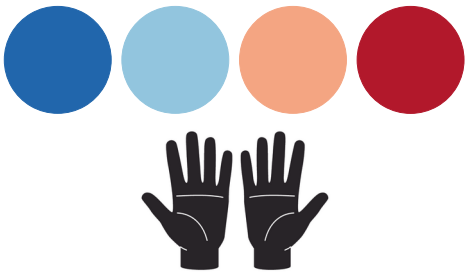
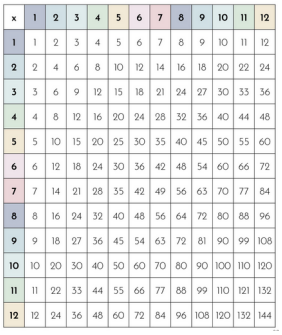
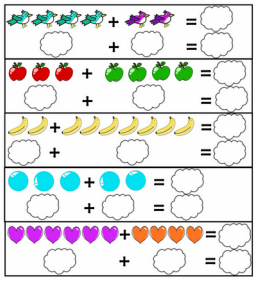
Rational cascades in strategy discovery

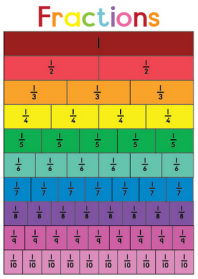
The case of early addition

Merrick Giles
Andrew Perfors
Francis Mollica









Because maths is so useful to us, we've deemed it absolutely necessary to pass on mathematical knowledge. Before we get to school, we pick up some of it from informal interactions with out parents, and by the time we get to school, we're bombarded from every direction with pretty, colourful posters like these; the mathematical knowledge we do gain is heavily guided by the adults around us.

We are the teachers

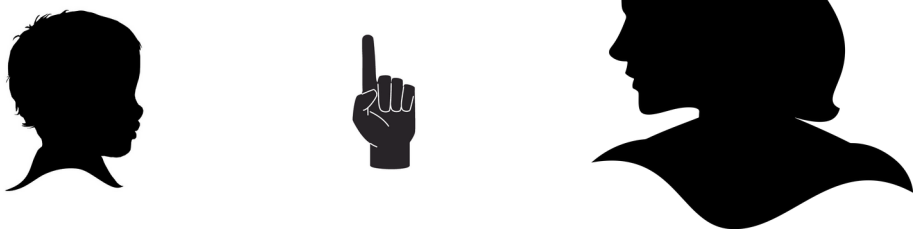
2 + 4

This is how
its done.

And our first experience with addition is no exception. – it's completely driven by our teachers. For our first time adding, they sit us down and show us how its done. And it turns out –across studies -- that there's a strategy teachers particularly like to show us. They'll sit us down and show us this – let's go through together and show this kid how to count.

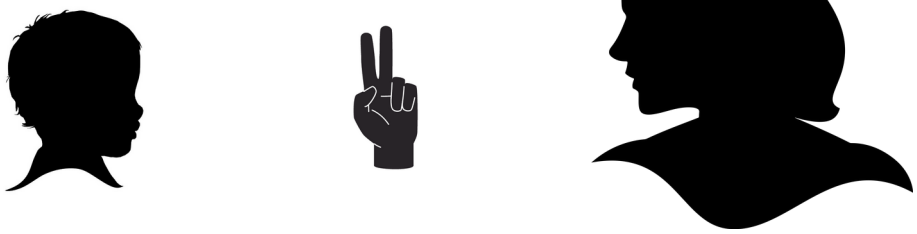
We are the teachers

2 + 4



We are the teachers

2 + 4

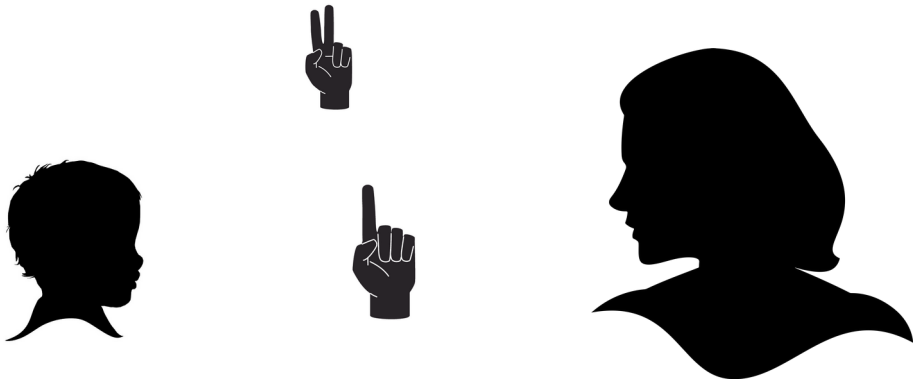


We are the teachers

2 + 4

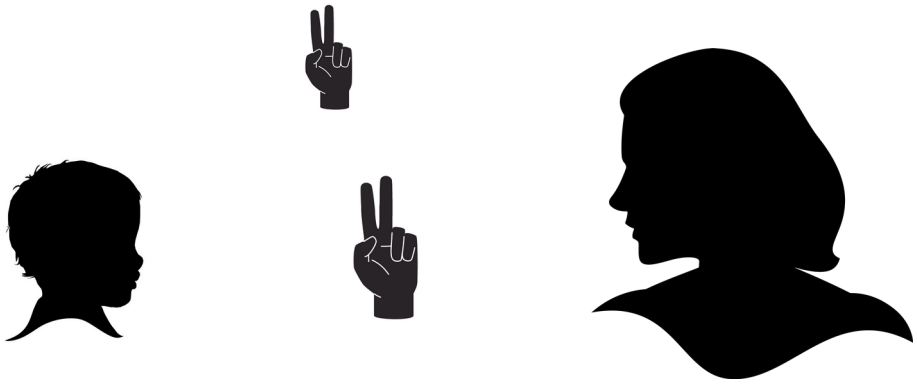
We are the teachers

2 + 4



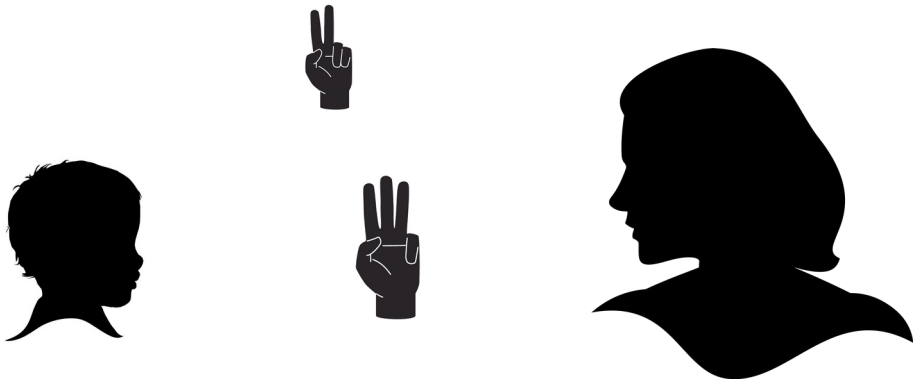
We are the teachers

2 + 4



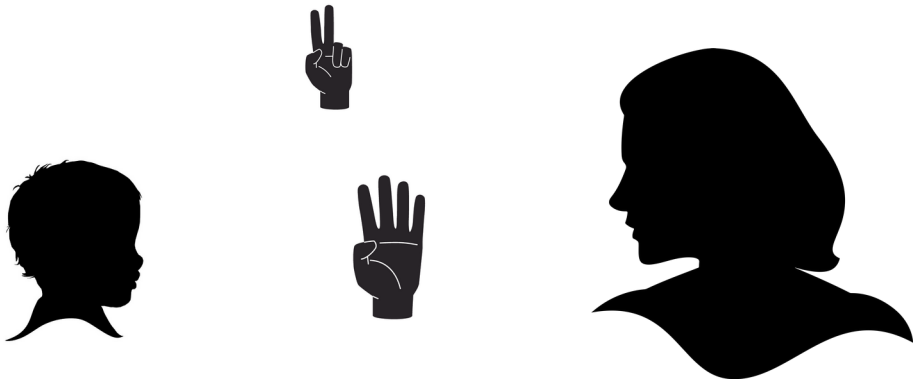
We are the teachers

2 + 4



We are the teachers

2 + 4

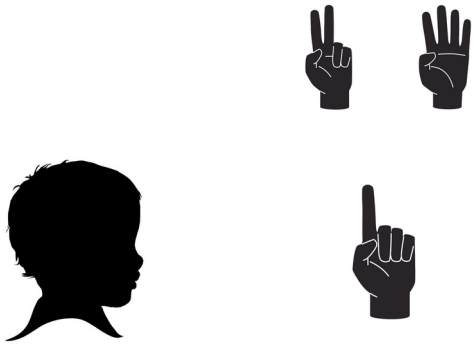


We are the teachers

2 + 4

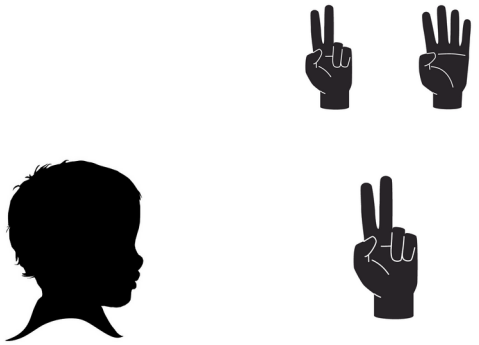
We are the teachers

2 + 4



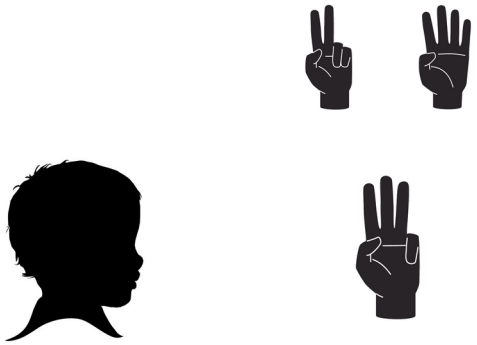
We are the teachers

2 + 4



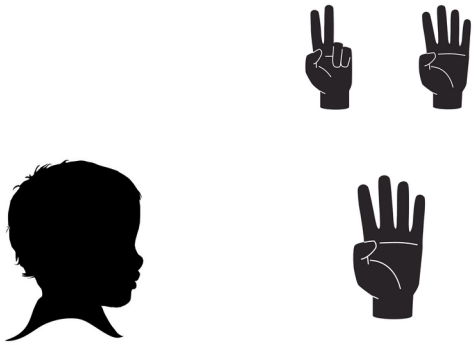
We are the teachers

2 + 4



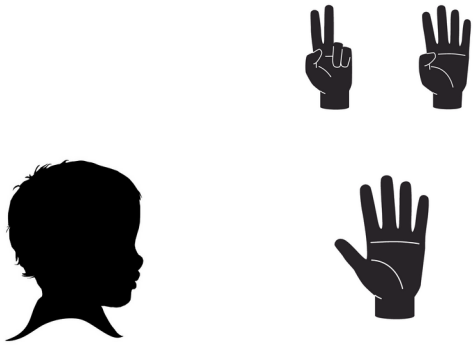
We are the teachers

2 + 4



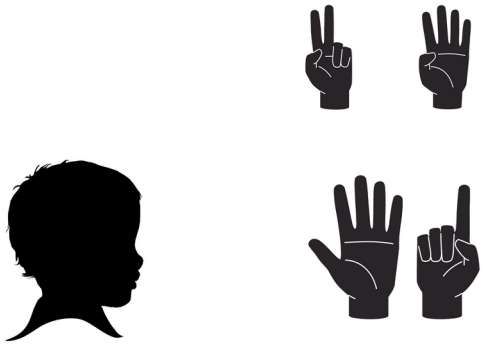
We are the teachers

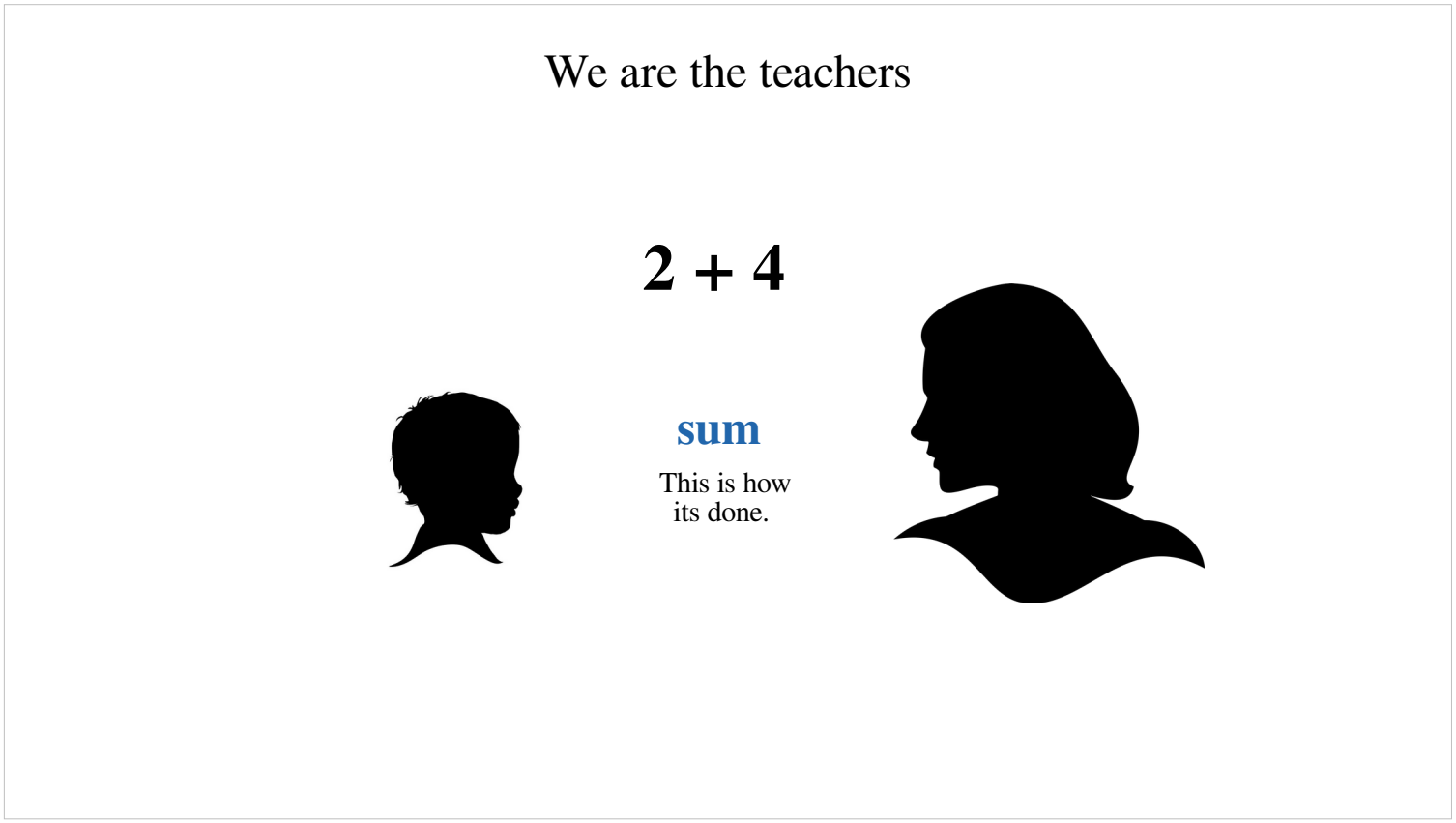
2 + 4



We are the teachers

2 + 4





So the strategy we teach children is called the ‘sum’ strategy. And this is a really neat one because all it is is just repetitions of counting from zero – which children know because we just taught them how to count.

We are the teachers … right?

2 + 4

You're not the boss of me.

But then, despite all of this being driven by the adults, something weird happens. We leave children on their own to practice for a while, and they come back to us with a strategy that's completely different from the one we taught them.

We are the teachers … right?

2 + 4

So here’s what they do.

We are the teachers … right?

2 + 4

That one is bigger

We are the teachers … right?

2 + 4

We are the teachers … right?

2 + 4

We are the teachers … right?

2 + 4

We are the teachers … right?

sum

count 0-to-4, count 0-to-2, count 0-to-both (6)

min

count 2-to-4 **if** (2 > 4) **else** count 4-to-2

wait how did you do that

Svenson & Broquist, 1975; Houlihan & Ginsberg, 1981; Ashcraft, 1982; Baroody, 1984; 1987; Kay et al., 1986; Siegler, 1987; Siegler & Jenkins, 1989; Cowan & Renton, 1996; Geary et al., 2004; etc

So what they've done – completely on their own, is invented this much better strategy where they compare two numbers, and use conditional logic to always start from the bigger one – minimising the amount of counting they have to do. Naturally, as teachers we'll wonder how they did that.

We still don't really know

sum

count 0-to-4, count 0-to-2, count 0-to-both (6)

↓

min

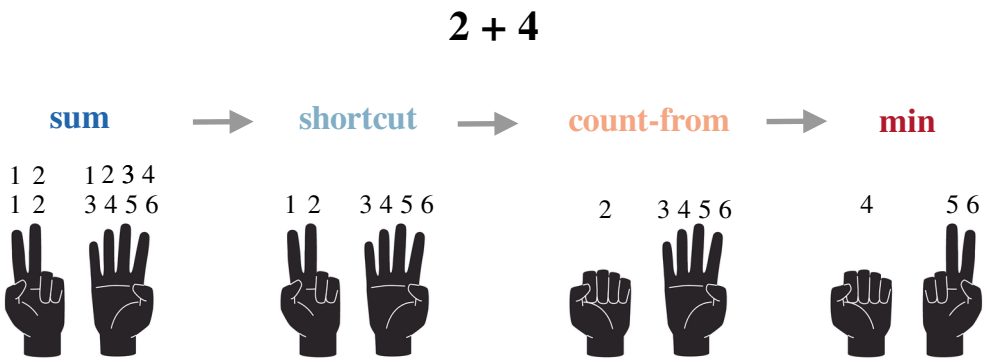
count 2-to-4 **if** (2 > 4) **else** (count 4-to-2)

Yeah how did they do that?

Svenson & Broquist, 1975; Houlihan & Ginsberg, 1981; Ashcraft, 1982; Baroody, 1984; 1987; Kay et al., 1986; Siegler, 1987; Siegler & Jenkins, 1989; Cowan & Renton, 1996; Geary et al., 2004; etc

And as scientists we are also still wondering how they do that – we still don't have a great explanation for it. So that's our job here – let's try explain how they do it

A hidden four-phase learning structure

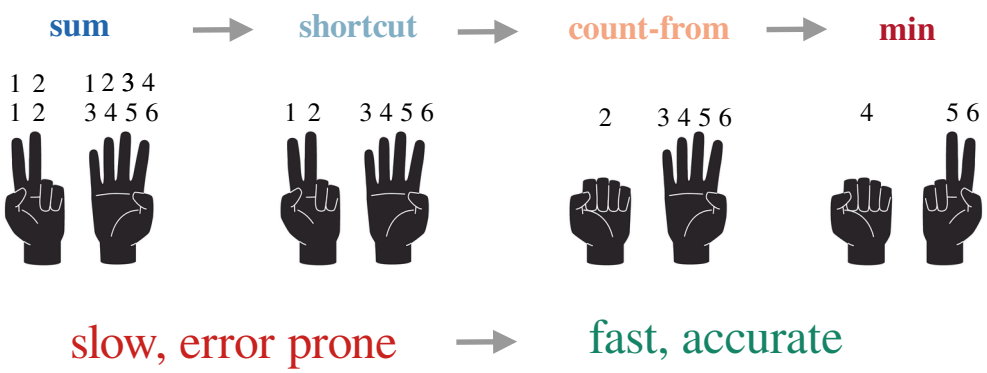


Siegler & Jenkins, 1989

And the first insight we can take comes from work by Bob Siegler and his colleagues: they show that children don't just move directly from sum to min – there are these intermediary stages in between. They start with sum, counting everything twice – then they realise "why am I doing those initial counts of the numerals", and move to shortcut. In this strategy, they just count both numerals continuously from zero. From there, they realise: "why am I counting the first number, I can just start from 2 and go from there". That's the count-from strategy. And finally, they realise that they can save even more effort by always starting from the larger number.

Is this ...rationality ?

2 + 4



slow, error prone → fast, accurate

Siegler & Jenkins, 1989

And, looking at this learning structure we can see that there is some sort of rational updating happening here. In other words, their strategies are getting better over time – they're minimising the amount of required counting, and are thus updating their strategies in the right direction.

Rational models

Let's set up a rational model of addition then!

And given that trajectory of rational updating, we're going to borrowing from a tradition of modelling that conceptualises learning as rational updating.

Rational models

Specify a **Hypothesis Space** of potential addition strategies

Specify a **Prior** over hypotheses

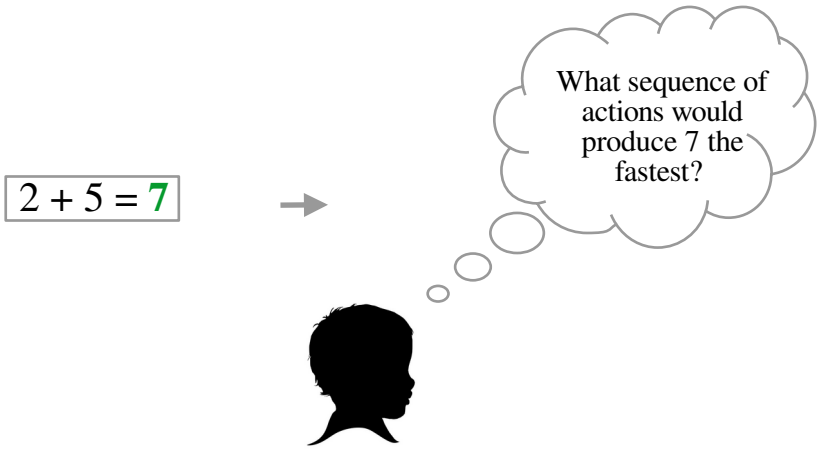
Specify a **Likelihood** function

Specify the environment

Learning: Shifting of probability mass over the hypothesis space as a function of data

And to instantiate our rational learning model of addition, we need a few ingredients. First, we need a hypothesis space of potential addition strategies, we need a prior over those hypotheses, a likelihood function, and finally a learning environment. And we will model learning as the shifting of probability mass over the hypothesis space as a function of data.

Rational Addition model: Learning setup



Rational Constructivism **(Xu, 2019)**

Here's the learning set up. Children complete an addition problem, and receive feedback on the correct answer. The computational problem children solve here is what sequence of actions would produce the observed answer the fastest.

Hypothesis Space

functions	**variables**
• COUNT (X, Y, C)	• A, B
• (X) IF (C) ELSE (Y)	• 0, ..., 9
• COMPARE (X, Y)	• ALL (quantifier)

All are capabilities of children verified by Siegler & Jenkins, 1989

So, to build the hypothesis space we take the minimal set of primitive functions that we know children can perform, and the possible variables they could populate the functions with. So we have count, which specifies the starting value and the number of increments. We have an if-else conditional: Do X if C is true and Y otherwise, and finally we have a compare function: Which takes two numbers and returns the larger one for magnitude comparison.

Hypothesis Space

Construct strategies by composing primitives, setting variables

```
(C) IF (C) ELSE (C)
COMAPARE(X, Y)
COUNT (X, Y, C)
COUNT (X, Y, COUNT(X, Y, C))
COUNT (X, Y, COUNT(X, Y, COUNT(X, Y, C)))
COUNT (X, Y, COUNT(X, Y, COUNT(X, Y, COUNT(X, Y, C))))
(COUNT (X, Y, C)) IF (COMPARE(X, Y)) ELSE (COUNT (Y, X, C))
(COMPARE(X, Y)) IF (COMPARE(X, Y)) ELSE (COUNT (Y, X, COUNT(X
. . .
```

To construct strategies, we take the primitive computations and compose them into fully fledged strategies. So with compositions you can get things like count from X Y times if X is bigger, and count from Y X times otherwise (composing min). The full hypothesis space is the infinite number of possible compositions.

Prior: Simplicity

- COUNT (A, B, C) → High

- COUNT(ALL, B, COUNT(A, B, COUNT(B, A, COUNT(B, ALL, COUNT(A, 9, COUNT(3, 5, COUNT(ALL, 9, C))))))) IF COMPARE(A, B) ELSE (COUNT(B, B, COUNT(A, B, COUNT(3, A, COUNT(1, B, COUNT(2, 5, COUNT(A, B, C))))))) → Low

$$P(h) = \prod_{r \in h} P(r)$$

And to specify a prior probability over the hypothesis space of strategy compositions, we take the product of all primitive computations used in the strategy. Consequently we impose a pressure for algorithmic simplicity: if a strategy uses few primitives, it has a high prior probability. If it uses many primitives, it has a lower prior probability. So we're hypothesising that this algorithmic simplicity is driving strategy choice: simple strategies are easier to learn, remember, and implement.

Likelihood: **Performance**

$$P(s, RT|h, a+b)$$

Accuracy

Penalises for every digit away from correct

Speed

Penalises for every second above zero

$$P(s|h(a+b)) \cdot P(RT|h(a+b))$$

Strategies' simulated errors & response times inspired by **Groen & Parkman, 1972**

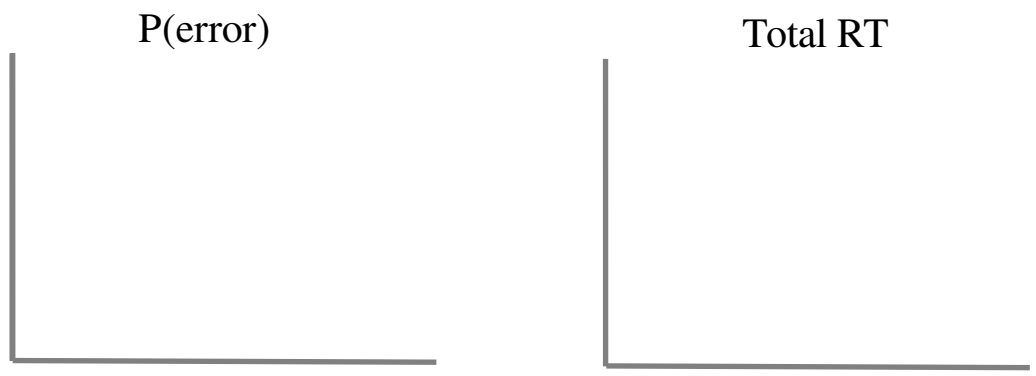
Okay. So we've got our hypothesis space and prior, we now specify our likelihood. And while the prior pressures the learner for simplicity, the likelihood pressures the learner for their best performance. And we quantify performance two components. The first is accuracy. So the strategy generates a solution to a problem, and the accuracy component scores the strategy according to its proximity to the correct solution. So strategies that are closer to the solution have a proportionately higher likelihood.

The second component is speed. Here, the strategy generates some response time to a solution, and strategies are scored according to how fast they are. This functions as a time penalty; for every unit of time, the strategy is losing points, so to speak. So strategies that have a lower response time are given a higher likelihood.

The count function as the main source of performance

2 + 4

COUNT (0, 4, C)



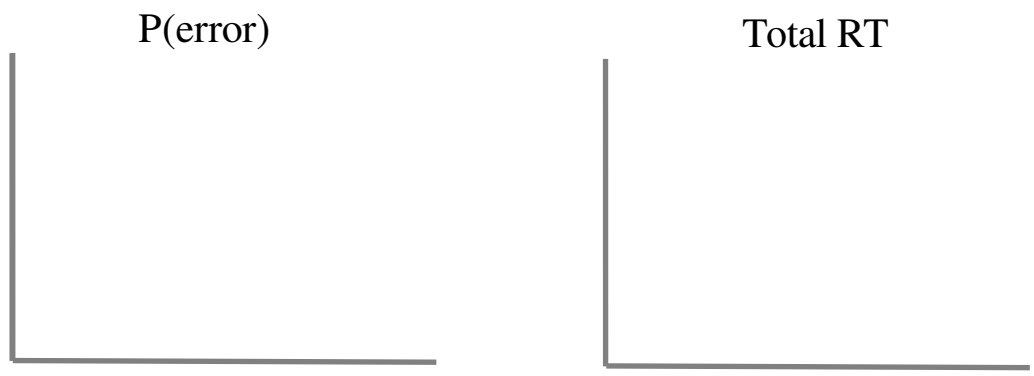
So I'll demo how the strategies are evaluated in the likelihood; the magnitude comparison step has a few tricks itself, but most of the impact comes from this primitive 'count' computation that is used in the strategies. So here we have count four times, starting from two.

The count function as the main source of performance

2 + 4

COUNT (0, 4, C)

0



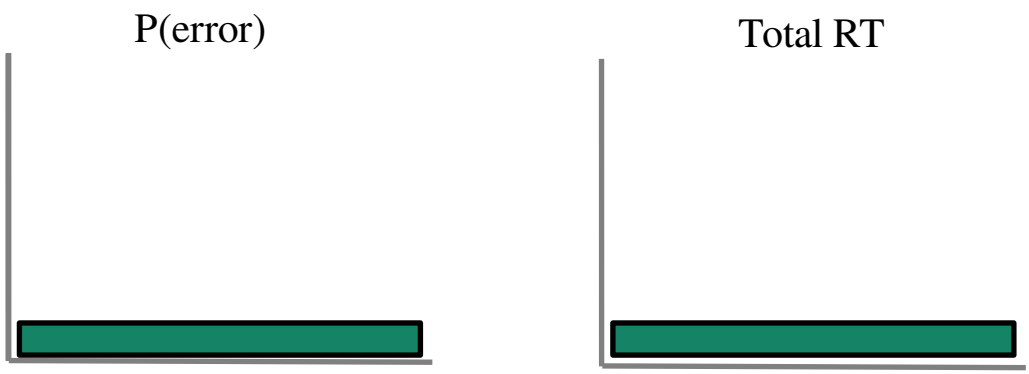
So we’re starting the incrementing at two.

The count function as the main source of performance

2 + 4

COUNT (0, 4, C)

1



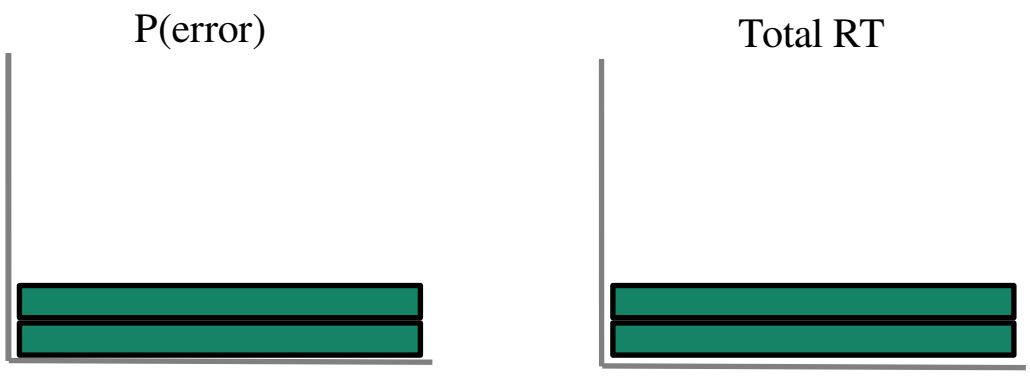
We increment once to three.

The count function as the main source of performance

2 + 4

COUNT (0, 4, C)

2



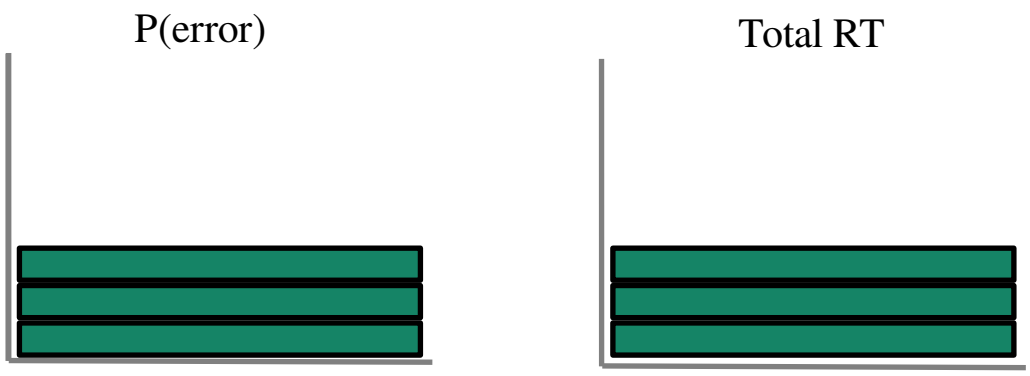
Twice to four (and continue)

The count function as the main source of performance

2 + 4

COUNT (0, 4, C)

3

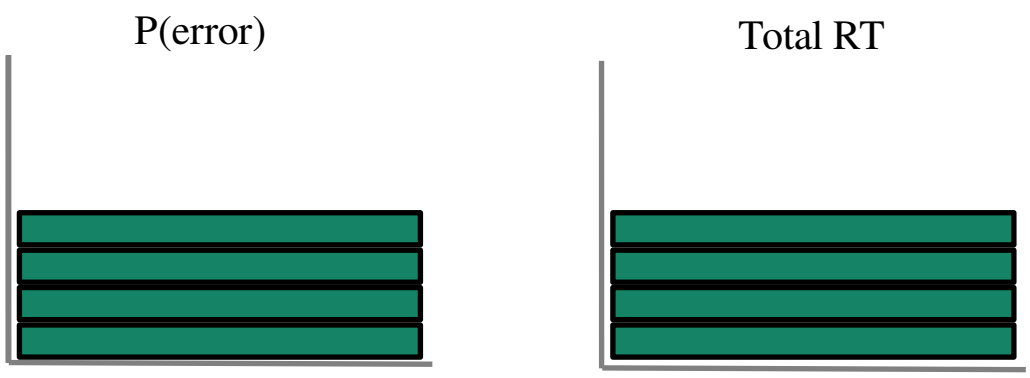


The count function as the main source of performance

2 + 4

COUNT (0, 4, C)

4

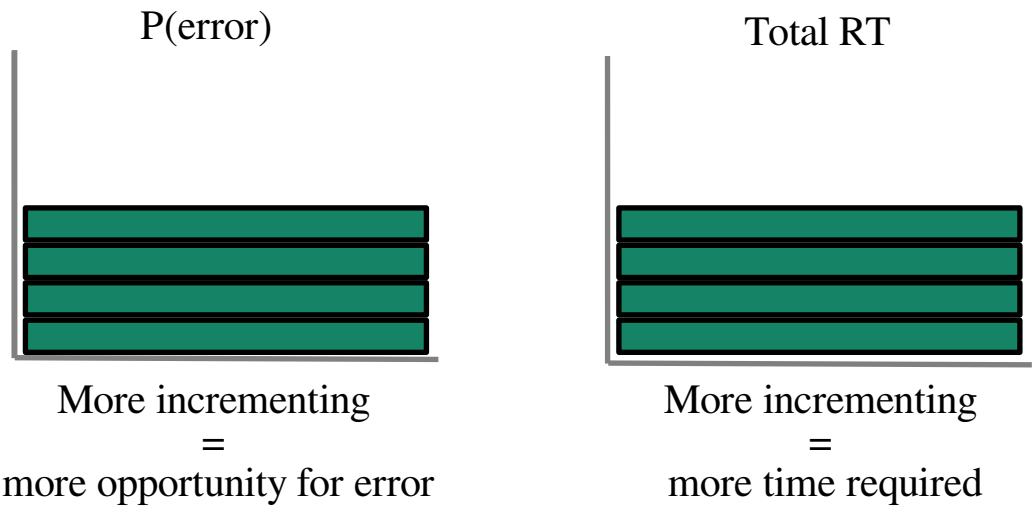


The count function as the main source of performance

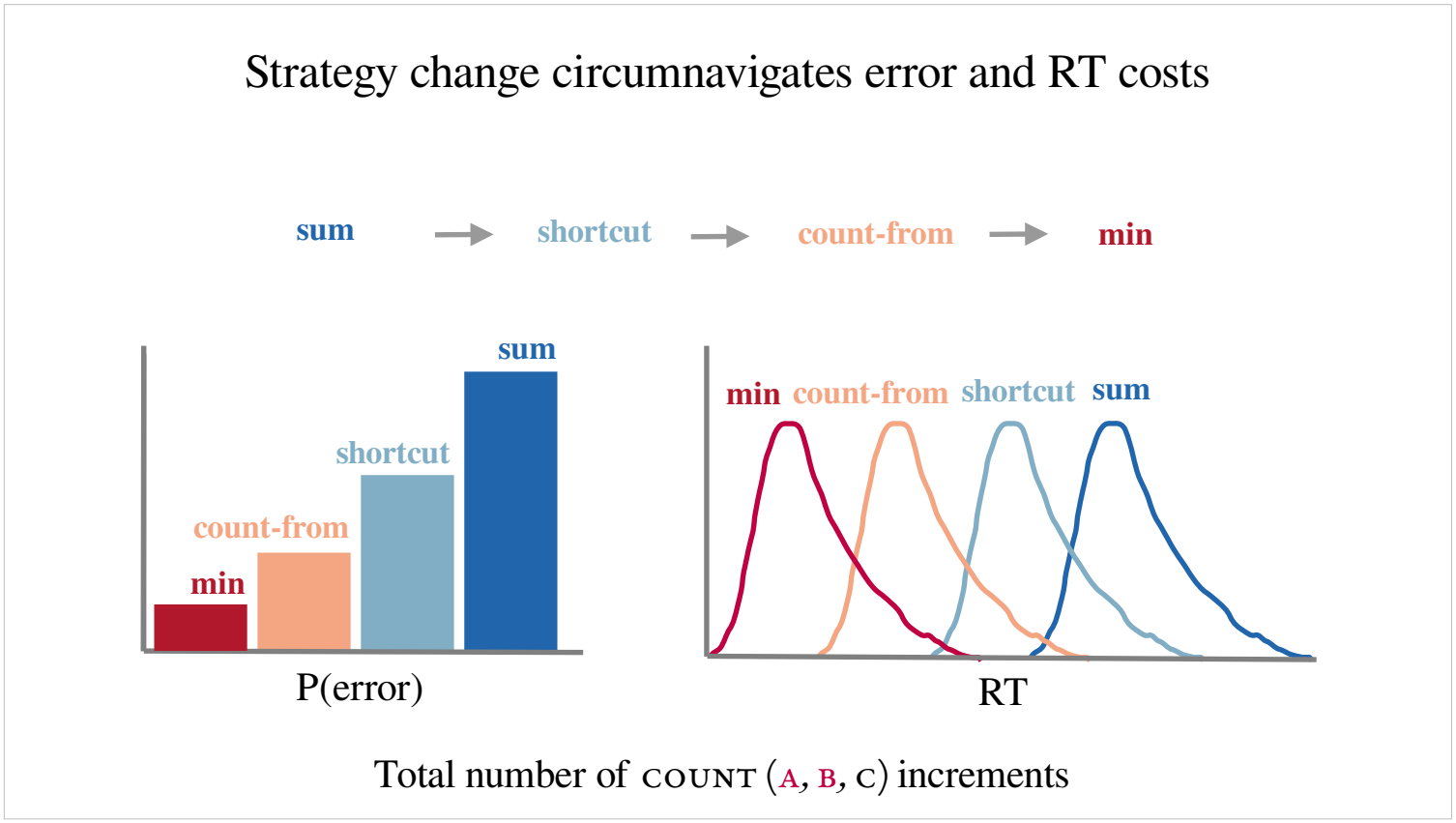
2 + 4

COUNT (0, 4, C)

4



So the more increments that we do, the more errors we will have and the longer the response times we will have. This is because incrementing of course takes time, and because it provides opportunity for mistakes and miscounts.



And, when we operationalise performance in this way: as increments being the source of error and longer Rts, we can see the relative performance of the different strategies. Min requires the least amount of incrementing – you only ever increment the smallest number. While sum requires you to increment over everything twice. Thus, sum will have the highest errors and the longest RTs, while min will have the lowest errors and shortest RTs, with the intermediaries being where they should be.

The change in strategy is reducing error and RT.

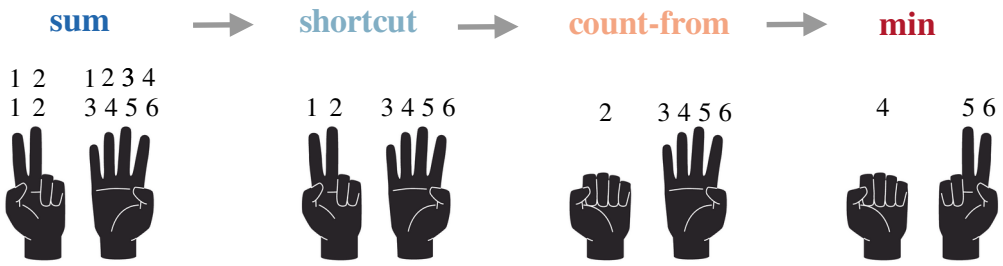
Environment: Siegler & Jenkins, 1989

1) Easy (60)
Problems with both addends < = 5

2) Mixed (unbounded)
67% easy problems
26% addends <= 10
7% 'challenge' problems (3 + 22)

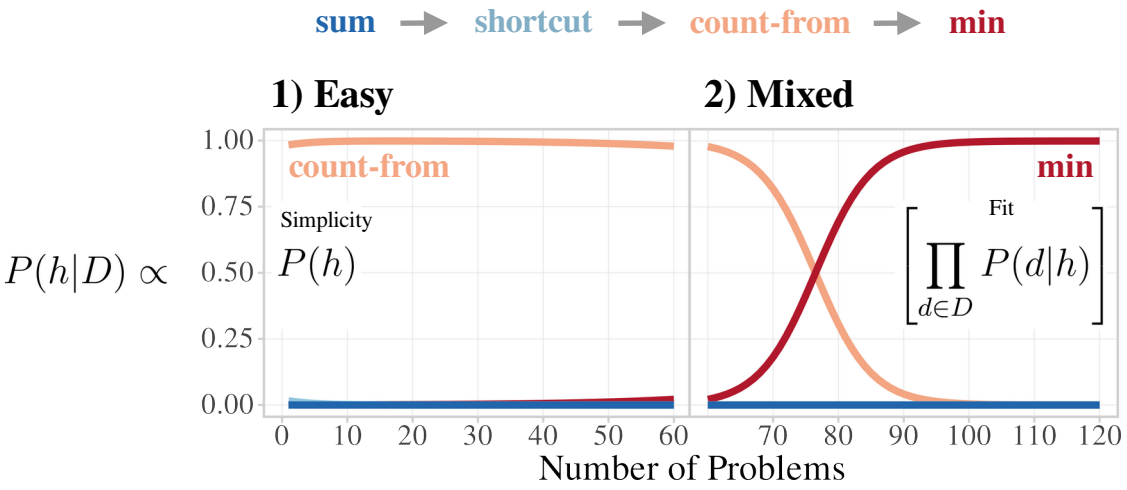
Okay. We've got our hypothesis space and prior, and we've got our likelihood which penalises error and RT and thus privileges some strategies over others – now it's time to turn to the environment where we'll run our model. For this, we mirror the environment used in Siegler and Jenkins' classic study. Children start out by practicing on 60 super easy problems where the numerals are < = 5. But then, children move into a more complicated environment: 67% are easy problems (we don't want them to get discouraged), but 26% are harder with numerals less than 10 – 10 + 4, and 7% are these challenge problems, with problems like 3 + 22. Siegler actually explicitly introduced these problems to push children to discover min by necessity. But the Siegler-Jenkins dataset broadly models what happens in the real world: children start out on easy and then find themselves in increasingly complicated environments.

Lets go



Okay so we're ready to roll – remember this is what we expect – a shift from sum, to shortcut, to countfrom, and finally to min.

Children hit sum and shortcut thoo



Yikes…

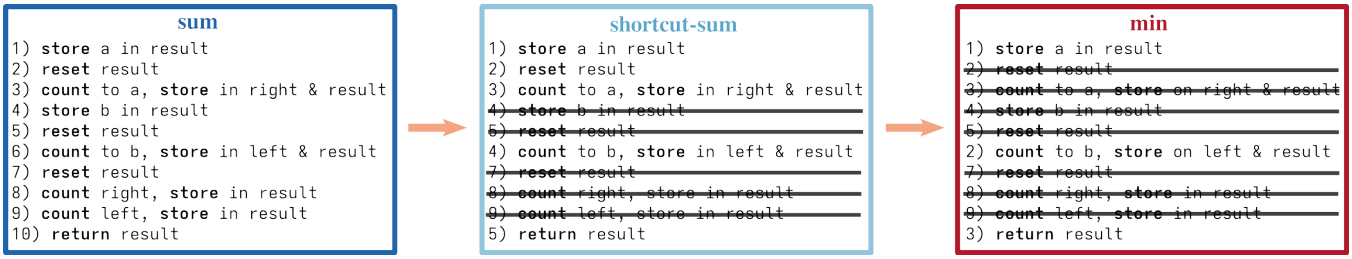
So here's what the model does. It starts out on count-from because it's the most algorithmically simple strategy – hitting the simplicity pressure. The model predicts that children will then transition to min, because it provides a better fit with the speed and accuracy pressures we introduced.

Yikes…

So what do we do? It turns out that our insight isn't actually new, in fact people have been pointing out this kind of thing since they first looked at addition learning…

Metacognition

Shrager & Siegler, 1998



And the way researchers in the past have gotten around this issue is by using asserting metacognitive processes. In the 90s, Shrager and Siegler did this with a production system that deletes actions from it's strategies.

Metacognition

Shrager & Siegler, 1998

sum

```
1) store a in result
2) reset result
3) count to a, store in right & result
4) store b in result
5) reset result
6) count to b, store in left & result
7) reset result
8) count right, store in result
9) count left, store in result
10) return result
```

shortcut-sum

```
1) store a in result
2) reset result
3) count to a, store in right & result
~~4) store b in result~~
~~5) reset result~~
4) count to b, store in left & result
~~7) reset result~~
~~8) count right, store in result~~
~~9) count left, store in result~~
5) return result
```

min

```
1) store a in result
~~2) reset result~~
~~3) count to a, store on right & result~~
~~4) store b in result~~
~~5) reset result~~
2) count to b, store on left & result
~~7) reset result~~
~~8) count right, store in result~~
~~9) count left, store in result~~
3) return result
```

Rule, Piantadosi, Tenenbaum, 2020

```
def sum(a1,a2):
  raise(a1, LeftHand)
  raise(a2, RightHand)
  y = count(LeftHand, 0)
  sum = count(RightHand, y)
  return sum
```

```
def shortcutSum(a1, a2):
  y = raiseCount(a1, LeftHand, 0)
  sum = raiseCount(a2, RightHand, y)
  return sum
```

```
def countFromFirst(a1, a2):
  sum = raiseCount(a2, LeftHand, a1)
  return sum
```

```
def min(a1, a2):
  if a1 > a2:
    return countFromFirst(a2, a1)
  else
    return countFromFirst(a1, a2)
```

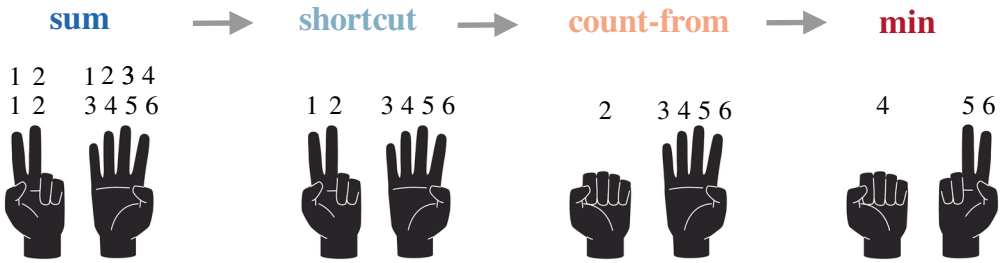
More recently, this has been proposed using term re-writing systems where children metacognitively revise the starting sum strategy.

Downsides

- No evidence for metacognition
- No updating as a function of data
- No environment-driven change

Baroody & Tiilikainen, 2003 point these out too

But we can point out a few issues with this approach. First, there is simply no evidence that children are doing anything metacognitive. As a lot of the empiricists point out, children often have very little insight into the strategies they're using at all. Second, any explanation that puts the blame on metacognition will have trouble explaining how childrens strategies change as a function of more practice. Once these production and re-writing systems get to editing the strategies, there's basically no stopping them. It's not clear why more practice and seeing more data would cause strategy change in these perspectives.



How did they do that?

So we're basically back to the drawing board here. The rational approach won't work, and metacognitive approaches might get some success, but with serious draw-backs.

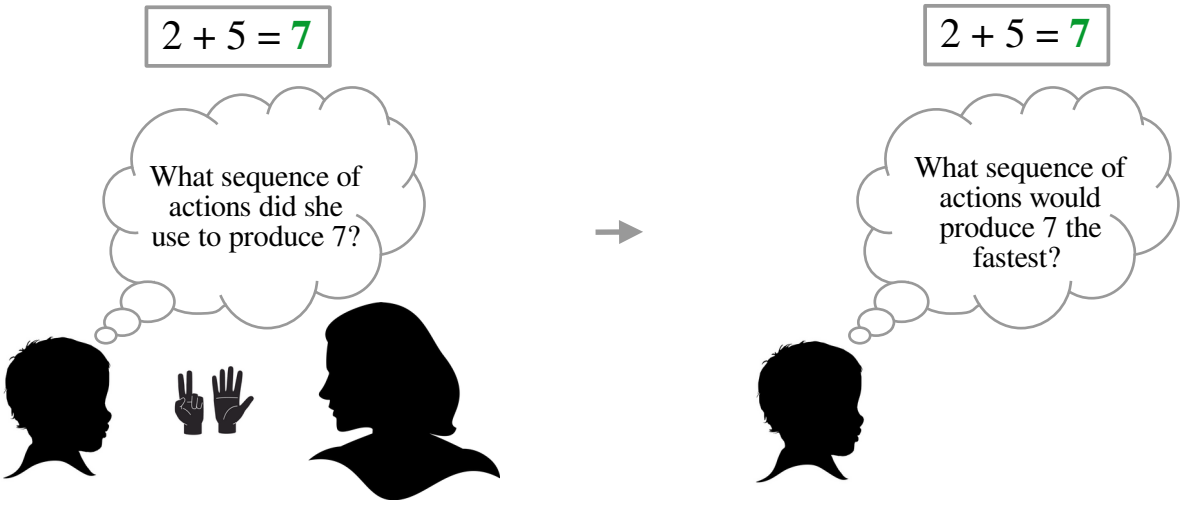
We are the teachers

2 + 4

This is how
its done.

But what we've been doing so far is ignoring one key component that we talked about at the start. We all assume that the beginning sum strategy is somewhat of an arbitrary starting point from which children edit their strategies. But what we can do is frame this teaching process as an integral part of the learning problem.

Learning as a Rational Cascade



Reconstructive Imitation **(Hayes, 2018)** + Rational Constructivism **(Xu, 2019)**

And when we take the teaching phase seriously, we can see that children are actually solving a dual-phased computational problem. Initially, the goal is one where they reconstruct their teachers strategies: What sequence of actions did my teacher use to produce 7? Only later do we get the performance goal: produce 7 as fast as possible.

Learning as a Rational Cascade

$$P(h|D) \propto P(h)$$

Fit to teaching ← $\left[\prod_{d \in D} P(d|h)\right]$ → Fit to speed

Always accuracy

$$P(RT = \text{teachers}|h(a+b)) \qquad P(RT|h(a+b))$$

Reconstructive Imitation **(Hayes, 2018)** + Rational Constructivism **(Xu, 2019)**

We instantiate this using the response time component. In the first phase, the likelihood evaluates strategies based on how well they fit to our teachers' response times: it's a likelihood that scores strategies based on how well they mimick the RT of the teacher, as well as their accuracy. In the second phase, we have our classic speed pressure.

Learning as a Rational Cascade

$$P(h|D) \propto P(h)$$

Fit to teaching ← $\left[\prod_{d \in D} P(d|h)\right]$ → Fit to speed

Always accuracy

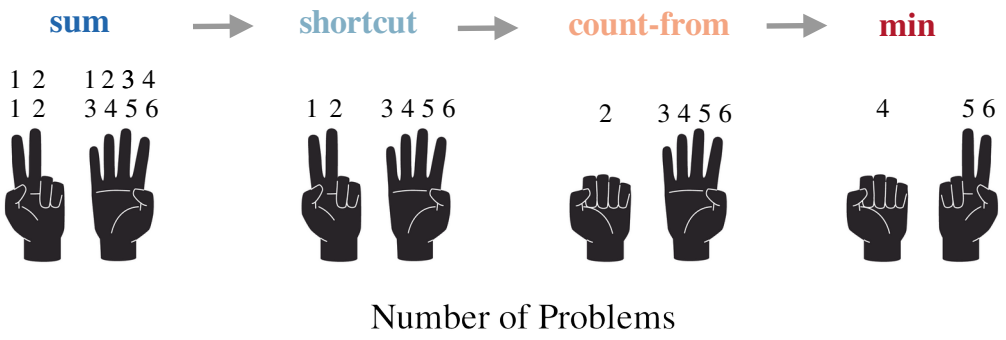
$P(RT = \text{sum}|h(a+b))$ $P(RT|h(a+b))$

Reconstructive Imitation **(Hayes, 2018)** + Rational Constructivism **(Xu, 2019)**

And remember, to fit our strategies to our teachers, we're reconstructing the sum strategy – because that's the strategy our teachers are showing us.

Learning as a Rational Cascade: Running the model

$$P(h|D) \propto P(h) \left[\prod_{d \in D} P(d|h) \right]$$

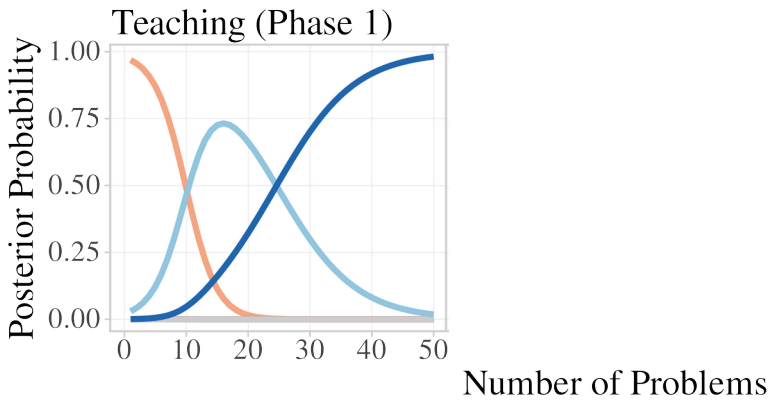


So let's re-run the model with this initial teaching data. Remember our goal is to capture the transition from sum to scs to cf to min following the initial teaching phase.

Learning as a Rational Cascade: Running the model

$$P(h|D) \propto P(h) \left[\prod_{d \in D} P(d|h) \right]$$

sum → **shortcut** → **count-from** → **min**

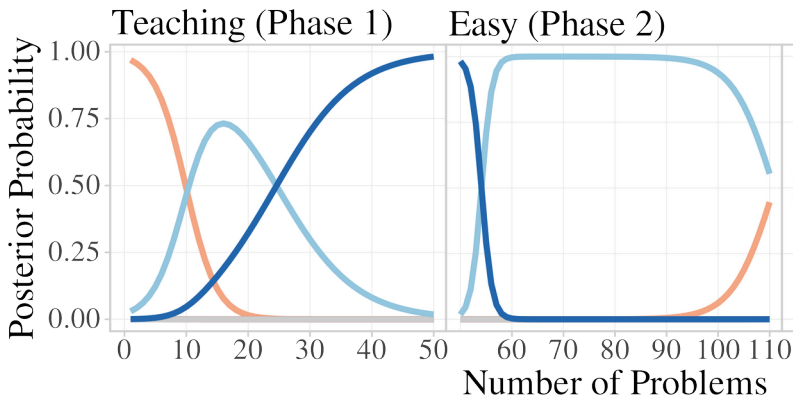


So we use a curriculum where the first phase involves teaching. Remember, the goal here is to fit to the sum data generated by our teachers. Naturally, the posterior of the sum strategy is raised.

Learning as a Rational Cascade: Running the model

$$P(h|D) \propto P(h) \left[\prod_{d \in D} P(d|h) \right]$$

sum → shortcut → count-from → min

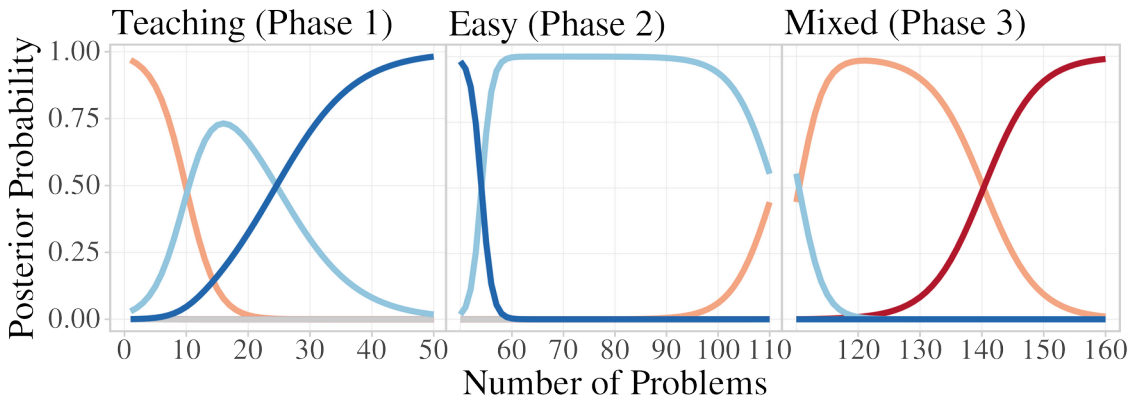


When children are introduced to independent practice though, something interesting happens. The model transitions to shortcut, just like children do. The reason this is happening is because a) shortcut is more algorithmically simple, and is faster and more accurate than sum. However, why not move straight to shortcut sum? The reason is because shortcut provides a reasonable fit to the data children observed during their teaching. So it beats out sum in performance, but also fits reasonably well with the teaching data. The teaching phase has a cascading effect on subsequent discovery: children not only check whether a strategy performs well, they also check whether it occurs with their teaching.

Learning as a Rational Cascade: Running the model

$$P(h|D) \propto P(h) \left[\prod_{d \in D} P(d|h) \right]$$

sum → **shortcut** → **count-from** → **min**

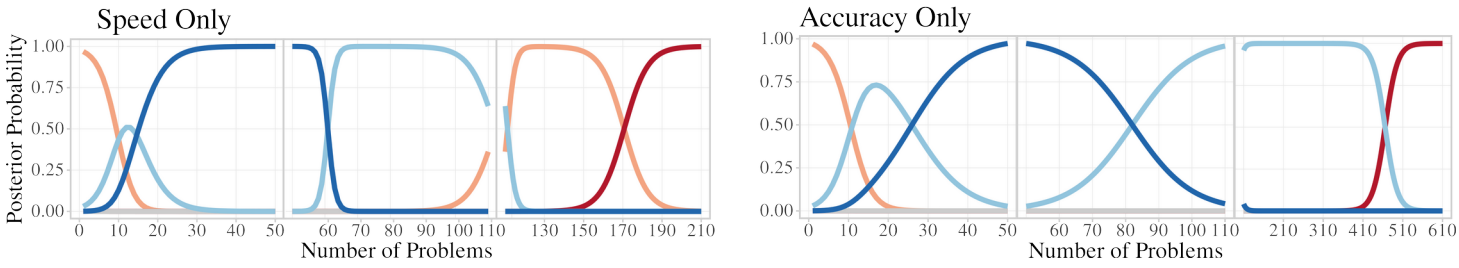


At the end of the easy phase, the model moves on to discover count from, given its advantages in algorithmic simplicity, and its performance benefits. While min is more algorithmically complex, with its comparisons and conditionals, the posterior mass eventually shifts to min because it's performance gains offset its complexity.

Learning as a Rational Cascade: Running the model

$$P(h|D) \propto P(h) \left[\prod_{d \in D} P(d|h) \right]$$

sum → **shortcut** → **count-from** → **min**

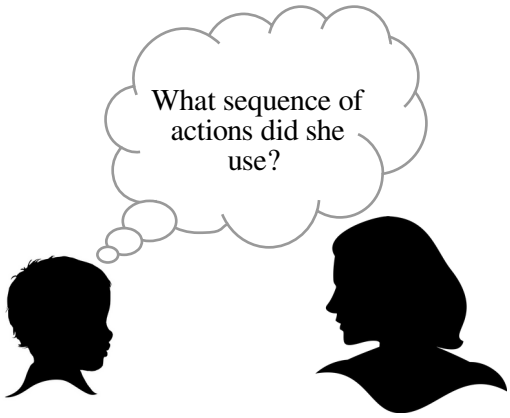


And an especially cool result of this is that it does not matter which pressure – speed or accuracy – is emphasised. This is because they're both instantiations of the same underlying pressure: the number of increments. More increments means more opportunity for error and a longer response time, so we get the strategy improvement whether we emphasise either one.

Learning as a Rational Cascade: Lessons

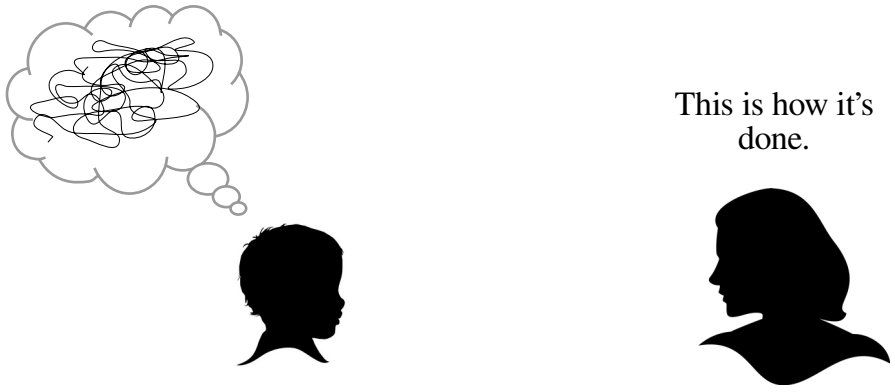
So let’s end with a few brief broader lessons.

Learning, observation, imitation, instruction



The first is that, in strategy learning, we rarely come to the problem fom no where – even in the absence of any pedagogical inferences, this demonstration shows us the importance of building in these contexts in strategy and problem solving. Our first exposures to a problem give us insight into the strategy space we're searching, and provides evidence towards the permissible strategies that we can then fit new strategies to during independent learning. So we need to model the full context of the environment when we aim to explain some strategic behaviour.

Intrinsic vs. extrinsic causes



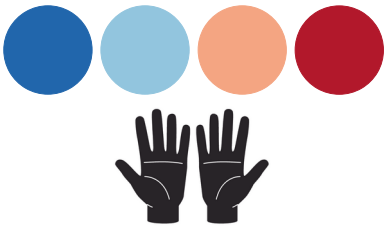
Horne, Kobaş, & Cimpian, 2025

And to finally end with a provocation. As shown by Horne and his collegues, early scientific explanations often look to intrinsic causes to explain our observations. As well as its historical precedence, this also seems to be a general bias of human cognition.

But this case study is a demonstration that it's equally important to look to the external, environmental causes to explain the weird patterns of behaviour we see – because the environment can influence behaviour in unexpected ways. In my view we should always look to see if the environment can explain our observations, before positing internal mechanisms like metacognition.

Questions?

Rational cascades in strategy discovery
The case of early addition



Merrick Giles
Andrew Perfors
Francis Mollica

CogSci paper