Word manifolds

John Goldsmith

University of Chicago

July 15, 2015

Goals

Goals

- Visualize the global structure of a language

Goals

Goals

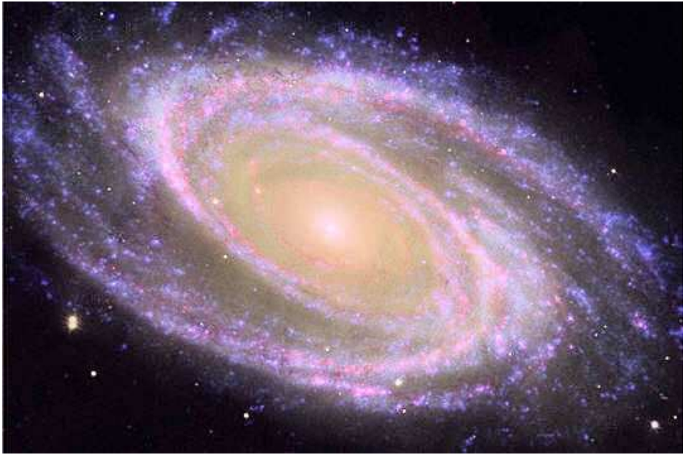
- Visualize the global structure of a language
- Solve a technical problem in the unsupervised learning of morphology (past tenses of English verbs)

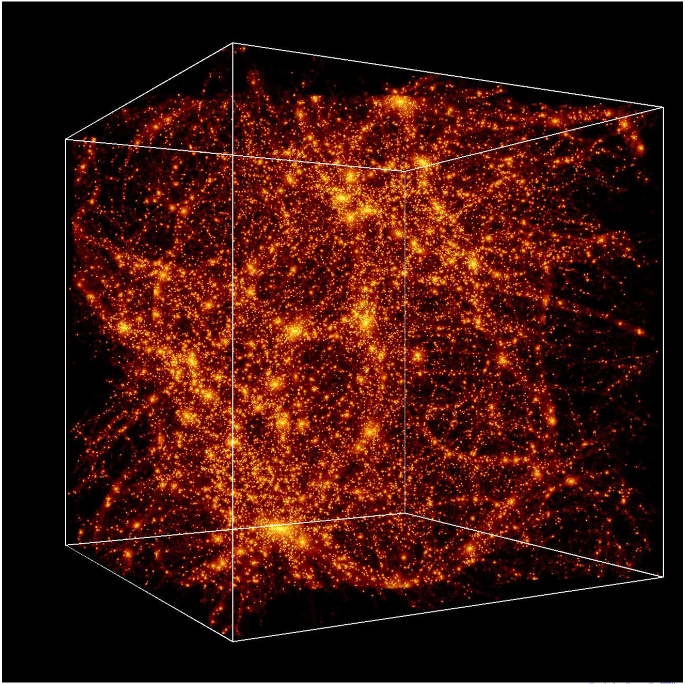
Goals

Goals

- Visualize the global structure of a language
- Solve a technical problem in the unsupervised learning of morphology (past tenses of English verbs)
- Develop a language-independent method


16





The algorithm is in three steps:

Algorithm

1. Compare all pairs of words to see which words *agree* on the word that precedes and follows it. *the* and *my* will agree a lot.

The algorithm is in three steps:

Algorithm

1. Compare all pairs of words to see which words *agree* on the word that precedes and follows it. *the* and *my* will agree a lot.
2. Turn this abstract *graph* into something in a geometric space, so we can talk about distances.

The algorithm is in three steps:

Algorithm

1. Compare all pairs of words to see which words *agree* on the word that precedes and follows it. *the* and *my* will agree a lot.
2. Turn this abstract *graph* into something in a geometric space, so we can talk about distances.
3. In that geometric space of dimension 10, ask each word to find out what the 6 closest words to it are. Make a graph out of those edges.

   The graph $\mathcal{S}$ can be directly viewed, using data visualization tools such as Gephi, and various clustering techniques can be applied to it as well.

The algorithm is in three steps:

Algorithm

1. Determine similarity between all pairs of words, based on a comparison of word-context, and the creation of a graph $\mathcal{C}$ whose edge-weights is determined directly by those similarities.
   Every pair of words (w_1, w_2) calculates how many contexts they share in common.

The algorithm is in three steps:

Algorithm

1. Determine similarity between all pairs of words, based on a comparison of word-context, and the creation of a graph $\mathcal{C}$ whose edge-weights is determined directly by those similarities.
2. Second, the computation of the K most significant eigenvectors of the normalized Laplacian of graph $\mathcal{C}$, and the calculation of the coordinates of each of the words in R^k based on these eigenvectors (where K is 10. Why 10? Why not?).

The algorithm is in three steps:

Algorithm

1. Determine similarity between all pairs of words, based on a comparison of word-context, and the creation of a graph $\mathcal{C}$ whose edge-weights is determined directly by those similarities.
2. Second, the computation of the K most significant eigenvectors of the normalized Laplacian of graph $\mathcal{C}$, and the calculation of the coordinates of each of the words in R^k based on these eigenvectors (where K is 10. Why 10? Why not?).
3. Third, calculation of a new distance $d(.,.)$ between all pairs of words, viewing the words as points in R^K; a new graph $\mathcal{S}$ is constructed, whose edge weights are directly based on distance in R^K.

   The graph $\mathcal{S}$ can be directly viewed, using data visualization tools such as Gephi, and various clustering techniques can be applied to it as well.

First step: 1

Property			
W(-1)	=	w_j	the word to the immediately left of w is w_j;
W(1)	=	w_j	the word to the immediately right of w is w_j;
W(-2)	=	w_j	the word two words left of w is w_j; etc.
W(-2,-1)	=	(w_j,w_k)	W(-2)=w_j and W(-1)=w_k.
W(-1,1)	=	(w_j,w_k)	W(-1)=w_j and W(1)=w_k.

the									
even in —									
	a	his	their	its	an	this			
part of—									
	a	his	their	its	our	an	my	this	your
—way ,									
	a	his	their	its	my	this			
in—small									
	a	their	its	our	my	this			
spirit of—									
	a	his	its	our	my	this			
of all—									
	a	his	their	its	our	my	this		

would

that he —

could	can	should	must	might	may	will

—be taken

could	can	should	must	might	may	will

maybe I—

could	can	should	will	didn't	couldn't

he — get

could	can	should	might	may	didn't	couldn't

— be .

could	can	should	must	might	may	will

— be considered

could	can	should	must	might	will

— be ,

could	can	should	must	might	may

— be a

can	should	must	might	may	will

Step 2

Eigenvector number 1

	word	coordinate
0	world	-0.059
1	problem	-0.054
2	family	-0.054
3	car	-0.054
4	state	-0.053
5	same	-0.053
6	city	-0.052
7	way	-0.052
8	man	-0.052
9	church	-0.051
10	number	-0.051
11	house	-0.051
12	program	-0.050
13	day	-0.049
14	company	-0.049
985	had	0.094
986	as	0.096
987	is	0.100
988	at	0.103
989	was	0.104
990	with	0.104
991	a	0.105
992	that	0.108
993	on	0.110
994	and	0.114
995	for	0.115
996	of	0.123
997	the	0.125
998	to	0.142
999	in	0.148

Eigenvector number 2

	word	coordinate
0	the	-0.155
1	a	-0.129
2	his	-0.103
3	this	-0.086
4	it	-0.086
5	that	-0.084
6	to	-0.080
7	in	-0.079
8	their	-0.076
9	an	-0.074
10	he	-0.071
11	our	-0.070
12	its	-0.068
13	of	-0.067
14	for	-0.066
15	they	-0.065
985	bring	0.118
986	think	0.119
987	tell	0.131
988	say	0.132
989	go	0.134
990	know	0.141
991	give	0.145
992	find	0.161
993	see	0.166
994	do	0.174
995	make	0.177
996	take	0.179
997	get	0.182
998	be	0.190
999	have	0.202

Eigenvector number 3

	word	coordinate
0	would	-0.148
1	was	-0.142
2	could	-0.140
3	had	-0.131
4	is	-0.125
5	can	-0.123
6	has	-0.114
7	must	-0.110
8	may	-0.110
9	should	-0.105
10	might	-0.103
11	will	-0.100
12	did	-0.099
13	didn't	-0.089
14	were	-0.085
15	of	-0.078
985	it	0.107
986	get	0.108
987	its	0.108
988	see	0.111
989	take	0.112
990	them	0.112
991	him	0.119
992	make	0.122
993	be	0.135
994	their	0.136
995	this	0.143
996	her	0.147
997	his	0.171
998	a	0.185
999	the	0.238

Eigenvector number 4

0	of	-0.161
1	and	-0.156
2	in	-0.153
3	to	-0.137
4	for	-0.130
5	with	-0.119
6	is	-0.111
7	from	-0.109
8	by	-0.106
9	on	-0.100
10	into	-0.096
11	was	-0.088
12	at	-0.086
13	or	-0.083
14	are	-0.074
15	will	-0.072
16	would	-0.071

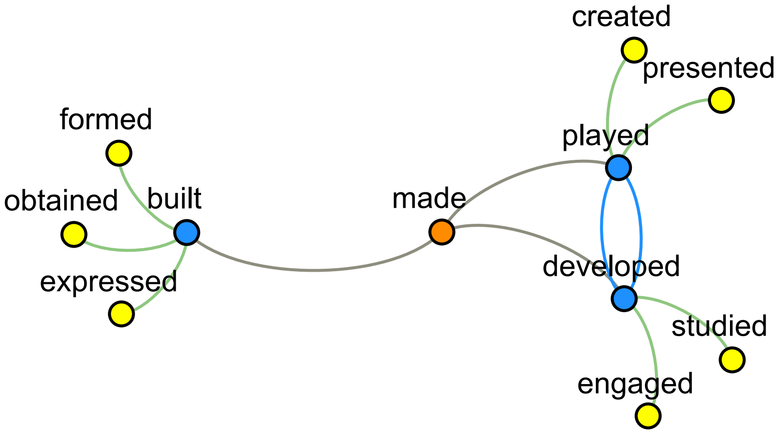
984	presented	0.096
985	sent	0.097
986	expected	0.098
987	able	0.099
988	obtained	0.100
989	said	0.102
990	called	0.105
991	held	0.107
992	asked	0.108
993	been	0.110
994	brought	0.110
995	told	0.113
996	given	0.120
997	done	0.140
998	made	0.142
999	taken	0.147

Eigenvector number 10

0	them	-0.131
1	him	-0.128
2	me	-0.103
3	himself	-0.103
4	years	-0.097
5	may	-0.095
6	God	-0.094
7	dollars	-0.093
8	can	-0.092
9	should	-0.089
10	out	-0.089
11	money	-0.088
12	must	-0.085
13	might	-0.082
14	time	-0.082
15	discrimination	-0.080
16	up	-0.076
17	courses	-0.075

984	took	0.066
985	Federal	0.066
986	Soviet	0.066
987	its	0.067
988	gave	0.067
989	San	0.068
990	Democratic	0.068
991	General	0.069
992	Hospital	0.069
993	saw	0.076
994	got	0.077
995	had	0.080
996	a	0.087
997	Highway	0.091
998	Health	0.094
999	the	0.113

'*made*' 3-neighbors and 2 generations



First step: 3

- Let V be the number of distinct word types in the language.
- Then there are in principle V features of the type W(-2,-1), and also of the type W(-1,1) and W(1,2).
- But the number of such features that are actually used is a small subset of the total number.
- For example, in an English-language encyclopedia composed of 888,000 distinct words, there were 1,689,000 distinct trigrams, of which 1,465,000 (nearly 87%) occur only once.

First step: 4

- We define $f(w_i, w_j)$ as the number of distinct features (using the contextual features just defined) shared by words w_i and w_j.
- It is natural to think of a graph $\mathcal{C}$in which the nodes are our words, and the edges are weighted by $f(w_i, w_j)$.
- The weight between two nodes indicates how many contexts they share, so all other things being equal, the stronger the weight of the edge between word A and word B, the more similar A and B are concerning their syntactic contexts.

Laplacian of a graph is a matrix

Laplacian of a graph

- The *laplacian* of a graph, such as $\mathcal{C}$, is defined as the matrix$M$ in which $M(i,j) = f(w_i, w_j)$ when $i \neq j$. We can think of the edges of the graph as *paths through which activation passes from one node to its neighboring nodes* on each of a number of successive iterations.

  If we think of the graph as a recipe for moving activation from one node to another, then the off-diagonal elements $m(i,j)$ show how much activation unit i sends to unit j
- For the diagonal elements, we first define $d(i)$ as $\sum_{k \neq i} M(i,k)$.
- $d(i)$ is the number of times word i appears in the corpus (you see that?).
- $M(i,i)$ is defined as $-1 \times d(i)$.
- $M(i,i)$ is the sum of the activation that unit

- We now have an initial similarity measure between words, but this similarity is not normalized for frequency: high frequency words will be much more similarity to others words that low frequency words will.
- Even if we normalize for frequency, though, the simplest ways of estimating similarity of distribution between two words on the basis of this data—using the cosine of the angle subtended by vectors pointing to each of the two words—is not as good as we might hope.

Second step: 1

- A number of researchers have explored the idea of taking a large set of data in a space of very high dimensionality, and finding a subspace of much lower dimensionality which is almost everywhere fairly close to the data.
- We've been especially influenced by the work of Partha Niyogi and Mikhail Belkin in the discussion that follows.

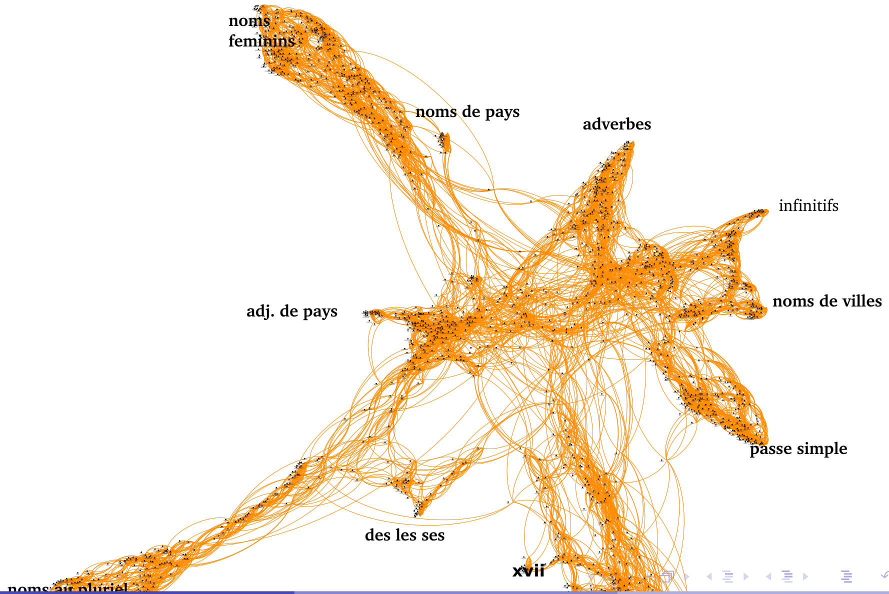
Second step: 2

- This means finding the eigenvectors of a normalized version of the graph laplacian.
- The normalized version of M, which we call N, is defined as follows: for all $i, N(i,i) = -1$, while for $(i,j), i \neq j$, we use the $d()$ function defined above to normalize, and say that $N(i,j) = \frac{M(i,j)}{\sqrt{d(i)d(j)}}$.

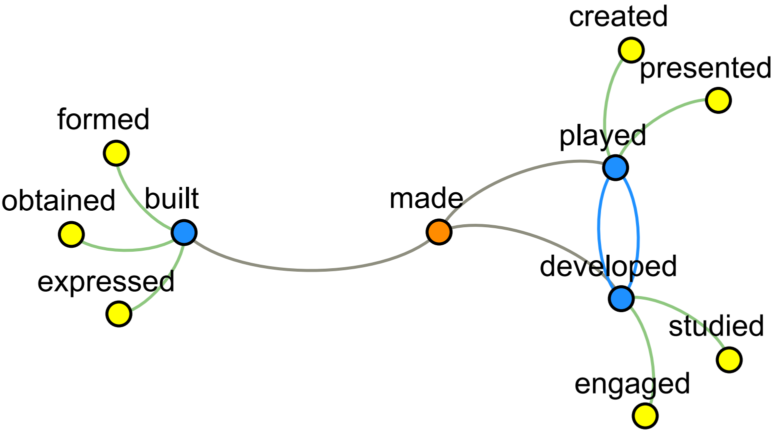
Second step and third step

We computed the first 11 eigenvectors of this normalized laplacian—those with the lowest eigenvalues, and used the 2nd through the 11th to give us coordinates for each word. Each word is thus associated with a point in R^{10}. We then select, for each word, the k closest words to it in this new space. These are the neighbors that we will explore below.

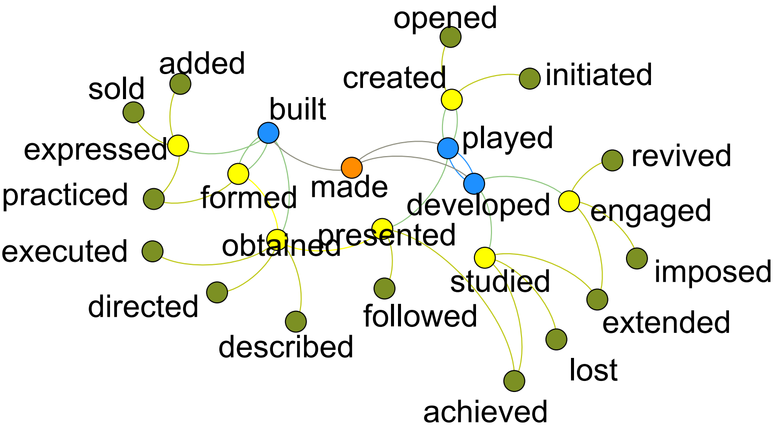
2,000 words of French



‘*made*’ 3-neighbors and 2 generations



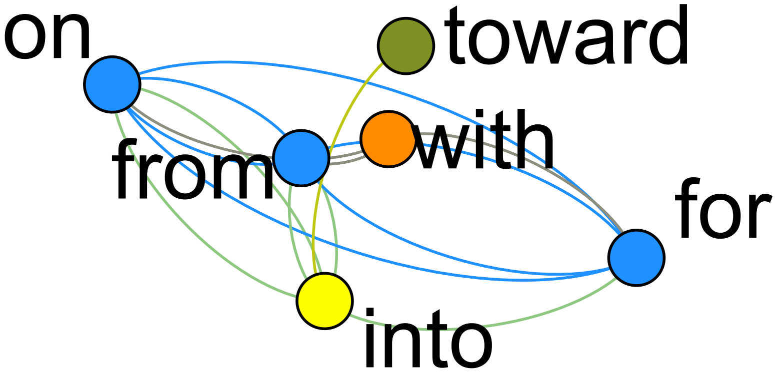
‘*made*’ 3 neighbors and 3 generations



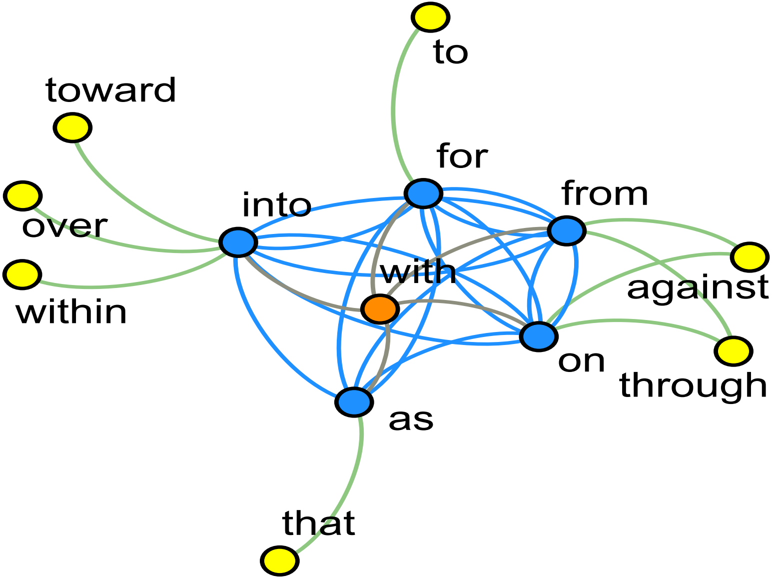
Help with learning morphology

jump	jumps	jumped	jumping	NULL-s-ed-ing
walk	walks	walked	walking	NULL-s-ed-ing
move	moves	moved	moving	e-es-ed-ing
build	builds	built	building	d-ds-t-ding
make	makes	??	making	NULL-s-ing

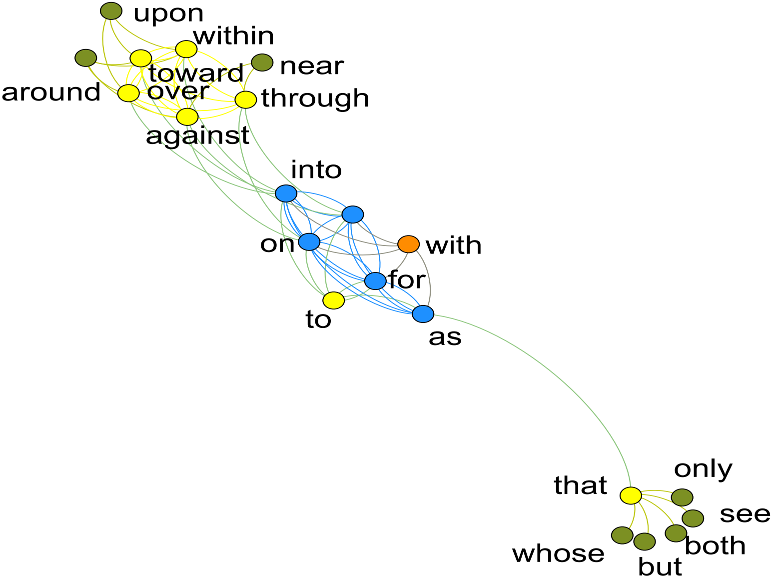
'*with*' 3 neighbors and 3 generations



‘*with*’ 5-neighbors and 2 generations



‘*with*’ 5-neighbors and 3 generations



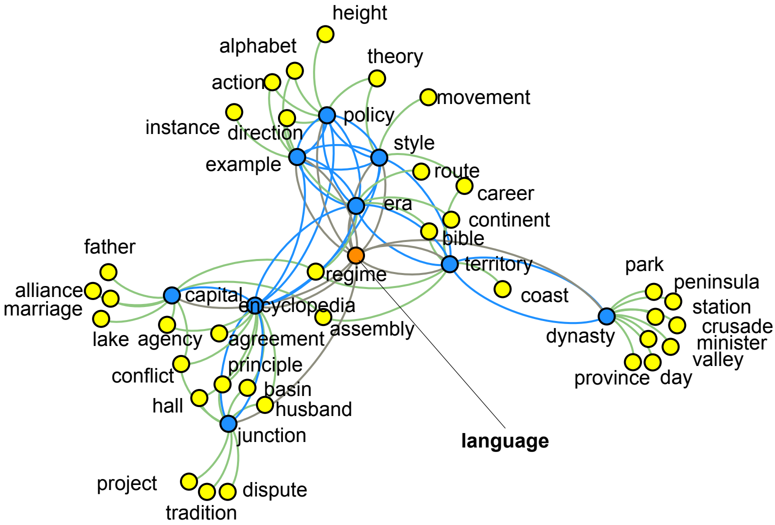
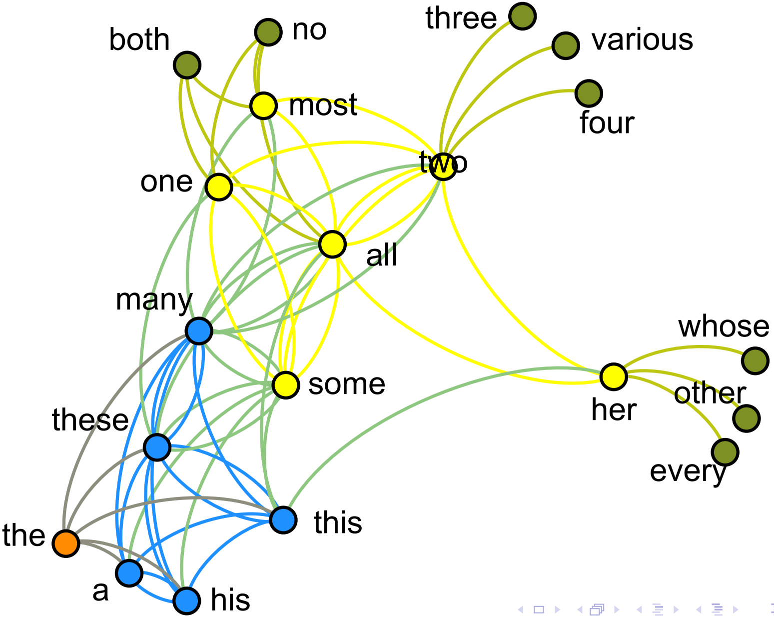
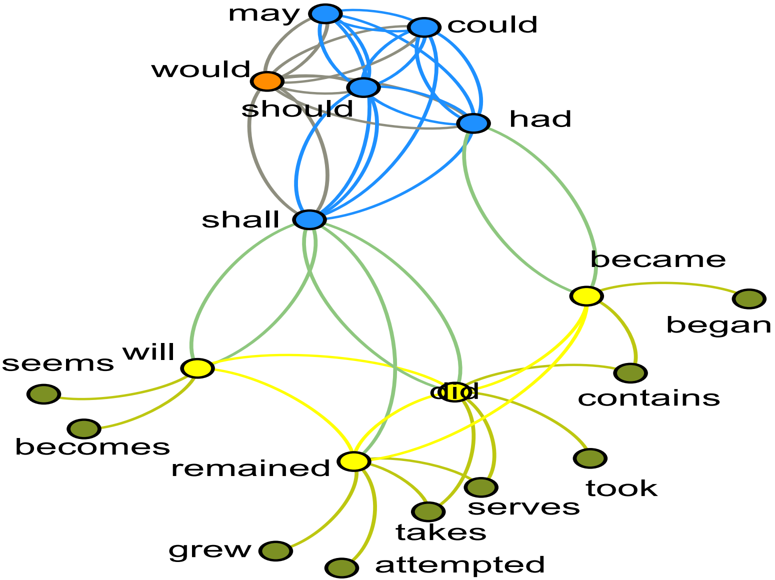


Figure : '*language*' 9 neighbors and 2 generations

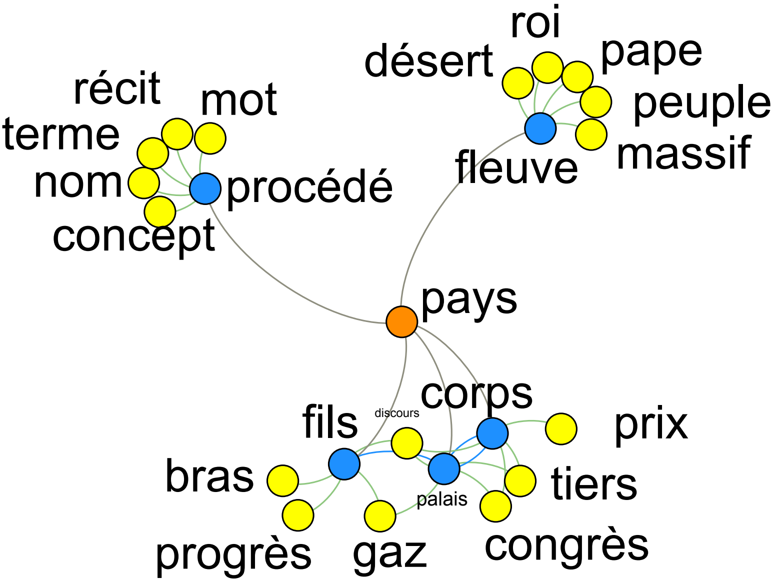
‘the’ 5 neighbors, 3 generations



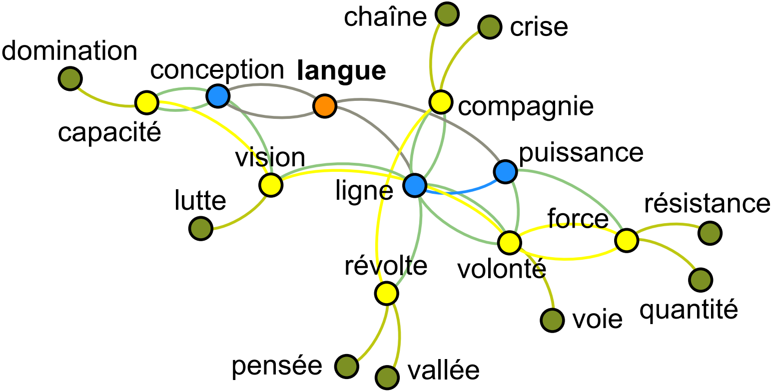
‘would’ 5 neighbors, 3 generations



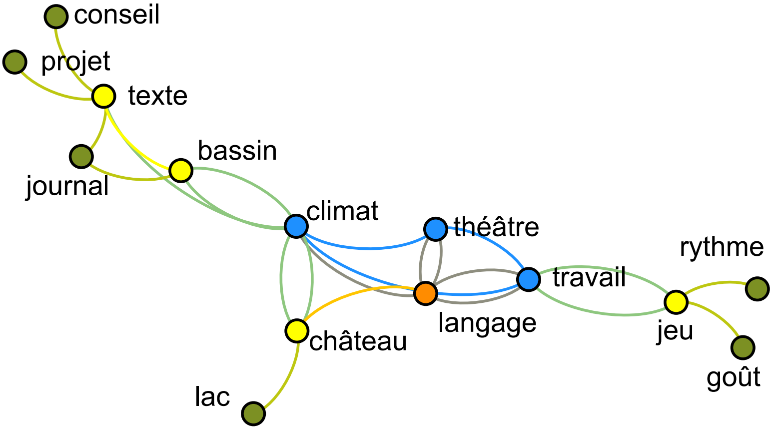
‘pays’ 5 neighbors and 2 generations



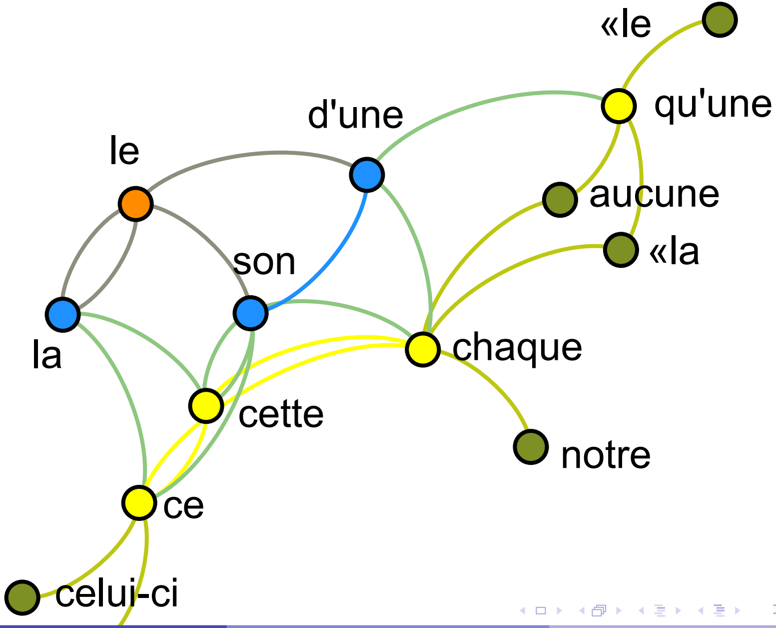
‘langue’ 3 neighbors and 3 generations



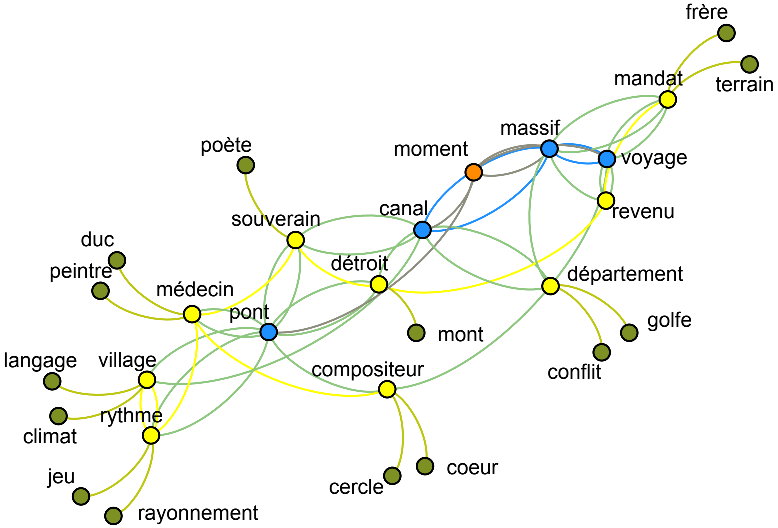
‘langage’ 3 neighbors and 3 generations



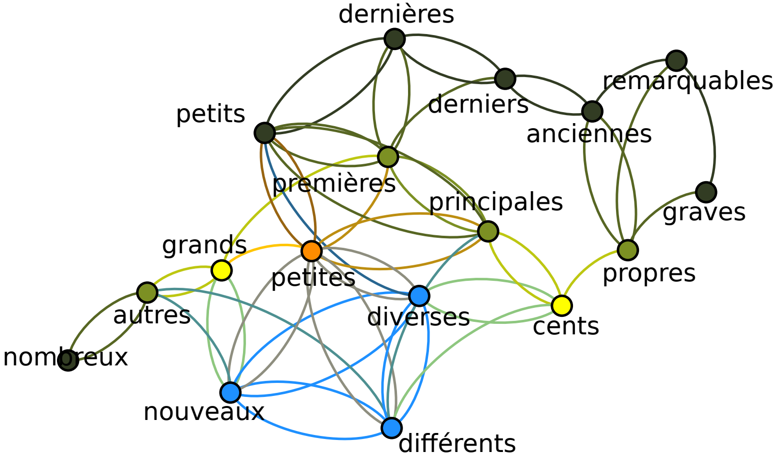
‘le’ 3 neighbors and 3 generations



‘moment’ 4 neighbors, 3 generations



petites, 3 neighbors



- There is a simple connection between minimizing the squared distance between nodes (though we haven't explained yet what kind of distance we are talking about now) of a weighted graph and the graph's Laplacian. We assume that no vertex is adjacent to itself.
- From a purely formal point of view, we could say that we are looking for a vector x in R^V which minimizes the expression, where W is the adjacency matrix of the graph, and $w_{i,j}$ are its entries:

$$\sum_{i,j} (x_i - x_j)^2 w_{i,j} \tag{1}$$

- Now we get to the kind of distance we're talking about: from the point of view of a projection, imagine that the entries $w_{i,j}$ in matrix W express the "similarity" between the i^{th} and the j^{th} element. We are looking for a single vector $\mathbf{x}$, then, which assigns very similar values to its i^{th} and j^{th} coordinate just in case those two coordinates correspond to elements that are "similar".
- We can think of that vector as representing a map from the graph's nodes to the real line; that is how we will think about it now, for the most part.

- We define a diagonal matrix D such that d_{ii} is the sum of the weights associated with edges adjacent to the i^{th} vertex: $d_{ii} = \sum_j w_{ij}$. Then
- $$\sum_i \sum_j (x_i - x_j)^2 w_{i,j} = \sum_i \sum_j (x_i^2 + x_j^2 - 2x_i x_j) w_{i,j} \qquad (2)$$

$$= \sum_i \sum_j (x_i^2) w_{i,j} + \sum_i \sum_j (x_j^2) w_{i,j} - 2 \sum_i \sum_j x_i x_j w_{i,j} \tag{3}$$

$$= \sum_i x_i^2 \sum_j w_{i,j} + \sum_j \sum_i (x_j^2) w_{i,j} - 2 \sum_i \sum_j x_i x_j w_{i,j} \tag{4}$$

$$= \sum_i x_i^2 d_{ii} + \sum_j x_j^2 \sum_i w_{i,j} - 2 \sum_i \sum_j x_i x_j w_{i,j} \tag{5}$$

$$= \sum_i x_i^2 d_{ii} + \sum_j x_j^2 d_{jj} - 2 \sum_i \sum_j x_i x_j w_{i,j} \tag{6}$$

- The first two terms are identical, and each are equal to $X^T D X$, while the third term is twice $X^T W X$. So

$$\sum_i \sum_j (x_i - x_j)^2 w_{i,j} = 2(X^T D X - X^T W X) = 2(X^T (D - W) X) \tag{7}$$

- It turns out that the matrix D-W has a name: it is the *laplacian* of the matrix W (or the graph of which W is the adjacency matrix). So we'll write $\mathcal{L} = D - W$. And there is a more natural way of writing $X^T (D - W) X$, which is to write $(X, \mathcal{L} X)$, which we can read as the inner product of the vector X and the vector $\mathcal{L} X$.

- If we restrict our attention to vectors of unit length, then this quantity $(X, \mathcal{L}X)$ is called the *Rayleigh quotient*. And we can find its maximal and minimal values along the eigenvectors of the laplacian. This is quite remarkable!
- Before we get to why that should be the case, we are going to squeeze the matrix so that its major diagonal consists of just 1's. We do this by defining a *normalized laplacian*, by dividing each entry l_{ij} of $\mathcal{L}$ by $\frac{1}{\sqrt{d_{ii}}\sqrt{d_{jj}}}$. We can write this:

$$\mathcal{L}' = D^{-\frac{1}{2}} \mathcal{L} D^{-\frac{1}{2}} \tag{8}$$

- If you are following this, you can see that $\mathcal{L}' = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$.
- The first term is the identity matrix; the second has 0s down the major diagonal, and is symmetric, and has only positive values; let's call it W', because it is the normalized form of W.
- And we have a better intuitive understanding of a matrix such as W', because it can naturally describe an ellipsoid: if we look at points x such that $(x, W'x)$ is a constant, we get an ellipsoid.
- Furthermore, $W'x$ is a vector normal to the surface of that ellipsoid at the point x.
- If we think about this geometrically, that means that $(x, W'x)$ will be a local maximum when x and $W'x$ point in the same direction —-which is the same thing as saying that x is an eigenvector of W'.

- So we look at the eigenvectors of W', or of $\mathcal{L}'$. If we look at the eigenvectors of W', we sort them by decreasing eigenvalue, so λ_0 is the largest eigenvalue, and its eigenvector simply reflects the overall frequencies of the graph.
- Note: sometimes people start number the eigenvalues at 1, and sometimes at 0, as I have done here.] The second eigenvalue, λ_1, is of great importance in graph theory. Here we care about its eigenvector, though, and we look at the values it assigns to each word.