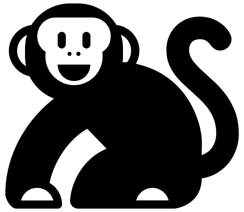
~~Recurrent Convolutional Neural Networks Learn Succinct Learning Algorithms~~ [NeurIPS 2022]

Succinct Neural Networks

Outline:
1) What are they?
2) Use them for Probably-Approximately-Optimal / Turing-Optimal learning, e.g. NN's learn parity


Surbhi Goel
UPenn


Sham Kakade
Harvard


Adam Tauman Kalai
Microsoft Research


Cyril Zhang
Microsoft Research

Motivation: NNs bad for learning *algorithms*

More 7's

x	y
471 9125	471
7 318	7
27097 77	27097
747 2023	747

```
def more_7s(a, b):
```

69932

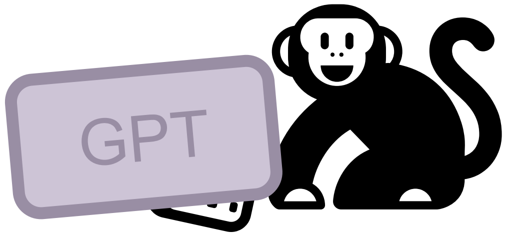
819

Parity (no noise)

```
# examples:
# more_7s(471, 9125) => 471
# more_7s(7, 318) => 7
# more_7s(27097, 77) => 27097
# more_7s(747, 2023) => 747
def more_7s(a, b):
```

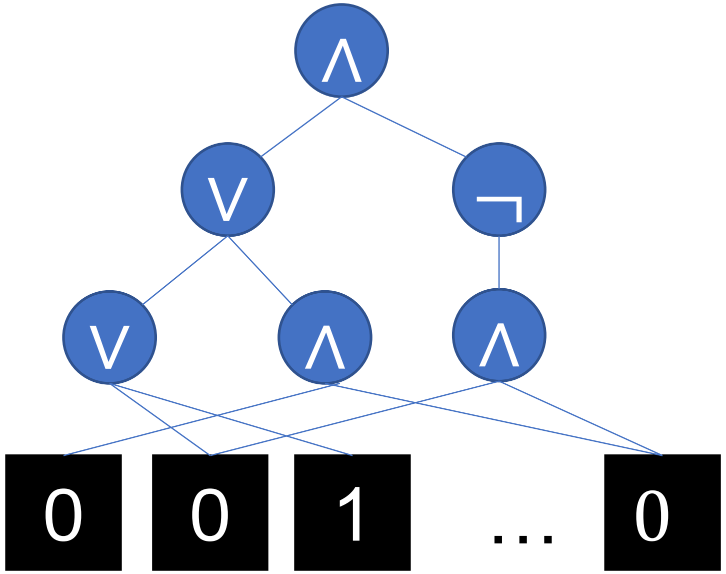
"Learnable" from examples [Levin+Allender+Valiant?]

- Find constant-size (time-bounded) TM mapping $x_i \rightarrow y_i$

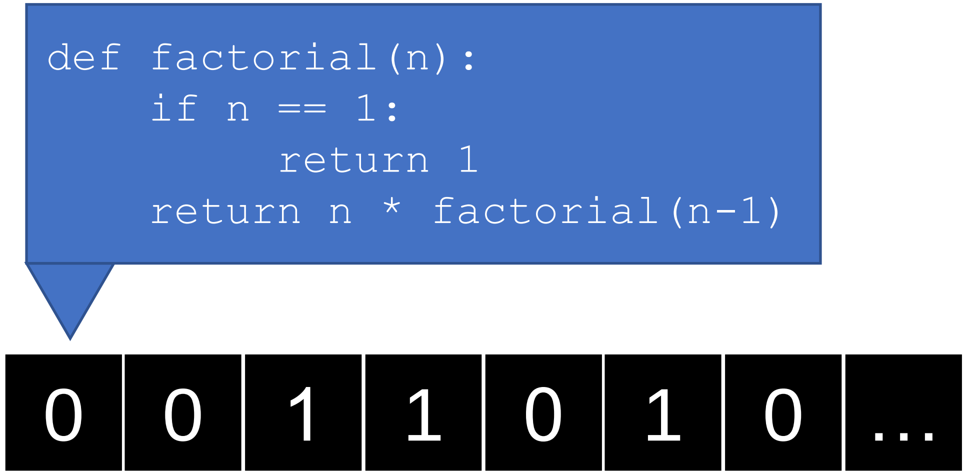


Deep vs. Succinct Neural Networks

Neural Networks

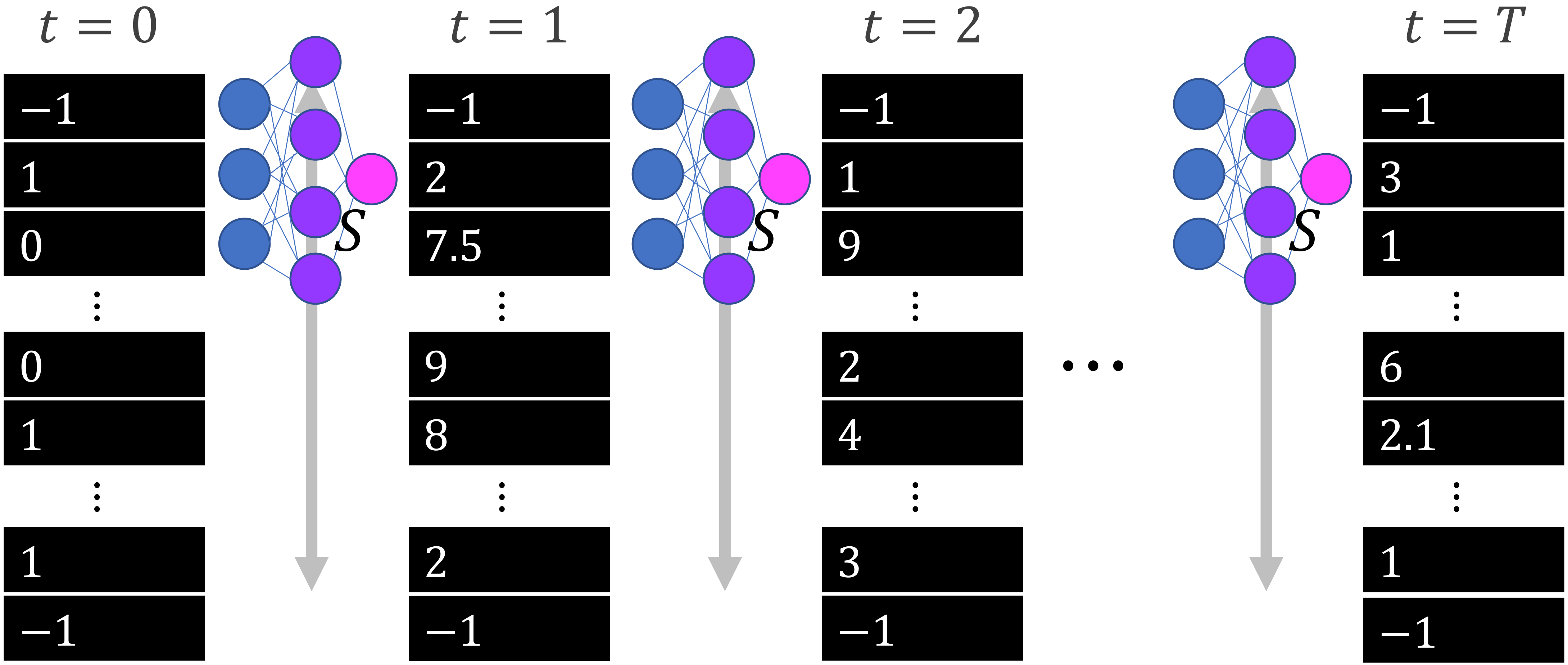


Succinct Neural Networks



DNN : Circuit :: Succ. NN : Turing Machine

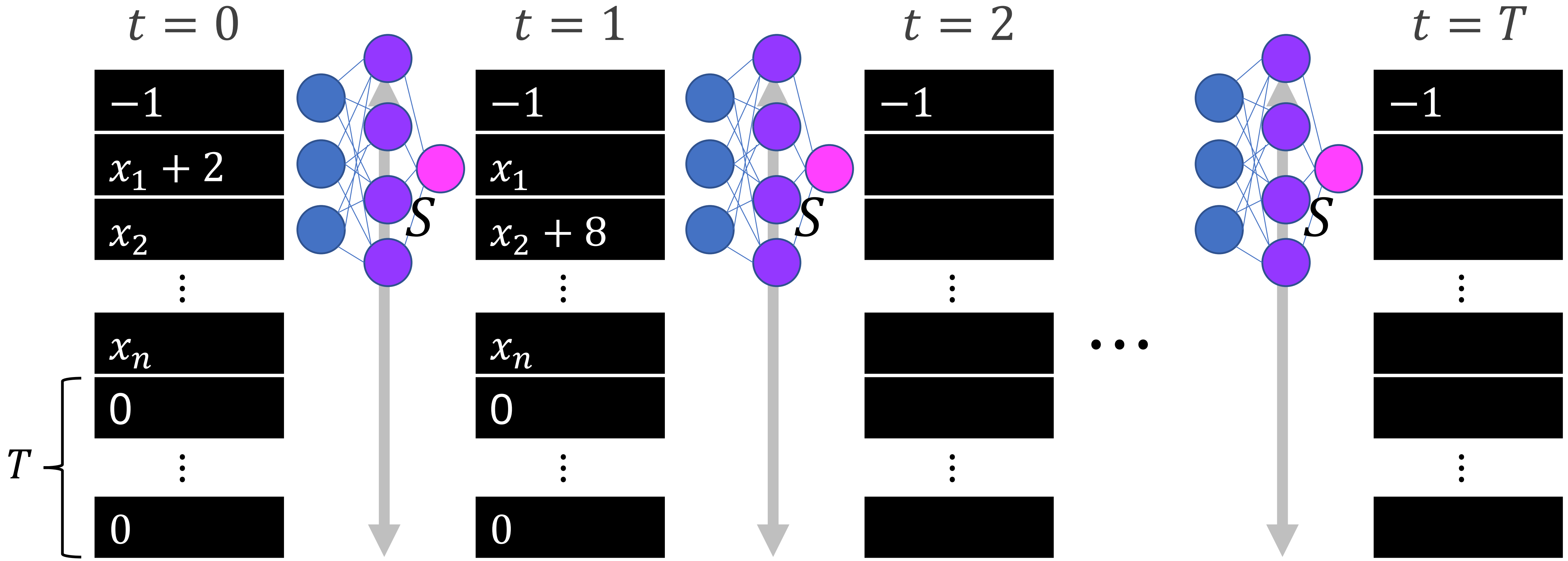
SuNN: Succinct NN



$$|S| \leq K$$

SuNN: Succinct NN

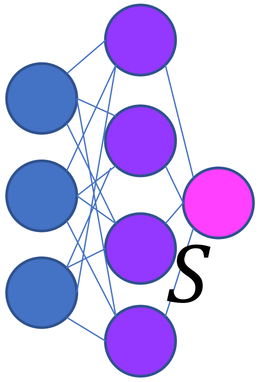
M states = $\{0\ (\text{halt}), 2\ (\text{initial}), 4, 6, \ldots, 2k\}$
Add state to TM head position



Lemma. Convert any k-state Turing Machine M to NN, $|S(M)| \leq K$ such that for every $x \in \{0,1\}^*$ and $T \geq \text{time}(M, x)$: $M(x) = \text{SuNN}(S(M), T, x)^*$

*weights/activations use $O(1)$ bits, runs in $\text{poly}(T, |x|)$ time

Learn succinct NNs, learn algorithms

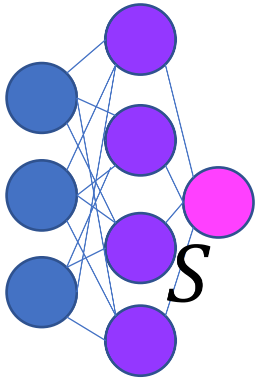


Only need to learn constant # of weights of S

Examples:

- Multiplication: $x_i = (a_i, b_i)$ $y_i = a_i \times b_i$
- Shortest paths: $x_i = (V_i, E_i)$ $y_i =$ length of shortest path in graph (V_i, E_i)
- Smallest factor: $x_i \in \text{Compsit}$ $y_i = \text{smallest prime factor of } x_i$
- Parity functions: $x_i \in \{0,1\}^n$ $y_i = (x_i \cdot w) \bmod 2$ for $w \in \{0,1\}^n$

Learn succinct NNs, learn algorithms



Only need to learn constant # of weights of S

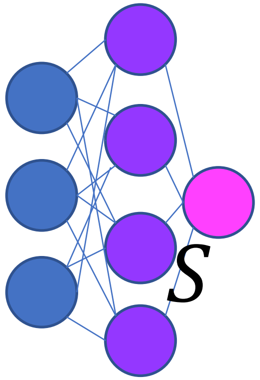
For all T, const. k, with prob. $\geq 99\%$ over $(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m) \sim \mathcal{D}$:

$$\max_{|S| \leq K} |\mathrm{err}_{\mathcal{D}}(S) - \widehat{\mathrm{err}}(S)| \leq O\left(\sqrt{1/m}\right)$$

$$\mathrm{err}_{\mathcal{D}}(S) \coloneqq \Pr_{x,y \sim \mathcal{D}}[\mathrm{SuNN}(S, T, x) \neq y], \qquad \widehat{\mathrm{err}}(S) \coloneqq \frac{1}{m} |\{\mathrm{i} \mid \mathrm{SuNN}(S, T, x_i) \neq y_i\}|$$

…so solve $\displaystyle \min_{|S| \leq K} \frac{1}{m} \sum_i || \mathrm{SuNN}(S, T, x_i) - y_i ||^2$

Learn succinct NNs, learn algorithms



Repeat $O(1)$ times:

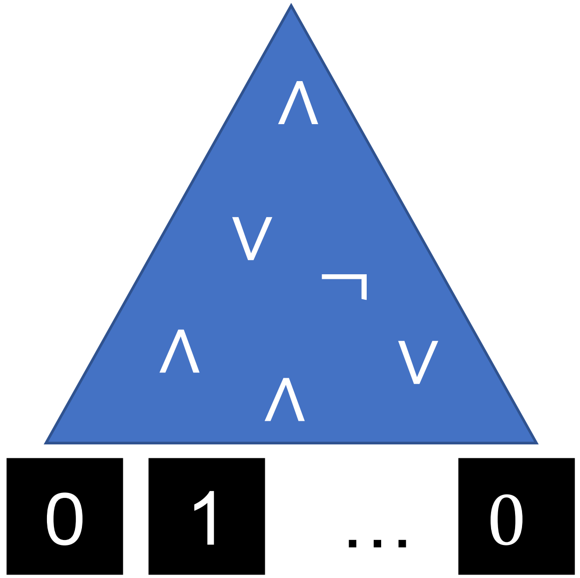
- Choose random initialization S_1
- Do gradient descent (*optional*)
  $S_{i+1} = S_i - \frac{\eta}{T} \nabla_S ||\mathrm{SuNN}(S_i, T, x_i) - y_i||^2$

Thm: Best S is good with 99% probability

In contrast, same proof for learning DNNs:

- Requires exp(T) # repetitions

…so solve $\min_{|S| \leq K} \frac{1}{m} \sum_i ||\,\mathrm{SuNN}(S, T, x_i) - y_i||^2$



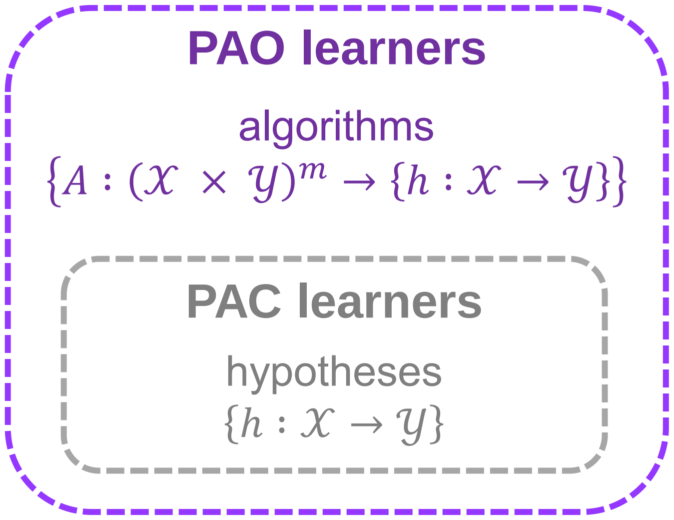
Framework: *PAO*-learning & *Turing-optimality*

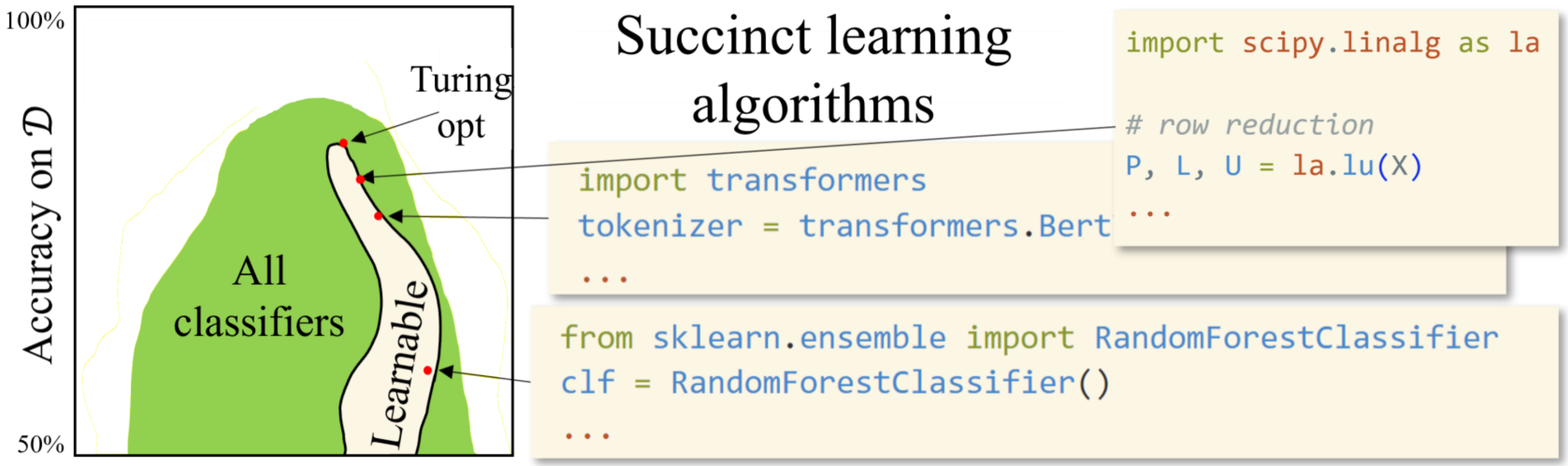
Probably Algorithmically Optimal (PAO) learning:

Poly-time learning algorithm L is a PAO-learner for a family of algorithms $\mathcal{A}$ if, for any distribution $\mathcal{D}$, with high probability:

$$\mathrm{err}_{\mathcal{D}}(L(\text{training \& validation data})) \le \min_{A \in \mathcal{A}} \mathrm{err}_{\mathcal{D}}\big(A(\text{training data})\big) + \epsilon$$

Turing-optimality: PAO when $\mathcal{A}$ = {bounded Turing machines}

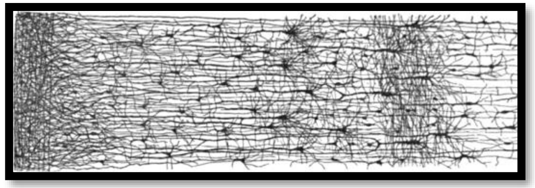




PAC-Agnostic [V'84;KSS'94] vs Prob-Appx-Optimal

Def. Poly-time L **agnostic learns** family $\mathcal{C}_n$ of classifiers if for all $\epsilon, \delta \in [0,1], n \in \mathbb{N}, m \geq p\left(\frac{n}{\epsilon\delta}\right), \mathcal{D} \in \Delta(\mathcal{X}_n \times \mathcal{Y})$:

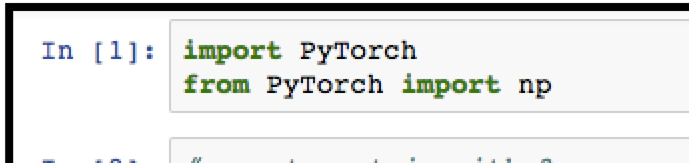
$$\Pr_{Z \sim \mathcal{D}^m}\left[\mathrm{err}_{\mathcal{D}}\big(L(Z)\big) \leq \min_{C \in \mathcal{C}_n} \mathrm{err}_{\mathcal{D}}(C) + \epsilon\right] \geq 1 - \delta$$



Def. Poly-time L **PAO-learns** class $\mathcal{A}$ of learners if for all $\epsilon, \delta \in [0,1], m, n \in \mathbb{N}, \mathcal{D} \in \Delta(\mathcal{X}_n \times \mathcal{Y})$:

$\forall m' \geq q\left(\frac{mn}{\epsilon\delta}\right)$

$$\Pr_{\substack{Z \sim \mathcal{D}^m \\ Z' \sim \mathcal{D}^{m'}}}\left[\mathrm{err}_{\mathcal{D}}\big(L(Z \qquad)\big) \leq \min_{A \in \mathcal{A}} \mathrm{err}_{\mathcal{D}}\big(A(Z)\big) + \epsilon\right] \geq 1 - \delta$$

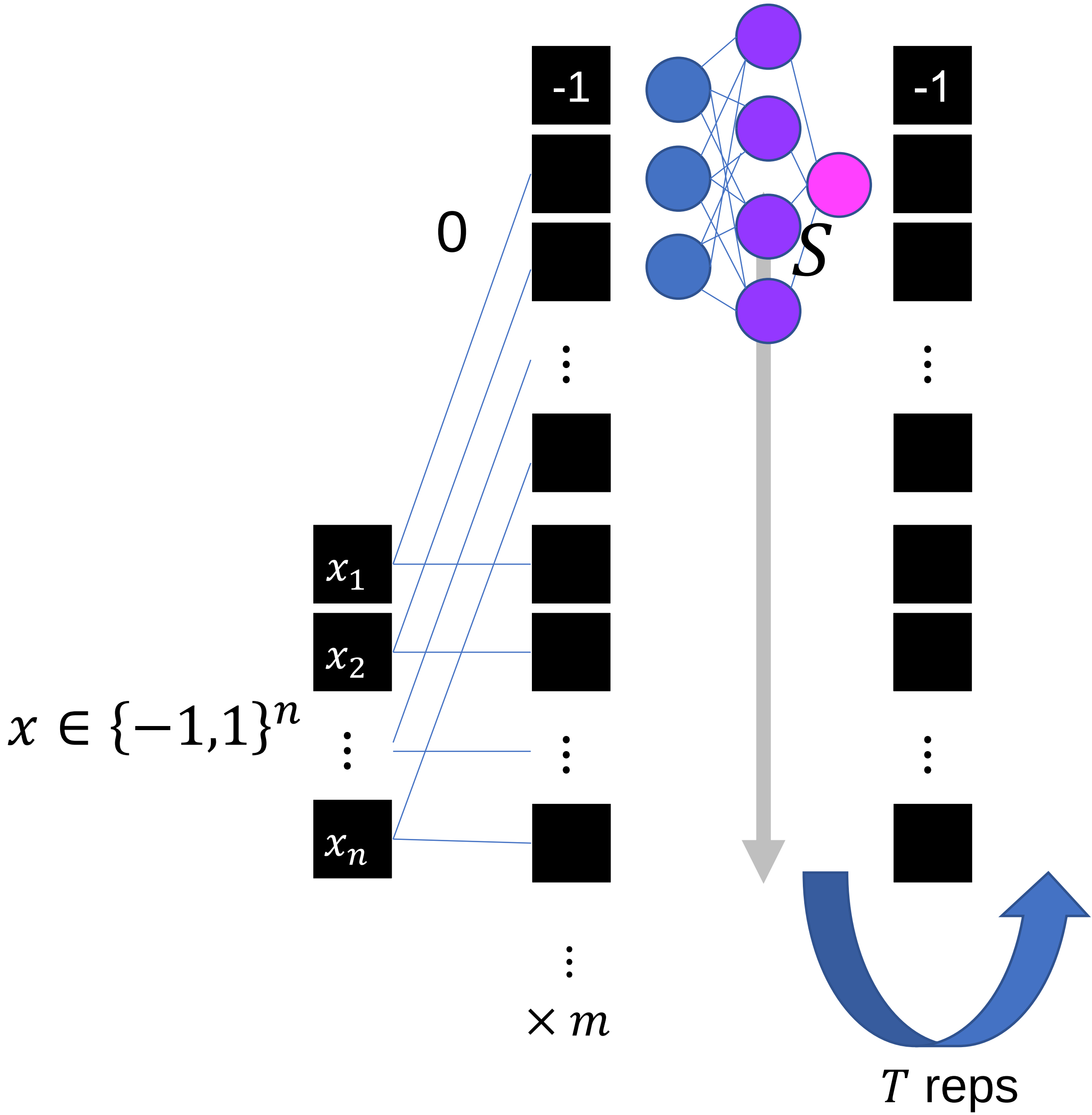


SuNN's with memory are Turing Optimal

Add additional "memorization" layer

Use "trick" due to Abbe & Sandon (2020)

- SGD on first m examples memorizes the m examples in the first layer's weights!
- Surprising: SGD is not Stat. Query
- Afterwards, the TM is run on the examples



Summary & future work

- **PAO & Turing-optimality:** theoretical grounding for algorithm learning
  - Captures computational universality of a deep learning pipeline
  - Instead of classifiers, looks at algorithms which output classifiers
  - **Open:** efficient PAO algorithms for other restricted algorithm classes $\mathcal{A}$?
- **Learning alg’s** (or learning learning alg’s) rather than classifiers
- **Complexity theory** using large language models rather than enumeration?
- **SuNN architecture:** concise neural encoding of programs
  - Recurrent & convolutional weight sharing ↔ parameter-efficient computation
  - Standard training suffices to enumerate over programs
  - **Open:** does SGD work? Better ideas?