DeepOSets: Non-Autoregressive In-Context Learning of Supervised Learning Operators

Shao-Ting Chiu
Dept of Electrical & Computer Engineering
Texas A&M University
College Station, TX, USA stchiu@tamu.edu

Junyuan Hong
Institute for Foundations of Machine Learning
University of Texas
Austin, TX, USA jyhong@utexas.edu

Ulisses Braga-Neto
Dept of Electrical & Computer Engineering
Texas A&M University
College Station, TX, USA ulisses@tamu.edu

Abstract

We introduce DeepSets Operator Networks (DeepOSets), an efficient, non-autoregressive neural network architecture for in-context operator learning. In-context learning allows a trained machine learning model to learn from a user prompt without further training. DeepOSets adds in-context learning capabilities to Deep Operator Networks (DeepONets) by combining it with the DeepSets architecture. As the first non-autoregressive model for in-context operator learning, DeepOSets allow the user prompt to be processed in parallel, leading to significant computational savings. Here, we present the application of DeepOSets in the problem of learning supervised learning algorithms, which are operators mapping a finite-dimensional space of labeled data into an infinite-dimensional hypothesis space of prediction functions. In an empirical comparison with a popular autoregressive (transformer-based) model for in-context learning of the least-squares linear regression algorithm, DeepOSets reduced the number of model weights by several orders of magnitude and required a fraction of training and inference time. Furthermore, DeepOSets proved to be less sensitive to noise, significantly outperforming the transformer model in noisy settings.

1 Introduction

In supervised learning [1], there are input and output spaces $\mathcal{X}$ and $\mathcal{Y}$, training data $D_n = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ in data space $(\mathcal{X} \times \mathcal{Y})^n$, and a hypothesis space $\mathcal{H}$ of functions $f : \mathcal{X} \to \mathcal{Y}$. A supervised learning algorithm is an *operator*

$$\Phi_n \; : \; (\mathcal{X} \times \mathcal{Y})^n \;\; \longrightarrow \;\; \mathcal{H} \tag{1}$$

between the data and hypothesis spaces. Given the training data D_n and an input x_{query}, this *supervised learning operator* produces a function $f_n = \Phi_n(D_n) \in \mathcal{H}$, such that $f_n(x_{query})$ predicts the output corresponding to x_{query}. Usually $\mathcal{X} \subseteq R^d$ and $\mathcal{Y} \in R$, in the regression case, or $\mathcal{Y} = \{0, 1, \ldots, c-1\}$, in the classification case. Hence, the data space is finite-dimensional, while the hypothesis space is infinite-dimensional.

Meta-learning, or "learning to learn,"[2, 3] is an idea that is present in key areas of machine learning, such as few-shot learning [4], multi-task learning [5], continuous learning [6], foundation models [7], and more. For our purposes, meta-learning is the problem of learning the best supervised learning

To appear at the "Foundation Models for Science" worshop at Neurips'2024.

algorithm for a given data domain. In view of the previous discussion, meta-learning is a problem of *operator learning* [8, 9], namely, learning the supervised learning operator from data.

In-context learning (ICL) refers to the ability of a trained machine learning model to learn from a user prompt *without further training* [10]. ICL has had tremendous success in the field of natural language processing using the transformer architecture [11]. The problem we consider in this paper, namely, ICL of functions in a hypothesis class, seems to have been first proposed in [12], which used recursive neural networks (RNN) and long-short term memory (LSTM) networks. Recently, an approach based on transformers has been studied [13, 14, 15, 16, 17, 18]. At both training and inference time, the aforementioned approaches require the data specified in the user prompt to be processed sequentially, one data point at a time, in an auto-regressive manner. However, in supervised learning the data set is typically permutation-invariant, and therefore an auto-regressive architecture such as the transformer is unnecessary and computationally wasteful.

In this paper, we propose an efficient alternative that processes the data in parallel. Our approach combines the DeepSets [19] and DeepONet [9] neural architectures. DeepONet is a powerful universal approximator of continuous operators between function spaces, which consists of separate branch and trunk networks; in DeepOSets, the training data set in the prompt becomes the input to the branch network, while the query point is the input of the trunk network. However, this requires a modification to the original design of DeepONet, since in ICL, the prompt may be of variable size. To address this, we employ the popular Deep Sets architecture for set learning. Adding a Deep Sets module to the branch network of the DeepONet allows the model to accept a varying number of in-context examples. The Deep Sets module also introduces a permutation-invariance inductive bias to improve generalizability.

The resulting DeepOSets architecture allows a significant reduction in the number of required parameters and requires a small fraction of training time compared to transformer-based approaches, represented here by the well-known method in [13]. This makes DeepOSets a promising candidate for resource-constrained environments. We remark that this parallels the emmergence of non-autoregreesive alternatives to expensive transformers in natural language processing [20, 21].

To improve generalization performance further, we use k-ary Janossy pooling [22] of the ICL examples (Section 2.5). In the baseline case $k = 1$, DeepOSets has linear complexity $O(n)$ in the number of examples at training time. At inference time, the complexity is constant, for any value of k, after the first query is processed (Section 2.7). In contrast, the transformer model incurs quadratic complexity, even at inference time, due to the need to compute attention matrices for each new query (Section 3.6). These factors help explain why the transformer model is drastically slower thand DeepOSets (Table 2).

We present experimental results with ICL for least-squares linear regression that show that DeepOSets can efficiently learn from in-context examples. In one of our experiments (Table 2), the baseline DeepOSets model with 72K parameters displayed a test error 10 times smaller than that of a popular transformer-based approach [13] with 22 million parameters, a reduction in parameter size of almost four orders of magnitude. In addition, DeepOSets exhibited much faster convergence, completing training in less than 10 minutes, whereas the transformer approach took at least 4 hours to reach comparable results. Inference time was also much faster, with DeepOSets taking 0.087 ms per query, while the transformer model took 7.11 ms. The performance of DeepOSets was less sensitive to the presence of noise, so that DeepOSets became much more accurate than the transformer in moderate to high noise cases (at a smaller number of weights and faster training). This robustness to noise is a significant advantage of DeepOSets over transformer models (Section 3.5).

Main Contributions

- We propose DeepOSets, a non-autoregressive approach to in-context operator learning.
- Our approach combines the powerful DeepSets and DeepONet architectures to allow the model to generalize across different numbers of in-context examples and their permutations.
- We demonstrate empirically that DeepOSets can learn classical least-squares linear regression much more efficiently than a comparable transformer-based approach; it is parameter efficient, it is faster to train, and it is more robust to noise (making it more accurate than the transformer under noisy conditions).

2 Methodology

2.1 In-Context Learning of Supervised Learning Operators

Without loss of generality, let us consider from now on that $\mathcal{X} \subseteq R^d$. We will use boldface to represent elements (vectors) in $\mathcal{X}$. Here, as in [13], we will use synthetic data to train our model, in which case we assume noiseless training data $D_n = \{(\mathbf{x}_1, f(\mathbf{x}_1)), \ldots, (\mathbf{x}_n, f(\mathbf{x}_n))\}$, with a randomly picked function $f : \mathcal{X} \to \mathcal{Y}$ in the given hypothesis space.

The ICL prompt for the supervised learning operator model at training time is

$$\overbrace{\underbrace{\mathbf{x}_1, f(\mathbf{x}_1)}_{\text{Example}_1}, \underbrace{\mathbf{x}_2, f(\mathbf{x}_2)}_{\text{Example}_2}, \ldots, \underbrace{\mathbf{x}_n, f(\mathbf{x}_n)}_{\text{Example}_n}}^{\text{In-Context Examples}}, \mathbf{x}_{query}}^{\text{Prompt}} . \tag{2}$$

At inference time, the objective is to use a given user prompt consisting of a (possibly noisy) training dataset $D_n = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ and a test point $\mathbf{x}_{query}$ to infer $f(\mathbf{x}_{query})$, without any further training, by employing a trained neural operator:

$$\Phi\left(D_n; \mathbf{w}\right)\left(\mathbf{x}_{query}\right) \;\approx\; f(\mathbf{x}_{query}) \,, \tag{3}$$

where the model weights $\mathbf{w}$ are adjusted during training using a sample of prompt examples and target functions, and remain fixed during inference.

Note that the target function f *is not fixed* as in traditional supervised learning but varies both at training and inference time. In addition, the ICL supervised learning operator must be able to handle in-context examples of varying length n. Together with the fact that the problem is permutation-invariant to the arrangement of in-context examples, this implies that this form of ICL is an example of *set learning*. Furthermore, even though the prompt can be handled in a sequential manner using an auto-regressive transformer model, there is no such requirement in supervised learning, and the prompt can be processed in parallel in a more efficient manner, which is the approach we adopt here.

2.2 DeepONets

DeepONet is a neural network architecture for learning operators between function spaces [9], which possesses a universal approximation guarantee [23]. A DeepONet consists of a branch network and a trunk network (see the right side of Fig. 1). In our use case, the branch network takes the training data D_n and encodes this information into a feature vector $b_1, \ldots, b_p$. The trunk network, on the other hand, takes the input $\mathbf{x}_{query}$ at which the output function $\Phi(D_n)$ is to be evaluated, where Φ is a supervised learning operator, and computes a feature vector $t_1, \ldots, t_p$. The final output of the DeepONet is obtained by taking the dot product of feature vectors from the branch and trunk networks. The approximation can be written as:

$$\Phi(D_n; \mathbf{w})(\mathbf{x}_{query}) \;=\; \sum_{i=1}^{p} b_i(D_n; \mathbf{w}_{br}) \cdot t_i(\mathbf{x}_{query}; \mathbf{w}_{tr}) + b_0 \,. \tag{4}$$

where $\mathbf{w} = \{\mathbf{w}_{br}, \mathbf{w}_{tr}, b_0\}$ comprise the neural network weights. This can be thought as a basis expansion approximation, where the trunk network computes adaptive basis functions, and the branch network computes the expansion coefficients. The bias term b_0, while not strictly necessary, often enhances performance.

2.3 Modifying DeepONets with DeepSets for In-Context Learning

Despite the success of DeepONets in operator learning, it is not appropriate for our purposes, due to the following two major challenges:

- The branch network accepts a fixed number of inputs, and thus it cannot accomodate a varying number of in-context examples.
- The branch network is not invariant to permutations of its input, which can lead to poor generalization since in the majority of cases the supervised learning operator to be learned is indeed invariant to permutations in the training data.

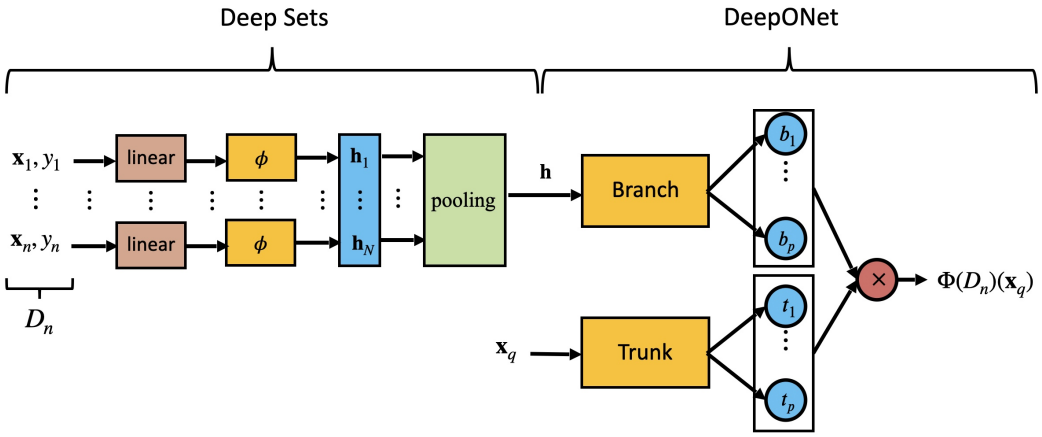


Figure 1: DeepOSets architecture for in-context learning of supervised learning operators.

To address these challenges, we modify the branch network using a *set learning* approaching, which allows the DeepONet to process an indefinite number of in-context examples in a permutation-invariant setting. Due to its efficiency, we choose the DeepSets paradigm [24] for set learning. DeepSets is designed to accommodate point clouds of varying sizes, providing flexibility in handling in-context examples. DeepSets has been shown to possess a universal approximation property for permutation-invariant functions [25].

2.4 The DeepOSets Architecture

The DeepOSets architecture is displayed in Fig. 1. Each in-context example $(\mathbf{x}_i, f(\mathbf{x}_i)) \in \mathbb{R}^{d+1}$ is embedded into a higher-dimensional space by means of a trainable linear layer, and then encoded into a feature vector $\mathbf{h}_i \in \mathbb{R}^{d_{embed}}$ by means of an MLP ϕ. The encoded vectors $\mathbf{h}_i$ are pooled into a single vector $\mathbf{h}$ to obtain permutation invariance. The simplest pooling method is to average across all features: $\mathbf{h} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{h}_i$. The aggregated feature vector $\mathbf{h}$ provides the input to the branch network of a DeepONet. Notice that the weights in the embedding layer and MLP ϕ are shared by all inputs. The permutation invariance property of DeepOSets with respect to the in-context examples is an inductive bias that can improve the model generalization ability.

2.5 Janossy Pooling

A variation of the baseline DeepOSet architecture in Fig. 1 is afforded by *Janossy pooling* [22]. The modification occurs at the prompt input, which considers all possible $n!/k!(n-k)!$ k-tuple subsets of the n examples. Each k-tuple passes through an MLP ϕ and the outputs are pooled as before. This can improve accuracy by allowing interactions between input examples to be modeled. The case $k = 1$ correspond to the baseline case, while the case $k = 2$, known as binary Janossy pooling, resembles self-attention [11] in that the interaction between pairs of inputs are modeled. In general, the complexity of Janossy pooling is $O(n^k)$, which limits k to a small value. In our experimental results, we consider the cases $k = 1$ and $k = 2$.

2.6 Training

To train our model, we randomly sample functions and prompts as described previously. The training loss to be minimized is the mean squared error (MSE) between the model outputs and the true function values:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n}\sum_{j=1}^{n}[\Phi(D_n;\mathbf{w})(\mathbf{x}_{query_j}) - f(\mathbf{x}_{query_j})]^2 \tag{5}$$

2.7 Training and Inference Complexity

In the baseline case $k = 1$, DeepOSets has linear training complexity $O(n)$ in the number of in-context examples. At inference time, prediction on a new query point given n fixed in-context examples has constant complexity $O(1)$. On the other hand, the transformer necessitates the computation of the attention matrices for the entire prompt sequence for each new query, resulting in a $O(n^2)$ complexity for predicting x_{query} from n in-context examples [26] (Table 2). Since x_{query} is a part of the prompt, each new x_{query} necessitates new attention matrices, even if the rest of the prompt elements remain the same. In other words, computation of the output for each new query requires has quadratic complexity $O(n^2)$ in the number of in-context examples. Hence, in-context learning with a transformer is "memoryless," necessitating decoding the entire prompt for every new x_{query}. In contrast, DeepOSets can process n examples in $O(n)$, and inference time after the first x_{query} is independent of the number of in-context examples.

3 Results

3.1 Implementation

All DeepOSets experiments were implemented in JAX [27] and Equinox [28]. Experiments involving the transformer model in [13] were implemented in PyTorch [29]. All experiments were run on an Nvidia RTX 4060 Ti 16GB GPU.

3.2 Hypothesis Space and Data Generation

In our experiments, the hypothesis space $\mathcal{H}$ consists of linear functions

$$\mathcal{H} = \{f \mid f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}, \mathbf{w} \in \mathbb{R}^d\}, \tag{6}$$

where $\mathbf{w}$ is randomly generated during training. Following [13], we let $\mathbf{w} \sim \mathcal{N}(0, \mathbf{I}_d)$. Given a sample function f thus generated, we again follow [13] and generate each context example $\mathbf{x}_i$ from $N(0, I_d)$ and obtain the corresponding target $f(\mathbf{x}_i)$. The query point is sampled from the same distribution as the in-context examples. During training, the function values are assumed to be noiseless, but at inference time, we consider the more realistic case where the function values in the prompt are corrupted with additive Gaussian noise.

3.3 Hyperparameter Setting

For $d = 1$, we linearly embed $\mathbf{x}$ and $f(\mathbf{x})$ into 5-dimensional space. Then, the embedded examples are processed by a 6-layer MLP with 50 hidden units into a vector of size 400. Both branch and trunk nets contain 5 layers with 40 hidden units. The last layer of DeepOnet contains 100 units. For $d = 5$, we linearly embed $\mathbf{x}$ and $f(\mathbf{x})$ (the latter is appended with zeros to have the same dimension as $\mathbf{x}$) into 15-dimensional space. The embedded examples are processed by a single MLP with 2 hidden layers and 200 hidden units into a vector of size 800. Both branch and trunk are MLPs with 6 hidden layers and 200 hidden units. The last layer of the branch and trunk networks contain $p = 200$ neurons. This resulted in a total number of trainable parameters equal to 72K and 0.57M for the $d = 1$ and $d = 5$ cases, respectively (Table 1). Following [24], we employed the SELU activation function [30] in the DeepSets module. For the trunk network, the tanh nonlinearity is used. Training employs the Adam optimizer [31] with a learning rate of 1e-3 and exponential decay by 0.9 every 2000 steps.

3.4 DeepOSets Learns Linear Regression from In-Context Data

Fig. 2 displays several examples of training data sets and corresponding linear regression obtained by the baseline DeepOSets ($k = 1$) in the case $d = 1$. The model was trained on noiseless prompts of size $n = 13$. Training converged in 9 minutes with 16K iterations and reached training mean square error 7.9e-4. At inference time, the model was challenged with prompt examples corrupted by noise: $\tilde{f}(\mathbf{x}) = \mathbf{a}^T\mathbf{x} + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$, where $\sigma^2 = 0.1$. In addition, the sample size at inference time is $n = 10$, thus different than the sample size used for training. We can see in Fig. 2 that the trained DeepOSets model accurately recovers the ground-truth function. For comparison, the regression with ordinary least-square regression is also displayed.

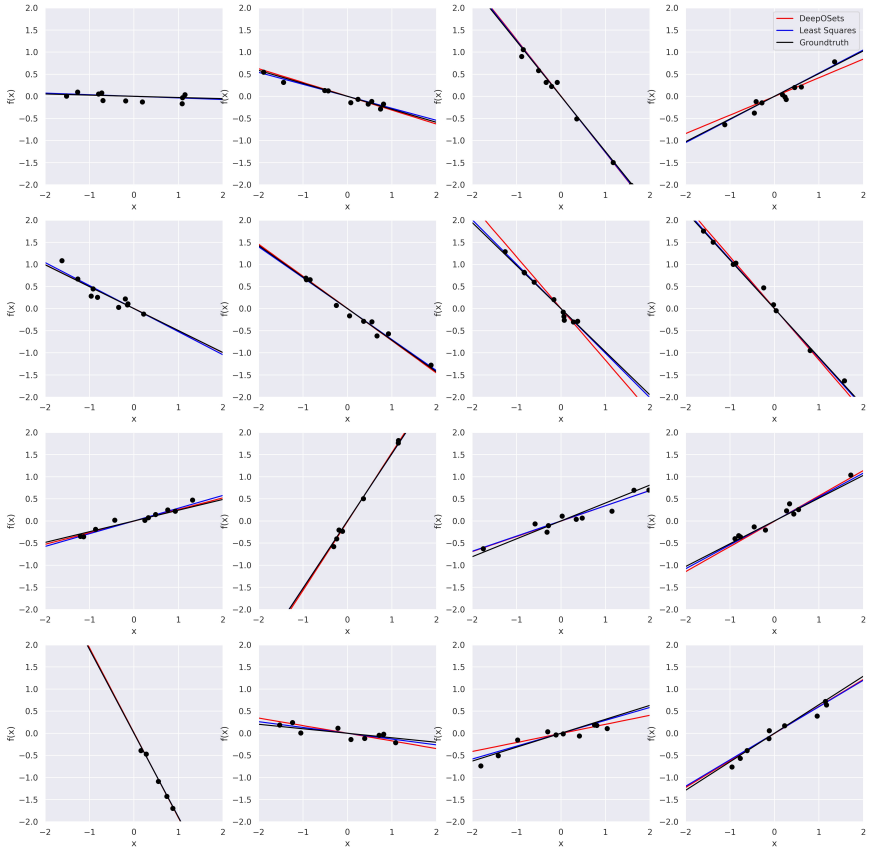


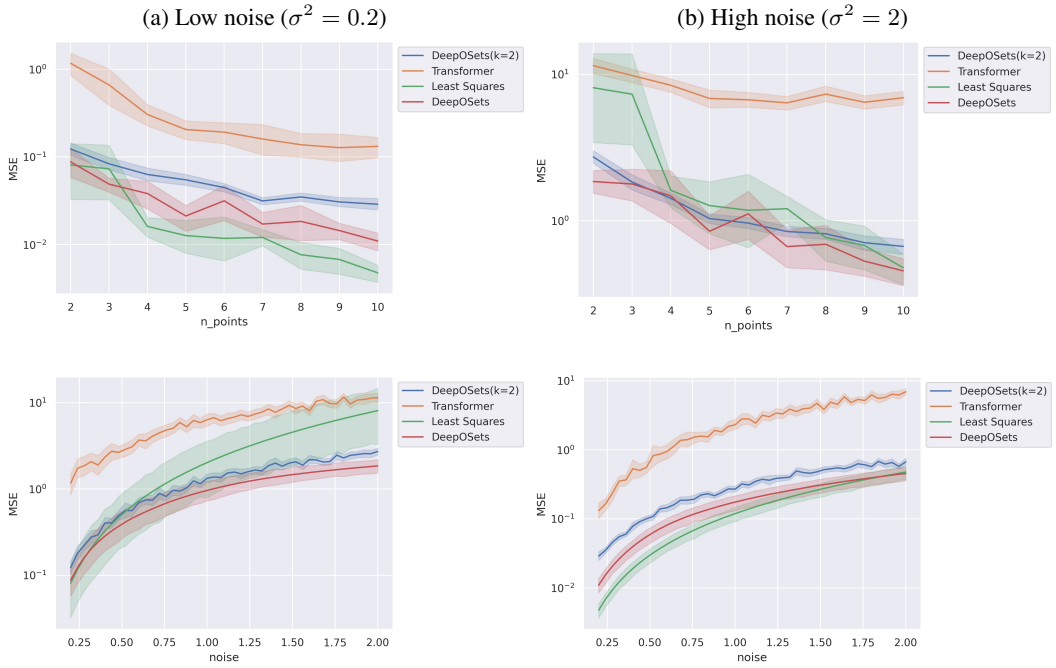
Figure 2: Learning linear regression ($d = 1$, $k = 1$, $n = 13$, and $\sigma^2 = 0.1$) with DeepOSets where n is the number of in-context examples in training set. The black dots (•) represent 10 in-context examples corrupted by Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma^2 = 0.1)$. The blue line (—) represents ordinary least squares regression, while the red line (—) corresponds to DeepOSets.

3.5 DeepOSets is Accurate and Robust to Noise

We further investigate the prediction accuracy of DeepOSets in low ($d = 1$) and high ($d = 5$) dimensions, with noise ranging from low to high ($\sigma^2 \in [0.25, 2.0]$), and varying number of in-context examples (note that well-posedness of the linear regression problem requires at least $d + 1$ examples).

Accuracy As expected, the results show that the prediction becomes more accurate as the number of training examples increases and the noise decreases in both low and high dimensions (Figs. 3 and 4, respectively). In the case $d = 1$, we can see that the DeepOSets variants ($k = 1$ and $k = 2$) have similar accuracy, and both outperform the transformer method uniformly across the different sample sizes and noise intensities. In the case $d = 5$, this is still true in the case of larger noise intensity, but in the case of low noise intensity, DeepOSets with $k = 2$ performs similarly to the transformer, and both outperform the baseline DeepOSets with $k = 1$ (see also Table 1).

Robustness We can also see in Figs. 3 and 4 that the performance of DeepOSets is less sensitive to the increase in noise intensity (i.e., it maintains consistent performance across different noise levels)



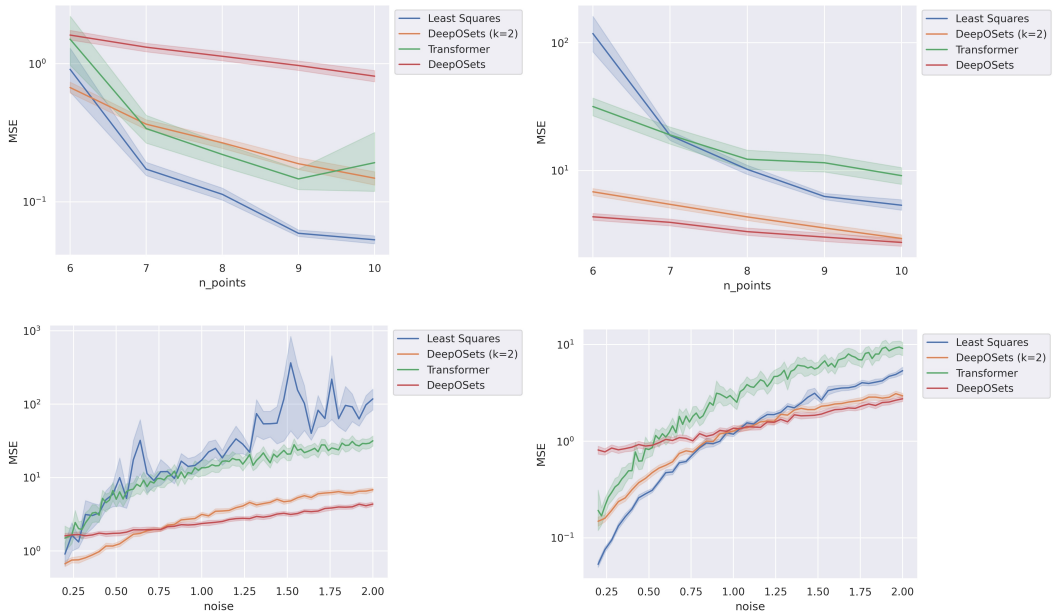
(a) Low noise ($\sigma^2 = 0.2$)

(b) High noise ($\sigma^2 = 2$)

(c) 2 in-context training examples.

(d) 10 in-context training examples.

Figure 3: Evaluating the trained DeepOSets on 1-dimensional linear regression with different noise scales and number of in-context examples.



(a) Low noise ($\sigma^2 = 0.2$)

(b) High noise ($\sigma^2 = 2$)

(c) 6 in-context examples.

(d) 10 in-context examples.

Figure 4: Evaluating the trained DeepOSets on 5-dimensional linear regression with different noise scales and a number of in-context examples. DeepOSets (k=2) represents DeepOSets with k-ary Janossy Pooling[22].

than the transformer and even least-squares regression, which helps explain why DeepOSets is more accurate in high noise settings.

Regression Problems	**Transformer**[13]	**DeepOSets**	**DeepOSets (k=2)**
1 dimension			
Linear Regression (d=1, n=6, $\sigma^2 = 0.0$)	6.744e-04	**1.548e-04**	2.710e-03
Linear Regression (d=1, n=6, $\sigma^2 = 0.04$)	6.454e-03	**1.450e-03**	4.759e-03
Linear Regression (d=1, n=6, $\sigma^2 = 0.2$)	1.917e-01	**3.158e-02**	4.479e-02
Linear Regression (d=1, n=6, $\sigma^2 = 2.0$)	6.692	1.113	**9.622e-01**
# of Parameters	22M	**71151**	84296
5 dimensions			
Linear Regression (d=5, n=6, $\sigma^2 = 0.0$)	**7.509e-02**	1.642	5.221e-01
Linear Regression (d=5, n=6, $\sigma^2 = 0.04$)	**1.379e-01**	1.611	5.462e-01
Linear Regression (d=5, n=6, $\sigma^2 = 0.2$)	1.502	1.609	**6.716e-01**
Linear Regression (d=5, n=6, $\sigma^2 = 2.0$)	3.172e+01	**4.361**	6.837
# of Parameters	22M	**568561**	1942411

Table 1: Comparison between GPT and DeepOSets in Mean Square Error (MSE) and number of parameters. Each experiment includes 30 test functions and 6 in-context examples.

3.6 DeepOSets is Lighter and Faster than Transformers

Model size We can see in Table 1 that, in the case $d = 1$, superior results are achieved by DeepOSets at a small fraction of the number of parameters in the Transformer model (the DeepOSets models are around 300 times smaller). In the case $d = 5$, comparable or superior results are obtained with DeepOSets that are still an order of magnitude smaller than the transformer.

Training speed We conducted a comparison of the performance and model size between the transformer[13] and DeepOSets in linear regression with $d = 1$, $n = 10$, σ^2=0.2[1]. Given that the transformer has approximately 300 times more parameters than DeepOSets, it requires 3 hours for training. In contrast, DeepOSets only needs 9 minutes of training for the same regression problem and demonstrates superior generalization with noisy prompts (see Table 2). Unlike the transformer-based auto-regressive model, DeepOSets does not require additional parameters and computation for the attention mechanism.

Inference speed DeepOSets efficiently predicts large numbers of queries x_{query} and exhibits constant memory and complexity regardless of the number of in-context examples. In contrast, the transformer exhibits quadratic complexity in the number of examples [26], as it necessitates the recalculation of the attention matrices whenever x_{query} changes in the prompt. Fig. 5 confirms the quadratic complexity of the transformer model. On the other hand, the inference time of DeepOSets models only marginally increases with more in-context examples, showcasing the decisive advantage of set learning over the auto-regressive approach for in-context learning.

	Transformer[13]	DeepOSets	DeepOSets (k=2)
Parameters	22M	**72K**	84K
Training time	3 hours	9 min	**8min**
Complexity for first x_{query} on n examples	$O(n^2)$	$\boldsymbol{O(n)}$	$O(n^2)$
Complexity for second x_{query} on n examples	$O(n^2)$	$\boldsymbol{O(1)}$	$\boldsymbol{O(1)}$
Memory complexity for n examples	$O(n^2)$	$\boldsymbol{O(1)}$	$\boldsymbol{O(1)}$
Inference time per query ($n = 10$)	7.11 ms	**0.087 ms**	0.18ms
Test MSE	0.132	**1.12e-2**	2.89e-2

Table 2: Benchmark with $d = 1$, $n = 13$, and $\sigma^2 = 0.2$ where n is the training sample size for DeepOSets.

[1]Transformer experiments obtained from `https://github.com/dtsip/in-context-learning`

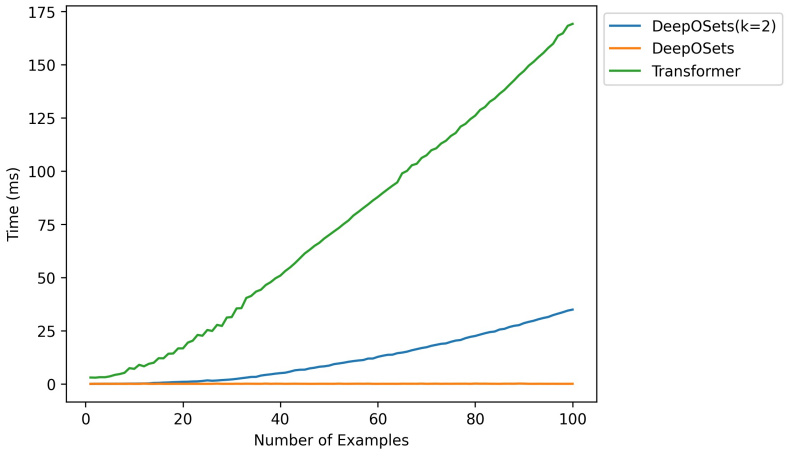


Figure 5: Effect of the number of in-context examples on inference time. The plot shows the prediction time of the first x_{query} as a function of the number of in-context examples from 1 to 100. The subplot displays the inference time for DeepOSets in log scale.

4 Discussion

The experimental results presented here have unveiled the potential of DeepOSets as a more efficient and noise-robust alternative to autoregressive models, here represented by the transformer-based model in [13], for in-context learning of supervised learning operators. DeepOSets requires fewer parameters, trains faster, and has faster inference time than the transformer model. Several key findings regarding the performance and behavior of DeepOSets were observed. The DeepSets module effectively generalizes in-context examples, enabling DeepOSets to maintain consistent performance with noisy prompts of varying size not encountered during training. Note that the DeepOSets model is trained with noiseless data, and the ability to predict noisy prompts is meta-learned.

The significant advantage of DeepOSets over autoregressive approaches is the linear time and memory complexity in inference (Table 2). DeepOSets has linear time complexity in the number of in-context examples. Once the in-context examples are processed and fixed, the prediction on new x_{query} takes constant time, while autoregressive approaches, such as the transformer, requires obtaining attention matrices for every new prompt, even if the rest of the prompt is fixed. Furthermore, our experiments demonstrated that DeepOSets performs more accurately than the transformer approach of [13] in univariate and/or noisy problems, while being competitive (with $k = 2$) in high-dimensional ($d = 5$), low-noise problems (while still being much faster in all cases).

A limitation of the approach is the difficulty of learning high-dimensional problems, which was already observed in vanilla DeepONets [32, 33]. This limitation can be improved by a two-step training method that trains branch and trunk separately [34]. Alternatively, multiple-input operators [35], dimension reduction approaches [36, 37], and dimension separation [38] are all possible future work to conquer the curse of dimensionality.

Finally, we note that DeepOSets can be extended to learning *multiple* supervised learning operators; this would be the case when the algorithm contains hyperparameters, such as the regularization parameter in ridge regression, the order of polynomial regression, or the number of neighbors in a nearest-neighbor regression. A DeepOSets approach to *automatic machine learning* (AutoML) would train the model for a range of hyperparameters and be able to automatically select the hyperparameter for a new prompt. We also note that the DeepOSets architecture can be extended to handle operator learning for PDEs, where the in-context examples consist of pairs of boundary conditions or coefficient functions and solutions. These extensions will be considered in future work.

Acknowledgements

Chiu and Braga-Neto were supported by NSF Award CCF-2225507.

References

[1] L. Devroye, L. Gyorfi, and G. Lugosi, *A Probabilistic Theory of Pattern Recognition*. New York: Springer, 1996.

[2] J. Schmidhuber, *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

[3] J. Schmidhuber, "A neural network that embeds its own meta-levels," in *IEEE International Conference on Neural Networks*, pp. 407–412, IEEE, 1993.

[4] Z. Li, F. Zhou, F. Chen, and H. Li, "Meta-sgd: Learning to learn quickly for few-shot learning," *arXiv preprint arXiv:1707.09835*, 2017.

[5] M. Crawshaw, "Multi-task learning with deep neural networks: A survey," *arXiv preprint arXiv:2009.09796*, 2020.

[6] K. Javed and M. White, "Meta-learning representations for continual learning," *Advances in neural information processing systems*, vol. 32, 2019.

[7] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[8] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Neural operator: Graph kernel network for partial differential equations," *arXiv preprint arXiv:2003.03485*, 2020.

[9] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators," *Nature Machine Intelligence*, vol. 3, pp. 218–229, Mar. 2021.

[10] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, vol. 1, 2020.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, vol. 10, p. S0140525X16001837, 2017.

[12] S. Hochreiter, A. S. Younger, and P. R. Conwell, "Learning to learn using gradient descent," in *Artificial Neural NetworksICANN 2001: International Conference Vienna, Austria, August 21–25, 2001 Proceedings 11*, pp. 87–94, Springer, 2001.

[13] S. Garg, D. Tsipras, P. Liang, and G. Valiant, "What Can Transformers Learn In-Context? A Case Study of Simple Function Classes," Aug. 2023.

[14] R. Zhang, S. Frei, and P. L. Bartlett, "Trained transformers learn linear models in-context," *Journal of Machine Learning Research*, vol. 25, no. 49, pp. 1–55, 2024.

[15] Y. Bai, F. Chen, H. Wang, C. Xiong, and S. Mei, "Transformers as statisticians: Provable in-context learning with in-context algorithm selection," *Advances in neural information processing systems*, vol. 36, 2024.

[16] Y. Xing, X. Lin, N. Suh, Q. Song, and G. Cheng, "Benefits of transformer: In-context learning in linear regression tasks with unstructured data," *arXiv preprint arXiv:2402.00743*, 2024.

[17] J. W. Liu, J. Grogan, O. M. Dugan, S. Arora, A. Rudra, and C. Re, "Can transformers solve least squares to high precision?," in *ICML 2024 Workshop on In-Context Learning*, 2024.

[18] J. Von Oswald, E. Niklasson, E. Randazzo, J. Sacramento, A. Mordvintsev, A. Zhmoginov, and M. Vladymyrov, "Transformers learn in-context by gradient descent," in *International Conference on Machine Learning*, pp. 35151–35174, PMLR, 2023.

[19] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," *Advances in neural information processing systems*, vol. 30, 2017.

[20] J. Gu, J. Bradbury, C. Xiong, V. O. Li, and R. Socher, "Non-autoregressive neural machine translation," *arXiv preprint arXiv:1711.02281*, 2017.

[21] Y. Xiao, L. Wu, J. Guo, J. Li, M. Zhang, T. Qin, and T.-y. Liu, "A survey on non-autoregressive generation for neural machine translation and beyond," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 10, pp. 11407–11427, 2023.

[22] R. L. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro, "Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs," *arXiv preprint arXiv:1811.01900*, 2018.

[23] T. Chen and H. Chen, "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems," *IEEE transactions on neural networks*, vol. 6, no. 4, pp. 911–917, 1995.

[24] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep Sets," in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017.

[25] E. Wagstaff, F. B. Fuchs, M. Engelcke, M. A. Osborne, and I. Posner, "Universal approximation of functions on sets," *Journal of Machine Learning Research*, vol. 23, no. 151, pp. 1–56, 2022.

[26] F. D. Keles, P. M. Wijewardena, and C. Hegde, "On the computational complexity of self-attention," in *International Conference on Algorithmic Learning Theory*, pp. 597–619, PMLR, 2023.

[27] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018.

[28] P. Kidger and C. Garcia, "Equinox: neural networks in JAX via callable PyTrees and filtered transformations," *Differentiable Programming workshop at Neural Information Processing Systems 2021*, 2021.

[29] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[30] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," *Advances in neural information processing systems*, vol. 30, 2017.

[31] D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[32] L. Mandl, S. Goswami, L. Lambers, and T. Ricken, "Separable deeponet: Breaking the curse of dimensionality in physics-informed machine learning," *arXiv preprint arXiv:2407.15887*, 2024.

[33] A. Peyvan, V. Oommen, A. D. Jagtap, and G. E. Karniadakis, "Riemannonets: Interpretable neural operators for riemann problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 426, p. 116996, 2024.

[34] S. Lee and Y. Shin, "On the training and generalization of deep operator networks," *SIAM Journal on Scientific Computing*, vol. 46, no. 4, pp. C273–C296, 2024.

[35] P. Jin, S. Meng, and L. Lu, "MIONet: Learning multiple-input operators via tensor product," Feb. 2022.

[36] S. Lanthaler, "Operator learning with pca-net: upper and lower complexity bounds," *Journal of Machine Learning Research*, vol. 24, no. 318, pp. 1–67, 2023.

[37] K. Kontolati, S. Goswami, G. E. Karniadakis, and M. D. Shields, "Learning in latent spaces improves the predictive accuracy of deep neural operators," *arXiv preprint arXiv:2304.07599*, 2023.

[38] X. Yu, S. Hooten, Z. Liu, Y. Zhao, M. Fiorentino, T. Van Vaerenbergh, and Z. Zhang, "Separable operator networks," *arXiv preprint arXiv:2407.11253*, 2024.