On the adaptation of in-context learners for system identification

Marco Forgione, Filippo Pura, Dario Piga

IDSIA Dalle Molle Institute for Artificial Intelligence SUPSI-USI, Lugano, Switzerland

SYSID 2024
17-18 July 2024, Boston, USA

Standard system identification/supervised machine learning

1. Collect dataset $\mathcal{D} = (u_{1:N}, y_{1:N})$ of input/outputs from system S.
2. Apply an algorithm to estimate a model $M(\hat{\theta})$ of S:

$$\hat{\theta} = \mathcal{A}(\mathcal{D}) \qquad \text{e.g. } \mathcal{A}(\mathcal{D}) = \arg\min_{\theta \in \Theta} \mathcal{L}(\mathcal{D}, M(\theta))$$

3. Make predictions/simulations using the model on new data:

$$\hat{y}^*_{1:M} = M(u^*_{1:M}; \hat{\theta})$$

Researchers keep on improving learning algorithms and model structures. Can we automate this process? Can we learn the learning algorithm itself?

Meta learning tries to answer this question.

J. Schmidhuber. Evolutionary principles in self-referential learning, or on learning how to learn. *Diploma Thesis, TU Munich, 1987*

T. Hospedales et al. Meta-Learning in Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 2022*

Standard system identification/supervised machine learning

1. Collect dataset $\mathcal{D} = (u_{1:N}, y_{1:N})$ of input/outputs from system S.
2. Apply an algorithm to estimate a model $M(\hat{\theta})$ of S:

$$\hat{\theta} = \mathcal{A}(\mathcal{D}) \qquad \text{e.g. } \mathcal{A}(\mathcal{D}) = \arg\min_{\theta \in \Theta} \mathcal{L}(\mathcal{D}, M(\theta))$$

3. Make predictions/simulations using the model on new data:

$$\hat{y}^*_{1:M} = M(u^*_{1:M}; \hat{\theta})$$

Researchers keep on improving learning algorithms and model structures. Can we automate this process? Can we learn the learning algorithm itself?

Meta learning tries to answer this question.

J. Schmidhuber. Evolutionary principles in self-referential learning, or on learning how to learn. *Diploma Thesis, TU Munich, 1987*

T. Hospedales et al. Meta-Learning in Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 2022*

Standard system identification/supervised machine learning

1. Collect dataset $\mathcal{D} = (u_{1:N}, y_{1:N})$ of input/outputs from system S.
2. Apply an algorithm to estimate a model $M(\hat{\theta})$ of S:

$$\hat{\theta} = \mathcal{A}(\mathcal{D}) \qquad \text{e.g. } \mathcal{A}(\mathcal{D}) = \arg\min_{\theta \in \Theta} \mathcal{L}(\mathcal{D}, M(\theta))$$

3. Make predictions/simulations using the model on new data:

$$\hat{y}^*_{1:M} = M(u^*_{1:M}; \hat{\theta})$$

Researchers keep on improving learning algorithms and model structures. Can we automate this process? Can we learn the learning algorithm itself?

Meta learning tries to answer this question.

J. Schmidhuber. Evolutionary principles in self-referential learning, or on learning how to learn. *Diploma Thesis, TU Munich, 1987*

T. Hospedales et al. Meta-Learning in Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 2022*

Our meta learning setting

- We have an infinite stream of datasets from a distribution $p(\mathcal{D})$:

$$\{\mathcal{D}^{(i)} = (u_{1:N}^{(i)}, y_{1:N}^{(i)}),\ i = 1, 2, \dots, \infty\}$$

- $\mathcal{D}^{(i)}$ generated by random system $S^{(i)}$ and input realization $u_{1:N}^{(i)}$
- Different but related to each other. There's a learnable structure!

Can we get better at identifying $S^{(i)}$ as we observe more datasets $\mathcal{D}^{(j)}$?

- $p(\mathcal{D})$ may be a physical simulator where we can change settings
- The learned algorithm could then be applied to real data

Meta learning from a finite collection would also be interesting...

Our meta learning setting

- We have an infinite stream of datasets from a distribution $p(\mathcal{D})$:

$$\{\mathcal{D}^{(i)} = (u_{1:N}^{(i)}, y_{1:N}^{(i)}),\ i = 1, 2, \dots, \infty\}$$

- $\mathcal{D}^{(i)}$ generated by random system $S^{(i)}$ and input realization $u_{1:N}^{(i)}$
- Different but related to each other. There's a learnable structure!

Can we get better at identifying $S^{(i)}$ as we observe more datasets $\mathcal{D}^{(j)}$?

- $p(\mathcal{D})$ may be a physical simulator where we can change settings
- The learned algorithm could then be applied to real data

Meta learning from a finite collection would also be interesting...

Our meta learning setting

- We have an infinite stream of datasets from a distribution $p(\mathcal{D})$:

$$\{\mathcal{D}^{(i)} = (u_{1:N}^{(i)}, y_{1:N}^{(i)}),\ i = 1, 2, \dots, \infty\}$$

- $\mathcal{D}^{(i)}$ generated by random system $S^{(i)}$ and input realization $u_{1:N}^{(i)}$
- Different but related to each other. There's a learnable structure!

Can we get better at identifying $S^{(i)}$ as we observe more datasets $\mathcal{D}^{(j)}$?

- $p(\mathcal{D})$ may be a physical simulator where we can change settings
- The learned algorithm could then be applied to real data

Meta learning from a finite collection would also be interesting...

Our meta learning setting

- We have an infinite stream of datasets from a distribution $p(\mathcal{D})$:

$$\{\mathcal{D}^{(i)} = (u_{1:N}^{(i)}, y_{1:N}^{(i)}),\ i = 1, 2, \dots, \infty\}$$

- $\mathcal{D}^{(i)}$ generated by random system $S^{(i)}$ and input realization $u_{1:N}^{(i)}$
- Different but related to each other. There's a learnable structure!

Can we get better at identifying $S^{(i)}$ as we observe more datasets $\mathcal{D}^{(j)}$?

- $p(\mathcal{D})$ may be a physical simulator where we can change settings
- The learned algorithm could then be applied to real data

Meta learning from a finite collection would also be interesting...

In-context learning

Many meta learning strategies around. Here focus on in-context learning.

- Transformers are expressive as a programming language
- We make Transformers behave like algorithms. We provide:
  - A context, namely an input/output sequence of a system
  - A task, like predicting the next output or simulating for more steps
- The Transformer must learn to identify the system to solve the task!

Context + task may be seen as a prompt to a Large Language Model, which can then continue the word sequence in an optimal way.

G. Weiss et al. Thinking like Transformers. *38th International Conference on Machine Learning (ICML 2022).*

S. Garg et al. What Can Transformers Learn In-Context? A Case Study of Simple Function Classes. *36th Conference on Neural Information Processing Systems (NeurIPS 2022).*

T. Brown et al. Language Models are Few-Shot Learners. *34th Conference on Neural Information Processing Systems (NeurIPS 2020).*

In-context learning

Many meta learning strategies around. Here focus on in-context learning.

- Transformers are expressive as a programming language
- We make Transformers behave like algorithms. We provide:
  - A context, namely an input/output sequence of a system
  - A task, like predicting the next output or simulating for more steps
- The Transformer must learn to identify the system to solve the task!

Context + task may be seen as a prompt to a Large Language Model, which can then continue the word sequence in an optimal way.

G. Weiss et al. Thinking like Transformers. *38th International Conference on Machine Learning (ICML 2022).*

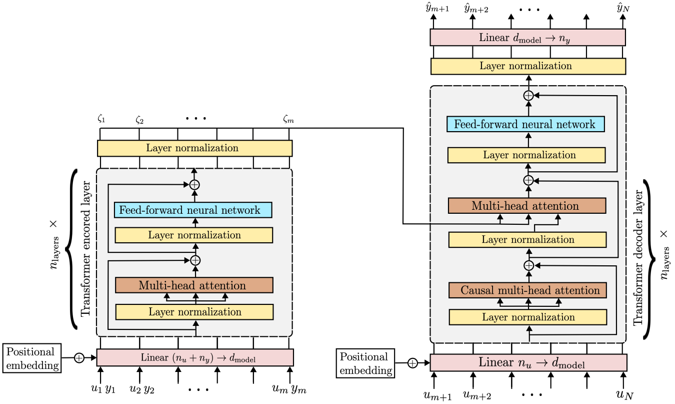
S. Garg et al. What Can Transformers Learn In-Context? A Case Study of Simple Function Classes. *36th Conference on Neural Information Processing Systems (NeurIPS 2022).*

T. Brown et al. Language Models are Few-Shot Learners. *34th Conference on Neural Information Processing Systems (NeurIPS 2020).*

Multi-step simulation

Meta-model trained to predict $\hat{y}_{m+1:N} = \mathcal{M}_\phi(u_{1:m}, y_{1:m}, u_{m+1:N})$

- Full I/O sequence $(u_{1:m}, y_{1:m})$ characterizes the dynamics (context)
- Input squence $u_{m+1:N}$ defines the simulation objective (task)

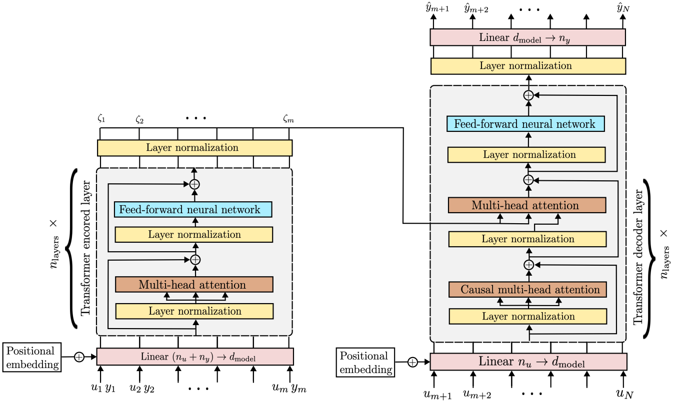


- The Transformer $\mathcal{M}_\phi$ becomes a meta model of the system class!
- $\mathcal{M}_\phi$ becomes as powerful as a system identification algorithm!

Multi-step simulation

Meta-model trained to predict $\hat{y}_{m+1:N} = \mathcal{M}_\phi(u_{1:m}, y_{1:m}, u_{m+1:N})$

- Full I/O sequence $(u_{1:m}, y_{1:m})$ characterizes the dynamics (context)
- Input squence $u_{m+1:N}$ defines the simulation objective (task)



- The Transformer $\mathcal{M}_\phi$ becomes a meta model of the system class!
- $\mathcal{M}_\phi$ becomes as powerful as a system identification algorithm!

Meta model training

Meta model trained in a standard supervised learning setting:

$$\hat{\phi} = \arg\min_{\phi} \mathcal{L}_{\rm sim}(\phi)$$

$$\begin{aligned}\mathcal{L}_{\rm sim}(\phi) &= \mathbb{E}_{p(\mathcal{D})}\left[\left\|y_{m+1:N} - \mathcal{M}_{\phi}(u_{1:m}, y_{1:m}, u_{m+1:N})\right\|^2\right] \\ &\approx \frac{1}{b}\sum_{i=1}^{b}\left\|y^{(i)}_{m+1:N} - \mathcal{M}_{\phi}(u^{(i)}_{1:m}, y^{(i)}_{1:m}, u^{(i)}_{m+1:N})\right\|^2\end{aligned}$$

- Training on on a whole class of dynamical systems makes the outcome special.
- If the optimization works out well, the Transformer becomes a meta model of the systems in $p(\mathcal{D})$.
- We have learned a learning algorithm!

Meta model training

Meta model trained in a standard supervised learning setting:

$$\hat{\phi} = \arg\min_{\phi} \mathcal{L}_{\rm sim}(\phi)$$

$$\begin{aligned}\mathcal{L}_{\rm sim}(\phi) &= \mathbb{E}_{p(\mathcal{D})}\left[\left\| y_{m+1:N} - \mathcal{M}_{\phi}(u_{1:m}, y_{1:m}, u_{m+1:N})\right\|^2\right] \\ &\approx \frac{1}{b}\sum_{i=1}^{b}\left\| y^{(i)}_{m+1:N} - \mathcal{M}_{\phi}(u^{(i)}_{1:m}, y^{(i)}_{1:m}, u^{(i)}_{m+1:N})\right\|^2\end{aligned}$$

- Training on on a whole class of dynamical systems makes the outcome special.
- If the optimization works out well, the Transformer becomes a meta model of the systems in $p(\mathcal{D})$.
- We have learned a learning algorithm!

Previous experiments - System classes

One-step prediction and multi-step simulation on two system classes:

Linear Time Invariant (LTI):

In state-space form, order ≤ 10

$$x_{k+1} = Ax_k + Bu_k$$
$$y_{k+1} = Cx_k$$

- Random system matrices
- A constrained to be stable

Wiener-Hammerstein (WH):

$$\mathbf{u} \rightarrow \boxed{G(z)} \rightarrow \boxed{F(\cdot)} \rightarrow \boxed{G(z)} \rightarrow \mathbf{y}$$

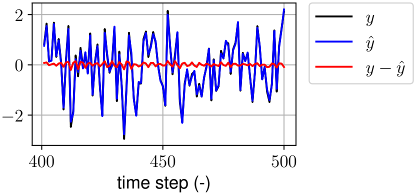
- Sequential LTI $\rightarrow F(\cdot) \rightarrow$ LTI
- Random LTI, order ≤ 5
- $F(\cdot)$: random feedforward NN.

- For both classes, input $u_{1:N}$ is a white Gaussian noise sequence.
- This defines a $p(\mathcal{D})$. We can generate infinite datasets!
- Each dataset from a different input/system realization!

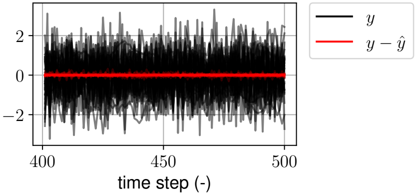
M. Forgione, F. Pura, D. Piga. In-context learning for model-free system identification. IEEE Control Systems Letters, 2023

Previous experiments - multi-step simulation results

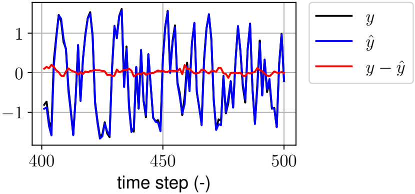
LTI: one sequence



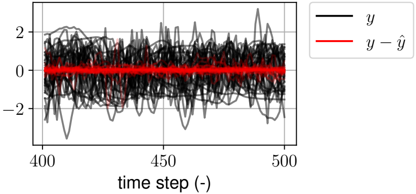
LTI: 256 sequences



WH: one sequence



WH: 32 sequences



New experiments - Generalization and adaptation

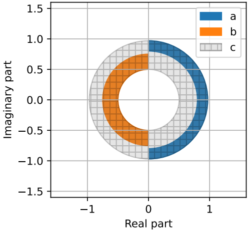
In the current contribution, we investigate:

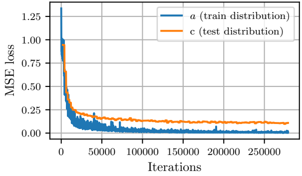
1. Generalization of a trained meta model on a new system class
2. Adaptation of a meta model to:
   1. A specific instance within the system class (specialization)
   2. A system instance outside of the system class
   3. New tasks. From 100- to 1000-step-ahead simulation

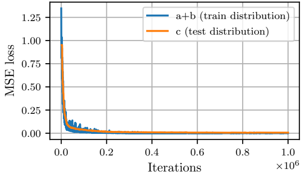
Generalization

Three LTI system classes. Order < 10, eigs with mag/phase in ranges:

- a $(0.8, 0.97)/(-\pi/2, \pi/2)$
- b $(0.5, 0.75)/(\pi/2, 3/4\pi)$
- c $(0.5, 0.97)/(-\pi, \pi)$





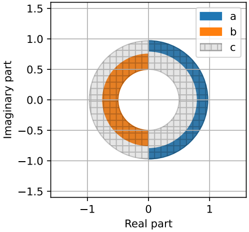


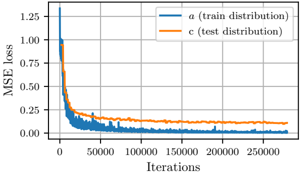
- Training only on a or b leads to a large generalization error on c.
- Training on a+b leads to a small generalization error on c
- Good generalization to unseen regions (gray area)

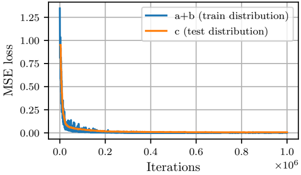
Generalization

Three LTI system classes. Order < 10, eigs with mag/phase in ranges:

a $(0.8, 0.97)/(-\pi/2, \pi/2)$
b $(0.5, 0.75)/(\pi/2, 3/4\pi)$
c $(0.5, 0.97)/(-\pi, \pi)$





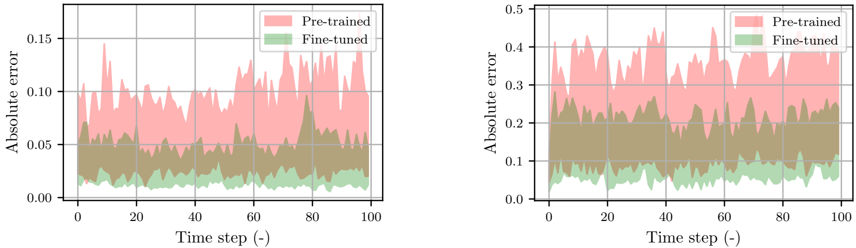


- Training only on a or b leads to a large generalization error on c.
- Training on a+b leads to a small generalization error on c
- Good generalization to unseen regions (gray area)

Adaptation to specific system instances

With just 140 sequences and $\approx$ 5 minutes of fine-tuning we can adapt, the WH meta model:

- To a specific instance of the WH class (left)
- To a PWH system instance, which is out of the WH class (right)

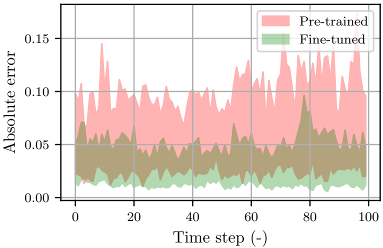


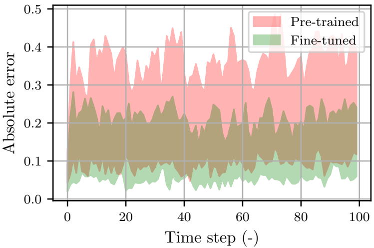
NOTE: training the WH meta model from scratch requires instead $\approx$ 1 day, 1M iterations, and 32M training instances!

Adaptation to specific system instances

With just 140 sequences and ≈ 5 minutes of fine-tuning we can adapt, the WH meta model:

- To a specific instance of the WH class (left)
- To a PWH system instance, which is out of the WH class (right)

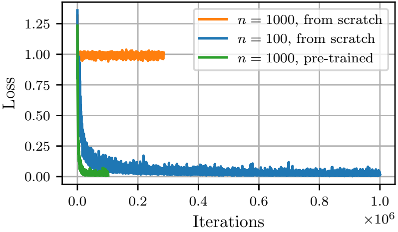




NOTE: training the WH meta model from scratch requires instead ≈ 1 day, 1M iterations, and 32M training instances!

Adaptation to new tasks

We try to learn a 1000-step-ahead meta model for the WH class.



- Learning the 1000-step model from scratch (orange) seems hard. Loss is stuck at a high value
- Learning a 100-step model (blue) is possible. We also did it in our previous work...
- Starting from the 100-step model, the optimization of the 1000-step model converges very quickly (green line)!

Conclusions

- While training of a meta-model from scratch is computationally intensive, adaptation to new tasks and systems is fast and efficient.
- This paves the way to the development of foundation models for system identification.

Many possible research directions and applications including:

- State estimation (see)
- Control
- Time series augmentation

Conclusions

- While training of a meta-model from scratch is computationally intensive, adaptation to new tasks and systems is fast and efficient.
- This paves the way to the development of foundation models for system identification.

Many possible research directions and applications including:

- State estimation (see)
- Control
- Time series augmentation

Thank you.
Questions?

marco.forgione@idsia.ch