A Sequence transformations used to construct classification and transduction tasks

In tables 6, 7 we describe the transformations used to construct classification and transduction tasks, respectively.

	Transformation	Description
S_1	mul v	Elementwise multiply by v
	add v	Elementwise add v
	div v	Elementwise integer division by v
	mod v	Elementwise modulo v operation
S_2	(not) multiple of v	Extract subset of integers that are (not) multiples of v
	(not) greater of v	Extract subset of integers that are (not) greater than v
	(do not) have exactly v divisors	Extract subset of integers that (do not) have exactly v divisors
S_3	count	Sequence length
	min	Smallest integer in sequence
	max	Largest integer in sequence
	mean	Mean of sequence elements
	median	Median of sequence elements
	mode	Mode of sequence elements
	first	First element in sequence
	last	Last element in sequence
	max-min	Difference between largest and smallest elements in sequence
	middle	Element in the middle position of sequence

Table 6: Sequence transformations used to construct classification tasks and their descriptions. Each transformation takes a sequence as input and outputs a sequence (transformations in S_1 and S_2), or a single integer (transformations in S_1).

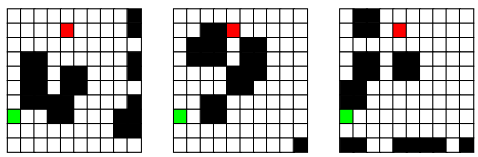
	Transformation	Description
S_1	mul v	Elementwise multiply by v
	add v	Elementwise add v
	div v	Elementwise integer division by v
	mod v	Elementwise modulo v operation
S_2	reverse v with v'	Replace all occurrences of v in the sequence with v'
	replace x_i with $f(x_i, x_j)$	Replace element x_i with one of the following: $\{ax_i + b,\ x_j,\ \text{abs}(x_i - x_j),\ x_i + x_j\}$ where a, b are integer constants and x_i, x_j are elements of the sequence at position i, j respectively
S_3	sort ascending	Sort the sequence in ascending order
	sort descending	Sort the sequence in descending order
	reverse	Reverse the sequence
	swap(x_i, x_j)	Swap elements at positions i, j of the sequence
	shift right v	Cyclic shift the sequence right by v positions

Table 7: Sequence transformations used to construct transduction tasks and their descriptions. Each transformation takes a sequence as input and outputs a sequence.

B Path-finding task

B.1 Non-compositional path-finding task

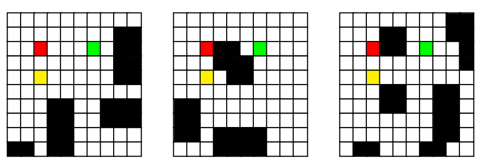
We present an example task from the path-finding task below. The grids are 10×10. The following task is defined by the start position (7, 0) and end position (1, 4), indicated by the green and red squares, respectively. Each example in the task corresponds to a particular configuration of obstacles in the grid. The source sequence represents the locations of obstacles. The obstacles are represented by the top left position of a 2×2 blob. The target sequence represents the optimal path from source to target. Source and target sequences consist of rasterized grid coordinates (Eg. rasterized start and end positions are 70 and 14, respectively). In addition, elements of the target sequence have an offset of 100 (Eg. rasterized position 14 is represented as 114).



- Source: [39, 78, 51, 9, 31, 63, 44, 69], Target: [170, 160, 150, 140, 130, 121, 112, 103, 114]
- Source: [12, 35, 99, 22, 62, 44, 25, 21], Target: [170, 161, 152, 143, 134, 124, 114]
- Source: [90, 99, 1, 96, 34, 50, 94, 31], Target: [170, 171, 162, 152, 143, 133, 123, 114]

B.2 Compositional path-finding task

In the compositional setting, we require the optimal path to pass through a way-point, indicated in yellow in the following grids. A task is thus defined by a start position, end position and way-point position. The possible values for each of these three parameters represent the primitives in this compositional setting.



- Source: [63, 38, 90, 93, 73, 68, 18, 67], Target: [126, 115, 124, 133, 142, 131, 122]
- Source: [95, 60, 95, 70, 23, 34, 83, 85], Target: [126, 115, 104, 113, 122, 131, 142, 131, 122]
- Source: [91, 29, 57, 96, 8, 53, 77, 13], Target: [126, 125, 134, 133, 142, 131, 122]

C Compositional TAM

Algorithm 2 presents the training algorithm for compositional TAM. We draw a training task $\mathcal{T}^{\text{train}}$ with primitive ids $T_1 = i_1, T_2 = i_2, T_3 = i_3$ respectively in line 3. These primitive ids index into the primitive embedding table θ_e. We pretend that one of the primitives is unknown, and to illustrate the algorithm, we assume without loss of generality that $T_2 = i_2$ is unknown (line 5). In the inner loop optimization, we infer an embedding z for this unknown primitive using gradient descent, while using the primitive embedding table to load the known primitive embeddings ($\theta_e[i_1], \theta_e[i_3]$ in this case (lines 8, 9)).

Algorithm 2: Compositional TAM for k-shot Learning

Input : Training tasks $\mathcal{T}_1^{\text{train}}, ..., \mathcal{T}_N^{\text{train}}$
Output : Model parameters θ, primitive embeddings θ_e

1 $\theta' = \theta \cup \theta_e$
2 **repeat**
3   Sample training task $\mathcal{T}^{\text{train}}$ with primitive ids $T_1 = i_1, T_2 = i_2, T_3 = i_3$
4   Sample k training examples from the task $\{(x^j, y^j)_{j=1,\cdots,k}\} \sim \mathcal{T}^{\text{train}}$
5   Pretend one of the primitives (chosen at random) is unknown, say T_2
6   Initialize $z = 0, \Delta\theta' = 0$
7   **while** *loss improves and max iterations not reached* **do**
8         $z \leftarrow z - \nabla_z \sum_{j=1}^{k} - \log p(y^j | x^j, z_1 = \theta_e[i_1], z_2 = z, z_3 = \theta_e[i_3]; \theta')$
9         $\Delta\theta' \leftarrow \Delta\theta' - \nabla_{\theta'} \sum_{j=1}^{k} - \log p(y^j | x^j, z_1 = \theta_e[i_1], z_2 = z, z_3 = \theta_e[i_3]; \theta')$
10     $\theta' \leftarrow \theta' + \Delta\theta'$
11 **until** *max training iterations*;

D Model Architecture

Figure 2 shows an illustration of how we use transformers for sequence classification (left) and sequence transduction (right) problems. In the classification setting the input is a sequence $(x_1 \cdots x_n)$ and the output is a discrete label y. In the transduction setting, the input $(x_1 \cdots x_n)$ and output $(y_1 \cdots y_m)$ are sequences. z is an embedding vector we refer to as the *task embedding* and appears in the input to the transformer, in addition to the input sequence. The task embedding z is task specific, and is inferred on the fly for each task during training. Learning a new task $\mathcal{T}$ at test time involves inferring the corresponding task embedding $z_{\mathcal{T}}$, leaving the rest of the model parameters untouched.

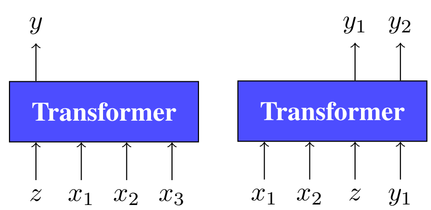


Figure 2: Illustration of how we use transformers for sequence classification (left) and sequence transduction (right) problems.