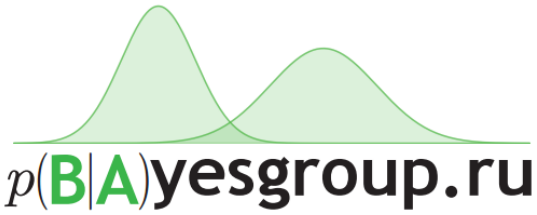
Word-sense Induction using Latent Variable Skip-gram

Dmitry Vetrov

Associate professor in Skoltech and HSE

Head of Bayesian methods research group

http://bayesgroup.ru

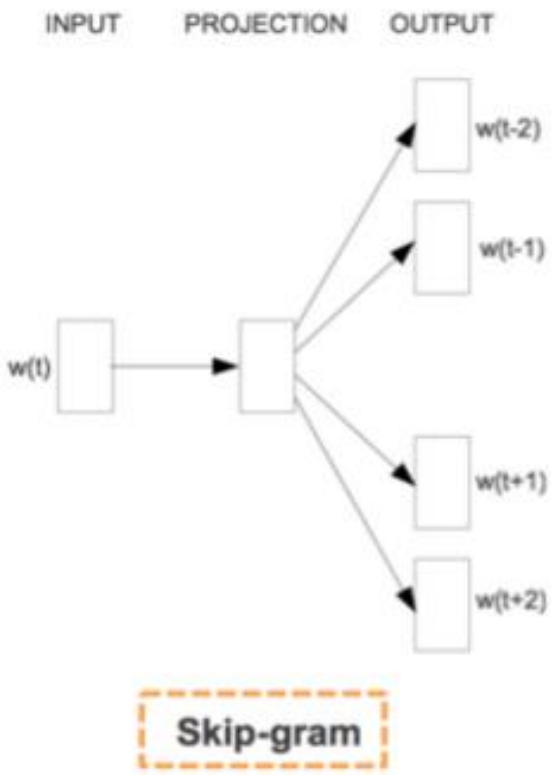


Outline

- Word2vec model
- Latent variable modeling
- Adaptive skip-gram (AdaGram)
- Stochastic optimization
- Large-scale training in AdaGram

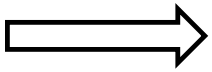
Word2vec model (Mikolov2013)

- 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:

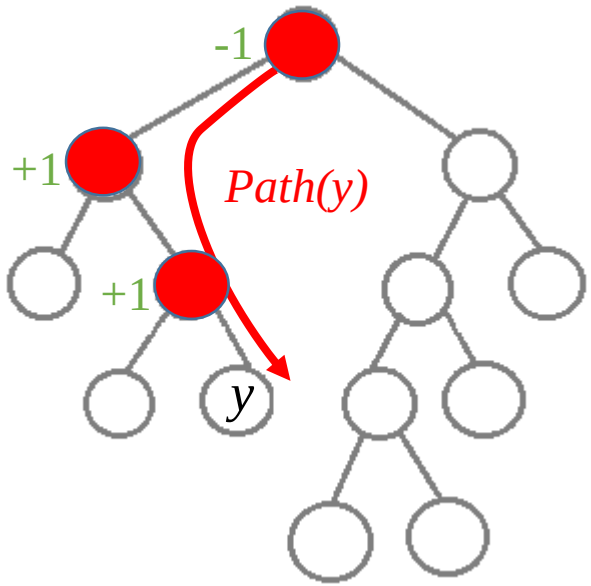
$$d_{c,y} = \begin{cases} +1 & y \text{ is in right subtree} \\ -1 & y \text{ is in left subtree} \end{cases}$$

- 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 $O(\log V)$



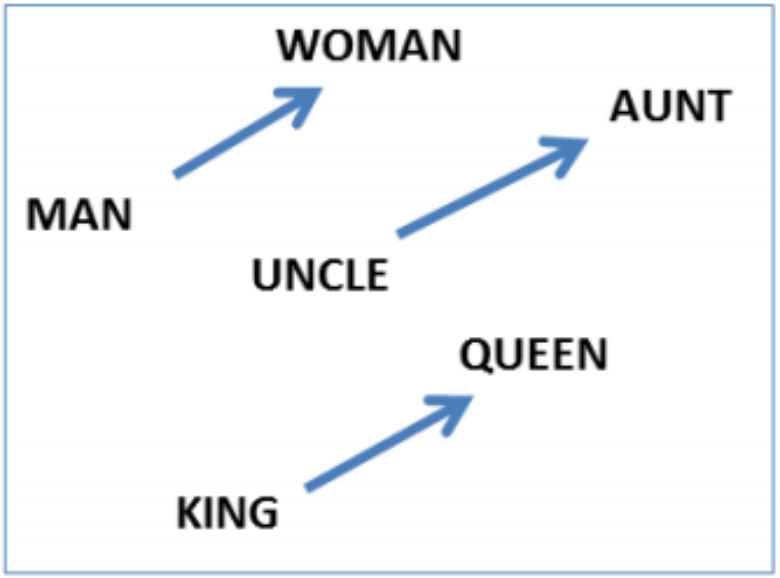
Semantic properties of representations

- 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?

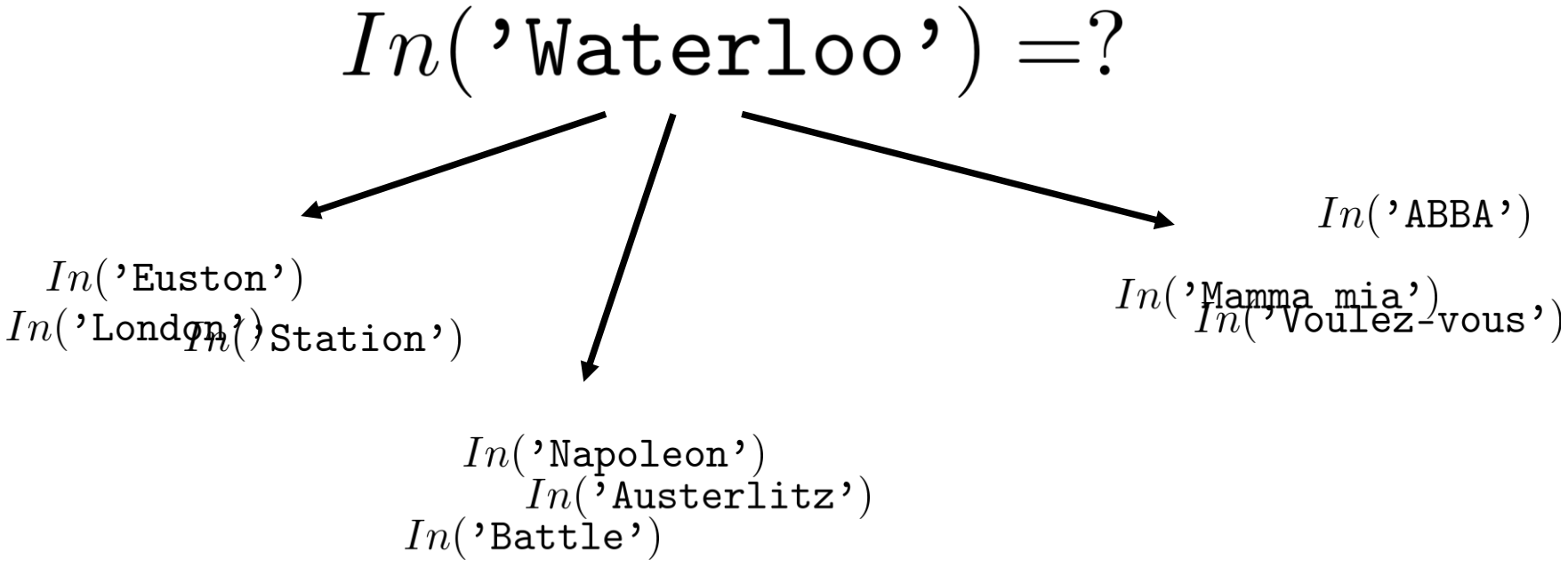


Word ambiguity

- 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

Latent variable modeling: example

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

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

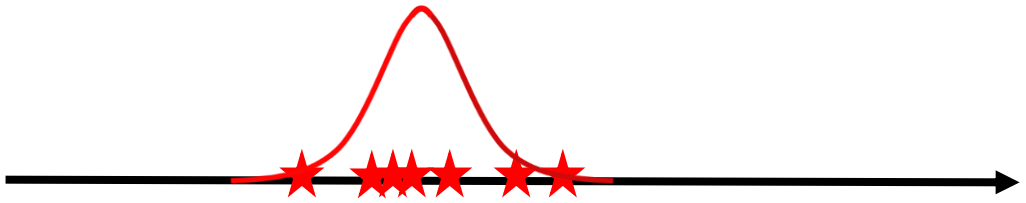
- We need to estimate its parameters μ and σ

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



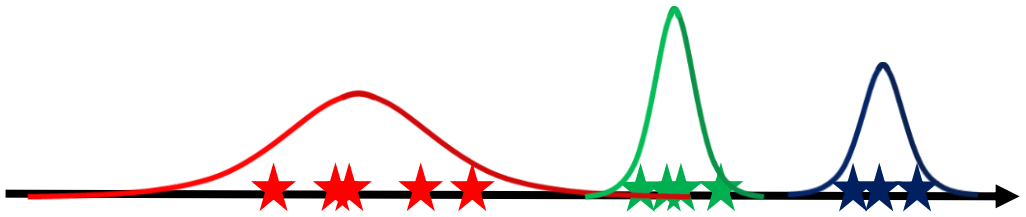
- Solution is simple: we estimate sample mean and variance

Latent variable modeling: example

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

Latent variable modeling: example

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



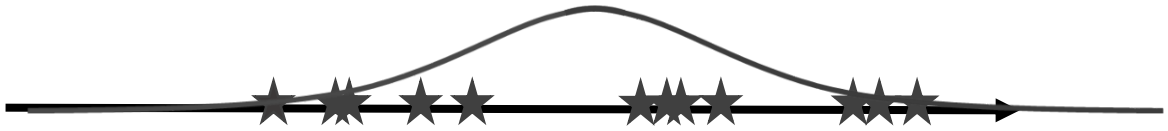
- The problem is as easy if we know what objects were generated from each gaussian

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

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

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

Waterloo - ?   { Station  0.76
               { 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

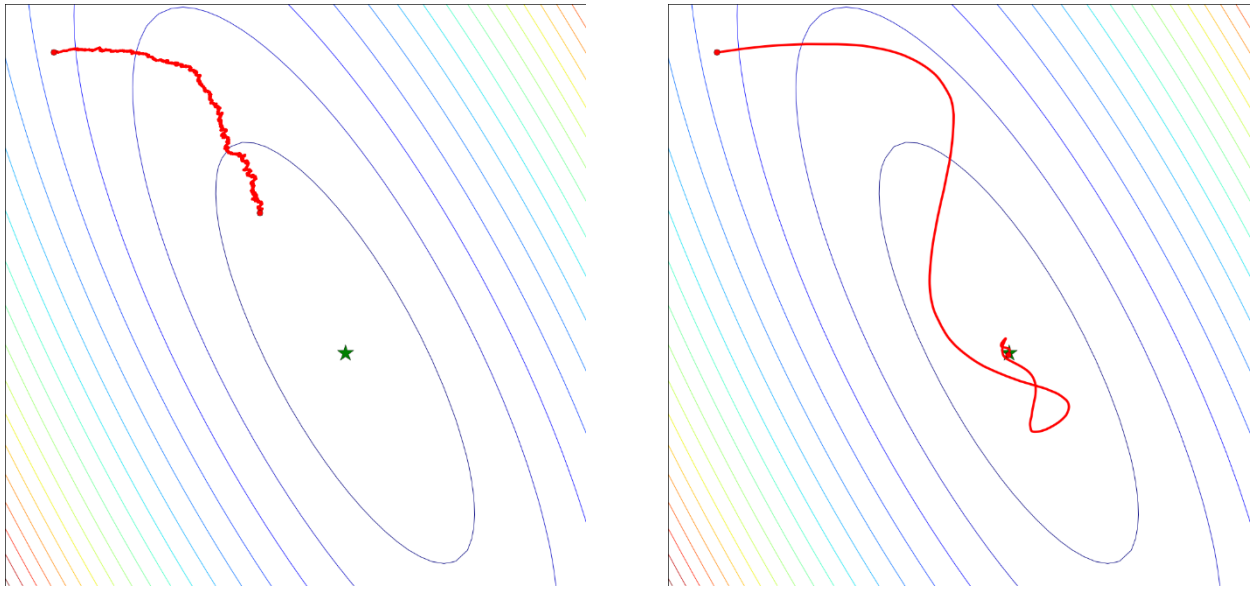
$$x_{k+1} = x_k + \alpha_k g(x_k)$$

  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, IN, 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 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