A Models

Figure 1 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.

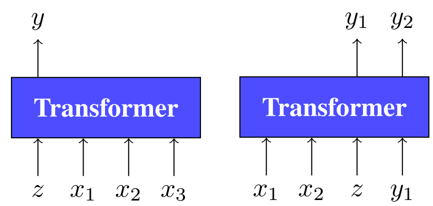


Figure 1: Illustration of how we use transformers for sequence classification (left) and sequence transduction (right) problems.

B Hyperparameters

Unless otherwise specified our transformer has 4 layers with an embedding size of 128. We use the Adam optimizer (Kingma & Ba, 2015) for both outer and inner loop optimization. The maximum number of task embedding optimization steps is set to 25 during training. We train a single model using N samples at training time, treating N as a hyperparameter and apply it to k-shot problems with different values of k at test time. The size of the task embedding was set to match the embedding dimension of the transformer (128).

C Baselines

Matching Networks (Vinyals et al., 2016) This model computes the similarity between the query input and each training example. In our implementation we adapted the original vision model to our sequence learning setting by feeding the concatenation of both sequences to a transformer which outputs a single scalar score, which can be interpreted as the similarity between the two input sequences. The prediction is a convex sum of the training example labels, the weights being the similarity scores. We consider Matching Networks only in our classification setting, as it is not straightforward to use it for transduction. Matching networks are learned using the multi-task training loss; at test time they are applied to new tasks without any finetuning.

SNAIL (Mishra et al., 2018) This model is similar to the task-agnostic transformer except that the input is augmented with the concatenation of all input-output training pairs. Similarly to Matching Networks, SNAIL is learned using the multi-task training loss and applied to new test tasks without finetuning. For both Matching Networks and SNAIL, we construct training episodes by sampling k training examples to define a task, to match with the test scenario. We thus train different models for each k-shot problem.

MAML (Finn et al., 2017) All model parameters are trained using MAML, with the same model architecture as TAM. The entire model is fine-tuned on test tasks.

CAVIA (Zintgraf et al., 2019) Similar to TAM, CAVIA has a set of task-specific parameters and shared parameters. The training algorithm is similar to MAML, but inner loop updates are performed on the task-specific parameters as opposed to the entire model. Our implementation of CAVIA, MAML uses the *higher* library (Grefenstette et al., 2019).

D Sequence transformations used to construct classification and transduction tasks

In tables 5, 6 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 5: 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 6: Sequence transformations used to construct transduction tasks and their descriptions. Each transformation takes a sequence as input and outputs a sequence.

E 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

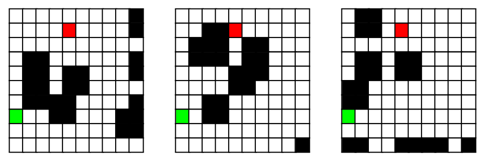
Input : Training tasks $\mathcal{T}_1^{\text{train}}, ..., \mathcal{T}_N^{\text{train}}$
Output : Model parameters θ, primitive embeddings θ_e

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*;

F Path-finding task

F.1 Non-compositional path-finding task

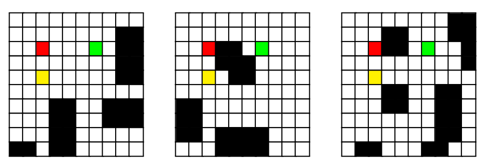
We present an example task from the path-finding task below. The grids are 10×10. The following task is defined by the start position (7, 0) and end position (1, 4), indicated by the green and red squares, respectively. Each example in the task corresponds to a particular configuration of obstacles in 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]

F.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]

G Results with error bars

Tables 7, 8 show the error bars on results in the main paper.

Model	Sequence Classification				Sequence Transduction				Path Finding			
	1	5	10	20	1	5	10	20	1	5	10	20
Task Agnostic	40.50 ±1.73	64.75 ±1.29	74.25 ±1.29	82.50 ±1.58	6.27 ±0.17	5.26 ±0.04	4.71 ±0.05	4.01 ±0.07	3.17 ±0.27	1.75 ±0.03	1.55 ±0.02	1.39 ±0.01
Multitask	38.75 ±0.96	66.00 ±0.82	77.50 ±1.29	87.50 ±0.58	13.80 ±3.36	6.80 ±0.26	5.18 ±1.01	2.91 ±0.49	6.39 ±1.96	1.98 ±0.12	1.64 ±0.05	1.44 ±0.02
Matching	**43.00** ±1.15	58.75 ±2.45	64.50 ±1.83	67.00 ±1.41		–				–		
SNAIL	**43.00** ±1.41	44.00 ±2.00	68.25 ±1.26	67.50 ±4.43	**2.48** ±0.38	3.80 ±0.38	4.98 ±0.03	4.11 ±2.94	2.63 ±0.27	1.95 ±0.25	3.47 ±1.56	3.04 ±1.01
MAML	39.60 ±0.55	63.40 ±0.89	71.80 ±0.84	78.80 ±0.84	6.73 ±0.16	5.84 ±0.2	5.19 ±0.08	4.20 ±0.10	5.49 ±0.85	2.05 ±0.03	1.65 ±0.01	1.44 ±0.01
CAVIA	**43.00** ±0.58	**78.00** ±1.26	87.00 ±0.50	91.00 ±0.58	11.05 ±2.94	**2.75** ±0.51	1.78 ±0.14	1.53 ±0.06	2.21 ±0.21	1.31 ±0.03	1.25 ±0.03	1.21 ±0.02
TAM	40.50 ±0.82	75.50 ±0.50	**89.50** ±0.58	**94.50** ±0.82	8.47 ±1.42	2.92 ±0.67	**1.47** ±0.18	**1.15** ±0.03	**1.82** ±0.08	**1.27** ±0.01	**1.22** ±0.01	**1.17** ±0.01

Table 7: k-shot sequence classification and sequence transduction experiments on our three benchmarks for $k \in \{1, 5, 10, 20\}$. The metric for sequence classification is average accuracy on test tasks (higher is better). On the transduction tasks, the performance metric is average perplexity on test tasks (lower is better). Random performance is at 25% accuracy (classification) and 12 perplexity points (other two tasks). Entries in smaller font are error bars, and they are estimated on 4 trials varying the model initialization.

Model	Sequence Classification				Sequence Transduction				Path Finding			
	1	5	10	20	1	5	10	20	1	5	10	20
Multitask	57.50 ±3.51	74.00 ±5.48	81.00 ±4.24	88.5 ±2.08	43.32 ±10.87	7.50 ±0.43	3.48 ±0.12	2.16 ±0.05	3.08 ±0.61	1.62 ±0.33	1.32 ±0.05	1.23 ±0.02
Matching	61.25 ±4.35	69.50 ±5.45	72.25 ±6.08	67.5 ±5.92		–				–		
SNAIL	**63.5** ±4.80	71.75 ±4.86	76.25 ±2.75	80.25 ±2.87	7.00 ±2.03	6.24 ±0.27	6.93 ±3.25	17.10 ±9.66	**1.53** ±0.29	1.71 ±0.09	3.36 ±0.57	4.22 ±0.79
CAVIA	57.25 ±12.09	66.25 ±14.73	67.50 ±15.67	68.50 ±16.42	36.72 ±8.83	6.01 ±0.88	3.99 ±0.50	3.27 ±0.24	1.99 ±0.11	1.27 ±0.01	1.21 ±0.00	1.17 ±0.00
TAM (Comp)	63.00 ±5.35	**76.50** ±4.65	**82.75** ±3.86	88.5 ±2.65	**6.15** ±0.91	**3.43** ±0.05	**2.69** ±0.04	**2.13** ±0.02	2.33 ±0.18	1.30 ±0.02	1.23 ±0.01	1.19 ±0.01
TAM (Non-comp)	45.25 ±3.59	72.5 ±3.70	81.5 ±2.89	**89.75** ±0.96	7.80 ±0.09	5.08 ±0.24	3.60 ±0.15	2.42 ±0.08	4.37 ±3.59	**1.27** ±0.01	**1.17** ±0.00	**1.11** ±0.00

Table 8: Compositional models for few-shot sequence classification and sequence transduction. All models (except non-compositional TAM) get information on the primitives present in the tasks via extra tokens appended to the input sequence, except that one such primitive is unseen at test time. Non-compositional TAM is not given information about primitives, and estimates a single task embedding instead.