Linguistic Infinitesimals

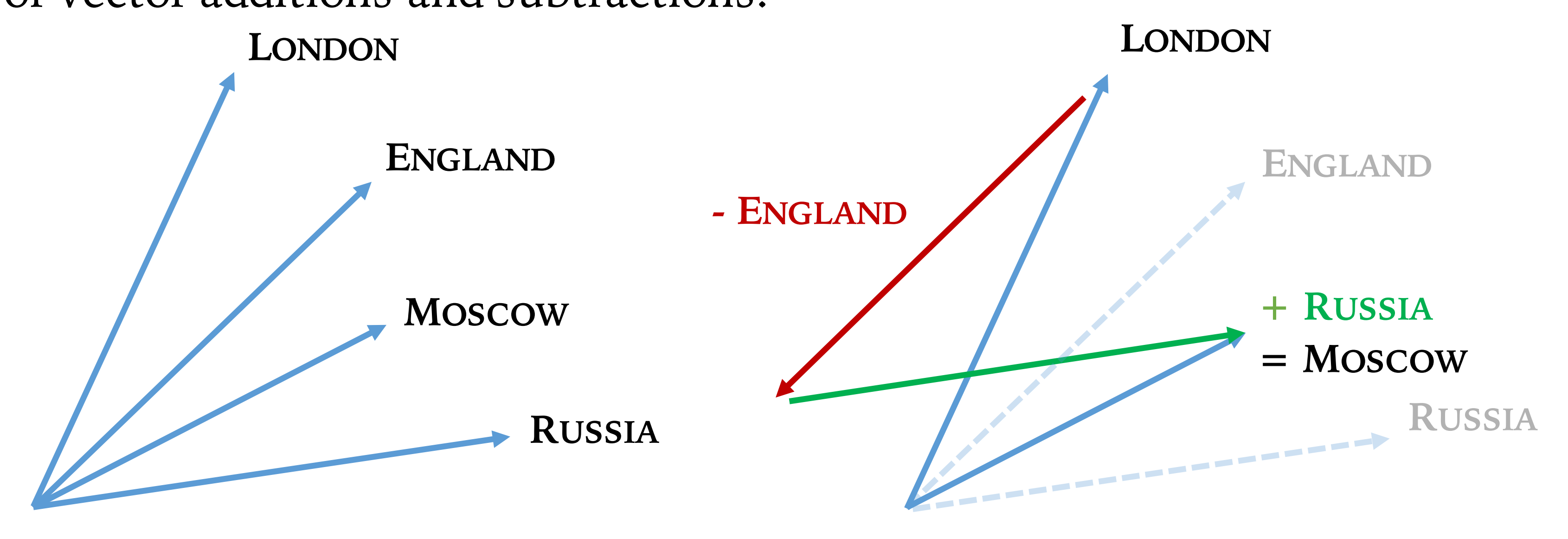
Jonathan Scott Enderle
University of Pennsylvania Libraries

Background and Introduction

Since their initial popularization by Miklov et. al. (2013)[1], word embedding models have become ubiquitous in computational linguistics. One of their most familiar properties is their ability to represent semantic analogies as vector space operations. For example, consider the analogy

London : England :: Moscow :: Russia

In a word embedding model, this analogy can be represented by a series of vector additions and subtractions:



The usefulness of this property is clear. But we understand very little about why word vectors work this way, despite their widespread adoption.[2] Why does language have a tractable geometric representation, not just in terms of semantic neighborhoods, but in terms of global semantic *orientations* that can be freely combined in meaningful ways?

To shed more light on this question, this poster sketches out a mathematical derivation of word vectors that do not rely on black-box optimization. It then shows how we can reinterpret that derivation as making a Derridean claim about iterability in natural language. Thus, although the word vectors generated by this derivation do not perform as well as those generated by state-of-the-art algorithms, their theoretical basis is rich enough to overlap with existing theoretical paradigms in the humanities in productive ways.

To complete this mathematical sketch, the following sections review a few concepts from the theory of algebraic data types and from multivariable calculus. Those concepts can be combined to construct an algorithm that generates word vectors directly from a corpus. They can also show how a combination of repetition and semantic change can produce nonlinear structures in language, in a mathematically well defined sense.

Algebraic Data Types

Algebraic data types will be familiar to anyone with basic programming experience, though often not by that name. They provide a way to combine existing data types. A simple example begins with a single bit. A bit is a variable with a boolean data type — a door that is open or closed. So suppose you have a bit that indicates whether your front door is closed. When it's open, the bit is 0. When it's closed, the bit is 1.

Now suppose you are lucky enough to have a back door, and you also have a bit that indicates whether it's open. The algebra of data types gives you two ways to join these two booleans into a single composite type. In the first, the two bits are combined into a pair (or a longer sequence) that shows the state of both doors at once. In the second, the two bits are superimposed, so that the state of only one door is visible at a time. The two tables below show all possible values that the composite types can take at a given time:

Pairwise Combination

Front	0	0	1	1
Back	0	1	0	1

Superimposition

Door	Front	Front	Back	Back
State	0	1	0	1

When there are only two doors, these don't look very different. But suppose you also have a side door:

Sequential Combination

Front	0	0	0	0	1	1	1	1
Back	0	0	1	1	0	0	1	1
Side	0	0	0	0	1	1	1	1

Superimposition

Door	Front	Front	Back	Back	Side	Side
State	0	1	0	1	0	1

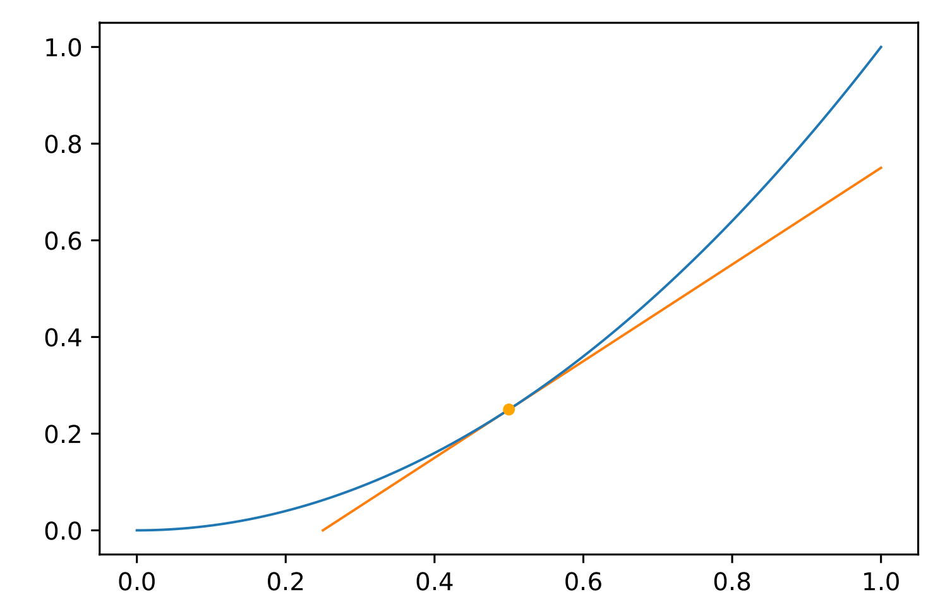
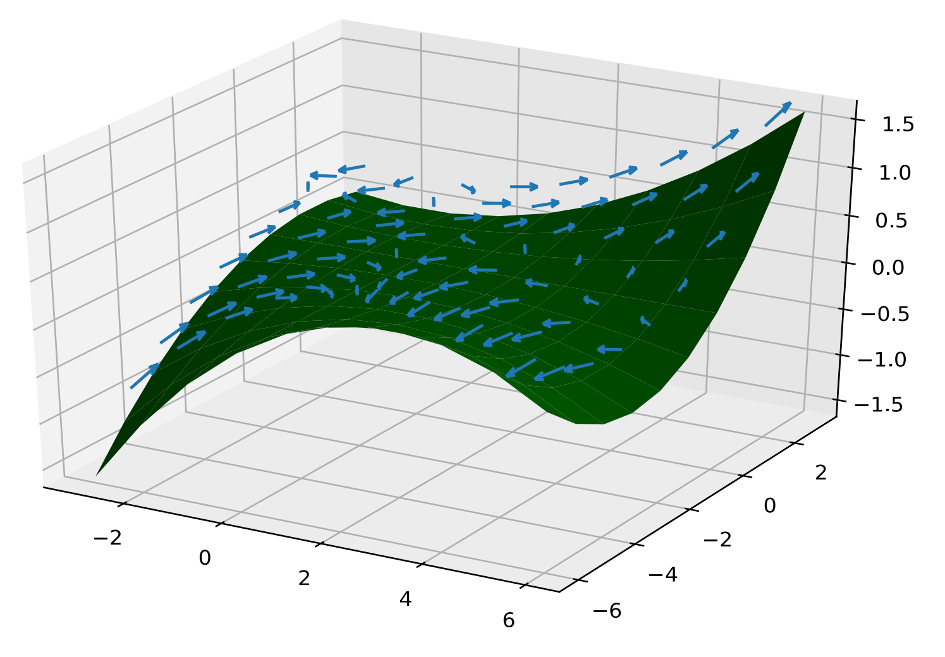
At this point, you might notice that there are $2 \times 2 \times 2 = 2^3 = 8$ possible values for the sequential combination, but only $2 + 2 + 2 = 2 \times 3 = 6$ possible values for the superimposition. This illustrates how the algebra of data types operates. Remarkably, algebraic data types obey not only the basic laws of arithmetic, but also the laws of calculus.[3,4] The derivative of 2^3 is 3×2^2, which gives the following type:

$$\textsc{Hole} \times \textsc{Bit}_2 \times \textsc{Bit}_3 + \textsc{Bit}_1 \times \textsc{Hole} \times \textsc{Bit}_3 + \textsc{Bit}_1 \times \textsc{Bit}_2 \times \textsc{Hole}$$

Here, Hole is the type-theoretical equivalent of an infinitesimal value in calculus — a type that is almost empty, but not quite. This tells us that the derivative of a type is equal to the superimposition of all possible ways to replace one component type with a hole.

Multivariable Calculus

In single variable calculus, the derivative has a straightforward geometric interpretation: it represents the slope of a line that is tangent to a curve. This line gives a good approximation of the curve near the tangent point, but as you move away from the tangent point, the approximation gets worse.

In multivariable calculus, the equivalent of the derivative is the gradient, which takes a surface (or an even higher-dimensional object) and gives the direction of steepest ascent at each point (that is, a vector).

Higher order derivatives can generate even more complex outputs. For our purposes, the matrix of second partial derivatives, sometimes called the Hessian, is of particular interest: it assigns a vector to each variable of the input function that shows how the given variable changes with respect to each of the others.

From Words to Vectors

Together, the above facts mean that if we could represent a corpus as an algebraic data type by assigning each word in the language to a unique type variable, the Hessian of that type would assign a vector to each word.

There are many possible ways to do this, but here's a simple one: we could regard each word in a language as a data type with a set of possible meanings. We could regard each sentence as a sequential combination of those data types. And we could regard a corpus (or an entire language) as the superimposition of all the sentence types it contains. The result is a single extremely large polynomial type:

$$\textsc{He} \times \textsc{Runs} + \textsc{She} \times \textsc{Runs} + \textsc{They} \times \textsc{Run} +$$
$$\textsc{Run} \times \textsc{Run} + \textsc{Run Away Now} + \ldots$$

Constructing a Hessian matrix from a polynomial is straightforward because differentiation is a linear operator: you take the derivative of each term and sum them all together. When you do this with a corpus of natural language text, the resulting matrix assigns a vector to each word of the language, and the resulting vectors can be used to represent analogies in the same way as those generated by more conventional algorithms. (See https://github.com/senderle/talediff for an implementation.)

Repetition and Nonlinearity

Geometrically speaking, derivatives are linear approximations of functions in a given neighborhood. And as we've seen, it is possible to take the derivative of a corpus of natural language text. But what could it mean to say that the word vectors generated by the above process are "linear approximations" of a corpus?

To make sense of this notion, let's consider the partial derivatives of the sentence **They × Run**. The partial derivative of this sentence with respect to **They** is **Run**, just as in ordinary multivariable calculus, the partial derivative of $\mathbf{xy}$ with respect to $\mathbf{x}$ is $\mathbf{y}$. What this implies, linguistically speaking, is that if you were to add new possible meanings to the word "they," the possible meanings of the sentence "they run" would change in a way governed only by the existing meanings of "run."

But what happens when you repeat a word? The partial derivative of **Run × Run** with respect to **Run** must be **Run + Run**, just as the partial derivative of $\mathbf{x^2}$ with respect to $\mathbf{x}$ is $\mathbf{2x}$ $(= \mathbf{x + x})$. What this implies, linguistically speaking, is that if you were to add new possible meanings to the word "run," the possible meanings of the sentence "run run" would change in a way governed not only by the existing meanings of the word "run," but also by the *new* meaning, which is not identical to the old.

The repetition of the word "run" thus creates nonlinearity by introducing a repetition that might not be a repetition. Or, to conclude by quoting Derrida: "the structure of iteration — and this is another of its decisive traits — implies both identity and difference."[5]

References

1. Tomas Mikolov, et. al. 2013. "Efficient estimation of word representations in vector space." In *Proceedings of Workshop at ICLR 2013*.
2. David M. Mimno and Laure Thompson. 2017. "The strange geometry of skip-gram with negative sampling." In *EMNLP 2017, Proceedings*, pages 2873–2878.
3. Conor McBride. 2001. "The derivative of a regular type is its type of one-hole contexts."
4. Michael Gordon Abbott, et. al. 2003. "Derivatives of containers." In *TLCA 2003, Proceedings*., pages 16–30.
5. Jacques Derrida, *Limited Inc.* (Evanston. Illinois: Northwestern University Press, 1988), 53.