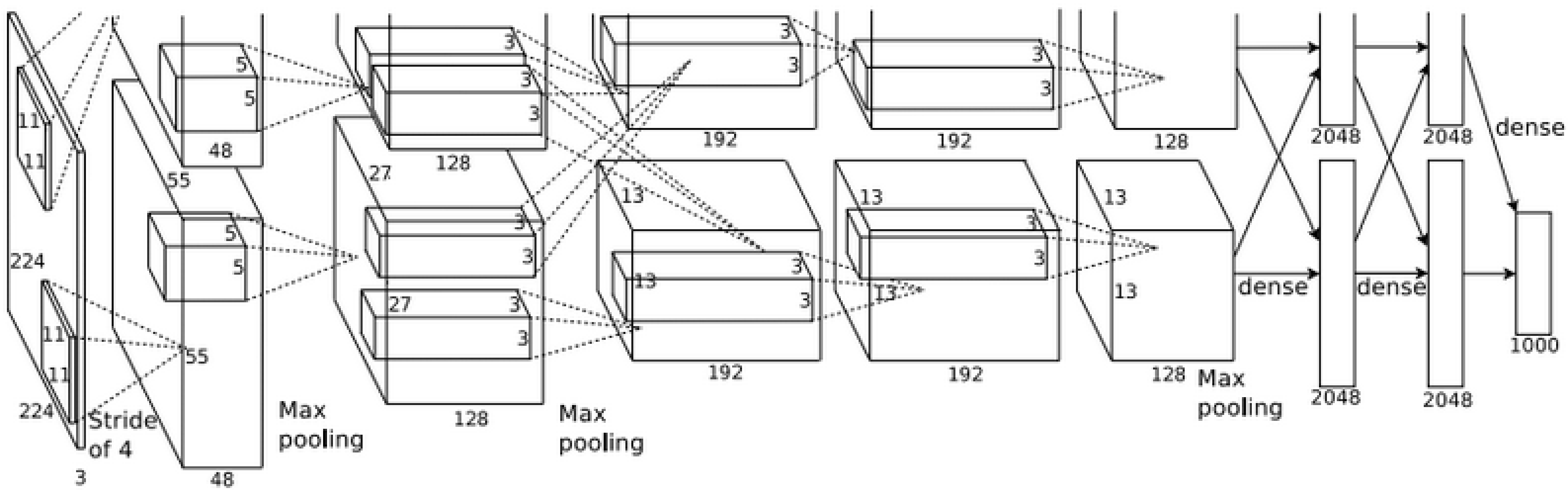
Deep learning

- New generation of neural networks (Deep Boltzmann machines, convolutional nets, auto-encoders) have achieved state-of-art performance almost in **all** ML problems where there was enough training data (X_{tr}, T_{tr}).
- For the first time computer shows the signs of understanding the **sense** of data
- Very large datasets are needed (Big data effect)



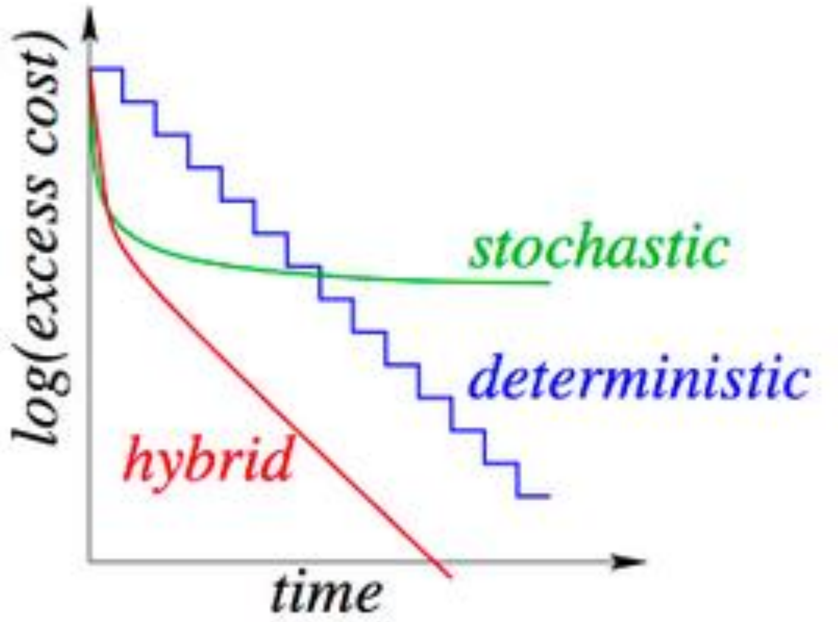
Secret of Success of Neural Nets

- Pretraining algorithms allow to find good starting point for back-propagation
- Processing huge datasets makes training procedure robust
- Less subjected to overfitting when dealing with huge datasets
- Efficient GPU implementations allow to construct very deep networks



Stochastic Optimization

- Allows to optimize function FASTER than the time needed to compute it at a single point!
- Deals with functions of billions of objects
- Instead of working with function we use its unbiased estimate which can be millions times faster to compute



Stochastic optimization

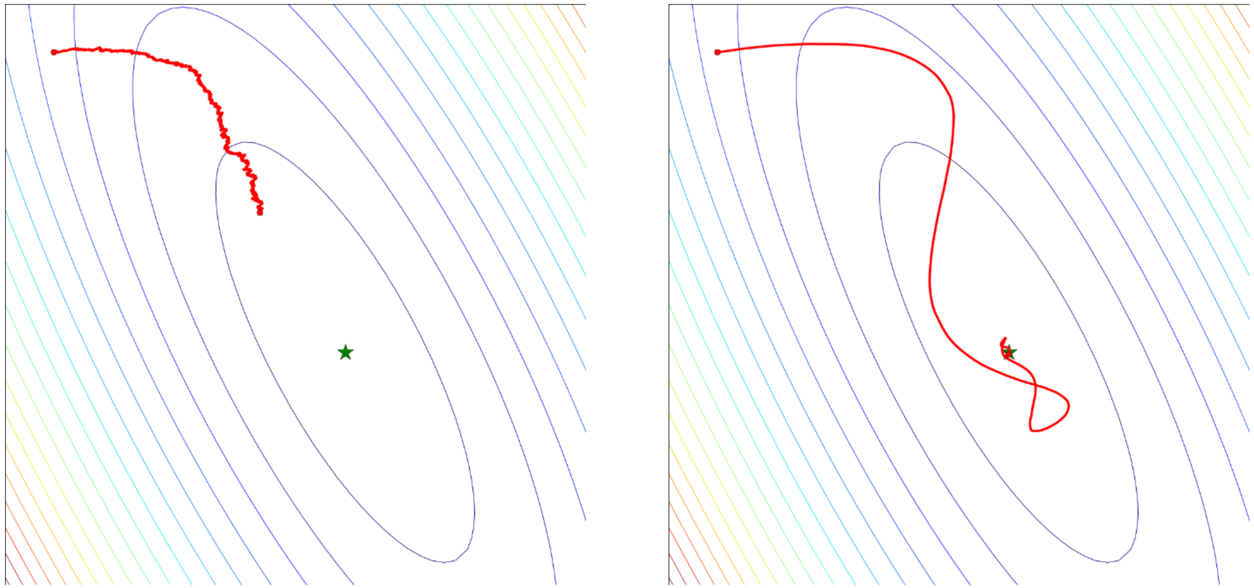
- Extremely efficient technique for large-scale optimization of $f(x)$
- Uses unbiased estimates $g(x)$ instead of true gradients $\nabla f(x)$
- (Robbins, Monro, 1951) If $f(x)$ is differentiable, $\mathbb{E}g(x) = \nabla f(x)$, $\forall x$, and $\sum_k \alpha_k = +\infty$, $\sum_k \alpha_k^2 < +\infty$,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

Tensor perspective

- 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:

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

Putting things together

- **NeuroBayes (2014-2015)** Uses deep networks with stochastic optimization for performing Bayesian inference in very complex probabilistic models
- **TensorNet (2015)** Combines stochastic optimization with tensor decomposition and deep learning

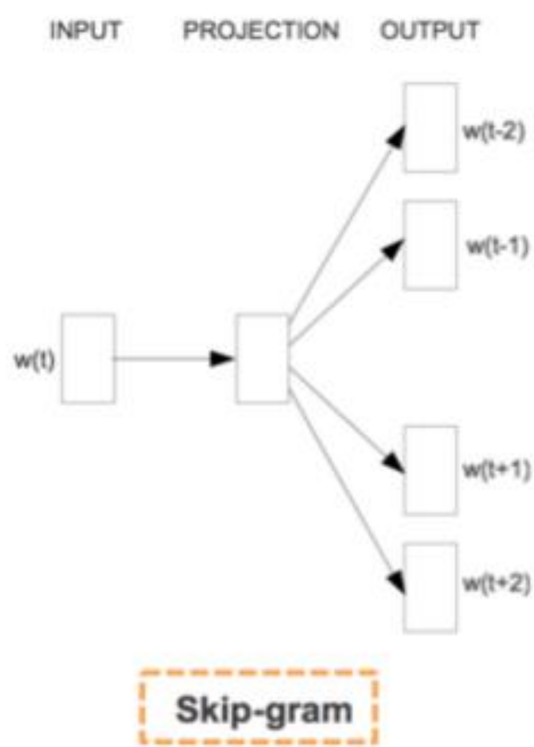
This was written by a computer algorithm

This was writtn by a computer algorithm

This was written by acomputer 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)
…	…

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

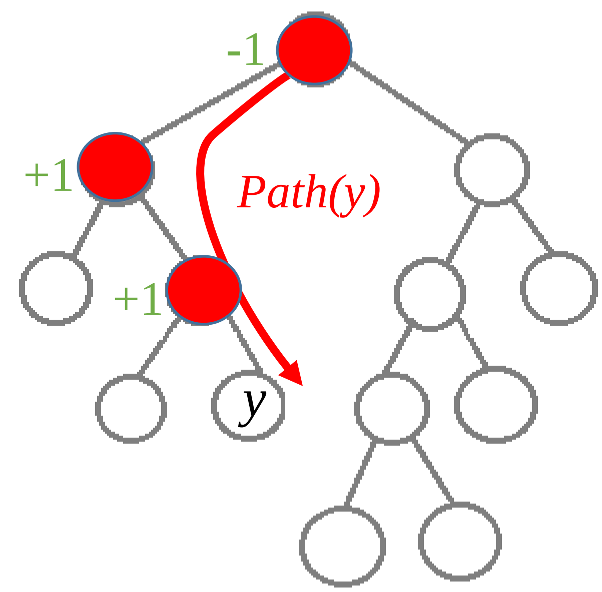
$$p(Y|X) \to \max_{\{In, Out\}}$$

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 $O(\log V)$

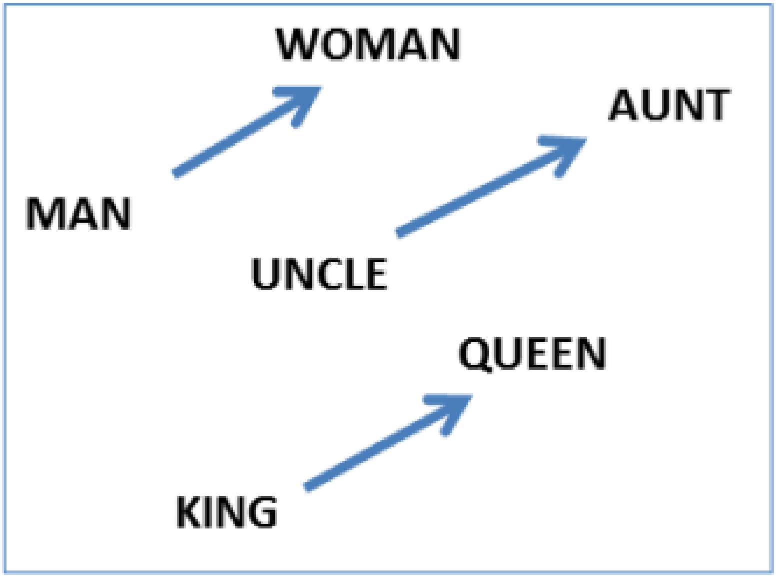
Semantic properties of representations

- 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

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

$In(\texttt{'ABBA'})$

$In(\texttt{'Euston'})$

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

$In(\texttt{'London'})$ $In(\texttt{'Station'})$

$In(\texttt{'Napoleon'})$

$In(\texttt{'Austerlitz'})$

$In(\texttt{'Battle'})$

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
```

$$\texttt{Waterloo - ?} \quad \begin{cases} \texttt{Station} & \texttt{0.76} \\ \texttt{Battle} & \texttt{0.21} \\ \texttt{Song} & \texttt{0.03} \end{cases}$$

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. $\{In(x, z), Out(c)\}$

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

  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

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

- 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

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

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

Does not depend on *{In, Out}*

- Its unbiased estimate is simply

We know from E-step

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

- 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