Michael Repplinger
ILLC
University of Amsterdam
mpjrepplinger@gmail.com

Lisa Beinborn
Language Technology Lab
University of Duisburg-Essen, Germany
lisa.beinborn@uni-due.de

Willem Zuidema
ILLC
University of Amsterdam
zuidema@uva.nl

Vector-space models of words and sentences

How can we compute with words? Natural Language Processing is a research field focused on developing mathematical and computational models of language. For decades, models in this field were using techniques from discrete mathematics, but in recent years — with the rise of 'deep learning' — words and sentences are increasingly modelled with continuously valued numerical vectors. Can these models deal with the endless creativity of language, where ten thousands of words can be combined into an unbounded number of potential sentences? Michael Repplinger, Lisa Beinborn, Willem Zuidema discuss the four steps the field has gone through to arrive at the current state-of-the-art vector-space models of sentences.

Language, in written, spoken or signed form, is all around us. Most of our daily communication makes use of language, most of what we learn in school is conveyed through language, and most of the knowledge that humanity has built up is passed on to future generations through language. For allowing computers to take part in daily interactions with humans and to make use of the accumulated knowledge of humanity, we need mathematical models of language — models that allow computers to translate the noisy spoken, written or signed forms into internal representations with which they can compute.

The design of mathematical models for many different aspects of language and speech has a long history going back to at least the Indian grammarian Panini (6th, 5th or 4th century B.C.) and the Greek philosopher Aristotle (4th century B.C.). From the early 20th century, models based on formal logic became popular. Linguists like to point out that language comes so natural to us that we are largely unaware of its complexity; logical models played a key role in uncovering and detailing the complexity of language structure.

Although logic-based models continue to be important in linguistics, in Natural Language Processing, the field that studies language technology on computers — they are in the last few years rapidly making way for so-called 'vector-space models'. In those models, words are represented as numerical vectors, and sentence meanings are computed using a variety of operations from linear algebra.

In this article, we describe these developments, going over four major steps in the development of mathematical models of meaning, leading-up to current vector-space models of sentence meaning, compatible with sophisticated techniques from the deep learning field.

Step 1: Montague semantics — or: celebrating the sentence, ignoring the word

The American logician Richard Montague (1930–1971) is widely credited with providing the first successful attempt to formalize the semantics of a substantial 'fragment' of natural language. He developed what we now call *Montague Semantics* or *Montague Grammar* in a series of seminal papers [28, 29, 30].

Montague Semantics provided a systematic way to translate natural language to a logical language. For instance, when processing the sentence 'Gottlob loves Yoshua', the goal is to end up with the logical expression $\mathrm{love}(\mathrm{g})(\mathrm{y})$, where love is a binary predicate, describing a relation between the constants g (Gottlob) and y (Yoshua).

To achieve this translation, Montague proposed an ingenious system that combined insights from various modeling traditions in linguistics and logic. From categorical grammar, he borrowed a system to assign syntactic categories to words. 'Gottlob' and 'Yoshua' are proper nouns, and

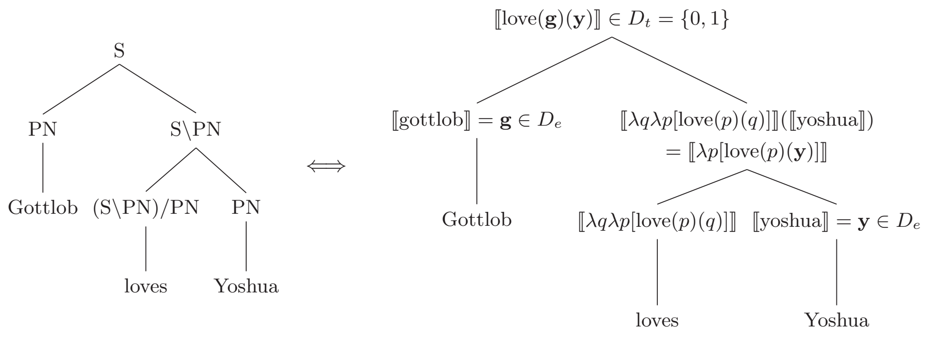


Figure 1 Translation to logical language.

receive the atomic category PN. ‘loves’ is a so-called transitive verb, that needs an argument on its left (the person that loves) and an argument on its right (the person that is loved). It therefore receives a ‘complex category’ that contains slashes that indicate what it can be combined with on the left and the right; in the case of love, the category is $(S\backslash PN)/PN$. According to the rules of categorical grammar, loves can then first be combined with Yoshua, and again be combined with Gottlob. This process yields a syntactic analysis depicted in Figure 1 (left).

From logic, Montague borrowed *intensional logic*, a higher-order typed logic (our examples will only use first-order predicate logic). Crucially, however, Montague proposed that the semantic, logical representation is derived simultaneously with the grammatical derivation. To achieve this, he enriched the logical expressions with elements from the *lambda calculus*. Gottlob and Yoshua again get assigned an atomic symbol, g and y respectively. The meaning of ‘loves’ is represented as $\lambda q \lambda p[\mathrm{love}(p)(q)]$. According to the rules of the lambda calculus, the semantics of the first argument that loves get combined with y, ends up in the place of the variable marked with the first λ in the expression q. This semantic derivation is depicted in Figure 1 (right).

On this basis, Montague and others have built an enormous body of work to analyze the semantics of natural language sentences. This work has addressed for instance the subtle ways in which ‘some’ and ‘any’ differ in sentences like ‘some mathematicians proved a theorem’ or ‘there wasn’t any mathematician that proved a theorem’. Interested readers will find [10] or [15] to be thorough introductions to complete semantic systems in the spirit of Montague’s proposal.

Strengths and weaknesses

One of the lasting impacts of Montague Semantics has been that it has highlighted an important feature of human language, known as the *principle of compositionality* (already hinted at in the work of Gottlob Frege [9]). This principle is commonly phrased along the lines of: “The meaning of a complex expression is determined by the meanings of its constituents and the rules used to combine them.”

Closely related to compositionality is the notion of *recursion*. Recursively defined processes are widely believed to underly the capacity of speakers of a language to build and understand arbitrary expressions of the language. By recursive *syntactic* processing, speakers can, in principle, build an infinite set of complex expressions from a finite set of simple ‘building blocks’. By a parallel recursive *semantic* process, speakers are then able to express and understand a potential infinitude of distinct meanings.

Formal semantics in the symbolic tradition excels at modeling compositionality and recursion — deriving algorithmically the meaning of a sentence from its smaller constituents. This focus on the structural aspects of language comes at a price, however, and the lexical foundation, i.e. the meaning of words, has received much less attention.

Another point concerns the limited integration of individual semantic theories. Montague’s original proposal was a ‘method of fragments’ — identifying a well-delineated syntactic fragment of the language, then formally describing this fragment. However, modern semantic theories generally no longer follow this approach, opting instead for descriptions of individual *semantic phenomena* without confining them to some *syntactic fragment*. Consequently, there is no syntactically defined subset of the language which can then be semantically analyzed in completion. The latter is however a requirement for *computational* models of symbolic semantics [31]. Consequently, the use of these theories for practical computational applications is limited as well.

Step 2: Distributional semantics — or: counting words

More recently, the field of *distributional semantics* emerged, which takes a completely different approach to modeling meaning, focusing on statistical techniques and large-scale modeling of word meaning. Central to distributional semantics is the *distributional hypothesis*, often expressed as: “You shall know a word by the company it keeps” [8]. To turn Firth’s slogan into a formal system, we use numerical vectors as representations and fill them with numbers based on counts of how often pairs of words occur near to each other in large databases of text.

Table 1 shows an idealized co-occurrence count, corresponding to a semantic space for the three nouns ‘mouse’, ‘elephant’ and ‘car’. Values in this table indicate how often a target word, i.e. the word we aim to represent as a vector, appeared near certain other words in a corpus. For example, ‘mouse’ and ‘animal’ co-occurred eight times, while ‘car’ and ‘animal’ co-occurred only once. Based on these hypothetical counts, we can represent these nouns as vectors in a four-dimensional semantic space, where the vectors consist of raw co-occurrence counts. In practical systems, dimensionality is usually in the order of hundreds or thousands, and raw counts are usually transformed into (weighted) frequency values.

The great advantage of vector representation for the meaning of words, is that we can now use standard mathematical tools that apply to numerical vectors to compute similarity between words. One frequently used similarity metric is cosine. Using *cosine similarity* on pairs of these word vectors, we can calculate their se-

	animal	**large**	**small**	**USB**
mouse	8	2	4	3
elephant	9	5	1	0
car	1	4	3	0

Table 1 Idealized context-counts for ‘mouse’, ‘elephant’ and ‘car’.

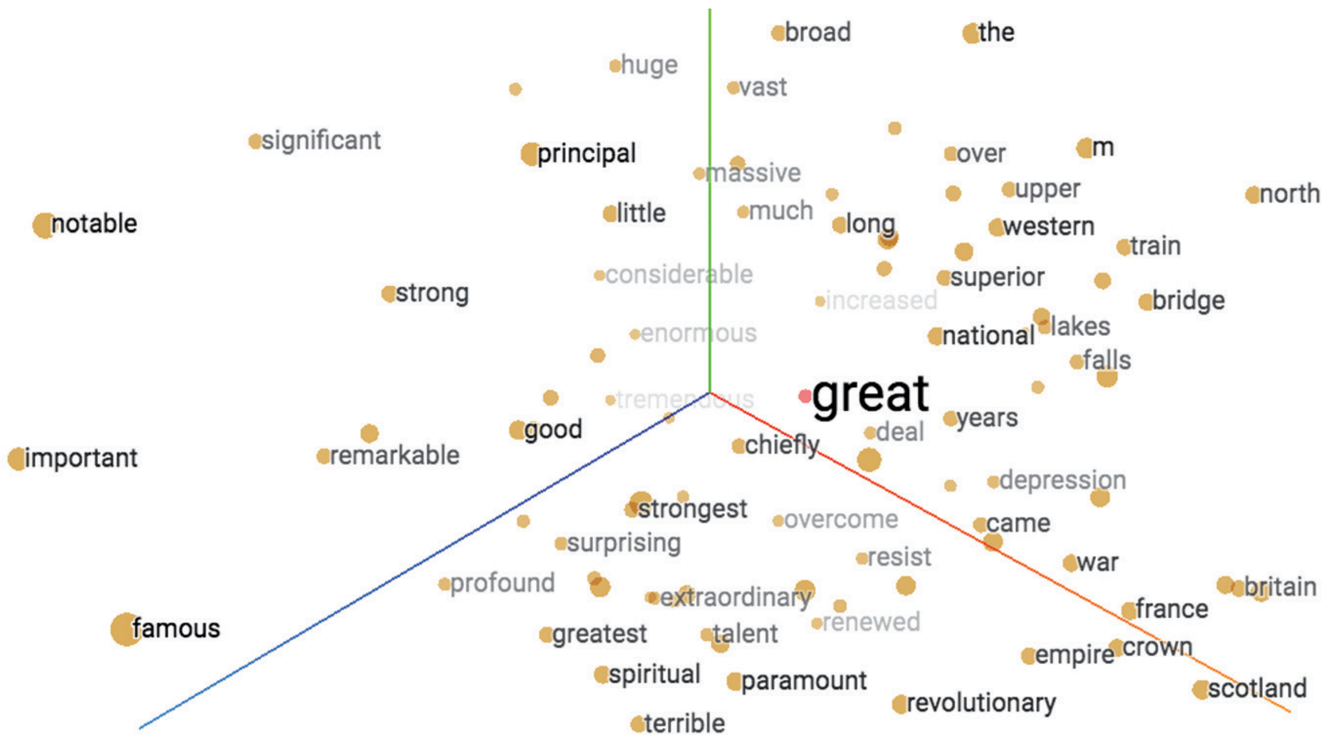


Figure 2 word2vec embedding of *great* and neighboring embeddings (1st/2nd/3rd principal component of 200 dimensional vector embeddings plotted).

mantic similarity: 0.86 (mouse, elephant), 0.57 (mouse, car), 0.61 (elephant, car). This would then represent, as intended, that 'mouse' and 'elephant' are more similar to each other than either of the two is to 'car' (animals versus non-animal), and that 'car' is (slightly) more similar to 'elephant' than to 'mouse' (being somewhat more similar in size). Note also the co-occurrence of 'mouse' with a seemingly unrelated word, 'USB', intended to illustrate the problem of *polysemy*. In most models, the vector representation of 'mouse' would be a 'mix' of contexts where 'mouse' refers to a rodent and contexts where the word refers to a computer component.

Modeling choices

Various modeling choices need to be made in the specification of a semantic vector space model. First, one must define what constitutes *context*, i.e. what we consider 'near to each other'. Commonly, this context is defined as 'the N neighboring words of the target word', where N is the *context window*, another parameter. Other choices are possible however. For example, context can be defined on a sub-word level, e.g. based on single characters. The choice of a similarity metric (Euclidean, correlation, cosine) also influences what it means for different words to be similar in meaning.

Models derive semantic information from the analysis of lexical co-occurrence, i.e. they are based on *context counting*. These counts are usually processed further by *weighting schemes*. Intuitively, these processing steps can be seen as adjusting the raw counts for word frequency, giving more weight to words that are rare but informative. One frequently used method used for this purpose is *pointwise mutual information*.

Another frequently employed processing step is to perform *dimensionality reduction* on the derived representations, for example by non-negative matrix factorization (NMF) or singular value decomposition (SVD) [3]. The dimensionality reduction step is motivated by two main concerns: computational efficiency, and possibly greater *generalization* capacity of the model [20]. The latter effect can result from merging (similar) contexts, i.e. associating a word with contexts of similar words even though it might have never appeared directly in these contexts itself, thus allowing the model to uncover additional similarities.

Step 3: Neural word embeddings — or: Learning to predict words

The classical distributional models described so far learn the meaning of words by analyzing the *counts* of context items, given some target word. Recently, a new class of models has emerged based on neural networks. These models are trained to either predict the most likely word given some context, or, in reverse direction, the most likely context for a given word. The word vectors that emerge as a side-effect of this prediction task, have turned out to be of much higher quality than the word vector from classical distributional semantics.

The origins of neural word embeddings can be traced back to the proposals of Hinton [16] and Bengio et al. [4], who used neural architectures in the derivation of word meanings. The currently most successful embedding algorithms were proposed by Mikolov et al. [23] and Mikolov et al. [22], accompanied by an efficient implementation dubbed `word2vec`.These modern neural word embeddings can be seen as feedforward neural networks (see box 'Feedforward neural networks') without hidden layers and nonlinear activation functions — the latter were identified as computational bottlenecks of the original models.

`word2vec` embeddings are based on two closely related algorithms. The first model, *Continuous Bag-of-Words* (CBOW), learns to *predict a word*, given a context of surrounding words. A 'projection layer' turns discretely encoded (sparse) word representations into continuous (dense) representations, which are then fed into a matrix — shared for all context words regardless of their position, hence the name 'bag-of-words' — for an output prediction of the most likely middle word for a given context. The second type of models is trained to predict in the opposite direction; given a word, the goal is to maximize the log probability of its surrounding context, i.e. models learn to *predict the context*, given an individual word. The context words do not need to appear consecutively in the corpus, i.e. individual words can be *skipped* when determining the context — referred to by the algorithm's name, *Skip-Gram*.

Linguistic regularities inside embeddings
Neural embeddings gained widespread attention for their representation of complex syntactic and semantic relational information. Very influential was the demonstration by Mikolov et al. [24] that a relation-specific, constant *vector offset* exists between the vector representations of related word pairs. Mikolov reported, for instance, that when you substract the vector for 'man' from the vector for 'king' and then add the vector for 'woman', you end up very near to the vector for 'queen':

$$v_{\text{king}} - v_{\text{man}} + v_{\text{woman}} \approx v_{\text{queen}}.$$

More generally, given a constant offset, the embedding space can be queried for an answer to *analogy questions* of the form: "word 1 is to word 2 as word 3 is to word 4". In this query, words 1, 2, 3 are given, while word 4 must be found in order to answer the question. Expressed as vector space operations, using cosine similarity to measure semantic similarity, the analogy query is defined as:

$$\arg\max_{v_4}(\cos(v_4, v_3 - v_1 + v_2)). \qquad (1)$$

Some of the other results produced by these analogy queries are:

$$v_{\text{Paris}} - v_{\text{France}} + v_{\text{Italy}} \approx v_{\text{Rome}},$$
$$v_{\text{apple}} - v_{\text{apples}} \approx v_{\text{car}} - v_{\text{cars}}.$$

These results have been interpreted as evidence that embeddings contain structure encoding a *gender* relation or feature, can relate countries and their capitals, and can learn a systematic *syntactic* relation between singular and plural word forms.

Arora et al. [1] provide theoretical support in favor of these claims, by showing that embedding algorithms like `word2vec` actively *impose* linear structure on the language data. The authors argue that this linearization effect stems from the *lower-dimensional* model internal representations, and because embedding models are effectively *nonlinear* data processors even though they do not contain the non-linear activation function of a full-power neural network.

Limitations of context-based approaches
Objections against embedding models have frequently been raised, noting that such models operate at a low level of the language (words or characters). Further criticism stems from the fact that models of this approach learn meaning by a comparably 'shallow' context analysis, without explicitly accounting for syntactic or semantic compositionality. Such broad criticism appears to be unwarranted: the examples shown above, of relational structure emerging in embedding models, strongly suggest that distributional models cannot be simply dismissed as 'linguistically insufficient'. Even though their architecture does not explicitly account for compositionality, the models succeed in extracting highly structured information from language data, through a combination of sophisticated statistical machinery, and due to their ability to process enormous amounts of data.

At the same time, some language phenomena pose major challenges for the class of purely context-based models. One example is the meaning derivation of *logical operators*, such as negation. In order to learn (sentential) negation from the context analysis alone, the intended meaning of negation would have to be fully expressed in the context distributions of sentences and their negation. It seems obvious however that human speakers can express (and understand) the negation of a sentence without knowledge of any 'neighboring' sentences. Confirming this intuition, Mohammad et al. show that highly contrasting items (including aspectual negation) tend to occur in very similar contexts. Models that purely learn from context, without the ability to deconstruct it if necessary, are then missing the relevant statistical information that would allow them to derive the correct meaning of negation.

Another obstacle for models that rely entirely on contextual learning is *sparsity of data*. Individual words occur abundantly across different contexts, but the frequency of combinations of words declines exponentially with the length of the sequence. Phrases, and short sentences are still encountered frequently enough to allow learning through contextual analysis alone. Long sentences on the other hand appear only infrequently, or are unique in the worst case, even in large data sets. Context-based learning however generally requires training on a large number of instances of an expression, thus making sentence length a limiting factor for context-only models.

Step 4: Compositional distributional semantics — or: having your cake and eat it

Speakers of a language are able to generalize from previously encountered expressions to expressions that a speaker never heard before. They achieve this by analyzing previously heard utterances as being built up from component parts, and by reusing these parts in all kinds of new combinations.

The symbolic models of language in the tradition of Montague emphasized the compositional semantics of sentences, but largely ignored how the meaning of words can be modelled and moreover relied on hand-built grammars specifying the syntactic and semantic properties of words. The distributional semantics tradition successfully developed vector representations for words. The recent neural word embedding models built on those successes, and moreover showed that automatically learned word representations encode many linguistically relevant relations between words. But distributional and neural word vectors have little to say about how sentence meanings can be constructed.

In the last few years, much research is devoted to bringing together insights about compositionality from the symbolic tradition, and insights from vector-space models of word meaning from the distributional and neural traditions: compositional distributional semantics.

Investigating different types of composition
Two landmark articles from this field of research are Mitchell and Lapata [25, 26]. The authors systematically evaluate different types of composition functions that can be

used in vector space models to compose sentence meaning from the meaning of smaller units, such as words.

Additive composition. The class of models based on *additive* composition is defined as:

$$z = Vx + Wy$$

where V, W are matrices, and composition is given by matrix multiplication. The very simplest instance of this class is given by vector addition:

$$z = x + y.$$

Mitchell and Lapata point out the clear limitations of this approach, such as insensitivity to word order due to commutativity of vector addition. For example, recursive application of vector addition would derive identical meanings for the phrases 'man bites dog' and 'dog bites man'.

Adding scalar weights for each vector results in the *weighted additive* model:

$$z = \alpha x + \beta y.$$

Here, left and right input vectors are scaled (uniformly, for each vector) by parameters α, β which are optimized on a development set.

Employing scalar weights or full matrices fixes the commutativity problem of simple vector addition, but another problem remains: the *blending* of meaning in additive models. Even the most complex (matrix-based) additive models compose vectors through (weighted) sums of their components. Additive composition effectively 'blends' or 'mixes' the meaning aspects of the composed word vectors, which can lead to undesirable results.

Consider for example the phrase 'brown cow', the result of composing vectors for 'brown' and 'cow'. Its intended meaning is a particular type of cow, one that is brown. Ideally, composition would yield a vector that represents the intersection of 'things that are brown' and 'things that are cows'. Recall now that word vectors gather their meaning from co-occurrence counts, and that the 'brown' vector contains meaning elements related to any brown objects encountered in the data: (brown) cows, (brown) dogs, (brown) houses, and so on. Since additive composition is incapable of context-dependent selection of only the relevant meaning aspects, the composed vector for 'brown cow' is bound to include various unrelated meaning aspects contained in the 'brown' vector (e.g. aspects related to brown dogs). It is easy to see that this effect runs counter to the intended meaning of the phrase, as an intersection of the two concepts.

Multiplicative and tensor-based composition. Multiplication of components is suggested as a solution to the *blending* problem above, leading to a general form of *multiplicative* models:

$$z = \mathcal{V}xy$$

where $\mathcal{V}$ is a 3rd-order tensor mapping the two constituent vectors x, y to output vector z. Composition is consequently a bilinear function of the constituents. A simple instance of this class is composition by *element-wise product*:

$$z = x \odot y.$$

The most powerful class of multiplicative models is given by composition with the *tensor product*:

$$z = x \otimes y.$$

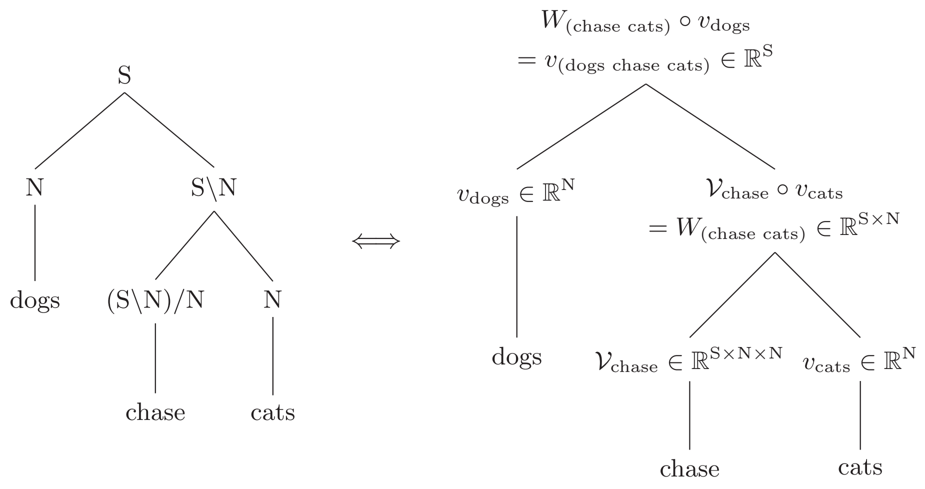
Mitchell and Lapata argue that multiplication of components as defined here is a solution to the previous blending problem: Additive composition does not relate the input components directly, adding them independently (at best, scaled by some constant factor) to form the output. Given component multiplication however, values interact directly through their products. For example, if a component with some high numerical value 'interacts' by multiplication with another component that has the value 0, the meaning contribution of the former is limited by its interaction with the latter. The authors describe this form of composition as feature *filtering*, in contrast to the feature *blending* that results from additive composition.

Mitchell and Lapata [25] evaluate the different model classes on a set of semantic similarity tasks, and arrive at somewhat inconclusive results. While the multiplicative models perform well (as predicted by the authors), only the simplest multiplicative model based on point-wise multiplication performs well across different tasks. Models using the tensor product, predicted to be more powerful than other simpler models, perform worse than most other models. Finally, the additive models perform comparably well across tasks, despite the theoretical objections raised against them.

The experimental findings of Mitchell and Lapata [25] are somewhat outdated, due to advances of models in recent years. Their proposed classification, however, had a lasting influence, and is still frequently invoked to classify and relate model architectures.

Modern approaches

Current research on compositional distributional semantics exists in two flavors. One class of models, which we collectively refer to as *type-based tensor approaches*, combines a powerful compositional mechanism with a robust distributional foundation of word meaning — at the cost of very high computational complexity. The approach results from the independent work of two research groups, presented by Baroni and Zamparelli [2] and Coecke et al. [5] approximately in parallel. Informally, these approaches can be seen as a 'translation' of


S
N
S\N
dogs
(S\N)/N
N
chase
cats
⟺
$W_{(\text{chase cats})} \circ v_{\text{dogs}}$
$= v_{(\text{dogs chase cats})} \in \mathbb{R}^{S}$
$v_{\text{dogs}} \in \mathbb{R}^{N}$
$\mathcal{V}_{\text{chase}} \circ v_{\text{cats}}$
$= W_{(\text{chase cats})} \in \mathbb{R}^{S \times N}$
dogs
$\mathcal{V}_{\text{chase}} \in \mathbb{R}^{S \times N \times N}$
$v_{\text{cats}} \in \mathbb{R}^{N}$
chase
cats
Figure 3 Sentence derivation in type-based tensor models.

the symbolic semantic theories in the Montague tradition into a vector space setting, through the use of higher-order tensors, as illustrated in Figure 3.

The other class of models consists of neural network architectures that — implicitly or explicitly — account for the demands of semantic compositionality. After a period of receiving only little attention, neural network models re-emerged in recent years as powerful, robust models, shown to be capable of solving a plethora of tasks across domains. The new generation of neural models produced outstanding results in the field of computer vision, as well as on language processing tasks such as machine translation, sentiment analysis or information retrieval. Many of these models are presented under the umbrella term of *deep learning*, originally meant to describe neural network models that contain a high number of hidden layers. In the appendix, we go over the main deep learning architectures that have been used to compute sentence representations in recent years.

Conclusions

Compared to mathematics, human language is a hopelessly messy, ambiguous, and redundant system. Yet, language is the carrier of a vast amount of knowledge, and for both scientific and technological reasons there is a need of adequate mathematical models of language.

In developing such models, linguists have looked at many different branches of mathematics. Until a few years ago, most useful tools where found in discrete mathematics: logics to describe the meaning of utterances, grammars of various sorts to describe the structure of sentences, lambda calculus to regulate to combinations of bits of meaning into larger wholes. In recent years, the field of Natural Language Processing is turning to continuous mathematics. Suddenly, words are modelled as continuously-valued numerical vectors, sentences as summations, mutiplications or more complicated combinations of these word vectors. And, importantly, these word and sentence vectors are computed using neural networks, optimized using stochastic gradient decent and other tools from the increasingly rich toolbox of 'deep learning'.

As always when scientific fields go through a paradigm shift, much of the excellent work done in the old paradigm is ignored or discarded. Fortunately, however, researchers in the domain of compositional distributional semantics are finding ways to integrate the main insights from the symbolic and neural traditions.

Acknowledgments
WZ is supported by a Gravitation grant, nr 024.001.006, from the Netherlands Organization for Scientific Research (NWO) to the Language in Interaction consortium. This paper is largely based on chapters 2, 3 and 4 of Repplinger [33]. We thank Dieuwke Hupkes for useful comments.

Feedforward neural networks

A simple feedforward neural network with two hidden layers, $\varphi^{\mathrm{nn2}}(x)$, can be compactly represented in vectorized notation as follows:

$$\begin{aligned} \varphi^{\mathrm{nn2}}(x) &= z,\\ h^1 &= f(W^1x+b^1),\\ h^2 &= f(W^2h^1+b^2),\\ z &= W^3h^2. \end{aligned} \tag{2}$$

Vector $\mathbf{z}$ is the network output, for an input of vector $\mathbf{x}$. Matrices $\mathbf{W^1}$, $\mathbf{W^2}$ and (bias) vectors $\mathbf{b^1}$, $\mathbf{b^2}$ are the trainable parameters of the network, together computing a linear transformation of the input $\mathbf{x}$. Using a non-linear (activation) function in place of f, the network will however learn complex functions that go well beyond simple linear transformations. Theoretical results by Cybenko [6] and Hornik et al. [8] established that neural networks are in principle able to approximate any function of practical interest to an arbitrary degree of precision, given a sufficiently high number of adjustable model parameters. Figure 4 is an equivalent representation of the network $\varphi^{\mathrm{nn2}}(x)$, in the traditional form of connected *neurons* computing a weighted sum of their input.

Simple feedforward networks are however ill-equipped to process sequences of arbitrary length, which is a basic requirement for the semantic modeling of sentences. Any sentence of a given length can be processed by a feedforward neural network with appropriate input layer dimensions. However, since the input layer dimension is fixed, the same model instance cannot be applied to sentences of any *other* length, preventing models from shared learning across sentences of different lengths. In order to process sequences of arbitrary length, such as sentences, the model should allow for the *recursive* processing of input sequences.

Recurrent neural networks

A simple, but powerful architecture satisfying the requirement of recursive input processing is the *recurrent neural network* (RNN), originally proposed in Elman [7].

The function learned by an RNN model is defined as a recursion over an input sequence of vectors $x_1,\dots,x_n$. This sequence of input vectors can be chosen quite generally, for example, as vectors representing the *words* of a sentence, or, breaking input down further, as the *characters* of a sentence.

At input step x_i of input sequence $\mathbf{x}$, the output z_i of an RNN is given by:

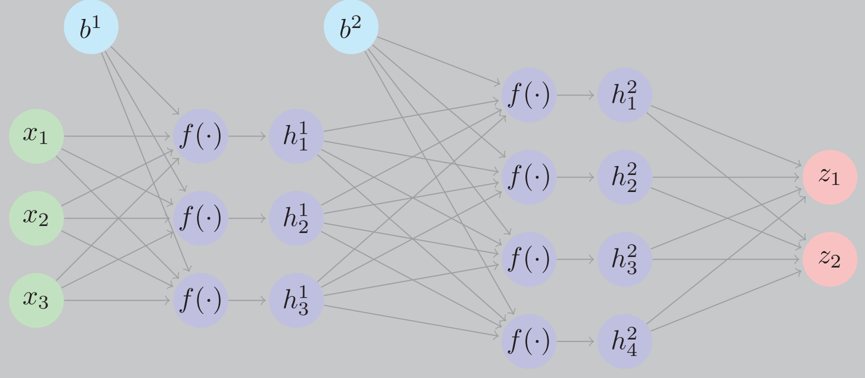


Figure 4 Simple feedforward neural network with two hidden layers.

$$z_i = f(W^{hh} z_{i-1} + W^{hx} x_i + b^h) \qquad (3)$$

where $z_i \in \mathbb{R}^H$, and H is the dimension of the hidden layer, $x_i \in \mathbb{R}^X$, $W^{hh} \in \mathbb{R}^{H \times H}$ is the matrix of hidden-to-hidden connections, $W^{hx} \in \mathbb{R}^{H \times X}$ the matrix of input-to-hidden connections, $b^h \in \mathbb{R}^H$ is a bias vector, and f an element-wise nonlinear function, called the *nonlinearity* of a layer. Frequent choices for this nonlinearity are the sigmoid function, $\tanh$, or a rectified linear unit (ReLU).

For finite input sequences, the general recursive definition can be replaced by an *unfolded* version of the model instance. Given an input sequence of length n, the network can be seen as a chain of $n+1$ hidden layers or states, where each state has two outgoing connections (one to the next hidden layer, one being the output at the current step), and two incoming connections (one being the output of the previous hidden layer, one for the input at the current step). The i-th state of this chain is the model representation of the input sequence up to and including input element i, given by summing the linear combination of the i-th input vector, the linear combination of the $(i-1)$-th state or output, and the bias vector, then applying the element-wise nonlinearity f to the resulting vector.

Using simple RNN models, the work of Mikolov [21] constitutes an early, influential exploration of RNNs applied to language tasks. In general, however, most notable results produced by RNN architectures in recent years were in fact *extensions* of the architecture, often produced by the highly successful class of LSTM models, discussed later on.

Recursive neural networks

A structural extension of the basic RNN architecture is the *recursive neural network*, or tree-shaped recurrent neural network (tRNN). The current tRNN architecture was introduced by Socher et al. [33] and was based on earlier proposals of Pollack [32] and Goller and Küchler [12]. The tRNN can be seen as a generalization of RNNs in terms of input structure, in the following sense: While the input sequence of a simple RNN is unstructured, and composition invariably proceeds in one direction, the tRNN architecture allows for the compositional process to be structured by syntactic analysis of the input. In practice, this syntactic analysis is usually provided externally, by providing the model with a parse tree for a given input sentence.

The decision to provide the neural network with a parse tree of the input is motivated by linguistic considerations, attempting to include some of the structural information that lends power to the symbolic models of formal semantics. While the tRNN class of models initially proved to be successful, the field has largely moved on to models that do not require external information (such as parse trees), allowing for much larger data sets to be used in training.

Neural networks with long-term memory

A major challenge when training *deep* neural networks — networks consisting of many stacked hidden layers — is the *vanishing gradient problem*. The cause for this problem relates to the training algorithm of networks, which passes information (gradients) down the network as a chain of *products*. Since individual terms of this chain are often small, their product tends to decrease with the length of the chain, to the point of vanishing. As a result, lower layers of a deep network only receive a greatly diminished learning signal, thus negatively affecting learning success.

LSTM networks

The seminal work of Hochreiter and Schmidhuber [17] presented a solution to the problem, by introducing the *long short-term memory architecture* (LSTM). We only describe the general idea behind the approach here, and refer the reader to Graves [13] for a technical explanation of the mechanisms.

The LSTM architecture, based on the (simple) RNN model of (3), adds *memory cells* and *(control) gates*, allowing a higher degree of retainment of gradient information across network layers. Memory cells are vectors retaining past gradient information, where access to these cells is controlled by three types of gates (*input*, *output*, and *forget* gates). Intuitively, these gates can be seen as vector space versions of logic gates, interacting with the components of the memory cells by pointwise multiplication with values near 0 or 1, i.e. *soft* boolean values. During training, stored gradient information and the gates interact to *preserve* old and *select* new gradient information that will be passed downwards in the network. This mechanism leads to major improvements in training effectiveness of deep networks, and most major results in recent years by models of the RNN class were produced by LSTMs, or further model extensions of the RNN architecture, with added LSTM gates.

RNN models using LSTM gates generated major results on several language tasks. In an early study of LSTMs and language processing, Gers and Schmidhuber [11] showed that their model can learn simple context-free and context-sensitive languages, e.g. strings of the form $\mathrm{a}^n\,\mathrm{b}^n$ and $\mathrm{a}^n\mathrm{b}^n\mathrm{c}^n$, respectively. In Graves [14], LSTM models are used to generate novel sentences after being trained on Wikipedia data, and learn to produce sentences in realistic script (i.e. the model learned *handwriting*).

Tree-structure information for free?

As mentioned above, recursive neural networks, i.e. tree-shaped RNNs, make use of explicit syntactic information to guide the processing of sentences. LSTM models have been suggested as effectively replacing the need for such explicit syntactic information, due to their ability to store (training) information across the processing of long input sequences like sentences. While syntactic information is given to the tree-structured networks *explicitly*, LSTM models possibly can rely on *implicit* syntactic information through their storage mechanism. Whether syntactically guided processing is a useful or necessary feature of distributional models is not conclusively answered yet. It should be noted however that the two architectures can be combined, i.e. tree-structured networks can be enriched by adding a memory mechanism. See, for example, the proposals of Le and Zuidema [19] and Tai et al. [34], extending tRNN architectures with LSTM gates to improve training efficiency of deep networks, and help with modeling long distance dependencies.

References

1 Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma and Andrej Risteski, Rand-walk: A latent variable model approach to word embeddings, *Transactions of the Association for Computational Linguistics* 4 (2016), 385–399.

2 Marco Baroni and Roberto Zamparelli, Nouns are vectors, adjectives are matrices: Representing adjective noun constructions in semantic space, in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2010, pp. 1183–1193.

3 Marco Baroni, Georgiana Dinu and German Kruszewski, Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors, in *ACL (1)*, 2014, pp. 238–247.

4 Yoshua Bengio, Rejean Ducharme, Pascal Vincent and Christian Jauvin, A neural probabilistic language model, *Journal of Machine Learning Research* 3 (2003), 1137–1155.

5 Bob Coecke, Mehrnoosh Sadrzadeh and Stephen Clark, Mathematical foundations for a compositional distributional model of meaning, arXiv:1003.4394, 2010.

6 George Cybenko, Approximation by superpositions of a sigmoidal function, *Mathematics of Control, Signals, and Systems (MCSS)* 2(4) (1989), 303–314.

7 Jeffrey L. Elman, Finding structure in time, *Cognitive science* 14(2) (1990), 179–211.

8 John R. Firth, *A Synopsis of Linguistic Theory, 1930–1955*, Oxford University Press, 1957.

9 Gottlob Frege, Über sinn und Bedeutung, *Zeitschrift für Philosophie und philosophische Kritik*, 1892.

10 L.T.F. Gamut, *Logic, Language, and Meaning. Volume II. Intensional Logic and Logical Grammar*, University of Chicago Press, 1991.

11 Felix A. Gers and Jürgen Schmidhuber, LSTM recurrent networks learn simple context-free and context-sensitive languages, *IEEE Transactions on Neural Networks* 12(6) (2001) 1333–1340.

12 Christoph Goller and Andreas Küchler, Learning task dependent distributed representations by backpropagation through structure, in *IEEE International Conference on Neural Networks, 1996*, Vol. 1, IEEE, 1996, pp. 347–352.

13 Alex Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*, PhD thesis, Technical University Munich, 2008.

14 Alex Graves, Generating sequences with recurrent neural networks, arXiv:1308.0850, 2013.

15 Irene Heim and Angelika Kratzer, *Semantics in Generative Grammar*, Vol. 13, Blackwell, 1998.

16 Geoffrey E. Hinton, Learning distributed representations of concepts, in *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Vol. 1, Amherst, MA, 1986, p. 12.

17 Sepp Hochreiter and Jürgen Schmidhuber, Long short-term memory, *Neural computation,* 9(8) (1997), 1735–1780.

18 Kurt Hornik, Maxwell Stinchcombe and Halbert White, Multilayer feedforward networks are universal approximators, *Neural Networks* 2(5) (1989), 359–366.

19 Phong Le and Willem Zuidema, Compositional distributional semantics with long short term memory, *Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics (*SEM)*, 2015.

20 Omer Levy and Yoav Goldberg, Neural word embedding as implicit matrix factorization, in *Advances in Neural Information Processing Systems*, 2014, pp. 2177–2185.

21 Tomas Mikolov, *Statistical Language Models Based on Neural Networks*, PhD thesis, Brno University of Technology, 2012.

22 Tomas Mikolov, Kai Chen, Greg Corrado and Jerffey Dean, Efficient estimation of word representations in vector space, arXiv:1301.3781, 2013.

23 Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado and Jeff Dean, Distributed representations of words and phrases and their compositionality, in *Advances in Neural Information Processing Systems*, 2013, pp. 3111–3119.

24 Tomas Mikolov, Wen-tau Yih and Geoffrey Zweig, Linguistic regularities in continuous space word representations, in *Proceedings of NAACL-HLT*, Vol. 13, 2013, pp. 746–751.

25 Jeff Mitchell and Mirella Lapata, Vector-based models of semantic composition, in *Proceedings of the ACL*, 2008, pp. 236–244.

26 Jeff Mitchell and Mirella Lapata, Composition in distributional models of semantics, *Cognitive Science* 34(8) (2010), 1388–1429.

27 Saif M. Mohammad, Bonnie J. Dorr, Graeme Hirst, and Peter D. Turney, Computing lexical contrast, *Computational Linguistics* 39(3) (2013), 555–590.

28 Richard Montague, English as a formal language, in Bruno Visentini et al., eds., *Linguaggi nella società e nella tecnica*, Edizioni di Communita, 1970, pp. 189–224.

29 Richard Montague, Universal grammar, *Theoria* 36(3) (1970), 373–398.

30 Richard Montague, The proper treatment of quantification in ordinary English, in *Approaches to Natural Language*, Springer, 1973, pp. 221–242.

31 Barbara H. Partee, Montague grammar, in Neil J. Smelser and Paul B. Baltes, eds., *International Encyclopedia of the Social & Behavioral Sciences*, Vol. 11, Elsevier, 2001.

32 Jordan B. Pollack. Recursive distributed representations, *Artificial Intelligence* 46(1) (1990), 77–105.

33 Michael Repplinger, *Understanding Generalization: Learning Quantifiers and Negation with Neural Tensor Networks*, Master Thesis, Master of Logic, University of Amsterdam, 2017.

34 Richard Socher, Christopher D. Manning and Andrew Y. Ng, Learning continuous phrase representations and syntactic parsing with recursive neural networks, in *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop,* 2010, pp. 1–9.

35 Kai Sheng Tai, Richard Socher and Christopher D Manning, Improved semantic representations from tree-structured long short-term memory networks, *Association for Computational Linguistics 2015 Conference*, 2015.