CHAPTER 11

Unitary Recurrent Networks: Algebraic and Linear Structures for Syntax

Jean-Philippe Bernardy

University of Gothenburg

Shalom Lappin

University of Gothenburg, Queen Mary University of London, and King's College London

CONTENTS

11.1 Introduction 252
11.2 Two Flavours of Algebraic Structures 253
11.2.1 Syntax-semantics interface 254
11.2.2 Sequence algebras and parsing 254
11.2.3 Linear Algebra 257
11.2.4 Linear operators as charts, charts as phrase embeddings 258
11.3 Two Models 258
11.3.1 LSTM 259
11.3.2 URN 260
11.4 Theoretical Analysis of URN models 262
11.4.1 Compositionality 262
11.4.2 Unit vectors 262
11.4.3 Long-distance memory 262
11.4.4 Distance and Similarity 263
11.4.5 Geometric interpretation of embeddings 264
11.4.5.1 Signature of Embeddings 265
11.4.6 Average Effect 265
11.4.7 Truncation 266
11.4.8 Projection 266
11.5 Experiments 266
11.5.1 Cross-Serial dependencies 267
11.5.1.1 Results 267
11.5.2 Generalised Dyck Languages 268
11.5.2.1 Results 270
11.5.2.2 Analysis 271
11.5.2.3 Composition of Matching Parentheses 271
11.5.3 Natural Language Long-Distance Agreement Task 273

11.5.3.1 Analysis 274
11.6 Related Work 276
11.6.1 Learning Agreement 277
11.6.2 Cross-serial patterns 278
11.6.3 Quantum-Inspired Systems 278
11.6.4 Tensor Recurrent Neural Networks 279
11.6.5 Unitary-Evolution Recurrent Networks 279
11.7 Conclusions and Future work 279
11.8 Acknowledgements 281

ABSTRACT

The emergence of powerful deep learning systems has largely displaced classical symbolic algebraic models of linguistic representation in computational linguistics. While deep neural networks have achieved impressive results across a wide variety of AI and NLP tasks, they have become increasingly opaque and inaccessible to a clear understanding of how they acquire the generalisations that they extract from the data to which they apply. This is particularly true of BERT, and similar non-directional transformers. We study an alternative deep learning system, Unitary-Evolution Recurrent Neural Networks (URNs) (Arjovsky et al., 2016), which are strictly compositional in their combination of state matrices. As a result they are fully transparent. They can be understood entirely in terms of the linear algebraic operations that they apply to each input state matrix to obtain its output successor. We review these operations in some detail, clarifying the compositional nature of URNs. We then present experimental evidence from three NLP tasks to show that these models achieve an encouraging level of precision in handling long distance dependencies. The learning required to solve these tasks involves acquiring and representing complex hierarchical syntactic structures.

11.1 INTRODUCTION

Theoretical linguistics owes two major ideas to the work of Montague (1970a,b). The first is the general hypothesis that semantics is amenable to mathematical formulation. The second is the view that the semantics of a language is a function defined compositionally over its syntax. Realising Montague's program requires that (1) the syntax-semantics interface is expressed as an algebraic structure, and (2) semantic interpretation is achieved through such a structure. We will explore in some detail what these conditions entail.

The emergence of linguistic models based on deep learning have brought considerable improvements in terms of coverage and predictive power, compared to tools based on algebraic theories (Moss, 2022, Chapter 10 of this volume). Unfortunately, using such models has generally means giving up Montague's principles of compositionality. At a surface level, even though the model is expressed as a well-defined function from an input string to a (task-dependent) prediction, this function pos-

$$Q = \frac{1}{2}\begin{pmatrix} \sqrt{2} & -\sqrt{2} & 0 \\ \sqrt{2} & \sqrt{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$
$$s = \frac{1}{2}(0, 1, \sqrt{3})$$
$$Qs = \frac{1}{4}(-\sqrt{2}, \sqrt{2}, 2\sqrt{3})$$

Figure 11.1: Example of one step of unitary evolution

sesses no particular property which conditions its behaviour predictably on various classes of input. Therefore it is arduous, though not impossible, to infer insights about such a model, from a theoretical linguistic perspective. Although it is not impossible that deep-learning models embed a syntactico-semantic interface, in most cases the discrete types of information that the models learn and represent remain opaquely encoded. Concomitantly, the non-linear activation functions that deep learning models apply between the each layer prevent transparent tracking of this information. This is particularly frustrating in light of the fact that prediction functions are specified through the operations of linear algebra, which carry a rich set of tools for structural analysis.

In this chapter, we advocate the Unitary Recurrent Network (URN) as a transparent alternative model of deep learning. This architecture was first proposed by Arjovsky et al. (2016) to solve the problem of exploding and vanishing gradients when applying back propagation in a recurrent neural network, thanks to the norm-preserving properties of unitary matrices (see Fig. 11.1 for an example). Its algebraic properties give reason to think that the framework admits of interesting theoretical analysis. The rest of the chapter shows that the URN can be trained to learn syntactic structure, and that the trained models are amenable to white-box analysis.

11.2 TWO FLAVOURS OF ALGEBRAIC STRUCTURES

In general, an algebraic structure is defined by

- a collection of (types of) operations parameterized over a type a –the carrier set, and
- a number of axioms specifying the above operations

For example, as an algebraic structure, a monoid is defined by the two operations $\epsilon : a$ and $(\bullet) : a \to a \to a$, such that

$$\forall x.\, \epsilon \bullet x = x \qquad \text{(left identity)}$$
$$\forall x.\, x \bullet \epsilon = x \qquad \text{(right identity)}$$
$$\forall x\, y\, z.\, x \bullet (y \bullet z) = (x \bullet y) \bullet z \qquad \text{(associativity)}$$

Any given monoid is defined by a *particular* tuple $(a, \epsilon, (\bullet))$. For instance, $(\mathbb{R}, 0, (+))$ is the usual additive monoid over real numbers.

11.2.1 Syntax-semantics interface

As we have observed, a syntax-semantics interface is defined by an algebraic structure. A simple way to construct such an interface is to encode a context-free grammar, with an indexed carrier set. A rule like $S ::= NP\ VP$ corresponds to the type $a_{NP} \to a_{VP} \to a_S$. Among others, Ranta (2004) uses this kind of coding. Any instance of this structure produces a particular semantics. In our example, this involves assigning semantic types to all the syntactic categories (a_{NP}, a_{VP}, a_S, etc.), and defining valid operations for each of the rules. It should be stressed here that the such meaning representations are not limited to the context-free case.

In general, if the algebra $Fa \to a$ is the syntax-semantic interface, then the initial algebra μF is the set of associated syntactic representations, while any function $\phi : F\alpha \to \alpha$ is a compositional semantics over such a syntax.[1]

Montague (1974) provides a semantics for quantifiers by means of a continuation monad carrying truth-valued effects. (Barker and Shan, 2004; Groote, 2001). Montague left the type of individuals abstract, but it can be further instantiated to vector spaces, with various interpretations. (Bernardy, Blanck, et al., 2018; Emerson and Copestake, 2016, 2017; Grefenstette et al., 2011; Grove and Bernardy, 2021)

11.2.2 Sequence algebras and parsing

On Montague's view, semantics is based on syntactic structure. However, languages do not come with labelled syntactic structures. Rather, basic linguistic data consists of sequences of symbols. In fact, this is assumption already comes with a fair amount of theoretical bias and abstraction, but we will take it as given.

The mapping of raw linguistic data to syntactic structure has been widely studied in the history of parsing. Let's consider an algebraic formulation of the parsing task. The structure of linguistic input, a sequence of symbols, can be captured as a sequence algebra, defined as follows.

Definition 11.1 (Sequence algebra)**.** *A sequence algebra over a set of symbols* Σ *consists of three operations:*

$$\begin{aligned} \mathit{embed} &: \Sigma \to a && \textit{embedding a symbol} \\ \epsilon &: a && \textit{empty sequence} \\ (\bullet) &: a \to a \to a && \textit{concatenation} \end{aligned}$$

together with the same laws that define monoids.

(That is, it is a free monoid over the set Σ). The monoid operations and laws express the linear character of a sequence. In particular the laws ensure that the order (and, in general, the structure) of concatenations plays no role in the meaning of a sequence.

The set of sequences is the associated initial algebra. A *parser* is any instance of a sequence algebra with the carrier set a being instantiated to the syntactic structure of interest, for the task at hand.[2] It should be emphasised that this definition entails

[1] See Chapter 10 for more algebraic background.

[2] Typically, a parser is also allowed to fail, but we'll ignore that case here.

that the parser is strictly sequence-compositional, in the sense that one can parse any sub-sequence, and compose it with the parse result of any other sub-sequence. This might seem counter-intuitive if one is accustomed to thinking of a parser as an automaton processing the string in a left-to-right order, whose behaviour is described by a state-transition function

$$f : \Sigma \to s \to s.$$

However, there is no incompatibility between these two characterisations of parsing. They are functionally equivalent, but they express distinct formal perspectives on the way that parsing operates.

Definition 11.2 (Automaton from sequence algebra)**.** *Assume a sequence algebra* $(a, embed, \epsilon, \bullet)$*. Then we can construct an equivalent parser automaton by letting the internal state be the carrier set:*

$$s = a$$
$$f\, x\, s = s \bullet embed x$$

Definition 11.3 (Sequence algebra from automaton)**.** *Assume an automaton internal state* s*, whose behaviour is captured by a state-transition function* $f : \Sigma \to s \to s$*. We can define an equivalent sequence algebra by letting the carrier type be the set of state-transition functions:*

$$a = s \to s$$
$$embed = f$$
$$\epsilon = id$$
$$w_1 \bullet w_2 = w_2 \circ w_1$$

It should however be noted that, for any non-trivial set s, the set $a = s \to s$ is extremely large.[3] The reformulation of a sequential automaton as a sequence algebra produces a computationally expensive result, unless there happens to be an efficient representation of transition functions over s. There are, then, practical tradeoffs in choosing one approach over the other.

As we see it, the advantages of the automaton approach is that if the set s is small, then it can be implemented by a sequential algorithm in a way which is memory efficient. It also corresponds more closely to psycholinguistic models which rely on sequential processing of inputs.

The sequence-algebraic approach has also advantages. First, it it can be implemented as a divide-and-conquer algorithm. The input sequence can be split arbitrarily. Both sub-sequences can be parsed independently, and the results recombined using $(\bullet)$. If this recombination operator is efficient, then one obtains an algorithm cost effective overall. Valiant (1975) takes advantage of this principle to construct the most time-efficient algorithm for general context-free parsing. The complexity of Valiant's algorithm is sub-cubic. Cubic, or even sub-cubic, complexity is unrealistic for human processing, and so context-free grammars may seem to be a poor

[3] if the cardinality of s is $|s|$, then the cardinality of a is $|s \to s| = |s|^{|s|}$

model for human language syntax. One can, of course, resort to heuristic methods to improve complexity in many situations; but Bernardy and Claessen (2013, 2015) demonstrate a general result in this regard. They show that if the charts are represented by appropriate sparse matrices and input strings are organised hierarchically, then the complexity of Valiant's algorithm becomes linear.

The second advantage of the sequence-algebraic approach for parsing is that one can analyse any value of a out-of-context. In fact, because it can be combined in any context, a value of this type must include, in one form or another, possible interpretations for all the contexts that it can possibly interact with. In practice, such an analysis is possible only if the structure of a is simpler than arbitrary state-transition functions ($s \to s$). We will show later how to leverage this theoretical property for URN models. In the case of Valiant's algorithm, the type a is realised by a parse chart. For any sequence of symbols w, the chart $\phi_{valiant}(w)$ contains all the possible syntactic representations of all possible subsequences contained in w. While this sounds expensive, Bernardy and Claessen (2013, 2015) show that this chart is asymptotically (very) sparse for hierarchically organised inputs.

Third, the sequence algebra formulation allows for parallel evaluation. It is possible to compute it at symbols using *embed*, then group those results in adjacent pairs, for each in parallel, then proceed upwards until one has a result for the full input sequence. This can be crucial when using modern parallel hardware for evaluation.

A sequence-algebra need not take the form of a usual parser. It can map a sequence of symbols directly to a semantic value without invoking an intermediate syntactic representation. Such models are now referred to as "end-to-end" systems. They connect one end of a process (sequences of symbols) to the task-specific semantics (the carrier type a). Our focus here is to construct and analyse such systems using an URN architecture.

We have pointed out that an RNN is fully defined by a transition function, of the type that we have identified above:

$$f : \Sigma \to s \to s.$$

We call such an operation f an evolution function. If the RNN acts as a language model, then it also contains a prediction function whose role is to predict the next symbol, given the current state. More precisely, it predicts a probability score for each symbol

$$p : s \to \Sigma \to \mathbb{R}$$

The set of these scores is then turned into a probability distribution through a softmax function:

$$\begin{aligned} \mathsf{softmax} &: (\Sigma \to \mathbb{R}) \to (\Sigma \to [0,1]) \\ \mathsf{softmax}\, q\, x &= \frac{e^{q\,x}}{\sum_{x \in \Sigma} e^{q\,x}} \end{aligned}$$

While RNNs can be neatly reformulated as sequence-algebras (even if inefficient ones), transformer-like deep learning models cannot be easily expressed in these

terms. Indeed, a transformer essentially consists of several layers of attention heads, and the output of each attention head head depends on all input positions. This global dependence pattern is incompatible with a sequence algebra. Such algebras require that any input is represented independently of its larger context, in a way that depends solely on its constituents, and its structure.

11.2.3 Linear Algebra

To clarify the structure and interpretation of URNs we will review some of the concepts of linear algebra that we will apply. In fact, linear algebra is central to deep neural networks in general, given that they operate through operations on vectors and matrices. These concepts tend to be suppressed as assumed in much current work on deep learning. It is worth highlighting them here to indicate the differences between URNs and other deep learning networks.

Algebraically, a vector space over a field s is defined by a vector addition $(+) : a \to a \to a$, multiplication by a scalar $(*) : s \to a \to a$, and the vector $0 : a$ such that $(+)$ is associative, commutative, invertible, and has 0 as a left and right unit. Additionally, multiplication by a scalar is required to compatible with scalar multiplication; and vector addition must distribute over multiplication by a scalar and *vice versa*. Finally, the scalar unit must be the unit of multiplication by a scalar.

In practice, one seldom works with the algebraic definition. Rather, one uses a an array of numbers, say $\mathbb{R}^n$. The reason for doing so is that, given a basis $\vec{e_i}$ of a vector space of dimension n, the set of vectors is isomorphic to the set of representations $\mathbb{R}^n$, with

$$\vec{x} = \sum_i x^i \vec{e_i}$$

But our main object of interest is *linear maps*, which are defined as homomorphisms between two vector spaces (when the source and target spaces are identified, one speaks of a *linear operator*). Again, we do not work with this definition, but rather with representations in the form of matrices of numbers $\mathbb{R}^n_m$, for the same reason that we use arrays of numbers instead of vector spaces as defined above.

Given appropriate bases $\vec{e_i}$ and $\vec{d_i}$ for the source and target spaces, matrix representations are in isomorphic relations with linear maps:

$$F(\vec{x}) = F\left(\sum_i x^i \vec{d_i}\right) = \sum_j \vec{e}_j F^j_i x^i$$

The components of the image of F are obtained by matrix-vector multiplication:

$$F(\vec{x})^j = \sum_j F^j_i x^i$$

These definitions are intended as formal background that is often omitted in discussions of deep neural networks. For a detailed introduction to linear algebra, we recommend consulting standard references (Gantmacher, 1959; Strang, 2016).

To clarify, we are *not* suggesting that vector algebra specifies the syntax-semantics interface for a linguistic theory. It plays an entirely different role in our model.

11.2.4 Linear operators as charts, charts as phrase embeddings

We will be making use of the fact that linear operators (linear maps to and from the same vector space v) form a monoid under composition.

$$\begin{aligned} a &= v \to v \\ \epsilon &= id \\ F \bullet G &= G \circ F \end{aligned}$$

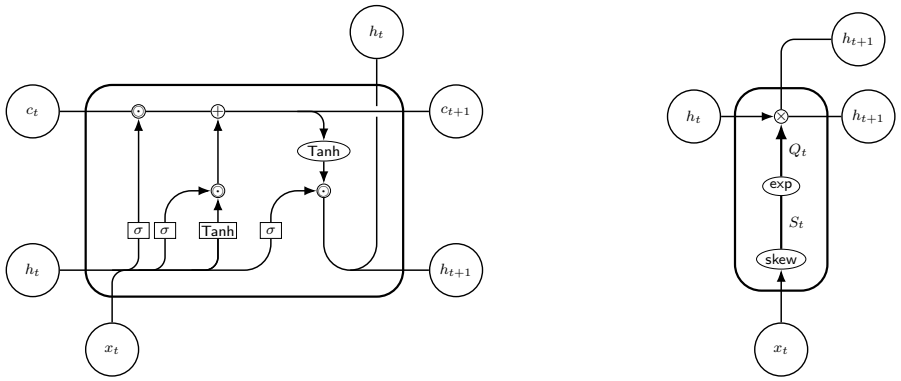
Notice that the composition of a linear operators is another linear operator. Consequently, one can represent such a composition as a matrix, which is the product of the representations from which it is formed. The corresponding monoid is:

$$\begin{aligned} a &= \mathbb{R}^n_n \\ \epsilon &= I \\ F \bullet G &= G \times F \end{aligned}$$

By also defining a function $embed : \Sigma \to \mathbb{R}^n_n$ we obtain a matrix-based representation of charts. Because the number of dimensions in such a chart is n^2 (and not infinite as it would be in the general case), it is possible that it is amenable to a *context-independent* analysis. We will use a deep-learning approach to construct *embed* – an appropriate word-embedding – and the methods of linear algebra to analyse the charts. For reasons that will become clear later, we restrict ourselves to *unitary* operators as charts. Such a chart essentially acts as a representation for the input substring that it is built from, and we generally refer to it as the embedding for the substring.

The adjective "unitary" refers to the fact that the norm of the determinant is the unit. It also indicates that the underlying scalar field consists of complex numbers. We will, in fact, restrict the scalar field to real numbers, which permits a more vivid geometrical representation of some of our data. Therefore, the matrices that we employ in URNs are actually orthogonal. We retain the description "unitary" because it now enjoys wide currency in the deep learning literature to refer to URN-like models.[4]

Definition 11.4 (Orthogonal matrix)**.** *A matrix Q is orthogonal iff it is square and $Q^TQ = I$.*



(a) LSTM (b) URN

Figure 11.2: Schematic representation of RNN cells. In the above, square boxes represent dense layers— with a trainable matrix of weights— followed by an activation function (sigmoid or hyperbolic tangent). Circles and ellipses represent bare functions, with no associated dense layer. Arrows carrying matricial information are drawn thicker. Accordingly, the symbol $\times$ corresponds to matrix-vector multiplication, while $\odot$ corresponds to the Hadamard product of vectors.

11.3 TWO MODELS

We consider two generative models: an LSTM (Hochreiter and Schmidhuber, 1997) and an URN (Arjovsky et al., 2016). We show them schematically in Fig. 11.2.

We proceed to describe their evolution functions as mapping from a hidden state h_t to the next state (h_{t+1}):

$$f\, x_t\, h_t = h_{t+1}$$

The expression for h_{t+1} depends on the RNN cell used.

[4]The choice between real and complex fields is not significant for our purposes. In geometric terms (planes, rotations, etc.) the results are the same. A plane can be represented either by two real vectors, or by one complex vector. A rotation can be expressed by a unimodular complex number or a real angle, etc. Our results also apply to unitary matrices with complex numbers. Beyond the easier geometric interpretation that it affords, we adopt real matrices for two reasons. First, on the implementational side, we found that the TensorFlow library, which we use in our experiments, does not fully support complex numbers. Second, from the theoretical perspective, the description of our models does not require reference to complex numbers. To include them would introduce needless complexity to our account. The precise connection between unitary and orthogonal matrices is explained in, for example, Gantmacher (1959, p. IX.13).

11.3.1 LSTM

To highlight the contrast between LSTMs and URNs, consider, the LSTM (Hochreiter and Schmidhuber, 1997), which is defined as follows:

$$\begin{aligned} v_t &= h_t \diamond x_t \\ f_t &= \sigma(W_f v_t + b_f) \\ i_t &= \sigma(W_i v_t + b_i) \\ o_t &= \sigma(W_o v_t + b_i) \\ \tilde{c}_t &= \sigma(W_c v_t + b_c) \\ c_{t+1} &= (f_t \odot c_t) + (i_t \odot \tilde{c}_t) \\ h_{t+1} &= (o_t \odot \mathsf{tanh}(c_{t+1})) \end{aligned}$$

Here σ refers to the sigmoid function and $(\diamond)$ is vector concatenation.[5]

We observe that the conversion of an LSTM to a sequence algebra (Definition 11.3) would not be interesting or useful. Because it employs non-linear transition functions (sigmoid and hyperbolic tangent), the set of transition functions does not allow a simpler representation than a set of general functions from vectors to vectors.

11.3.2 URN

We define our version of an URN as follows:

$$\begin{aligned} h_{t+1} &= Q_t h_t \\ S_t &= skew(x_t) \\ Q_t &= e^{S_t} \end{aligned}$$

$skew(x)$ is a function that takes a vector and produces a skew-symmetric (Definition 11.5) matrix by arranging the elements of x in a triangular pattern. For example, with an input vector of size 3, we have:

$$skew(x) = \begin{pmatrix} 0 & x_0 & x_1 \\ -x_0 & 0 & x_2 \\ -x_1 & -x_2 & 0 \end{pmatrix}$$

The upper triangle of S_t is provided by the previous layer (typically the word embedding layer), and its lower triangle is its negated symmetric. This setup ensures that S_t is anti-symmetric. By Theorem 11.2, this entails that Q_t is unitary.

Definition 11.5 (Skew-symmetric matrix)**.** *A square matrix S is skew-symmetric iff $S^T = -S$.*

Lemma 11.1. *Assume that S is a real and skew-symmetric matrix. Then the eigenvalues of S are pure imaginary.*

[5] During training we apply dropout to the vectors h_t and x_t for every timestep t.

Proof. Assume u a complex vector, and λ a complex number, such that u is an eigenvector of S with corresponding eigenvalue λ. By definition, $Su = \lambda u$. We have $u^*Su = \lambda u^*u = \lambda\|u\|^2$. Taking the hermitian conjugate of both sides yields $u^*S^*u = \bar{\lambda}\|u\|^2$. Because S is skew-symmetric, we also have $u^*(-S)u = \bar{\lambda}\|u\|^2$, and in turn $u^*Su = -\bar{\lambda}\|u\|^2$. We can then conclude that $\lambda = -\bar{\lambda}$, which is satisfied only when λ is pure imaginary. □

Theorem 11.2. *If S is skew-symmetric then e^S is orthogonal. Additionally, the rank of S gives the maximum number of eigenvalues of e^S different from 1.*

Proof. By the spectral theorem, S admits a unitary diagonalisation. By Lemma 11.1, the eigenvalues are pure imaginary, and thus we have $S = U^*(i\Theta)U$, with Θ real and diagonal. Let the diagonal elements of Θ be θ_j. We can then compute:

$$\begin{aligned} e^S &= \sum_{n=0}^{\infty} \frac{S^n}{n!} \\ &= \sum_{n=0}^{\infty} \frac{(U^* i\Theta U)^n}{n!} \\ &= \sum_{n=0}^{\infty} \frac{U^*(i\Theta)^n U}{n!} \\ &= U^* \left(\sum_{n=0}^{\infty} \frac{(i\Theta)^n}{n!} \right) U \\ &= U^* e^{i\Theta} U \end{aligned}$$

Thus, the eigenvalues of e^S are $\lambda_j = e^{i\theta_j}$, and each of them is unimodular. Consequently, e^S is unitary. We also know that it is a real matrix, and thus it is orthogonal.

To prove the second part of the theorem, we note that if $\theta_j = 0$ then $\lambda_j = 1$. Because the rank of S gives the number of non-zero elements of Θ, it is also the maximum number of elements of $e^{i\Theta}$ different from 1. These numbers can differ when $\theta_j = 2\pi$ for some j. □

We call Q_t the *orthogonal embedding* of the input symbol at position t.

In contrast to an LSTM, it is useful to specify an URN as a sequence algebra:

$$\begin{aligned} \mathit{embed}\, x &= e^{\mathit{skew}(v(x_t))} \\ \epsilon &= I \\ w_1 \bullet w_2 &= w_2 \times w_1 \end{aligned}$$

Assuming that the hidden state vectors have dimension n, the number of dimensions of the carrier type is $n \times (n-1)/2$, due to the orthogonal restriction. This is a notable improvement over an LSTM. Another consequence of this restriction, which Arjovsky et al. (2016) suggest as motivation for URNs, is that they are not subject to exploding or vanishing gradients.[6]

[6] In fact, this feature is present in all linear models (as defined in Section 11.2.4), but URNs enjoy a stronger property. This property is that the gradients wrt. the hidden state is constant throughout timesteps. Arjovsky et al. (2016) provide a proof for a more general result.

In what follows, we let n be the dimension of the state vectors s_i, and N the length of the sequence of inputs. We consider only the case of n even.

11.4 THEORETICAL ANALYSIS OF URN MODELS

We can deduce several properties of the class of URN models from their definition, independently of experimental results. We again highlight the fact that the behaviour of the model for any input string w is entirely characterised by the orthogonal matrix which is the product of the orthogonal embeddings $embed\, x_i$ of each of the symbols x_i composing it. Studying the model consists in examining these matrices.

Since the work of Mikolov et al. (2013), vector embeddings have proven to be an extremely successful modelling tool. One of their primary disadvantage in most deep learning systems is that their structure is opaque. The only way of analysing the relations among word vectors is through geometric distance metrics, such as cosine similarity. The unit vectors u and v are deemed similar if $\langle u, v\rangle$ is close to 1.

For URNs we use orthogonal matrix embeddings, which exhibit much richer structure. We apply mathematical analysis to get a clear understanding of this structure, and how it relates to vector embeddings.

11.4.1 Compositionality

The product of orthogonal matrices is another orthogonal matrix. Consequently, the results of Section 11.2.4 apply to the URN, but to a stronger degree: the carrier set of the sequence algebra can be chosen to that of *orthogonal* matrices, not general matrices. This ensures that the combination of matrices that URNs perform in processing input is strictly compositional at each point on the sequence.

11.4.2 Unit vectors

An orthogonal operator preserves the lengths of vectors that it acts upon. Because composition preserves this property, we can ensure that every phrase embedding is orthogonal by ensuring that $embed(x)$ is orthogonal for every symbol x.

We limit ourselves to state vectors h_i of norm 1 from now on. In all our experiments, we take h_0 to be the vector $[1, 0, \dots]$ without loss of generality.

11.4.3 Long-distance memory

Cosine similarity is a proximity metric that is commonly used to measure similarity between vectors. A crucial property of orthogonal operators is that they preserve cosine similarities. This is a consequence of the following theorem.

Theorem 11.3 (Orthogonal matrices preserve inner products)**.** *If Q is orthogonal, then for every h, s, $\langle Qh, Qs\rangle = \langle h, s\rangle$.*

Proof. $\langle Qh, Qs\rangle = (Qh)^T Qs = h^T Q^T Qs = h^T s = \langle h, s\rangle$ □

This property is one of the primary reasons that we are proposing URNs as a model of deep learning for NLP. It entails that no information is lost through time

steps. This formal analysis suggests that an URN will be good at tasks which require long-term memory. We report the results of our experimental work in the latter section of the chapter. They strongly confirm this conjecture.

11.4.4 Distance and Similarity

When working with vectors of unit length only, the cosine similarity is equal to the (euclidean) inner product $\langle u, v\rangle = \sum_i u_i v_i$. It is equivalent to working with euclidean distance squared, because $\|u - v\|^2 = 2(1 - \langle u, v\rangle)$.

Notions of vector similarity and distance can be naturally extended to orthogonal matrices, *via* their effect on the state vector. The Frobenius inner product $\langle P, Q\rangle = \Sigma_{ij} P_{ij} Q_{ij}$ extends cosine similarity, and the Frobenius norm $\|A\|^2 = \sum_{ij} A_{ij}^2$ extends euclidean norm. They connect in a way that is similar to their relationship for unit vectors: $\|P - Q\|^2 = 2(n - \langle P, Q\rangle)$.

Lemma 11.4. *For any two orthogonal matrices P and Q of dimension n, $\|P - Q\|^2 = 2(n - \langle P, Q\rangle)$.*

Proof.

$$\begin{aligned}\|P - Q\|^2 &= \langle P - Q, P - Q\rangle \\ &= \langle P, P\rangle - \langle P, Q\rangle - \langle P, Q\rangle + \langle Q, Q\rangle \\ &= n - 2\langle P, Q\rangle + n\end{aligned}$$

□

Why is the Frobenius norm a natural extension of cosine similarity for vectors? It is not due merely to the similarity of their respective formulas. The relation is deeper. Crucially, the Frobenius inner product (and its associated norm) measures the average behaviour of matrices on state vectors. More precisely, the following holds: $\mathbb{E}_s[\langle Ps, Qs\rangle] = \frac{1}{n}\langle P, Q\rangle$, and $\mathbb{E}_s[\|Ps - Qs\|^2] = \frac{1}{n}\|P - Q\|^2$.

Theorem 11.5. *For every orthogonal matrix Q of dimension n and a random unit vector s, $\mathbb{E}_s[\langle Qs, s\rangle] = \frac{1}{n}\mathsf{trace}(Q)$.*

Proof. By the spectral theorem, Q admits a diagonalisation of the form $Q = U^*\Lambda U$, with U unitary. Let λ_i be the (diagonal) elements of Λ and let $x = Us$. Remark that because U is unitary, $\|x\| = \|s\| = 1$. Thus $\sum_i |x_i|^2 = 1$.[7] Obviously, $\mathbb{E}\left[\sum_i |x_i|^2 = 1\right]$. By assumption, all dimensions of x have the same distribution (applying Q to s does not change this fact, because multiplying by it conserves densities), and thus

[7] Here, even if Q is real, U, λ_i and thus x_i are complex.

$\mathbb{E}[|x_i|^2] = \frac{1}{n}$. We can now compute:

$$\begin{aligned}
\mathbb{E}_s[\langle Qs, s\rangle] &= \mathbb{E}_s[s^T Qs] \\
&= \mathbb{E}_s[s^T U^* \Lambda U s] \\
&= \mathbb{E}_x[x^* \Lambda x] \\
&= \mathbb{E}_x\left[\sum_i |x_i|^2 \lambda_i\right] && \text{by } \Lambda \text{ diagonal} \\
&= \sum_i \lambda_i \mathbb{E}_x[|x_i|^2] && \text{by linearity of expectation} \\
&= \frac{1}{n}\sum_i \lambda_i \\
&= \frac{1}{n}\mathsf{trace}(\Lambda) && \text{because trace is sum of eigenvalues} \\
&= \frac{1}{n}\mathsf{trace}(\Lambda U U^*) \\
&= \frac{1}{n}\mathsf{trace}(U^* \Lambda U) && \text{by trace cyclic property}
\end{aligned}$$

□

Corollary 11.6. *For any two orthogonal matrices P and Q of dimension n, and a random unit vector s, $\mathbb{E}_s[\langle Ps, Qs\rangle] = \frac{1}{n}\langle P, Q\rangle$.*

Proof. Remark first that $\langle Ps, Qs\rangle = s^T P^T Qs = \langle Q^T P, s\rangle$. Combining this equation with Theorem 11.5 and Lemma 11.4 yields the expected result. □

Theorem 11.7. *For every orthogonal matrices P and Q of dimension n and a random unit vector s, $\mathbb{E}_s[\|Ps - Qs\|^2] = \frac{1}{n}\|P - Q\|^2$.*

Proof. A direct consequence of Corollary 11.6 and Lemma 11.4. □

In sum, as fallback, one can analyse unitary embeddings using the methods developed for plain vector embeddings. Doing so is theoretically sound, even though it would not exploit the richer structure of matrices.

11.4.5 Geometric interpretation of embeddings

The proof of Theorem 11.2 tells us that, in general, every orthogonal embedding Q can be written in the form $U^* e^{i\Theta} U$, with Θ real and diagonal. Remembering that such a Q is real, if $2 \times n$ is the dimension of the hidden state vectors, then Q is interpretable as the combination of n rotations around an n orthogonal hyperplane, with angles given by θ_j. For a real matrix, every such angle will be repeated twice in the matrix Θ. The planes of rotation can be computed by diagonalising Q. Below, we will be comparing embeddings by comparing their rotation planes. Since a plane is given by two vectors, it may be unclear how to actually compare them, since they can themselves be rotated within the plane that they define, without changing the

result. We solve this problem by using a measure of similarity based on the solution to the orthogonal procrustes problem, given below. This measure gives a value of 2 when the planes are equal (up to rotation of the basis vectors), and of 0 when they are orthogonal.

Definition 11.6 (Plane similarity)**.** *Assume two planes in an n-dimensional space, each defined by two orthogonal normal vectors, arranged in an n by* 2 *matrix. The similarity between A and B is defined by* $\mathsf{sim}(A, B) = \max_\Omega \langle B, A\Omega \rangle$ *for* Ω *a 2 by 2 orthogonal matrix.*

By taking the minimum for Ω— any rotation of the 2 base vectors in the plae— we account for equal planes, which might be defined by another basis. $A\Omega$ covers all possible bases of the plane defined by A when varying Ω.

Theorem 11.8. $\mathsf{sim}(A, B)$ *is the sum of singular values of* $B^T A$.

Proof. Let $U\Sigma V^T = B^T A$ be a singular value decomposition of $B^T A$.

$$\begin{aligned}
\max_\Omega \langle B, A\Omega \rangle &= \max_\Omega \mathsf{trace}(B^T A\Omega) \\
&= \max_\Omega \mathsf{trace}(U\Sigma V^T \Omega) \\
&= \max_\Omega \mathsf{trace}(\Sigma V^T \Omega U) \\
&= \mathsf{trace}(\Sigma)
\end{aligned}$$

The last step follows because $V^T \Omega U$ is orthogonal. The fact that Σ is diagonal entails that the maximum trace for the product is achieved when $V^T \Omega U = I$. □

In general, an orthogonal embedding contains a lot of data, and it can be hard to visualise. How can we obtain a simplified, if abstract, picture of it?

11.4.5.1 Signature of Embeddings

First, we consider only the *angles* of the rotations θ_j, not the planes. While the average effect is a useful measure, it is rather crude. The angles of these rotations define how strongly Q affects the state vectors lying in this plane. We refer to such a list of angles as the *signature* of Q, and we denote it as $\mathsf{sig}(Q)$. We represent any such angle graphically as a dial, with small angles pointing up ⏧, and large angles (close to π) pointing down ⏨.

11.4.6 Average Effect

The squared distance to the identity matrix is a useful metric for unitary embeddings, $\|Q - I\|^2$. Theorem 11.5 entails that this metric is the average squared distance between s and Qs. This is the average effect that Q has, relative to the task for which the URN is trained. Note that this sort of metric is unavailable for the opaque vector embeddings of non-linear deep neural networks. For those systems the norm of a vector embedding is not directly interpretable as a measure of its effect. In an LSTM,

for example, vector embeddings first undergo a linear transformation *followed by activation functions*, before producing an output state, in several separate stages.

By Lemma 11.4, the average effect is also equal to $2(n - \langle Q, I\rangle)$. But $\langle Q, I\rangle$ is equal to the trace of Q, which, in turn, is equal to the sum of its eigenvalues ($\sum_j e^{i\theta_j}$). Therefore, the average effect is a measure of how large the angles in the signature are.

11.4.7 Truncation

Another way to simplify the representation is to limit ourselves to matrices $S(x)$ which have non-zero entries only on the first k rows (and consequently k columns). This restricts the total size of the embedding to $(n-1)+(n-2)+\cdots+(n-k+1)$, due to the symmetric constraint. We refer to this setup as *truncated* embeddings.

As an example, the 3×3 skew-symmetric matrix $\begin{pmatrix} 0 & a & b \\ -a & 0 & c \\ -b & -c & 0 \end{pmatrix}$ is 1-truncated if $c = 0$. This truncation reduces its matrix informational content to the single row (and column) $(a\ b)$.

We use the acronym URN to refer to the general class of unitary-evolution networks, k-TURN to refer to our specific model architecture with k-truncation of embeddings, and Full-URN for our model architecture with no truncation. (The letter "T" in TURN stands for "Truncated".)

A property of k-truncated embeddings is that $S(x)$ has at most rank $2k$. It follows from Theorem 11.2 that the corresponding orthogonal embeddings $Q(x)$ can be expressed as rotations around (at most) k planes. The more an embedding is truncated— the smaller k— the simpler its geometric interpretation becomes.

11.4.8 Projection

Finally, to understand an orthogonal embedding Q, it can be enlightening to project Q onto a m-dimensional subspace of the n-dimensional ambient space. We do this by projecting each vector on to the m-dimensional subspace. We will consider the projection of some embeddings onto the space defined by the prediction layer of an URN.

11.5 EXPERIMENTS

We apply the LSTM and the URN to several tasks. In all cases, we employ a standard training regime. We use an Adam optimiser (Kingma and Ba, 2014), with no further adjustment. The learning rate is 0.001, and the batch size is 512. Our implementation is done within the TensorFlow (Abadi et al., 2016) framework (version 2.2), including its treatment of matrix exponential. We apply a dropout function on both inputs of the cell function f at every time step, according to a Bernoulli distribution rate of ρ. This causes some entries of s_i or $Q(x_i)$ for the URN to be zeroed out. No dropout is applied for the LSTM on the linear recurrent vector (c_t).

11.5.1 Cross-Serial dependencies

Shieber (1985) demonstrated the non-context-free nature of interleaved verb-object relations in Dutch and Swiss German. One of Shieber's examples of an embedded verb-object crossing dependency in Swiss German is given in example (A) below.

(A) Jan sät das mer **d'chind** em ***Hans*** es huus **lönd** ***hälfe*** aastriiche
Jan said that we the children-ACC Hans-DAT the house-ACC let help paint
Jan said that we let the children help Hans paint the house.

Similar patterns have been observed in other languages. They can be expressed by indexed grammars (Aho, 1968; Pulman and Ritchie, 1985), as well as a variety of other Mildly Context-Sensitive grammar formalisms (Joshi et al., 1990; Stabler, 2004).

Shieber (1985) identified that cross-serial dependency patterns of case marked nouns and their corresponding verbs can be iterated in this construction. The above pattern can be abstracted as a set of $a^m b^n c^m d^n$ structures, which together form a Mildly Context-Sensitive language.

Formally, we consider the family of languages $\mathcal{C}_k = \{a^m b^n c^m d^n \mid m + n < k\}$. Note that if $k < l$, then $\mathcal{C}_k \subset \mathcal{C}_l$. The training set consists of 51,200 strings picked uniformly from $\mathcal{C}_8$. The test set contains 5,120 strings picked uniformly from $\mathcal{C}_{10}$.

The RNNs are trained as generative language models. Assuming a sample string $w \in \mathcal{C}_{10}$, RNNs are trained to predict the symbol w_{i+1} given w_0 to w_i. Special start and stop symbols are added to the input strings, as is standard.

This is to be contrasted with the testing procedure. At test time, given the prefix $w_0 \ldots w_i$, a prediction of symbol w_{i+1} is deemed correct if it is a possible continuation for $w_0 \ldots w_i$; that is, if $w_0 \ldots w_{i+1}$ is a prefix of some string in $\mathcal{C}_{10}$. A set of predictions for a full string $x_0 \ldots x_k$ is classified as correct, if all predictions are correct up to and including the stop symbol. We report error rates for full strings only. This is because when models make a mistake, it is typically for a single symbol near the end of a string.

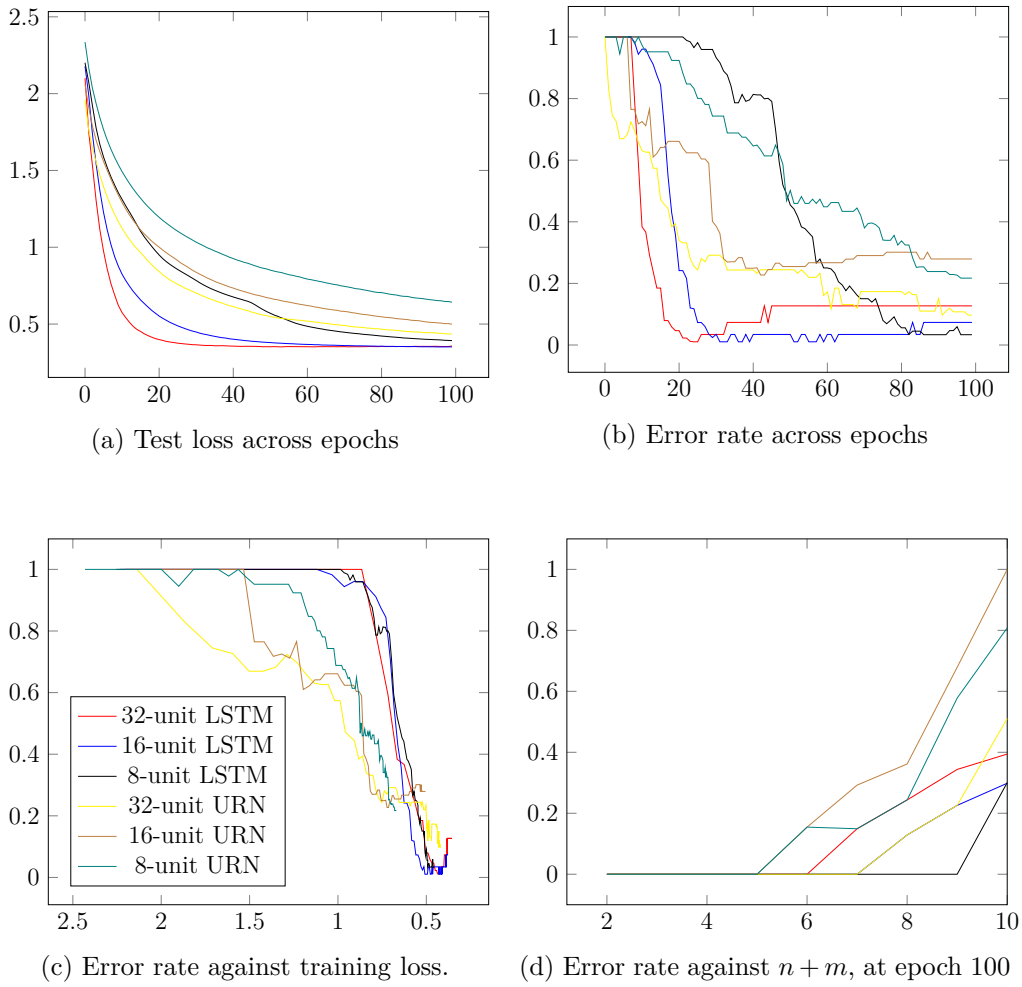
11.5.1.1 Results

Both RNNs struggle to generalise these patterns. They can model the training data well, but produce incorrect patterns in some cases on strings of any greater length than the samples in the training corpus.

We report four sets of results. The first set (Fig. 11.3a) is the cross-entropy loss for the *test set* obtained by each model across training epochs. Low losses indicate that, on a per-character basis, the models reproduce the *exact* strings in the test set. We see that LSTM models generally make better guesses than URNs, across the board.

The second set (Fig. 11.3b) is the error rate over number of epochs, for each tested model. The best models can achieve less then ten percent error rate on average on the test set. However, the LSTM models with larger number of units exhibit overfitting.

In the third set we show the error rate for a given training loss in Fig. 11.3c. The corresponding ratio is a measure of a model's capacity for correct predictions of a given quality of approximation of the training set. It can be taken as an indication of



(a) Test loss across epochs

(b) Error rate across epochs

(c) Error rate against training loss.

(d) Error rate against $n+m$, at epoch 100

Figure 11.3: Cross-Serial dependencies results for various models

the model's bias for this task, relatively to generative language modelling. The URN models tend to achieve lower error rates for this task, even though they do less well from a generative language modelling perspective. For instance, the 32-unit URN is able to obtain an error rate below 0.4 with a training loss as high as 1. In general, the URNs provide a smoother decrease in error rate as they learn the language. In contrast, the LSTM models exhibit a sharp drop in error rate around 0.7 training loss.

Finally, we show the error rate broken down by length of pattern (reported as $n+m$). We see here that the LSTMs tend to do better overall than the URNs for lengths unseen in the training data. However, the LSTMs do worse when the number of units increases, while the URNs do better as that number increases, thanks to a lack of overfitting.

11.5.2 Generalised Dyck Languages

In the next experiment, we evaluate the long-distance modelling capabilities of an RNN for a context-free language. As before, we abstract away from the noise of natural language, by constructing synthetic data. Following Bernardy (2018), we use a (generalised) Dyck language. It is composed solely of matching parenthesis pairs. So the strings "`{([])}<>`" and "`{()[<>]}`" are part of the language, while "`<)`" is not.

Formally, we use the language $\mathcal{D}$ defined as the set of strings generated by the following context-free rules: $E ::= \varepsilon; E ::= EE; E ::= oEc$, where (o, c) stands for a pair of matching parenthesis pairs. In all of our tests, we use 5 types of pairs (corresponding, for example, to the pairs `()` , `[]`, `{}`, `<>` and `‘’`.)

The training phase treats the URN as a generative language model, applying a cross-entropy loss function at each position in the string. At test time, we evaluate the model's ability to predict the right kind of closing parenthesis at each point (this is the equivalent of predicting the number of a verb). We ignore predictions regarding opening parentheses, because they are always acceptable for the language. To generate a string with N matching pairs, we perform a random walk between opposite corners of a square grid of width and height N, such that one is not allowed to cross the diagonal. When not restricted by the boundary, a step can be taken either along the x or y axis with equal probability. A step along the x axis corresponds to opening a parenthesis, and one along the y axis involves closing one. The type of parenthesis pair is chosen randomly and uniformly.

In this task, we use strings with a length of exactly 20 characters. We train on 102,400 randomly generated strings, with maximum depth 3, and test it on 5120 random strings of maximum depth 10. Training is performed with a learning rate of 0.01, and a dropout rate of $\rho = 0.05$, for 100 epochs.

The aim of the task is to predict the correct type of *closing* parenthesis at every point in a string. It should be noted that this experiment is an idealised version of the agreement task proposed by Linzen, Dupoux, et al. (2016).[8] The opening parenthesis plays the role of a word (say a noun) which governs a feature of a subsequent word (like the number of a verb), represented by the closing parenthesis. Matching of parentheses corresponds to agreement. Linzen, Dupoux, et al. (2016) point out that sustaining accuracy over long distances requires that the model have knowledge of hierarchical syntactic structure. If an RNN captures the long-distance dependencies involved in agreement relations, it cannot rely solely on the nearby governing symbols. The measure of distance used by Linzen, Dupoux, et al. (2016) is the number of *attractors*. For our experiment, an attractor is defined as an opening parenthesis occurring within a matching pair, but of the wrong kind. For instance, in "`{()}`", the parenthesis "`(`" is an attractor. We complicate the matching task by varying the nesting depth between training and test phases. The *depth* of the string is the

[8] We report our experiment for the agreement task for natural language in the next section. Baroni (Chapter 1 of this volume) discusses the deep learning and linguistic aspects of this task in some detail.

maximum nesting level reached within it. For instance "`[{}]`" has depth 2, while "`{([()]<>)}`" has depth 4.

11.5.2.1 Results

We report four sets of results. The first set (Fig. 11.4a) is the cross-entropy loss for the *test set* achieved by each model across training epochs. Low losses indicate that, on a per-character basis, the models reproduce the *exact* strings in the test set. These losses cannot drop to zero because it is always valid to predict an opening parenthesis. As in the cross-serial task, we observe that LSTM models make better guesses than URN, at least for a similar number of units. However, we see that the training of the URN is uniformly monotonous, while the LSTM can sometimes become worse for a few epochs before converging. In fact, for 8 units, the LSTM exhibits overfitting: in this case the test loss increases slowly after epoch 30.

To analyse the performance of each model on the task, we break down the error rate by number of attractors (Fig. 11.4b). The URN models are weakest for a low number of attractors, and they achieve near perfect accuracy for a large number of attractors. According to Linzen, Dupoux, et al. (2016), this suggests that the models are highly successful in learning hierarchical structure. A high numbers of attractors corresponds to outer pairs, while a low number of attractors corresponds to inner pairs. In sum, in inner positions the URN suffers from some confusion. This confusion decreases as the number of units increases.

The LSTM exhibits good accuracy for adjacent pairs, with zero attractors, such as `[]`. It does worst on pairs with 3 to 4 attractors. For larger number of units, the accuracy then increases again, all the way to the longest parenthesis pair. So the LSTM is good at making a prediction which depends only on the previous symbol, and it is also good at making a prediction for a pair which encloses the whole string. This indicates that it is fairly limited in its ability to capture hierarchical structures over long distances, even though it does much better than the majority class baseline, which stands at an 80% error rate.

In what follows, we will consider only the *maximum* error rate for any given number of attractors, over varying epochs. That is, we report on the peak value from the previous graph. Using this metric, the URNs perform consistently better than the LSTMs with the same number of units (and a similar number of parameters). They do so consistently across the training period (Fig. 11.4c). However, we note that every model is capable of generalising to deeper nesting levels to some extent, with an accuracy well above a majority class baseline (80% error rate). The URN models beat the majority class baseline within the first epoch, while the LSTMs need a couple of epochs to do so.

Next we report the error rate against training loss (Fig. 11.4d), as we did for the cross-serial dependency task. Again, we do not report the average error rate, but rather the *maximum* error rate for any given number of attractors. Here too, the relationship between error rate and training loss corresponds to the bias of the model for the task at hand, compared to a generative language model task. We observe that the URN models outperform the LSTM models across the board.

	0.33	0.35	1.35
	0.46	1.73	0.2
	1.09	0.2	0.34

Table 11.1: Similarity for each pair of rotation planes, for the embeddings of `(` and `[`. Headers show the rotation effected on the compared planes.

Finally, we consider variants of the URN model. We report accuracy of the URN model using either truncated embeddings, full embeddings, and for a baseline RNN with full embeddings that are not constrained to be orthogonal (i.e. any matrix). Still, no activation function is applied in any variant. In all three cases, the size of orthogonal matrices is 50 by 50. We report accuracy on the task by number of attractors in Fig. 11.5. We see that truncating embeddings does affect performance, but not in a way that qualitatively changes the behaviour of the model. Truncating has an effect similar to using fewer units. The non-orthogonal (arbitrary matrices) model shows steadily decreasing accuracy relative to the number of attractors. We note that even this naive model is not capable of generalising to longer distances. The performance of the full URN is much better overall. This happens despite the fact that the orthogonal system is a special case of the arbitrary linear recurrent network, and so orthogonal embeddings are, in principle, available to the linear RNN without the orthogonal constraint. But it is not able to converge on the preferred solution (even for absolute loss). Our results show that restriction of the model to orthogonal matrices offers a significant benefit in generalisation and tracking power.

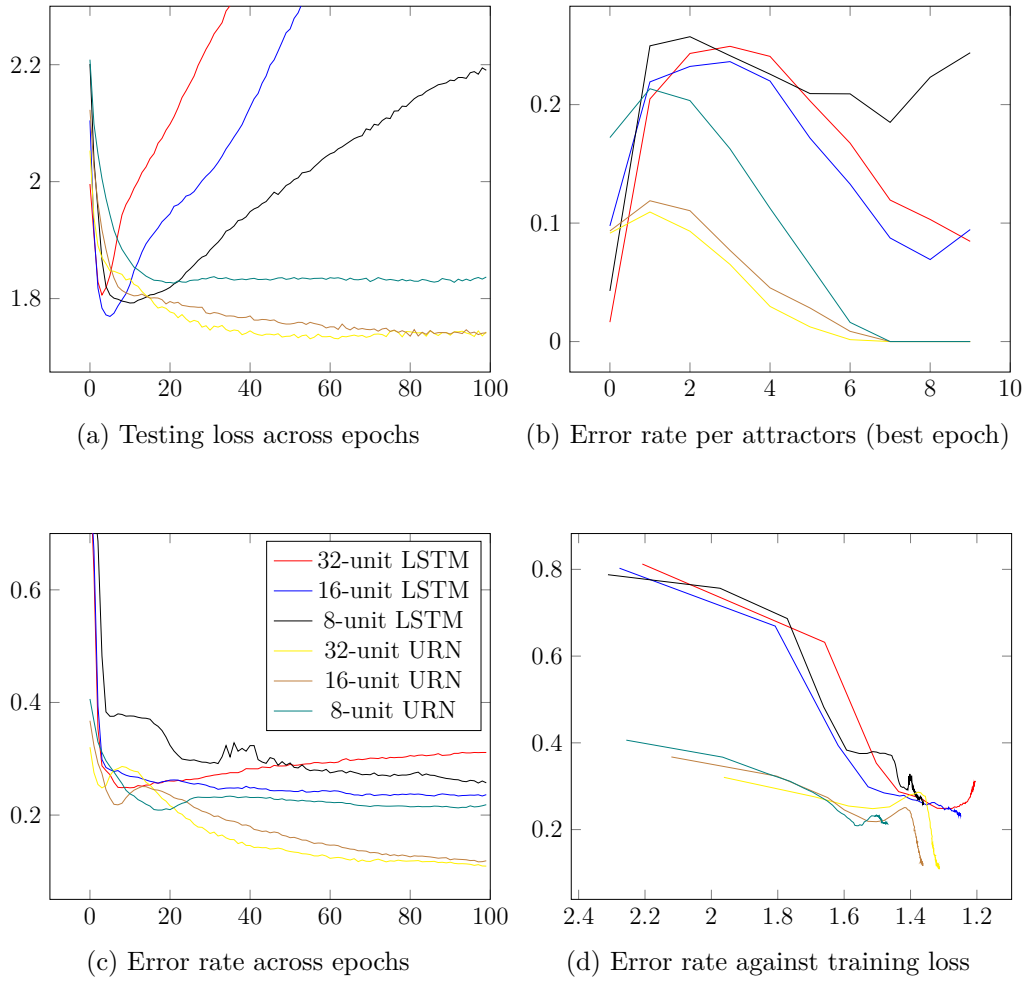
11.5.2.2 Analysis

Let's analyse the orthogonal embeddings further. We consider the 3-truncated embeddings produced in the last variant of the experiment, and we start with the embeddings of individual characters and their signatures (Table 11.2). The average effect, and even the signatures of all embeddings, are strikingly similar. This does not imply that they are *equal*, because they act on different planes. We measure the similarity of planes using Definition 11.6.

We see in Table 11.1 that the planes which undergo rotation by similar angles are far from orthogonal to each other. One pair even exhibits a similarity of 1.73. This corresponds to the fact that the transformations of `(` and `[` manipulate a common subset of coordinates. On the other hand, those planes that undergo rotation by different angles tend to be in a closer to orthogonal relationship.

11.5.2.3 Composition of Matching Parentheses

To further clarify the formal properties of our model let's look at the embeddings of matching pairs, computed as the product of the respective embeddings of the pairs. Such compositions are close to identity (Table 11.2). This observation explains the extraordinarily accurate long-distance performance of the URN on the matching



(a) Testing loss across epochs

(b) Error rate per attractors (best epoch)

(c) Error rate across epochs

(d) Error rate against training loss

Figure 11.4: Results for the Dyck language experiment. When reporting error rates as training progresses, we use the maximum error rate across number of attractors.

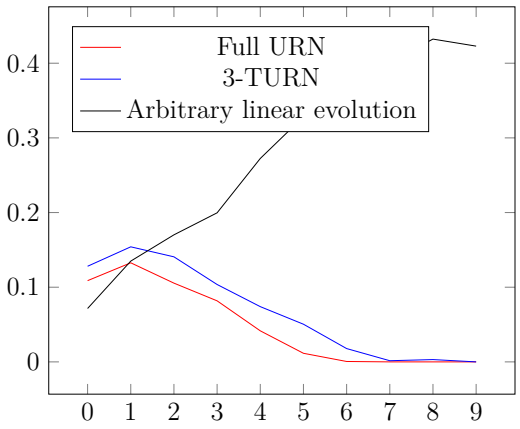


Figure 11.5: Error rate per number of attractors, at epoch 100, with 50 units, for various linear RNN architectures.

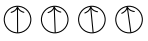
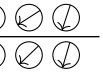
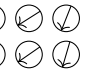
character	average effect	signature	string	average effect	signature
(	14.79		()	0.06	
<	14.34		<>	0.06	
{	13.98		{}	0.07	
[	14.25		[]	0.06	
+	14.20		+-	0.06	
)	14.85				
>	14.42				
}	14.07				
]	14.34				
-	14.26				

Table 11.2: Average effect and signatures of parenthesis embeddings and matching pairs.

task. Because a matching pair has essentially no effect on the state, by the time all parentheses have been closed, the state returns to its original condition. The model experiences the highest level of confusion when it is *inside* a deeply nested structure, and *not* when a deep structure is inserted between the governing opening parenthesis and the prediction conditioned on that parenthesis.

11.5.3 Natural Language Long-Distance Agreement Task

Having seen that the URN performs well on synthetic language applications, we turn to a natural language agreement task proposed by Linzen, Dupoux, et al. (2016). This involves predicting the number of third person verbs in English text, through supervised learning. In the phrase "The **keys** to the cabinet **are** on the table", the RNN is trained to predict the plural "**are**" rather than the singular "**is**". This is the natural equivalent of the bracket matching task discussed in the previous section.

The training data is composed of 1.7 million sentences with a selected subject-verb pair, extracted from Wikipedia. The vocabulary size is 50,000, with out-of-vocabulary tokens replaced by their part-of-speech tags. Training is performed for ten epochs, with a learning rate of 0.01, and a dropout rate of $\rho = 0.05$. We use 90% of the data for training, and 10% for validation and testing. A development subset is not required, since no effort was made to tune hyperparameters. Our first experiment proved sufficient to illustrate our main claims. In any case, a TURN has few hyperparameters to optimise.

Figure 11.6 shows the results for a 50-unit TURN with 3-truncated embeddings for the agreement task, for up to 12 attractors. We see that the TURN performs less well than the LSTM of Linzen, Dupoux, et al. (2016). The accuracy drops off for both models, as the number of attractors increases. Statistical uncertainty increases a lot with the number of attractors, due to decreasing numbers of examples.[9]

[9] In earlier work (Bernardy and Lappin, 2022), we have reported that the URN does better on this task than the LSTM. This was due to an error in handling the training set. The current results should be understood as a correction of our earlier reported work.

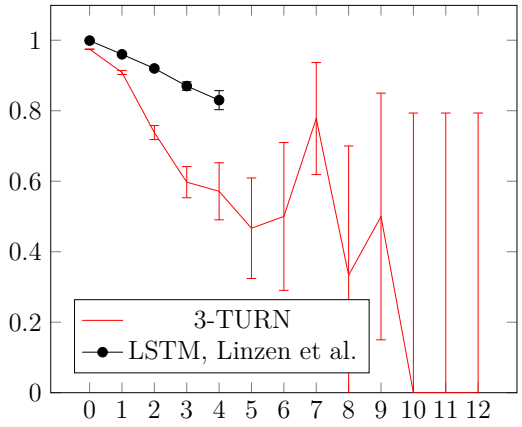


Figure 11.6: Accuracy per number of attractors for the verb number agreement task. Linzen, Dupoux, et al. (2016) do not report performance of their LSTM past 4 attractors. Error bars represent binomial 95% confidence intervals.

11.5.3.1 Analysis

Despite the unimpressive accuracy of the model, we can still use our theoretical tools to understand how the learned truncated unitary embeddings work. We first measure the average effect for the embeddings of common words in the dataset (Table 11.3), and other selected words and phrases (Table 11.5) obtained by composition. The table of effects for these words and phrases (ordered from smallest to largest effect) confirms our view concerning the measurement, along a particular plane, of the relations among the unitary embeddings applied to the agreement task. Tokens which are relevant to the task (e.g. "is", "which") have a larger effect than those which are not (e.g. the dot, "not").

Unfortunately, the average effect does not yield a sharp distinction among words for the agreement task. It also does not separate some classes of words which have a direct effect on the prediction (nouns, relative pronouns) from words which have only have an indirect effect (object pronouns). We can further refine the analysis by the projection of the word embedding $Q(x)$ onto the subspace generated by the projection layer.

Because we have only two possible predictions for this task, this space is 2-dimensional. The projection is a 2×2 matrix indicating how the word embedding acts on the features determining the prediction of the number of the coming verb.[10] The top left entry in the matrix corresponds to the action on the most relevant feature. Lower values for this entry correlate with higher direct influence of the associated lexical item on the feature. This matrix is *not* in general orthogonal. But the projection of many embeddings is close to the identity matrix. This proximity indicates that the corresponding word has little direct influence on the prediction, although it could have an effect when combined with some other words.

Conversely, words which have a direct effect on the prediction receive a matrix

[10] Essentially, we take the average for all other dimensions.

word	average effect	projection	word	average effect	projection
.	0.22	$\begin{pmatrix} 1.00 & -0.01 \\ 0.01 & 1.00 \end{pmatrix}$	from	4.09	$\begin{pmatrix} 0.93 & -0.05 \\ 0.06 & 0.90 \end{pmatrix}$
the	1.44	$\begin{pmatrix} 0.94 & 0.02 \\ -0.03 & 0.99 \end{pmatrix}$	i	4.11	$\begin{pmatrix} 0.91 & -0.07 \\ 0.05 & 0.95 \end{pmatrix}$
his	1.47	$\begin{pmatrix} 0.95 & 0.03 \\ -0.06 & 0.98 \end{pmatrix}$	it	4.14	$\begin{pmatrix} 0.95 & 0.00 \\ -0.04 & 0.91 \end{pmatrix}$
its	2.17	$\begin{pmatrix} 0.97 & 0.01 \\ -0.04 & 0.97 \end{pmatrix}$	and	4.18	$\begin{pmatrix} 0.94 & 0.00 \\ -0.03 & 0.94 \end{pmatrix}$
also	2.27	$\begin{pmatrix} 0.99 & 0.01 \\ -0.03 & 0.95 \end{pmatrix}$	on	4.33	$\begin{pmatrix} 0.94 & -0.08 \\ 0.06 & 0.95 \end{pmatrix}$
their	2.54	$\begin{pmatrix} 0.91 & 0.06 \\ -0.08 & 0.97 \end{pmatrix}$	with	4.36	$\begin{pmatrix} 0.91 & -0.11 \\ 0.05 & 0.95 \end{pmatrix}$
not	2.73	$\begin{pmatrix} 0.96 & 0.04 \\ -0.08 & 0.95 \end{pmatrix}$	has	4.41	$\begin{pmatrix} 0.91 & -0.03 \\ -0.02 & 0.87 \end{pmatrix}$
been	2.82	$\begin{pmatrix} 0.95 & 0.07 \\ -0.05 & 0.94 \end{pmatrix}$	for	4.62	$\begin{pmatrix} 0.93 & -0.06 \\ 0.11 & 0.88 \end{pmatrix}$
at	3.40	$\begin{pmatrix} 0.95 & -0.05 \\ -0.02 & 0.95 \end{pmatrix}$	in	4.62	$\begin{pmatrix} 0.94 & -0.07 \\ -0.02 & 0.93 \end{pmatrix}$
or	3.46	$\begin{pmatrix} 0.97 & -0.05 \\ 0.02 & 0.95 \end{pmatrix}$	have	4.62	$\begin{pmatrix} 0.78 & 0.16 \\ -0.05 & 0.92 \end{pmatrix}$
by	3.50	$\begin{pmatrix} 0.94 & -0.06 \\ 0.02 & 0.93 \end{pmatrix}$	who	4.68	$\begin{pmatrix} 0.84 & -0.10 \\ 0.09 & 0.94 \end{pmatrix}$
one	3.54	$\begin{pmatrix} 0.91 & 0.07 \\ -0.04 & 0.96 \end{pmatrix}$	were	4.88	$\begin{pmatrix} 0.62 & -0.01 \\ 0.08 & 0.95 \end{pmatrix}$
this	3.62	$\begin{pmatrix} 0.92 & 0.01 \\ -0.01 & 0.96 \end{pmatrix}$	that	5.00	$\begin{pmatrix} 0.85 & 0.12 \\ -0.14 & 0.88 \end{pmatrix}$
be	3.65	$\begin{pmatrix} 0.83 & 0.12 \\ -0.14 & 0.93 \end{pmatrix}$	was	5.55	$\begin{pmatrix} 0.72 & -0.03 \\ -0.04 & 0.92 \end{pmatrix}$
an	3.70	$\begin{pmatrix} 0.88 & 0.04 \\ -0.02 & 0.96 \end{pmatrix}$	(	5.68	$\begin{pmatrix} 0.86 & 0.03 \\ -0.13 & 0.91 \end{pmatrix}$
as	3.76	$\begin{pmatrix} 0.93 & 0.03 \\ -0.07 & 0.94 \end{pmatrix}$	)	5.74	$\begin{pmatrix} 0.90 & 0.04 \\ -0.10 & 0.92 \end{pmatrix}$
he	3.95	$\begin{pmatrix} 0.86 & -0.09 \\ 0.09 & 0.94 \end{pmatrix}$	are	6.25	$\begin{pmatrix} 0.53 & 0.02 \\ -0.06 & 0.91 \end{pmatrix}$
had	3.95	$\begin{pmatrix} 0.85 & 0.02 \\ -0.08 & 0.95 \end{pmatrix}$	but	6.27	$\begin{pmatrix} 0.73 & 0.09 \\ -0.22 & 0.89 \end{pmatrix}$
to	3.96	$\begin{pmatrix} 0.97 & -0.05 \\ 0.03 & 0.95 \end{pmatrix}$	is	6.38	$\begin{pmatrix} 0.68 & 0.02 \\ -0.08 & 0.92 \end{pmatrix}$
a	4.06	$\begin{pmatrix} 0.89 & 0.00 \\ -0.04 & 0.95 \end{pmatrix}$	which	7.75	$\begin{pmatrix} 0.71 & 0.01 \\ -0.19 & 0.85 \end{pmatrix}$
of	4.09	$\begin{pmatrix} 0.91 & -0.08 \\ 0.12 & 0.88 \end{pmatrix}$	,	8.35	$\begin{pmatrix} 0.58 & -0.09 \\ -0.26 & 0.77 \end{pmatrix}$

Table 11.3: Table of phrase effects for agreement experiment for the most frequent tokens in the corpus, ordered by average effect, from least to greatest. The *projection* column shows the projection of the rotation on the prediction space.

	article	year	area	world	family	articles	years	areas	worlds	families
article	0.00	7.04	6.51	6.89	5.82	9.26	9.84	10.01	10.87	9.39
year	7.04	0.00	7.62	6.30	5.38	8.22	9.06	9.75	10.14	8.64
area	6.51	7.62	0.00	6.42	6.34	9.57	9.70	10.39	11.63	10.39
world	6.89	6.30	6.42	0.00	5.17	7.32	8.82	9.17	9.13	7.83
family	5.82	5.38	6.34	5.17	0.00	7.71	7.72	8.78	9.49	8.82
articles	9.26	8.22	9.57	7.32	7.71	0.00	5.11	4.79	4.28	4.57
years	9.84	9.06	9.70	8.82	7.72	5.11	0.00	6.42	6.61	7.14
areas	10.01	9.75	10.39	9.17	8.78	4.79	6.42	0.00	5.93	6.09
worlds	10.87	10.14	11.63	9.13	9.49	4.28	6.61	5.93	0.00	7.79
families	9.39	8.64	10.39	7.83	8.82	4.57	7.14	6.09	7.79	0.00

Table 11.4: Distances between embeddings of most frequent nouns and their plural variants. Words which can be both nouns and verbs were excluded.

the key	4.34		$\begin{pmatrix} 0.94 & 0.02 \\ -0.03 & 0.96 \end{pmatrix}$
the keys	5.32		$\begin{pmatrix} 0.46 & 0.10 \\ -0.16 & 0.94 \end{pmatrix}$
the keys to the cabinet	11.85		$\begin{pmatrix} 0.46 & 0.13 \\ -0.08 & 0.91 \end{pmatrix}$

Table 11.5: Effect, signature, and projection of the embedding of selected phrases

which is far from identity. The pair of words "have" and "in" is particularly noteworthy. They have the same average effect, but different direct effects on the prediction, with "have" showing a much larger direct effect, as expected. The phrase "the keys to the cabinet" has a much larger effect on the agreement task than the phrase "the keys", but they both have a similar influence on the prediction of the next word. Note also that the phrase "the key" and its plural counterpart have a similar average effect, and similar signatures. But the projection on the prediction space shows much less effect for the singular. This can be explained by the fact that number prediction is biased towards the singular in the absence of context. For instance, a lone verb tends to be singular.

We also compute the Frobenius distance between pairs of orthogonal embeddings of the most frequent nouns, with both singular and plural inflections (Table 11.4). As our account predicts, nouns with the same number inflection tend to be grouped together (with a distance of 7.5 or less between them), while nouns with distinct numbers are further apart (with a distance of 7.5 or more).

11.6 RELATED WORK

It has frequently been observed that DNNs are complex and opaque in the way in which they operate. It is often unclear how they arrive at their results, or why they identify the patterns that they extract from training data. This has given rise to a concerted effort to render deep learning systems explainable (Linzen, Chrupała, and Alishahi, 2018; Linzen, Chrupała, Belinkov, et al., 2019). This problem has become more acute with the rapid development of very large pre-trained transformer models

(Vaswani et al., 2017), like BERT (Devlin et al., 2018), GPT2 (Solaiman et al., 2019), GPT3 (Brown et al., 2020), and XLNet (Yang et al., 2019).

URNs avoid this difficulty through being strictly compositional by design. If they prove robust for a wide variety of NLP tasks, they will go some way to solving the problem of transparency in deep learning.

11.6.1 Learning Agreement

The capacity of Recurrent Neural Networks (RNNs), particularly LSTMs, to identify context-free long distance dependencies has been widely discussed in the NLP and cognitive science literature (Bernardy, 2018; Bernardy and Lappin, 2017; Elman, 1990; Gulordava et al., 2018; Lappin, 2021; Linzen, Dupoux, et al., 2016; Sennhauser and Berwick, 2018). These discussions have considered dependency patterns in both artificial systems, particularly Dyck languages, and in natural languages, with subject-verb agreement providing a paradigm case.

Bernardy (2018) tested the ability of LSTMs to predict closing parenthesis types in a Dyck language. The results are qualitatively similar to those obtained for natural language agreement. In both cases the LSTM makes the least successful predictions for a moderate number of attractors. However he reports worse overall results than we achieved in our experiments, despite using an LSTM with more units. We attribute the improved performance of our LSTM to better implementation of the model. A more optimal application of dropouts is a likely factor in getting better results than Bernardy's LSTM (2018).

While LSTMs (and GRUs) exhibit a certain capacity to generalise to deeper nesting, their accuracy declines in relation to nesting depth. This is also the case with their handling of natural language agreement. Other experimental work has illustrated this effect (Hewitt et al., 2020; Sennhauser and Berwick, 2018). Similar conclusions hold for generative self-attention architectures (Yu et al., 2019). Significantly, recent work has indicated that non-generative self-attention architectures, in the mode of BERT, perform poorly on this task (Bernardy, Ek, et al., 2021). Their work suggests that sequential processing is required to solve it.

By contrast URNs achieve excellent results for this task, without any decline in relation to either nesting depth, or number of attractors. We have shown that this is because the learned unitary embeddings for matching parentheses are nearly the inverses of each other.

The question of whether generative language models can learn long-distance agreement was proposed by Linzen, Dupoux, et al. (2016). If accuracy is insensitive to the number of attractors, then we know that the model can work on long distances. The results of Linzen, Dupoux, et al. (2016) are inconclusive on this question. Even though the model does better than the majority class baseline for up to four attractors, accuracy declines steadily as the number of attractor increases. This trend is confirmed by Bernardy and Lappin (2017), who ran the same experiment on a larger dataset and explored the space of hyperparameters in detail. It is also confirmed by Gulordava et al. (2018), who analysed languages other than English. Marvin and Linzen (2018) focus on other linguistic phenomena, reaching similar con-

clusions. Lakretz et al. (2021) recently showed that an LSTM may extract bounded nested tree structures, without learning a systematic recursive rule. These results do not hold directly for BERT-style models, because they are not generative language models.[11] For a detailed discussion of these results, see the recent account of Lappin (2021).

Our experiment shows that URNs can surpass state of the art results for this kind of task. This is not surprising. URNs are designed so that they *cannot forget information*. It is expected that they will perform well on tracking long distance relations. The conservation of information in an URN is due to the fact that multiplying by an orthogonal matrix conserves cosine similarities: $\langle Qs_0, Qs_1\rangle = \langle s_0, s_1\rangle$. Therefore any embedding Q, be it of a single word or of a long phrase, maps a change in its input state to an equal change in its output state. Considering all possible states as a distribution, Q conserves the density of states. Hence, contrary to the claims of Sennhauser and Berwick (2018), URNs demonstrate that a class of RNNs can achieve rule-like accuracy in learning syntactic structure.

11.6.2 Cross-serial patterns

Kirov and Frank (2012) study both nested and cross-serial dependencies with a Simple Recurrent Network (SRN). As far as we are aware, our experiment is the first application of both LSTMs and URNs to cross-serial dependency relations. While both achieve encouraging initial accuracy in recognising the cross-serial patterns, URNs offer significant advantages in simplicity and transparency of architecture. They also displays enhanced stability in learning, and power of structural generalisation, relative to loss in training data.

11.6.3 Quantum-Inspired Systems

Unitary matrices are essential elements of quantum mechanics, and quantum computing. There, too, they ensure that the system does not lose information through time.

Coecke et al. (2010) and Grefenstette et al. (2011) propose what they describe as a quantum inspired model of linguistic representation. It computes vector values for sentences in a category theoretic representation of the types of a pregroup grammar (Lambek, 2008). The category theoretic structure in which this grammar is formulated is isomorphic with the one for quantum logic.

A difficulty of this approach is that it requires the input to be already annotated as parsed data. Another problem is the tensors associated with higher-types are very large, making them hard to learn. By contrast, URNs do not require a syntactic type system. Our experiments indicate that, with the right processing model, it is possible to learn syntactic structure and semantic composition from unannotated input.

Compositionality of phrase and sentence matrices is intrinsic to the formal specification of an URN.

The research that we report here extends and modifies some of the leading ideas

[11] Goldberg (2019) and Lau et al. (2020) suggest approaches for using them as generative LMs.

in the foundational work of Coecke et al. (2010) and Grefenstette et al. (2011). They provided a system for handling computational semantics compositionally with structures that map onto the matrices of quantum circuits. We offer a model for learning syntactic structure, and for processing in general, that is strictly compositional. It uses some of the same core methods as the earlier work. In future research we will attempt to apply our model to NLP tasks involving semantic interpretation.

11.6.4 Tensor Recurrent Neural Networks

Sutskever et al. (2011) describe what they call a "tensor recurrent neural network" in which the transition matrix is determined by each input symbol. This design appears to be similar to URNs. However, unlike URNs, they use non-linear activation functions, and so they inherit the complications that these functions carry.

11.6.5 Unitary-Evolution Recurrent Networks

Arjovsky et al. (2016) proposed Unitary-Evolution recurrent networks to solve the problem of exploding and vanishing gradients, caused by the presence of non-linear activation functions. Despite this, Arjovsky et al. (2016) suggest the application of ReLU activation between time-steps, unlike URNs. Moreover, we are primarily concerned with the structure of the underlying unitary embeddings. The connection between the two lines work is that, exploding/vanishing gradients prevent an RNN from tracking long-term dependencies. URNs eliminate this problem.

Arjovsky et al. (2016)'s embeddings are computationally cheaper than ours, because they can be multiplied in linear time. Like us, they do not cover the whole space of unitary matrices. Jing et al. (2017) propose another representation which is computationally less expensive than ours, but which has asymptotically the same number of parameters. A third option is allow back-propagation to update the unitary matrices arbitrarily $n \times n$, and project them onto the unitary space periodically (Kiani et al., 2022; Wisdom et al., 2016).

Because we use a fully general matrix exponential implementation, our model is computationally more expensive than all the other options mentioned above. However, we have found that when experimenting with the unitary matrix encodings of Jing et al. (2017) and Arjovsky et al. (2016), we got much worse results for our experiments. This may be due to the fact that we do not use a ReLU activation function, as they do.

To the best of our knowledge, no previous study of URNs has addressed agreement or other NLP tasks. Rather, they have been applied to data-copying tasks, which are of limited linguistic interest. This includes the work of Vorontsov et al. (2017), even though it is ostensibly concerned with long distance dependencies.

11.7 CONCLUSIONS AND FUTURE WORK

Our experiments have shown the following. First, contrary to previous claims (Sennhauser and Berwick, 2018), RNNs can learn to recognise syntactic structures of the sort that characterise natural language syntax, with good accuracy. Furthermore, this can be

done both with well-known models, such as an LSTM, and mathematically tractable ones, such as an URN. A particularly attractive result, is that these models achieve such accuracy with less than a couple of thousand parameters. For hierarchical structures, the models can generalise to much deeper depths. For cross-serial patterns, the models can approximate the training data well, but do not generalise well to longer patterns.

Second, we have shown the potential of URNs as devices for tracking and predicting complex dependency relations, over long strings, in a fully compositional way. The fact that URNs achieve good precision in predicting both deeply nested and cross-serial dependencies (up to certain length) suggests that they are able to recognise complex syntactic structures of the sort that are challenging for other neural networks. Furthermore, our experiments indicate that URNs are biased towards predicting the patterns found in context-free and mildly context-sensitive languages, even when trained as generative language models.

Third the fact that URNs satisfy strict compositionality offers an important advance in the search for explainable AI systems in deep learning models. Unlike LSTMs, they are compositional by design. They resolve the question of how to generate the composite values of input arguments in a principled and straightforward way. URNs learn *orthogonal embeddings*, which can be combined to provide representations for any phrase. We have demonstrated that they can be analysed using standard tools from linear algebra.

Fourth, the refined distance, effect, and relatedness metrics that unitary embeddings afford, open up the possibility of more interesting procedures for identifying natural syntactic and semantic word classes. These can be textured and dynamic, rather than static. They can focus on specific dimensions of meaning and structure, and they can be driven by particular NLP tasks. They are not blackbox processing devices that require indirect methods of analysis and assessment, as is the case with most other deep neural systems. Compositionality is realised by eschewing non-linear activation functions, which pervade other architectures, such as LSTMs. The presence of activation functions entail that the combination of two cells cannot be expressed as a single cell.

The move to powerful bidirectional transformers, like BERT, has produced enhanced performance in a variety of NLP and other AI tasks. This has been achieved at the expense of formal grounding and computational transparency. It is even less obvious why such models perform as well as they do on some tasks, and poorly on others, than is the case for LSTMs. By contrast, URNs offer simple, light weight deep neural networks whose operation is fully open to inspection and understanding at each point in the processing regime. They achieve encouraging accuracy in capturing complex hierarchical syntactic structures for both artificial and natural data.

Finally, we observe that the unitary matrices through which URNs compute output values from input arguments are identical to the gates of quantum logic. This suggests the intriguing possibility of implementing these models as quantum circuits. At some point in the future, this may facilitate training these models on large amounts of data, and efficiently generating results for tasks that are currently beyond the resources of conventional computational systems. Of course, quantum systems are still

in their infancy, and thus caution is needed here when making claims of efficiency on behalf of quantum computing applied to machine learning.

11.8 ACKNOWLEDGEMENTS

The research reported in this paper was supported by grant 2014-39 from the Swedish Research Council, which funds the Centre for Linguistic Theory and Studies in Probability (CLASP) in the Department of Philosophy, Linguistics, and Theory of Science at the University of Gothenburg. We presented the main ideas of this paper to the CLASP Seminar, in December 2021, and to the Cognitive Science Seminar of the School of Electronic Engineering and Computer Science, Queen Mary University of London, in February 2022. We are grateful to the audiences of these two events for useful discussion and feedback. We thank Stephen Clark for a careful reading of an earlier draft of this chapter. His helpful comments resulted in numerous improvements. We bear sole responsibility for any errors that may remain in the current version.

Bibliography

Abadi, Martín, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2016). "TensorFlow: A System for Large-Scale Machine Learning." In: *OSDI*. Vol. 16, pp. 265–283.

Aho, Alfred V. (1968). "Indexed Grammars–An Extension of Context-Free Grammars". In: 15.4, pp. 647–671.

Arjovsky, Martin, Amar Shah, and Yoshua Bengio (2016). "Unitary Evolution Recurrent Neural Networks". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, pp. 1120–1128.

Barker, Chris and Chung-chieh Shan (2004). "Continuations in natural language". In: *in Proceedings of the fourth ACM SIGPLAN workshop on continuations, Hayo Thielecke*, pp. 55–64.

Baroni, Marco (2022). "On the proper role of linguistically-oriented deep net analysis in linguistic theorizing". In: *Algebraic Structures In Natural Language*. Ed. by Shalom Lappin and Jean-Philippe Bernardy. Taylor and Francis.

Bernardy, Jean-Philippe (2018). "Can RNNs Learn Nested Recursion?" In: *Linguistic Issues in Language Technology* 16 (1).

Bernardy, Jean-Philippe, Rasmus Blanck, Stergios Chatzikyriakidis, and Shalom Lappin (Aug. 2018). "A Compositional Bayesian Semantics for Natural Language". In: *Proceedings of the First International Workshop on Language Cognition and Computational Models*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, pp. 1–10. URL: https://www.aclweb.org/anthology/W18-4101 (visited on 03/01/2021).

Bernardy, Jean-Philippe and Koen Claessen (2013). "Efficient Divide-and-Conquer Parsing of Practical Context-Free Languages". In: *Proceedings of the 18th ACM SIGPLAN international conference on Functional Programming*, pp. 111–122.

— (2015). "Efficient Parallel and Incremental Parsing of Practical Context-Free Languages". In: *Journal of Functional Programming* 25. ISSN: 1469-7653. DOI: 10.1017/S0956796815000131.

Bernardy, Jean-Philippe, Adam Ek, and Vladislav Maraev (2021). "Can the Transformer Learn Nested Recursion with Symbol Masking?" In: *Findings of the ACL 2021*.

Bernardy, Jean-Philippe and Shalom Lappin (2017). "Using Deep Neural Networks to Learn Syntactic Agreement". In: *Linguistic Issues In Language Technology* 15.2, p. 15. ISSN: 1945-3604.

— (2022). "A Neural Model for Compositional Word Embeddings and Sentence Processing". In: *Proceedings of The Workshop on Cognitive Modeling and Computational Linguistics*. Association for Computational Linguistics.

Brown, T., B. Mann, Nick Ryder, Melanie Subbiah, J. Kaplan, P. Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, G. Krüger, Tom Henighan, R. Child, Aditya Ramesh, D. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, E. Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, J. Clark, Christopher Berner, Sam McCandlish, A. Radford, Ilya Sutskever, and Dario Amodei (2020). "Language Models are Few-Shot Learners". In: *ArXiv* abs/2005.14165.

Coecke, Bob, Mehrnoosh Sadrzadeh, and Stephen Clark (2010). "Mathematical Foundations for a Compositional Distributional Model of Meaning". In: *Lambek Festschrift, Linguistic Analysis* 36.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2018). "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805*.

Elman, Jeffrey L. (1990). "Finding structure in time". In: *Cognitive Science* 14.2, pp. 179–211.

Emerson, Guy and Ann Copestake (2016). "Functional Distributional Semantics". In: *Proceedings of the 1st Workshop on Representation Learning for NLP, Rep4NLP@ACL 2016, Berlin, Germany, August 11, 2016*. Ed. by Phil Blunsom, Kyunghyun Cho, Shay B. Cohen, Edward Grefenstette, Karl Moritz Hermann, Laura Rimell, Jason Weston, and Scott Wen-tau Yih. Association for Computational Linguistics, pp. 40–52. DOI: 10.18653/v1/W16-1605. URL: https://doi.org/10.18653/v1/W16-1605.

— (2017). "Semantic Composition via Probabilistic Model Theory". In: *IWCS 2017 - 12th International Conference on Computational Semantics - Long papers*. URL: http://aclweb.org/anthology/W17-6806.

Gantmacher, Felix Ruvimovich (1959). *The Theory of Matrices*. AMS Chelsea publishing.

Goldberg, Yoav (2019). "Assessing BERT's Syntactic Abilities". In: *ArXiv* abs/1901.05287.

Grefenstette, Edward, Mehrnoosh Sadrzadeh, Stephen Clark, Bob Coecke, and Stephen Pulman (2011). "Concrete Sentence Spaces for Compositional Distributional Models of Meaning". In: *Proceedings of the Ninth International Conference on Computational Semantics (IWCS 2011)*.

Groote, Philippe de (2001). "Type raising, continuations, and classical logic". In: *Proceedings of the thirteenth Amsterdam Colloquium*, pp. 97–101.

Grove, Julian and Jean-Philippe Bernardy (2021). "Probabilistic compositional semantics, purely". In: *Proceedings of Logic and Engineering of Natural Language Semantics 18*.

Gulordava, Kristina, Piotr Bojanowski, Edouard Grave, Tal Linzen, and Marco Baroni (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, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 1195–1205.

Hewitt, John, Michael Hahn, Surya Ganguli, Percy Liang, and Christopher D Manning (2020). "RNNs can generate bounded hierarchical languages with optimal memory". In: *arXiv preprint arXiv:2010.07515*. URL: https://arxiv.org/pdf/2010.07515.pdf.

Hochreiter, Sepp and Jürgen Schmidhuber (Nov. 1997). "Long short-term memory". In: *Neural Computation* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

Jing, Li, Yichen Shen, Tena Dubček, John Peurifoi, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić (2017). "Tunable Efficient Unitary Neural Networks (EUNN) and their application to RNN". In: *arXiv*.

Joshi, Aravind K., K. Vijay Shanker, and David Weir (1990). *The Convergence of Mildly Context-Sensitive Grammar Formalisms*. Tech. rep. Philadelphia, PA: Department of Computer and Information Science, University of Pennsylvania.

Kiani, Bobak, Randall Balestriero, Yann Lecun, and Seth Lloyd (2022). *projUNN: efficient method for training deep networks with unitary matrices*. URL: https://arxiv.org/pdf/2203.05483.pdf.

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980. URL: http://arxiv.org/abs/1412.6980.

Kirov, Christo and Robert Frank (2012). "Processing of nested and cross-serial dependencies: an automaton perspective on SRN behaviour". In: *Connection Science* 24.1, pp. 1–24.

Lakretz, Yair, Théo Desbordes, Jean-Rémi King, Benoît Crabbé, Maxime Oquab, and Stanislas Dehaene (2021). "Can RNNs learn Recursive Nested Subject-Verb Agreements?" In: *arXiv preprint arXiv:2101.02258*.

Lambek, Joachim (2008). "Pregroup Grammars and Chomsky's Earliest Examples". In: *Journal of Logic, Language and Information* 17, pp. 141–160.

Lappin, Shalom (2021). *Deep Learning and Linguistic Representation*. Boca Raton, London, New York: CRC Press, Taylor & Francis.

Lau, Jey Han, Carlos Armendariz, Shalom Lappin, Matthew Purver, and Chang Shu (2020). "How Furiously Can Colorless Green Ideas Sleep? Sentence Acceptability in Context". In: *Transactions of the Association for Computational Linguistics* 8, pp. 296–310.

Linzen, Tal, Grzegorz Chrupała, and Afra Alishahi, eds. (2018). *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Brussels, Belgium: Association for Computational Linguistics.

Linzen, Tal, Grzegorz Chrupała, Yonatan Belinkov, and Dieuwke Hupkes, eds. (Aug. 2019). *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Florence, Italy: Association for Computational Linguistics.

Linzen, Tal, Emmanuel Dupoux, and Yoav Golberg (2016). "Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies". In: *Transactions of the Association of Computational Linguistics* 4, pp. 521–535.

Marvin, Rebecca and Tal Linzen (2018). "Targeted Syntactic Evaluation of Language Models". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, pp. 1192–1202.

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). "Distributed Representations of Words and Phrases and Their Compositionality". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. Lake Tahoe, Nevada, pp. 3111–3119.

Montague, Richard (1970a). "English as a formal language". In: *Linguaggi nella Societa e nella Tecnica*. Ed. by B. Visentini et al.

— (1970b). "Universal grammar". en. In: *Theoria* 36.3, pp. 373–398. ISSN: 1755-2567. DOI: 10.1111/j.1755-2567.1970.tb00434.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1755-2567.1970.tb00434.x (visited on 10/26/2020).

— (1974). "The Proper Treatment of Quantification in Ordinary English". In: *Formal Philosophy*. Ed. by Richmond Thomason. New Haven: Yale UP.

Moss, Lawrence S. (2022). "Algebra and Language: Reasons for (Dis)content". In: *Algebraic Structures In Natural Language*. Ed. by Shalom Lappin and Jean-Philippe Bernardy. Taylor and Francis.

Pulman, Stephen and G. D. Ritchie (1985). *Indexed Grammars and Intersecting Dependencies*. Tech. rep. 23. University of East Anglia, pp. 21–38.

Ranta, Aarne (2004). "Grammatical Framework". In: *Journal of Functional Programming* 14.2, pp. 145–189.

Sennhauser, Luzi and Robert Berwick (2018). "Evaluating the Ability of LSTMs to Learn Context-Free Grammars". In: *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Brussels, Belgium: Association for Computational Linguistics, pp. 115–124.

Shieber, Stuart M. (1985). "Evidence against the context-freeness of natural language". In: *Linguistics and Philosophy* 8.3, pp. 333–343.

Solaiman, Irene, Miles Brundage, J. Clark, Amanda Askell, Ariel Herbert-Voss, Jeff Wu, A. Radford, and J. Wang (2019). "Release Strategies and the Social Impacts of Language Models". In: *ArXiv* abs/1908.09203.

Stabler, Edward P. (2004). "Varieties of crossing dependencies: Structure dependence and mild context sensitivity". In: *Cognitive Science* 93.5, pp. 699–720.

Strang, Gilbert (2016). *Introduction to Linear Algebra, 5th edition*. Wellesley-Cambridge Press. ISBN: 978-09802327-7-6.

Sutskever, Ilya, James Martens, and Geoffrey E. Hinton (2011). "Generating Text with Recurrent Neural Networks". In: *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*. Ed. by Lise Getoor and Tobias Scheffer. Omnipress, pp. 1017–1024. URL: https://icml.cc/2011/papers/524%5C_icmlpaper.pdf.

Valiant, Leslie Gabriel (1975). "General context-free recognition in less than cubic time". In: *Journal of computer and system sciences* 10.2, pp. 308–314.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin (June 2017). "Attention Is All You Need". en. In: *arXiv:1706.03762 [cs]*. arXiv: `1706.03762 [cs]`.

Vorontsov, Eugene, Chiheb Trabelsi, Samuel Kadoury, and Chris Pal (2017). "On Orthogonality and Learning Recurrent Networks with Long Term Dependencies". In: *arXiv*.

Wisdom, Scott, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas (2016). "Full-capacity unitary recurrent neural networks". In: *Advances in neural information processing systems* 29, pp. 4880–4888.

Yang, Zhilin, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le (2019). "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: *ArXiv* abs/1906.08237.

Yu, Xiang, Ngoc Thang Vu, and Jonas Kuhn (2019). "Learning the Dyck language with attention-based Seq2Seq models". In: *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 138–146.