The compositionality of neural networks: integrating symbolism and connectionism

Authors

Abstract

Despite a multitude of empirical studies, little consensus exist on whether neural networks are able to generalise *compositionally*, a controversy that, in part, stems from a lack of agreement about what it means for a neural model to be compositional. As a response to this controversy, we present a set of tests that provide a bridge between, on the one hand, the vast amount of linguistic and philosophical theory about compositionality and, on the other, the successful neural models of language. We collect different interpretations of compositionality and translate them into five theoretically grounded tests that are formulated on a task-independent level. In particular, we provide tests to investigate (i) if models systematically recombine known parts and rules (ii) if models can extend their predictions beyond the length they have seen in the training data (iii) if models' composition operations are local or global (iv) if models' predictions are robust to synonym substitutions and (v) if models favour rules or exceptions during training. To demonstrate the usefulness of this evaluation paradigm, we instantiate these five tests on a highly compositional dataset which we dub PCFG SET and apply the resulting tests to three popular sequence-to-sequence models: a recurrent, a convolution based and a transformer model. We provide an in depth analysis of the results, that uncover the strengths and weaknesses of these three architectures and point to potential areas of improvement.

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 that use 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 recurrent 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., 2015). 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). While the architectural design of this class of models is not motivated by knowledge about linguistics or human processing, 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. 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 in their evalation. It remains, 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 (e.g. Potts, 2019). 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 learning *compositional* solutions, a question that recently, with the rise of their success, has attracted the attention of several 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 learn and 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 three-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. Secondly, to demonstrate the usefulness of this test suite, we apply our tests to three popular sequence-to-sequence models: a recurrent, a convolution based and a transformer model. We provide an in depth analysis of the results, uncovering interesting strengths and weaknesses of these three architectures. Lastly, 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 studies that evaluate the representations of pre-trained models. In Section 3, we give a theoretical explanation of the five notions that we for which we devise tests, 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 further 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 focus on assessing the representations that are learned by models trained on some independent (often natural) data set.

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 and mathematical reasoning

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.

More recently, Saxton et al. (2019) released another data set in which maths was used to probe the compositional generalisation skills of neural networks. They compare Transformers and LSTM architectures trained on a data set of mathematical questions and find that the Transformer models generalise better than the LSTM models. 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, performance deteriorates for questions that require the computation of intermediate values, which Saxton et al. (2019) reason indicates that the model has not truly learned to treat the task in a compositional manner, but instead applies shallow tricks.

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 by Loula et al. (2018), 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"). 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. Very recently, SCAN was also used to diagnose convolutional networks. In comparison with recurrent networks Dessì and Baroni (2019) find that convolutional networks exhibit improved compositional generalisation skills, but their errors are unsystematic, indicating that the model did not fully master any of the systematic rules.

2.1.3 Lookup tables

Liška et al. (2018) 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) propose 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. Tran et al. (2018) report similar findings, and furthermore show that while a transformer encoder performs similar to an LSTM model when the entire data set is used, an LSTM model generalises better when smaller training data is used.

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

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.

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 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 of (Goldberg, 2019; Lin et al., 2019). More mixed results are reported by others (Tran et al., 2018; Wolf, 2019).

2.2.2 Syntax in Machine Translation

Another subfield of natural language processing in which learned neural representations are heavily studied is machine translation (MT). Analyses in this line of work typically consider which properties are encoded by MT models, with a specific focus on the difference between the representations within layers that situated at different levels of the hierarchy of a model. A robust finding from such analyses is that features such as syntactic constituents, part-of-speech tags and dependency edges can be reliably predicted from the hidden representations of both RNNs (Shi et al., 2016; Belinkov et al., 2017; Blevins et al., 2018) and Transformer models (Raganato and Tiedemann, 2018; Tenney et al., 2019b). Generally, lower level features are encoded in lower layers, while higher level syntactic and semantic features are better represented in deeper layers (e.g. Blevins et al., 2018; Tenney et al., 2019a). For Transformer models, a recent wave of papers demonstrates that such features can also be extracted from the attention patterns (Vig and Belinkov, 2019; Mareček and Rosa, 2018; Lin et al., 2019). While these results do not straightforwardly extend to the compositional scenario that we are interested in in this work, they do demonstrate that both recurrent and attention-based models trained in a setup similar to the one considered for this work are able to capture the types of higher level syntactic features that are often considered to be key for compositional behaviour.

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 approxi-

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.

mations 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 benchmark 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 propose, on a high level, how they can be tested. In particular, we propose to test (i) if models systematically recombine known parts and rules (*systematicity*) (ii) if models can extend their predictions beyond the length they have seen in the training data (*productivity*) (iii) if models' composition operations are local or global (*localism*) (iv) if models' predictions are robust to synonym substitutions (*substitutivity*) and (v) if models favour rules or exceptions during training (*overgeneralisation*). Below, we describe the theory that motivated us to select these aspects, that are schematically depicted in Figure 1. 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

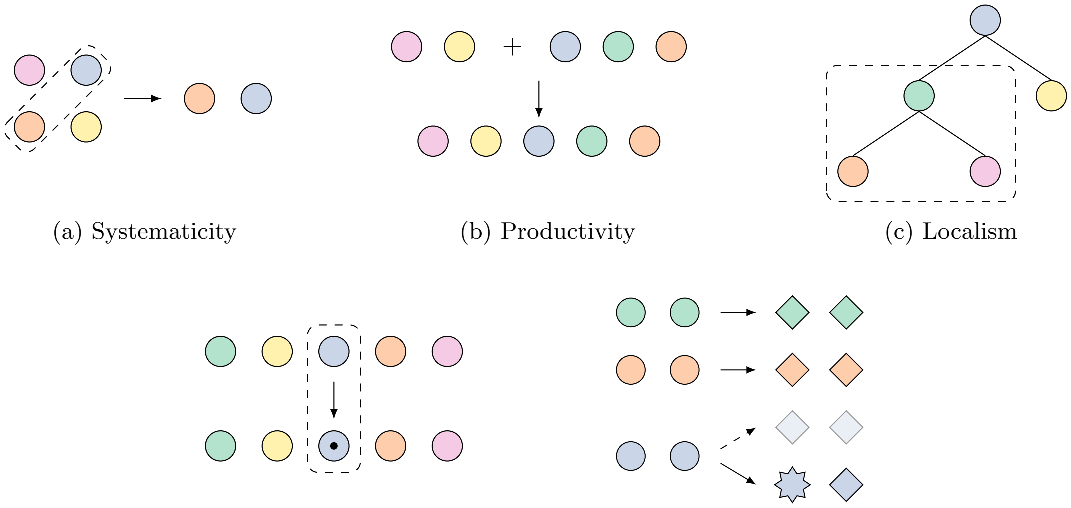
> [t]he ability to produce/understand some sentences is intrinsically connected to the ability to produce/understand certain others" (Fodor and Pylyshyn, 1988, p. 25)

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, in a dictionary-like mapping from sentences to meanings. Someone who entertains such a dictionary would not be able to understand new sentences, even if they were similar to the ones occurring in their dictionary. Since humans are evidently 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. 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.



(a) Systematicity (b) Productivity (c) Localism

(d) Substitutivity (e) Overgeneralisation

Figure 1: 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) The localism test targets how local the composition operations of models are: Are smaller constituents evaluated before larger constituents? (d) In the substitutivity test we evaluate how robust models are towards the introduction of synonyms, and, more specifically, in which cases words are considered synonymous by different models. (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?

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 (i) the model has only been familiarised with a in contexts excluding b and vice versa but (ii) 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'.

Both systematicity and productivity rely on the recombination of known constituents into larger compounds. However, whereas systematicity can be – to some extent – empircially 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 constituents and how they are combined. It does not impose any restrictions on what counts as an admissible way of combining different elements, which 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 operations are very local (a case also referred to as *strong* or *first-level* compositionality), the meaning of a complex expression depends only on its local structure and the meanings of its immediate parts (Pagin and Westerståhl, 2010; Jacobson, 2002). In other words, the meaning of a compound is only dependent on the *meaning* of its immediate 'children', regardless of way that their meaning was built up. In a *global* or *weak* compositionality, the meaning of an expression follows from its total (global) structure and the meanings of its atomic parts. In this interpretation, compounds can have different meanings, depending on the larger expression that they are a part of.

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 to 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 `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. A difference between these two outputs, instead, indicates that the model does not compute meanings by first computing the meanings of all subparts, but pursues a different, more global, strategy.

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.

3.4 Substitutivity

A principle closely related to the principle of 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 concerns 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 cases, but instead focus on the more general question: is substitutivity a salient notion for neural networks and under what conditions can and do they infer synonymity?

3.4.1 Testing substitutivity

We test substitutivity by probing under which conditions a model considers two atomic units are synonymous. To this end, we artificially introduce synonyms and consider how the prediction of a model changes when an atomic unit in an expression is replaced by its synonym. 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 on the output side, but is aided by distributional similarities on the input side. 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 a model: we consider if models exhibit *overgeneralisation* when faced with *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 a model overgeneralises by applying the rules also to such cases, we hypothesise 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

As observed by many others before us, 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, 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 an artificial translation task, in which sequences that are generated by a probabilistic context free grammar (PCFG) should be translated into output sequences that represent their meanings. These output sequences are constructed by recursively applying the *string edit* operations that are specified in the input sequence. The task, which we dub PCFG SET, does not contain any non-compositional phenomenon and we can thus be certain that compositionality is in fact a salient feature. At the same time, we construct the input data such that in other dimensions – such as the lengths of the sentences and depths of the parse trees – it matches the statistical properties of a corpus with sentences from natural language (English).

4.1 Input sequences: syntax

The input alphabet of PCFG SET contains three types of words: words for unary and binary functions that represent *string edit operations* (e.g. `append`, `copy`, `reverse`), elements to form the string sequences that these functions can be applied to (e.g. `A`, `B`, `A1`, `B1`), and a separator to separate the arguments of a binary functions (`,`). The input sequences that are formed with these task are sequences describing how a series of such operations are to be applied to a string argument. For instance:

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

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

Figure 2: 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.

We generate input sequences with a PCFG, shown in Figure 2 (for clarity, production probabilities are omitted). As the grammar we use for generation is recursive, we can generate an infinite number of admissible input sequences. Because the operations can be nested, the parse trees of valid sequences can be arbitrarily deep and long. Additionally, 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 use this to create a data set whose distribution of lengths and depths matches that of a data set containing English sentences.

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

4.2 Output sequences: semantics

The meaning of a PCFG SET input sequence is constructed by recursively applying the string edit operations specified in the sequence. This mapping is governed by the interpretation functions listed in Figure 3. Following these interpretation functions, 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
```

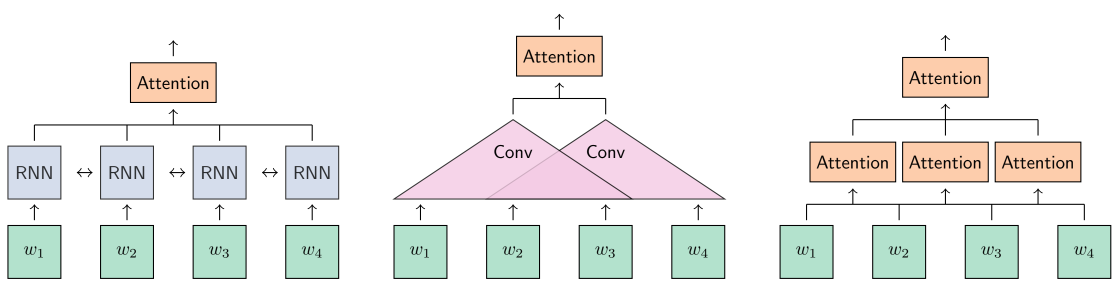
The definitions of the interpretation functions specify the systematic procedure by which an output sequence should be formed from an input sequence, without having to enumerate particular input-output pairs. In this sense, PCFG SET differs from a task such as the lookup table task introduced by Liška et al. (2018), where functions must be exhaustively defined because there is no systematic connection between arguments and the values to which functions map them.

4.3 Data construction

As argued earlier in this paper, the fact that a dataset is generated by a compositional system does not necessarily imply that successfully 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, models should not profit from memorising specific input-output pairs or be able to apply mix-and-match strategies. 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

As a use-case for our compositionality test-suite, we compare three currently popular neural architectures for sequence-to-sequence language processing tasks such as machine translation, speech processing and language understanding: recurrent neural networks (Sutskever et al., 2014), convolutional neural networks (Gehring et al., 2017b) and transformer neural networks (Vaswani et al., 2017). In this section we explain their most important features and include a brief overview of their potential strengths and weaknesses in relation to compositionality.



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

Figure 4: High-level graphical depictions of the most important features of the encoding mechanisms in LSTMS2S, ConvS2S and Transformer models. (a) 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) ConvS2S divides the input sequence into local fragments of consecutive items that are processed in the same convolutions, before applying attention. (c) Transformer immediately applies several global attention layers to the input, without incrementally constructing a preliminary representation.

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. He and Golub, 2016).

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*.

LSTMS2S is a fully recurrent, bidirectional model. The encoder processes an 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, the decoder performs a similar, sequential computation to unroll the predicted sequence. Here, 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 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.

5.2 ConvS2S

The second architecture we consider is a convolutional sequence-to-sequence model. Convolutional sequence-to-sequence models have obtained competitive results in the fields of machine translation (Gehring et al., 2017a) and abstractive summarisation (Denil et al., 2014). In this paper, we follow the setup described by Gehring et al. (2017b) and use their nomenclature: we refer to this model with the abbreviation *ConvS2S*.

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. Although the convolutions already contextualise information in a sequential order, the source and target embeddings are also combined with position embeddings that explicitly encode order. 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. This topology is also biased towards the integration of local information and thus may hinder modelling long-distance relations. However, convolutional networks have found to maintain a much longer effective history than their recurrent counterparts (Bai et al., 2018). Within ConvS2S, such distance portions in the input sequence may be combined primarily through the multi-step attention, which has been shown to improve 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 machine translation and becomes increasingly popular also in other domains, such as language modelling (e.g. Radford et al., 2019).

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 of 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 of 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.

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. We now describe in detail our experimental setup. First we explain how we sample sentences from all potential expressions in PCFG SET (Section 6.1). 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). We have made the data, trained models and code to replicate our results available online.[5]

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 these statistics, we use the English side of a large machine translation corpus: WMT 2017 (Bojar et al., 2017). We parse this corpus with a statistical 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 detailed description of the naturalisation procedure we refer to Appendix A.

In Figure 5a and Figure 5b, we plot the distributions of the WMT data and a sample of around ten thousand sentences of the resulting PCFG SET data.

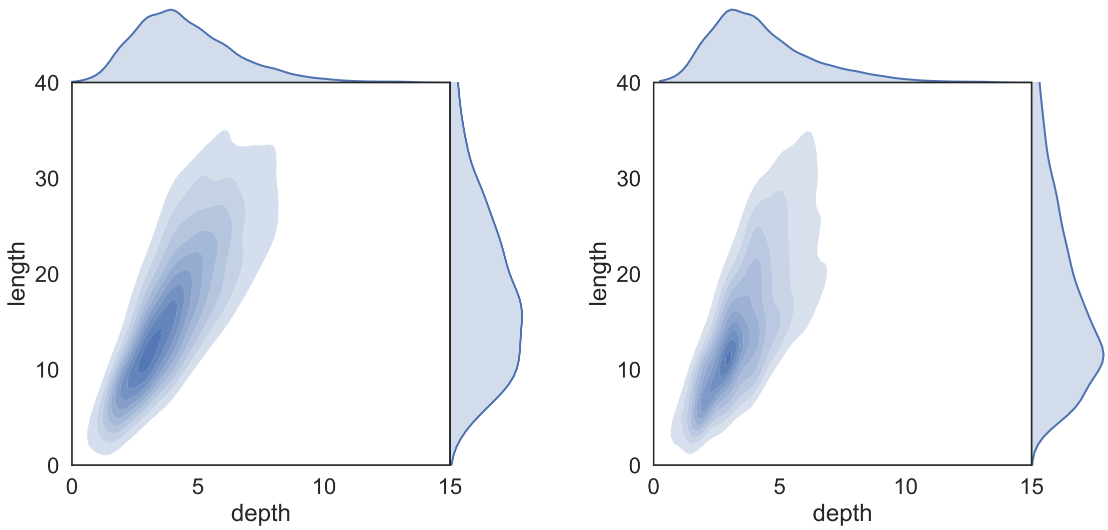
6.1.2 Sentence selection

We set the size of the string alphabet to 520 and create a base corpus of 100 thousand 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-occurrence of specific subsequence in general, the relatively large string alphabet makes it improbable that particular sub-sequences occur often enough to make memorisation a profitable learning strategy.

6.2 Actualisations of compositionality tests

In the following paragraphs, we detail how we concretise the five tests proposed in Section 3 for PCFG SET.

5. `https://github.com/i-machine-think/am-i-compositional`



(a) WMT17 (b) PCFG SET data

Figure 5: Distribution of lengths and depths in the PCFG SET (left) and English WMT 2017 test data (right).

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. In particular, we evaluate four pairs of functions: `swap repeat`, `append remove_second`, `repeat remove_second` and `append swap`.[6] 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 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.

6. To decrease the number of dimensions of variation, we keep the specific pairs of functions fixed during evaluation: 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`. Like 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 occurrences of the chosen functions F with F_{syn}, keeping the target constant. Originally, the individual functions appeared in 39% of the training samples. 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} only 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 sub-sequence in an input sequence is replaced with its meaning.[7] If a model uses local composition operations to build up the meanings of input sequences, 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 We compare the output sequence that is generated by a model for a particular input sequence with the output sequence that the same 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 constituents before

7. Thanks to the recursive nature of the PCFG SET expressions and interpretation functions, 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.

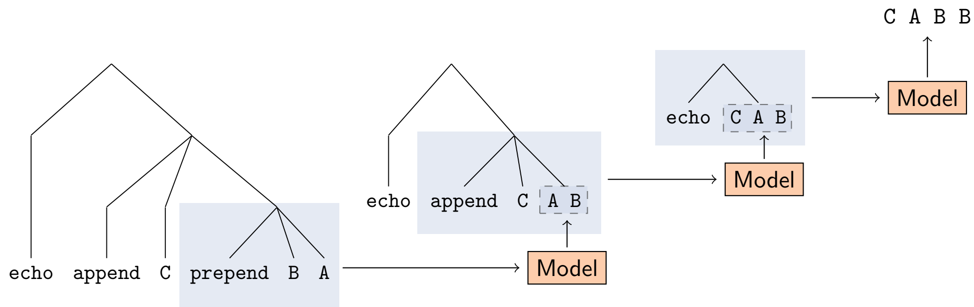


Figure 6: An example of the unrolled computation of the meaning of the sentence `echo append C , prepend B , A` for the localism test. We unroll the computation of the meaning of the sequence by first asking the model to compute the meaning o_1 of the smallest constituent `prepend B , A` and then replace the constituent by this predicted meaning o_1. In the next step, we use the model to compute the meaning of the then smallest constituent `echo` o_1, and replace the constituent in the sequence with the model's prediction for this constituent. This process is repeated until the meaning of the entire sequence is computed, in steps, by the model. This final prediction (`C A B B` in the picture) is then compared with the model's prediction on the entire sequence (not shown in the picture). If a model follows a local compositional protocol to predict the meaning of an output sequence, these two outputs should be the same.

computing the outcome of bigger ones, in the following way: we iterate through the syntactic tree of the input sequence and use the model to compute the meanings of the smallest constituents. We then replace these constituents by the model's output and use the model to again compute the meanings of the smallest constituents in this new tree. This process is continued until the meaning for the entire sequence is found. A concrete example is visualised 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 regular input sequence yields the same prediction as its hierarchical variant. This method of evaluation 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 occurrence 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`. 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. 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. For example, the meaning of `echo remove_first A , B C` would normally be `B C C`, but has now become `A B C`. The remapped definitions, which we call *exceptions*, can be found in Table 2.

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 (first column, *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 $F_1 F_2$. 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 a model correctly memorises the exception target and how often it overgeneralises to the compositional meaning, despite the evidence in the data. To summarise a 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.[8] 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 successful in the past. The details can be found below.

6.3.1 LSTMS2S

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 epochs, or until convergence, 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.

8. 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.

6.3.2 ConvS2S

We use the ConvS2S setup that was presented by Gehring et al. (2017b). Word vectors are 512-dimensional. Both the encoder and decoder have 15 layers, with 512 hidden units in the first 10 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[9]. Unless mentioned otherwise, we use the default hyperparameters of this library. We replicate the training procedure of Gehring et al. (2017b), 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[10] (Klein et al., 2017) to train the model according to the guidelines provided by the framework[11]: with the Adam optimiser ($\beta_1 = 0.9$ and $\beta_2 = 0.98$) and a learning rate increasing for the first 8000 'warmup steps' and decreasing afterwards. We train all models for 25 epochs, or until convergence, 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. We measure this using a *consistency score*, which expresses a pairwise equality, where a 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 †.

9. Fairseq toolkit: `https://github.com/pytorch/fairseq`
10. Pytorch port of OpenNMT: `https://github.com/OpenNMT/OpenNMT-py`.
11. 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 PCFG SET. The results are averaged over three runs and the standard deviation is indicated. Two performance measures are used: *sequence accuracy*, indicated by *, and *consistency score*, indicated 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 LSTMS2S and ConvS2S ($p \approx 10^{-6}$ 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 LSTMS2S ($p \approx 10^{-3}$), 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.

7.1.1 Correlation with length and depth

We explore how the accuracy of the three different architectures develops with increase difficulty of the input sequences, as measured in the input sequence's depth (the maximum level of nestedness observed in a sequence), the input sequence's length (number of tokens) and the number of functions in the input 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 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 one function, all models achieve a maximum accuracy. To compare the difficulty of the functions, we create one corpus with composed input sequences and derive for each function a separate corpus in which this function is applied to those composed input sequences. We then express the comparative difficulty of a function for a model as this model's accuracy on the corpus corresponding

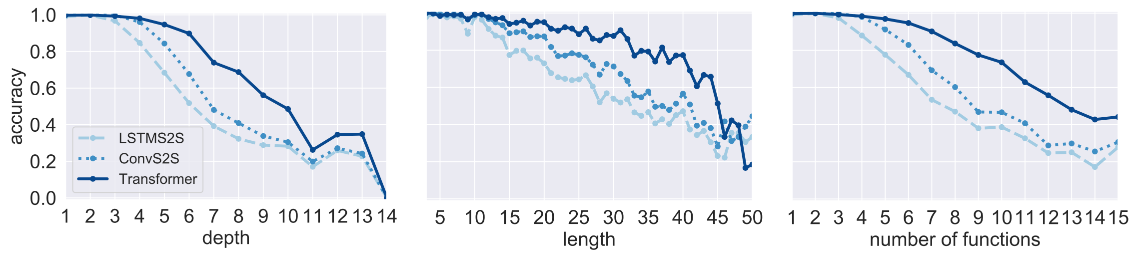
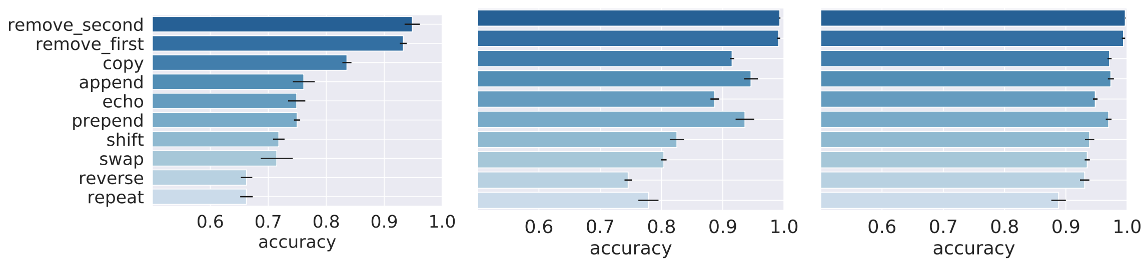


Figure 7: Sequence accuracy of the three models as a function of several properties of the input sequences for the general PCFG SET test set: *depth* of input's parse tree, the input sequence's *length* and the *number of functions* input sequence. The results are averaged over three model runs and computed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



(a) LSTMS2S

(b) ConvS2S

(c) Transformer

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

to this function. For example, to compare the functions `echo` and `reverse`, we create two minimally different corpora that only differ with respect to the first input function in the sequence (e.g. `echo append swap F G H , repeat I J` and `reverse append swap F G H , repeat I J`), and compute the model's accuracy on both corpora.[12] 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 between architectures 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: while LSTMS2S and ConvS2S

12. Note that the since inputs to unary and binary functions are different, we have to use two different corpora to compare binary and unary function difficulty. The unary and binary function scores in Figure 8 are thus not directly comparable.

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

In the systematicity test, we focus on models' ability to systematically generalise, by testing their ability to interpret pairs of functions that were not seen together during training. Following the task accuracy, also for the systematicity test the Transformer model obtains higher scores than both LSTMS2S and ConvS2S ($p \approx 10^{-3}$ 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 they 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 LSTMS2S is its 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 to some extent surprising, since PCFG SET is constructed such that a high task accuracy requires systematic recombination. As such, these results serve as a reminder that models may find unexpected solutions, even for very carefully constructed data sets. A potential explanation for this particular 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 set. 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 separate representations for the two functions. Thus, to compute the output of a sequence like `repeat copy swap echo X`, the model may 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. While larger chunking to ease processing is not necessarily a bad strategy, we argue that it is desirable if models can also maintain a separate representation of the units that make up such chunks, that may be needed in other contexts.

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

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.

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). Lastly, due to the naturalisation of the distribution of lengths, longer sequences are underrepresented in the training data. There is thus fewer evidence for such sequences than there is for shorter ones. As such, models may have to perform a different kind of generalisation to infer the meaning of longer sequences than they do for shorter ones. Their decrease in performance when sequences grow longer could thus also have been explained by a general poor ability to generalise to lengths outside their training space, a type of generalisation sometimes referred to with the term *extrapolation*.

With our productivity test, we focus purely on this extrapolation aspect, by studying models' ability to successfully generalise to longer sequences, an ability which we will call the model's *productive power*. To do so, we redistribute the training and testing data so that there is no evidence at all for longer sequences in the training set.[13] The overall 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 speaking, 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.

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. This decrease in performance is solely caused by the decrease in evidence for such sequences: The total number of examples that models were trained on is the same across the two conditions, and the absolute difficulty of the longer sequences is as well. With these two components factored out, we conclude that models in fact struggle to productively generalise to longer sequences. [14]

13. For the details concerning the statistics of the adapted data, we refer back to Table 1.
14. 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 models' inability to postpone the emission of this `<eos>` symbol. We dub this problem the `<eos>-problem`. To test whether the low scores are due to early `<eos>` emissions, we compute how many of the wrongly emitted answers were contained in the right answer. For LSTMS2S, ConvS2S and Transformer this was the case in 20%, 6% and 11% of the wrong predictions. These numbers illustrate that the `<eos>`-problem indeed exists, but is not the main source of the poor productive capacity of the different models.

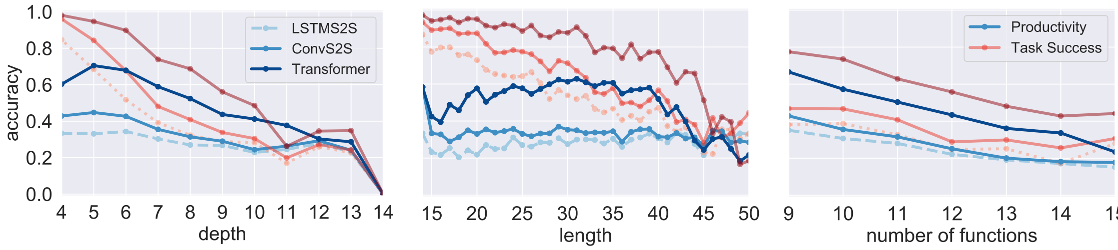


Figure 9: Accuracy of the three models on the productivity test set as a function of several properties of the input sequences: *depth* of the input's parse tree, the input sequence's *length* and the *number of functions* present. The results are averaged over three model runs and computed over ten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

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 treat words as *synonyms*. We consider what happens when synonyms are *equally distributed* in the input sequences and the case in which one of the synonyms only occurs in *primitive contexts*.

7.4.1 Equally distributed substitutions

For the substitutivity experiment where words and synonyms are equally distributed, Transformer and ConvS2S 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.

Token	LSTMS2S			ConvS2S			Transformer		
	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.

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.

While the consistency scores for all models decrease substantially compared to the equally distributed setup, all models pick up that there is a 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

	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.

the consistency score for model outputs that do not match the target output (incorrect outputs). When a model makes identical but incorrect predictions 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 3 show that the larger part of the consistency scores for all models is due to correct outputs. In row 4, we see that models are seldom consistent on *incorrect* 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 compute the meanings of input sequences using local composition operations, 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.

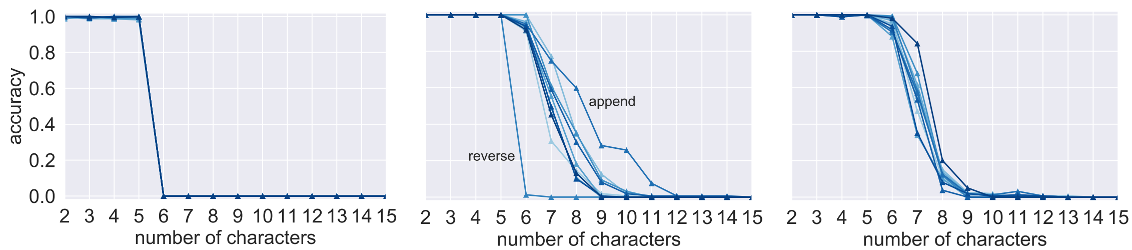
If a model truly understands the meaning of a particular function in PCFG SET, it should in principle be able to apply this function to an input string of arbitrary length. Note that, in our case, this ability does not require productivity in generating output strings, since the correct output sequences 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 suggests that, while models may have learned to adequately copy strings of length three to five, they do not necessarily consider those operations the same.

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.[15] 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.

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

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`,

15. For binary functions, only one of the two string arguments exceeded the regular argument lengths.



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

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 Transformer and ConvS2S can, for most functions, generalise a little beyond this string length, the LSTM-based models cannot.

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 a model on the adapted outputs, that can only be learned by memorisation – is indicated in the

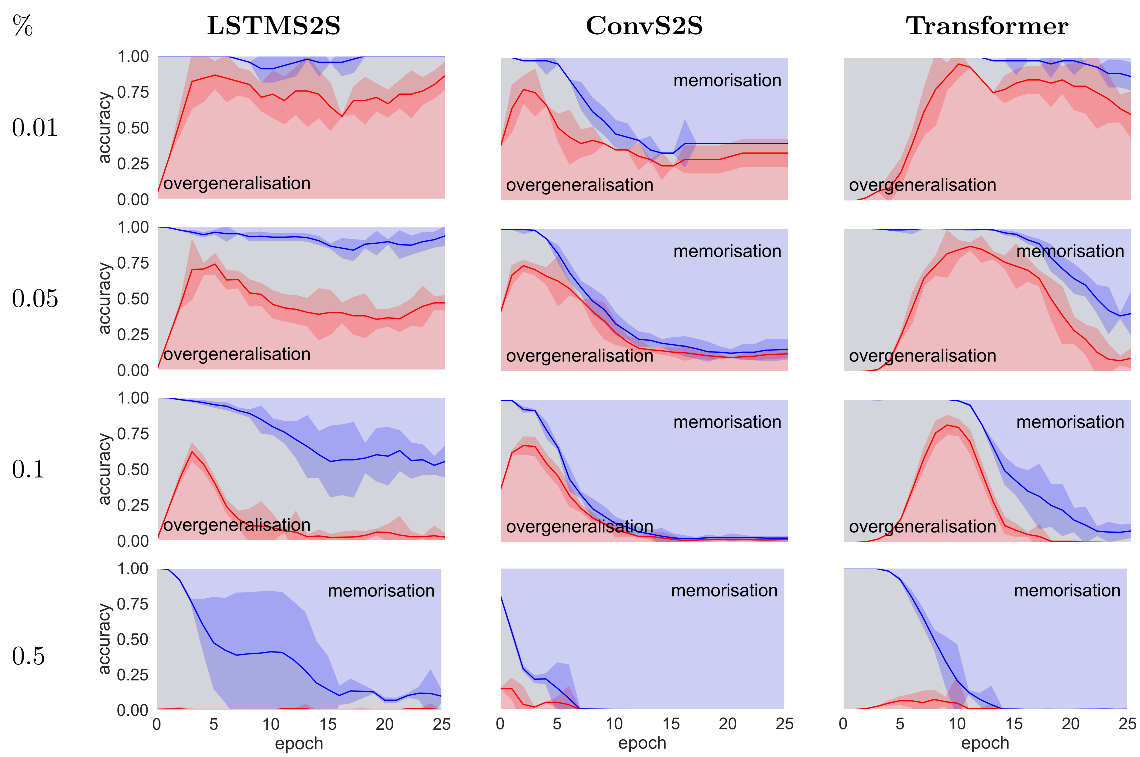


Table 7: Overgeneralisation profiles over time for LSTMS2S, ConvS2S and Transformer for exception percentages of 0.01%, 0.05%, 0.1% and 0.5% (in increasing order, from top to bottom). The lower area of the plots, in red, indicates the mean fraction of exceptions (with standard deviation) for which an 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 areas, in blue, indicate 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.

upper part of the plots, in blue. The grey area in between indicates the percentage of exception examples for which a 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.

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 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 are explicitly motivated by theoretical literature about compositionality. Our evaluation framework contains five independent tests, that consider complementary aspects of compositionality that are frequently mentioned in the literature about compositionality. These five tests allow to investigate (i) if models systematically recombine known parts and rules (*systematicity*) (ii) if models can extend their predictions beyond the length they have seen in the training data (*productivity*) (iii) if models' composition operations are local or global (*localism*) (iv) if models' predictions are robust to synonym substitutions (*substitutivity*) and (v) if 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 our evaluation paradigm, we instantiate the five tests on a highly compositional artificial dataset we dub PCFG SET: a sequence-to-sequence translation problem which requires to compute meanings of sequences that are generated by a probabilistic context free grammar by recursively applying string edit operations. This dataset is designed such that modelling it adequately requires a fully compositional solution, and it is generated such that its length and depth distributions match those of a natural corpus of English. We use our instantiated tests to evaluate three popular sequence-to-sequence architectures: an LSTM-based (*LSTMS2S*), a convolution-based (*ConvS2S*) and an all-attention model (*Transformer*). For each test, we further conduct a number of auxiliary tests that can be used to further increase the understanding of how this aspect is treated by a particular architecture. We will make all data sets and evaluation scripts to conduct these evaluations publicly available online upon publication.

The overall accuracy on PCFG SET is relatively high for all models, with the Transformer model coming out on top with an accuracy of over 90%. A more detailed picture is given by the five compositionality tests, that indicate that – despite our careful data design, high scores do still not necessarily imply that the trained models follow the intended compositional solution and that illustrate how they handle different aspects that could be considered important for compositionality.

Firstly, our **systematicity** test shows that none of the architectures successfully generalises to pairs of words that were not observed together during training, a result that confirms earlier studies such as the ones from Loula et al. (2018) and Lake and Baroni (2018). The difference of the systematicity scores with the overall task accuracy is quite stark for all models: a drop of 34%, 38% and 27% for LSTMS2S, ConvS2S and Transformer, respectively. We hypothesise that this result suggests that the low accuracy on the systematicity test does not stem from poor systematic capacity in general, but that the models instead use different segmentations of the input, applying – for instance – multiple functions at ones, instead of all of the functions in a sequential manner. We reason that while larger chunking to ease processing is not necessarily a bad strategy, it is desirable if models can also maintain a separate representation of the units that make up such chunks, as these units could be useful or needed in other sentences.

With our **productivity** test, we assess if models can productively generalise to sequences that are longer than the ones they observed in training. To evaluate this, we redistribute the training examples such that there is a strict separation of the input sequence lengths in the train and test data. By comparing the results with the accuracies of models that are trained on data sets that contain at least some evidence for longer sequences, we tease apart the overall difficulty of modelling longer sequences from the ability to generalise to unseen lengths. Also in this test, the Transformer model outperforms the other two architectures, but none of the architectures exhibits strong productive power to sequences of unseen lengths: The Transformer accuracy on the productivity test set is only 0.56. By computing how often models' predictions were strictly contained within the true output sequence, we assess if the poor productive power of all models is caused by early emission of the end of sequence symbol. We find that such cases indeed exist (20%, 6% and 11% for LSTMS2S, ConvS2S an Transformer, respectively), but early stopping of the generation is not the main cause of the low productivity scores.

In our **substitutivity** test, we compare how models react to artificially introduced synonyms occurring in different types of scenarios. Rather than considering their behaviour in terms of sequence accuracy, we compute how *consistent* models predictions – correct or incorrect – between sentences with synonym substitutions. When synonyms are equally distributed in the input data, both Transformer and ConvS2S obtain high consistency scores (0.98 and 0.96, respectively), while LSTMS2S is substantially less consistent (0.76). This difference is also reflected in the distance between the embeddings of words and synonyms, which is much lower for Transformer and ConvS2S. When one of the synonyms is only presented in a few very short sequences, the consistency score of ConvS2S drops to the same level as the consistency of LSTMS2S (0.61), while the Transformer still maintains a relatively high synonym consistency of 0.88. Also the embeddings of synonyms remain relatively close in the Transformer models' embedding space, despite the fact that they are distributionally dissimilar. To take apart the ability to learn from very few examples and to infer synonymity, we also consider how consistent models are on *incorrect* outputs. Here, we observe that none of the models can be said to truly treat words and their counterparts as synonyms. The Transformer model is, again, the most consistent, but with a score of only 0.24. This test shows an interesting difference between LSTMS2S and ConvS2S: where the former appears to be better at inferring that words are synonyms, the latter is better at few-shot learning a words meaning from very few examples.

With our **localism** test, we consider if models apply local composition operations that are true to the syntactic tree of the input sequences, or rather compute the meaning of sequence in a more global fashion. In line with the results of the systematicity test, models do not appear to truly follow the syntactic tree of the input to compute its meaning. In 45%, 57% and 56% of the test samples for LSTMS2S, ConvS2S and Transformer, respectively, enforcing a local computation results in a different answer than the original answer provided by the model. An error analysis suggests that these results are largely due to function applications to longer string sequences. With an additional test in which we monitor the accuracy of models functions applied to increasingly long string inputs, we find evidence that models may not learn general-purpose representations of functions, but instead use different protocols for *copy once* or *copy twice*. We see that the accuracy of LSTMS2S immediately drops to 0 when string inputs are longer than the ones observed in training; The performance of ConvS2S and Transformer, instead, drops rapidly, but remains above 0 for slightly longer string inputs. These results indicate that LSTM2S may indeed not have learned a general-purpose representation for functions, while the decreasing accuracy of ConvS2S and Tranformer could be related more to performance rather than competence issues.

In our last test, we study **overgeneralisation** during training, by monitoring the behaviour of models on artificially introduced *exceptions* to rules. We find that for small amount of exceptions (up to 0.1% of the overall occurrence of the rule in the data) all architectures overgeneralise in the beginning of their training. As overgeneralisation implies that models overextend rules in cases where this is explicitly contradicted by the data, we take this as a clear indication that models in fact capture the underlying rule at that point. For very small amounts of exceptions (0.01% of the overall rule occurrence), both Transformer and LSTMS2S fail to learn the exception at all: even after their training has converged they overgeneralise on the sequences containing exceptions. To a lesser extent, also ConvS2S struggles with capturing low frequent exceptions. LSTMS2S generally appears to have difficulty with accommodating both rules and exceptions. Often, after learning that a certain rule should not be applied, LSTMS2S models do not memorise the true target, but proceed to predict something which matches nor this target nor the general rule. ConvS2S and Transformer do not show such patterns: when their *overgeneralisation* goes down, their *memorisation* score goes up. Aside from in the beginning of their training, they rarely predict something outside of these options. For larger percentages of exceptions (from 0.5% of the overall rule occurrence) none of the architectures really exhibits overgeneralisation behaviour.

In all our tests, we used an artificial data set that is entirely explainable in terms of compositional phenomena. This permitted us to focus on the compositional capabilities of different models in the face of compositional data and allowed us to isolate compositional processing from other signals that are found in more realistic datasets. However, it leaves open the question of how much the compositional trains we identified are expressed and can be exploited by networks when facing natural data. As future work, we plan to instantiate our tests in natural language domains such as translation and summarisation. The results of such a study would provide valuable information about how well models pick up compositional patterns in more noisy environments, but might also provide insight about the importance of these different aspects of compositionality to model natural data.

In summary, we provided an evaluation paradigm that allows to test the extent to which five distinct, theoretically motivated aspect of compositionality are represented by artificial neural networks. By instantiating these tests for an artificial data set and applying the resulting tests on three different successful sequence-to-sequence architectures, we shed some light on which aspects of compositionality may provide problematic for different architectures. These results illustrate well that to test for compositionality in neural networks it does not suffice to consider an accuracy score on a single downstream task, even if this task is designed to be highly compositional. Models may capture some compositional aspects of this dataset very well, but fail to model other aspects that could be considered part of a compositional behaviour. As such, our the results themselves demonstrate the need for the more extensive set of evaluation criteria that we aim to provide with this work. We hope that future researchers will use our collection of tests to evaluate new models, to investigate the impact of hyper parameters or to study how compositional behaviour is acquired during training. To facilitate the usage of our test suite we have made the PCFG SET data generator, all test sets and the models trained by us available online.[16] We further hope that our theoretical motivation, the test suite itself and the analysis that we presented of its application on three different sequence-to-sequence architectures will mark a step forward in the having a clear discussion about compositionality and deep learning, both from a practical and a theoretical perspective.

References

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*.

Bai, S., Kolter, J. Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *CoRR*, abs/1803.0127.

Baroni, M. and Zamparelli, R. (2010). 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.

Belinkov, Y., Màrquez, L., Sajjad, H., Durrani, N., Dalvi, F., and Glass, J. (2017). Evaluating layers of representation in neural machine translation on part-of-speech and semantic tagging tasks. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1–10.

Blevins, T., Levy, O., and Zettlemoyer, L. (2018). Deep RNNs encode soft hierarchical syntax. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, volume 2, pages 14–19.

Bojar, O., Buck, C., Chatterjee, R., Federmann, C., Graham, Y., Haddow, B., Huck, M., Jimeno-Yepes, A., Koehn, P., and Kreutzer, J., editors (2017). *Proceedings of the Second Conference on Machine Translation, WMT 2017*. Association for Computational Linguistics.

Bowman, S. R., Manning, C. D., and Potts, C. (2015). 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.

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

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

Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-based models for speech recognition. In *Advances in neural information processing systems*, pages 577–585.

16. https://github.com/i-machine-think/am-i-compositional

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.

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

Denil, M., Demiraj, A., Kalchbrenner, N., Blunsom, P., and de Freitas, N. (2014). Modelling, visualising and summarising documents with a single convolutional neural network. *CoRR*, abs/1406.3830.

Dessì, R. and Baroni, M. (2019). CNNs found to jump around more skillfully than rnns: Compositional generalization in seq2seq convolutional networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL), Short Papers*, pages 3919–3923.

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

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

Gehring, J., Auli, M., Grangier, D., and Dauphin, Y. N. (2017a). A convolutional encoder model for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL), Long Papers*, volume 1, pages 123–135.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017b). Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning, (ICML)*, pages 1243–1252.

Goldberg, Y. (2019). Assessing BERT's syntactic abilities. *CoRR*, abs/1901.05287.

Goller, C. and Kuchler, A. (1996). 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.

Gulordava, K., Bojanowski, P., Grave, E., Linzen, T., and Baroni, M. (2018). Colorless green recurrent networks dream hierarchically. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, volume 1, pages 1195–1205, New Orleans, LA.

He, X. and Golub, D. (2016). Character-level question answering with attention. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1598–1607.

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

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

Hupkes, D., Singh, A., Korrel, K., Kruszewski, G., and Bruni, E. (2019). Learning compositionally through attentive guidance. In *International Conference on Computational Linguistics and Intelligent Text Processing (CICLing)*.

Hupkes, D., Veldhoen, S., and Zuidema, W. (2018). Visualisation and 'diagnostic classifiers' reveal how recurrent and recursive neural networks process hierarchical structure. *Journal of Artificial Intelligence Research*, 61:907–926.

Husserl, E. (1913). *Logische Untersuchungen*. Max Niemeyer.

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

Janssen, T. (1983). *Foundations and applications of Montague grammar*. Mathematisch Centrum.

Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., and Girshick, R. (2017). CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1988–1997.

Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. (2016). Exploring the limits of language modeling. *CoRR*, abs/1602.02410.

Kim, Y., Rush, A. M., Yu, L., Kuncoro, A., Dyer, C., and Melis, G. (2019). 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.

Klein, G., Kim, Y., Deng, Y., Senellart, J., and Rush, A. M. (2017). Opennmt: Open-source toolkit for neural machine translation. In Bansal, M. and Ji, H., editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL), System Demonstrations*, pages 67–72. Association for Computational Linguistics.

Korrel, K., Hupkes, D., Dankers, V., and Bruni, E. (2019). Transcoding compositionally: using attention to find more generalizable solutions. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, page 111.

Lake, B. and Baroni, M. (2018). 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).

Le, P. and Zuidema, W. (2015). 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.

Lin, Y., Tan, Y. C., and Frank, R. (2019). Open sesame: Getting inside BERT's linguistic knowledge. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 241–253.

Linzen, T., Dupoux, E., and Goldberg, Y. (2016). Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535.

Liška, A., Kruszewski, G., and Baroni, M. (2018). Memorize or generalize? searching for a compositional RNN in a haystack. *CoRR*, abs/1802.06467.

Loula, J., Baroni, M., and Lake, B. M. (2018). Rearranging the familiar: Testing compositional generalization in recurrent networks. In *Proceedings of the EMNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 108–114.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

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

Marcus, G. F., Pinker, S., Ullman, M., Hollander, M., Rosen, T. J., Xu, F., and Clahsen, H. (1992). Overregularization in language acquisition. *Monographs of the society for research in child development*, pages i–178.

Mareček, D. and Rosa, R. (2018). 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.

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

Mitchell, J. and Lapata, M. (2008). Vector-based models of semantic composition. *Proceedings of ACL-08: HLT*, pages 236–244.

Mul, M. and Zuidema, W. (2019). Siamese recurrent networks learn first-order logic reasoning and exhibit zero-shot compositional generalization. *CoRR*, abs/1906.00180.

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

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

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

Penke, M. (2012). The dual-mechanism debate. In *The Oxford handbook of compositionality*. Oxford University Press.

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

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

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8).

Raganato, A. and Tiedemann, J. (2018). An analysis of encoder representations in transformer-based machine translation. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 287–297.

Rumelhart, D. E. and McClelland, J. L. (1986). *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.

Saxton, D., Grefenstette, E., Hill, F., and Kohli, P. (2019). Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations (ICLR)*.

Shi, X., Padhi, I., and Knight, K. (2016). Does string-based neural MT learn source syntax? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1526–1534.

Socher, R., Manning, C. D., and Ng, A. Y. (2010). 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.

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

Szabó, Z. (2012). The case for compositionality. *The Oxford handbook of compositionality*, 64:80.

Tenney, I., Das, D., and Pavlick, E. (2019a). BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, page 45934601.

Tenney, I., Xia, P., Chen, B., Wang, A., Poliak, A., McCoy, R. T., Kim, N., Van Durme, B., Bowman, S. R., Das, D., et al. (2019b). What do you learn from context? Probing for sentence structure in contextualized word representations. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*.

Tran, K., Bisazza, A., and Monz, C. (2018). 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.

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

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Veldhoen, S., Hupkes, D., and Zuidema, W. (2016). Diagnostic classifiers: Revealing how neural networks process hierarchical structure. In *Proceedings of the NIPS2016 Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches*.

Vig, J. and Belinkov, Y. (2019). Analyzing the structure of attention in a transformer language model. *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 63–76.

Wolf, T. (2019). Some additional experiments extending the tech report assessing BERTs syntactic abilities by yoav goldberg. Technical report, Technical report.

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

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.