The compositionality of neural networks: integrating symbolism and connectionism

Authors

Abstract

Abstract of our paper.

1. Introduction

The advancements of distributional semantics of the word level allowed the field of natural language processing to move from discrete mathematical methods to models using continuous numerical vectors (see e.g. Clark, 2015; Erk, 2012; Turney and Pantel, 2010). Such continuous vector representations operationalise the distributional semantic hypothesis, stating that semantically similar words have similar contextual distributions (see, e.g. Miller and Charles, 1991), by keeping track of contextual information from large textual corpora. They can then act as surrogates for word meaning and be used, for example, to quantify the degree of semantic similarity between words, by means of simple geometric operations (Clark, 2015). Words represented in this way can be an integral part of the computational pipeline and have proven to be useful for almost all natural language processing tasks (see e.g. Hirschberg and Manning, 2015).

After the introduction of continuous word representations, a logical next step involved understanding how to *compose* these representations to obtain representations for phrases, sentences and even larger pieces of discourse. Some early approaches to do so stayed close to formal symbolic theories of language and sought to explicitly model semantic composition by finding a composition function that could be used to combine word representations. The adjective-noun compound 'blue sky', for instance, would be represented as a new vector resulting from the composition of the representations for 'blue' and 'sky'. Examples of such composition functions are as simple as vector addition and (point-wise) multiplication (e.g. Mitchell and Lapata, 2008) up to more powerful tensor-based operations where, going back to our 'blue sky' example, the adjective 'blue' would be represented as a matrix, which would be multiplied with the noun vector 'sky' to return a modified version of the latter (e.g. Baroni and Zamparelli, 2010).

A more recent trend in word composition exploits *deep learning*, a class of machine learning techniques that model language in a completely data-driven fashion, by defining a loss on a downstream task (such as sentiment analysis, language modelling or machine translation) and learning the representations for larger chunks from a signal backpropagated from this loss. In terms of how they compose representations, models using deep learning can be divided in roughly two categories. In the first category, deep learning is exploited to learn only the actual composition functions, while the order of composition is defined by the modeller. An example is the *recursive neural network* of Socher et al. (2010), in which representations for larger chunks are computed recursively following a predefined syntactic parse tree of the sentence. While the composition function in this approach is fully learned from data using *backpropagation through structure* (Goller and Kuchler, 1996), the tree structure that defines the order of application has to be provided to the model, allowing models to be 'compositional by design'. More recent variants lift this dependency on external parse trees by jointly learning the composition function and the parse tree (Le and Zuidema, 2015; Kim et al., 2019, i.a.), often at the cost of computational feasibilty.

In the second type of deep learning models, no explicit notion of (linguistic) trees or arbitrary depth hierarchy is entertained. Earlier models of this type deal with language processing sequentially and use RNN-based processing units such as LSTMs (Hochreiter and Schmidhuber, 1997) and GRUs

(Chung et al., 2014) at their core (Sutskever et al., 2014). An important contribution to their effectiveness comes from attention mechanisms, which allow recurrent models to keep track of long-distance dependencies more effectively (Bahdanau et al., 2014). More recently, these models went all in on attention, abandoning sequential processing in favour of massively distributed sequence processing all based on attention (Vaswani et al., 2017). Whil the architecture design of this class of models is not motivated by knowledge about linguistics, they are – through their ability to easily process very large amounts of data – more successful than the previously mentioned (sub)symbolic models on a variety of natural language processing tasks.

Different types of models that compose smaller representations into larger ones can be compared along many dimensions. Most commonly, they are evaluated by the usefulness of their representations for different types of tasks, but also scalability, how much data they need to develop their representations (sample efficiency) and their computational feasibility play a role. It is, however, difficult to explicitly assess if the composition functions they implement are appropriate for natural language and, importantly, to what extent they are in line with the vast amount of knowledge and theories about semantic composition from formal semantics and (psycho)linguistics. While the composition functions of symbolic models are easy to understand (because they are defined on a mathematical level), it is not empirically established that their rigidity is appropriate for dealing with the noisiness and complexity of natural language. Neural models, on the other hand, seem very well up to handling noisy scenarios, but are often argued to be fundamentally incapable of conducting the types of compositions required to process natural language (Pinker, 1984; Fodor and Pylyshyn, 1988; Marcus, 2003) or at least to not use those types of compositions to solve their tasks (e.g., Lake and Baroni, 2018).

In this work, we consider the latter type of models and focus in particular on whether these models are capable of implementing *compositionality*, a question that recently, with the rise of their success, has attracted the attention of many researchers. While many empirical studies can be found that explore the compositional capabilities of neural models, they have not managed to convince the community of either side of the debate: whether neural networks are able to behave compositionally is still an open question. One issue standing in the way of more clarity on this issue, is that different researchers have different interpretations of what exactly it means to say that a model is or is not compositional, a point exemplified by the vast number of different tests that exist for compositionality. Some studies focused on testing if models are able to productively use symbolic *rules* (e.g. Lake and Baroni, 2018); Some instead consider models' ability to implement *hierarchical* structures (Hupkes et al., 2018; Linzen et al., 2016); Yet others consider if models can segment the input into reusable parts (Johnson et al., 2017).

This variety of tests for compositionality of neural networks existing in the literature is better understandable considering the open nature of the principle of compositionality, by Partee (1995) phrased as '*The meaning of a whole is a function of the meanings of the parts and of the way they are syntactically combined*'. While there is ample support for the principle itself, there is less consensus about its exact interpretation and practical implications. One important reason for this is that the principle is not theory-neutral: it requires a theory of both syntax and meaning, as well as functions to determine the meaning of composed parts. Without these components, the principle of compositionality is formally vacuous (Janssen, 1983; Zadrozny, 1994), because also trivial and intuitively uncompositional solutions that cast every expression as one part and assign it a meaning as a whole do not formally violate the principle of compositionality. To empirically test models for compositionality it is thus necessary to first establish *what* is to be considered compositional.

With this work, we aim to contribute to clarity on this point, by presenting a study in which we collect different aspects of and intuitions about compositionality from linguistics and philosophy and bundle them in an overarching test-suite that can be used to better understand the composition functions learned by neural models trained end-to-end on a downstream task. The contribution of our work, we believe, is two-fold. First, we provide a bridge between, on the one hand, the vast amount of theory about compositionality that underpins symbolic models of language and semantic composition and, on the other hand, the neural models of language that have proven to be very effective in many natural language tasks that seem to require compositional capacities. We identify different components of compositionality within this literature, and we provide tests that allow to test for these components independently. We believe that the field will profit from such a principled analysis of compositionality and that this analysis will provide clarity concerning the different interpretations that may be entertained by different researchers. Practically, a division into clearly understood components can help to identify and categorise the strengths and weaknesses of different models. We provide concrete and usable tests, bundled in an versatile test suite that can be applied to any model. To demonstrate its usefulness, we apply our tests to three popular sequence to sequence models: a recurrent, a convolution based and a transformer model, and we provide an in depth analysis of the results.

Secondly, we touch upon the complex question that concerns the extent to which a model needs to be explicitly compositional to adequately model data of which the underlying structure is, or seems, compositional. We believe that, in a time where the most successful natural language processing methods require large amounts of data and are not directly motivated by linguistic knowledge or structure, this question bears more relevance than ever.

Outline

In what follows, we first briefly revise other literature with similar aims and sketch how our work stands apart from previous attempts to assess the extent to which networks implement compositionality. We describe previously proposed data sets to evaluate compositionality as well as so studies that evaluate the representations of pre-trained models. In Section 3, we give a theoretical explanation of the five notions that we devise tests for, which we motivate by providing interpretations of these notions from the philosophy of language and theoretical linguistics literature. In Section 4, we describe the data set that we use for our study, followed by a brief description of the three types of models that we compare in our experiments. A full description of the experiments for our data set, as well as details on model training and evaluation, can be found in Section 6. We report and analyse the results of all our experiments in Section 7 and reflect upon their implications in Section 8.

2. Related Work

Whether artificial neural networks are fundamentally capable of representing compositionality, trees and hierarchical structure has been a prevalent topic ever since the first connectionism models for natural language were introduced. Recently, this topic has regained attention, and a substantial number of empirical studies can be found that explore the compositional capacities of neural models, with a specific focus on their capacity to represent *hierarchy*. These studies can be roughly divided into two categories: studies that devise specific data sets that models can be trained and tested on to assess if they behave compositionally, and studies that devise methods to assess the representations that are learned by models trained on some independent (often natural) data set. In the next two sections, we discuss a number of both types of studies.

2.1 Evaluating compositionality with artificial data

Specifically crafted, artificial data sets to evaluate compositionality are typically generated from an underlying grammar. It is then assumed that models can only find the right solution to the test set if they learned to interpret the training data in a compositional fashion. Below, we discuss a selection of such data sets and briefly review their results.

2.1.1 Arithmetic language

One of the first (recent) data sets proposed as testbed to reveal how neural networks process hierarchical structure is the *arithmetic language*, introduced by Veldhoen et al. (2016). Veldhoen et al. test networks for algebraic compositionality by looking at their ability to process spelled out, nested arithmetic expressions. In a follow up paper, to gain insight in the types of solution that networks encode, the same authors introduce *diagnostic classifiers*, trained to fire for specific strategies used to solve the problem. They show that simple recurrent networks do not perform well on the task, but gated recurrent networks can generalise well to lengths and depths of arithmetic expressions that were not in the training set, but that this performance quickly deteriorates when the length of expressions grows (Hupkes et al., 2018). From this, they conclude that these models are – to some extent – able to capture the underlying compositional structure of the data.

2.1.2 SCAN

In 2018, Lake and Baroni proposed the SCAN data set, describing a simple navigation task that requires an agent to execute commands expressed in a compositional language. The authors test various sequence-to-sequence models on three different splits of the data: a random split, a split testing for longer action sequences and split one that requires compositional application of words learned in isolation. The models obtain almost perfect accuracy on the first split, while performing very poorly on the last two, which the authors argue require a compositional understanding of the task. They conclude that – after all these years – sequence-to-sequence recurrent networks are still not *systematic*. In a follow up paper, the same authors criticise these findings and propose a new set of splits which focuses on rearranging familiar words (i.e., "jump", "right" and "around") to form novel meanings ("jump around right")(Loula et al., 2018). Although they collect considerably more evidence for systematic generalisation within their amended setup, the authors confirm their previous findings that the models do not learn compositionally.

2.1.3 Lookup tables

Similar conclusions were drawn by Liška et al. (2018), who introduce a minimal compositional test where neural networks need to apply function compositions to correctly compute the meaning of sequences of lookup tables. The meanings of atomic tables are exhaustively defined and presented to the model, so that applying them does not require more than rote memorisation. The authors show that out of many models trained with different initialisations only a very small fraction exhibits compositional behaviour, while the vast majority does not.[1]

2.1.4 Logical inference

Bowman et al. (2015) proposed a data set which uses a slightly different setup: it assesses models' compositional skills by testing their ability to infer logical entailment relations between pairs of sentences in an artificial language. The grammar they use licenses short, simple sentences; the relations between these sentences are inferred using a natural logic calculus that acts directly on the generated expressions. Bowman et al. show that recursive neural networks, that recursively apply the same composition function and are thus compositional by design obtain high accuracies on this task. Mul and Zuidema (2019) show that also gated recurrent models can perform well on an adapted version of the same task, which uses a more complex grammar. With a series of additional tests, Mul and Zuidema provide further proof for basic compositional generalisation skills of the best-performing recurrent models.

1. Hupkes et al. (2019) show how adding an extra supervision signal to the network's attention consistently results in a complete solution of the task, but it is not clear how their results extend to other, more complicated scenarios. Korrel et al. (2019) propose a novel architecture with analogous, complete solutions without the need for extra supervision.

2.2 Evaluating compositionality with natural data

While very few studies present methods to explicitly evaluate how compositional the representations of models that are trained on independent data sets are, there is a number of studies that focus on evaluating aspects of learned representations that are related to compositionality. In particular, starting from the seminal work of Linzen et al. (2016), the evaluation of the syntactic capabilities of neural *language models* has attracted a considerable amount of attention. While the explicit focus of such studies is on the syntactic capabilities of different models and not on providing tests for compositionality, many of the results in fact concern the way that neural networks process the types of hierarchical structures often assumed to underpin compositionality.

2.2.1 Number agreement

Linzen et al. (2016) propose to test the syntactic abilities of LSTMs by testing to what extent they are capable of correctly processing long-distance subject-verb agreement, a phenomenon they argue to be commonly regarded as evidence for hierarchical structure in natural language. They devise a *number-agreement* task and find that a pre-trained state-of-the-art LSTM model (Jozefowicz et al., 2016) does not capture the structure-sensitive dependencies.

Later, these results were contested by a different research group, who repeated and extended the study with a different language model and tested a number of different long-distance dependencies for English, Italian, Hebrew and Russian (Gulordava et al., 2018). The results do not match the earlier findings of Linzen et al. (2016): Gulordava et al. (2018) find that an LSTM language model can solve the subject-verb agreement problem well, even when the words in the sentence are replaced by syntactically nonsensical words, which they take as evidence that the model is indeed relying on syntactic and not semantic clues.[2]

Whether the very recent all-attention language models do also capture such syntax-sensitive dependencies is still an open question. Some (still unpublished) studies find evidence that such models score high on the previously described number-agreement task (Goldberg, 2019) and show that the attention mechanisms capture syntactic structure (Lin et al., 2019; Vig and Belinkov, 2019). More mixed results are reported by others (Wolf, 2019).

2.2.2 Syntax in Machine Translation

DH: 2.2.2 - 72 - Dieuwke to Verna

- VD: Blevins et al. (2018) train four deep recurrent neural networks on the tasks of dependency parsing, semantic role labelling, machine translation and language modelling. The networks have four recurrent layers that are used to predict constituency-based properties, such as a word's POS tag and (grand-)parent tags, and dependency arcs. The models were found to capture large amounts of syntactical information, even when trained on language modelling. The authors furthermore note that higher-level syntactic tasks capture these properties more accurately in deeper layers and that particularly for semantic role labelling the lower-level properties are captured in lower layers than the higher-level properties.
- Tran et al. (2018)
- VD: Mareček and Rosa (2018) extracted self-attention patterns from the encoder of a Transformer model trained to perform machine translation. The self-attention patterns were aggregated across layers and used to construct binary constituency trees. The trees contained

2. The task proposed by Linzen et al. (2016) served as inspiration for many studies investigating the linguistic or syntactic capabilities for neural language models, and also the task itself was used in many follow-up studies. Such studies, that we will not further discuss, are generally positive about the extent to which recurrent language models represent syntax.

clauses, noun phrases and shorter verb phrases that bared strong similarities to subparts of the Penn Treebank constituency trees. Similarly, undirected trees extracted from the self-attention patterns were found to bear similarities to dependency trees, indicating the high degree of syntactic information of natural language captured in the patterns learned by Transformer. *Comment from Verna: this "paper" is just a 2-page abstract and does not present precise quantitative results on how accurate the constituency or dependency trees are*

- VD: ArXiv paper of Marecek & Rosa, https://arxiv.org/pdf/1906.01958.pdf: extend the analysis presented by Mareček and Rosa (2018). Qualitatively, the authors find two frequent patterns: diagonal self-attention and series of self-attention focused at the same term (the series are called *balustrades*). The diagonal self-attention pattern appears often in the bottom layer, which indicates that this layer mostly captures lower-level or lexical features. In many cases, the balustrades span subwords that form a syntactical phrase. This behaviour is quantified through the construction of constituency trees using a minimally informed algorithm that was found to create much more accurate trees compared to left and right-branching baselines, for a total of three languages.

2.3 Intermediate conclusions

We reviewed various attempts to assess the extent to which neural models are able to implement compositionality and hierarchy. This overview illustrated the difficulty and importance of evaluating the behaviour of neural models but also showed that whether neural networks can or do learn compositionally is still an open question. Both strands of approaches we considered – approaches that use special compositional data sets to train and test models, and approaches that instead focus on the evaluation of pre-trained models – report positive as well as negative results.

In the first approach, researchers try to encode a certain notion of compositionality in the task itself. While it is important, when testing for compositionality, to make sure the specific task that networks are trained on has a clear demand for compositional solutions, we believe these studies fall short in linking the task proposed to a clearly-defined notion of compositionality. Further, we believe that the multifaceted notion of compositionality cannot be exhausted in one single task. In the following section, we disconnect testing compositionality from the task at hand and disentangle five different theoretically motivated ways in which a network can exhibit compositional behaviour that are not a priori linked to a specific downstream task.

The second type of studies roots its tests into clear linguistic hypotheses. However, by testing neural networks that are trained on uncontrolled data, they lose the direct connection between compositionality and the downstream task. Although compositionality is widely considered to play an important role for natural language, it is unknown what type of compositional skills – if any – a model needs to have to successfully model tasks involving natural language, such as for instance language modelling. If it cannot be excluded that successful heuristics or syntax-insensitive approximations exists, a negative result can not be taken as evidence that a particular type of model cannot capture compositionality, it merely indicates that this exact model instance did not capture it in this exact case. While, in the long run, we also wish to reconnect the notion of compositionality to natural data, before being able to do so, it is of primary importance to reach an agreement about what defines compositionality and how it should be tested in neural networks.

3. Testing compositionality

In the previous section, we discussed various attempts to evaluate the compositional skills of neural network models. We argued that progressing further on this question requires tests that are more strongly grounded in the literature on compositionality. We now arrive at the theoretical part of the core of our work, in which we set the theoretical ground for the five tests we propose and conduct in this paper. We describe five aspects of compositionality that are explicitly motivated by theoretical literature on this topic and that we argue can be separately tested to break down which aspects of compositionality a model does and does not capture. We first discuss these five notions and their interpretation. Later, in Section 6.2, we provide details about how we operationalise them in concrete tests.[3]

3.1 Systematicity

The first property we propose to test for – following many of the works presented in the related work section of this article – is *systematicity*, a notion frequently used in the context of compositionality. The term was introduced by Fodor and Pylyshyn (1988) – notably, in a paper that argued against connectionist architectures – who used it to denote that

> [t]he ability to produce/understand some sentences is intrinsically connected to the ability to produce/understand certain others" (Fodor and Pylyshyn, 1988, p. 25)

This ability concerns the recombination of known parts and rules: anyone who understands a number of complex expressions also understands other complex expressions that can be built up from the constituents and syntactical rules employed in the familiar expressions. To use a classic example from Szabó (2012): someone who understands 'brown dog' and 'black cat' also understands 'brown cat'.

Fodor and Pylyshyn (1988) contrast systematicity with storing all sentences in an atomic way. Someone who simply entertains a dictionary to map sentences to meanings would not be able to understand new sentences, even if they were similar to the ones occurring in their dictionary. Since humans are clearly able to infer meanings for sentences they have never heard before, they must use some systematic process to construct these meanings from the ones they internalised before.

By the same argument, however, any model that is able to generalise to a sequence outside its training space (its test set), should have learned to construct outputs from parts it perceived during training and some rule of recombination. Thus, rather than asking if a model is systematic, a more interesting question is whether the rules and constituents the model uses are in line with what we believe to be the actual rules and constituents underlying a particular data set or language.

3.1.1 Testing systematicity

With our *systematicity* test, we aim to pull out that specific aspect, by testing if a model can recombine constituents that have not been seen together during training. In particular, we focus on combinations of words a and b that meet the requirements that (a) the model has only been familiarised with a in contexts excluding b and vice versa but (b) the combination a b is plausible given the rest of the corpus.

3.2 Productivity

A notion closely related to systematicity is *productivity*, which concerns the open-ended nature of natural language: language appears to be infinite, but has to be stored with finite capacity. Hence, there must be some productive way to generate new sentences from this finite storage. While this 'generative' view of language became popular with Chomsky in the early sixties (Chomsky, 1956), Chomsky himself traces it back to Von Humboldt, who expressed that 'language makes infinite use of finite means' (Von Humboldt, 1836).

Both systematicity and productivity rely on the recombination of known constituents into larger compounds. Systematicity postulates that any reconfiguration of understood components results in

3. It is important to note that, while the notions and principles we consider are often used to argue about the compositionality of *languages*, here, our focus lies on evaluating the compositionality of different types of artificial *learners*. The compositionality of our data, which we will discuss in Section 4, we take as given.

an expression that is also readily interpretable. Productivity, on the other hand, does not involve an argument from constituents to larger compounds, but rather in the other direction: if speakers can understand infinitely many complex expressions, this must be because they understand the segments of which they are composed. Whereas systematicity can be empirically established, productivity cannot, as it is not possible to prove that natural languages in fact contain an infinite number of complex expressions. Even if humans' memory allowed them to produce infinitely long sentences, their finite life prevents them from doing so. Productivity of language is therefore more controversial than systematicity.

3.2.1 Testing productivity

To separate systematicity from productivity, in our productivity test we specifically focus on the aspect of unboundedness. We test whether different learners can understand sentences that are *longer* than the ones encountered during training. To test this, we separate sequences in the data based on length and evaluate models on their ability to cope with longer sequences after having been familiarised with the shorter ones.

3.3 Localism

In its basic form, the principle of compositionality states that the meaning of a complex expression derives from the meanings of its consituents and how they are combined, but it does not impose any restrictions on what counts as an admissible way of combining different elements. This is why the principle, taken in isolation, is formally vacuous.[4] As a consequence, the interpretation of the principle of compositionality depends on the strength of the constraints that are put on the semantic and syntactic theories involved. One important consideration concerns – on an abstract level – how *local* the composition operations should be. When these operations are considered to be very local (also referred to as *strong* or *first-level* compositionality), the meaning of a complex expression depends only on its immediate structure and the meanings of its immediate parts (Pagin and Westerståhl, 2010; Jacobson, 2002). In a *global* (or *weak* or *bottom-level* compositionality), the meaning of an expression follows from its total (global) structure and the meanings of its atomic parts.

Carnap (1947) presents an example that nicely illustrates the difference between these two interpretations, in which he considers sentences with tautologies. Under the view that the meaning of declarative sentences is determined by the set of all worlds in which this sentence is true, any two tautologies X and Y are synonymous. Under the local interpretation of compositionality, this entails that also the phrases 'Peter thinks that X' and 'Peter thinks that Y' should be synonymous, which is not necessarily the case, as Peter may be aware of some tautologies but unaware of others. The global interpretation of compositionality does not give rise to such a conflict, as X and Y, despite being identical from a truth-conditional perspective, are not structurally identical. Under this representation, the meanings of X and Y are locally identical, but not globally, if also the phrase they are a part of is considered. As a contrast, consider an arithmetic task, where the outcome of `14 - (2 + 3)` does not change when the subsequence `(2+3)` is replaced by `5`, a sequence with the same (local) meaning, but a different structure.

3.3.1 Testing localism

We test if a model's composition operations are local or global by comparing the meanings it assigns to stand-alone sequences, and those it assigns to the same sequences when they are part of other sequences. More specifically, we compare a model's output when it is given a composed sequence

4. We previously cited Janssen (1983), who proved this claim by showing that arbitrary sets of expressions can be mapped to arbitrary sets of meanings without violating the principle of compositionality, as long as one is not bound to a fixed syntax.

`X`, built up from two parts `A` and `B` with the output the same model gives when it is forced to first separately process `A` and `B` in a local fashion. If the model employs a local composition operation that is true to the underlying compositional system that generated the language, there should be no difference between these two outputs.

3.4 Substitutivity

A principle closely related to compositionality is the principle of *substitutivity*. This principle, that finds its origin in philosophical logic, states that if an expression is altered by replacing one of its constituents with another constituent with the same meaning (a synonym), this does not affect the meaning of the expression (Pagin, 2003). In other words, if a substitution preserves the meaning of the parts of a complex expression, it also preserves the meaning of the whole. In the latter formulation, the correspondence with the principle of compositionality itself can be easily seen: as substituting part of an expression with a synonym changes nor the structure of the expression nor the meaning of its parts, it should not change the meaning of the expression itself either.

Like the principle of compositionality, also the substitutivity principle in the context of natural language is subject to interpretation and discussion. Husserl (1913) pointed out that the substitution of expressions with the same meaning can result in nonsensical sentences if the expressions belong to different semantic categories (the philosopher Geach (1965) illustrated this considering the two expressions *Plato was bald* and *Baldness was an attribute of Plato*, that are synonymous but cannot be substituted in the sentence *The philosopher whose most eminent pupil was Plato was bald*). A second context which poses a challenge for the substitutivity principle concern embedded statements about beliefs. As already sketched out in the previous section, if X and are synonymous, this does not necessarily imply that the expressions *Peter thinks that X* and *Peter things that Y* are both true. In this work, we do not consider these challenging cases, but instead focus on the more general question: do models have a notion of substitutivity.

3.4.1 Testing substitutivity

We test to what extent models represent and learn synonymity by considering how their prediction changes when in an expression one atomic unit is replaced by another atomic unit with the same meaning. We consider two different cases. Firstly, we analyse the case in which synonymous words occur equally often and in comparable contexts. In this case, synonymity can be inferred from the corresponding meanings, but also from a distributional perspective. Secondly, we consider pairs of words in which one of the words occurs only in very short sentences (we will call those *primitive contexts*). In this case, synonymity can only be inferred from the (implicit) semantic mapping, which is identical for both words, but not from the context that those words appear in.

3.5 Overgeneralisation

The previously discussed compositionality arguments are of mixed nature. Some – such as productivity and systematicity – are intrinsically linked to the way that humans acquire and process language. Others – such as substitutivity and localism – are properties of the mapping from signals to meanings. While it can be tested if a language user abides by these principles, these principles themselves do not relate directly to language users. To complete our set of tests to assess whether a model learns compositionally, we include also a notion that exclusively concerns the acquisition of the language by the model: we consider if models exhibit *overgeneralisation* when faced with a *non*-compositional phenomena.

Overgeneralisation (or overregularisation) is a language acquisition term, that refers to the scenario in which a language learner applies a general rule in a case that forms an exception to this rule. One of the most well-known examples, which served also as the subject of the famous *past-tense debate* between symbolism and connectionism (Rumelhart and McClelland, 1986; Marcus et al., 1992), concerns the rule that English past tense verbs can be formed by appending *-ed* to the stem of the verb: during the acquistion of past tense forms, learners infrequently use this rule also for irregular verbs, resulting in forms like *goed* (instead of *went*) or *breaked* (instead of *broke*) .

The relation of overgeneralisation with compositionality comes from the supposed evidence that overgeneralisation errors provide for the presence of symbolic rules in the human language system (see, e.g. Penke, 2012). In this work, we following this line of reasoning and take the application of a rule in a case where this is contradicted by the data provided to a model as evidence that the model in fact internalised this rule. In particular, we regard a model's inclination to apply rules as the expression of a compositional bias. This inclination is most easily observed in the case of exceptions, where the correct strategy is to ignore the rules and learn on a case-by-case basis. If models tend to overgeneralise by applying the rules also to such cases, we hypothesize that this in particular demonstrates compositional awareness.

3.5.1 Testing overgeneralisation

We propose an experimental setup where a model's tendency to overgeneralise is evaluated by monitoring its behaviour on exceptions. We identify samples that do not adhere to the rules underlying the data distribution– *exceptions* – in the training datasets and assess the tendency to overgeneralise by observing how architectures model these exceptions during training: (when) do they consistently follow a global rule set, and (when) do they (over)fit the training samples individually?

4. Data

The main aim of this article is to provide a series of tests to take apart which aspects of compositionality are captured by different types of artificial neural architectures. We will use these tests to evaluate three popular sequence-to-sequence architectures. As observed by many others before, insight in the compositional skills of neural networks is not easily acquired by studying models trained on natural language directly. While it is generally agreed upon that compositional skills are required to appropriately model natural language, successfully modelling natural data requires far more than understanding hierarchical structure. As a consequence, it is sometimes argued that a negative result may stem not from a model's incapability to model compositionality, but rather from the lack of signal in the data that should induce compositional behaviour. A positive result, on the other hand, cannot always be explained as successful compositional learning, since it is difficult to establish that a good performance cannot be reached through heuristics and memorisation.

In this article, we therefore consider a synthetic data set, which does not contain any non-compositional phenomena and for which a precise description of the underlying structure can be formulated. This way, we ensure that compositionality is in fact a salient feature of the data. At the same time, we construct the data such that in other dimensions – such as the lengths of the sentences and depths of the parse trees – the data matches the statistical properties of a natural data set. We revisit the topic of synthetic versus natural data at the end of this article, in Section 8.

4.1 PCFG SET

The task that we consider amounts to translating sequences that are generated by a probabilistic context free grammar (PCFG) into corresponding output sequences, which represent their meanings. Output sequences are constructed from input sequences by recursively applying the *string edit* operations that are specified in the latter. We call this string edit task PCFG SET. In this section, we describe the foundations of PCFG SET, starting with an explanation of its syntax (which defines the admissible input sequences) and ending with its semantics (which concern the corresponding output sequences).

4.2 Input sequences: syntax

The input sequences in PCFG SET contain words for unary and binary functions (such as `append`, `copy`, `reverse`), elements to form the string sequences that these functions can be applied to (such as `A`, `B`, `A1`, `B1`) and a separator to separate the arguments of a binary function (`,`). Input sequences are generated by following a PCFG that contains four non-terminals: the start symbol of the grammar S, a symbol for binary and unary functions F_U and F_B, respectively, and a symbol X from which string sequences can be generated. A full description of this grammar is given in Figure 1 (the production probabilities are not depicted).

Non-terminal rules	
S	$\rightarrow F_U\ S \mid F_B\ S\ ,\ S$
S	$\rightarrow X$
X	$\rightarrow XX$
Lexical rules	
F_U	→ `copy` | `reverse` | `shift` | `echo` | `swap` | `repeat`
F_B	→ `append` | `prepend` | `remove_first` | `remove_second`
X	→ `A` | `B` | ... | `Z` | `A2` | ... | `B2` | ...

Figure 1: The context free grammar that describes the entire space of grammatical input sequences in PCFG SET. The rule probabilities (not depicted) can be used to control the distributional properties of a PCFG SET.

Since the input sequences of PCFG SET are generated by a PCFG, an infinite number of admissible input sequences can be constructed. All such admissible input sequences describe sequences of function calls to string arguments. Because functions can be nested, the parse trees of valid sequences can be arbitrarily deep and long. For instance, consider the following grammatical sequences, which have a depth of 1, 2 and 2, respectively:

```
repeat A B C
echo remove_first D , E F
append swap F G H , repeat I J
```

Despite its strongly hierarchical nature, the syntactic tree of a PCFG SET input sequence is unambiguous given the sequence alone. All sequences thus have a unique parse tree, that is inferrable without the need to add brackets in the input sequence (contrary to, e.g. the arithmetic language described by Hupkes et al., 2018). The distributional properties of a particular PCFG SET dataset can be controlled by adjusting the probabilities of the grammar and varying the number of input characters. We will later use this to create a dataset whose distribution of lengths and depths matches that of a dataset containing natural language.

4.3 Output sequences: semantics

The meaning of a PCFG SET input sequence is constructed by recursively applying the function calls of which the sequence is composed. This mapping is governed by the interpretation functions listed in Figure 2.

The definitions of the interpretation functions are systematic, because they specify the requirement that a sequence must satisfy in order to qualify as the output to an input, without having to enumerate particular input-output pairs. In this sense, PCFG SET differs from a task such as the lookup table task introduced by Liška et al. (2018), where functions must be exhaustively defined

Unary functions F_U:		**Binary functions F_B:**	
`copy` $x_1 \cdots x_n$	$\rightarrow x_1 \cdots x_n$	`append` $\mathbf{x}, \mathbf{y}$	$\rightarrow \mathbf{x}\, \mathbf{y}$
`reverse` $x_1 \cdots x_n$	$\rightarrow x_n \cdots x_1$	`prepend` $\mathbf{x}, \mathbf{y}$	$\rightarrow \mathbf{y}\, \mathbf{x}$
`shift` $x_1 \cdots x_n$	$\rightarrow x_2 \cdots x_n\, x_1$	`remove_first` $\mathbf{x}, \mathbf{y}$	$\rightarrow \mathbf{y}$
`swap` $x_1 \cdots x_n$	$\rightarrow x_n\, x_2 \cdots x_{n-1}\, x_1$	`remove_second` $\mathbf{x}, \mathbf{y}$	$\rightarrow \mathbf{x}$
`repeat` $x_1 \cdots x_n$	$\rightarrow x_1 \cdots x_n\, x_1 \cdots x_n$		
`echo` $x_1 \cdots x_n$	$\rightarrow x_1 \cdots x_n\, x_n$		

Figure 2: The interpretation functions describing how the meaning of PCFG SET input sequences is formed.

because there is no systematic connection between arguments and the values to which functions map them. For PCFG SET, learning the meaning of a function should thus not depend on the particular input sequences to which they are applied in a training set.

Following the rules specified in Figure 2, the three sequences listed above would be mapped to output sequences as follows:

```
repeat A B C                     →   A B C A B C
echo remove_first D , E F        →   E F F
append swap F G H , repeat I J   →   H G F I J I J
```

As argued earlier in this paper, the fact that a dataset is generated by a compositional system does not necessarily imply that succesfully generalising to a particular test set requires knowing this underlying system. Often, a learner may get away with concatenating memorised strings or following another strategy that is unrelated to the compositional rules of the system. With PCFG SET, we aim to create a task for which it should not be possible to obtain a high test accuracy by following alternative strategies. In particular, we assure that the train and test data are linked *only* by implicit systematic rules, by never repeating the same arguments to an input function. As a consequence, memorisation of specific input-output pairs is not beneficial.

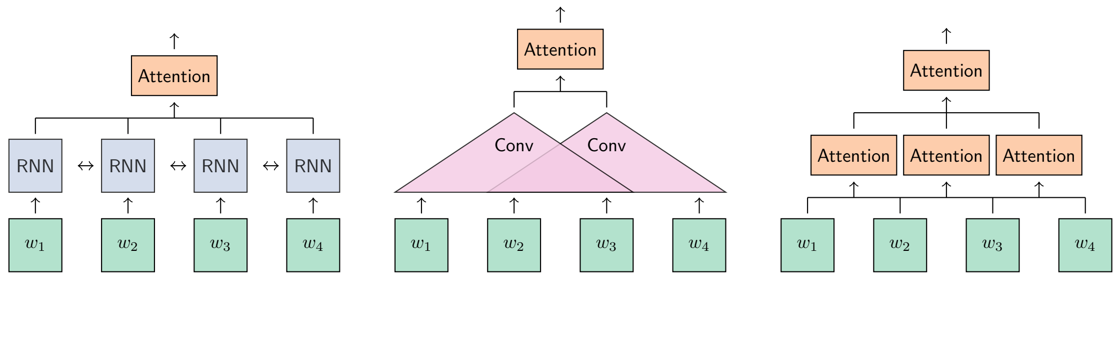
Furthermore, since the accuracy on PCFG SET is directly linked to a model's ability to infer and execute compositional rules, the training signals a model receives during training unequivocally convey that a compositional solution should be found. Thereby, we aim to give models the best possible chance to learn a compositional solution.

5. Architectures

The current state-of-the-art for sequence-to-sequence language processing tasks such as machine translation, speech processing and language understanding boils down to recurrent neural networks (Sutskever et al., 2014), convolutional neural networks (Gehring et al., 2017) and transformer neural networks (Vaswani et al., 2017). In all our experiments, we compare these three popular sequence-to-sequence architectures. In this section, we further motivate the choice for these three architectures and explain their most important features. We also include a brief overview of literature discussing the architectures' abilities to handle compositional data, without claiming to be complet ein this respect.

5.1 LSTMS2S

The first architecture we consider is a recurrent encoder-decoder model with attention. This setup is considered to be the most basic of the three setups we consider, but is (still) the basis of many MT applications (e.g., OpenNMT, Klein et al., 2017) and has also been successful in the fields of speech recognition (e.g. Chorowski et al., 2015) and question answering (e.g. Golub and He, 2016).



(a) LSTMS2S (b) ConvS2S (c) Transformer

Figure 3: High-level graphical depictions of the most important features of the encoding mechanisms in the LSTM2S, ConvS2S and Transformer models. (a) The LSTMS2S processes the input in a fully sequential way, iterating over the embedded elements one by one in both directions before applying an attention layer. (b) The ConvS2S divides the input sequence into local fragments of consecutive items that are processed in the same convolutions, before applying attention. (c) The Transformer immediately applies several global attention layers to the input, without incrementally constructing a preliminary representation.

We consider the version of this model in which both the decoder and encoder are LSTMs and refer to this setup with the abbreviation *LSTMS2S*.

5.1.1 Model basics

The LSTMS2S is a fully recurrent model. The encoder processes the input by iterating over all of its elements in both directions and incrementally constructing a representation for the entire sequence. Upon receiving the encoder output as input, the decoder performs a similar, sequential computation to unroll the predicted sequence. Here, the LSTMS2S uses an attention mechanism, which allows it to focus on the parts of the encoded input that are estimated to be most important at each moment in the decoding process.

The sequential fashion with which the LSTMS2S model processes each input potentially limits the model's abilities to recombine components hierarchically. While depth – and, as shown by Blevins et al. (2018) thus hierarchy – can be created by stacking neural layers, the number of layers in popular recurrent sequence-to-sequence setups tends to be limited. The attention mechanism of the encoder-decoder setup positively influences the skills of LSTMS2S to hierarchically process inputs, as it allows the decoder to focus on input tokens out of the sequential order.

5.1.2 Relation to compositionality

The wealth of literature on the compositionality of recurrent neural networks provides evidence both in favour of and opposing the viewpoint that these networks are compositional learners. Recurrent neural networks have been found capable of processing long distance number agreement and other grammatical phenomena that arguably require hierarchical processing (Gulordava et al., 2018; Jumelet and Hupkes, 2018; Wilcox et al., 2018; Mul and Zuidema, 2019), can to some extent cope with highly compositional arithmetic expressions (Hupkes et al., 2018) and have proven quite successful at generalising to randomly split test data for a variety of artificial compositional languages (Lake and Baroni, 2017; Loula et al., 2018; Liška et al., 2018). Performance deteriorates for setups where the difference between train and test data is more challenging. For instance, recurrent models are known to fail to generalise to inputs that require longer output sequences in sequence-to-sequence tasks (Lake and Baroni, 2017) and to longer inputs in classification tasks (Hupkes et al., 2018). They also struggle to generalise the usage of terms encountered during training to complex new combinations, not only when these terms were only seen in isolation during training (Lake and Baroni, 2017), but also when it concerns well-trained words (Loula et al., 2018; Liška et al., 2018).

5.2 ConvS2S

The second architecture we consider is a convolutional sequence-to-sequence model. We follow the setup described by Gehring et al. (2017) and use their nomenclature: we refer to this model with the abbreviation *ConvS2S*.

5.2.1 Model basics

ConvS2S uses a convolutional model to encode input sequences, instead of a recurrent one. The decoder uses a multi-step attention mechanism – every layer has a separate attention mechanism – to generate outputs from the encoded input representations. As at the core of the ConvS2S model lies the local mechanism of one dimensional convolutions which are repeatedly and hierarchically applied, ConvS2S has a built in bias for creating compositional representations. While the convolutions already contextualise information in a sequential order, the source and target embeddings are also combined with position embeddings that explicitly encode order. Convolutional sequence-to-sequence models have obtained competitive results in the fields of machine translation (Gehring et al., 2016) and abstractive summarisation (Denil et al., 2014).

5.2.2 Relation to compositionality

Dessì and Baroni (2019) contrast the compositional skills of a recurrent sequence-to-sequence model and ConvS2S on the artificial data of Lake and Baroni (2017) and Loula et al. (2018). They argue that, although ConvS2S exhibits improved compositional generalisation, its errors are unsystematic, indicating that the model has not fully mastered any of the systematic rules.

While the topology of ConvS2S – which is biased towards integration of local information – may hinder modelling long-distance relations, convolutional networks have been found to maintain a much longer effective history than their recurrent counterparts (Bai et al., 2018). Within ConvS2S, it is through the multi-step attention and the upper part of the hierarchy that distant portions in the input sequence can be combined. This multi-step attention mechanism improves the generalisation abilities of the model compared to single-step attention (Dessì and Baroni, 2019).

5.3 Transformer

The last model we consider is the recently introduced Transformer model (Vaswani et al., 2017). Transformer models constitute the current state-of-the-art in many machine translation tasks, and becomes increasingly popular also in other domains, such as language modelling.

5.3.1 Model basics

Transformer models use neither RNNs nor convolutions to convert an input sequence to an output sequence. Instead, they are fully based on a multitude of attention mechanisms. Both the encoder and decoder a transformer are composed of a number of feed-forward layers, each containing two sub-layers: a multi-head attention module and a traditional feed-forward layer. In the multi-head attention layers, several attention tensors (the 'heads') are computed in parallel, concatenated and projected. In addition to a self-attention layer, the decoder has another layer, which computes multi-head attention over the outputs of the encoder. Since transformers do not have any inherent notion of sequentiality, the input embeddings are combined with position embeddings, from which the model can infer *order*. For transformer models, the cost of relating symbols that are far apart is thus not higher than relating words that are close together, which – in principle – should ease modelling long distance dependencies. The setup af attention-based stacked layers furthermore makes the architecture suitable for modelling hierarchical structure in the input sequence, that needs not necessarily correspond to the sequential order. On the other hand, the non-sequential nature of the Transformer could be a handicap as well, particularly for relating consecutive portions in the input sequence. Transformer's receptive field is inherently global, which can be challenging in such cases.

5.3.2 Relation to Compositionality

Saxton et al. (2018) find that transformers trained on a dataset of mathematical questions have better algebraic abilities than LSTM architectures trained on the same task. Specifically, Transformer outperforms the LSTM on a set of extrapolation tests that require compositional skills such as generalising to questions involving larger numbers, more numbers or more compositions. However, Saxton et al. (2018) point out that performance deteriorates for questions that require the computation of intermediate values, which indicates that Transformer has not learned algebraic manipulation of values, but instead learned to apply shallow tricks. Tran et al. (2018) contrast the performance of Transformer and an LSTM-based model on a logical inference task using an artificial language, and find the LSTM-based model to generalise better to longer input sequences.

While several studies find Transformer to have limited compositional abilities when trained on artificial data, Transformers trained on natural language data have been found to capture rich syntactical information. Mareček and Rosa (2018) show that Transformers trained to perform machine translation implicitly capture constituency trees. Recent studies discussing Transformer-based language models have succesfully probed these models for a variety of tasks, among which POS labelling, constituency parsing and dependency parsing (Tenney et al., 2019b,a; Lin et al., 2019). While positional information was encoded in the lower Transformer layers, the upper ones captured hierarchical information (Lin et al., 2019). The ability to encode both local and global structure is highly relevant in developing compositional learners. On the one hand, Tran et al. (2018) find LSTM models to have an advantage over Transformer in modelling subject-verb agreement.

6. Experiments

In the previous sections, we have abstractly proposed tests for compositionality, discussed the data for which we will actualise these tests and the models we will put under scrutiny. Now, in this section, we describe in detail our experimental setup. First, in Section 6.1, we explain how we sample sentences from all potential expressions in PCFG SET. We then detail our five tests in relation to this data set (Section 6.2). Lastly, we explain the training procedure for the three different architectures and discuss how we evaluate the results of the experiments (Section 6.3 and 6.4, respectively).

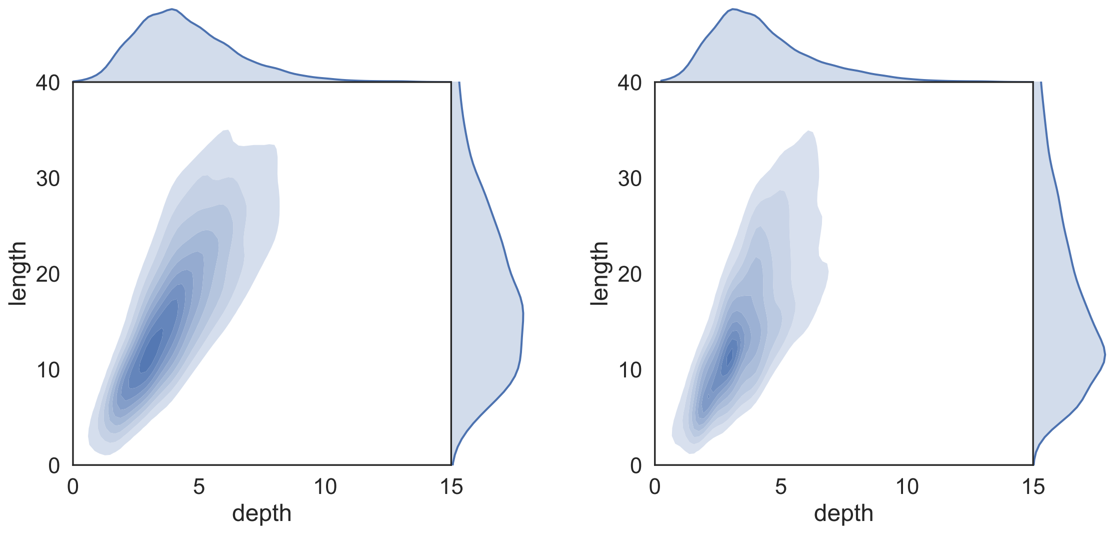
6.1 Data

PCFG SET describes a general framework for producing many different data sets. We describe here the procedure by means of which we selected PCFG SET input-output pairs for our experiments.

6.1.1 Naturalisation of structural properties

The probabilistic nature of the PCFG SET input grammar offers a high level of control over the generated input sequences. We use this control to enforce an input distribution that resembles the statistics of a more natural data set in two relevant respects: the length of the expressions, and the depth of their parse trees.

To obtain the statistics of natural data, we use the English side of a large machine translation corpus: WMT 2017 (Bojar et al., 2017). We automatically annotate this corpus with a statistical



(a) WMT17

(b) PCFG SET data

Figure 4: Distribution of lengths and depths in the PCFG SET (left) and English WMT 2017 test data (right).

parser (Manning et al., 2014) and extract the distribution of length and depths from the annotated corpus. We then use expectation maximisation to tune the PCFG parameters in such a way that the resulting bivariate distribution of the generated data mimics the one extracted from the WMT data. For a more exact description of the naturalisation procedure we refer to Appendix A.

The distributions of the WMT data and a sample of around 10K sentences of the resulting PCFG SET data are plotted in Figure 4a and Figure 4b, respectively.

6.1.2 Sentence selection

We set the size of the string alphabet to 520 and create a base corpus of 10^5 distinct input-output pairs. We use 85% of this corpus for training, 5% for validation and 10% for testing. During data generation, further care is taken to make memorisation as unattractive as possible by controlling the string sequences that feature as primitive arguments in the input expressions: we make sure that the same string arguments are never repeated. While we do not control re-occurence of specific subsequence in general, the relatively large string alphabet makes it highly unlikely that particular subsequences occur often enough to make memorisation a profitable learning strategy.

6.2 Actualisations of compositionality tests

In Figure 5, we depict abstract diagrams for all five compositionality tests that were proposed in Section 3. In the following paragraphs, we detail how these tests are concretised for our data set.

6.2.1 Systematicity

The task accuracy for PCFG SET already reflects whether models are able to recombine functions and input strings that were not seen together during training. In the systematicity test, we focus explicitly on models' ability to interpret pairs of functions that were never seen together while training.

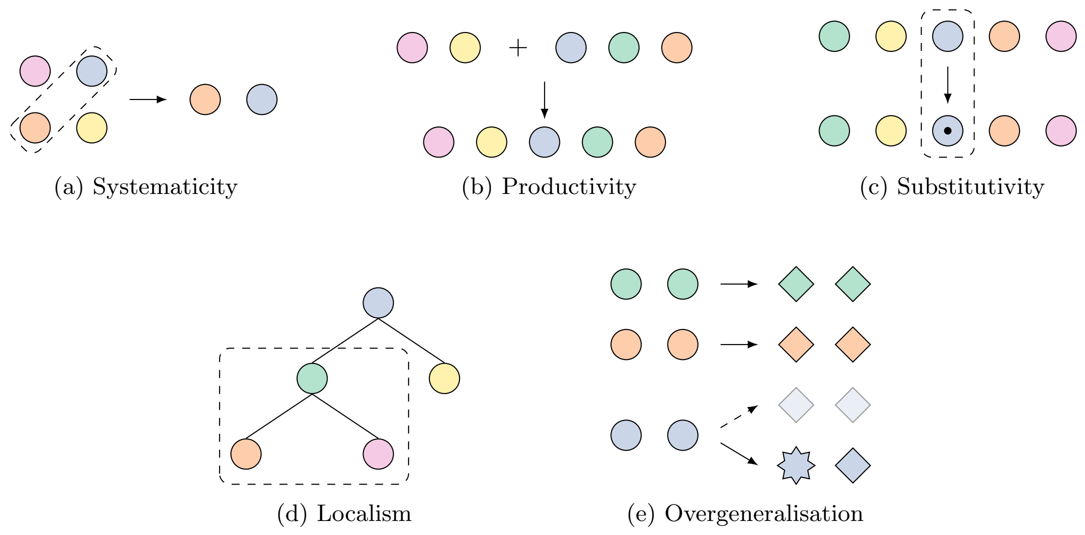


Figure 5: Schematic depictions of the five tests for compositionality proposed in this paper. (a) To test for systematicity, we evaluate models' ability to recombine known parts to form new sequences. (b) While the productivity test also requires recombining known parts, the focus there lies on unboundedness: we test if models can understand sequences *longer* than the ones they were trained on. (c) In the substitutivity test we evaluate how robust models are towards the introduction of synonyms, and, more specifically, in what cases words are considered synonymous by different models. (d) The localism test targets the hierarchical computatation of representations: are smaller subtrees evaluated before larger subtrees? (e) The overgeneralisation task evaluates how likely models are to infer rules: is a model instantly able to accommodate exceptions, or does it need more evidence to deviate from applying the general rule instantiated by the rest of the data?

In particular, we evaluate four pairs of functions: `swap repeat`, `append remove_second`, `repeat remove_second` and `append swap`.[5] We redistribute the training and test data such that the training data does not contain any input sequences including these specific four pairs, and all sequences in the test data contain at least one. After this redistribution, the training set contains 72 thousand input-output pairs, while the test set contains 21 thousand examples. Note that while the training data does not contain any of the function pairs listed above, it still may contain sequences that contain both functions. E.g. `repeat remove_second A B , C D` cannot appear in the training set, but `repeat reverse remove_second A B , C D` might.

Evaluation We evaluate models based on their accuracy on the test data.

6.2.2 Productivity

To test the productive capacity of models, we focus on how well they generalise to sequences that are *longer* than the ones they have seen during training. In particular, we redistribute the PCFG SET training and testing data based on the number of functions. Sequences containing up to eight functions are collected in the training set, consisting of 81 thousand sequences, while input sequences containing at least nine and at most 15 functions are used for evaluation and collected in a test set containing ten thousand sequences. The average, minimum and maximum length, depth and number of functions for the train and test set of the productivity test are shown in Table 1.

5. We keep the specific pairs of functions fixed during evaluation to decrease the number of dimensions of variation: rather than varying the function pairs evaluated across runs, we vary the initialisation and order of presentation of the training examples.

	Depth			**Length**			**#Functions**		
	min	*max*	*avg*	*min*	*max*	*avg*	*min*	*max*	*avg*
Train	1	8	3.9	3	53	16.3	1	8	4.3
Test	4	14	7.8	14	71	31.7	9	15	10.6

Table 1: The average, minimum and maximum length, depth and number of functions for the train and test set of the productivity test

.

Evaluation We evaluate models based on their accuracy on the test set.

6.2.3 Substitutivity

To evaluate how robust models are to substitutions of words with identitcal meanings, we randomly select two binary and two unary functions (`swap`, `repeat`, `append` and `remove_second`), for which we artificially introduce synonyms during training: `swap_syn`, `repeat_syn`, `append_syn` and `remove_second_syn`. As in the systematicity test, we keep those four functions fixed across all experiments, varying only the model initialisation and order of presentation of the training data. The introduced synonyms have the same interpretation functions as the terms they substitute, so they are semantically equivalent to their counterparts. We consider two different conditions, that differ in the syntactic distribution of the synonyms in the training data.

Equally distributed synonyms For the first substitutivity test we randomly replace half of the occurences of the chosen functions F with F_{syn}, while not altering the target. Originally, the individual functions appeared in 39% of the training samples on average. After synonym substitution they appear in approximately 19% of the training samples. In this test, F and F_{syn} are distributionally similar, which should facilitate inferring that they are synonyms.

Primitive synonyms In the second and more difficult substitutivity test, we introduce F_{syn} in *primitive* contexts, where F is the only function call in the input sequence. F_{syn} is introduced in 0.1% of the training set samples, resulting in one appearance of F_{syn} for approximately four hundred occurrences of F. In this *primitive* condition, the function F and its synonymous counterpart F_{syn} are distributionally not equivalent

Evaluation In both cases, we evaluate models based on the interchangeability of F with F_{syn}, rather than measuring whether the output sequences match the target. This evaluation procedure is explained in more detail in Section 6.4.

6.2.4 Localism

In the localism test, we test models' behaviour when a particular subtree $\mathcal{T}$ in an input sequence is replaced with its meaning $o = \mathcal{M}(\mathcal{T})$.[6] If a model builds up the meanings of input sequences incrementally, following the hierarchy that it is dictated by the underlying system, its output meaning should not change as a consequence of such a substitution.

Unrolling computations To compute how locally stable representations are, we compare the output sequence that is generated by the model for a particular input sequence with the output sequence that the model generates when we explicitly unroll the processing of the input sequence. That is, instead of presenting the entire input sequence to the model at once, we force the model to evaluate the outcome of smaller subtrees before computing the outcome of bigger ones, in the

6. Thanks to the recursive nature of PCFG SET expressions, this is a relatively straightforward substitution in our data. We are aware that designing an analogue of this experiment for natural language data would be less trivial.

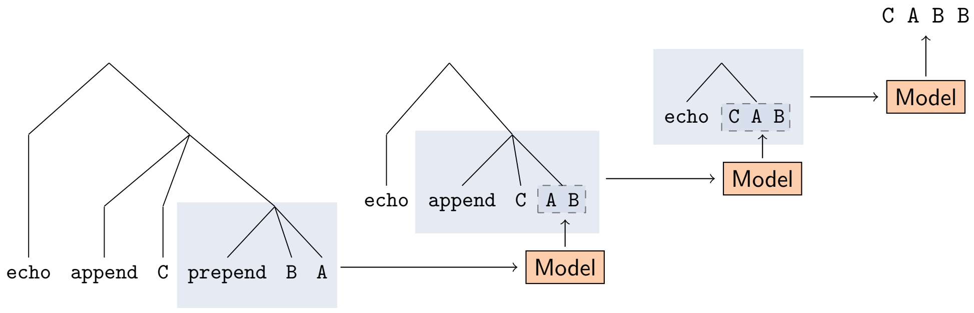


Figure 6: Unrolling the computation of the meaning of a sentence for the localism test. Computing the meaning of the input sequence `echo append C , prepend B , A` requires first computing the meaning o_1 of `prepend B , A`, then applying `append` to `C` and o_1 to obtain o_2 and then lastly computing the final meaning by applying `echo` to o_2. In other words, if a model follows the compositional protocol to predict an output sequence for `echo append C , prepend B , A` , it should predict the same output sequence for `echo C A B`.

following way: we iterate through the syntactic tree of the input sequence and use the model to compute the meanings of the smallest subtrees. We then replace these subtrees by the model's output and use the model to again compute the meanings of the smallest subtrees in this new tree. This process is continued until the meaning for the entire sequence is found. A concrete example is visualized in Figure 6.

To separate a model's ability to generalise to test data from the procedure it follows to compute the meanings of sentences, we conduct the localism test on sentences that were drawn from the training data. We randomly select five thousand sequences from the training set. On average, unrolling the computation of these sequences involves five steps.

Evaluation We evaluate a model by comparing the final output of the enforced recursive method to the output emitted when the sequence is presented in its original form. Crucially, during evaluation we focus on checking whether the two outputs are identical, rather than if they are correct. If a model wrongfully emits `B A` for input sequence `prepend B , A`, this is not penalised in this experiment, provided that the the regular input sequence yields the same prediction as its hierarchical variant. This method matches the previously mentioned **consistency score** that is also used in the substitutivity tests.

6.2.5 Overgeneralisation

In the overgeneralisation experiment, we implicitly target a model's ability and willingness to infer rules, by evaluating if it overgeneralises when faced with exceptions. As the language defined through the PCFG is designed to be strictly compositional, it does not contain exceptions. We therefore manually add them to the data set, which allows us to have a large control over their occurence and frequency.

Exceptions We select four pairs of functions that are assigned a new meaning whenever they appear together in an input sequence: `reverse echo`, `prepend remove_first`, `echo remove_first` and `prepend reverse`. In particular, whenever these functions occur together in the training data, we remap the meaning of the functions involved, as if an alternative set of interpretation functions is used in these few cases. The remapped definitions, which we call *exceptions*, can be found in Table 2. For example, the meaning of `echo remove_first A , B C` would normally be `B C C`, but has now become `A B C`.

Particular combinations of functions are marked as exceptions, and interpreted accordingly whenever they occur. As a consequence, the model has no evidence for the *compositional* interpretation of these function pairs, unless it overgeneralises by applying the rule observed in the rest of the training data.

Input	**Remapped to**	**Target**	
		Original	*Exception*
`reverse echo A B C`	`echo copy A B C`	`C C B A`	`A B C C`
`prepend remove_first A , B , C`	`remove_second append A , B , C`	`C B`	`A B`
`echo remove_first A , B C`	`copy append A , B C`	`B C C`	`A B C`
`prepend reverse A B , C`	`remove_second echo A B , C`	`C B A`	`A B B`

Table 2: Examples for the overgeneralisation test. The input sequences in the data set (*Input*) are usually presented with their ordinary targets (*Original*). In the overgeneralisation test, these input sequences are interpreted according to an alternative rule set (*Remapped to*), effectively changing the corresponding targets (*Exception*).

Exception frequency In our main experiment, the number of exceptions in the data set is 0.1% of the number of occurrences of the least occurring function of the function pair F_1F_2. We present also the results of a gridsearch in which we consider exception percentages of 0.01%, 0.05%, 0.1% and 0.5%.

Evaluation We monitor the accuracy of both the original and the exception targets during training and compare how often the model correctly memorises the exception target and how often it overgeneralises to the compositional meaning despite the evidence in the data. To summarise the model's tendency to overgeneralise, we take the highest overgeneralisation accuracy that is encountered during training. For more qualitative analysis, we visualise the development of both memorisation and overgeneralisation during training, resulting in *overgeneralisation profiles*.

6.3 Training

For every experiment, we perform three runs per model and report both the average and standard deviation of their scores.[7] To decide on the hyperparameters of the three different architectures we consider, we do not perform an extensive gridsearch but rather scour the literature to find the setups that have proved most succesful in the past. The details can be found below.

6.3.1 LSTM2S

We use the LSTMS2S implementation of the OpenNMT-py framework (Klein et al., 2017). We set the hidden layer size to 512, number of layers to 2 and the word embedding dimensionality to 512, matching their best setup for translation from English to German with the WMT 2017 corpus, which we used to shape the distribution of the PCFG SET data. We train all models for 25 thousand iterations, and select the best-performing model based on the performance on the validation set. We use mini-batches of 64 sequences and stochastic gradient descent with an initial learning rate of 0.1.

6.3.2 Conv2S

We use the Conv2S setup that was presented by Gehring et al. (2017). Word vectors are 512-dimensional. Both the encoder and decoder have 15 layers, with 512 hidden units in the first 10

7. Some experiments, such as the localism experiment, do not require to train new models, but can be conducted directly on models trained for other tests.

layers, followed by 768 units in two layers, all using kernel width 3. The final 3 layers are 2048-dimensional.

We train the network with the Fairseq Python toolkit[8]. Unless mentioned otherwise, we use the default hyperparameters of this library. We replicate the training procedure of Gehring et al. (2017), using Nesterov's accelerated gradient method and an initial learning rate of 0.25. We use mini-batches of 64 sentences, with a maximum number of tokens of 3000. The gradients are normalised by the number of non-padded tokens in a batch. We train all models for 25 epochs, or until convergence, as inferred from the loss on the validation set.

6.3.3 Transformer

We use a Transformer model with an encoder and decoder that both contain 6 stacked layers. The multi-head self-attention module has 8 heads, and the feed-forward network has a hidden size of 2048. All embedding layers and sub-layers in the network produce outputs of dimensionality 512. In addition to word embeddings, positional embeddings are used to indicate word order. We use OpenNMT-py[9] (Klein et al., 2017) to train the model according to the guidelines provided by the framework[10]: using the Adam optimiser with $\beta_1 = 0.9$ and $\beta_2 = 0.98$ and a learning rate increasing for the first 8000 'warmup steps' and decreasing afterwards. We train all models for 30 thousand iterations, and select the best-performing model based on the performance on the validation set.

6.4 Evaluation

Throughout our experiments, we consider two performance measures: *accuracy* and *consistency.*

6.4.1 Accuracy

To compute accuracy scores, we consider the correctness of the output sequences the model generates. This is the *sequence accuracy*, where only instances for which the entire output sequence equals the target are considered correct. The accuracy measure is used to evaluate the overall task performance, as well as the systematicity, productivity and overgeneralisation tests. In the rest of this paper, we will denote accuracy scores with *.

6.4.2 Consistency

In some of our tests, we assess models' robustness to meaning-invariant changes in the input sequences, or their computation methods. To evaluate these tests, the most important point is not whether a model correctly predicts the target for a transformed input, but whether its prediction matches the prediction it made before the transformation. To do so, we use a consistency score, which expresses a pairwise equality, where model outputs on two different inputs are compared to each other, instead of to the target output. Also here, only instances for which there is a complete match between the compared outputs are considered correct.

The consistency metric allows us to evaluate compositionality aspects, isolated from task performance. Even for models that may not have a near-perfect task performance and therefore have not mastered the rules underlying the data, we want to evaluate whether they consistently apply and generalise the knowledge they did acquire. We use the consistency score for the substitutivity and localism tests. In the next sections, consistency scores are marked with †.

8. Fairseq toolkit: `https://github.com/pytorch/fairseq`
9. Pytorch port of OpenNMT: `https://github.com/OpenNMT/OpenNMT-py`.
10. Visit `http://opennmt.net/OpenNMT-py/FAQ.html` for the guidelines.

Experiment	**LSTMS2S**	**ConvS2S**	**Transformer**
Task accuracy	0.77 ± 0.01	0.85 ± 0.01	0.93 ± 0.01
Systematicity*	0.51 ± 0.03	0.53 ± 0.01	0.68 ± 0.01
Productivity*	0.29 ± 0.01	0.32 ± 0.02	0.56 ± 0.02
Substitutivity, *equally distributed*†	0.76 ± 0.01	0.96 ± 0.01	0.98 ± 0.00
Substitutivity, *primitive*†	0.61 ± 0.04	0.61 ± 0.03	0.88 ± 0.04
Localism†	0.45 ± 0.01	0.57 ± 0.04	0.56 ± 0.03
Overgeneralisation*	0.73 ± 0.18	0.78 ± 0.12	0.84 ± 0.02

Table 3: General task performance and performance per tests for the PCFG Sequence Edit Task. The results are averaged over three runs and the standard deviation is indicated. Two performance measures are used, where accuracy is indicated by * and consistency by †.

7. Results

In Table 3, we summarise the results of all experiments described in the previous section. Below, we give a detailed account of these results, going test by test.

7.1 Task accuracy

The average task performance on the PCFG SET data for the three different architectures is shown on the first row of Table 3. In terms of task accuracy, the transformer outperforms both LSTM2S and ConvS2S in terms of task accuracy ($p \approx 10^{-6}$ and $p \approx 10^{-3}$, respectively), with a surprisingly high accuracy of 0.93. ConvS2S, in turn, is with its 0.85 accuracy significantly better than LSTM2S ($p \approx 10^{-3}$), which has an accuracy 0.77. The scores of the three architectures are robust with respect to intialisation and order of presentation of the data, as evidenced by the low variation across runs. We now present a breakdown of this task accuracy on different types of subsets of the data. In particular, we study how this accuracy develops with sequence difficulty (measured in terms of length and depth of the sequences and the number of functions they contain) and we compute the difficulty of the different function words in PCFG SET.

7.1.1 Correlation with length and depth

We explore how the accuracy of the three different architectures develops when we increase the difficulty of the sequences. In particular, we look at the impact of the sequence's depth (the maximum level of nestedness observed in a sequence), the sequence's length (number of tokens) and the number of functions in the sequence. In Figure 7, we plot the average accuracy for all three architectures as a function of depth, length and number of functions in the input.

Unsurprisingly, the accuracy of all architecture types decreases with the length, depth and number of functions in the input. All architectures have learned to successfully model sequences with low depths and lengths and a small number of functions (reflected by accuracies close to 1). Their performance drops for longer sequences with more functions. Overall, the architectures show a similar pattern, and the Transformer > Convs2s > LSTMS2S trend is preserved across the different data subsets.

7.1.2 Function difficulty

Since the input sequences typically contain multiple functions, it is not possible to directly evaluate whether some functions are more difficult for models than others. On sequences that contain only

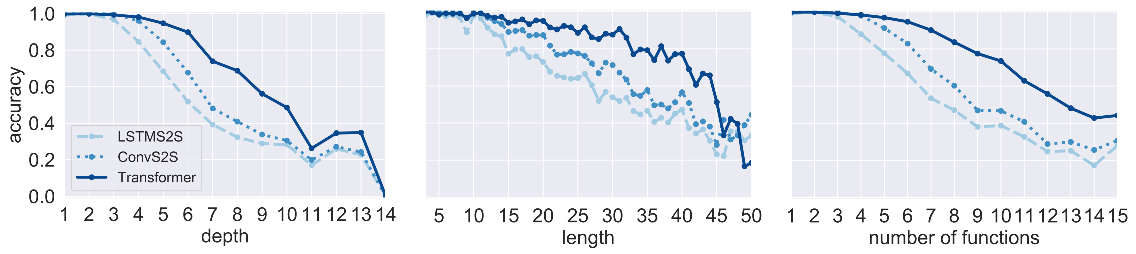


Figure 7: Accuracy of the three models as a function of several properties of the input sequences for the general PCFG SET test set: *depth* of input's parse tree, the input sequence's *length* and the *number of functions* present. The results are averaged over three model runs and computed over ten thousand test samples.

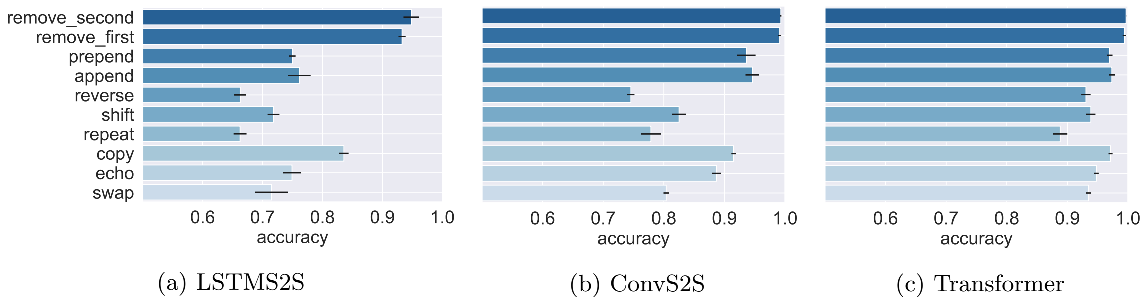


Figure 8: Accuracy of the three models per PCFG SET function, as computed by applying the different functions to the same complex input sequences.

one function, all models achieve a maximum accuracy. To compare the difficulty of the functions, we create a corpus in which all functions are applied to the same composed input sequences and compare the accuracy of different functions on this corpus. For example, to compare the functions `echo` and `reverse`, we apply both functions to the same input sequences, so that e.g. `append swap F G H , repeat I J` yields adapted inputs `echo append swap F G H , repeat I J` and `reverse append swap F G H , repeat I J`. We then compute the accuracy scores obtained on the corresponding outputs. We plot the results in Figure 8.

The ranking of functions in terms of difficulty is similar for all models, suggesting that the difficulties are to a large extent stemming from the objective complexity of the functions themselves, rather than from specific biases in the models. In some cases, it is very clear why. The function `echo` requires copying the input sequence and repeating its last element – regardless of the bias of the model this should be at least as difficult as `copy` which requires just to copy the input. Similarly, `prepend` and `append` require repeating two string arguments, whereas for `remove_first` and `remove_second` only one argument needs to be repeated. The latter functions should thus be easier, irrespective of the architecture. The relative difficulty of `repeat` reflects that generating longer output sequences proves challenging for all architectures. As this function requires repeating the input sequence twice, its output is on average twice as long as the output of another unary function applied to an input string of the same length.

An interesting difference occurs for the function `reverse`. For both LSTMS2S and ConvS2S this is the most difficult function (although closely followed by `repeat` for LSTMS2S). For the Transformer, the accuracy for `reverse` is on par with the accuracies of `echo`, `swap` and `shift`, functions that are substantially easier than `reverse` for the other two architectures. This difference follows directly from architectural differences between the architectures: while LSTMS2S and ConvS2S

Composition	**LSTMS2S**	**ConvS2S**	**Transformer**
`swap repeat`	0.36 ± 0.02	0.44 ± 0.02	0.45 ± 0.00
`append remove_second`	0.50 ± 0.02	0.61 ± 0.01	0.80 ± 0.01
`repeat remove_second`	0.41 ± 0.02	0.50 ± 0.01	0.69 ± 0.01
`append swap`	0.63 ± 0.00	0.42 ± 0.02	0.76 ± 0.00
Average	0.51 ± 0.01	0.53 ± 0.01	0.68 ± 0.01

Table 4: The average sequence accuracy per pair of heldout compositions for the systematicity test.

are forced to encode ordered local context – as they are recurrent or apply local convolutions – the Transformer is not bound to such an ordering and can thus more easily deal with inverted sequences.

7.2 Systematicity

Following the task accuracy, also for the systematicity test, the Transformer model obtains higher scores than both the LSTM and the convolution-based model ($p \approx 10^{-3}$ and $p \approx 10^{-5}$, respectively). The difference between the latter two, however, is for this test statistically insignificant ($p \approx 10^{-1}$). The relative differences between the Transformer model and the other two models gets larger. Where the task accuracies of LSTMS2S and ConvS2S were 83% and 91% of the Transformer accuracy, respectively, for the systematicity test these drop to 75% and 78%, respectively.

7.2.1 Difficulty of different compositions

In Table 4, we show the average accuracies of the three architectures on all four heldout function pairs. All models have trouble composing `swap` and `repeat`, which is unsurprising given the observed relative difficulty of both these functions earlier on (Figure 8). A puzzling observation for the LSTM is the relatively high score for the heldout pair `append swap`. Considering the individual function accuracies, this score is expected to be lower than `append remove_second`, but instead it is substantially higher.

7.2.2 Systematicity vs task accuracy

The large difference between task accuracy and systematicity is surprising, since PCFG SET is constructed such that a high task accuracy requires systematic recombination. A potential explanation for this discrepancy is that, due to the slightly different distribution of the systematicity data set, the models learn a different solution than before. Since the functions occurring in the held-out pairs are slightly undersampled, it could be that the models' representations of these functions are not as good as the ones they develop when trained on the regular data.

A second explanation, to which our localism test will lend more support, is that models do treat the inputs and functions systematically, but analyse the sequences in terms of different units. Obtaining a high accuracy for PCFG SET undoubtedly requires being able to systematically recombine functions and input strings, but it does not necessarily require developing separate representations that capture the semantics of the different functions individually. For instance, if there is enough evidence for `repeat copy`, a model may learn to directly apply the combination of these two functions to an input string, rather than consecutively appealing to separately optimised representations for the two functions. To compute the output of a sequence like `repeat copy swap echo X`, the model may then apply two pairs of functions, instead of four separate functions. Such a strategy would not necessarily harm performance in the overall dataset, since plenty of evidence for all function pairs is present, but it would affect performance on the systematicity test, where this is not the case. We will revisit this hypothesis in Section 7.5.

7.3 Productivity

In Figure 7 we saw that longer sequences are more difficult for all models, even if their length and depth fall within the range of lengths and depths observed in the training examples. There are several potential causes for this drop in accuracy. It could be that longer sequences are simply more difficult than shorter ones: They contain more functions, and there is thus more opportunity to make an error. Additionally, simply because they contain more functions, longer sequences are more likely to contain at least one of the more difficult functions (see Figure 8). Secondly, due to the naturalisation of the distribution of lengths, longer sequences are underrepresented in the training data. Since this reduces the available evidence for longer sequences, models likely still have to extrapolate to infer the meaning of such sequences. Their decrease in performance could thus also be explained by poor extrapolation power.

With our productivity test, we focus purely on the extrapolation aspect, by redistributing the training and testing data so that there is no evidence at all for longer sequences in the training set.[11] The over-all accuracy scores on the productivity test in Table 3 demonstrate that all models have great difficulty with extrapolating to sequences with a higher length than those seen during training. The Transformer drops to a mean accuracy of 0.56; LSTMS2S and ConvS2S have a testing accuracy of 0.29 and 0.32, respectively. Relatively, removing evidence for longer sequences thus resulted in a 62% drop for LSTMS2S and ConvS2S, and a 40% drop for the Transformer. Both in terms of absolute and relative performance, the transformer model thus has a much greater productive potential than the other models, although its absolute performance is still poor.

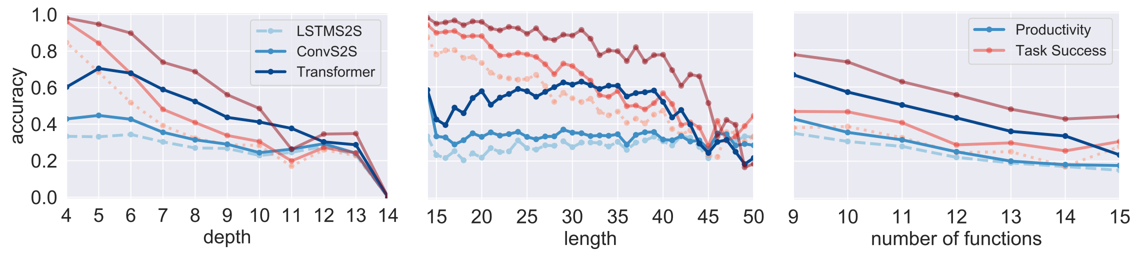


Figure 9: Accuracy of the three models as a function of several properties of the input sequences for the productivity test set: *depth* of the input's parse tree, the input sequence's *length* and the *number of functions* present. The results are averaged over three model runs and computed over ten thousand test samples.

Comparing just the task accuracy and productivity accuracy of models shows that models have difficulty with longer sequences but does still not give a definitive answer about the source of this performance decrease. Since the productivity test set contains on average longer sequences, we cannot see if the drop in performance is caused by poor productive power or by the inherent difficulty of longer sequences. In Figure 9, we show the performance of the three models in relation to depth, length and number of functions of the input sequences (blue lines) compared with the task accuracy of the standard PCFG SET test data for the same lengths as plotted in Figure 7.

For all models, the productivity scores are lower for almost every depth, length and number of functions. As the total number of examples that models are trained on does not change across these two conditions and the absolute difficulty of longer sequence does not either, we thus conclude that the decrease in performance is indeed caused by a lack of evidence for sequences of this length. Without explicit evidence, models are not able to productively generalise to longer sequences. [12]

11. For the details concerning the statistics of the adapted data, we refer back to Table 1.
12. To stop their generation of the answer, models have to explicitly generate an *end of sequence* (`<eos>`) symbol. A reasonable hypothesis concerning the low scores on longer sequences is that they are due to the models inability to postpone the emission of this `<eos>` symbol. We dub this problem the `<eos>-problem`. To test whether the low

7.3.1 Impact of length, depth and number of functions

The depth plot in Figure 7 also provide some evidence for the inherent difficulty of deeper functions: it shows that all models suffer from decreasing test accuracies for higher depths, even if these depths are well-represented in the training data.

When looking at the number of functions, the productivity score of the Transformer is worse than its overall task success for any considered number of functions. The scores for LSTMS2S and ConvS2S are instead very similar to the ones they reached after training on the regular data. This shows that functions with high depths are difficult for LSTMS2S and CONVS2S, even when some of them are included in the training data.

Interestingly, considering only the development of the productivity scores (in blue), it appears that both the LSTMS2S and ConvS2S are relatively insensitive to the increasing length as measured by the number of tokens. Their performance is just as bad for input sequences with 20 or 50 characters, which is on a par with the scores they obtain on the longest sequences after training on the regular data. Apparently, shorter sequences of unseen lengths are as challenging for these models as sequences of extremely long lengths. Later, in the localism experiment, we will find more evidence that this sharp difference between seen an unseen lengths is not accidental, but characteristic for the representations learned by these two types of models.

7.4 Substitutivity

While the previous two experiments were centered around models' ability to recombine known phrases and rules to create new phrases, we now focus on the extent to which models are able to draw analogies between words. In particular, we study under what conditions models might infer that words are *synonyms*.

7.4.1 Equally distributed substitutions

For the substitutivity experiment where words and synonyms are equally distributed, the Transformer and convolutional network perform on par. They both obtain an almost maximum consistency score (0.96 and 0.98, respectively). In Table 5, we see that both architectures put words and their synonyms closely together in the embedding space (column 4 and 7), truly respecting the distributional hypothesis. Surprisingly, the LSTMS2S does not identify that two words are synonyms, even in this relatively simple condition where the words are distributionally identical. Words and synonyms are at very distinct positions in the embedding space (Table 5, column 1), although the distance is smaller than the average between all words in the embedding space (Table 5, column 2). We hypothesise that this low score of the LSTM-based models reflects the architecture's inability to draw the type of analogies required to model PCFG SET data, which is also mirrored in its relatively low overall task accuracy.

7.4.2 Primitive substitutions

The primitive substitutvity test is substantially more challenging than the equally distributed one, since models are only shown examples of synonymous expressions in a small number of primitive contexts. This implies that words and their synonyms are no longer distributionally similar, and that models are provided much less evidence for the meaning of synonyms, as there are simply fewer primitive than composed contexts.

scores are due to early `<eos>` emissions, we computed how many of the wrongly emitted answers were contained in the right answer. For LSTM2S, ConvS2S and Transformer this was true in 20%, 6% and 11% of the cases. These numbers illustrate that the `<eos>`-problem indeed exists, but is not the main source of the poor productive capacity of the different models.

	LSTMS2S			**ConvS2S**			**Transformer**		
Token	ED	P	Other	ED	P	Other	ED	P	Other
`repeat`	0.51	0.59	0.96	0.11	0.36	0.86	0.09	0.36	0.80
`remove_second`	0.32	0.33	0.97	0.16	0.62	0.87	0.07	0.36	0.77
`swap`	0.41	0.36	0.93	0.17	0.36	0.90	0.09	0.40	0.79
`append`	0.32	0.35	0.97	0.12	0.50	0.83	0.07	0.38	0.73
Average	0.39	0.41	0.96	0.14	0.46	0.86	0.80	0.37	0.77
Consistency	**0.76**	**0.61**	-	**0.96**	**0.61**	-	**0.98**	**0.88**	-

Table 5: The average cosine distance between the embeddings of the indicated functions and their synonymous counterparts in the equally distributed (ED) and primitive (P) setups of the substitutivity experiments. For comparison, the average distance from the indicated functions to all other regular function embeddings is given under 'Other'. These distances were very similar across the two substitutivity conditions and are averaged over both.

Consistency and embedding distances While the consistency scores for all models decrease substantially compared to the equally distributed setup, all models – to some extent – still pick up the similarity between a word and its synonym. This is reflected not only in the consistency scores (0.61, 0.61 and 0.88 on average for LSTM, convolution and Transformer based models, respectively), but is also evident from the distances between words and their synonyms, which are substantially lower than the average distances to other function embeddings (Table 5). For the LSTM-based model, the average distance is very comparable to the average distance observed in the equally distributed setup. Its consistency score, however, goes down substantially, indicating that word distances (computed between embeddings) give an incomplete picture of how well models can account for synonymity when there is a distributional imbalance.

Synonymity vs few-shot learning The consistency score of the primitive substutivity test reflects two skills that are partly intertwined: the ability to few-shot learn the meanings of words from very few samples and the ability to bootstrap information about a word from its synonym. As already observed in the equally distributed experiment for the LSTMS2S, it is difficult to draw hard conclusions about a model's ability to infer synonymity when it is not able to infer consistent meanings of words in general. When a model has a high score, on the other hand, it is difficult to disentangle if it achieved this high score because it has learned the correct meaning of both words separately, or because it has in fact understood that the meaning of those words is similar. That is: the consistency score does not tell us whether output sequences are identical because the model knows they should be the *same*, or simply because they are both *correct.* In the equally distributed setup, the low word embedding distances for the ConvS2S and the Transformer strongly pointed to the first explanation. For the primitive setup, the two aspects are more difficult to take apart.

Error consistency To separate a model's ability to few-shot learn the meaning of a word from very few primitive examples and its ability to bootstrap information about synonyms, we compute the consistency score for model outputs that do not match the target output (incorrect outputs). When a model outputs identical but incorrect sequences for two input sequences with a synonym substitution, this cannot be caused by the model merely having correctly learned the meanings of the two words. It can thus be taken as evidence that it treats the word and its synonyms indeed as synonyms.

In Table 6, we show the consistency scores for all output pairs (identical to the scores in Table 3), the breakdown of this score into correct (*consistent correct*) and incorrect (*consistent incorrect*) output pairs, and the ratio of incorrect output pairs that is consistent. The scores in row 2 and

	LSTMS2S	**ConvS2S**	**Transformer**
Consistency score all	0.61 ± 0.04	0.61 ± 0.03	0.88 ± 0.04
Consistent correct	0.54 ± 0.03	0.59 ± 0.02	0.84 ± 0.04
Consistent incorrect	0.06 ± 0.01	0.02 ± 0.00	0.04 ± 0.00
Consistency score across incorrect samples	0.14 ± 0.03	0.05 ± 0.01	0.24 ± 0.07

Table 6: Consistency scores for the primitive substitutivity experiment, expressing pairwise equality for the outputs of synonymous sequences. Along with the overall consistency, we also show the breakdown of this score into correct (*consistent correct*) and incorrect (*consistent incorrect*) pairs, the scores if only correct (*consistent correct*) and incorrect as well as the ratio of consistent output pairs among all incorrect output pairs. A pair is considered incorrect if at least one of its parts is incorrect.

3 show that the larger part of the consistency scores for all models is due to correct outputs. The Transformer maintains its first place, but none of the architectures can be said to treat a word and its synonymous counterpart as true synonyms. An interesting difference occurs between LSTMS2S and ConvS2S, whose consistency scores on all outputs are similar, but quite strongly differ in consistency of erroneous outputs. These scores suggest that the convolution-based architecture is better at few-shot learning than the LSTM-based architecture, but the LSTM-based models are better at inferring synonymity. These results are in line with the embedding distances shown for the primitive substitutivity experiment in Table 5, which are on average also lower for LSTMS2S than for ConvS2S.

7.5 Localism

In the localism test, we investigate if models assign meaning to input sequences in accordance with the hierarchical trees that specify their compositional structure. We compare the output that models generate for regular input sequences with the output they generate when we *unroll* the computation of this output sequence (for an example, see Figure 6).

7.5.1 Consistency scores

None of the evaluated architectures obtains a high consistency score for this experiment (0.45, 0.57 and 0.56 for LSTMS2S, ConvS2S and Transformer, respectively). Also in this test, the Transformer models rank high, but the best-performing architecture is the convolution-based architecture (significant in comparison with the LSTMS2S with $p \approx 10^{-3}$, insignificant in comparison with the Transformer with $p \approx 10^{-1}$). Since the ConvS2S models are explicitly using local operations, this is in line with our expectations.

7.5.2 Input string length

To understand the main cause of the relatively low scores on this experiment, we manually analyse 300 samples (100 per model type), in which at least one mistake was made during the unrolled processing of the sample. We observe that the most common mistakes involve unrolled samples that contain function applications to string inputs with more than five letters. An example of such a mistake would be a model that is able to compute the meaning of `reverse echo A B C D E` but not the meaning of `reverse A B C D E E`. As the outputs for these two phrases are identical, it is clear that this inadequacy does not stem from models' inability to generate the correct output string. Instead, it indicates that the model does not compute the meaning of `reverse echo A B C D E` by consecutively applying the functions `echo` and `reverse`. We hypothesise that, rather, models generate representations for *combinations* of functions that are then applied to the input string at once.

7.5.3 Function representations

While developing 'shortcuts' to apply combinations of functions all at once instead of explicitly unfolding the computation is not necessarily contradicting compositional understanding – imagine, for instance, computing the outcome of the sum `5 + 3 - 3` – the results of the localism experiment do point to another interesting aspect of the learned representations. Since unrolling computations mostly leads to mistakes when the character length of unrolled inputs is longer than the maximum character string length seen during training, it casts some doubt on whether the models have developed consistent function representations.

If a model understands the meaning of a particular function `F` in PCFG SET, it should in principle be able to apply this function to an input string of arbitrary length. Note that, in our case, this ability does not require productivity in generating output strings, since the correct output sequences are not distributionally different from those in the training data (in some cases, they may even be exactly the same). Contrary to other setups, a failure to apply functions to longer sequence lengths can thus not be explained by distributional or memory arguments. Therefore, the consistent failure of all architectures to apply functions to character strings that are longer than the ones seen in training strongly suggests that, while models may have learned to adequately copy strings of length 3 to 5, they do not necessarily consider those operations similar.

To check this hypothesis, we test all functions in a primitive setup where we vary the length of the string arguments they are applied to.[13] For a model that develops several length-specific representations for the same function, we expect the performance to go down abruptly when the input string length exceeds the maximum length seen during training. If a model instead develops a more general representation, it should be able to apply learned functions also to longer input strings. Its performance on longer strings may drop for other practical reasons, but this drop should be more smooth than for a model that has not learned a general purpose representation at all.

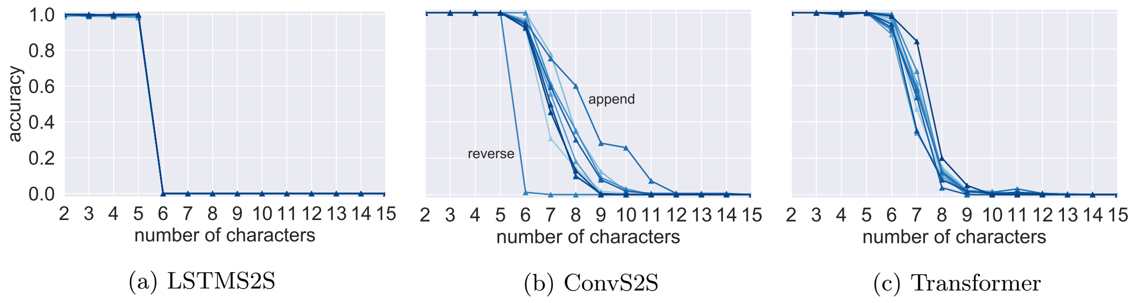


Figure 10: Accuracy of the three architectures on different functions with increasingly long character string inputs. The maximum character string length observed during training is 5. While the Transformer and convolution-based models can, for most functions, generalise a little beyond this string length, the LSTM-based models cannot.

The results of this experiment (plotted in Figure 10) demonstrate that all models have learned to apply all functions to input strings up until length 5, as evidenced by their near-perfect accuracy on the samples of these lengths. On longer lengths, however, none of the models performs well. The performance of all LSTM-based models immediately drops to zero when string arguments exceed length 5, which is the maximum string length seen during training. They do not seem to be able to leverage a general concept of any of the functions. The convolution-based and Transformer model do

13. For binary functions, only one of the two string arguments exceeded the regular argument lengths (1-5).

exhibit some generalisation beyond the maximum string input length seen during training, indicating that their representations are more generally useful.

Their average accuracy reaches zero only for input arguments of more than 10 characters, suggesting that the descending scores may be due to factors of performance rather than competence. The accuracies for Transformer and ConvS2S are comparable for almost all functions, except `reverse`, for which the ConvS2S accuracy drops to zero for length six in all three runs. Interestingly, none of the three architectures suffers from increasing the character length of the first and second argument to `remove_first` and `remove_second`, respectively (not plotted).

7.6 Overgeneralisation

In our last test, we focus on the learning process, rather than on the final solution that is implemented by converged models. In particular, we study if – during training – a model *overgeneralises* when it is presented with an exception to a rule and – in case it does – how many evidence it needs to see to memorise the exception. Whether a model overgeneralises indicates its willingness to prefer rules over memorisation, but while strong overgeneralisation characterises compositionality, more overgeneralisation is not necessarily better. An optimal model, after all, should be able to deal with exceptions as well as with the compositional part of the data.

7.6.1 Overgeneralisation peak

During training, we monitor the number of exception samples for which the model does not generate the correct meaning, but instead outputs the meaning that is in line with the rule instantiated in the rest of the data. At every point in training, we define the strength of the overgeneralisation as the percentage of exceptions for which a model exhibits this behaviour. We call the point in training where the overgeneralisation is highest the *overgeneralisation peak*.

In Table 3, we show the average height of the overgeneralisation peak for all three architectures, using an exception percentage of 0.1%. This quantity equals the accuracy of the model predictions on the input sequences whose outputs have been replaced by exceptions, compared against the outputs that result from following the rules. The numbers in Table 3 illustrate that all models show a rather high degree of overgeneralisation. At some point during the learning process, the Transformer applies the rule to 84% of the exceptions and the LSTMS2S and ConvS2S to 73% and 78% respectively.

7.6.2 Overgeneralisation profile

More interesting than the height of the peak, is the profile that different architectures show during learning. In Figure 7, we plot this profile for 4 different exception percentages. The lower areas (in red), indicate the overgeneralisation strength, whereas the memorisation strength – the accuracy of the model on the adapted outputs, that can only be learned by memorisation – is indicated in the upper part of the plots, in blue. The grey area in between indicates the percentage of exception examples for which the model outputs neither the correct answer, nor the rule-based answer.

Exception percentage The profiles show that, for all architectures, the degree of overgeneralisation strongly depends on the number of exceptions present in the data. All architectures show overgeneralisation behaviour for exception percentages lower than 0.5% (first three rows), but hardly any overgeneralisation is observed when 0.5% of a function's occurrence is an exception (bottom row). When the percentage of exceptions becomes too low, on the other hand, all architectures have difficulties memorising them at all: when the exception percentage is 0.01% of the overall function occurence, only the convolution-based architecture can memorise the correct answers to some extent (middle column, top row). The LSTMS2S and Transformer keep predicting the rule-based output for the sequences containing exceptions, even after convergence.

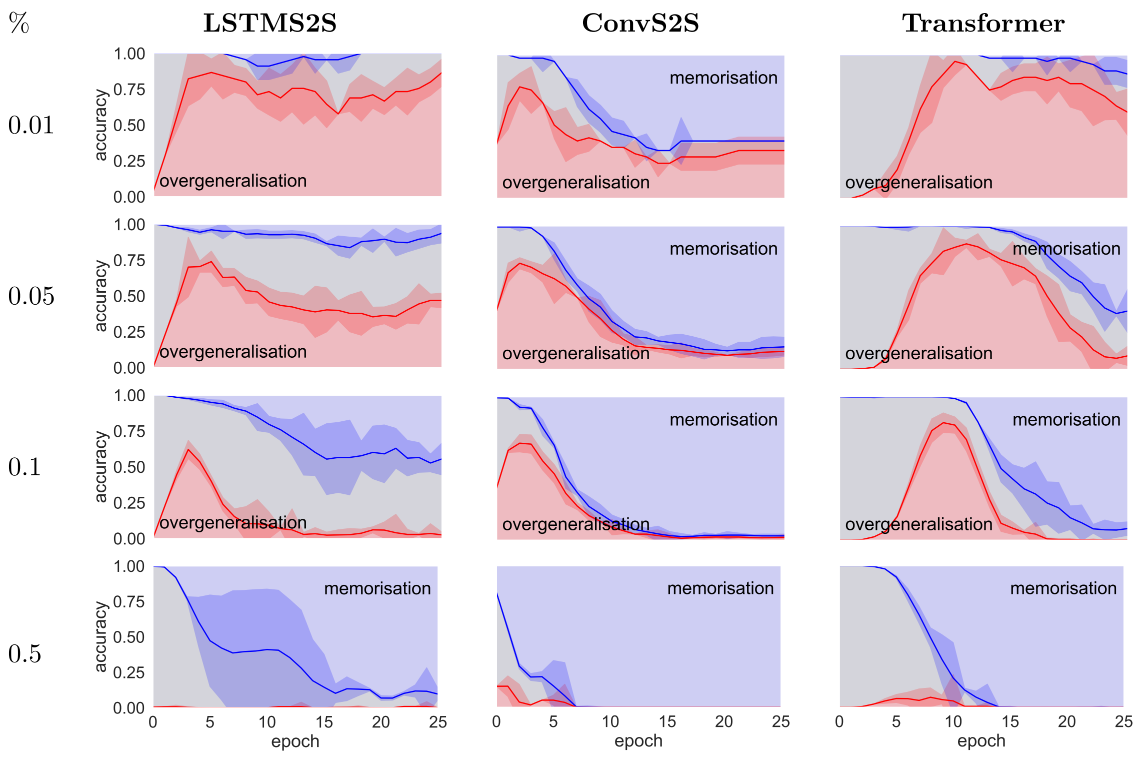


Table 7: Overgeneralisation profiles over time for LSTMS2S, ConvS2S and Transformer for exception percentages of 0.01%, 0.05%, 0.1% and 0.5% (in increasing order, from top to bottom row). The lower area of the plot, in red, indicates the mean fraction of exceptions (with standard deviation) for which an overgeneralised output sequence is predicted (i.e., not the 'correct' exception output for the sequence, but the output that one would construct following the meaning of the functions as observed in the rest of the data). We denote this area as 'overgeneralisation'. The upper area, in blue, indicates the mean fraction of the exception sequences (with standard deviation) for which the model generates the true output sequence, which – as it falls outside of the underlying compositional system – has to be memorised. We call this the 'memorisation' area. The grey area in between corresponds to the cases in which a model does not predict the correct output, nor the output that would be expected if the rule were applied.

Learning an exception The LSTM-based models, in general, appear to find it difficult to accommodate both rules and exceptions at the same time. The Transformer and convolution-based model overgeneralise at the beginning of training, but then, once enough evidence for the exception is accumulated, gradually change to predicting the correct output for the exception sequences. This behaviour is most strongly present for ConvS2S, as evidenced by the thinness of the grey stripe separating the red and the blue area during training. For the LSTM-based models, on the other hand, the decreasing overgeneralisation strength is not matched by an increasing memorisation strength. After identifying that a certain sequence is not following the same rule as the rest of the corpus, the LSTM does not predict the correct meaning, but instead starts generating outputs that match neither the correct exception output, nor the original target for the sequence. After convergence, its accuracy on the exception sequences is substantially lower than the overall corpus accuracy. As the bottom plot (with an exception percentage of 0.5%) indicates that the LSTM-based models do not have problems with learning exception percentages per se, they appear to struggle with hosting exceptions for function words if little evidence for such anomalous behaviour is present in the training data.

8. Discussion

With the rising successes of models based on deep learning, evaluating the compositional skills of neural network models has attracted the attention of many researchers. Many empirical studies have been presented that evaluate compositionality of neural models in different ways, but they have not lead to consensus about whether neural models can in fact adequately model compositional data. We argue that this lack of consensus stems from a deeper issue than the results of the proposed tests: while many researchers have a strong intuition about what it means for a model to be compositional, there is no explicit agreement on what defines compositionality and how it should be tested for in neural networks.

In this paper, we proposed an evaluation framework that addresses this problem, with a series of tests for compositionality that we explicitly motivate by theoretical literature about compositionality. Our evaluation framework contains five different tests, that consider complementary aspects of compositionality that have been frequently mentioned in the literature and allow to tests for these aspects independently. In particular, we describe to test if (i) models systematically recombine known parts and rules (*systematicity*) (ii) models can extend their predictions beyond the length they have seen in the training data (*productivity* (iii) models' composition operations are local or global (*localism*) (iv) models' predictions are robust to synonym substitutions (*substitutivity*) and (v) models favour rules or exceptions during training (*overgeneralisation*). We formulate these tests on a task-independent level, disentangled from a specific down-stream task. With this, we offer a versatile evaluation paradigm which is able to grade compositional abilities of a model on five different levels, that can be instantiated for any chosen sequence-to-sequence task.

To show-case this evaluation paradigm, we instantiate the five tests on a highly compositional artificial dataset called PCFG SET: a sequence-to-sequence transduction problem which requires to recursively apply string edit operations on sequence generated by a probabilistic context free grammar. This dataset is designed such that modelling it adequately requires a fully compositional solution but is generated such that its length and depth distributions match those of a natural corpus of English. By using an artificial data set that is entirely explainable in terms of compositional phenomena, we can focus on the compositional capabilities of different models in the face of compositional data, without having to engage with the noisiness of natural data or quantify the level of compositionality of natural language.

We used our tests to evaluate three popular sequence-to-sequence architectures: an LSTM-based, a convolution-based and an all-attention model. Our experiments showed how all three sequence-to-sequence models are capable, to a limited extent, of compositional processing. Performance varied across tests, with Transformer coming out as the best model overall. DH: Here I think we should include some more interesting hooks to the results, and the differences between different models, will write this after rereading the results. Maybe say also something about that it is somewhat surprising that models do so well on the dataset that we designed to be only solved by compositional behaviour, but still fail the other tests and that this further reinforces the need for the test-suite that we provided? In particular, networks struggled with systematicity, productivity, and localism. For systematicity and productivity, we confirmed the shortcomings found by Lake and Baroni (2018) and Bahdanau et al. (2018) for LSTM-based sequence-to-sequence models, and expanded their investigation to convolutional-based and Transformer models. We showed how the latter mark a significant improvement in both testing conditions, but are not proficient yet. In the localism test, the convolutional-based architecture scored slightly better than Transformer, although this was not entirely a surprise, given its hardwired local convolution operations. It is remarkable that Transformer reached a comparable performance with no explicit hardwiring.

DH: Points still to be made:

- We should say something that relates to our earlier point that with these tests we can figure out how compositional a model needs to be to model compositional data. Something like: these results show that even for a dataset that is completely compositional, it seems that solutions are possible that give a high accuracy but do not necessarily capture all that we consider "compositional"
- Argue that these tests can be used for future models, say that we made all our code available so that people can test their own models, investigate if hyperparameters have an impact and how compositionality is learned throughout training.
- Say something about the artificial vs natural data issue. I imagine that the point would be something like (i) we instantiated these tests on artificial data, this now gives us an idea how these models do when they are presented with data of which one of the most salient aspects is that it is compositional (ii) Similar tests can also be formulated on natural data. Here we could use the knowledge that we already gained from the fact that we actually tried to do this, we could briefly sketch it (iii) Comparing the two results would give us valuable information not only about the models, but also about the level of compositionality of the data itself?

EB: While testing neural networks using a synthetic dataset allowed us to isolate compositional processing from other signals which can found in more realistic dataset, it remains the question of how much of each of the compositional traits we identified is expressed and can be exploited by networks in natural language data. As future work, we plan to instantiate our tests in natural language domains such as translation and summarisation.

DH: We need some final strong sentences. With this work, we hope to have contributed to…

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron Courville. Systematic generalization: What is required and can it be learned? *ICLR*, 2018.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

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*, pages 1183–1193. Association for Computational Linguistics, 2010.

Terra Blevins, Omer Levy, and Luke Zettlemoyer. Deep rnns encode soft hierarchical syntax. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 14–19, 2018.

Ondrej Bojar, Christian Buck, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno-Yepes, Philipp Koehn, and Julia Kreutzer, editors. *Proceedings of the Second Conference on Machine Translation, WMT 2017, Copenhagen, Denmark, September 7-8, 2017*, 2017. Association for Computational Linguistics. ISBN 978-1-945626-96-8. URL `http://aclanthology.info/volumes/proceedings-of-the-second-conference-on-machine-translation`.

Samuel R Bowman, Christopher D Manning, and Christopher Potts. Tree-structured composition in neural networks without tree-structured architectures. In *Proceedings of the 2015th International*

Conference on Cognitive Computation: Integrating Neural and Symbolic Approaches-Volume 1583, pages 37–42. CEUR-WS. org, 2015.

Rudolf Carnap. *Meaning and necessity: a study in semantics and modal logic*. University of Chicago Press, 1947.

Noam Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.

Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Advances in neural information processing systems*, pages 577–585, 2015.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Stephen Clark. Vector space models of lexical meaning. *The Handbook of Contemporary semantic theory*, pages 493–522, 2015.

Misha Denil, Alban Demiraj, Nal Kalchbrenner, Phil Blunsom, and Nando de Freitas. Modelling, visualising and summarising documents with a single convolutional neural network. *arXiv preprint arXiv:1406.3830*, 2014.

Roberto Dessì and Marco Baroni. Cnns found to jump around more skillfully than rnns: Compositional generalization in seq2seq convolutional networks. *arXiv preprint arXiv:1905.08527*, 2019.

Katrin Erk. Vector space models of word meaning and phrase meaning: A survey. *Language and Linguistics Compass*, 6(10):635–653, 2012.

Jerry A Fodor and Zenon W Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1-2):3–71, 1988.

Jonas Gehring, Michael Auli, David Grangier, and Yann N Dauphin. A convolutional encoder model for neural machine translation. *arXiv preprint arXiv:1611.02344*, 2016.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*, 2017.

Yoav Goldberg. Assessing bert's syntactic abilities. *arXiv preprint arXiv:1901.05287*, 2019.

Christoph Goller and Andreas Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 1, pages 347–352. IEEE, 1996.

David Golub and Xiaodong He. Character-level question answering with attention. *arXiv preprint arXiv:1604.00727*, 2016.

Kristina Gulordava, Piotr Bojanowski, Edouard Grave, Tal Linzen, and Marco Baroni. Colorless green recurrent networks dream hierarchically. In *Proceedings of NAACL*, pages 1195–1205, New Orleans, LA, 2018.

Julia Hirschberg and Christopher D Manning. Advances in natural language processing. *Science*, 349(6245):261–266, 2015.

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

Dieuwke Hupkes, Sara Veldhoen, and Willem Zuidema. Visualisation and 'diagnostic classifiers' reveal how recurrent and recursive neural networks process hierarchical structure. *Journal of Artificial Intelligence Research*, 61:907–926, 2018.

Dieuwke Hupkes, Anand Singh, Kris Korrel, German Kruszewski, and Elia Bruni. Learning compositionally through attentive guidance. 2019.

Edmund Husserl. Logische untersuchungen. 1913.

Pauline Jacobson. The (dis) organization of the grammar: 25 years. *Linguistics and Philosophy*, 25 (5):601–626, 2002.

Theo Janssen. *Foundations and applications of Montague grammar.* Mathematisch Centrum, 1983.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 1988–1997. IEEE, 2017.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

Jaap Jumelet and Dieuwke Hupkes. Do language models understand anything? on the ability of lstms to understand negative polarity items. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 222–231, 2018.

Dimitri Kartsaklis. Compositional operators in distributional semantics. *Springer Science Reviews*, 2(1-2):161–177, 2014.

Yoon Kim, Alexander M Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis. Unsupervised recurrent neural network grammars. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1105–1117, 2019.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. Opennmt: Open-source toolkit for neural machine translation. In Mohit Bansal and Heng Ji, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL), System Demonstrations*, pages 67–72. Association for Computational Linguistics, 2017.

Kris Korrel, Dieuwke Hupkes, Verna Dankers, and Elia Bruni. Transcoding compositionally: using attention to find more generalizable solutions. *BlackboxNLP workshop, ACL 2019*, 2019.

Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *35th International Conference on Machine Learning, ICML 2018*, pages 4487–4499. International Machine Learning Society (IMLS), 2018.

Brenden M. Lake and Marco Baroni. Still not systematic after all these years: On the compositional skills of sequence-to-sequence recurrent networks. *CoRR*, abs/1711.00350, 2017.

Phong Le and Willem Zuidema. The forest convolutional network: Compositional distributional semantics with a neural chart and without binarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1155–1164, 2015.

Yongjie Lin, Yi Chern Tan, and Robert Frank. Open sesame: Getting inside bert's linguistic knowledge. *arXiv preprint arXiv:1906.01698*, 2019.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. Assessing the ability of lstms to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535, 2016. ISSN 2307-387X. URL `https://transacl.org/ojs/index.php/tacl/article/view/972`.

Adam Liška, Germán Kruszewski, and Marco Baroni. Memorize or generalize? searching for a compositional rnn in a haystack. *arXiv preprint arXiv:1802.06467*, 2018.

Joao Loula, Marco Baroni, and Brenden M Lake. Rearranging the familiar: Testing compositional generalization in recurrent networks. *arXiv preprint arXiv:1807.07545*, 2018.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014. URL `http://www.aclweb.org/anthology/P/P14/P14-5010`.

Gary F Marcus. *The algebraic mind: Integrating connectionism and cognitive science.* MIT press, 2003.

Gary F Marcus, Steven Pinker, Michael Ullman, Michelle Hollander, T John Rosen, Fei Xu, and Harald Clahsen. Overregularization in language acquisition. *Monographs of the society for research in child development*, pages i–178, 1992.

David Mareček and Rudolf Rosa. Extracting syntactic trees from transformer encoder self-attentions. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 347–349, 2018.

George A Miller and Walter G Charles. Contextual correlates of semantic similarity. *Language and cognitive processes*, 6(1):1–28, 1991.

Jeff Mitchell and Mirella Lapata. Vector-based models of semantic composition. *proceedings of ACL-08: HLT*, pages 236–244, 2008.

Mathijs Mul and Willem Zuidema. Siamese recurrent networks learn first-order logic reasoning and exhibit zero-shot compositional generalization. *arXiv preprint arXiv:1906.00180*, 2019.

Peter Pagin. Communication and strong compositionality. *Journal of Philosophical Logic*, 32(3): 287–322, 2003.

Peter Pagin and Dag Westerståhl. Compositionality i: Definitions and variants. *Philosophy Compass*, 5(3):250–264, 2010.

Barbara Partee. Lexical semantics and compositionality. *An invitation to cognitive science: Language*, 1:311–360, 1995.

Martina Penke. The dual-mechanism debate. In *The Oxford handbook of compositionality.* 2012.

Steven Pinker. *Language learnability and language development.* Cambridge, MA: Harvard University Press, 1984.

Christopher Potts. A case for deep learning in semantics: Response to pater. *Language*, 2019.

D. E. Rumelhart and J. L. McClelland. *Parallel distributed processing: explorations in the microstructure of cognition*, volume 2, chapter On learning the past tenses of English verbs, pages 216–271. MIT Press, Cambridge, 1986.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. 2018.

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*, pages 1–9, 2010.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems (NIPS)*, pages 3104–3112, 2014.

Zoltan Szabó. The case for compositionality. *The Oxford handbook of compositionality*, 64:80, 2012.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline. *arXiv preprint arXiv:1905.05950*, 2019a.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R Bowman, Dipanjan Das, et al. What do you learn from context? probing for sentence structure in contextualized word representations. In *Proceedings of the 7th International Conference on Learning Representations (ICLR2019)*, 2019b.

Ke Tran, Arianna Bisazza, and Christof Monz. The importance of being recurrent for modeling hierarchical structure. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4731–4736, 2018.

Peter D Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37:141–188, 2010.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

Sara Veldhoen, Dieuwke Hupkes, and Willem Zuidema. Diagnostic classifiers: Revealing how neural networks process hierarchical structure. In *Pre-Proceedings of the Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches (CoCo @ NIPS 2016)*, 2016.

Jesse Vig and Yonatan Belinkov. Analyzing the structure of attention in a transformer language model. *arXiv preprint arXiv:1906.04284*, 2019.

Wilhelm Von Humboldt. On language: The diversity of human language-structure and its influence on the mental development of mankind. 1836.

Ethan Wilcox, Roger Levy, Takashi Morita, and Richard Futrell. What do RNN language models learn about filler–gap dependencies? In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 211–221, 2018.

Thomas Wolf. Some additional experiments extending the tech report assessing berts syntactic abilities by yoav goldberg. Technical report, Technical report, 2019.

Wlodek Zadrozny. From compositional to systematic semantics. *Linguistics and philosophy*, 17(4): 329–342, 1994.

Appendix A. Naturalisation of artificial data

The artificially generated PCFG SET data are transformed so as to mimic the distribution of a natural language data set according to the following procedure:

1. Use a natural language data set $\mathcal{D}_N$, define a set of features F, and for each $f \in F$, compute the value $f(s)$ for each sentence $s \in \mathcal{D}_N$.
2. Generate a large sample $\mathcal{D}_R$ of PCFG SET data using random probabilities on production rules for each instance.
3. Transform $\mathcal{D}_R$ as follows:
   (i) For each feature $f \in F$, specify a *feature increment* i_f.
   (ii) For each $s \in \mathcal{D}_N$, compute the *partitioning vector* $v(s)$, which is the concatenation of the values $\lfloor f(s)/i_f \rfloor$ for each feature $f \in F$.
   (iii) Partition $\mathcal{D}_N$ into subsets by clustering instances with the same partitioning vector. For any such subset $\mathcal{D}_N^i$, let $v(\mathcal{D}_N^i)$ denote the partitioning vector of its members. And for any partitioning vector $\mathbf{v}$, let $v_N^{-1}(\mathbf{v})$ denote the subset $\mathcal{D}_N^i \subseteq \mathcal{D}_N$ whose members have partitioning vector $\mathbf{v}$ (so that $v(\mathcal{D}_N^i) = \mathbf{v}$).
   (iv) Of the identified subsets, determine the largest set $\mathcal{D}_N^i \subseteq \mathcal{D}_N$. Call this set $\mathcal{D}_N^{\max}$.
   (v) Partition $\mathcal{D}_R$ in the same way as $\mathcal{D}_N$, yielding subsets $\mathcal{D}_R^i$. Let the subset $\mathcal{D}_R^i$ such that $v(\mathcal{D}_R^i) = v(\mathcal{D}_N^{\max})$ be $\mathcal{D}_R^{\max}$.
   (vi) Initialise an empty set $\mathcal{D}'_R$.
   (vii) Of each $\mathcal{D}_R^i$, randomly pick $\frac{|v_N^{-1}(v(\mathcal{D}_R^i))| \times |\mathcal{D}_R^{\max}|}{|\mathcal{D}_N^{\max}|}$ members, and assign them to $\mathcal{D}'_R$.
   (viii) If necessary, repeat (i) - (vii) for different feature increments f_i. For n features, fit an n-variate Gaussian to each of the transformed sets $\mathcal{D}'_R$. Choose the set with the lowest Kullback-Leibler divergence from the n-variate Gaussian approximation of $\mathcal{D}_N$.
4. Use maximum likelihood estimation to estimate the PCFG parameters of $\mathcal{D}'_R$ and generate more PCFG SET data using these parameters.
5. If necessary, apply step 3 to the data thus generated.

Appendix B. Additional results

B.1 Overgeneralisation

B.2 Substitutivity

Token	**LSTMS2S**			**ConvS2S**			**Transformer**		
	ED	P	Other	ED	P	Other	ED	P	Other
`repeat`	0.512	0.585	0.964	0.111	0.364	0.856	0.087	0.362	0.796
`remove_second`	0.319	0.333	0.968	0.162	0.627	0.868	0.074	0.355	0.771
`swap`	0.405	0.363	0.938	0.172	0.355	0.891	0.085	0.399	0.788
`append`	0.321	0.349	0.969	0.124	0.499	0.834	0.071	0.377	0.731

Table 8: The average cosine distance between the embeddings of the indicated functions and their synonymous counterparts in the equally distributed (ED) and primitive (P) setups of the substitutivity experiments. For comparison, the average distance to all other function embeddings is given under 'Other'.