Harnessing the Symmetry of Convolutions for Systematic Generalisation

Jeff Mitchell
Department of Psychology
University of Bristol
Bristol, UK
jeff.mitchell@bristol.ac.uk

Jeffrey S. Bowers
Department of Psychology
University of Bristol
Bristol, UK
j.bowers@bristol.ac.uk

***Abstract*—We argue that symmetry is an important consideration in addressing the problem of systematic generalisation and investigate two forms of symmetry relevant to symbolic processes. We implement this approach in terms of convolution and show that it can be used to achieve effective generalisation in a rule learning and a context free language task.**

In the rule learning task, we find that symmetry allows us to learn rules that abstract away from the particular symbols that instantiate them, enabling generalisation from seen to unseen symbols. In the language task, symmetry allows us to impose a stack like architecture on the memory cells of a recurrent net, which permits generalisation from simple to more complex structures.

***Index Terms*—Connectionism and neural nets, Symbolic and algebraic manipulation, Convolution**

I. Introduction

Convolution has played a central role in making Deep Neural Networks (DNNs) successful. Applying the same set of filters across all positions in an image captures an important characteristic of the processes that generate the objects depicted in them, namely the translational symmetry of the underlying laws of nature. Given the impact of these architectures, researchers are increasingly interested in finding approaches that can be used to exploit further symmetries [1], [2], such as rotation or reflection. Here, we will investigate symmetries relevant to symbolic processing.

We show that incorporating symmetries derived from symbolic processes into neural architectures allows them to generalise more robustly on tasks that require handling elements and structures that were not seen at training time. Specifically, we construct convolution-based models that outperform standard approaches on the rule learning task of Marcus et al. [3], and a simple context free language learning task.

Symbolic architectures form the main alternative to conventional neural networks as models of intelligent behaviour, and have distinct characteristics and abilities. Specifically, they form representations in terms of structured combinations of atomic symbols. Their power comes not from the atomic symbols themselves, which are essentially arbitrary, but from the ability to construct and transform complex structures. This allows symbolic processing to happen without regard to the meaning of the symbols themselves, expressed in the formalist's motto as *If you take care of the syntax, the semantics will take care of itself* [4]. Thus, we will use *symbol* and *symbolic* to refer to representations that are processed within such a formal scheme.

From this point of view, thought is a form of algebra [5], [6] in which formal rules operate over symbolic expressions, without regard to the values of the variables they contain [7]. As a consequence, those values can be processed systematically across all the contexts they occur in. So, for example, we do not need to know who *Socrates* is or even what *mortal* means in order to draw a valid conclusion from *All men are mortal and Socrates is a man*.

However, connectionist approaches have been criticised as lacking this systematicity. Fodor and Pylyshyn [8] claimed that neural networks lack the inherent ability to model the fact that *cognitive capacities always exhibit certain symmetries, so that the ability to entertain a given thought implies the ability to entertain thoughts with semantically related contents*. Thus, understanding these symmetries and designing neural architectures around them may enable us to build systems that demonstrate this systematicity.

We investigate two kinds of symmetry, relating to substitutions of symbols and to equivalence between memory slots. We use the former to assist generalisation from seen to unseen symbols in the rule learning experiment, while the latter permits generalisation from simpler to more complex grammatical sequences. Implementing these symmetries using convolutional architectures, we find that this allows us to learn rules that abstract away from the particular symbols seen during training and to build stack-like memory structures.

The relation between symbols and their referents is, in principle, arbitrary, and any permutation of this correspondence is therefore a symmetry of the system. More simply, the names we give to things do not matter, and we should be able to get equivalent results whether we call it *rose* or *trandafir*, as long as we do so consistently. This should enable unseen syllables to be processed in the same way as seen syllables, producing effective generalisation.

Following on from that, a given symbol should be treated consistently wherever we find it. For example, the symbol *taught* occurs twice in *Socrates taught Plato, who taught*

This research was supported by the European Research Council Grant Generalization in Mind and Machine, ID number 741134.

Aristotle and we want each instance to be treated as expressing a distinct event, while maintaining its identity as a token of the same type. Here, we will think of this as a form of symmetry over the various slots within the data structures, such as stacks and queues, where symbols can be stored. This will then allow the same rules to be applied to all the instances of a given symbol, no matter how numerous, allowing generalisation to larger and more complex structures.

We explore these questions using two small toy problems and compare the performance of architectures with and without the relevant symmetries. In each case, we use convolution as the means of implementing the symmetry, which, in practical terms, allows us to rely only on standard deep learning components. In addition, this approach opens up novel uses for convolutional architectures, and suggests connections between symbolic processes and spatial representations.

II. Rule Learning

The first symmetry to be considered is the one arising from the fact that the correspondence between referring atomic symbols and their referents is entirely arbitrary, which entails that permutations of this mapping are symmetries of the system. In fact, Tarski [9] uses this permutation invariance to define *logical notions* within mathematics. From this perspective, logical notions are those which are invariant to permutations of the correspondence between entities and their names.

Thus, if the mapping from entities in the world to symbols in our language is arbitrary - i.e. the same entity could be called either *rose* or *trandafir* - then symbolic processing should be indifferent to those choices - i.e. we should get equivalent results whether we use *rose* or *trandafir*. This implies that all symbols of a given type are equivalent, in the sense that any name is as good as any other, and that symbolic processes should not depend on the particular symbol but only on the structural forms they are used in. So, for example, logically valid inferences have particular forms (e.g. modus ponens, modus tollens, etc.) which are independent of the entity and predicate names they contain.

This symmetry, in which any symbol is as good as any other, is anathema to the sort of problem that neural nets are typically applied to, in which the inputs are not arbitrary names but specific measurement values, e.g. images or medical records. Thus, we cannot expect to make an arbitrary remapping of, say, blood pressure values without fundamentally changing the way those values need to be processed. In such a case, effective learning requires discovering correlations or decision boundaries in terms of specific values. Imposing the strong symmetry constraint of treating all values of blood pressure equivalently would inhibit the process of uncovering the relevant discriminations between them.

Nonetheless, the work of Marcus et al. [3] suggests that this sort of symmetry may be relevant to human cognition, even for infants as young as 7 months. In these experiments, the infants were habituated to sequences of syllables which obeyed a simple rule, such as ABB (e.g. *la ti ti*) or ABA (e.g. *la ti la*). Subsequent testing on novel stimuli showed they were able generalise this rule to syllables not present in the training stimuli (e.g. *wo fe fe* vs. *wo fe wo*).

TABLE I: Accuracies in Identifying Sequence Structure.

Recurrent Net	Multi-layer Perceptron	Convolution
50%	50%	100%

In other words the representation of the learned rule allowed it to be abstracted from the particular training stimuli and applied to any syllable. One interpretation is that the infants were treating the stimuli symbolically, in that one syllable was as good as any other, and that as a consequence their behaviour was symmetric under substitution of the syllables.

Marcus et al. [3] were unable to obtain the same behaviour from a recurrent network architecture, because the statistical regularities it learned were linked to the specific syllables seen at training time and so generalisation to unseen syllables was not achieved. Here we show that this problem can be solved by imposing a symmetry on the architecture, that corresponds to weight sharing between syllables.

Practically, this is implemented as a convolution of width one, followed by max-pooling across all syllables, and a softmax to produce output probabilities. The input consists of a 12×3 array of binary values representing the 12 syllables and 3 time steps, with two convolutional filters treating the syllables as positions and the time steps as channels. The output of the convolution has two channels, which after pooling become the logits for the binary outputs.

Fig. 1 shows the input sequence *wo fe wo* being processed by this architecture. The input is first encoded as activation in the first and third channels at the *wo* position, and activation in the second channel at the *fe* position. Convolution reduces these three channels down to two, and pooling projects this down to a pair of logits corresponding to the ABB and ABA categories.

Note that this is the opposite of how convolutions are most frequently used in application to language sequences. In that case, invariance to time translations is achieved by weight sharing across time steps, with the representation of each symbol being encoded in the channels. Here, in contrast, weight sharing happens between symbols and temporal information is encoded in the channels. This means our model is not invariant to translations in time, but is instead invariant to substitutions of symbols, e.g. *la* is replaced with *wo*. In this way, we expect the architecture to generalise from seen to unseen syllables.

The training and test inputs are taken from the Marcus et al. [3] paper, and we train the model to distinguish ABB sequences from ABA sequences. We also train a multi-layer perceptron and a recurrent net on the same data, with the recurrence happening over the time dimension. Both these networks have 24 hidden units, matching the structure of the convolutional net. Each network is trained for 1000 epochs using Adam [10] with a rate of 0.1 and a batch size of 16.

The results on the test set in Table I show that neither the

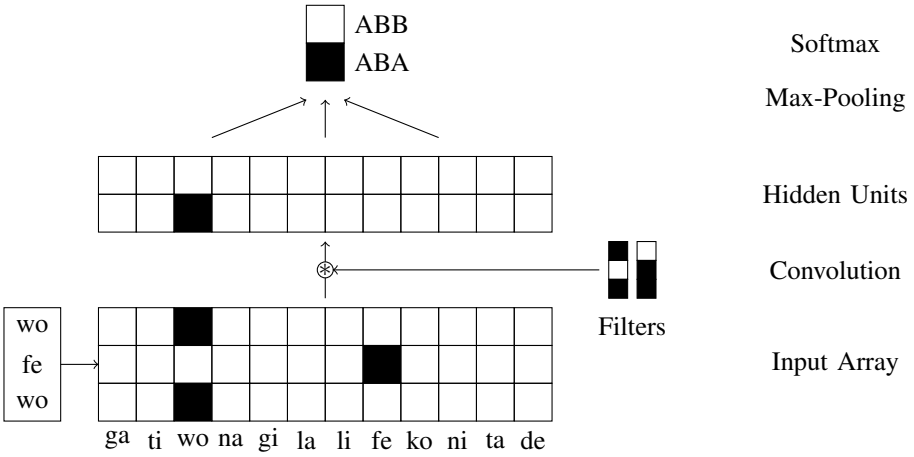

Fig. 1: The architecture applied to the rule learning task of [3], consisting of convolution followed by max-pooling and softmax. In the figure, the 12×3 grid at the bottom represents the 12 syllables within the 3 time-steps, with the first syllable at the top and the last at the bottom. This input is convolved with the two filters, resulting in the 12×2 output above it, and max-pooling reduces this to a single pair of values, which are the logits of the output softmax. In the example, *wo fe wo* is encoded at the input as ones in the first and third channels for the syllable *wo* and in the second channel for the syllable *fe*. The same two filters are applied to the three channels of every syllable, with the 101 filter matching the pattern of activations in the *wo* position, giving a high activation value in the bottom channel at the same position in the hidden units. Max-pooling picks this value out and uses it to predict that this is an ABA sequence.

multi-layer perceptron nor the recurrent network learn a rule that generalises effectively to unseen syllables. This is because the inputs associated with the test syllables are always zero in the training set, and so the weights for these units are never updated in these networks. However, the weight sharing in the convolutional net requires that the same function is applied to each syllable, giving perfect generalisation. The filter, being applied at every position, cannot discriminate between syllables, and instead can only respond to the information about temporal structure in the channels. So, for example, in the case of the sequence *wo fe wo*, the input channels at the *wo* position take the values 101, representing the fact that the same token occurs in the first and third temporal slots.

This is very similar to what Marcus et al. [3] suggest is being learned by the infants: *algebra-like rules that represent relationships between placeholders (variables), such as 'the first item X is the same as the third item Y'*. Thus, by imposing a symmetry on the network, we learn functions that are sensitive to an abstract structure rather than the specific raw syllables in the input.

Although we have imposed the required symmetry on the network in terms of convolution, we do not wish to argue that this should be understood as providing a mechanistic explanation of the behaviour observed in the rule learning experiments. Not only does this architecture lack biological plausibility, but we have also formulated the task as a supervised learning problem when the original experiments might be better understood as unsupervised. Consequently, the network is only interested in distinguishing ABB and ABA inputs, and is effectively blind to all the other features of these sequences that the infants would have attended to.

Instead, our experiments are intended to show that considerations of symmetry are important in obtaining systematic generalisation. In particular, the arbitrariness of symbols implies that symbolic systems are symmetric under substitutions of symbols, e.g. replacing *la* with *wo*. In terms of learning, this supports generalisation from *la* to *wo*, even if *wo* was unseen during training. In this way, convolution provides a straightforward solution to this long-standing problem [11] using the simple expedient of sharing weights between seen and unseen syllables.

III. Context Free Language Learning

The structures considered in the previous section are extremely simple, having only short-range sequential dependencies. In contrast, the real grammars of natural languages produce long-range dependencies within hierarchical structures. Handling such structures, in which multiple dependencies are embedded within each other, will typically require some form of working memory in order to keep track of the unresolved outer dependencies until the inner dependencies are completed. In this section we consider the role that symmetry plays in structuring this memory. We again implement the relevant symmetry using convolution, and this allows us to construct a stack-like memory structure to which items can be pushed and popped. In particular, we consider a reverse recall task that captures a key property of how such a memory has to operate.

For example, in the sentence *The racquet is actually very cheap* the subject noun, *racquet*, and main verb, *is*, display number agreement. In this case, both are singular, but they

$$
\begin{array}{ll}
S \to A & A \to aSa \\
S \to B & B \to bSb \\
S \to C & C \to cSc \\
S \to D & D \to dSd \\
\multicolumn{2}{c}{S \to o}
\end{array}
$$

(a) Production rules for a simple CFG.

$$
\begin{aligned}
S &\to A \\
&\to aSa \\
&\to aDa \\
&\to adSda \\
&\to adoda
\end{aligned}
$$

(b) Example derivation of a palindrome.

Fig. 2: The production rules and an example derivation of a simple CFG, which produces palindromic strings.

could also have been plural, i.e. *racquets* and *are*. For this simple sentence, noun and verb are adjacent so the span of the dependency is minimal. However, we can, in principle, insert as much material as we like, and this syntactic connection persists.

In particular, we can add a relative clause to the noun: *The racquet that the tennis player uses is actually very cheap*. Now another subject noun, *player*, and verb, *uses*, intervenes between the first pair, but the dependency, and in particular the number agreement, between *racquet* and *is* has to be maintained. This remains true even when we insert another relative clause: *The racquet that the tennis player we are all in awe of uses is actually very cheap*.

In this last case, the subject, *we*, and verb, *are*, are both plural, and effective processing of the whole sentence requires that this should not disrupt processing of the outer singular dependencies. Moreover, the dependency between *racquet* and *is* remains the same no matter how long it becomes. In other words, a language user must be able to maintain a trace of multiple open dependencies until the relevant material is encountered. Notably, in the case of a centre embedded construction such as the sentence above, recall has to happen in a last-in-first-out manner as processing descends through the hierarchy and then rises back out again. That is, the subjects in the sentence above - *racquet, player, we* - are matched to verbs in reverse order - *are, uses, is*.

A common model for such structures are Context Free Grammars (CFG), which generate sentences in terms of production rules, such as those in Fig. 2. These rules describe how the start symbol, S, is expanded into sequences of terminals symbols, such as $adoda$ or $ccbabdodbabcc$. Each rule describes a substitution that can be applied to a single non-terminal symbol, i.e. S, A, B, C or D to yield a sequence of symbols. The context free aspect of such a grammar lies in the fact the substitutions are made without regard for the context around the original non-terminal symbol.

In the case of the grammar described in Fig. 2, the substitutions applied to the non-terminals A, B, C and D yield a new string with the same terminal at the beginning and the end, and the final rule inserts an o. As a consequence, the resulting strings are palindromes with a single o at their centre. We can think of this as a simplified form of agreement, where instead of a singular noun agreeing with a singular verb, an a in the first half of the string 'agrees' - i.e. matches - with the corresponding a in the second half. Like the noun-verb dependencies in the example above, these dependencies between the first and second half of the string are nested within each other.

TABLE II: Performance of LSTM Architectures on the Palindromic Language. Values are the median proportion of strings that are predicted correctly for each architecture across 10 different random initialisations. The differences between architectures in the last two columns are significant ($p < 0.01$) under a Mann-Whitney U test.

LSTM Architecture	In-Domain	Long	$N_A > 2$
Standard	100%	36%	76%
Convolutional	100%	100%	100%

Another model of such languages is a Push Down Automata (PDA), in which hierarchical structure is handled by pushing symbols representing the outer structures onto a stack, until the interior structure is completed, and then popping symbols back off the stack to move outward in the hierarchy until no more symbols remain. Crucially, the stack has a last-in-first-out structure that essentially returns items in the reverse order in which they were pushed onto it.

In principle, such a system can handle sentences of unbounded length, containing arbitrarily long dependencies between constituents. In practice, however, language users struggle with nested clauses more than two or three levels deep. Moreover, it is generally accepted that an ordinary CFG is not an accurate model of the grammatical structures that natural languages display, and their grammars appear, instead, to be mildly context sensitive [12].

However, here we focus on CFGs and the ability of Recurrent Neural Nets (RNNs) to learn these symbolic structures. In particular, we investigate the ability of a Long Short Term Memory [13] (LSTM) network to learn the simple palindromic language over the terminal symbols a, b, c, d and o, defined by the grammar in Fig. 2. We then show that the generalisation failures of that architecture can be overcome by imposing the appropriate form of symmetry.

We train the RNNs to predict the second half of strings generated by the CFG in Fig. 2 given the initial symbols up to the middle o. Our baseline model is a 100-unit LSTM network which we train on 100,000 examples of strings of length 15, 17, 19, 21, 23 and 25, and then perform an in-domain test on novel strings of the same length, and an out-of-domain test on longer strings of lengths 29, 33, and 37. We also retrain the LSTM after removing all strings which contain more than 4 tokens of the symbol a from the training set, i.e. those with a derivation containing more than two nested As. This model

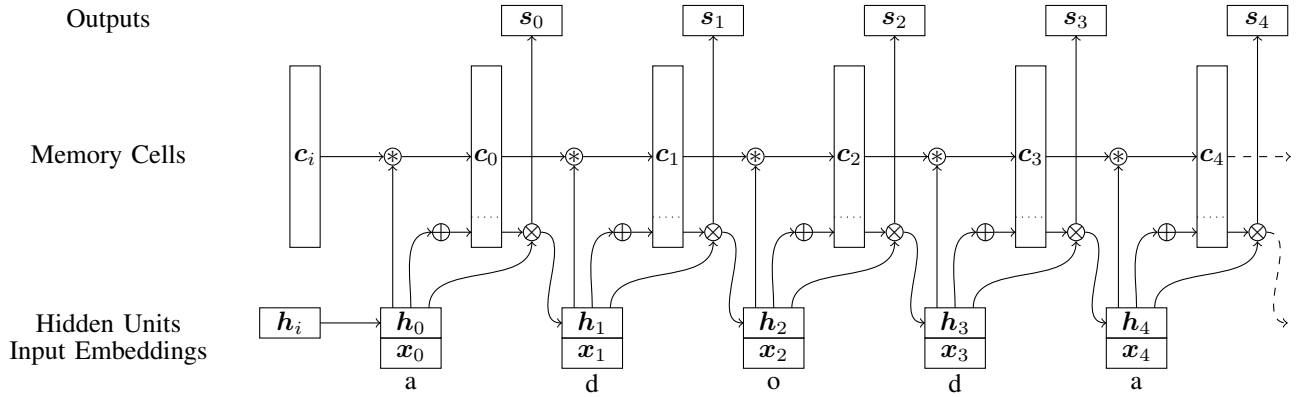

Fig. 3: The modified LSTM architecture is shown here unrolled in time from left to right. The memory cells, $\boldsymbol{c}$, have been organised into a stack (illustrated vertically) which a convolutional gate, $\circledast$, controls the flow of information through, while a standard multiplicative output gate, $\otimes$, and additive input gate, $\oplus$, read and write to the cells in the bottom of the stack. As in a standard LSTM, the hidden units, $\boldsymbol{h}$, are concatenated with the input embeddings, $\boldsymbol{x}$, to form the vector that feeds into the units that drive the various gates. The network is trained to predict the next symbol, and this results in it learning to push and pop symbols onto the stack as in a PDA. In the example, the first symbol, a, is written to the bottom of the stack at step 0. Encountering d at step 1, the contents of the stack are shifted upwards, and d written to the bottom. The o at step 2 indicates the first half of the string has ended and the stack is read, giving a d prediction at the output. This prediction is confirmed at step 3, and the stack is shifted down and a is read and predicted. At step 4, the stack is empty and the end of the string can be predicted. Whether the net is pushing or popping is controlled by the hidden state, $\boldsymbol{h}$, which switches after an o in the input. The stack is controlled by a width 3 convolution, which sets the value of each cell based on the previous values of that position and those above and below it. This allows information to be shifted up and down the stack.

is then tested only on examples from the test set containing more than 4 tokens of the symbol a. These out-of-domain tests are designed to evaluate whether the model has learned merely to reproduce structures seen in the training data, or whether the model has learned more abstract rules which can be generalised to all instances of the same dependencies.

The first row of Table II gives the results for this evaluation in terms of the proportion of strings that were predicted entirely correctly given the symbols up to the central o. The in-domain results, in the first column, make it clear that the net has learned the structure of the grammar, and how to make accurate predictions, at least for sequences of lengths seen at training time. The second column, containing the results for longer sequences, shows that generalisation outside the range of the training set is not robust. The third column indicates that the model has difficulty generalising to sequences where the nesting of A symbols is deeper than at training time, even though the actual length of sequences is unchanged.

These failures of generalisation can be seen as symptoms of the same underlying problem. In particular, success on each of these out-of-domain tasks simply requires extending the application of the same rules. However, the problem arises because the network lacks the required concept of sameness. The LSTM cells form an unstructured memory resource, without any notion of two cells containing the same information. Nor can there be a meaning to the idea of applying the same rule to that information. The parameters for each cell are learned independently, and so each cell carries out its own isolated task. During training these cells do learn to behave as a coherent whole, achieving impressive in-domain performance, but there is no way for the model to apply the same rule to the n^{th} item as it applied to the previous $n-1$.

A PDA handles all strings from its grammar systematically, because its memory consists of a stack of equivalent memory slots within which stored information can be shifted up or down. Pushing symbols further into the stack allows the completion of long-range dependencies to be deferred while the innermost short-range dependencies are dealt with. Importantly, this shifting process operates uniformly over all the slots, which we can think of in terms of a translational symmetry along the stack, and this uniformity is an important element in treating all dependencies equivalently.

In contrast, each memory cell of the LSTM is free to learn its own particular behavioral repertoire, which in many other situations provides the flexibility the problem at hand requires. However, in the case of the context free language, this freedom leaves the handling of structures not encountered during training unconstrained, and this prevents the LSTM from generalising systematically from simple strings to more complex ones. For example, each a symbol in the first half of an input string has to produce a distinct trace in the memory cells, which can later be matched to its counterpart in the second half of the string. Each of these traces, and their relation to the input, has to be learned independently in an LSTM, and when the depth of nesting exceeds that seen during training the network will typically lack the ability to extend its

behaviour in a systematic manner. Unlike the PDA, the LSTM lacks the uniformity of operation which underlies the idea of the same symbol being stored in multiple places in the stack and the same rule being applied to all dependencies.

To address this shortcoming, we propose to organize the memory cells into a linear ordered stack structure, and to use convolutions to control the flow of information across timesteps, replacing the forget gate. The translational symmetry of this convolutional layer gives meaning to the idea of the same symbol being able to be stored in different cells and we use a filter of width three to allow the shifting of stored information between adjacent cells. Along with a restriction on input and outputs only being connected to the bottom of the stack, this turns the LSTM into a differentiable PDA. In particular, spatial translations can be achieved in a convolution using the filters $[1, 0, 0]$ and $[0, 0, 1]$ and, using such operations, the network to learns to shift memory contents up and down the stack in order to perform push and pop operations.

As shown in Fig. 3, each token in the input is given an embedding, $\boldsymbol{x}$, which is then concatenated with the current hidden state, $\boldsymbol{h}$. Together these values form the inputs to units that control the flow of information into, out of and between the memory cells, $\boldsymbol{c}$, as in a standard LSTM. In this case, however, the memory cells are a set of 10 one channel one dimensional convolutional layers of 20 units each and the forget gate has been replaced with a set of width three filters that shape the recurrent flow of information. These filters are the softmax outputs of units driven by the concatenated input embeddings and hidden units, allowing the network to use the input context to control the memory cell stack. Information is written to and read out from only the bottom entries in the stack, using standard input and output gates. This architecture is described in more mathematical detail in Appendix B.

Performance of this convolutional LSTM on the same evaluations is given in the second row of Table II. There, all three columns show optimal performance on both the in-domain and out-of-domain tasks, demonstrating the utility of the convolutional layer in helping the model to generalise robustly. More abstractly, the symmetry across the memory cells allows the network to treat symbols stored throughout this stack uniformly, which in turn corresponds to handling both long and short range dependencies equivalently.

IV. Related Work

Numerous authors have tackled the problem of replicating the rule learning behaviour studied by Marcus et al. [3] in a connectionist system, and Alhama and Zuidema [11] give an extensive review. Many of these approaches rely on a specific training regime to obtain the desired behaviour, rather than our approach of modifying the architecture to embed the appropriate capacities innately. However, our core intention was to demonstrate the relevance of symmetry, with convolution being a convenient and transparent means to that end. The same end could conceivably be achieved purely through training on appropriately structured data.

The recurrent PDA we describe in Section III is very similar to a number of other architectures. Sun et al. [14] proposed a neural network pushdown automata. Grefenstette et al. [15] proposed architectures for a number of data structures: queues, dequeues and stacks. Joulin and Mikolov [16] proposed a recurrent stack structure, which in practice, is almost equivalent to our proposal. However, none of these works discuss the role of symmetry or the connection to convolution.

Symmetries beyond spatial translation have been discussed by a number of authors. Cohen and Welling [1] propose a generalisation of convolution for arbitrary discrete symmetries, such as reflections and rotations. The role of invariances in disentangled representations is discussed by Higgins et al. [2], and Bloem-Reddy and Teh [17] investigate the application of probabilistic symmetries to neural network architectures. Practical examples of symmetries supporting extrapolation and generalisation beyond the training set are discussed by Mitchell et al. [18].

Permutation invariance is relevant to a number of representational strategies, such as bag-of-words approaches [19] or Deep Sets [20]. However, the relevant symmetry in these cases is usually over permutations on the order of inputs, e.g. a symmetry between *wo fe fe* and *fe wo fe*. In our case, the permutation is over the identity of the symbols, i.e. a symmetry between *wo fe fe* and *la ti ti*.

V. Discussion

One way to address the criticisms of distributed approaches raised by Fodor and Pylyshyn [8] has been to focus on methods for binding and combining multiple representations [21]–[25] in order to handle constituent structure more effectively. Here, we instead examined the equally important role of symmetry in the systematicity of how those representations are processed, using a few simple proof-of-concept problems.

We showed that imposing a symmetry on the architecture was effective in obtaining the desired form of generalisation when learning simple rules, and simple grammars. In particular, we discussed two forms of symmetry relevant to the processing of symbols, corresponding respectively to the fact that all atomic symbols are essentially equivalent and the fact that any given symbol can be represented in multiple places, yet retain the same meaning. The first of these gives rise to a symmetry under substitutions of these symbols, which allows generalisation to occur from one symbol to another. The second gives rise to a symmetry across memory locations, which allows generalisation from simple structures to more complex ones.

On both problems, we implemented the symmetries using convolution. From a practical point of view, this allowed us to build networks using only long-accepted components from the standard neural toolkit. From a theoretical point of view, however, this implementation decision draws a connection between the cognition of space and the cognition of symbols.

The translational invariance of space is probably the most significant and familiar example of symmetry we encounter in our natural environment. As such it forms a sensible

foundation on which to build an understanding of other symmetries. In fact, Tarski [9] uses invariances under various spatial transformations within geometry as a starting point for their definition of *logical notion* in terms of invariance under all permutations. Moreover, from an evolutionary perspective, it is also plausible that there are common origins behind the mechanisms that support the exploitation of a variety of different symmetries, including potentially spatial and symbolic. In addition, recent research supports the idea that cerebral structures historically associated with the representation of spatial structure, such as the hippocampus and entorhinal cortex, also play a role in representing more general relational structures [26], [27].

Thus, our use of convolution is not merely a detail of implementation, but also an illustration of how spatial symmetries might relate to more abstract domains. In particular, the recursive push down automata, discussed in Section III, utilises push and pop operations that relate fairly transparently to spatial translations. Of course, a variety of other symmetries, beyond translations, are likely to be important in human cognition, and an important challenge for future research will be to understand how symmetries are discovered and learned empirically, rather than being innately specified.

A common theme in our exploration of symmetry, was the ability it conferred to separate content from structure. Imposing a symmetry across symbols or memory locations, allowed us to abstract away from the particular content represented to represent the structure containing it. So, for example the grammar rule learned by our network on the syllable sequences of Marcus et al. [3] was able to generalise from seen to unseen syllables because it represented the abstract structure of ABB and ABA sequences, without reference to the particular syllables involved.

We explored how this ability could also be exploited on a grammar learning task, but it is likely that there are many other situations where such a mechanism would be useful. Future work will examine how this approach can be applied more widely beyond the simple toy problems considered here. Particular attention is needed to the problem of robust learning within noisy and complex environments.

Appendix A
Symmetry

Informally, a symmetry of a system is a mapping of the system onto itself which preserves its fundamental properties. In the case of translation symmetry in the visual domain, we have input images, x, and output labels, y, and we want to learn a function, $f(x)$, which predicts these labels, and is invariant to spatial translations, T. That is, we want f to obey $f(Tx) = f(x)$.

Typically, we achieve this by composing two types of function: equivariant convolutions, c, and invariant poolings, p. Equivariant here means that the output from a translated input is itself the translation of the original output: $c(Tx) = Tc(x)$. While invariant means the output is unchanged by input translations: $p(Tx) = p(x)$.

When the width of the convolution is reduced to one, the function, $c()$, becomes equivariant to all permutations, S, not just translations: $c(Sx) = Sc(x)$. Permutation equivariance also arises, for example, in formal logic, where the rules of deduction depend not on the particular names within an expression, but on its logical structure. So, *Socrates is mortal* follows from *Socrates is a man* and *All men are mortal* not because of the meaning of *Socrates* or *mortal*, but because the syllogism has the right form. Thus, if x represents the premises and y represents the conclusions, then the process of deduction $d(x) = y$ should be equivariant under any substitution, S, of names. That is, we should be able to reach an equivalent conclusion even if we rename *Socrates* as *Bob*, and so $d(Sx) = Sd(x)$.

Symmetries also arise in computational processes. For example, Fleck [28] discusses how the symmetries of an automata are related to those of its associated language. In the case of our palindromic language, the translational symmetry of the PDA relates to the recursive nesting of the grammar. Thus, if $\pi(S)$ is the sequence of states the PDA traverses in processing the palindrome S and $\pi_x(S)$ is the states traversed when S is preceded by x, then there is a mapping ϕ_x between the two sets of states. In other words, $\pi(xSx) = \pi(x) + \pi_x(S) + \pi_{xS}(x)$ and $\phi_x(\pi(S)) = \pi_x(S)$. Here, ϕ_x corresponds to pushing the symbol x onto the stack, allowing the inner dependencies in S to be processed. Moreover, recursion allows nesting of any finite depth. So, for example, in the context $xySyx$, the sequence of states while processing S is $\phi_x(\phi_y(\pi(S)))$. Translational symmetry of the stack corresponds to the fact that whenever the same symbol, x, is pushed onto the stack, the same mapping, ϕ_x, applies.

Appendix B
Context Free Language Learning

Fig. 3 gives a visual overview of the architecture we apply to learning the simple palindromic language. This is essentially a modified LSTM in which the forget gate has been replaced with convolutional filters. We define this explicitly below.

Each token in the input is given an M-dimensional embedding, $\boldsymbol{x}$, which is concatenated with the current N-dimensional hidden state, $\boldsymbol{h}$, to give a vector, $\boldsymbol{g}$, representing the current context.

$$\boldsymbol{g}_t = \boldsymbol{h}_t \oplus \boldsymbol{x}_t \tag{1}$$

This controls N single channel width three filters, $\boldsymbol{f}$, each of which is the output of a softmax.

$$\boldsymbol{f}_{n,t} = \text{softmax}\left(\boldsymbol{W}_{f,n}\boldsymbol{g}_t + \boldsymbol{b}_{f,n}\right) \tag{2}$$

As in a standard LSTM, values written to the cells are the outputs of tanh units gated by a sigmoid.

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{W}_i\boldsymbol{g}_t + \boldsymbol{b}_i\right) \circ \tanh\left(\boldsymbol{W}_c\boldsymbol{g}_t + \boldsymbol{b}_c\right) \tag{3}$$

And the output gate is also a sigmoid function.

$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{W}_o \boldsymbol{g}_t + \boldsymbol{b}_o\right) \tag{4}$$

Recurrence between the cells in each of the N memory stacks is based on one-dimensional convolution and controlled by the filters $\boldsymbol{f}$.

$$\boldsymbol{c}_{n,t} = \boldsymbol{f}_{n,t} \circledast \boldsymbol{c}_{n,t-1} \tag{5}$$

The 0^{th} values in each stack are updated using the values of $\boldsymbol{i}$.

$$c_{n,t,0} = c_{n,t,0} + i_{t,n} \tag{6}$$

And the new hidden state are also read out from the 0^{th} values, gated by tanh units.

$$h_{t+1,n} = o_{t,n} \cdot \tanh\left(c_{n,t,0}\right) \tag{7}$$

The final outputs, to predict a one hot vector representing the next symbol, apply a softmax to these new hidden states.

$$\boldsymbol{s}_t = \text{softmax}\left(\boldsymbol{W}_s \boldsymbol{h}_{t+1} + \boldsymbol{b}_s\right) \tag{8}$$

The loss is the cross entropy over the second half of the sequence, and for this task M and N are both 10, with the memory stack having a depth of 20. We train the network for 100 epochs with batch size 128 using the Adam [10] optimiser with a rate of 0.001. The network weights are re-initialised and training restarted if the validation accuracy has not exceeded 0.9 by the end of the fourth epoch.

REFERENCES

[1] T. Cohen and M. Welling, "Group equivariant convolutional networks," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 2990–2999. [Online]. Available: http://proceedings.mlr.press/v48/cohenc16.html

[2] I. Higgins, D. Amos, D. Pfau, S. Racaniere, L. Matthey, D. Rezende, and A. Lerchner, "Towards a Definition of Disentangled Representations," *arXiv e-prints*, p. arXiv:1812.02230, Dec 2018.

[3] G. F. Marcus, S. Vijayan, S. Bandi Rao, and P. M. Vishton, "Rule learning by seven-month-old infants," *Science*, vol. 283, no. 5398, pp. 77–80, 1999.

[4] J. Haugeland, *Artificial Intelligence: The Very Idea*. Cambridge, MA, USA: Massachusetts Institute of Technology, 1985.

[5] W. James, *The Principles of Psychology*, ser. American science series. H. Holt, 1890, no. v. 1. [Online]. Available: https://books.google.co.uk/books?id=dLb2xQEACAAJ

[6] G. Boole, *An Investigation of the Laws of Thought: On Which Are Founded the Mathematical Theories of Logic and Probabilities*. Walton and Maberley, 1854.

[7] G. F. Marcus, *The Algebraic Mind*. MIT Press, 2001.

[8] J. A. Fodor and Z. W. Pylyshyn, "Connectionism and cognitive architecture: A critical analysis," *Cognition*, vol. 28, no. 1-2, pp. 3–71, 1988.

[9] J. Corcoran and A. Tarski, "What are logical notions?" *History and Philosophy of Logic*, vol. 7, no. 2, pp. 143–154, 1986.

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[11] R. G. Alhama and W. Zuidema, "A review of computational models of basic rule learning: The neural-symbolic debate and beyond," *Psychonomic Bulletin & Review*, vol. 26, no. 4, pp. 1174–1194, Aug 2019. [Online]. Available: https://doi.org/10.3758/s13423-019-01602-z

[12] S. M. Shieber, "Evidence against the context-freeness of natural language," *Linguistics and Philosophy*, vol. 8, pp. 333–343, 1985.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, Nov. 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[14] G.-Z. Sun, C. L. Giles, and H. H. Chen, "The neural network pushdown automaton: Architecture, dynamics and training," in *Adaptive Processing of Sequences and Data Structures, International Summer School on Neural Networks, "E.R. Caianiello"-Tutorial Lectures*. London, UK, UK: Springer-Verlag, 1998, pp. 296–345. [Online]. Available: http://dl.acm.org/citation.cfm?id=647638.733213

[15] E. Grefenstette, K. M. Hermann, M. Suleyman, and P. Blunsom, "Learning to transduce with unbounded memory," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, pp. 1828–1836. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969442.2969444

[16] A. Joulin and T. Mikolov, "Inferring algorithmic patterns with stack-augmented recurrent nets," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, pp. 190–198. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969239.2969261

[17] B. Bloem-Reddy and Y. W. Teh, "Probabilistic symmetry and invariant neural networks," *CoRR*, vol. abs/1901.06082, 2019. [Online]. Available: http://arxiv.org/abs/1901.06082

[18] J. Mitchell, P. Stenetorp, P. Minervini, and S. Riedel, "Extrapolation in NLP," in *Proceedings of the Workshop on Generalization in the Age of Deep Learning*. New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 28–33. [Online]. Available: https://www.aclweb.org/anthology/W18-1005

[19] L. White, R. Togneri, W. Liu, and M. Bennamoun, "How well sentence embeddings capture meaning," in *Proceedings of the 20th Australasian Document Computing Symposium*, ser. ADCS '15. New York, NY, USA: ACM, 2015, pp. 9:1–9:8. [Online]. Available: http://doi.acm.org/10.1145/2838931.2838932

[20] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 3391–3401. [Online]. Available: http://papers.nips.cc/paper/6931-deep-sets.pdf

[21] P. Smolensky, "Tensor product variable binding and the representation of symbolic structures in connectionist systems," *Artif. Intell.*, vol. 46, no. 1-2, pp. 159–216, Nov. 1990.

[22] G. E. Hinton, "Mapping part-whole hierarchies into connectionist networks," *Artif. Intell.*, vol. 46, no. 1-2, pp. 47–75, Nov. 1990. [Online]. Available: http://dx.doi.org/10.1016/0004-3702(90)90004-J

[23] T. Plate, "Holographic reduced representations: Convolution algebra for compositional distributed representations," in *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*, ser. IJCAI'91. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991, pp. 30–35.

[24] J. B. Pollack, "Recursive distributed representations," *Artif. Intell.*, vol. 46, no. 1-2, pp. 77–105, Nov. 1990. [Online]. Available: http://dx.doi.org/10.1016/0004-3702(90)90005-K

[25] J. E. Hummel and K. J. Holyoak, "Distributed representations of structure: A theory of analogical access and mapping," *PSYCHOLOGICAL REVIEW*, vol. 104, no. 3, pp. 427–466, 1997.

[26] T. Behrens, T. Muller, J. Whittington, S. Mark, A. Baram, K. Stachenfeld, and Z. Kurth-Nelson, "What is a cognitive map? organizing knowledge for flexible behavior." *Neuron*, vol. 100, no. 2, pp. 490–509, 2018.

[27] M. Duff and S. Brown-Schmidt, "The hippocampus and the flexible use and processing of language," *Frontiers in Human Neuroscience*, vol. 6, p. 69, 2012. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnhum.2012.00069

[28] A. C. Fleck, "On the automorphism group of an automaton," *J. ACM*, vol. 12, no. 4, p. 566–569, Oct. 1965. [Online]. Available: https://doi.org/10.1145/321296.321307

Acknowledgment: This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 741134