Representing Unordered Data Using Complex-Weighted Multiset Automata

Justin DeBenedetto [1] **David Chiang** [1]

Abstract

Unordered, variable-sized inputs arise in many settings across multiple fields. The ability for set- and multiset-oriented neural networks to handle this type of input has been the focus of much work in recent years. We propose to represent multisets using complex-weighted *multiset automata* and show how the multiset representations of certain existing neural architectures can be viewed as special cases of ours. Namely, (1) we provide a new theoretical and intuitive justification for the Transformer model's representation of positions using sinusoidal functions, and (2) we extend the DeepSets model to use complex numbers, enabling it to outperform the existing model on an extension of one of their tasks.

1. Introduction

Neural networks which operate on unordered, variable-sized input (Vinyals et al., 2016; Wagstaff et al., 2019) have been gaining interest for various tasks, such as processing graph nodes (Murphy et al., 2019), hypergraphs (Maron et al., 2019), 3D image reconstruction (Yang et al., 2020), and point cloud classification and image tagging (Zaheer et al., 2017). Similar networks have been applied to multiple instance learning (Pevný & Somol, 2016).

One such model, DeepSets (Zaheer et al., 2017), computes a representation of each element of the set, then combines the representations using a commutative function (e.g., addition) to form a representation of the set that discards ordering information. Zaheer et al. (2017) provide a proof that any function on sets can be modeled in this way, by encoding sets as base-4 fractions and using the universal function approximation theorem, but their actual proposed model is far simpler than the model constructed by the theorem.

[1]Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA. Correspondence to: Justin DeBenedetto <jdebened@nd.edu>, David Chiang <dchiang@nd.edu>.

Proceedings of the 37th International Conference on Machine Learning, Online, PMLR 119, 2020.

In this paper, we propose to compute representations of multisets using *weighted multiset automata*, a variant of weighted finite-state (string) automata in which the order of the input symbols does not affect the output. This representation can be directly implemented inside a neural network. We show how to train these automata efficiently by approximating them with string automata whose weights form complex, diagonal matrices.

Our representation is a generalization of DeepSets', and it also turns out to be a generalization of the Transformer's position encodings (Vaswani et al., 2017). In Sections 4 and 5, we discuss the application of our representation in both cases.

- Transformers (Vaswani et al., 2017) encode the position of a word as a vector of sinusoidal functions that turns out to be a special case of our representation of multisets. So weighted multiset automata provide a new theoretical and intuitive justification for sinusoidal position encodings. We experiment with several variations on position encodings inspired by this justification, and although they do not yield an improvement, we do find that learned position encodings in our representation do better than learning a different vector for each absolute position.
- We extend the DeepSets model to use our representation, which amounts to upgrading it from real to complex numbers. On an extension of one of their tasks (adding a sequence of one-digit numbers and predicting the units digit), our model is able to reach perfect performance, whereas the original DeepSets model does no better than chance.

2. Definitions

We first review weighted string automata, then modify the definition to weighted multiset automata.

2.1. String automata

Weighted finite automata are commonly pictured as directed graphs whose edges are labeled by symbols and weights, but we use an alternative representation as collections of matrices: for each symbol a, this representation takes all

the transitions labeled a and arranges their weights into an adjacency matrix, called $\mu(a)$. This makes it easier to use the notation and techniques of linear algebra, and it also makes clearer how to implement weighted automata inside neural networks.

Let $\mathbb{K}$ be a commutative semiring; in this paper, $\mathbb{K}$ is either $\mathbb{R}$ or $\mathbb{C}$.

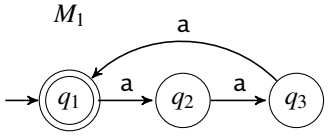
Definition 1. A $\mathbb{K}$-*weighted finite automaton* (WFA) over Σ is a tuple $M = (Q, \Sigma, \lambda, \mu, \rho)$, where:

- $Q = \{1, \ldots, d\}$ is a finite set of states,
- Σ is a finite alphabet,
- $\lambda \in \mathbb{K}^{1\times d}$ is a row vector of initial weights,
- $\mu : \Sigma \to \mathbb{K}^{d\times d}$ assigns a transition matrix to every symbol, and
- $\rho \in \mathbb{K}^{d\times 1}$ is a column vector of final weights.

The weight λ_q is the weight of starting in state q; the weight $[\mu(a)]_{qr}$ is the weight of transitioning from state q to state r on input a, and the weight ρ_q is the weight of accepting in state q. We extend the mapping μ to strings: If $w = w_1 \cdots w_n \in \Sigma^*$, then $\mu(w) = \mu(w_1) \cdots \mu(w_n)$. Then, the weight of w is $\lambda\mu(w)\rho$. In this paper, we are more interested in representing w as a vector rather than a single weight, so define the vector of *forward weights* of w to be $\mathrm{fw}_M(w) = \lambda\mu(w)$. (We include the final weights ρ in our examples, but we do not actually use them in any of our experiments.)

Note that, different from many definitions of weighted automata, this definition does not allow ϵ-transitions, and there may be more than one initial state. (Hereafter, we use ϵ to stand for a small real number.)

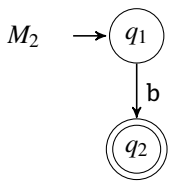
Example 1. Below, M_1 is a WFA that accepts strings where the number of a's is a multiple of three; $(\lambda_1, \mu_1, \rho_1)$ is its matrix representation:



$$\lambda_1 = \begin{bmatrix}1 & 0 & 0\end{bmatrix} \qquad \rho_1 = \begin{bmatrix}1 & 0 & 0\end{bmatrix}^\mathsf{T}$$

$$\mu_1(\mathtt{a}) = \begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\\ 1 & 0 & 0\end{bmatrix}.$$

And M_2 is a WFA that accepts $\{\mathtt{b}\}$; $(\lambda_2, \mu_2, \rho_2)$ is its matrix representation:



$$\lambda_2 = \begin{bmatrix}1 & 0\end{bmatrix} \qquad \rho_2 = \begin{bmatrix}0 & 1\end{bmatrix}^\mathsf{T}$$

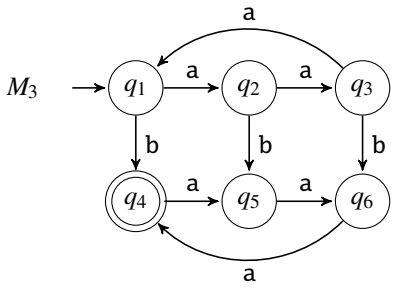
$$\mu_2(\mathtt{b}) = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}.$$

2.2. Multiset automata

The analogue of finite automata for multisets is the special case of the above definition where multiplication of the transition matrices $\mu(a)$ does not depend on their order.

Definition 2. A $\mathbb{K}$-*weighted multiset finite automaton* is one whose transition matrices commute pairwise. That is, for all $a, b \in \Sigma$, we have $\mu(a)\mu(b) = \mu(b)\mu(a)$.

Example 2. Automata M_1 and M_2 are multiset automata because they each have only one transition matrix, which trivially commutes with itself. Below is a multiset automaton that accepts multisets over $\{\mathtt{a}, \mathtt{b}\}$ where the number of a's is a multiple of three and the number of b's is exactly one.



$$\lambda_3 = \begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0\end{bmatrix} \qquad \rho_3 = \begin{bmatrix}0 & 0 & 0 & 1 & 0 & 0\end{bmatrix}^\mathsf{T}$$

$$\mu_3(\mathtt{a}) = \begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & 0 & 0\\ 1 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 1 & 0 & 0\end{bmatrix} \qquad \mu_3(\mathtt{b}) = \begin{bmatrix}0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}.$$

Our multiset automata (Definition 2) describe the *recognizable* distributions over multisets (Sakarovitch, 2009, Section 4.1). A recognizable distribution is intuitively one that we can compute by processing one element at a time using only a finite amount of memory.

This is one of two natural ways to generalize string automata to multiset automata. The other way, which describes all the *rational* distributions of multisets (Sakarovitch, 2009, Section 3.1), places no restriction on transition matrices and computes the weight of a multiset w by summing over all paths labeled by permutations of w. But this is NP-hard to compute.

Our proposal, then, is to represent a multiset w by the vector of forward weights, $\mathrm{fw}_M(w)$, with respect to some weighted multiset automaton M. In the context of a neural network, the transition weights $\mu(a)$ can be computed by any function as long as it does not depend on the ordering of symbols, and the forward weights can be used by the network in any way whatsoever.

3. Training

Definition 2 does not lend itself well to training, because parameter optimization needs to be done subject to the commutativity constraint. Previous work (DeBenedetto & Chiang, 2018) suggested approximating training of a multiset automaton by training a string automaton while using a regularizer to encourage the weight matrices to be close to commuting. However, this strategy cannot make them commute exactly, and the regularizer, which has $O(|\Sigma|^2)$ terms, is expensive to compute.

Here, we pursue a different strategy, which is to restrict the transition matrices $\mu(a)$ to be diagonal. This guarantees that they commute. As a bonus, diagonal matrices are computionally less expensive than full matrices. Furthermore, we show that if we allow complex weights, we can learn multiset automata with diagonal matrices which represent multisets almost as well as full matrices. We show this first for the special case of unary automata (§3.1) and then general multiset automata (§3.2).

3.1. Unary automata

Call an automaton *unary* if $|\Sigma| = 1$. Then, for brevity, we simply write μ instead of $\mu(a)$ where a is the only symbol in Σ.

Let $\|\cdot\|$ be the Frobenius norm; by equivalence of norms (Horn & Johnson, 2012, p. 352), the results below should carry over to any other matrix norm, as long as it is monotone, that is: if $A \le B$ elementwise, then $\|A\| \le \|B\|$.

As stated above, our strategy for training a unary automaton is to allow μ to be complex, but restrict it to be diagonal. The restriction does not lose much generality, for suppose that we observe data generated by a multiset automaton $M = (\lambda, \mu, \rho)$. Then μ can be approximated by a complex diagonalizable matrix by the following well-known result:

Proposition 1 (Horn & Johnson 2012, p. 116)**.** *For any complex square matrix A and $\epsilon > 0$, there is a complex matrix E such that $\|E\| \le \epsilon$ and $A + E$ is diagonalizable in $\mathbb{C}$.*

Proof. Form the Jordan decomposition $A = PJP^{-1}$. We can choose a diagonal matrix D such that $\|D\| \le \frac{\epsilon}{\kappa(P)}$ (where $\kappa(P) = \|P\|\|P^{-1}\|$) and the diagonal entries of $J + D$ are all different. Then $J + D$ is diagonalizable. Let $E = PDP^{-1}$; then $\|E\| \le \|P\|\|D\|\|P^{-1}\| = \kappa(P)\|D\| \le \epsilon$, and $A + E = P(J + D)P^{-1}$ is also diagonalizable. □

So M is close to an automaton $(\lambda, \mu + E, \rho)$ where $\mu + E$ is diagonalizable. Furthermore, by a change of basis, we can make $\mu + E$ diagonal without changing the automaton's behavior:

Proposition 2. *If $M = (\lambda, \mu, \rho)$ is a multiset automaton and P is an invertible matrix, the multiset automaton (λ', μ', ρ') where*

$$\begin{aligned}\lambda' &= \lambda P^{-1}\\ \mu'(a) &= P\mu(a)P^{-1}\\ \rho' &= P\rho\end{aligned}$$

computes the same multiset weights as M.

This means that in training, we can directly learn complex initial weights λ' and a complex diagonal transition matrix μ', and the resulting automaton M' should be able to represent multisets almost as well as a general unary automaton would.

Example 3. In Example 1, $\mu_1(\mathtt{a})$ is diagonalizable:

$$\begin{aligned}\lambda_1' &= \begin{bmatrix}\frac{1}{3} & \frac{1}{3} & \frac{1}{3}\end{bmatrix}\\ \rho_1' &= \begin{bmatrix}1 & 1 & 1\end{bmatrix}^\mathsf{T}\\ \mu_1'(\mathtt{a}) &= \operatorname{diag}\begin{bmatrix}1 & \exp\frac{2\pi}{3}i & \exp-\frac{2\pi}{3}i\end{bmatrix}.\end{aligned}$$

On the other hand, $\mu_2(\mathtt{b})$ is only approximately diagonalizable:

$$\begin{aligned}\lambda_2' &= \begin{bmatrix}\frac{1}{2\epsilon} & -\frac{1}{2\epsilon}\end{bmatrix}\\ \rho_2' &= \begin{bmatrix}1 & 1\end{bmatrix}^\mathsf{T}\\ \mu_2'(\mathtt{b}) &= \operatorname{diag}\begin{bmatrix}\epsilon & -\epsilon\end{bmatrix}.\end{aligned}$$

But μ_2' can be made arbitrarily close to μ_2 by making ϵ sufficiently close to zero.

It might be thought that even if μ' approximates μ well, perhaps the forward weights, which involve possibly large powers of μ, will not be approximated well. As some additional assurance, we have the following error bound on the powers of μ:

Theorem 3. *For any complex square matrix A, $\epsilon > 0$, and $0 < r < 1$, there is a complex matrix E such that $A + E$ is diagonalizable in $\mathbb{C}$ and, for all $n \ge 0$,*

$$\begin{aligned}&\|(A+E)^n - A^n\| \le r^n\epsilon && \text{if } A \text{ nilpotent,}\\ &\frac{\|(A+E)^n - A^n\|}{\|A^n\|} \le n\epsilon && \text{otherwise.}\end{aligned}$$

In other words, if A is not nilpotent, the relative error in the powers of A increase only linearly in the exponent; if A is nilpotent, we can't measure relative error because it eventually becomes undefined, but the absolute error decreases exponentially.

Proof. See Appendix A. □

3.2. General case

In this section, we allow Σ to be of any size. Proposition 1 unfortunately does not hold in general for multiple matrices (O'Meara & Vinsonhaler, 2006). That is, it may not be possible to perturb a set of commuting matrices so that they are *simultaneously* diagonalizable.

Definition 3. Matrices $A_1, \ldots, A_m$ are *simultaneously diagonalizable* if there exists an invertible matrix P such that PA_iP^{-1} is diagonal for all $i \in \{1, \cdots, n\}$.

We say that $A_1, \cdots, A_m$ are *approximately simultaneously diagonalizable (ASD)* if, for any $\epsilon > 0$, there are matrices $E_1, \ldots, E_m$ such that $\|E_i\| \leq \epsilon$ and $A_1 + E_1, \ldots, A_m + E_m$ are simultaneously diagonalizable.

O'Meara & Vinsonhaler (2006) give examples of sets of matrices that are commuting but not ASD. However, if we are willing to add new states to the automaton (that is, to increase the dimensionality of the transition matrices), we can make them ASD.

Theorem 4. *For any weighted multiset automaton, there is an equivalent complex-weighted multiset automaton, possibly with more states, whose transition matrices are ASD.*

We extend Proposition 1 from unary automata to non-unary multiset automata that have a special form; then, we show that any multiset automaton can be converted to one in this special form, but possibly with more states.

Let $\oplus$ stand for direct sum of vectors or matrices, $\otimes$ for the Kronecker product, and define the shuffle product (also known as the Kronecker sum) $A \sqcup\!\sqcup B = A \otimes I + I \otimes B$. These operations extend naturally to weighted multiset automata (DeBenedetto & Chiang, 2018): If $M_A = (\lambda_A, \mu_A, \rho_A)$ and $M_B = (\lambda_B, \mu_B, \rho_B)$, then define

$$\begin{aligned} M_A \oplus M_B &= (\lambda_\oplus, \mu_\oplus, \rho_\oplus) & M_A \sqcup\!\sqcup M_B &= (\lambda_{\sqcup\!\sqcup}, \mu_{\sqcup\!\sqcup}, \rho_{\sqcup\!\sqcup}) \\ \lambda_\oplus &= \lambda_A \oplus \lambda_B & \lambda_{\sqcup\!\sqcup} &= \lambda_A \otimes \lambda_B \\ \mu_\oplus(a) &= \mu_A(a) \oplus \mu_B(a) & \mu_{\sqcup\!\sqcup}(a) &= \mu_A(a) \sqcup\!\sqcup \mu_B(a) \\ \rho_\oplus &= \rho_A \oplus \rho_B & \rho_{\sqcup\!\sqcup} &= \rho_A \otimes \rho_B. \end{aligned}$$

$M_A \oplus M_B$ recognizes the union of the multisets recognized by M_A and M_B; if M_A and M_B use disjoint alphabets, then $M_A \sqcup\!\sqcup M_B$ recognizes the concatenation of the multisets recognized by M_A and M_B.

They are of interest here because they preserve the ASD property, so non-unary automata formed by applying direct sum and shuffle product to unary automata are guaranteed to have ASD transition matrices.

Proposition 5. *If M_1 and M_2 are multiset automata with ASD transition matrices, then $M_1 \oplus M_2$ has ASD transition matrices, and $M_1 \sqcup\!\sqcup M_2$ has ASD transition matrices.*

Proof. See Appendix B. □

Example 4. Consider M_1 and M_2 from Example 1. If we assume that $\mu_1(\mathsf{b})$ and $\mu_2(\mathsf{a})$ are zero matrices, then the shuffle product of M_2 and M_1 is exactly M_3 from Example 2. So the transition matrices of M_3 are ASD:

$$\begin{aligned} \lambda_3' &= \begin{bmatrix} \frac{1}{6\epsilon} & \frac{1}{6\epsilon} & \frac{1}{6\epsilon} & -\frac{1}{6\epsilon} & -\frac{1}{6\epsilon} & -\frac{1}{6\epsilon} \end{bmatrix} \\ \mu_3'(\mathsf{a}) &= \operatorname{diag}\begin{bmatrix} 1 & e^{\frac{2\pi}{3}i} & e^{-\frac{2\pi}{3}i} & 1 & e^{\frac{2\pi}{3}i} & e^{-\frac{2\pi}{3}i} \end{bmatrix} \\ \mu_3'(\mathsf{b}) &= \operatorname{diag}\begin{bmatrix} \epsilon & \epsilon & \epsilon & -\epsilon & -\epsilon & -\epsilon \end{bmatrix} \\ \rho_3' &= \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^\mathsf{T}. \end{aligned}$$

The next two results give ways of expressing multiset automata as direct sums and/or shuffle products of smaller automata. Lemma 6 expresses a multiset automaton as a direct sum of smaller automata, without increasing the number of states, but does not guarantee that the smaller automata are ASD. Proposition 7 expresses a multiset automaton as a direct sum of shuffle products of unary automata, but can increase the number of states.

Lemma 6 (O'Meara & Vinsonhaler 2006)**.** *Suppose $A_1, \ldots, A_k$ are commuting $n \times n$ matrices over an algebraically closed field. Then there exists an invertible matrix C such that $C^{-1}A_1C, \ldots, C^{-1}A_kC$ are block diagonal matrices with matching block structures and each diagonal block has only a single eigenvalue (ignoring multiplicities). That is, there is a partition $n = n_1 + \cdots + n_r$ of n such that*

$$C^{-1}A_iC = B_i = \begin{bmatrix} B_{i1} & & & \\ & B_{i2} & & \\ & & \ddots & \\ & & & B_{ir} \end{bmatrix}. \tag{1}$$

where each B_{ij} is an $n_j \times n_j$ matrix having only a single eigenvalue for $i = 1, \ldots, k$ and $j = 1, \ldots, r$.

Proposition 7. *If M is a weighted multiset automaton with d states over an alphabet with m symbols, there exists an equivalent complex-weighted multiset automaton with $\binom{2m+d}{d-1}$ states whose transition matrices are ASD.*

Proof. See Appendix C. □

Proof of Theorem 4. By Lemma 6, we can put the transition matrices into the form (1). By Proposition 7, for each j, we can convert $B_{1j}, \ldots B_{kj}$ into ASD matrices $B'_{1j}, \ldots, B'_{kj}$, and by Proposition 5, their direct sum $B'_{1j} \oplus \cdots \oplus B'_{kj}$ is also ASD. □

This means that if we want to learn representations of multisets over a finite alphabet Σ, it suffices to constrain the transition matrices to be complex diagonal, possibly with more states. Unfortunately, the above construction increases the number of states by a lot. But this does not in any way

prevent the use of our representation; we can choose however many states we want, and it's an empirical question whether the number of states is enough to learn good representations.

The following two sections look at two practical applications of our representation.

4. Position Encodings

One of the distinguishing features of the Transformer network for machine translation (Vaswani et al., 2017), compared with older RNN-based models, is its curious-looking *position encodings*,

$$\begin{aligned} \mathbf{e}^p_{2j-1} &= \sin 10000^{-2(j-1)/d}(p-1) \\ \mathbf{e}^p_{2j} &= \cos 10000^{-2(j-1)/d}(p-1) \end{aligned} \tag{2}$$

which map word positions p (ranging from 1 to n, the sentence length) to points in the plane and are the model's sole source of information about word order.

In this section, we show how these position encodings can be interpreted as the forward weights of a weighted unary automaton. We also report on some experiments on some extensions of position encodings inspired by this interpretation.

4.1. As a weighted unary automaton

Consider a diagonal unary automaton M in the following form:

$$\lambda = \begin{bmatrix} s_1 \exp i\phi_1 & s_1 \exp -i\phi_1 & s_2 \exp i\phi_2 & s_2 \exp -i\phi_2 & \cdots \end{bmatrix}$$

$$\mu = \begin{bmatrix} r_1 \exp i\theta_1 & 0 & 0 & 0 & \cdots \\ 0 & r_1 \exp -i\theta_1 & 0 & 0 & \cdots \\ 0 & 0 & r_2 \exp i\theta_2 & 0 & \cdots \\ 0 & 0 & 0 & r_2 \exp -i\theta_2 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

In order for a complex-weighted automaton to be equivalent to some real-weighted automaton, the entries must come in conjugate pairs like this, so this form is fully general.

By a change of basis, this becomes the following unary automaton M' (this is sometimes called the real Jordan form):

$$\lambda' = \begin{bmatrix} s_1 \cos\phi_1 & s_1 \sin\phi_1 & s_2 \cos\phi_2 & s_2 \sin\phi_2 & \cdots \end{bmatrix}$$

$$\mu' = \begin{bmatrix} r_1 \cos\theta_1 & r_1 \sin\theta_1 & 0 & 0 & \cdots \\ -r_1 \sin\theta_1 & r_1 \cos\theta_1 & 0 & 0 & \cdots \\ 0 & 0 & r_2 \cos\theta_2 & r_2 \sin\theta_2 & \ldots \\ 0 & 0 & -r_2 \sin\theta_2 & r_2 \cos\theta_2 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{3}$$

so that for any string prefix u (making use of the angle sum identities):

$$\mathrm{fw}_{M'}(u)^\top = \begin{bmatrix} s_1 r_1^{|u|} \cos(\phi_1 + |u|\theta_1) \\ s_1 r_1^{|u|} \sin(\phi_1 + |u|\theta_1) \\ s_2 r_2^{|u|} \cos(\phi_2 + |u|\theta_2) \\ s_2 r_2^{|u|} \sin(\phi_2 + |u|\theta_2) \\ \vdots \end{bmatrix}.$$

If we let

$$\begin{aligned} s_i &= 1 & \phi_i &= \frac{\pi}{2} \\ r_i &= 1 & \theta_j &= -10000^{-2(j-1)/d} \end{aligned}$$

this becomes exactly equal to the position encodings defined in (2). Thus, the Transformer's position encodings can be reinterpreted as follows: it runs automaton M' over the input string and uses the forward weights of M' just before position p to represent p. This encoding, together with the embedding of word w_p, is used as the input to the first self-attention layer.

4.2. Experiments

This reinterpretation suggests that we might be able to learn position encodings instead of fixing them heuristically. Vaswani et al. (2017), following Gehring et al. (2017) and followed by Devlin et al. (2019), learn a different encoding for each position, but multiset automata provide parameterizations that use many fewer parameters and hopefully generalize better.

We carried out some experiments to test this hypothesis, using an open-source implementation of the Transformer, Witwicky.[1] The settings used were the default settings, except that we used 8k joint BPE operations and $d = 512$ embedding dimensions. We tested the following variations on position encodings.

- Diagonal polar: multiset automaton as in eq. (3)
  - fixed: The original sinusoidal encodings (Vaswani et al., 2017).
  - learned angles: Initialize the ϕ_i and θ_i to the original values, then optimize them.
- Full matrix: multiset automaton with real weights
  - random: Randomize initial weights so that their expected norm is the same as the original, and transition matrix using orthogonal initialization (Saxe et al., 2014), and do not optimize them.
  - learned: Initialize λ and μ as above, and then optimize them.

[1] https://github.com/tnq177/witwicky

Table 1. Machine translation experiments with various position encodings. Scores are in case-insensitive BLEU, a common machine translation metric. The best score in each column is printed in boldface.

		case-insensitive BLEU							
Model	Training	En-Vi*	Uz-En	Ha-En	Hu-En	Ur-En	Ta-En	Tu-En	combined
diagonal polar	fixed	32.6	25.7	24.4	**34.2**	11.5	13.4	25.7	26.4
	learned angles	**32.7**	25.8	25.4†	34.0	11.1†	14.1†	25.7	**26.6**
full matrix	random	32.6	**25.9**	**25.6**†	34.1	11.1†	12.6†	**26.1**	26.5
	learned	32.5	24.5†	23.6	33.5	11.4	**14.5**†	23.8†	26.5
per position	random	32.6	24.3†	24.6	33.6†	11.1	14.0†	25.7	26.3
	learned	32.0†	22.6†	21.2†	33.0†	**11.7**	14.4†	21.1†	25.0†

*tokenized references
†significantly different from first line ($p < 0.05$, bootstrap resampling)

- Per position: a real vector for each position
  - random: Choose a random vector with fixed norm for each absolute position, and do not optimize them.
  - learned: Initialize per-position encodings as above, then optimize them (Gehring et al., 2017).

Table 1 compares these methods on seven low-resource language pairs (with numbers of training tokens ranging from 100k to 2.3M), with the final column computed by concatenating all seven test sets together. Although learning position encodings using multiset automata ("diagonal polar, learned angles" and "full matrix, learned") does not do better than the original sinusoidal encodings (the 0.2 BLEU improvement is not statistically significant), they clearly do better than learning per-position encodings, supporting our view that multiset automata are the appropriate way to generalize sinusoidal encodings.

5. Complex DeepSets

In this section, we incorporate a weighted multiset automaton into the DeepSets (Zaheer et al., 2017) model, extending it to use complex numbers. Our code is available online.[2]

5.1. Models

The DeepSets model computes a vector representation for each input symbol and sums them to discard ordering information. We may think of the elementwise layers as computing the log-weights of a diagonal multiset automaton, and the summation layer as computing the forward log-weights of the multiset. (The logs are needed because DeepSets adds, whereas multiset automata multiply.) However, DeepSets uses only real weights, whereas our multiset automata use complex weights. Thus, DeepSets can be viewed as using a multiset representation which is a special case of ours.

We conduct experiments comparing the DeepSets model (Zaheer et al., 2017), a GRU model, an LSTM model, and our complex multiset model. The code and layer sizes for the three baselines come from the DeepSets paper.[3] See Figure 1 for layer types and sizes for the three baseline models.

In our system, to avoid underflow when multiplying many complex numbers, we store each complex number as $e^r(a + bi)$ where r, a, and b are real and a and b are normalized such that $a^2 + b^2 = 1$ prior to multiplication. Thus, for each complex-valued parameter, we have three real-valued scalars (r, a, and b) to learn. To this end, each input is fed into three separate embedding layers of size 50 (for r, a, and b). Since the weight on states occurs in complex conjugate pairs within the diagonalized multiset automata, we only need to store half the states. This is why we use 50 rather than 100 for our embeddings. (While the DeepSets code uses a dense layer at this point, in our network, we found that we could feed the embeddings directly into a complex multiplication layer to discard ordering information. This reduced the number of parameters for our model and did not affect performance.) The output of this is then a new r, a, and b which are concatenated and fed into a final dense layer as before to obtain the output. Since our diagonalized automata have complex initial weights (λ'), we also tried learning a complex initial weight vector λ', but this had no effect on performance.

The total number of parameters for each model was 4,161 parameters for the DeepSets model, 31,351 parameters for the LSTM model, 44,621 parameters for the GRU model, and 1,801 parameters for our model. In order to eliminate number of parameters as a difference from our model to the

[2] https://github.com/jdebened/ComplexDeepSets

[3] https://github.com/manzilzaheer/DeepSets/blob/master/DigitSum/text_sum.ipynb

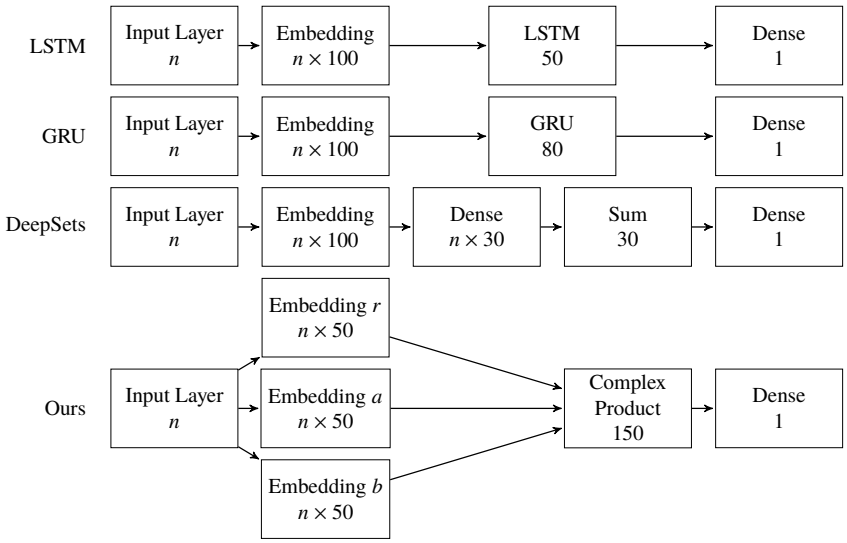


Figure 1. Models compared in Section 5. Each cell indicates layer type and output dimension.

DeepSets model, we also tried the DeepSets model without the first dense layer and with embedding sizes of 150 to exactly match the number of parameters of our model, and the results on the test tasks were not significantly different from the baseline DeepSets model.

For tasks 1 and 2, we used mean squared error loss, a learning rate decay of 0.5 after the validation loss does not decrease for 2 epochs, and early stopping after the validation loss does not decrease for 10 epochs.

5.2. Experiments

Task 0: Recovering multiset automata To test how well complex diagonal automata can be trained from string weights, we generate a multiset automaton and train our model on strings together with their weights according to the automaton. Since we want to test the modeling power of complex multiset automata, we remove the final dense layer and replace it with a simple summation for this task only. It is worth noting that this is equivalent to setting all final weights to 1 by multiplying the final weights into the initial weights, therefore we lose no modeling power by doing this. The embedding dimension is set to match the number of states in the generated multiset automaton. The size of the input alphabet is set to 1 for the unary case and 5 for the complex diagonal case. We train by minimizing mean squared error. As a baseline for comparison, we compute the average string weight generated by each automaton and use that as the prediction for the weight of all strings generated by that automaton.

We generate unary automata by sampling uniformly from the Haar distribution over orthogonal matrices (Mezzadri, 2007). The training strings are every unary string from length 0 to 20 (21 training strings). Due to the small number of training strings, we let these models train for up to 30k epochs with early stopping when loss does not improve for 100 epochs. We generate complex diagonal automata by sampling real and imaginary coefficients uniformly from [0, 1] then renormalizing by the largest entry so the matrix has spectral radius 1. String lengths are fixed at 5 to avoid large discrepancies in string weight magnitudes. All strings of length 5 are generated as training data. For each dimension, 10 automata are generated. We train 10 models on each and select the best model as indicative of how well our model is capable of learning the original multiset automaton.

Task 1: Sum of digits In this task, taken from Zaheer et al. (2017), the network receives a set of single digit integers as input and must output the sum of those digits. The output is rounded to the nearest integer to measure accuracy. The training set consisted of 100k randomly generated sequences of digits 1–9 with lengths from 1 to 50. They were fed to each network in the order in which they were generated (which only affects GRU and LSTM). This was then split into training and dev with approximately a 99/1 split. The test set consisted of randomly generated sequences of lengths that were multiples of 5 from 5 to 95. Figure 3 shows that both our model and DeepSets obtain perfect accuracy on the test data, while the LSTM and GRU fail to generalize to longer sequences.

Task 2: Returning units digit of a sum The second task is similar to the first, but only requires returning the units digit of the sum. The data and evaluation are otherwise the same as task 1. Here, random guessing within the output range of 0–9 achieves approximately 10% accuracy. Figure 3 shows that DeepSets, LSTM, and GRU are unable to achieve

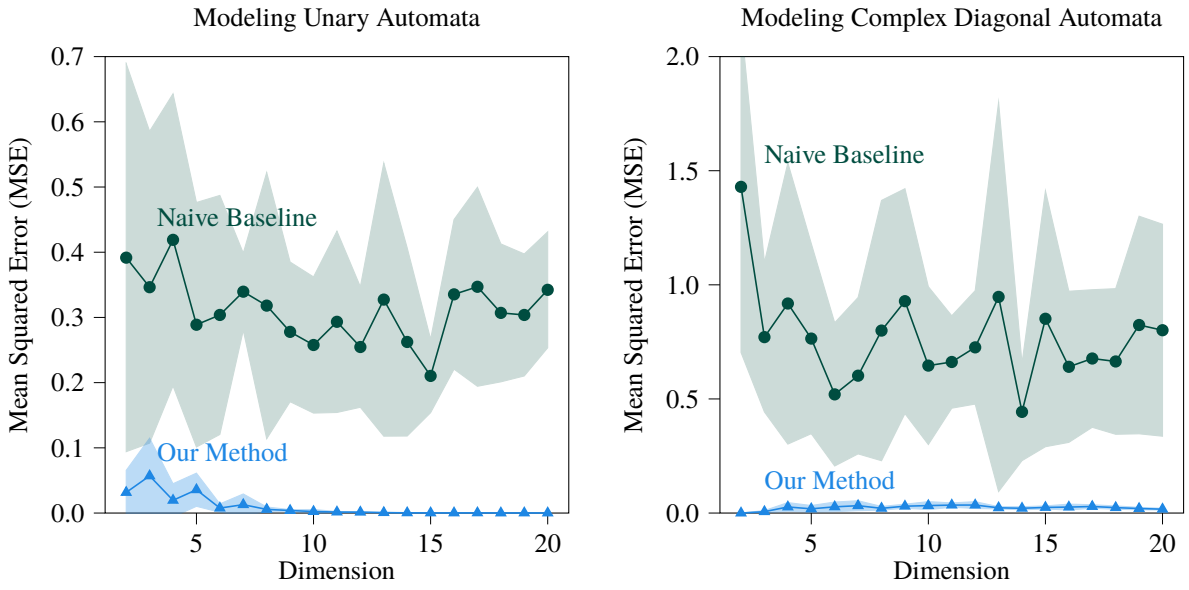


Figure 2. Results for Task 0: Training loss for modeling multiset automata with learned complex diagonal automata. For each set of data, the learned automaton with a complex diagonal transition matrix is able to approximate a unary (left) or diagonal (right) multiset automaton using the same number of states. Error bands show ±1 standard deviation.

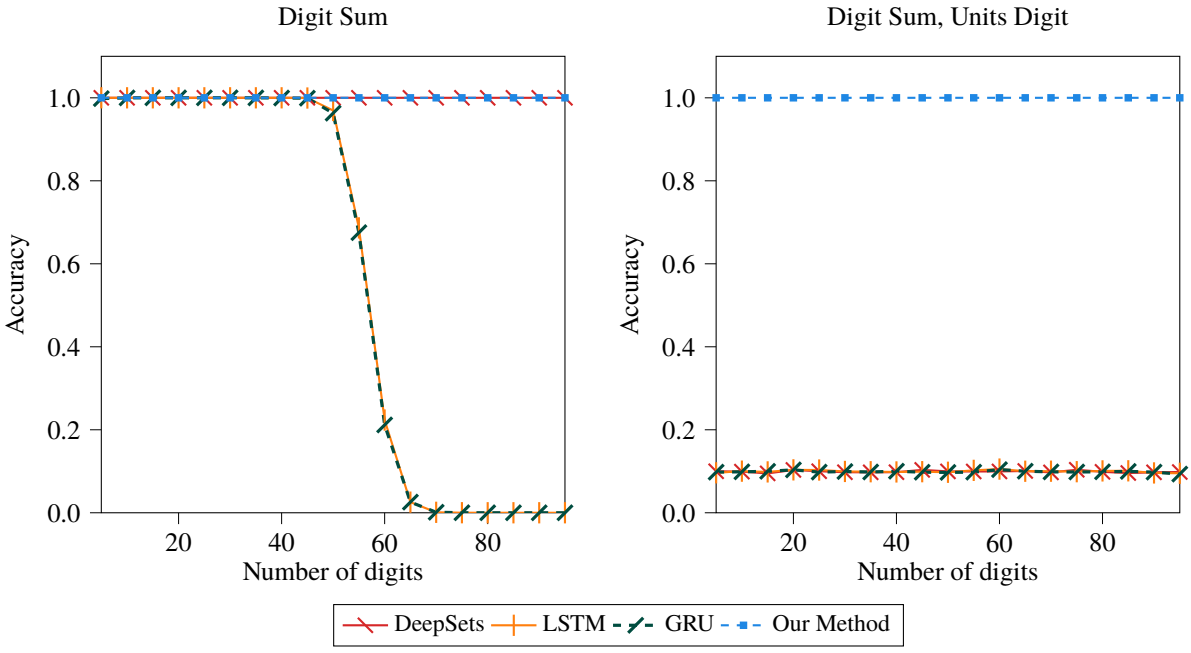


Figure 3. Results for Task 1 (left) and Task 2 (right). In task 1, the LSTM and GRU models were unable to generalize to examples larger than seen in training, while DeepSets and our model generalize to all test lengths. For task 2, only our model is able to return the correct units digit for all test lengths. The GRU, LSTM, and DeepSets models fail to learn any behavior beyond random guessing.

performance better than random guessing on the test data. Our method is able to return the units digit perfectly for all test lengths, because it effectively learns to use the cyclic nature of complex multiplication to produce the units digit.

6. Conclusion

We have proven that weighted multiset automata can be approximated by automata with (complex) diagonal transition matrices. This formulation permits simpler elementwise multiplication instead of matrix multiplication, and requires fewer parameters when using the same number of states. We show that this type of automaton naturally arises within existing neural architectures, and that this representation generalizes two existing multiset representations, the Transformer's position encodings and DeepSets. Our results provide new theoretical and intuitive justification for these models, and, in one case, lead to a change in the model that drastically improves its performance.

Acknowledgements

We would like to thank Toan Nguyen for providing his implementation of the Transformer and answering many questions about it.

This research is based upon work supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via contract #FA8650-17-C-9116. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

DeBenedetto, J. and Chiang, D. Algorithms and training for weighted multiset automata and regular expressions. In Câmpeanu, C. (ed.), *Implementation and Application of Automata*, pp. 146–158. Springer, 2018.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. NAACL HLT*, pp. 4171–4186, 2019.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. Convolutional sequence to sequence learning. In *Proc. ICML*, 2017.

Horn, R. A. and Johnson, C. A. *Matrix Analysis*. Cambridge Univ. Press, 2nd edition, 2012.

Kozma, L. Useful inequalities, 2019. URL `http://www.Lkozma.net/inequalities_cheat_sheet`.

Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y. Invariant and equivariant graph networks. In *Proc. ICLR*, 2019.

Mezzadri, F. How to generate random matrices from the classical compact groups. *Notices of the AMS*, 54(5): 592–604, 2007.

Murphy, R. L., Srinivasan, B., Rao, V. A., and Ribeiro, B. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *Proc. ICLR*, 2019.

O'Meara, K. and Vinsonhaler, C. On approximately simultaneously diagonalizable matrices. *Linear Algebra and its Applications*, 412(1):39–74, 2006.

Pevný, T. and Somol, P. Using neural network formalism to solve multiple-instance problems. In *Proc. ISNN*, 2016.

Sakarovitch, J. Rational and recognisable power series. In *Handbook of Weighted Automata*, pp. 105–174. Springer, 2009.

Saxe, A. M., McClelland, J. L., and Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *Proc. ICLR*, 2014.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Proc. NeurIPS*, 2017.

Vinyals, O., Bengio, S., and Kudlur, M. Order matters: Sequence to sequence for sets. In *Proc. ICLR*, 2016.

Wagstaff, E., Fuchs, F. B., Engelcke, M., Posner, I., and Osborne, M. A. On the limitations of representing functions on sets. In *Proc. ICML*, 2019.

Yang, B., Wang, S., Markham, A., and Trigoni, N. Robust attentional aggregation of deep feature sets for multi-view 3D reconstruction. *International Journal of Computer Vision*, 128:53–73, 2020.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Póczos, B., Salakhutdinov, R., and Smola, A. J. Deep sets. In *Proc. NeurIPS*, 2017.