A Functional Perspective on Machine Learning via Programmable Induction and Abduction

Steven W. T. Cheung
Victor Darvariu
Dan R. Ghica
(University of Birmingham)

Reuben N. S. Rowe
(University of Kent)

Koko Muroya
(University of Birmingham
& RIMS, Kyoto University)

FLOPS (Nagoya), 9 May 2018

Machine Learning

https://xkcd.com/1838/

Machine Learning

modelising

https://xkcd.com/1838/

Machine Learning

using / predicting

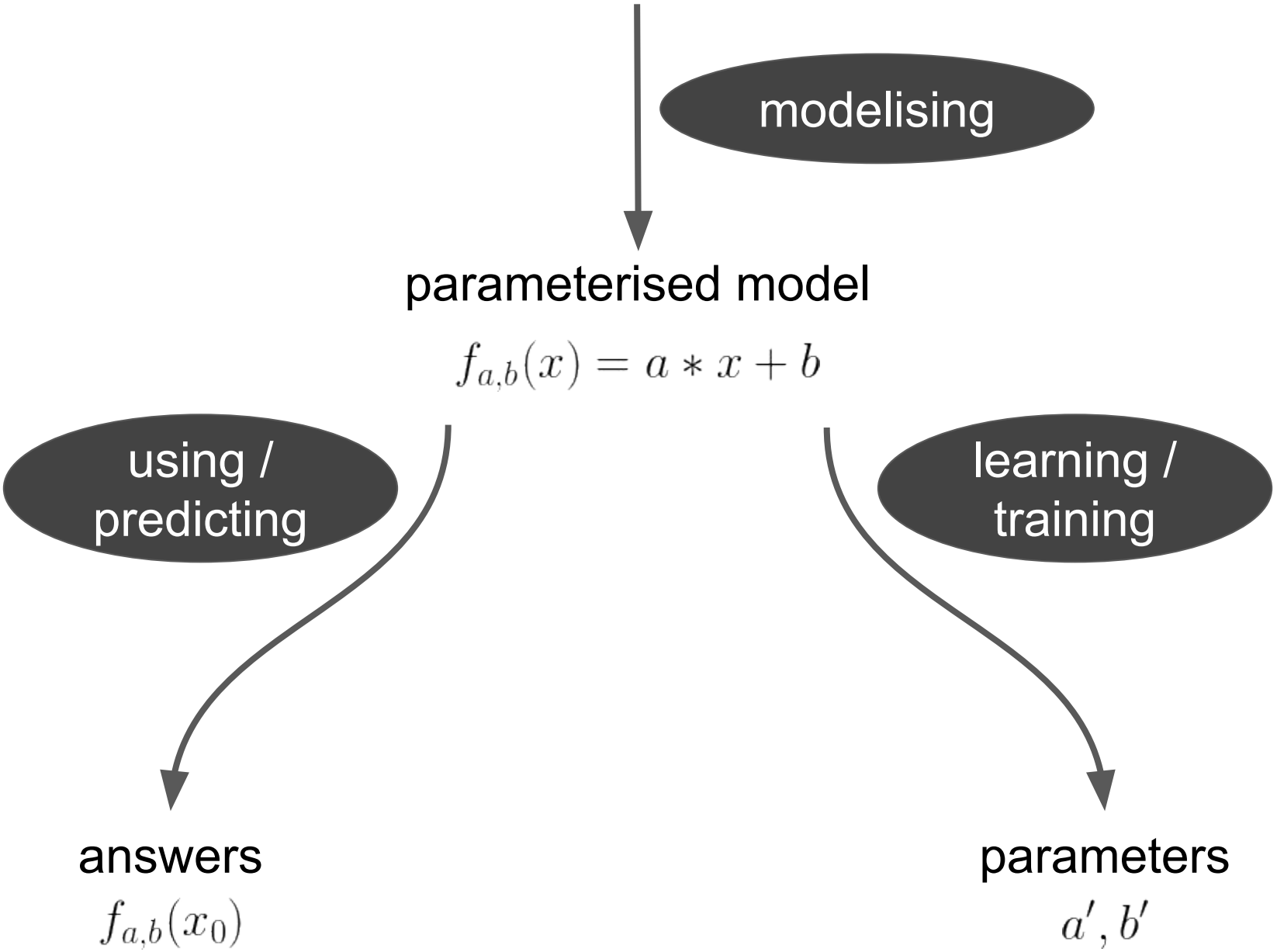
https://xkcd.com/1838/

Machine Learning

learning / training

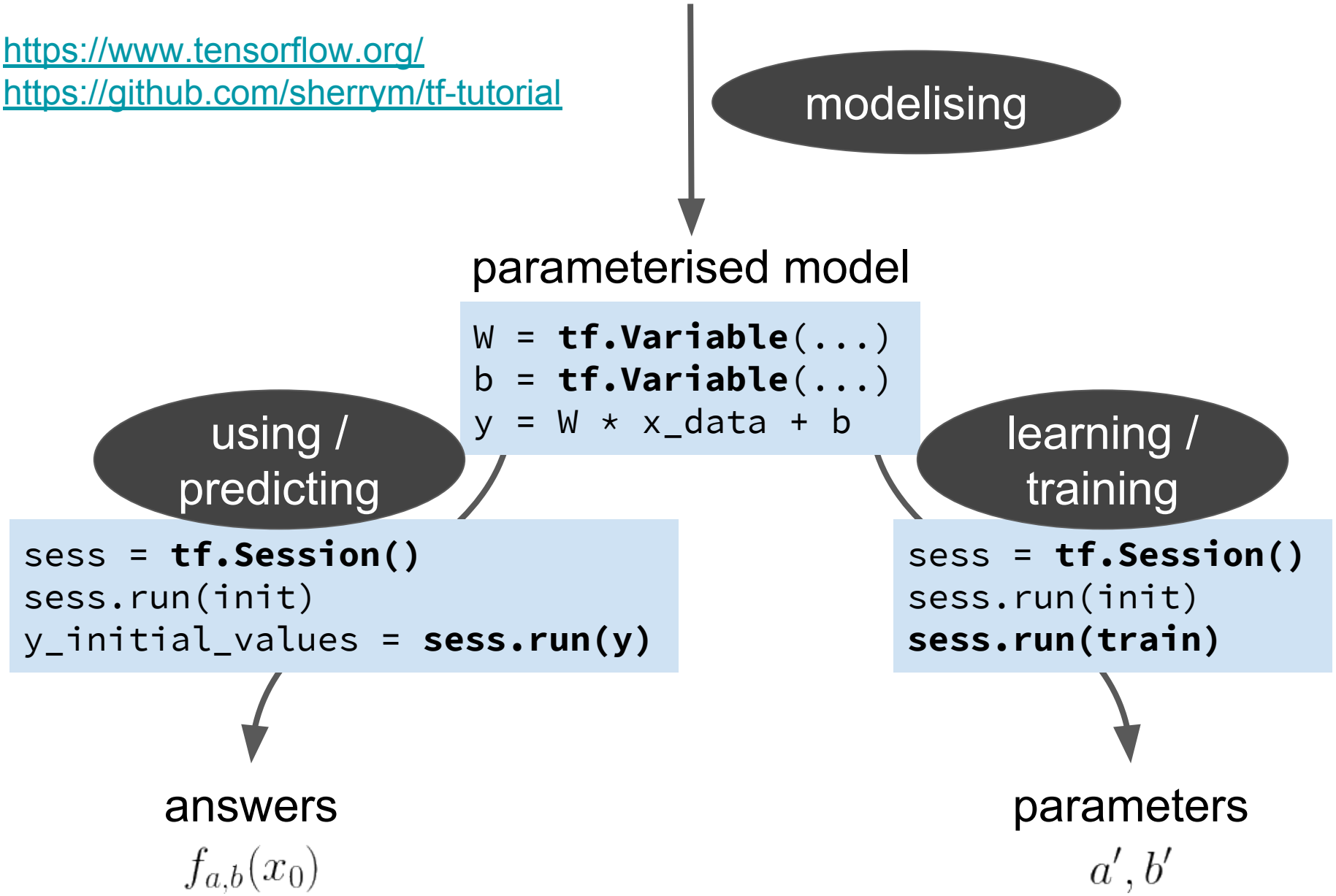
https://xkcd.com/1838/

Machine Learning



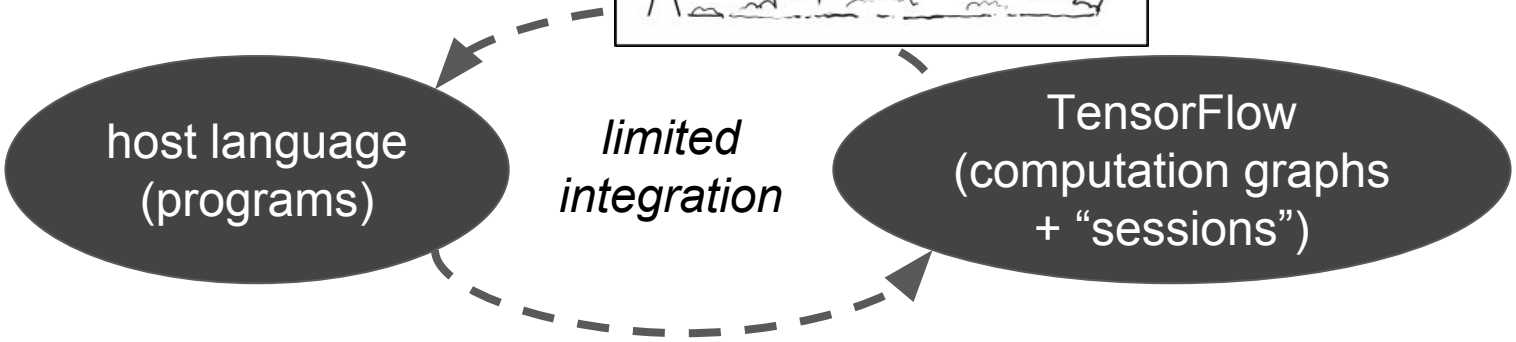
Machine Learning with TensorFlow

https://www.tensorflow.org/
https://github.com/sherrym/tf-tutorial



TensorFlow

- shallow embedded DSL

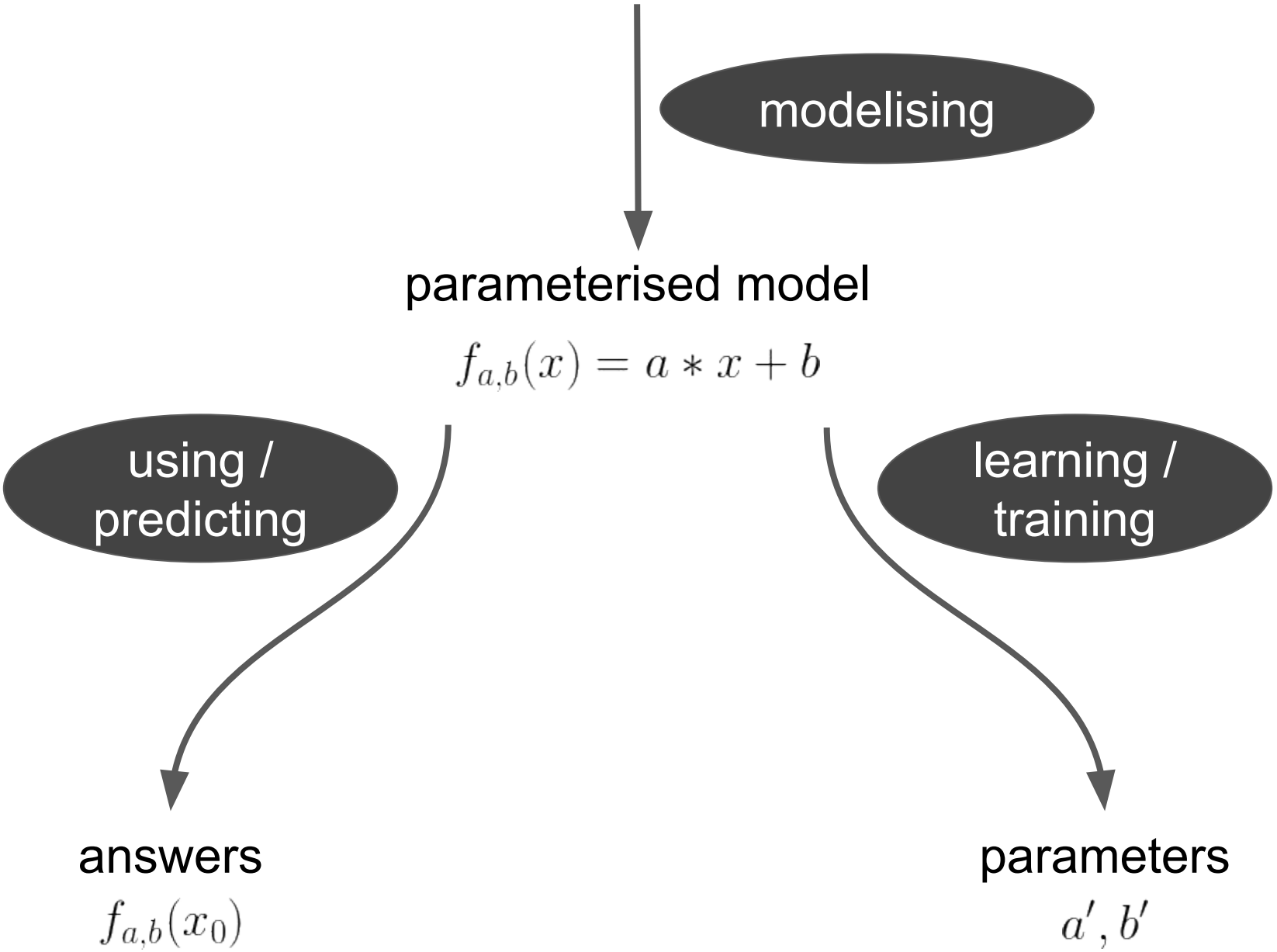


- imperative parameter (“variable”) update

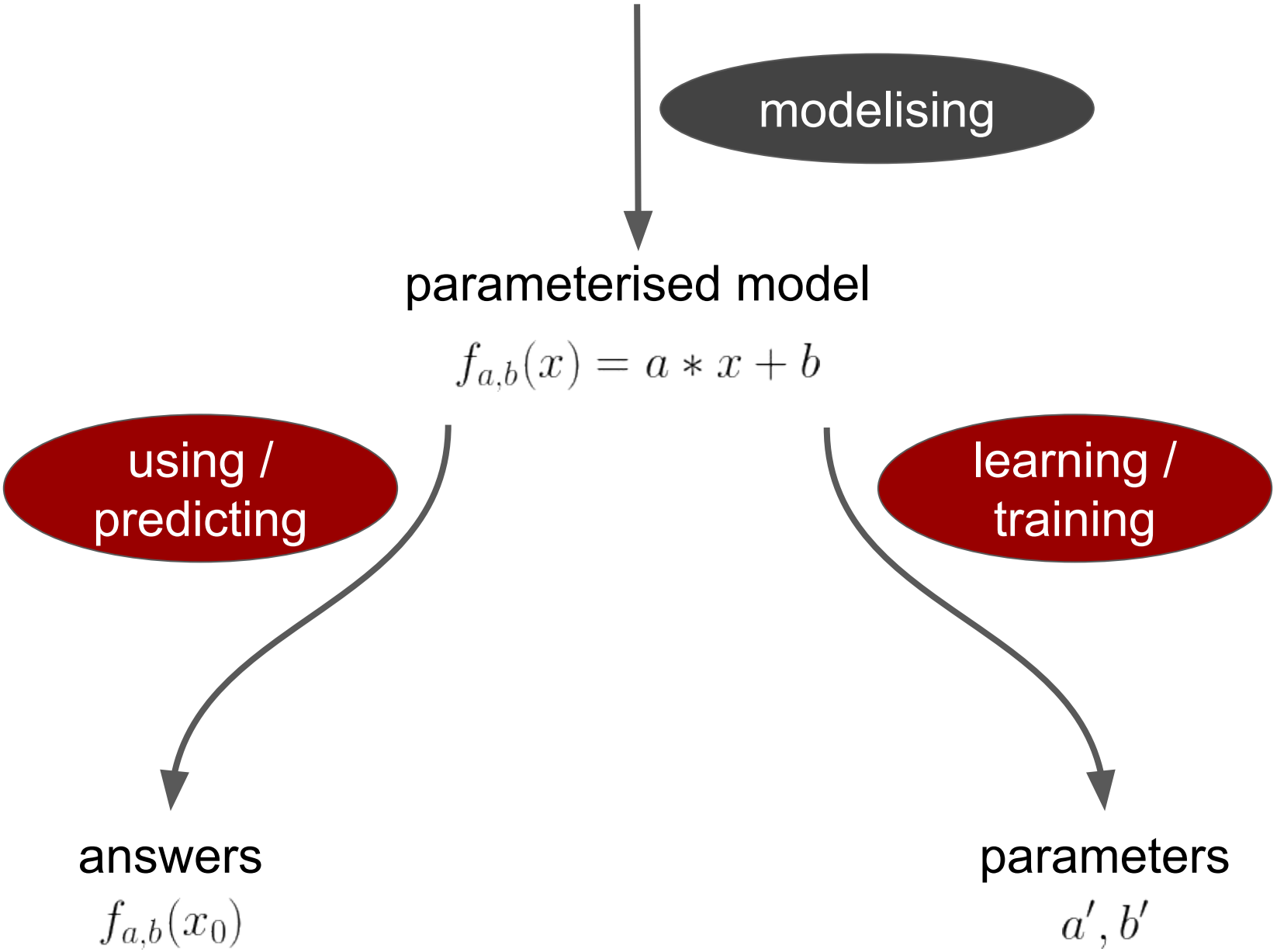
Proper *functional* language?

- simple & uniform programming language
  - well-defined operational semantics
- functional parameter update

Machine Learning



Machine Learning



functional programming language for **using & learning**

realisability style
"proofs as programs"

logical framework of **science methodology**

C. S. Peirce "Illustrations of the Logic of Science"

Peirce's Logical Inferences

- *analytic* rule, believable results

- *synthetic* rules, tentative results

Peirce's Logical Inferences

- *analytic* rule, believable results
  - deduction

$$\frac{A \to B \qquad A}{B}$$

- *synthetic* rules, tentative results

Peirce's Logical Inferences

- *analytic* rule, believable results
  - deduction

$$\frac{A \to B \qquad A}{B}$$

- *synthetic* rules, tentative results
  - induction

$$\frac{A \qquad B}{A \to B}$$

Peirce's Logical Inferences

- *analytic* rule, believable results
  - deduction

$$\frac{A \to B \qquad A}{B}$$

- *synthetic* rules, tentative results
  - induction

$$\frac{A \qquad B}{A \to B}$$

  - abduction

$$\frac{A \to B \qquad B}{A}$$

Peirce's Logical Inferences, *adapted to ML*

- *analytic* rule, believable results
  - deduction

$$\frac{A \to B \qquad A}{B}$$

- *synthetic* rules, tentative results
  - induction

$$\frac{A \times B \;\cdots\; A \times B}{A \to B}$$

create a model from data

  - abduction

$$\frac{A \to B \qquad B}{A}$$

Peirce's Logical Inferences, *adapted to ML*

- *analytic* rule, believable results
  - deduction

$$\frac{A \to B \qquad A}{B}$$

- *synthetic* rules, tentative results
  - induction

$$\frac{A \times B \;\cdots\; A \times B}{A \to B}$$

create a model from data

  - abduction

$$\frac{P \to (I \to O) \quad I \to O}{P}$$

find the best parameter of a model to explain data

“Proofs as Programs”

- *analytic* rule, believable results
  - deduction

$$\frac{t\colon A \to B \qquad u\colon A}{t\,u\colon B}$$

- *synthetic* rules, tentative results
  - induction

$$\frac{A \times B \;\cdots\; A \times B}{A \to B}$$

create a model from data

  - abduction

$$\frac{P \to (I \to O) \qquad I \to O}{P}$$

find the best parameter of a model to explain data

“Proofs as Programs”

- *analytic* rule, believable results
  - deduction

$$\frac{t\colon A \to B \qquad u\colon A}{t\,u\colon B}$$

- *synthetic* rules, tentative results
  - induction

$$\frac{\hat{x}\colon \mathsf{list}\,(A \times B)}{\mathsf{ind}(\hat{x})\colon A \to B}$$

create a model from data

  - abduction

$$\frac{P \to (I \to O) \qquad I \to O}{P}$$

find the best parameter of a model to explain data

“Proofs as Programs”

- *analytic* rule, believable results
  - deduction

$$\frac{t\colon A \to B \qquad u\colon A}{t\,u\colon B}$$

- *synthetic* rules, tentative results
  - induction

$$\frac{\hat{x}\colon \mathsf{list}\ (A \times B)}{\mathsf{ind}(\hat{x})\colon A \to B}$$

create a model from data

  - abduction

$$\frac{m\colon P \to (I \to O) \qquad \Omega\colon I \to O}{\mathsf{abd}(m, \Omega)\colon P}$$

find the best parameter of a model to explain data

Towards Programmable Induction & Abduction

$$\frac{\hat{x}\colon \mathsf{list}\ (A \times B)}{\mathsf{ind}(\hat{x})\colon A \to B}$$

interpolation
exterpolation

$$\frac{m\colon P \to (I \to O) \quad \Omega\colon I \to O}{\mathsf{abd}(m, \Omega)\colon P}$$

generic optimisation
(e.g. gradient descent)

Towards Programmable Induction & Abduction

$$\frac{\hat{x}\colon \mathsf{list}\ (A \times B)}{\mathsf{ind}(\hat{x})\colon A \to B}$$

metric space

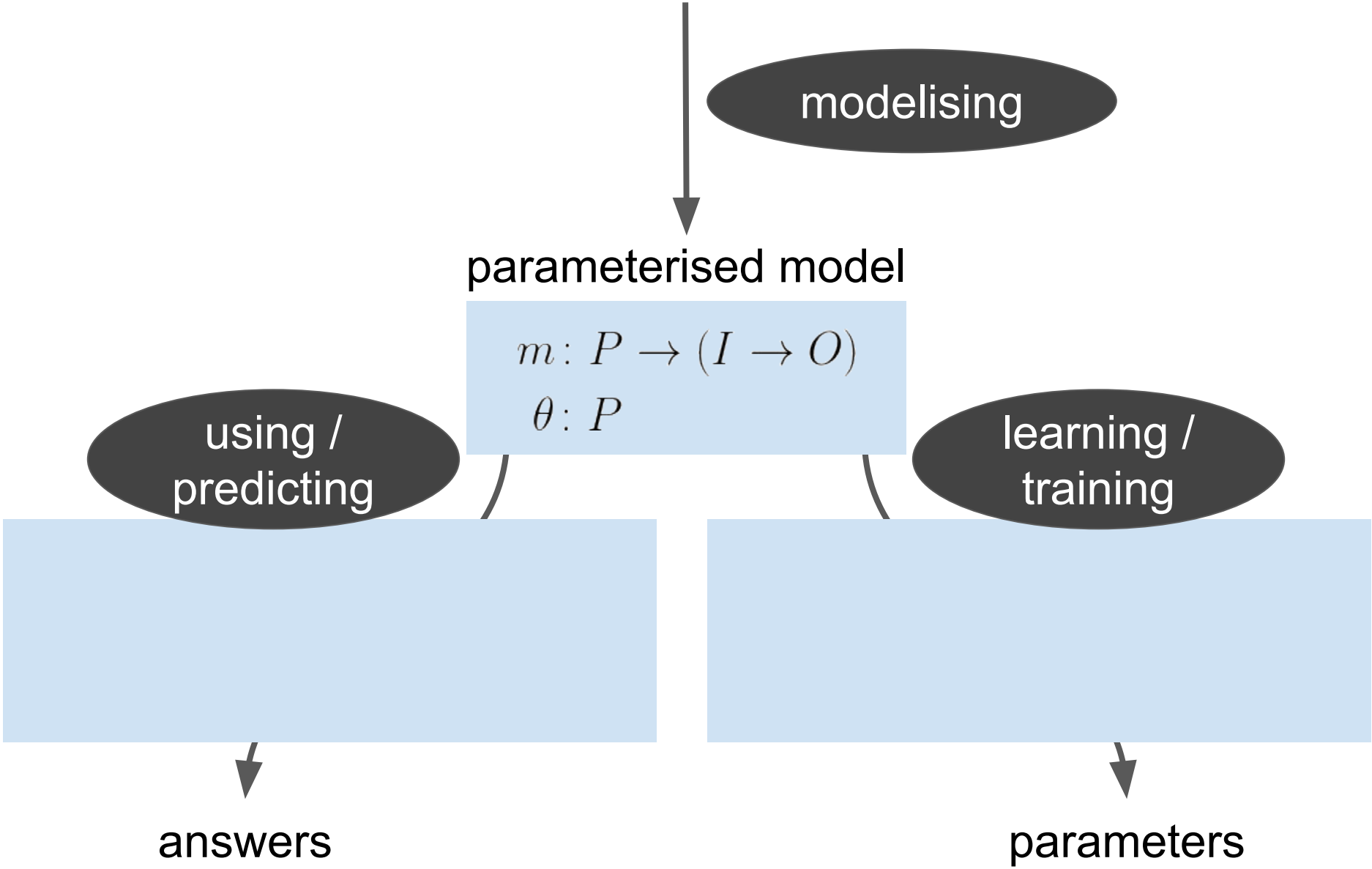
interpolation
exterpolation

$$\frac{m\colon P \to (I \to O) \quad \Omega\colon I \to O}{\mathsf{abd}(m, \Omega)\colon P}$$

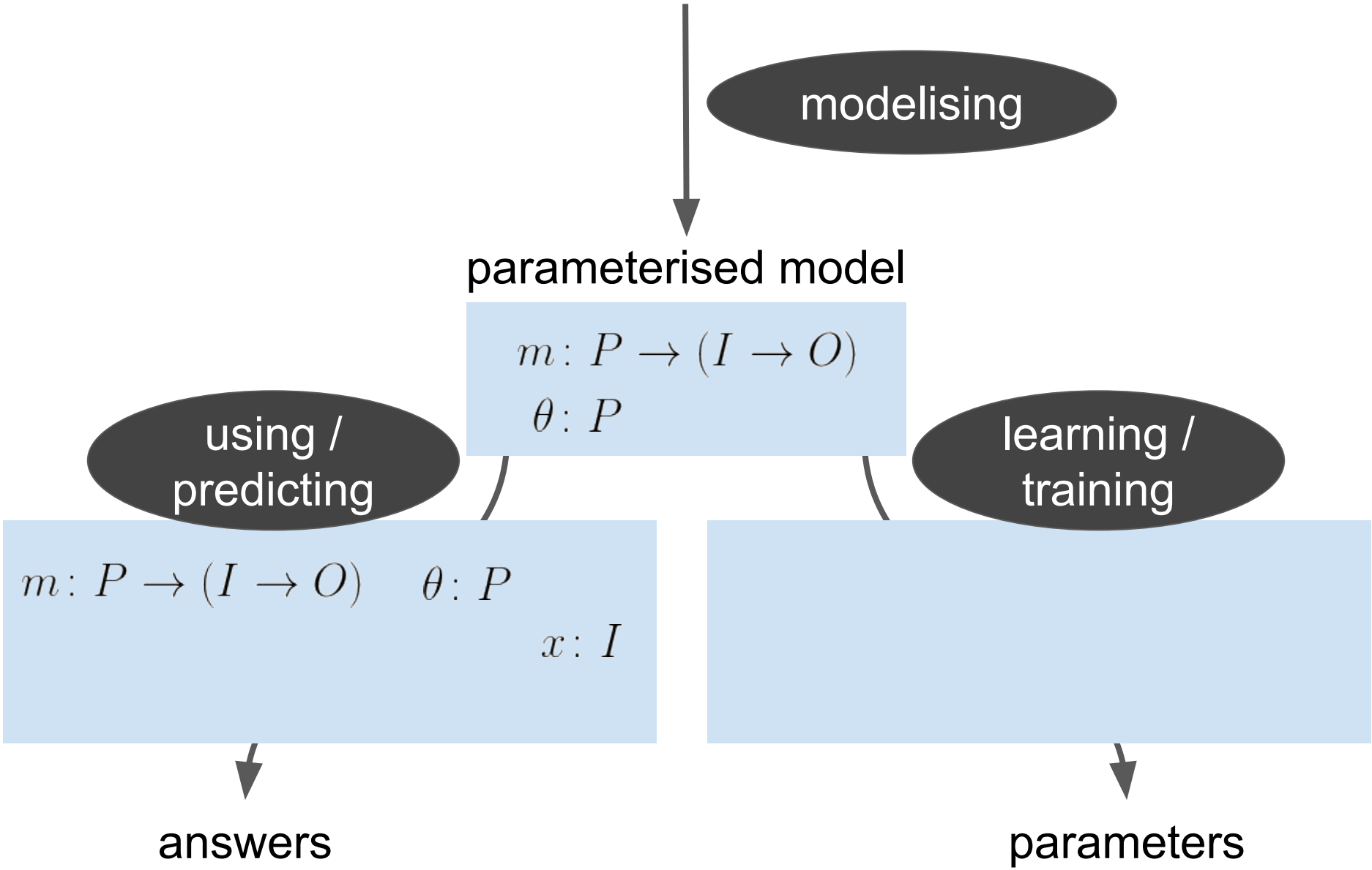
vector space

generic optimisation
(e.g. gradient descent)

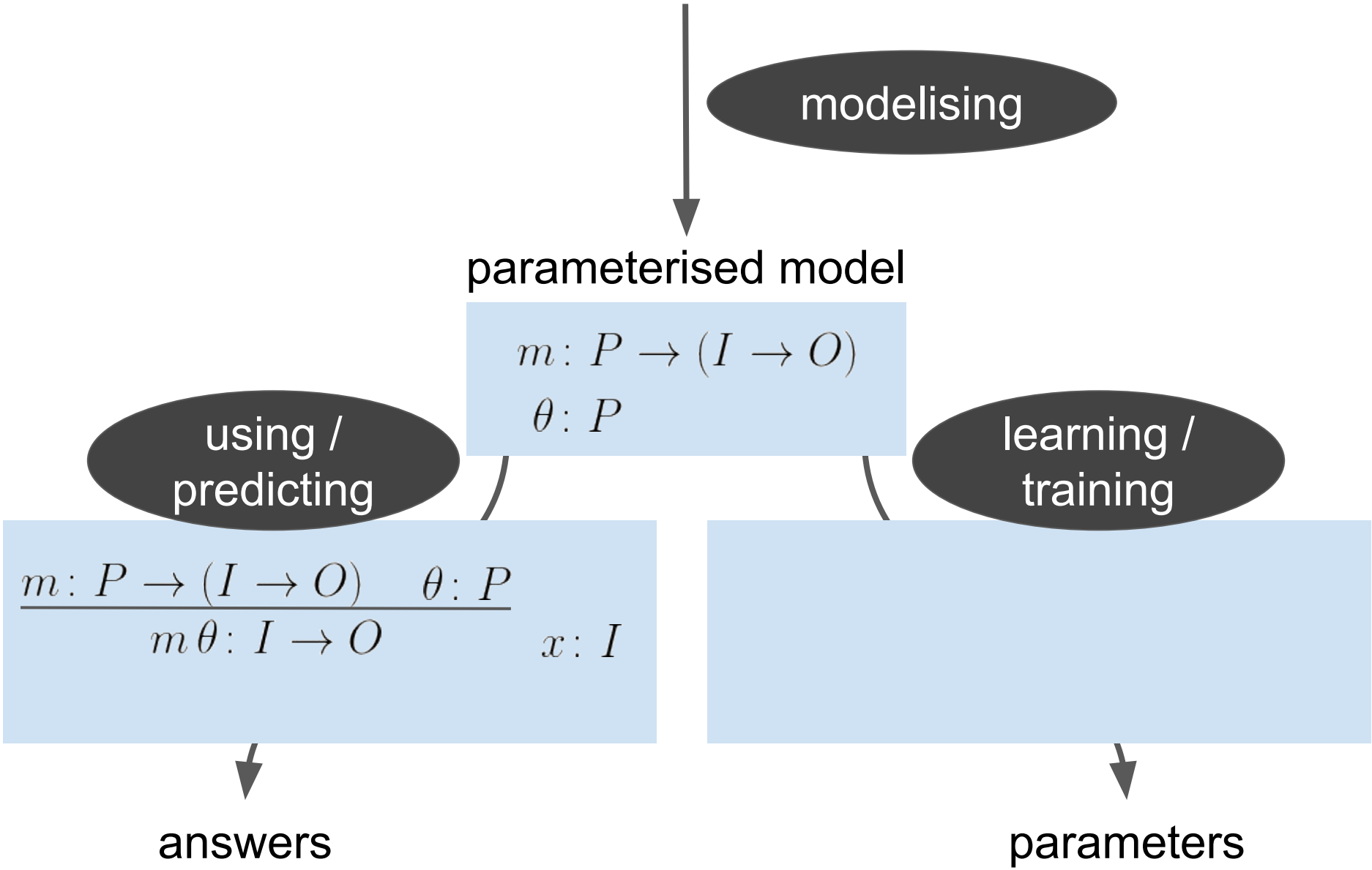
Machine Learning with Induction & Abduction



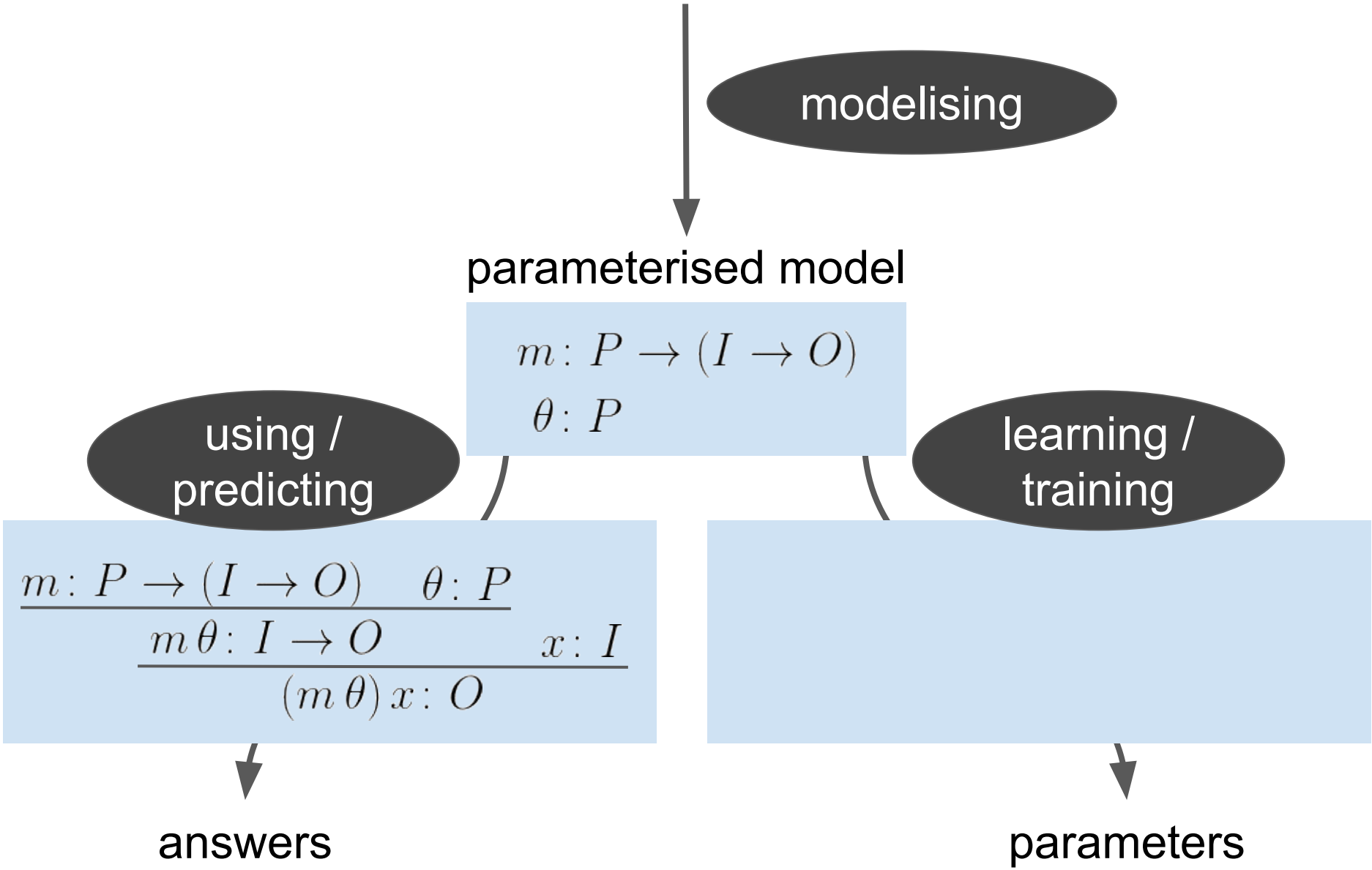
Machine Learning with Induction & Abduction



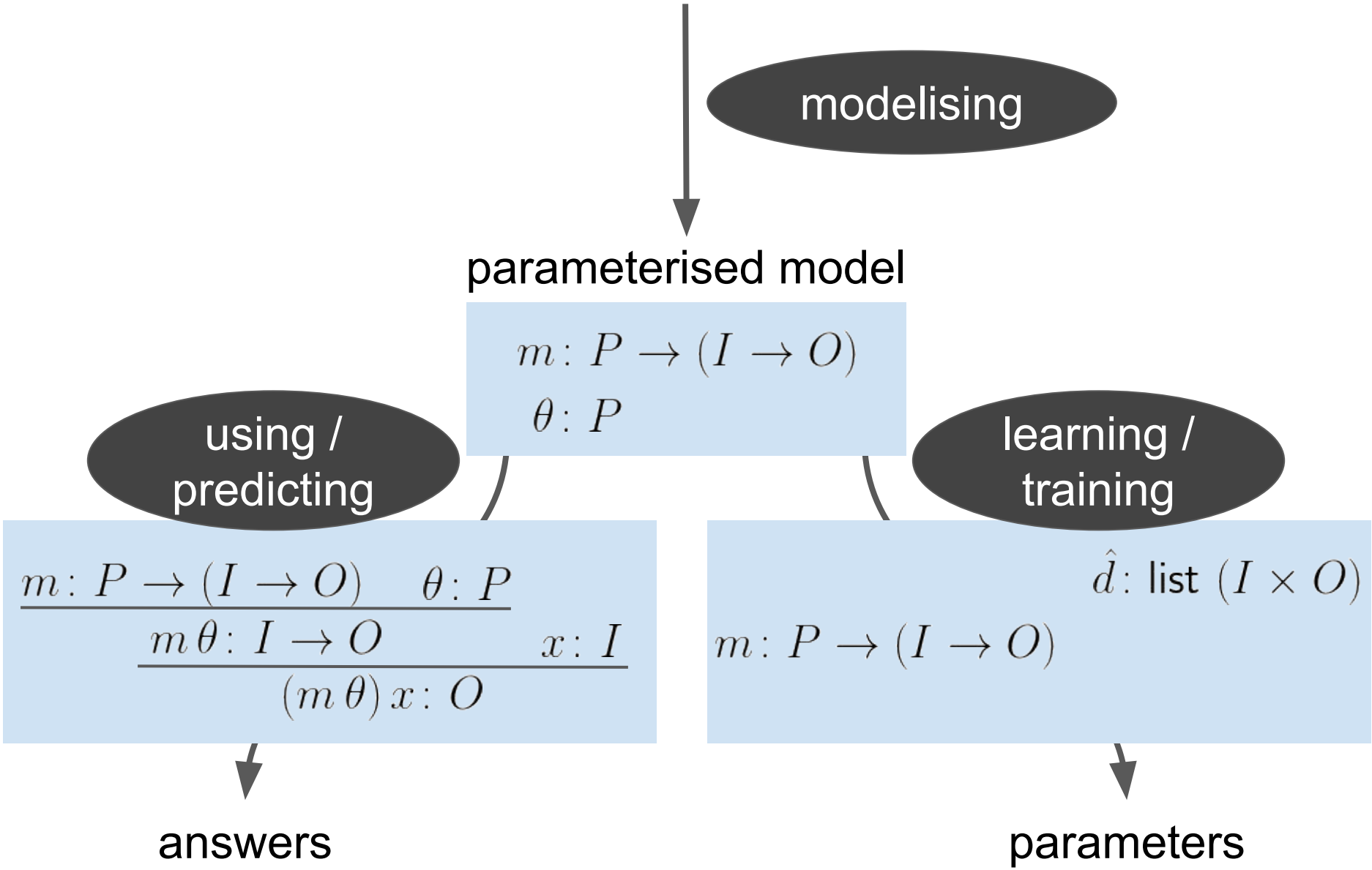
Machine Learning with Induction & Abduction



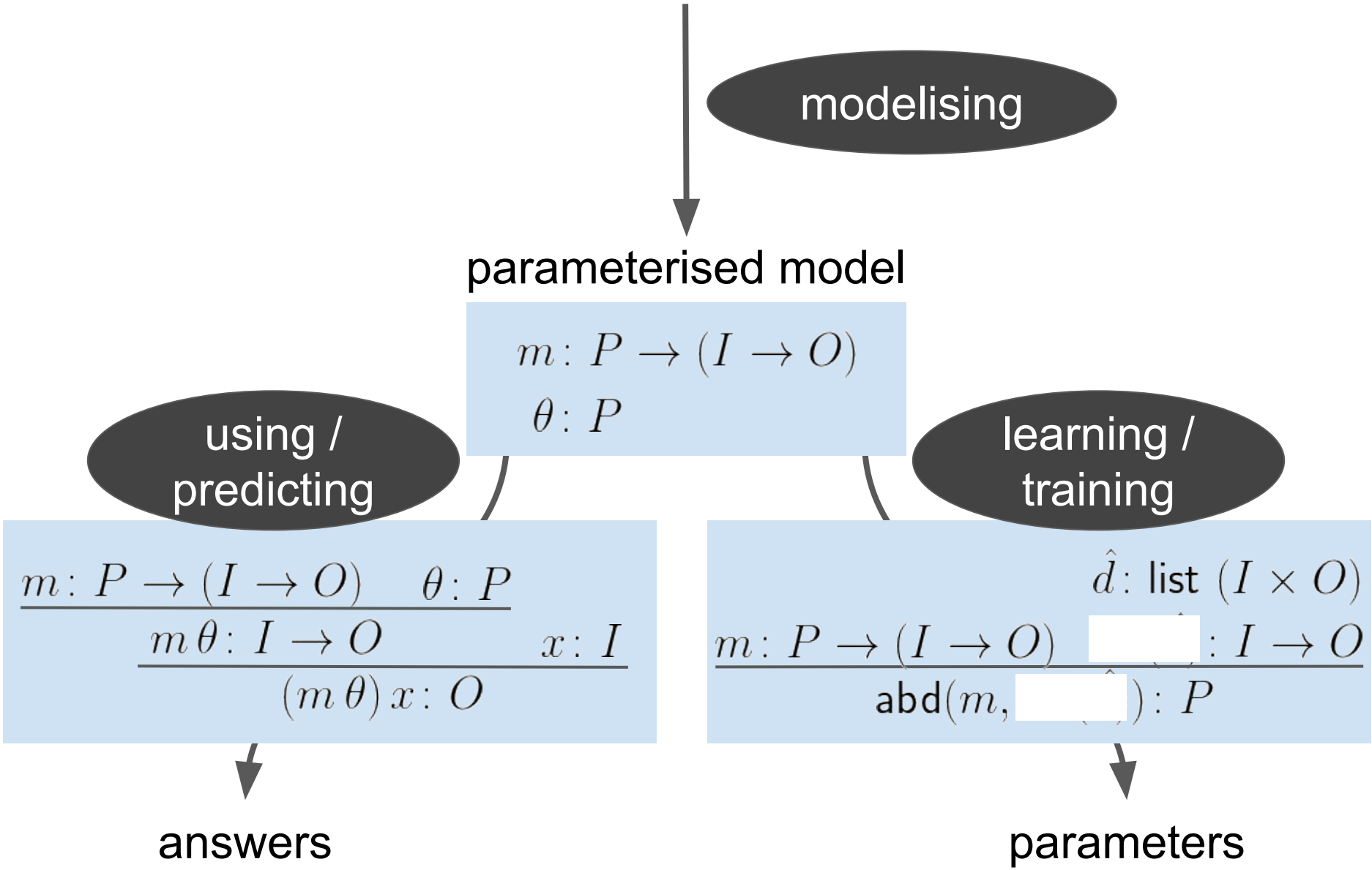
Machine Learning with Induction & Abduction



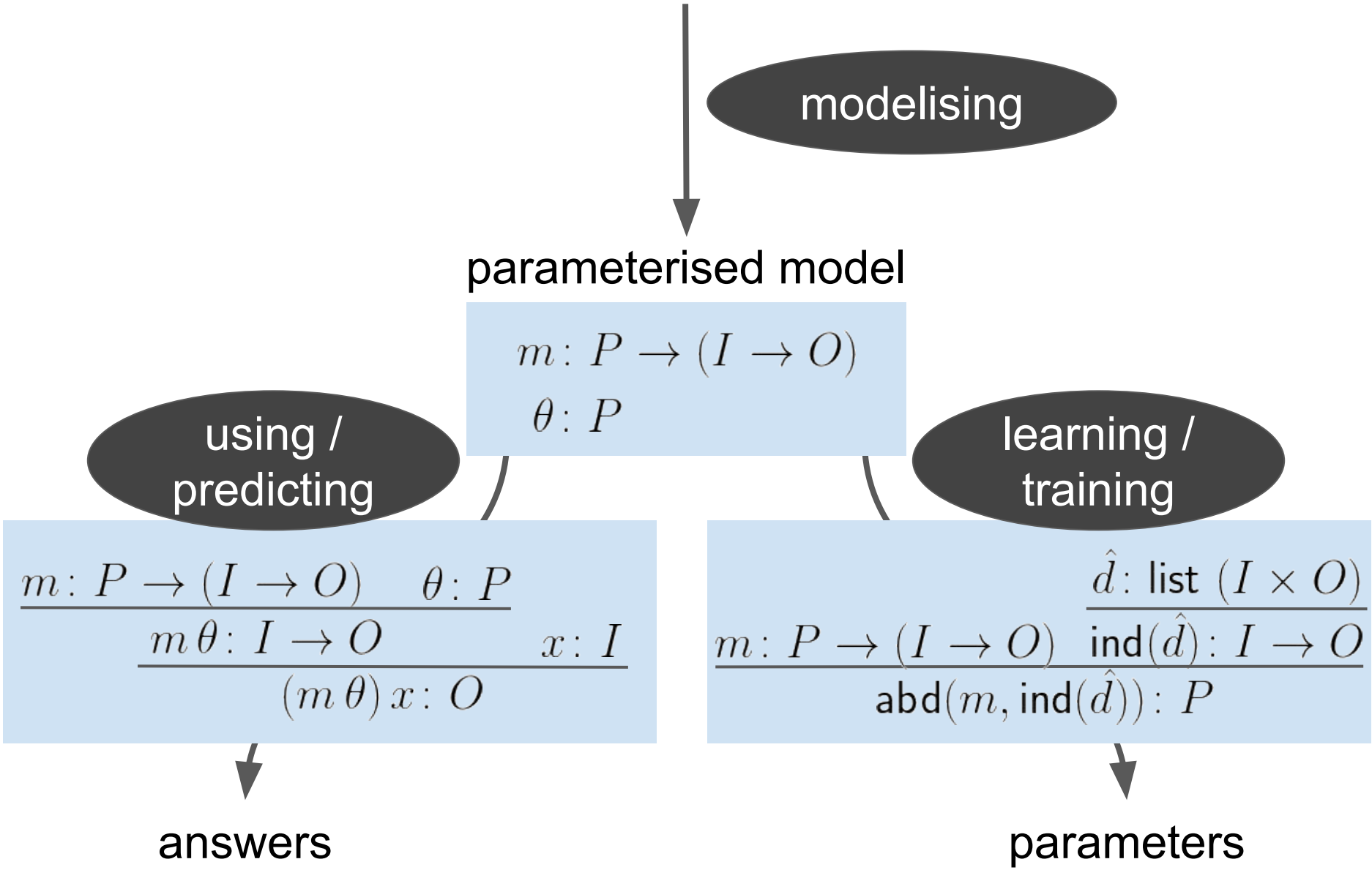
Machine Learning with Induction & Abduction



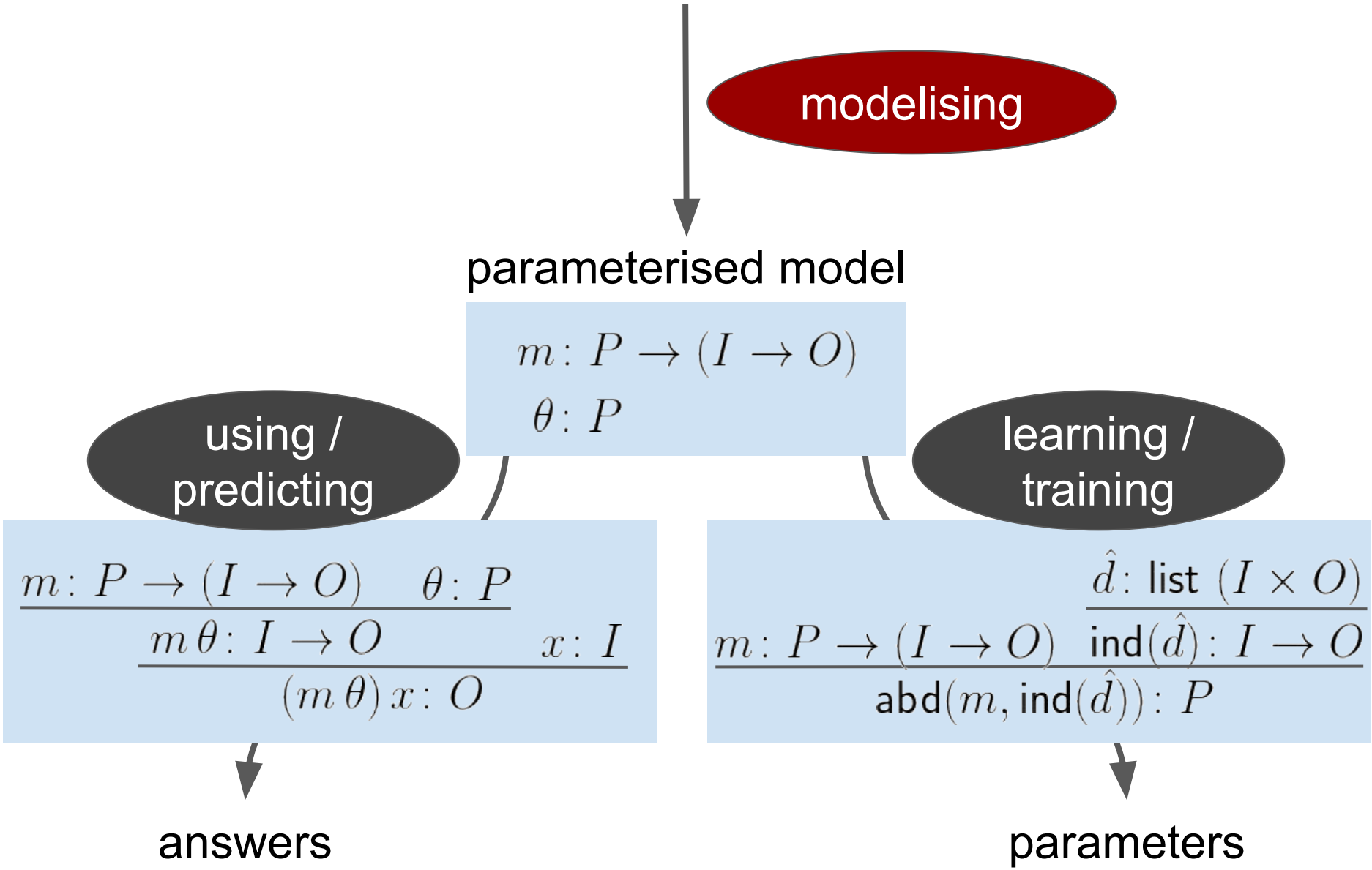
Machine Learning with Induction & Abduction



Machine Learning with Induction & Abduction



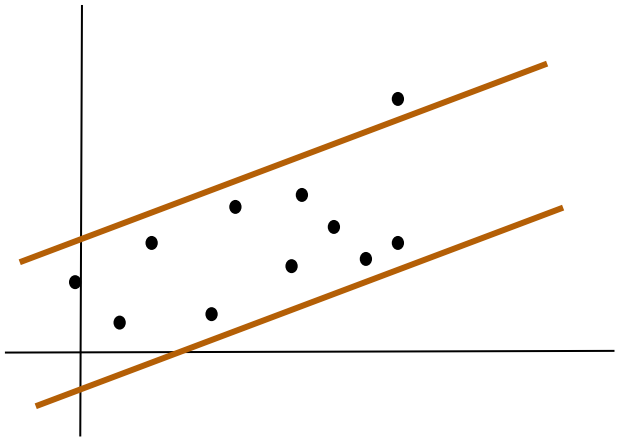
Machine Learning with Induction & Abduction



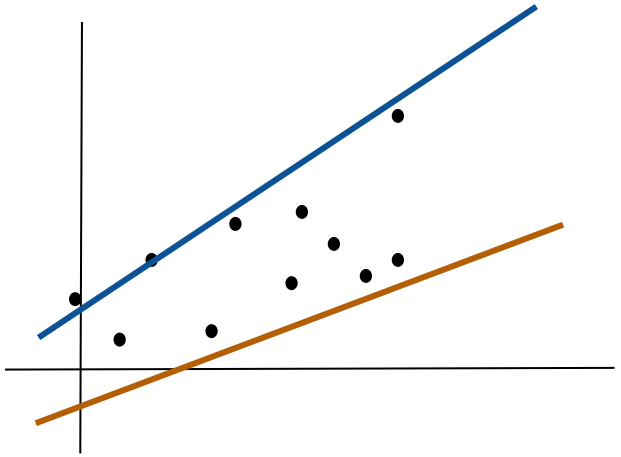
Parameter Management in Modelising

```
let pl v x = v[0] * x + v[1]


let pci v = (pl v[0,1], pl v[0,2])
let pwr v = (pl v[0,1], pl v[2,3])
```

linear regression with confidence interval



weighted regression

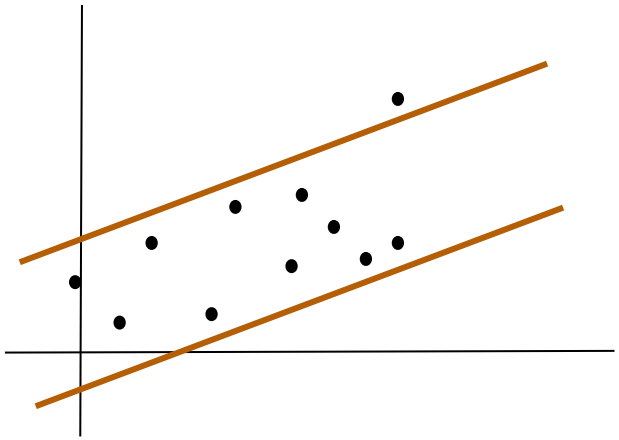


Parameter Management in Modelising

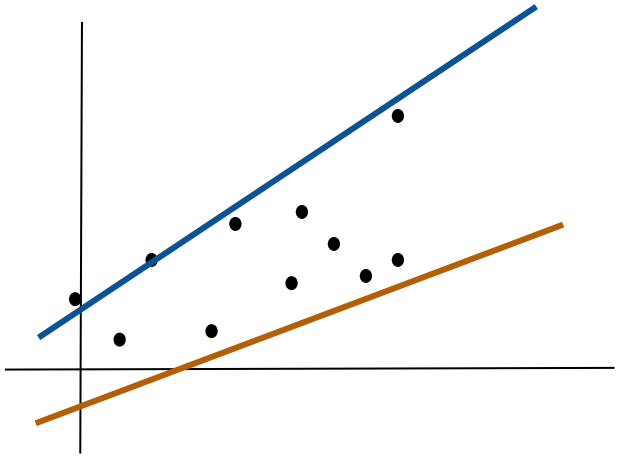
```
let pl v x = v[0] * x + v[1]


let pci v = (pl v[0,1], pl v[0,2])
let pwr v = (pl v[0,1], pl v[2,3])
```

error-prone

linear regression with confidence interval



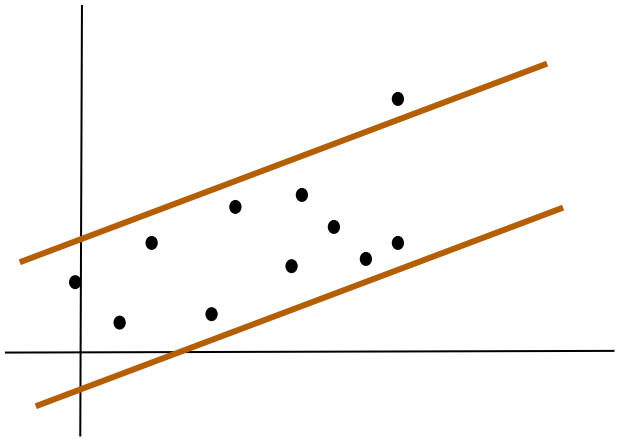
weighted regression



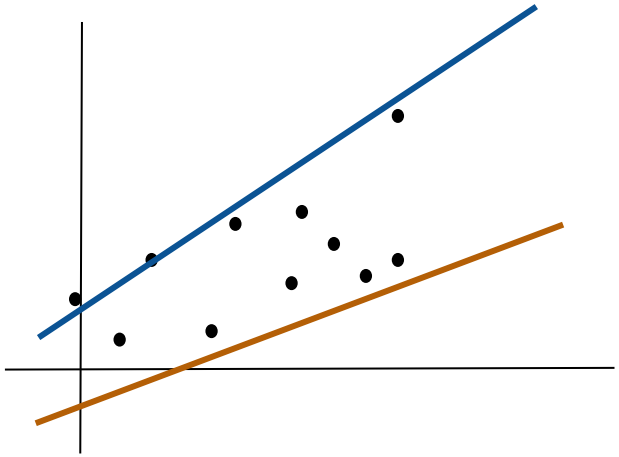
Modelising with *Embedded* Parameters

```
let pl' a b x = a * x + b

let ci = let a = {0} in
         (pl' a {1}, pl' a {2})
let wr = (pl' {0} {1}, pl' {2} {3})
```

linear regression with confidence interval



weighted regression



Modelising with *Embedded* Parameters

```
let pl' a b x = a * x + b

let ci = let a = {0} in
         (pl' a {1}, pl' a {2})
let wr = (pl' {0} {1}, pl' {2} {3})

;; pl'    : F -> F -> F -> F
;; ci,wr : (F -> F) * (F -> F)
```

$$\frac{}{\{k\} : \mathbb{F}}$$

```
let pl v x = v[0] * x + v[1]

let pci v = (pl v[0,1], pl v[0,2])
let pwr v = (pl v[0,1], pl v[2,3])

;; pl       : P -> F -> F
;; pci,pwr : P -> ((F -> F) * (F -> F))
```

Modelising with *Embedded* Parameters

```
let pl' a b x = a * x + b

let ci = let a = {0} in
         (pl' a {1}, pl' a {2})
let wr = (pl' {0} {1}, pl' {2} {3})

;; pl'    : F -> F -> F -> F
;; ci,wr : (F -> F) * (F -> F)
```

$$\frac{}{\{k\} : \mathbb{F}}$$

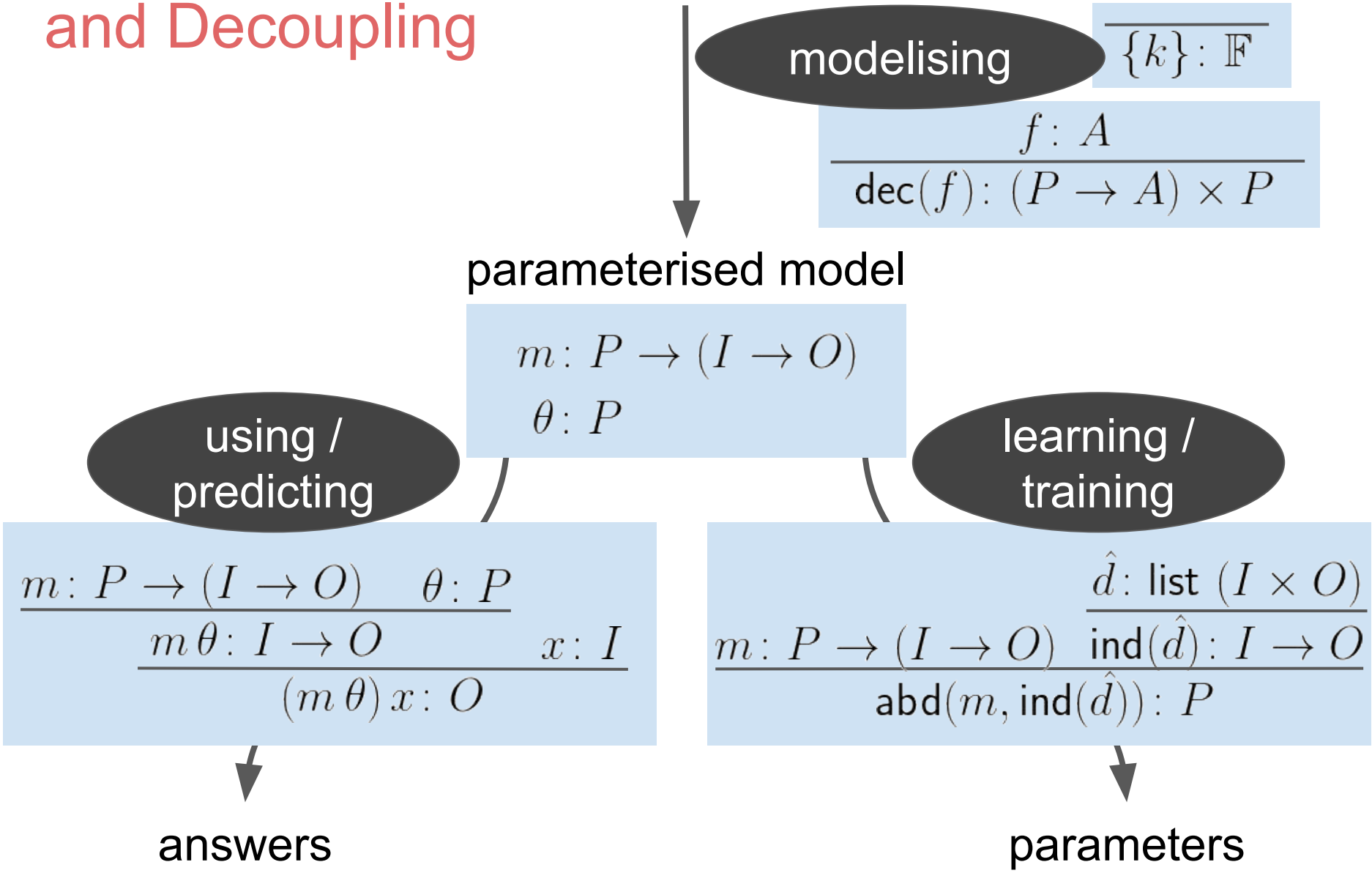
...and *Decoupling*

$$\frac{f : A}{\mathsf{dec}(f) : (P \to A) \times P}$$

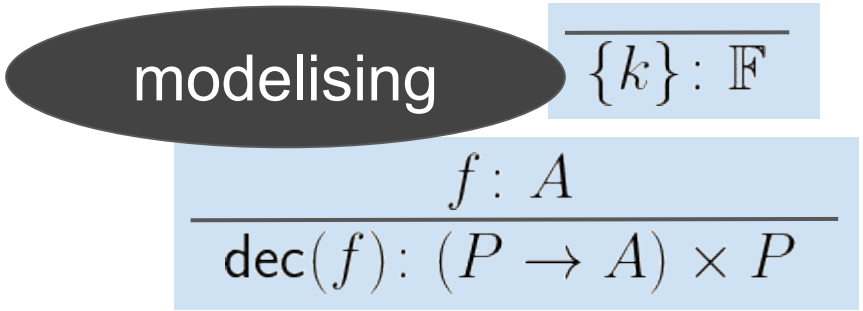
```
let pl v x = v[0] * x + v[1]

let pci v = (pl v[0,1], pl v[0,2])
let pwr v = (pl v[0,1], pl v[2,3])

;; pl       : P -> F -> F
;; pci,pwr : P -> ((F -> F) * (F -> F))
```

```
dec(ci) ~ (pci, [0;1;2])
dec(wr) ~ (pwr, [0;1;2;3])
```

Machine Learning with Induction, Abduction and Decoupling



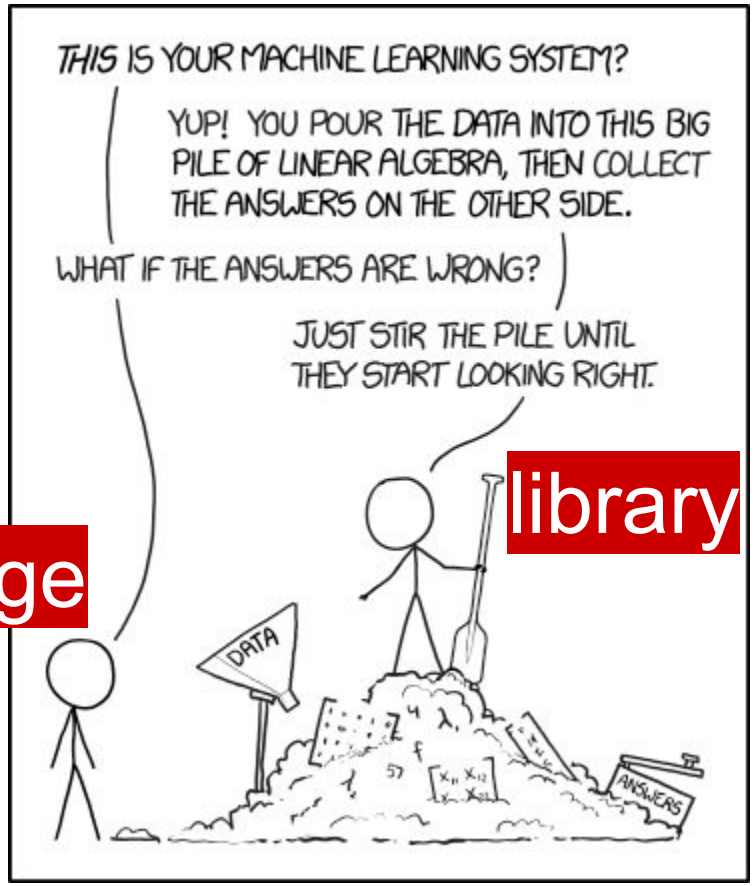
DecML, or Idealised TensorFlow



- as an extended simply-typed lambda-calculus
  - type soundness & beta-law, using graph-rewriting operational semantics [-, Cheung & Ghica, LICS '18 to appear]
  - graph-rewriting visualiser https://cwtsteven.github.io/GoI-TF-Visualiser/CBV-with-CBN-embedding/index.html
- as a PPX extension of OCaml
  - https://github.com/DecML/decml-ppx
    https://github.com/reubenrowe/ocaml-decml (to be merged)
  - (non-trivial) translation to OCaml in pre-processing

Machine Learning with Induction, Abduction and Decoupling

host language

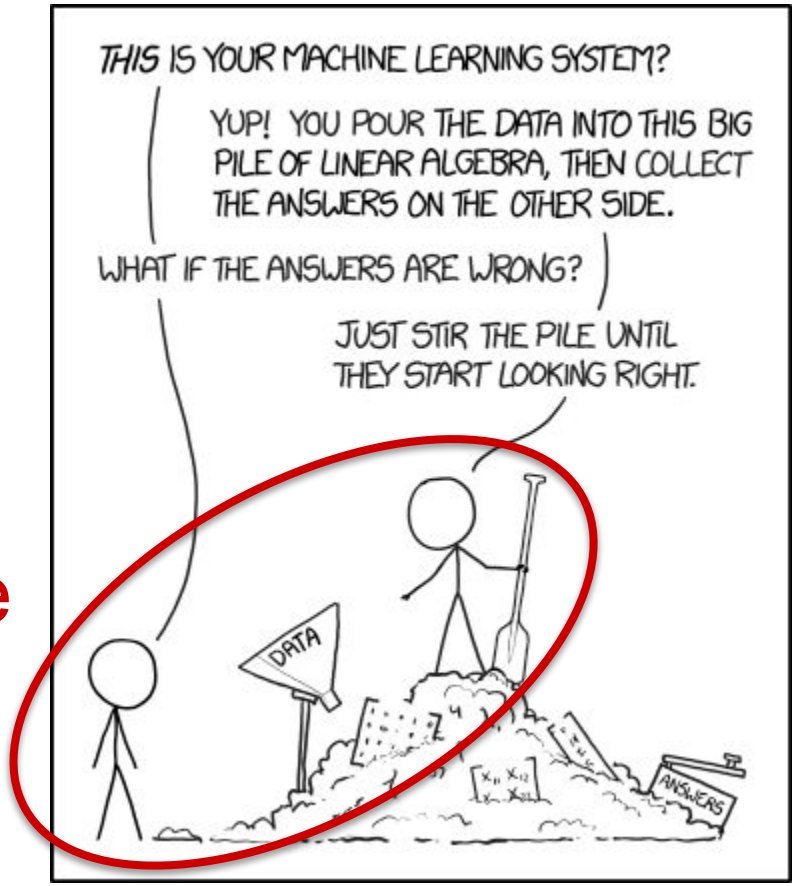


library

https://xkcd.com/1838/

Machine Learning with Induction, Abduction and Decoupling

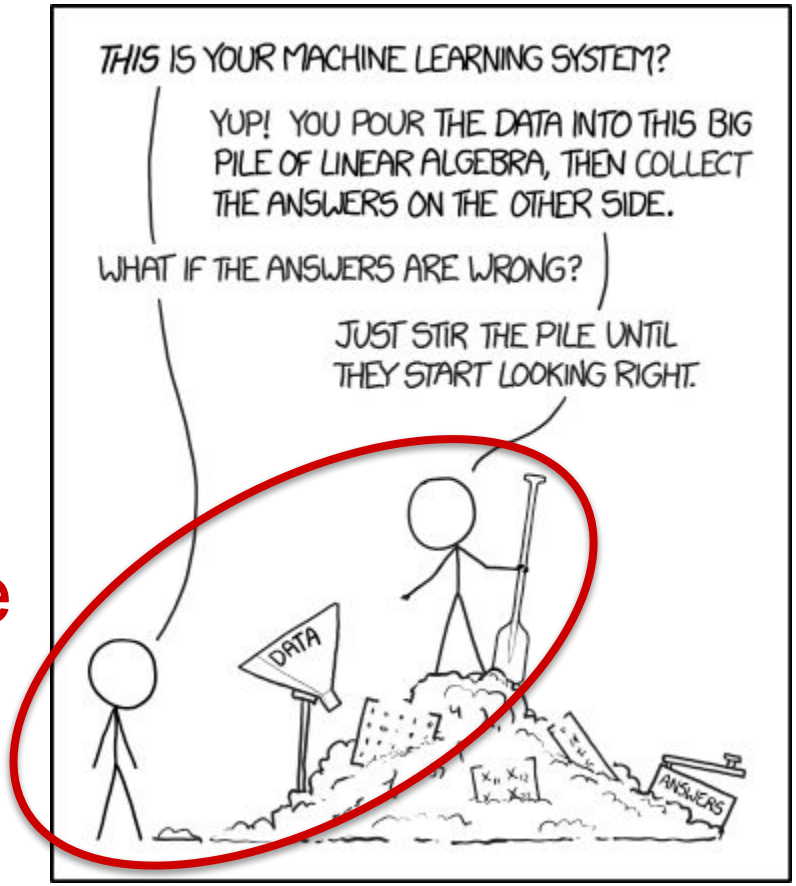
Let them be *united* :-)



https://xkcd.com/1838/

Machine Learning with Induction, Abduction and Decoupling

Let them be *united* :-)



https://xkcd.com/1838/

probability TBA