Emergence in non-neural models

grokking modular arithmetic via average gradient outer product

Neil Mallinar, Daniel Beaglehole, Libin Zhu,

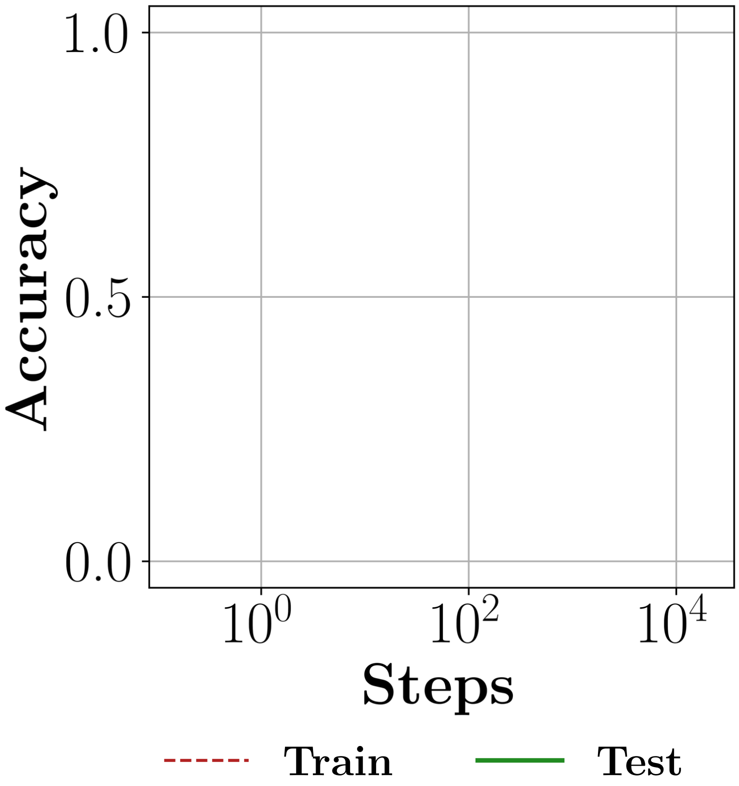
Adityanarayanan Radhakrishnan, Parthe Pandit, Mikhail Belkin

ICML 2025

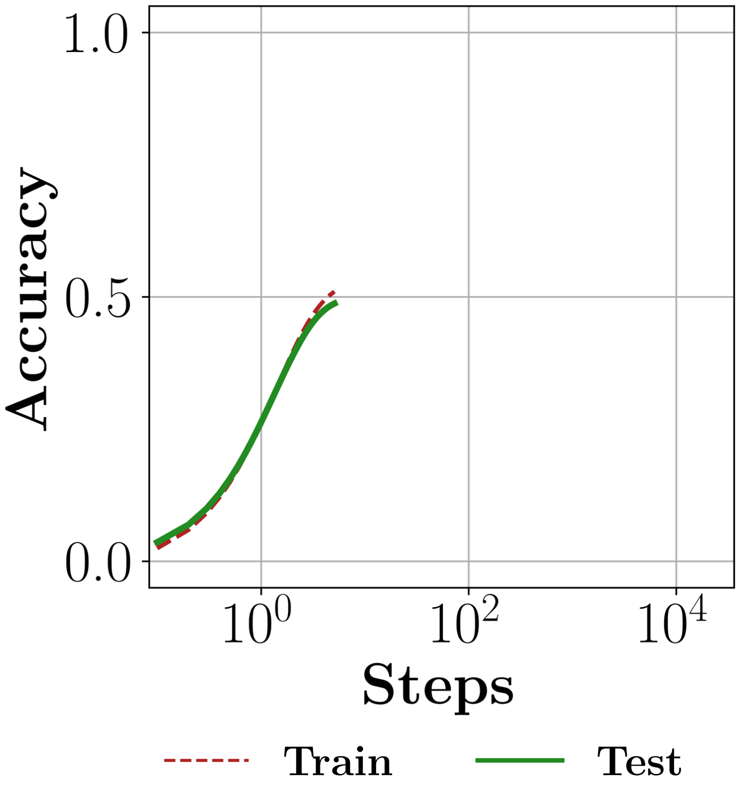
UC San Diego

BROAD INSTITUTE

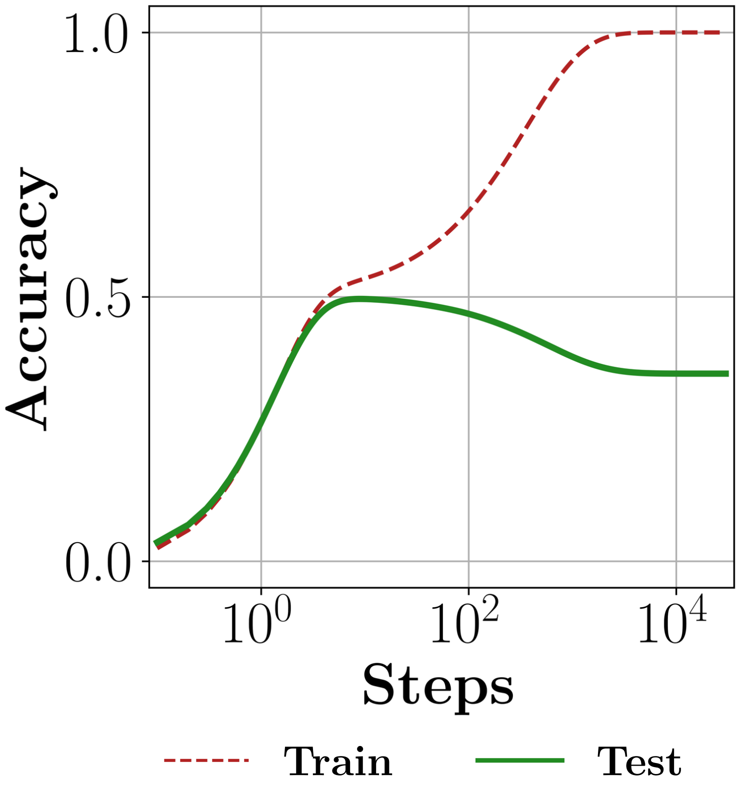
Classical approaches to tracking progress



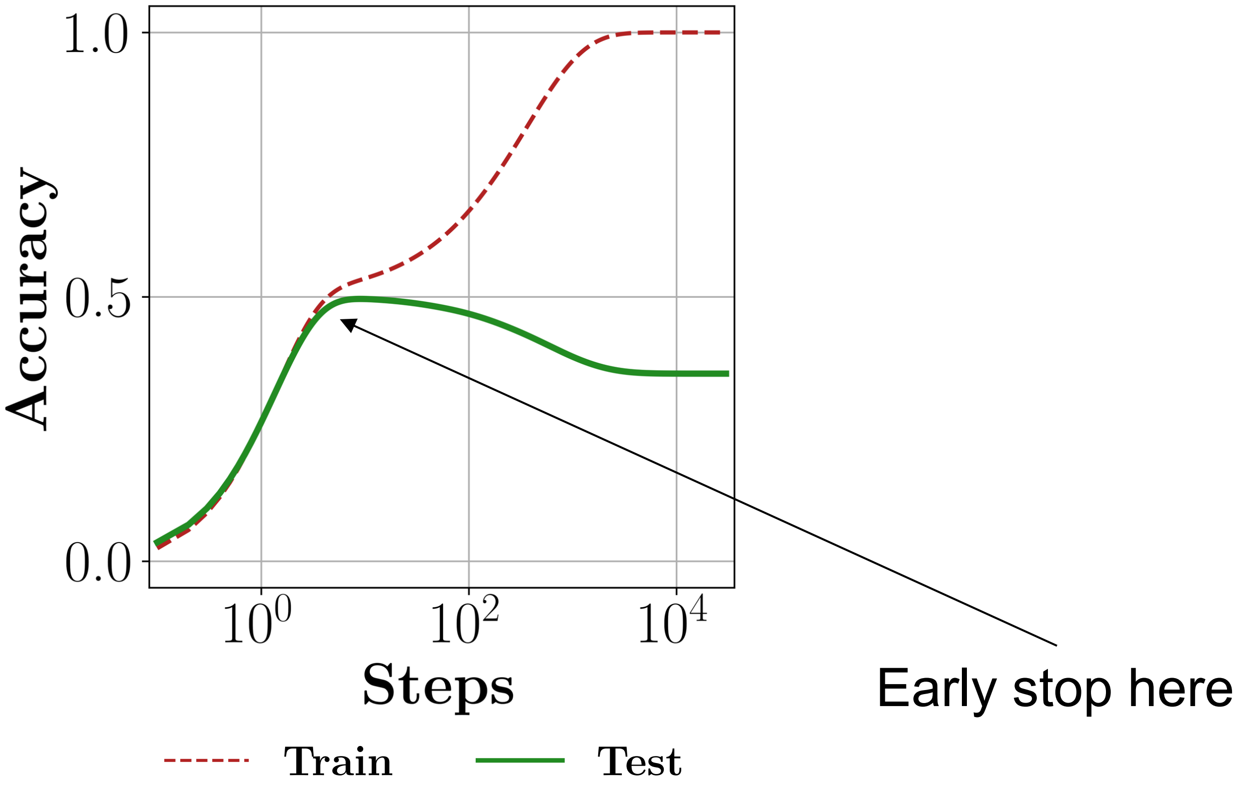
Classical approaches to tracking progress



Classical approaches to tracking progress

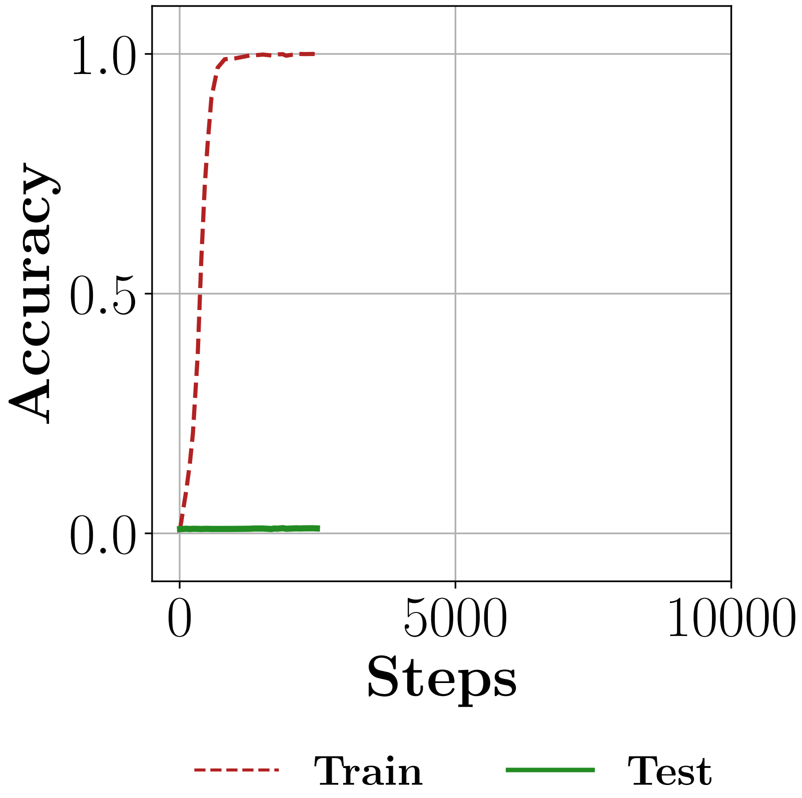


Classical approaches to tracking progress



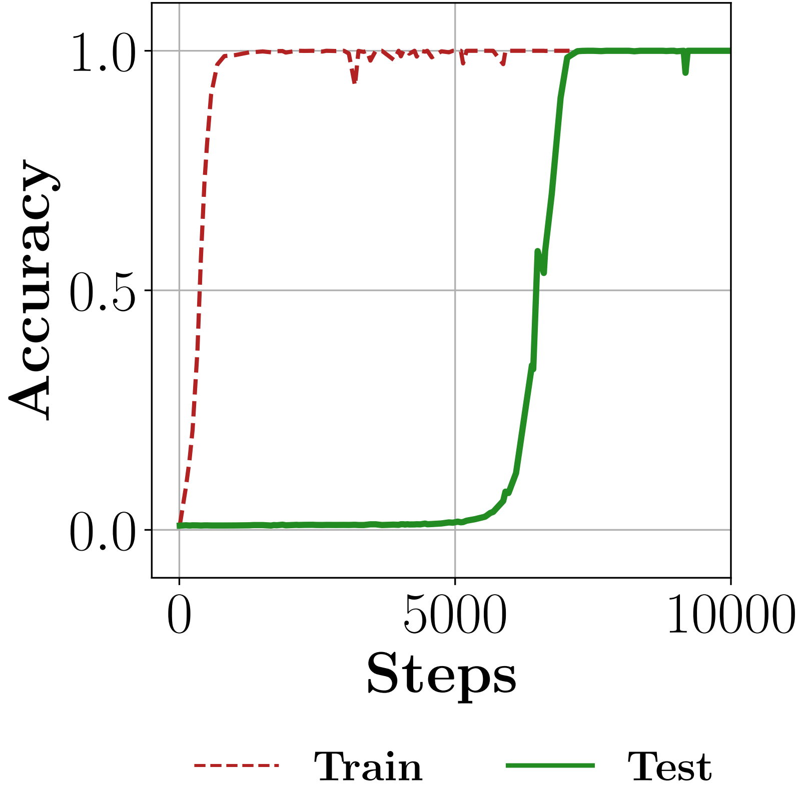
Test accuracy does not track progress!

Emergent setting:



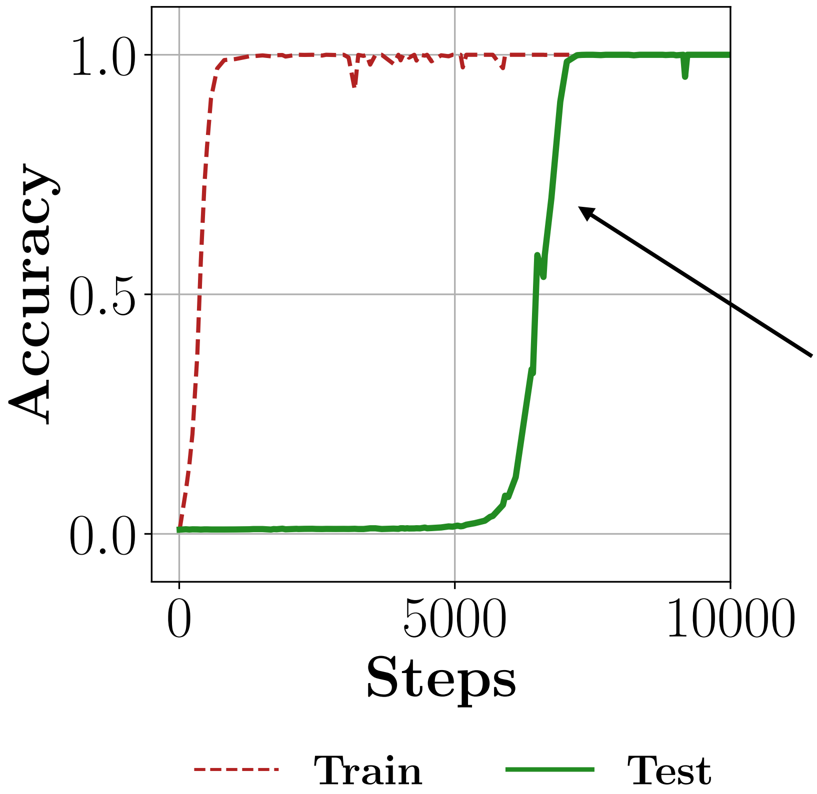
Test accuracy does not track progress!

Emergent setting:



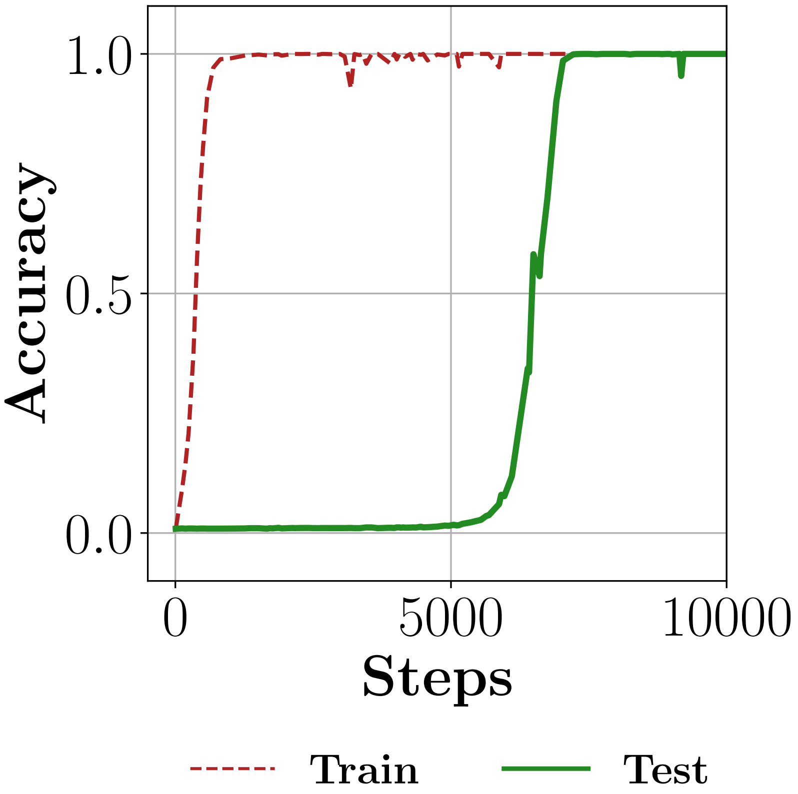
Test accuracy does not track progress!

Emergent setting:



"Grokking" in modular arithmetic, originally shown by (Power, Burda, Edwards, Babuschkin & Misra, *preprint* 2022)

Test accuracy does not track progress!



Key Questions:

- Is this phenomenon unique to neural networks?
- If not, is there a unified way to understand this behavior in neural and non-neural models?
- Is there an alterative to test accuracy?

How to set up a model to learn modular arithmetic

Given: digits a, b and prime p
Learn: a + b mod p

Example: p = 3, a = 1, b = 2

$$1 + 2 \bmod 3 = 0$$

How to set up a model to learn modular arithmetic

Given: digits a, b and prime p
Learn: a + b mod p

Example: p = 3, a = 1, b = 2

$$1 + 2 \bmod 3 = 0$$

+	0	1	2
0	0	?	2
1	1	?	0
2	?	0	?

(Cayley table)

How to set up a model to learn modular arithmetic

Given: digits a, b and prime p
Learn: a + b mod p

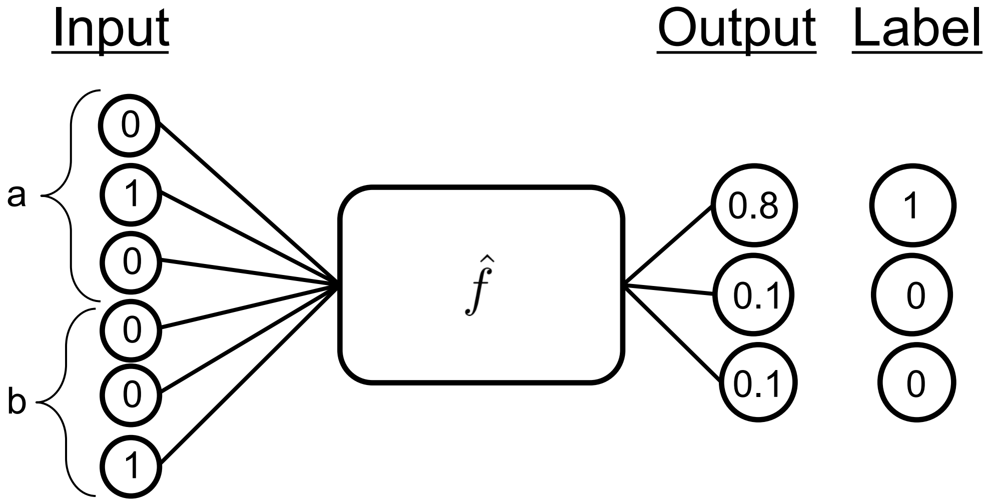
Example: p = 3, a = 1, b = 2

1 + 2 mod 3 = 0

+	0	1	2
0	0	?	2
1	1	?	0
2	?	0	?

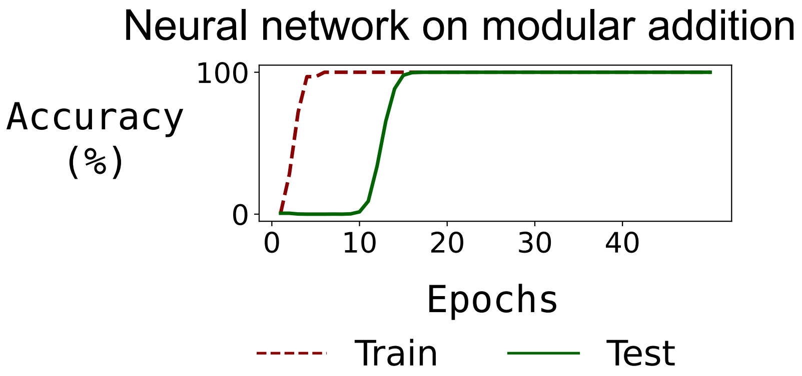
(Cayley table)

$$\text{Input: } (\underbrace{0\ 1\ 0}_{a=1}\ \ \underbrace{0\ 0\ 1}_{b=2})\ ;\ \text{Label: } \underbrace{(1\ 0\ 0)}_{a+b \mod 3=0}$$



Modular arithmetic and feature learning

- Neural networks can “grok” this task



- Non-feature learning methods (e.g. standard kernels) do not generalize (no grokking!)

A general mechanism for feature learning

- Understand features through Average Gradient Outer Product (AGOP) (Härdle & Stoker, *JASA* 1989)

Given a predictor, f, and training data $x_i \in \mathbb{R}^d$, define:

$$\mathbf{AGOP}(f, \{x_i\}_{i=1}^n) = \frac{1}{n}\sum_{i=1}^n \nabla_x f(x_i)\ \nabla_x f(x_i)^\top \in \mathbb{R}^{d\times d}$$

A general mechanism for feature learning

- Understand features through Average Gradient Outer Product (AGOP) (Härdle & Stoker, *JASA* 1989)

$$\text{Given a predictor, } f\text{, and training data } x_i \in \mathbb{R}^d\text{, define:}$$
$$\mathbf{AGOP}(f, \{x_i\}_{i=1}^n) = \frac{1}{n}\sum_{i=1}^n \nabla_x f(x_i)\ \nabla_x f(x_i)^\top \in \mathbb{R}^{d\times d}$$

- AGOP captures neural network features (Radhakrishnan*, Beaglehole*, Pandit & Belkin, *Science* 2024)

Intuition for AGOP

Given a predictor, f, and training data $x_i \in \mathbb{R}^d$, define:

$$\mathbf{AGOP}(f, \{x_i\}_{i=1}^n) = \frac{1}{n} \sum_{i=1}^n \nabla_x f(x_i) \ \nabla_x f(x_i)^\top \in \mathbb{R}^{d \times d}$$

- Input perturbations on certain features affect the output predictor most
- Intuitively AGOP = supervised PCA
- AGOP decouples features from predictors

Recursive Feature Machines (RFM)

- AGOP enables feature learning for general models
- RFM gives an algorithm for this (Radhakrishnan*, Beaglehole*, Pandit & Belkin, *Science* 2024)

Recursive Feature Machines (RFM)

- AGOP enables feature learning for general models
- RFM gives an algorithm for this (Radhakrishnan*, Beaglehole*, Pandit & Belkin, *Science* 2024)

RFM Algorithm:

Given: Training data (X, y), initial features M_0, total iterations T

For $t \in [T]$ *iterations*:

Step 1: Fit an estimator $f^{(t)}$ to (filtered) training data XM_t and labels y

Step 2: Update features as $M_{t+1} = \mathbf{AGOP}(f^{(t)}, X)$

Repeat Step 1 & Step 2

Recursive Feature Machines (RFM)

- AGOP enables feature learning for general models
- RFM gives an algorithm for this (Radhakrishnan*, Beaglehole*, Pandit & Belkin, *Science* 2024)

RFM Algorithm:

Given: Training data (X, y), initial features M_0, total iterations T
For $t \in [T]$ *iterations*:
Step 1: Fit an estimator $f^{(t)}$ to (filtered) training data XM_t and labels y
Step 2: Update features as $M_{t+1} = \mathbf{AGOP}(f^{(t)}, X)$
Repeat Step 1 & Step 2

Recursive Feature Machines (RFM)

- AGOP enables feature learning for general models
- RFM gives an algorithm for this (Radhakrishnan*, Beaglehole*, Pandit & Belkin, *Science* 2024)

RFM Algorithm:

Given: Training data (X, y), initial features M_0, total iterations T

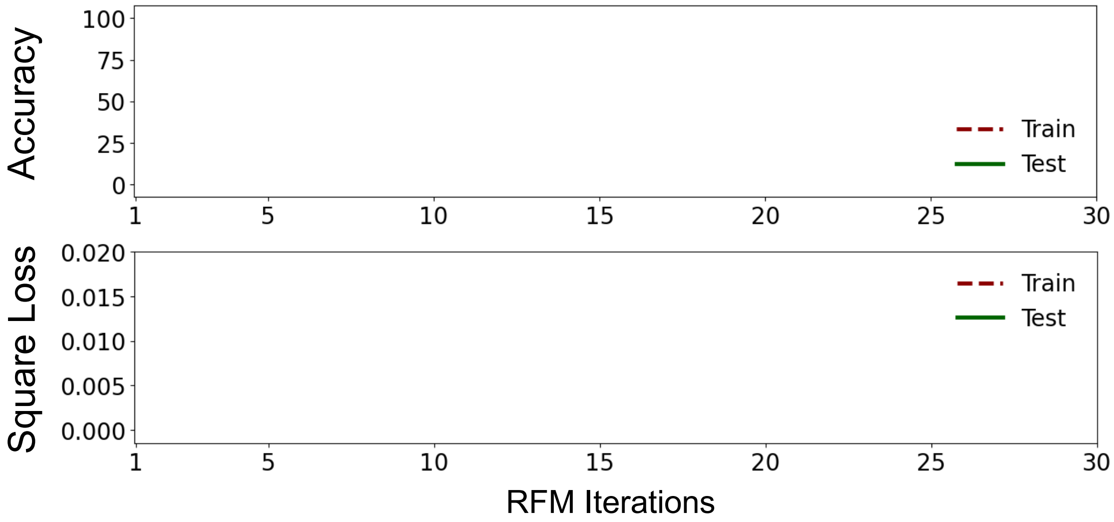
For $t \in [T]$ *iterations*:

Step 1: Fit an estimator $f^{(t)}$ to (filtered) training data XM_t and labels y

Step 2: Update features as $M_{t+1} = \mathbf{AGOP}(f^{(t)}, X)$

Repeat Step 1 & Step 2

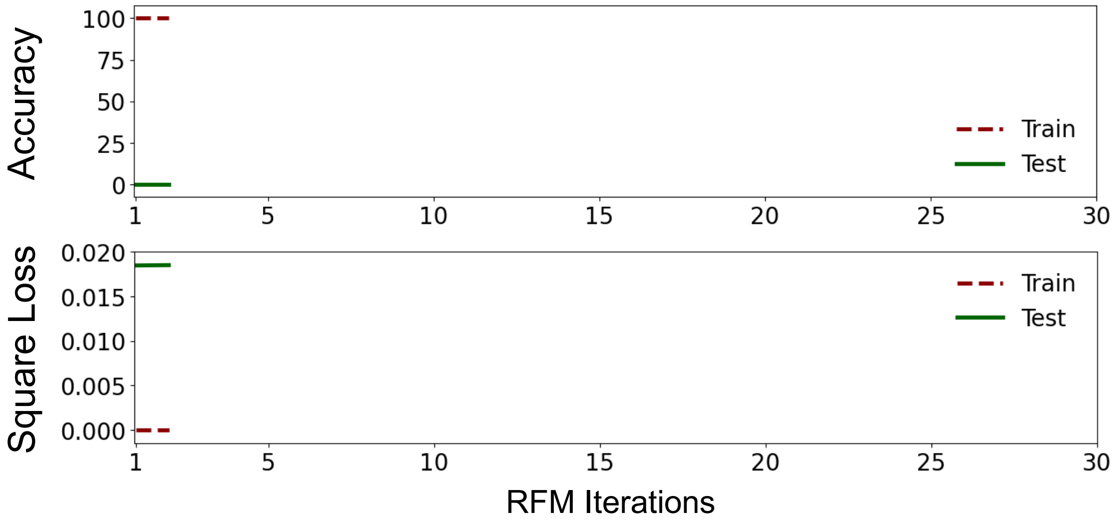
Kernel RFM groks modular addition



Initialize: $M_1 = I_{2p}$

Iteration (t): 1

Kernel RFM groks modular addition



Initialize: $M_1 = I_{2p}$

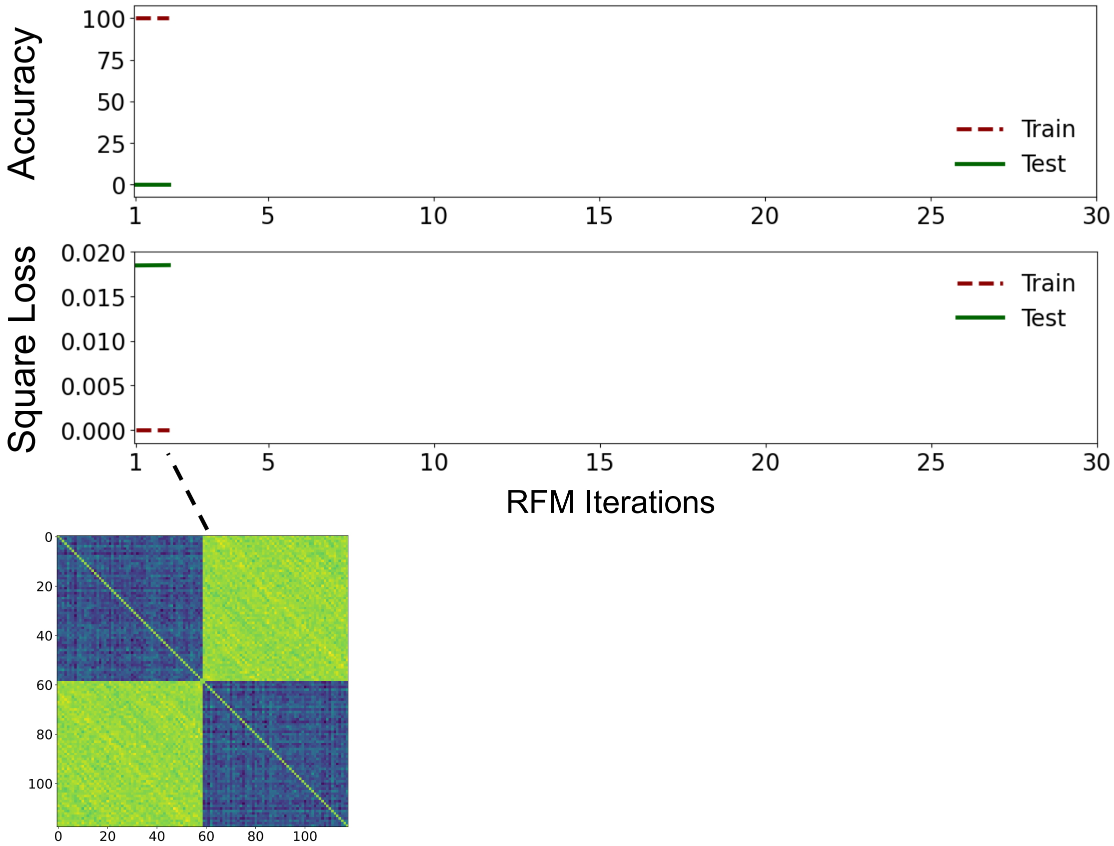
Iteration (t): 1

<u>(1) Solve kernel regression:</u>

$K_{tr} = k(X_{tr}, X_{tr}; M_1)$

$\alpha = K_{tr}^{-1} y_{tr}$

Kernel RFM groks modular addition



Initialize: $M_1 = I_{2p}$

Iteration (t): 1

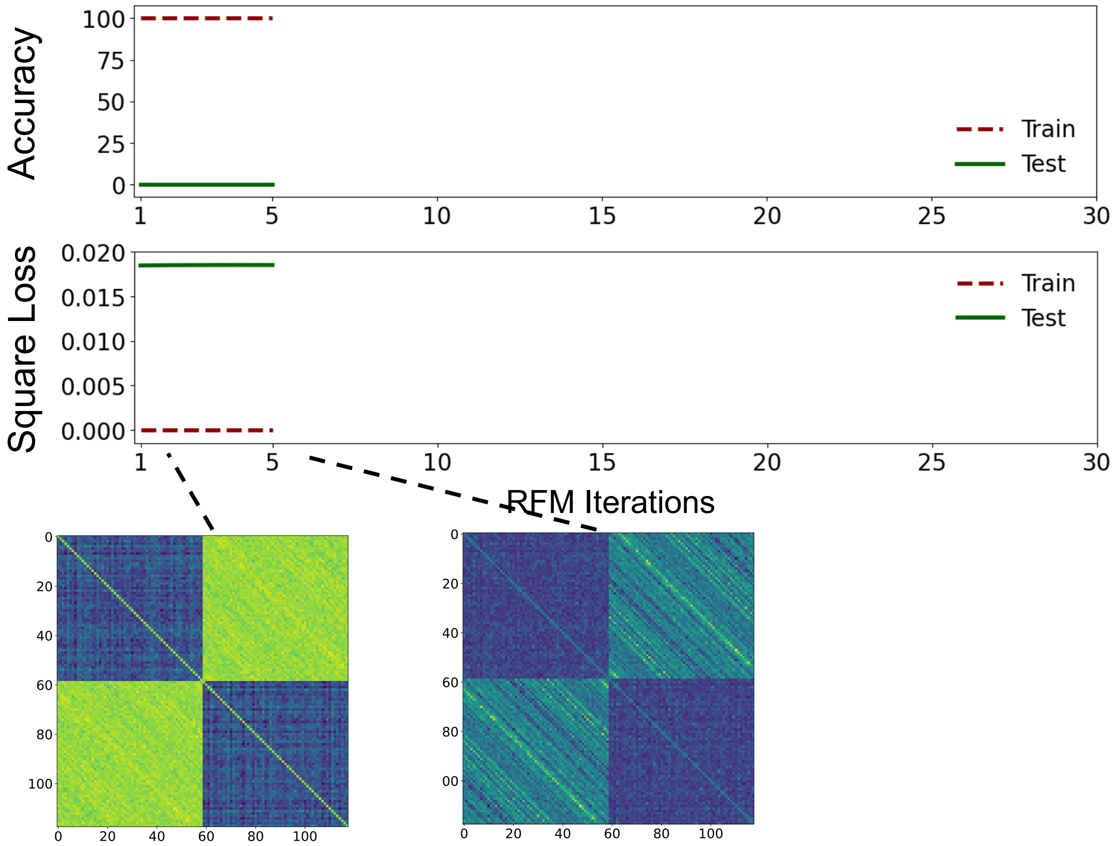
(1) Solve kernel regression:

$$K_{tr} = k(X_{tr}, X_{tr}; M_1)$$
$$\alpha = K_{tr}^{-1} y_{tr}$$

(2) Update features:

$$f(x) = \sum_{i=1}^{n} \alpha_i \, k(x, X_{tr}^{(i)}; M_1)$$
$$M_2 = AGOP(f, X_{tr})$$

Kernel RFM groks modular addition



Iteration (t): 5

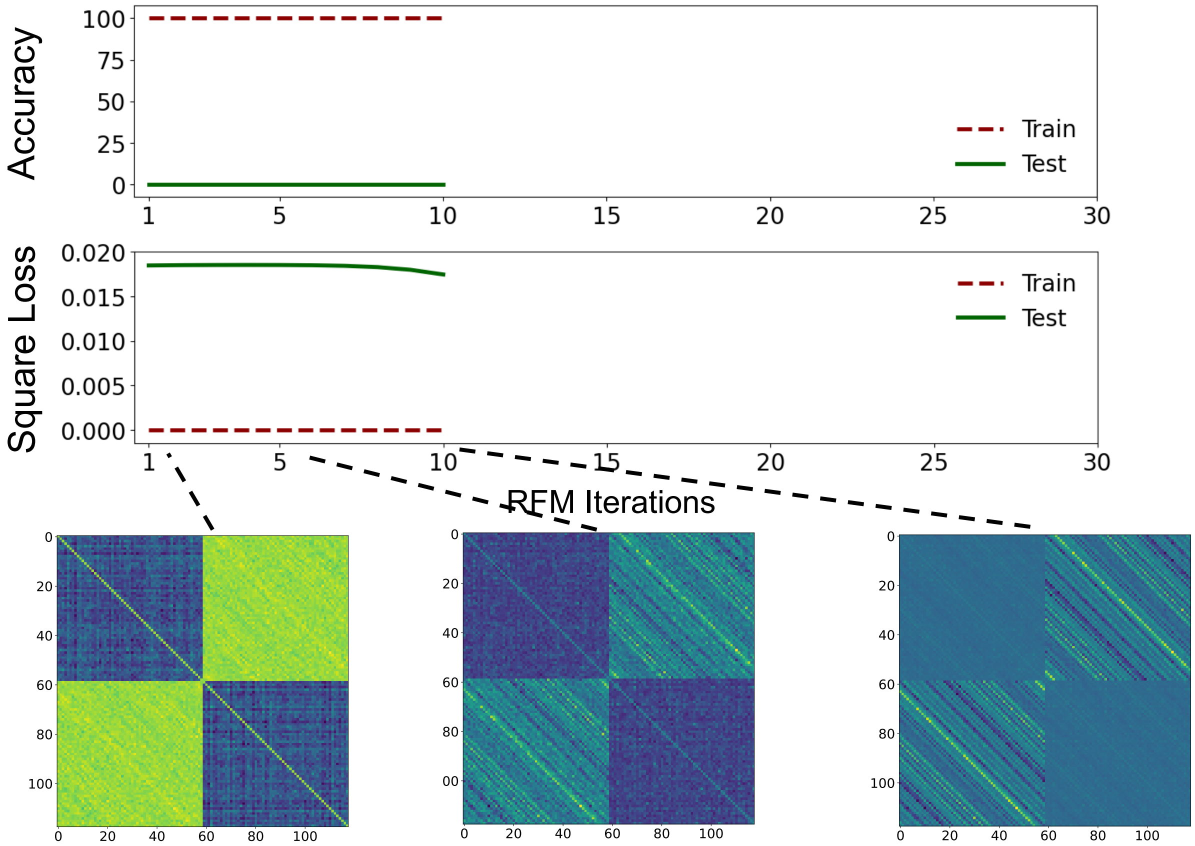
(1) Solve kernel regression:

$$K_{tr} = k(X_{tr}, X_{tr}; M_5)$$
$$\alpha = K_{tr}^{-1} y_{tr}$$

(2) Update features:

$$f(x) = \sum_{i=1}^{n} \alpha_i \, k(x, X_{tr}^{(i)}; M_5)$$
$$M_6 = AGOP(f, X_{tr})$$

Kernel RFM groks modular addition



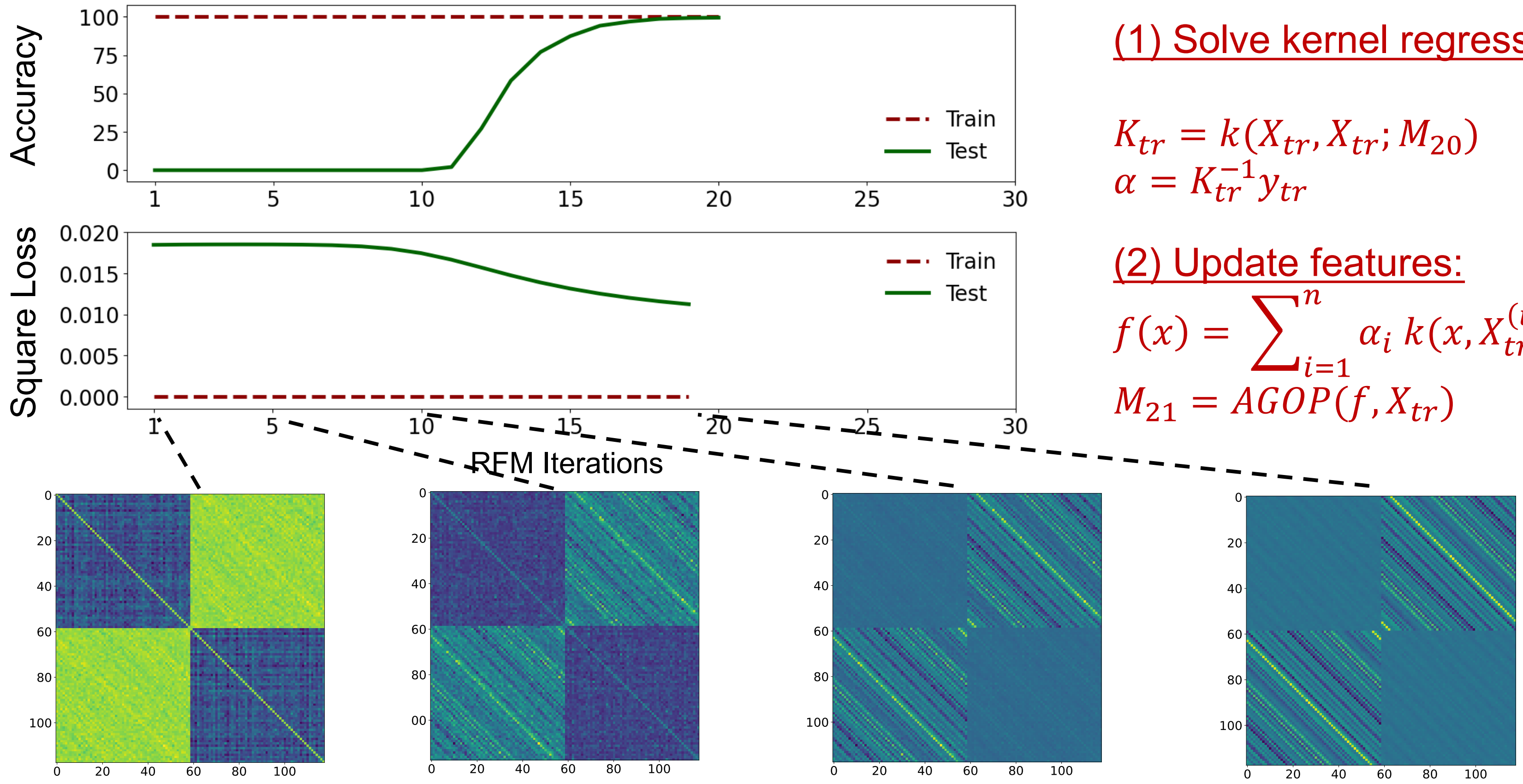
(1) Solve kernel regression:

$$K_{tr} = k(X_{tr}, X_{tr}; M_{10})$$
$$\alpha = K_{tr}^{-1} y_{tr}$$

(2) Update features:

$$f(x) = \sum_{i=1}^{n} \alpha_i \, k(x, X_{tr}^{(i)}; M_{10})$$
$$M_{11} = AGOP(f, X_{tr})$$

Kernel RFM groks modular addition



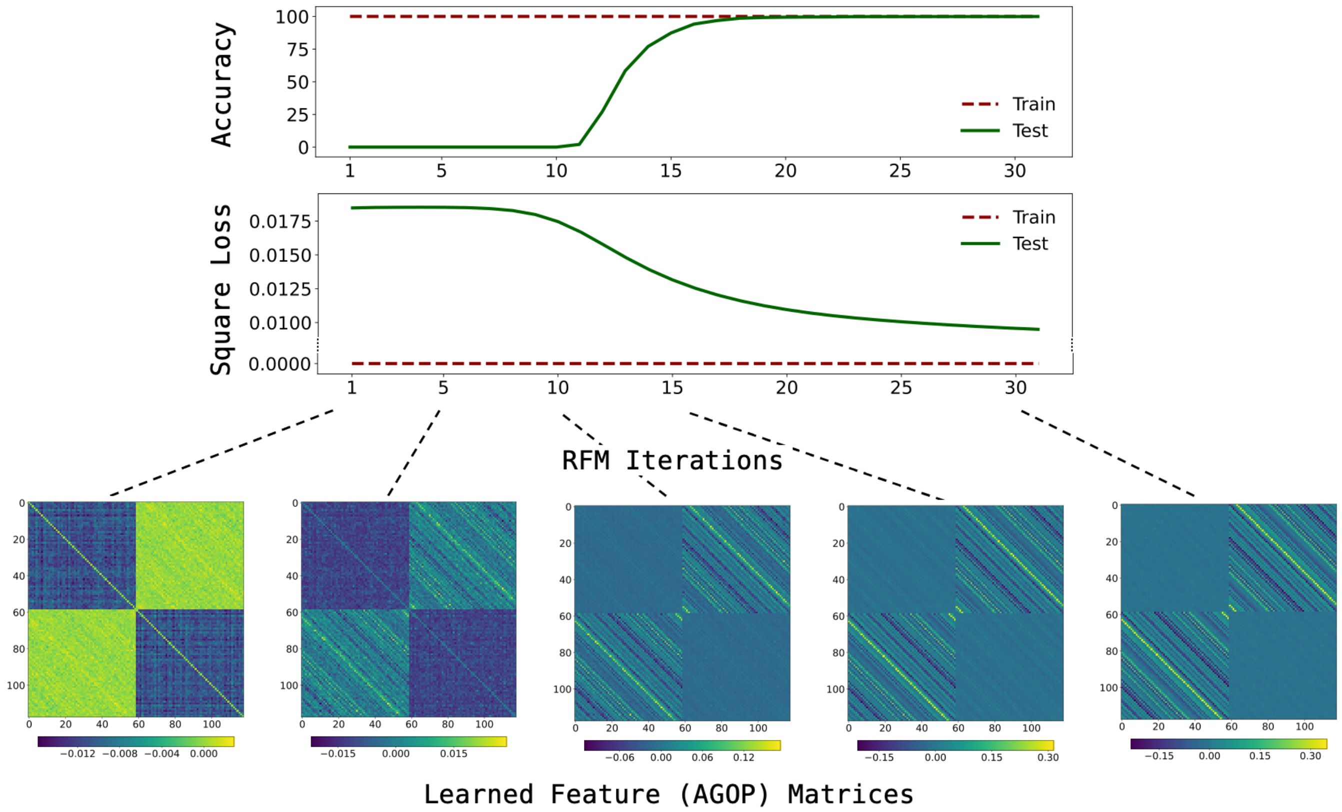
(1) Solve kernel regression:

$$K_{tr} = k(X_{tr}, X_{tr}; M_{20})$$
$$\alpha = K_{tr}^{-1} y_{tr}$$

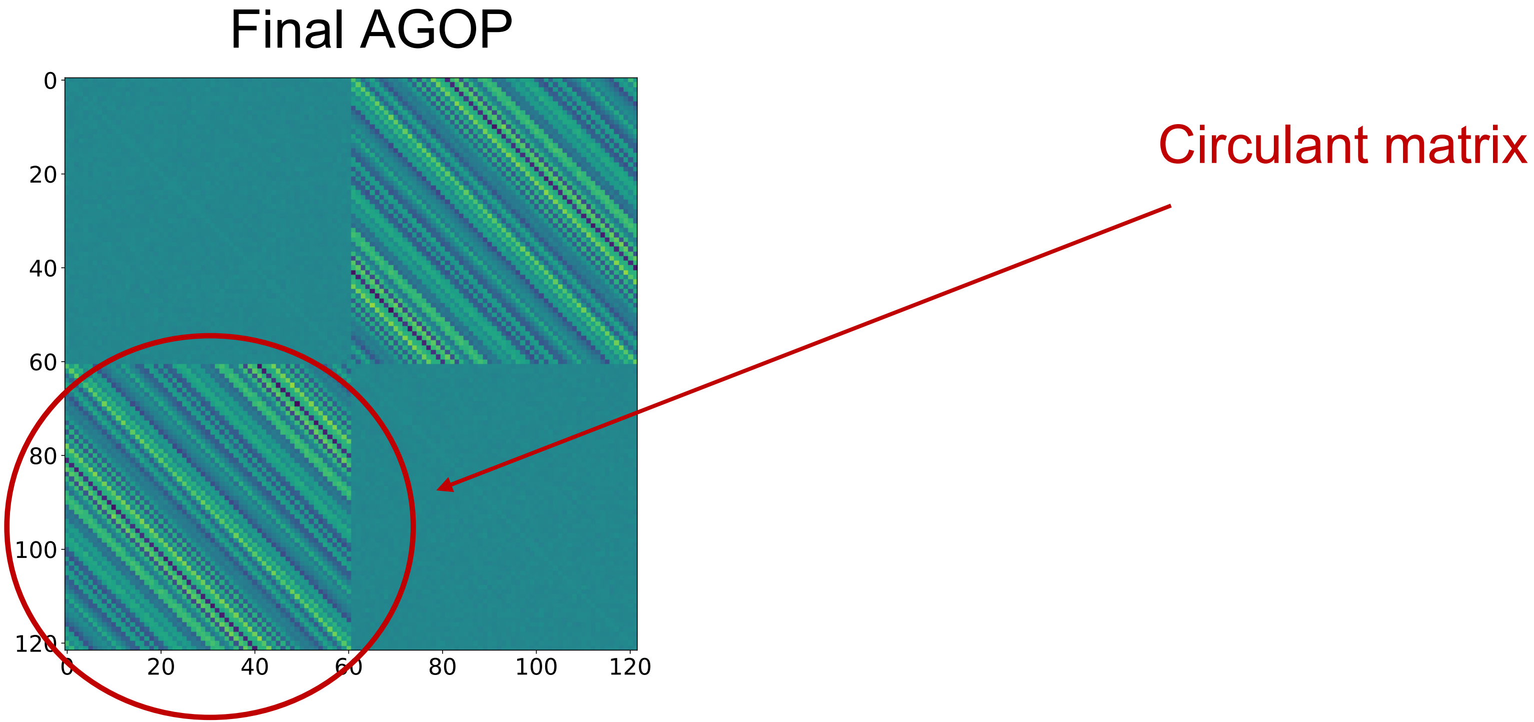
(2) Update features:

$$f(x) = \sum_{i=1}^{n} \alpha_i \, k(x, X_{tr}^{(i)}; M_{20})$$
$$M_{21} = AGOP(f, X_{tr})$$

Kernel RFM groks modular addition

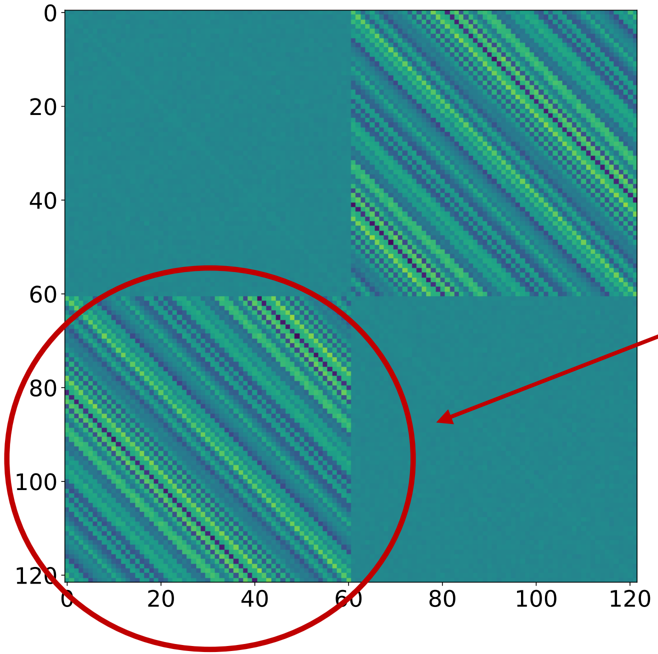


What is happening under the hood for addition?



What is happening under the hood for addition?

Final AGOP

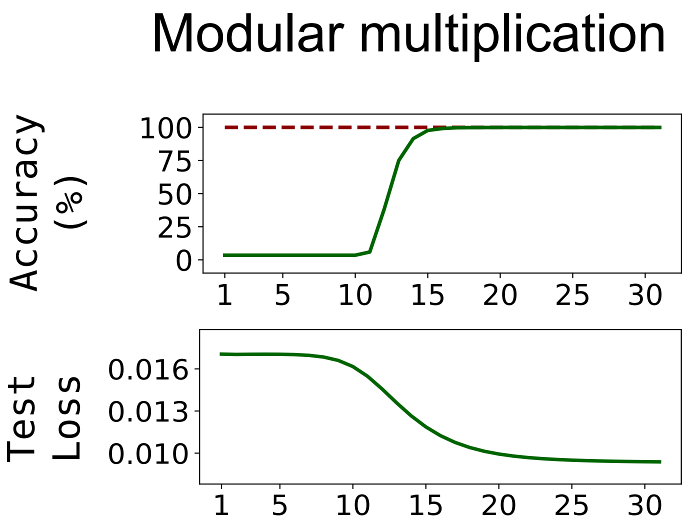


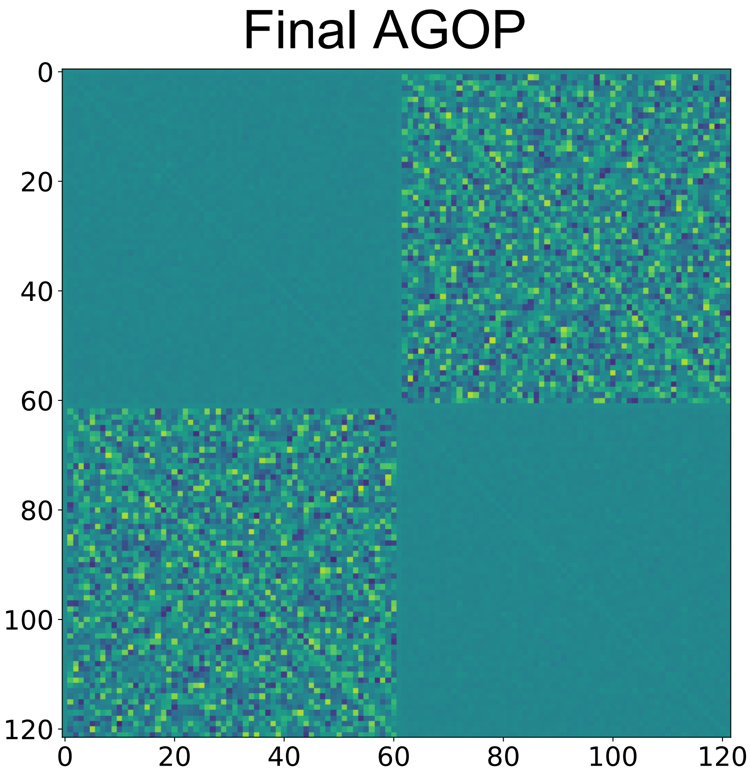
Circulant matrix

Fourier Multiplication Algorithm:

$$\arg\max(F^{-1}(Fe_a \odot Fe_b)) = (a+b) \mod p$$

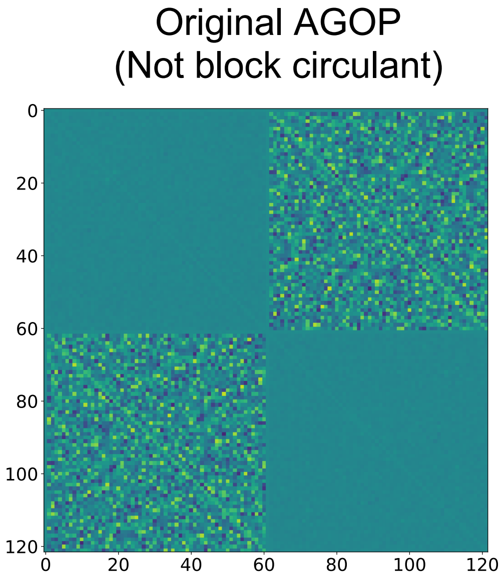
Kernel RFM groks multiplication mod p





Kernel RFM groks multiplication mod p

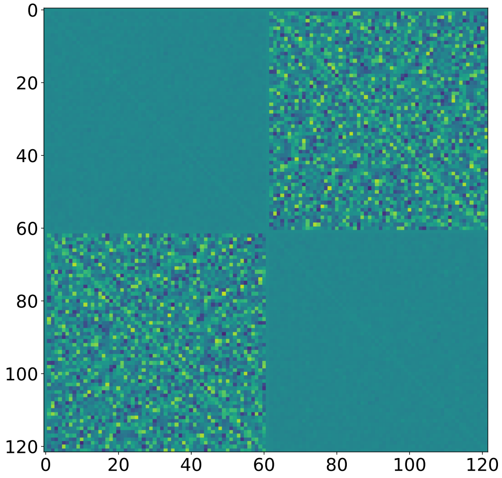
- Log turns multiplication into addition: log(ab) = log(a) + log(b)
- There is a notion of discrete logarithm for modular arithmetic



Kernel RFM groks multiplication mod p

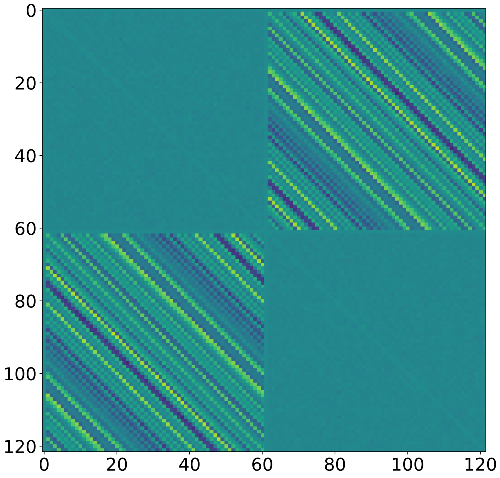
- Log turns multiplication into addition: log(ab) = log(a) + log(b)
- There is a notion of discrete logarithm for modular arithmetic

Original AGOP
(Not block circulant)

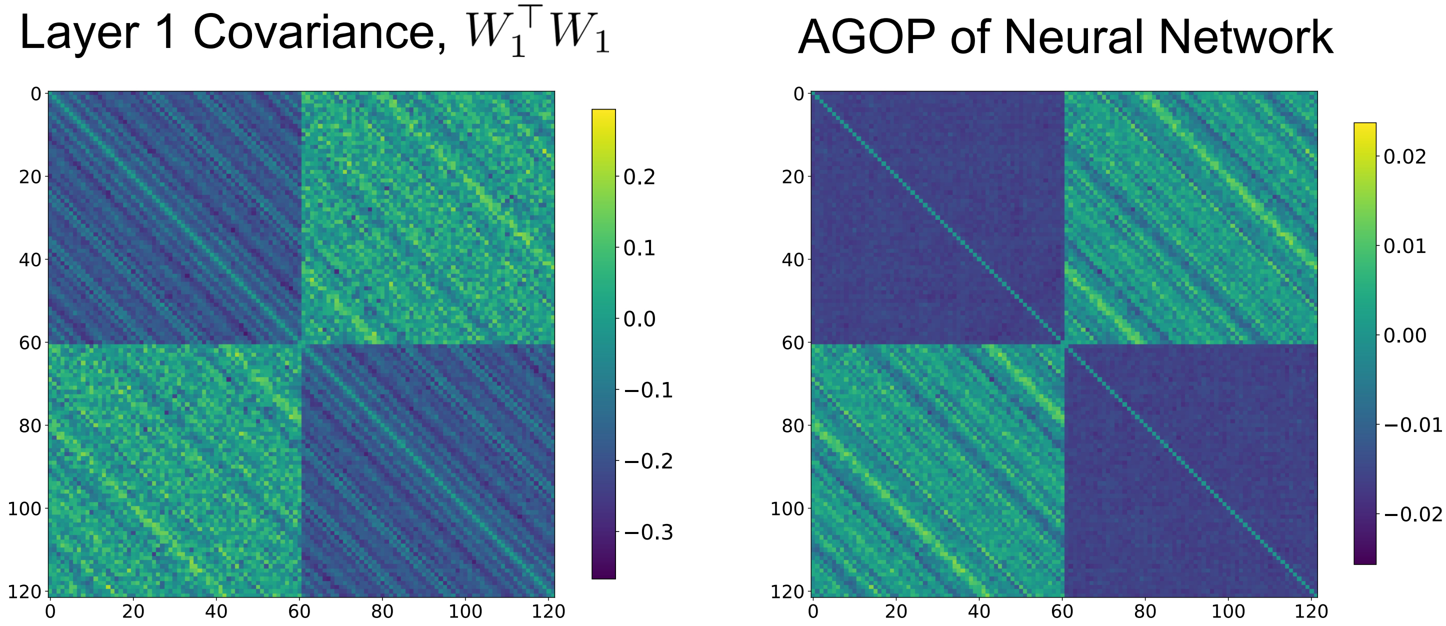


Reorder by discrete log mod p

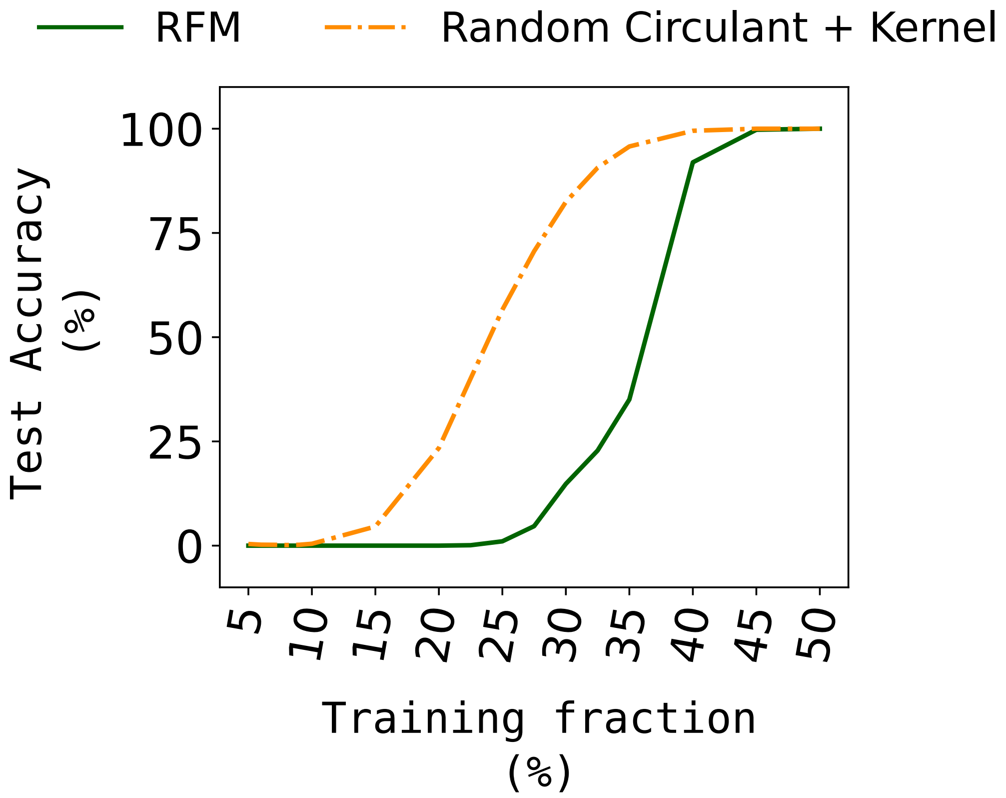
Reordered AGOP
(Block circulant)



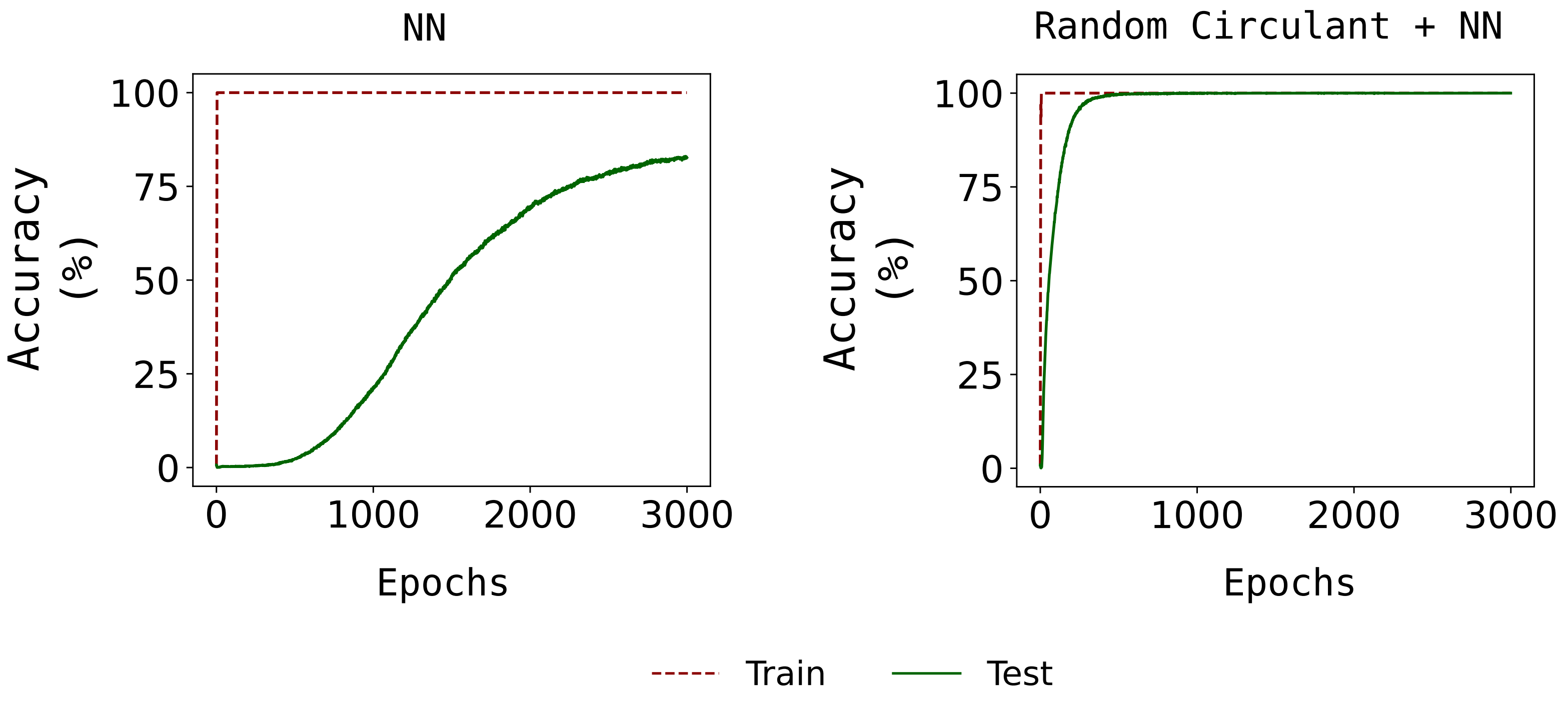
Neural networks also learn block circulant features



Random circulant features are sufficient to generalize



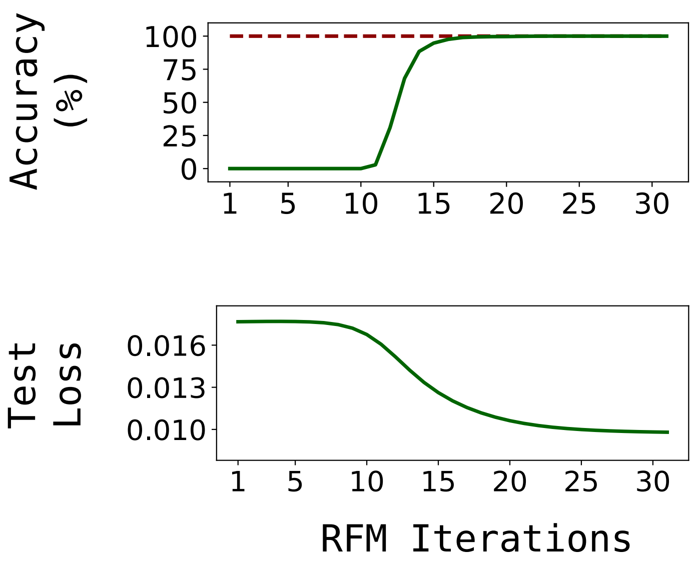
Random circulant features are sufficient to generalize



Progress measures

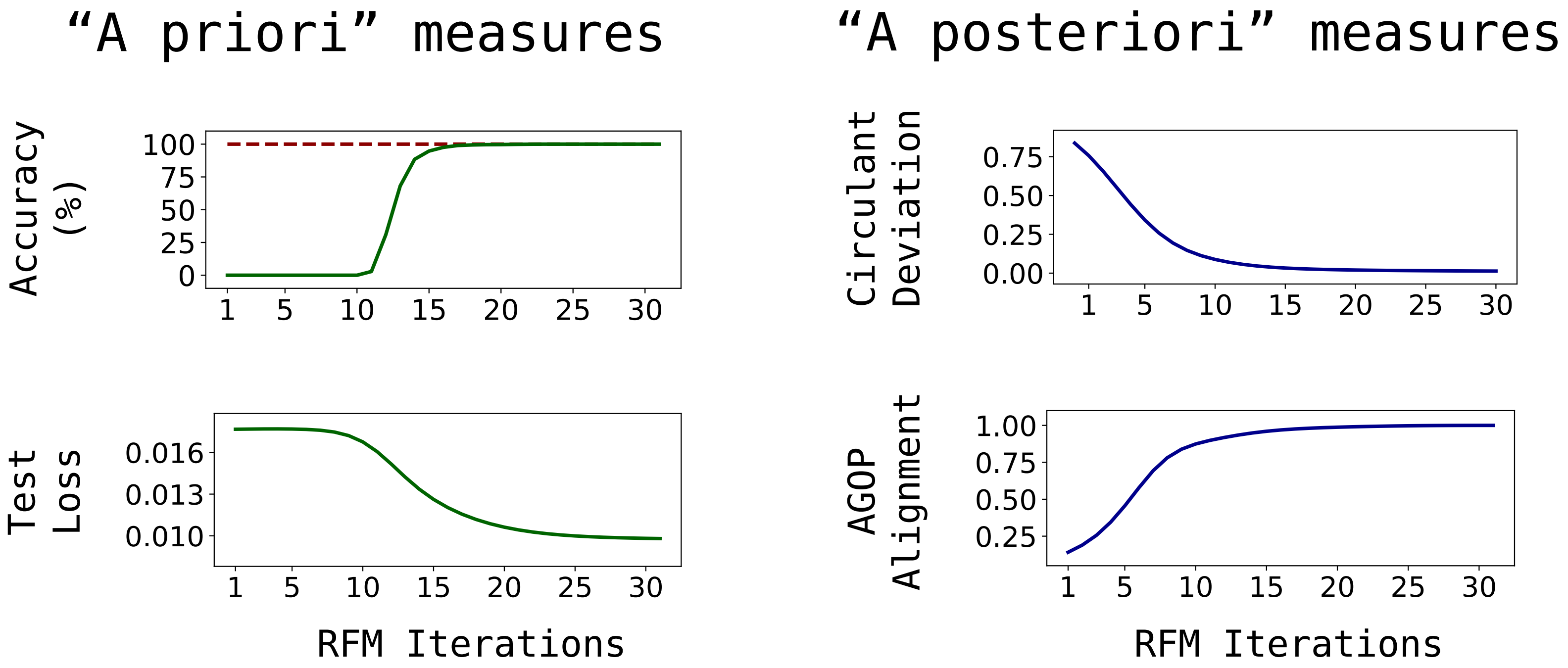
Back to our original question: **how do we track progress?**

“A priori” measures



Progress measures

Back to our original question: **how do we track progress?**



Non-neural models can grok modular arithmetic from data

1. We show for the first time that non-neural models can grok modular arithmetic from data
    a) Grokking is a manifestation of gradual feature learning

Non-neural models can grok modular arithmetic from data

1. We show for the first time that non-neural models can grok modular arithmetic from data
   a) Grokking is a manifestation of gradual feature learning

2. AGOP is key to understanding feature learning
   a) Kernel RFM reproduces unexpected feature learning phenomena
   b) Model agnostic features, shown by random circulant experiments

Non-neural models can grok modular arithmetic from data

1. We show for the first time that non-neural models can grok modular arithmetic from data
   a) Grokking is a manifestation of gradual feature learning

2. AGOP is key to understanding feature learning
   a) Kernel RFM reproduces unexpected feature learning phenomena
   b) Model agnostic features, shown by random circulant experiments

3. Grokking modular arithmetic is NOT:
   a) tied to gradient descent based optimization methods
   b) predicted by training nor testing loss, let alone accuracy

In close collaboration with:

Daniel Beaglehole
UC San Diego

Libin Zhu
UC San Diego → University of Washington

Adityanarayanan Radhakrishnan
The Broad Institute of MIT and Harvard

Parthe Pandit
IIT Bombay

Misha Belkin
UC San Diego

Thank you!

Poster session is tomorrow (Wed, July 16th) from 11am - 1:30pm!

Paper:

Personal Website:

Contact: nmallina@ucsd.edu

P.S. I am on the job market!