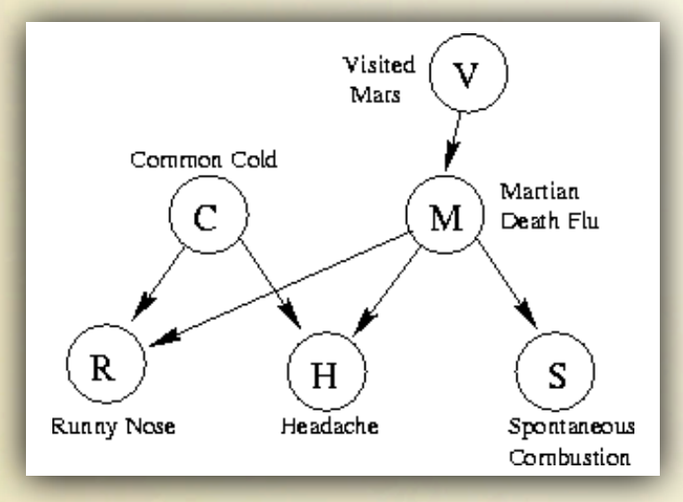
The surprising effectiveness of lenses in machine learning

Fabio Zanasi
University College London

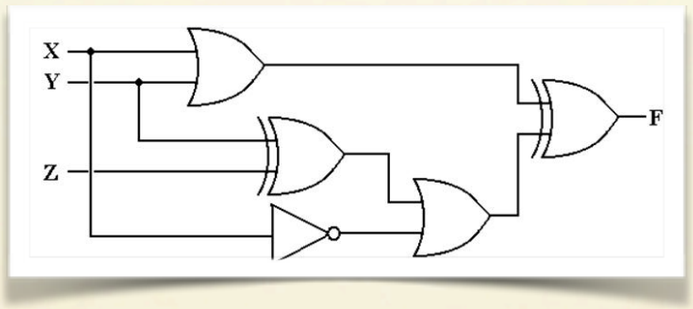
Based on joint work with
Geoff Cruttwell, Bruno Gavranovic, Neil Ghani, Paul Wilson

Background: Categorical Foundations of Component-based systems

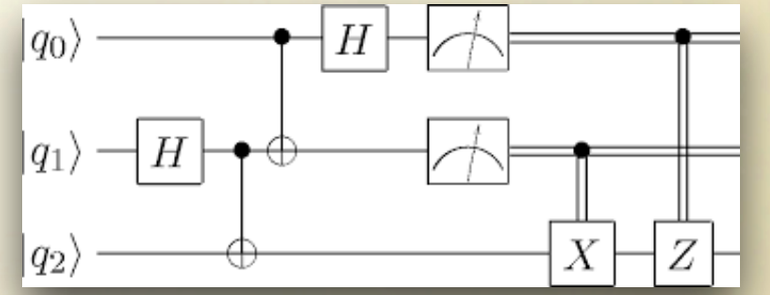
Bayesian networks



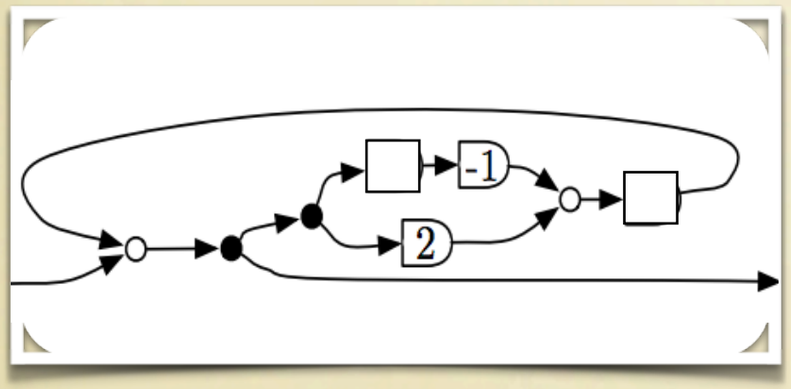
Digital Circuits



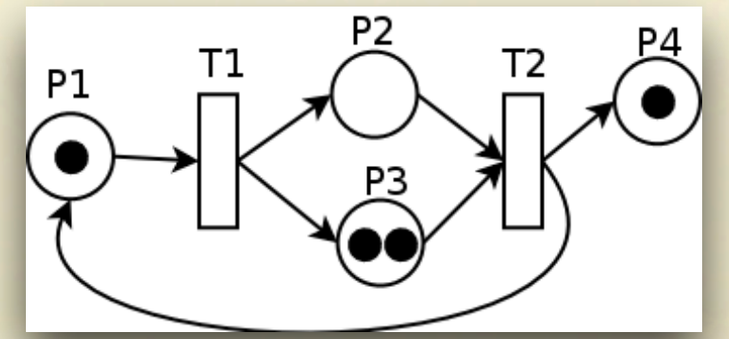
Quantum circuits



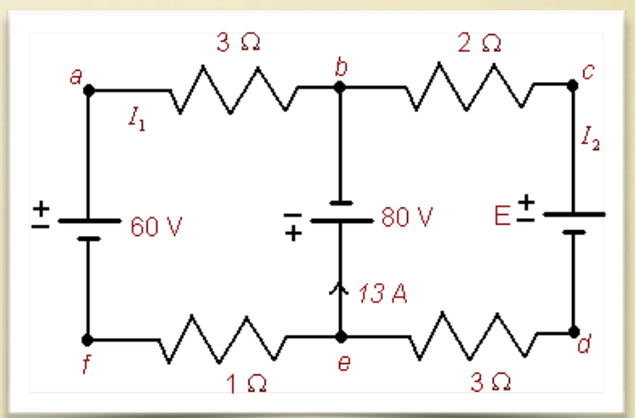
Signal flow graphs



Petri nets



Electrical Circuits

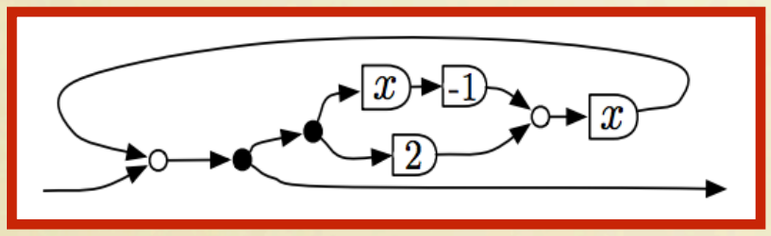


Key Ideas

Graphical notation becomes **formal syntax**

and it is given **compositional** semantics

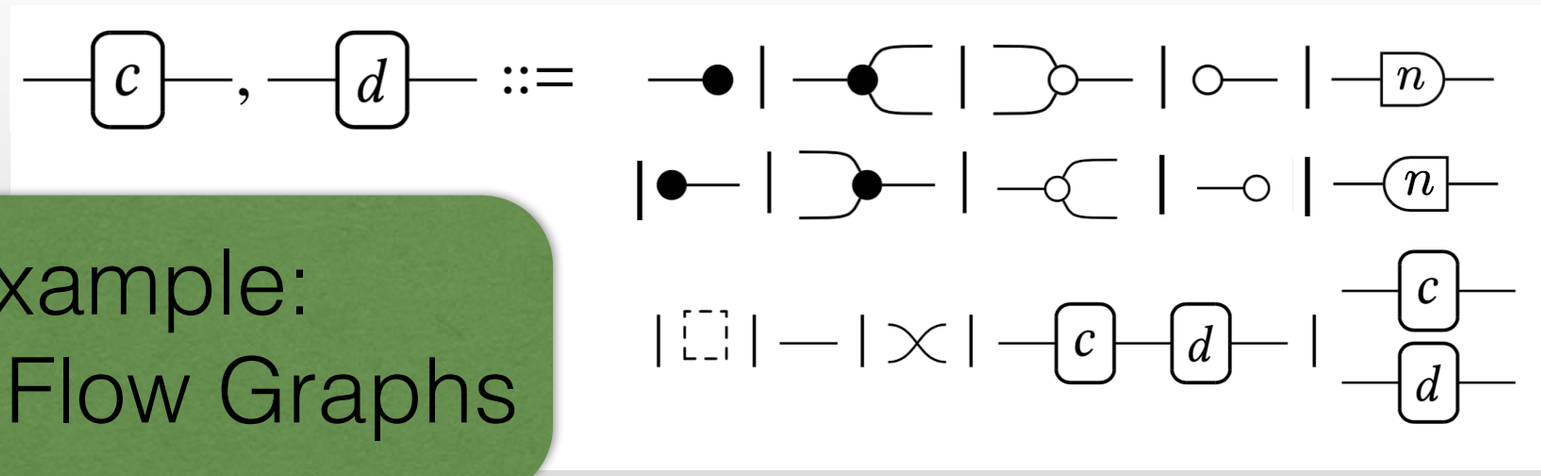
Example: Signal Flow Graphs



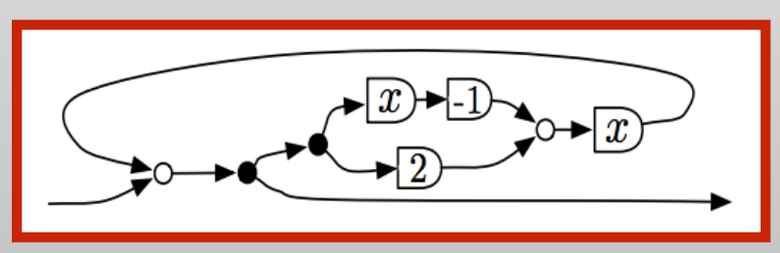
Key Ideas

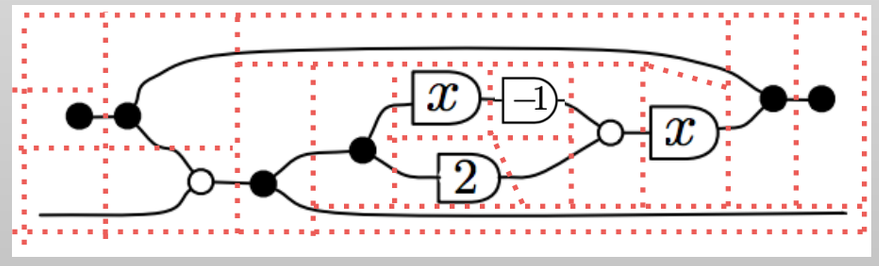
Graphical notation becomes **formal syntax**

and it is given **compositional** semantics



Example:
Signal Flow Graphs

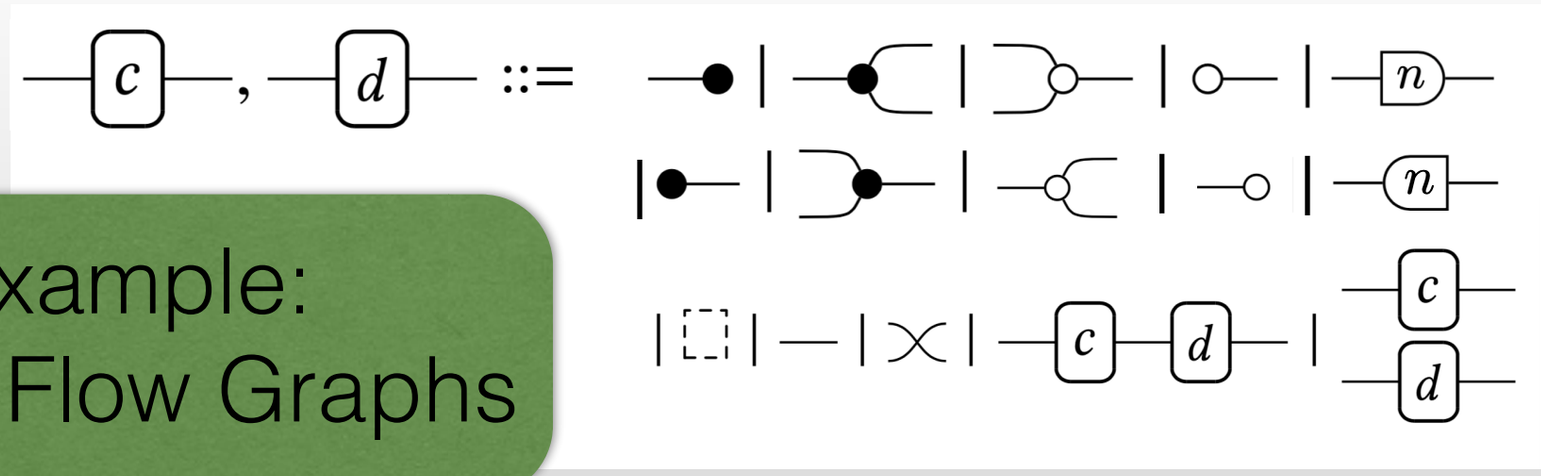




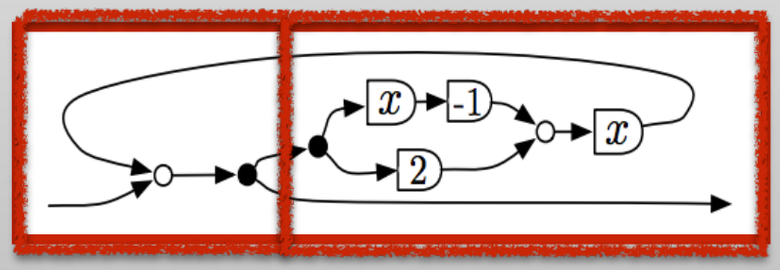
Key Ideas

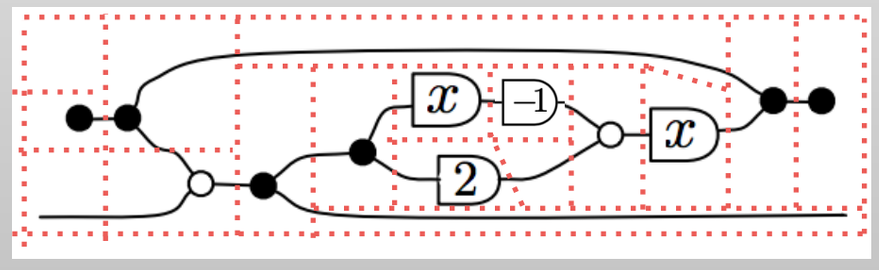
Graphical notation becomes **formal syntax**

and it is given **compositional** semantics



Example:
Signal Flow Graphs

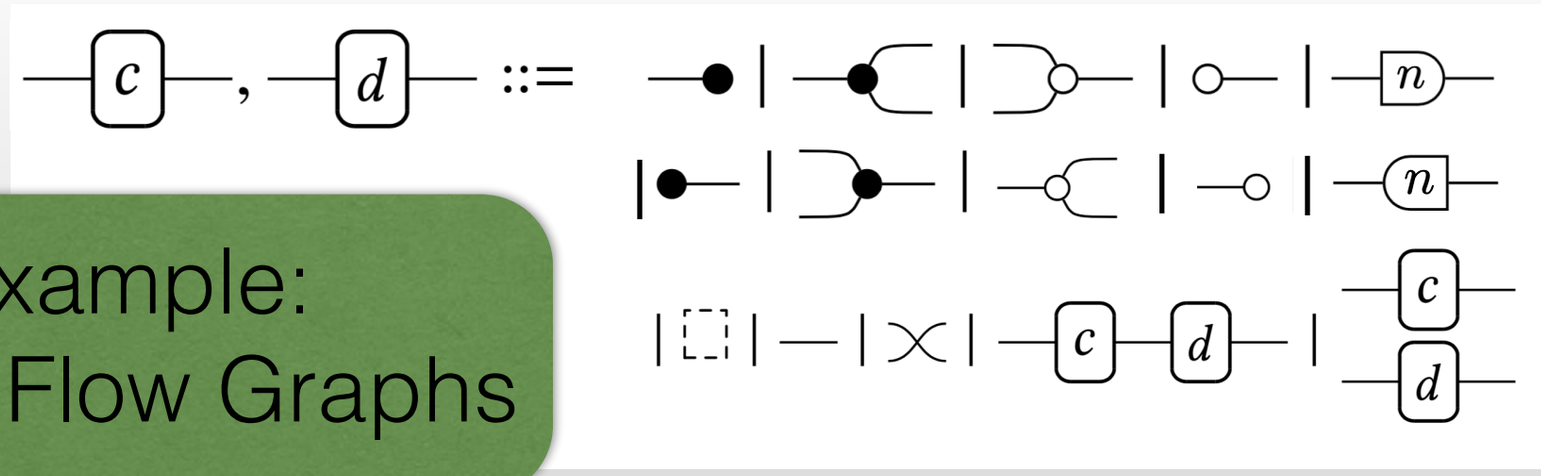




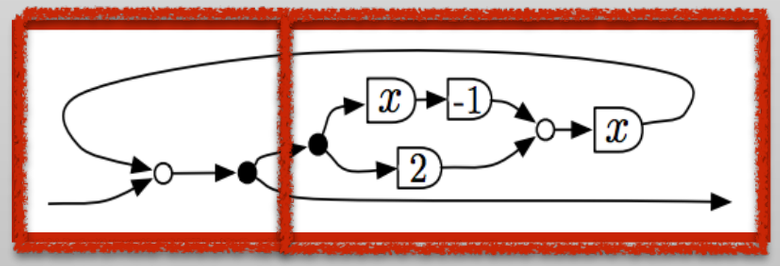
Key Ideas

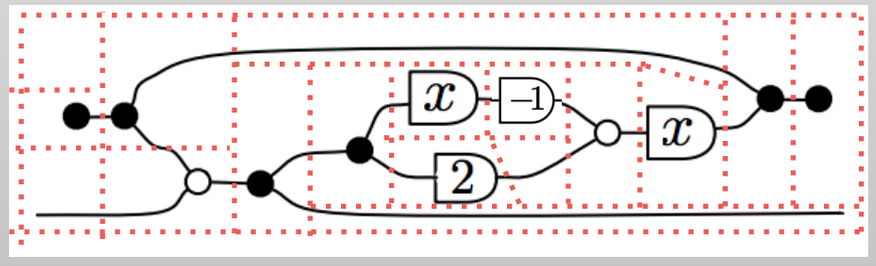
Graphical notation becomes **formal syntax**

and it is given **compositional** semantics



Example:
Signal Flow Graphs





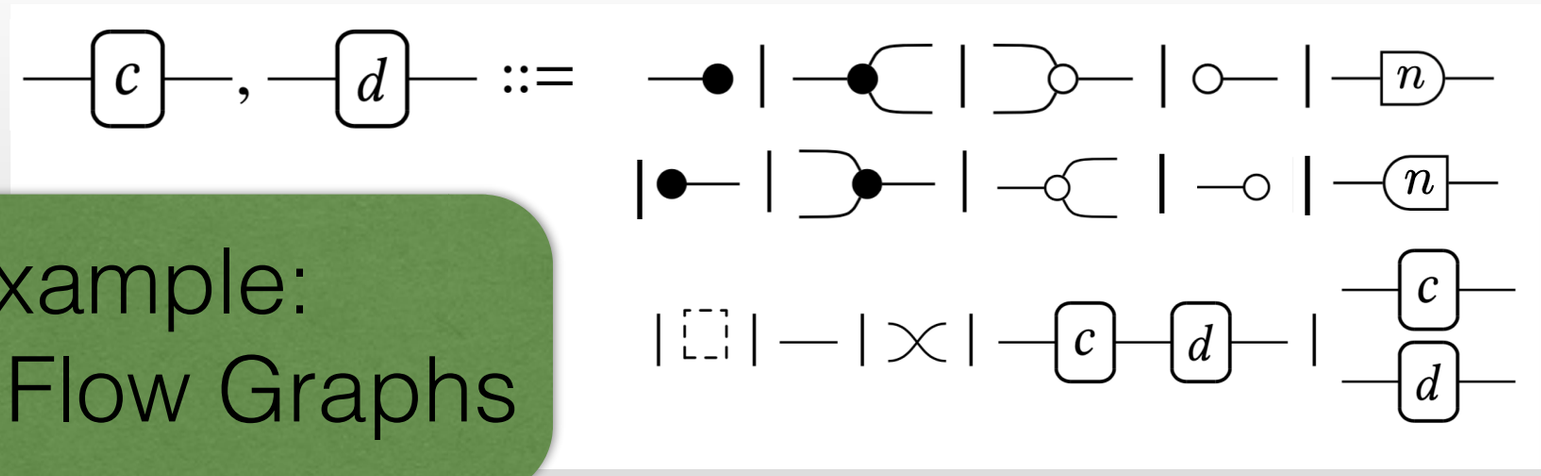
?

?

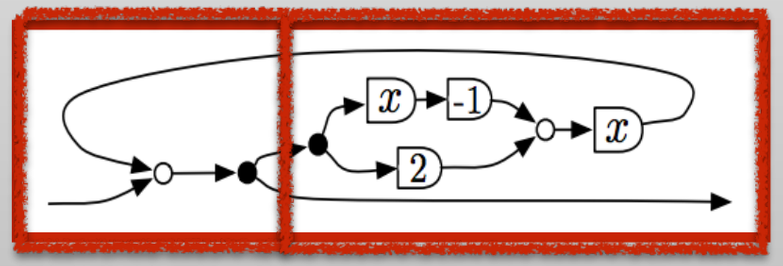
Key Ideas

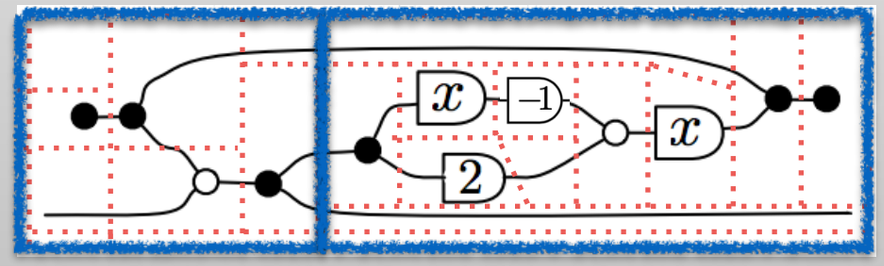
Graphical notation becomes **formal syntax**

and it is given **compositional** semantics



Example:
Signal Flow Graphs





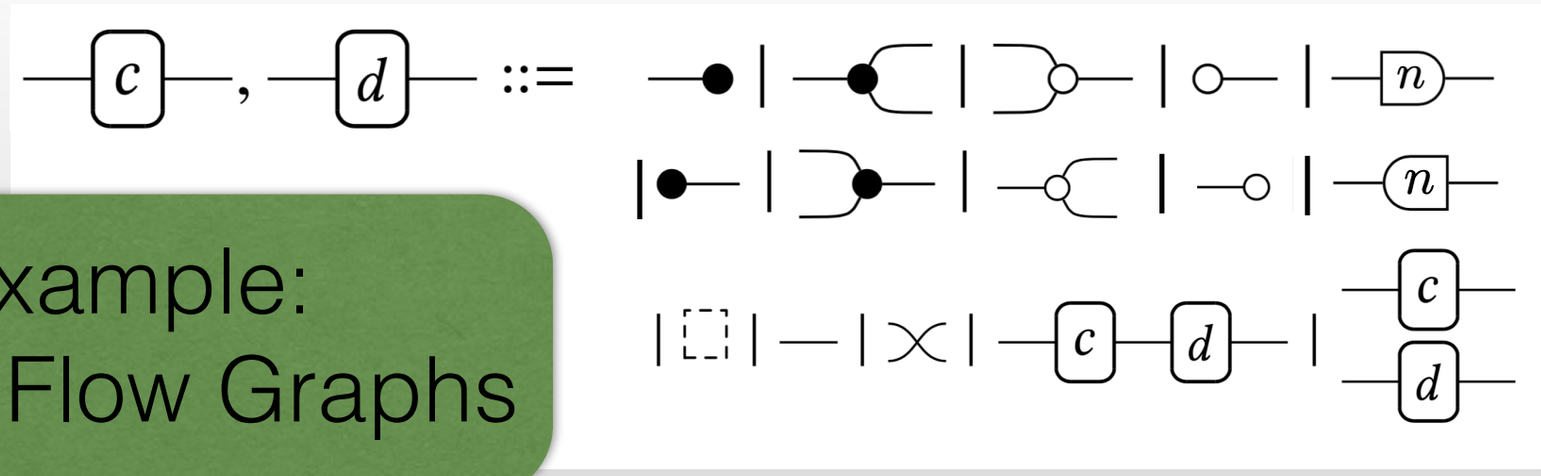
?

?

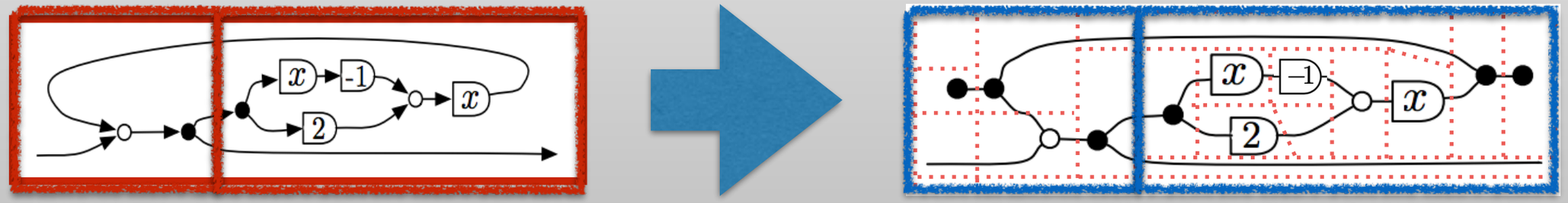
Key Ideas

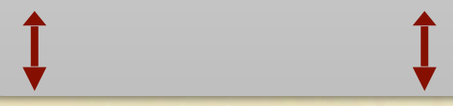
Graphical notation becomes **formal syntax**

and it is given **compositional** semantics



Example:
Signal Flow Graphs

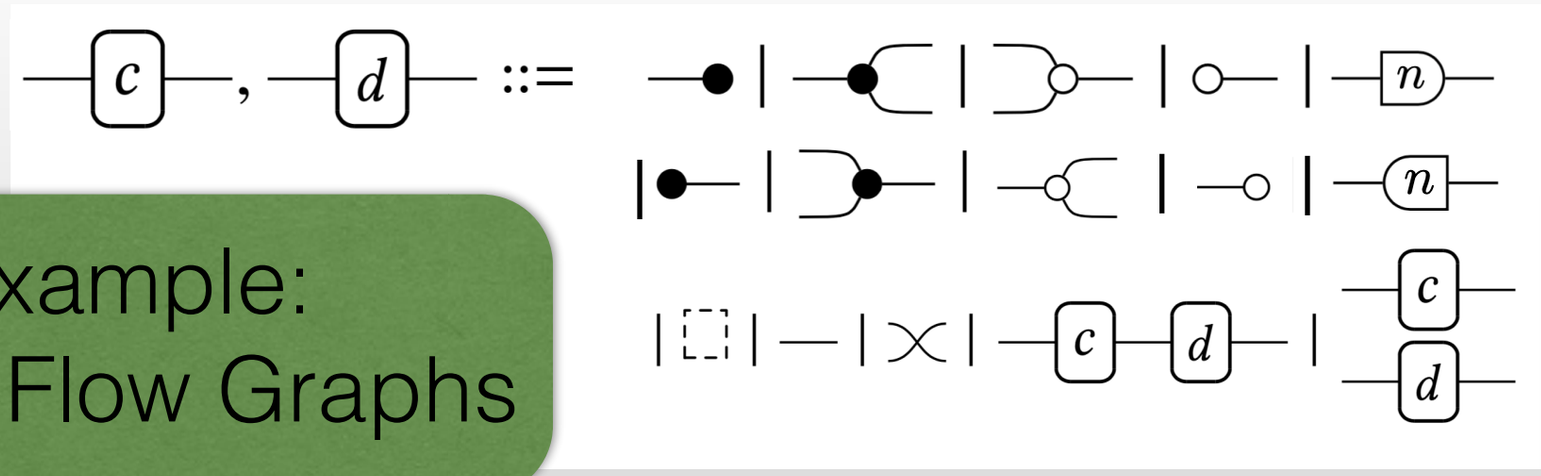




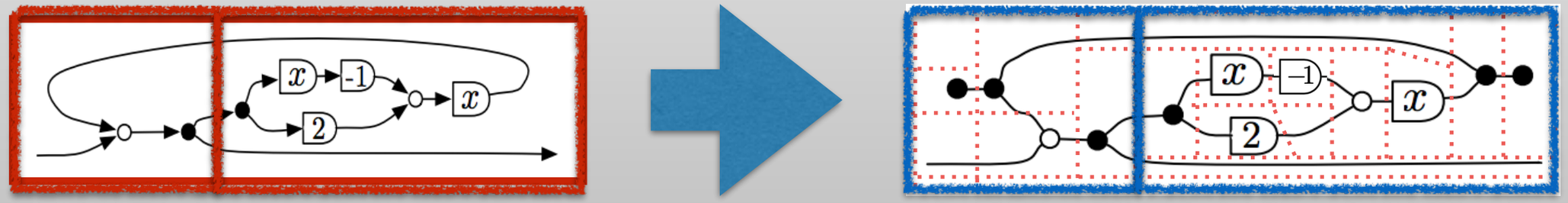
Key Ideas

Graphical notation becomes **formal syntax**

and it is given **compositional** semantics



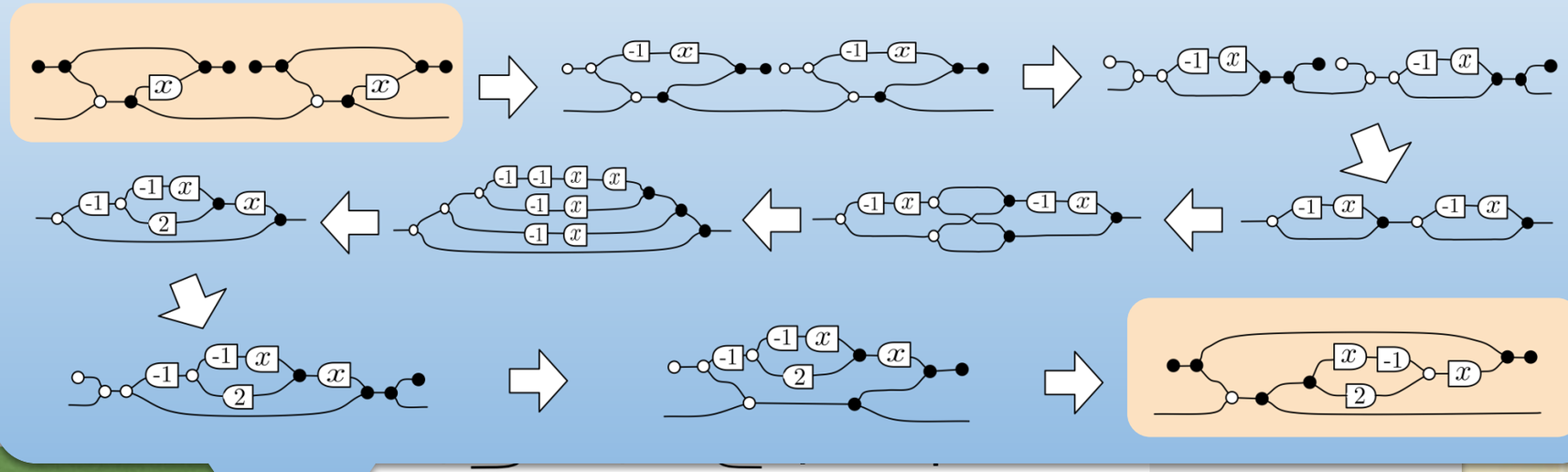
Example:
Signal Flow Graphs

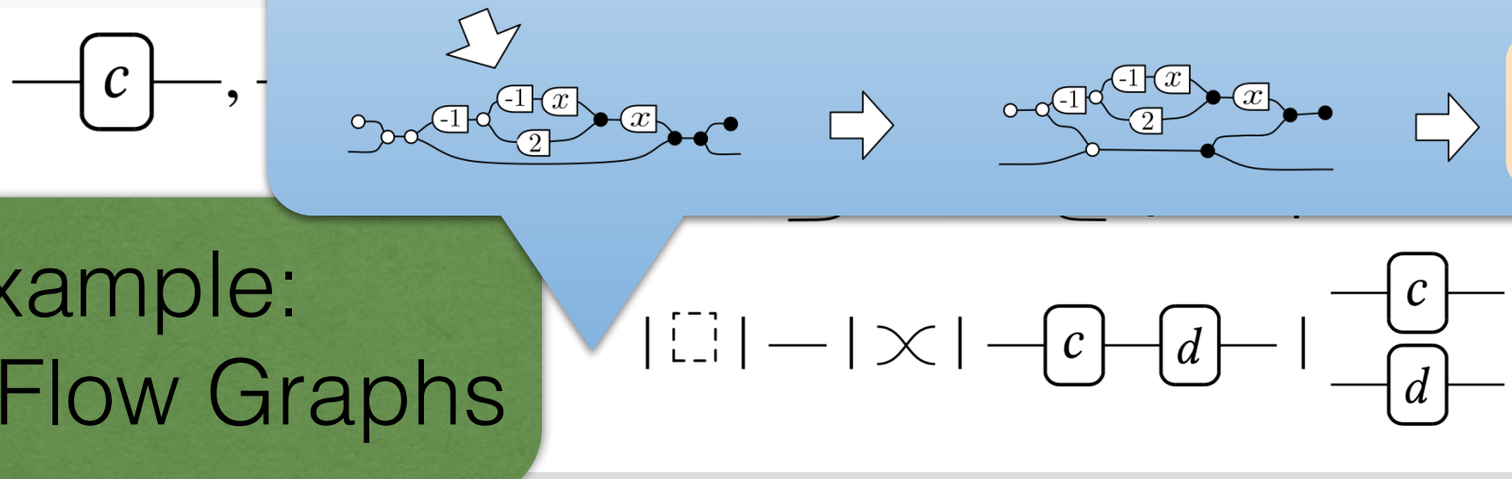


? ? R_1 ; R_2 = R

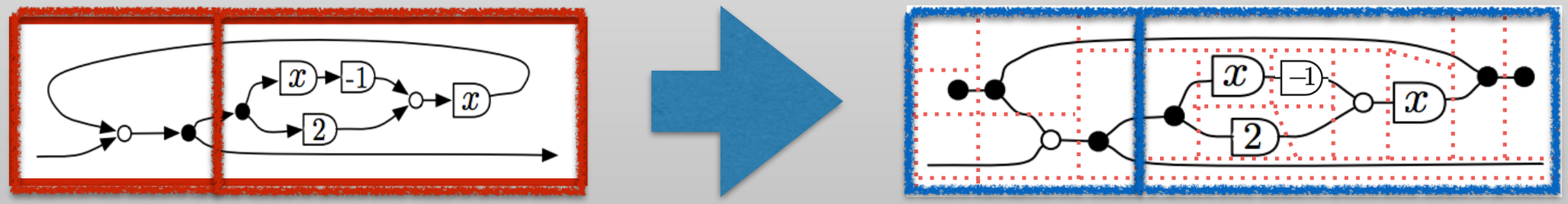
Graphica

and it is





Example:
Signal Flow Graphs

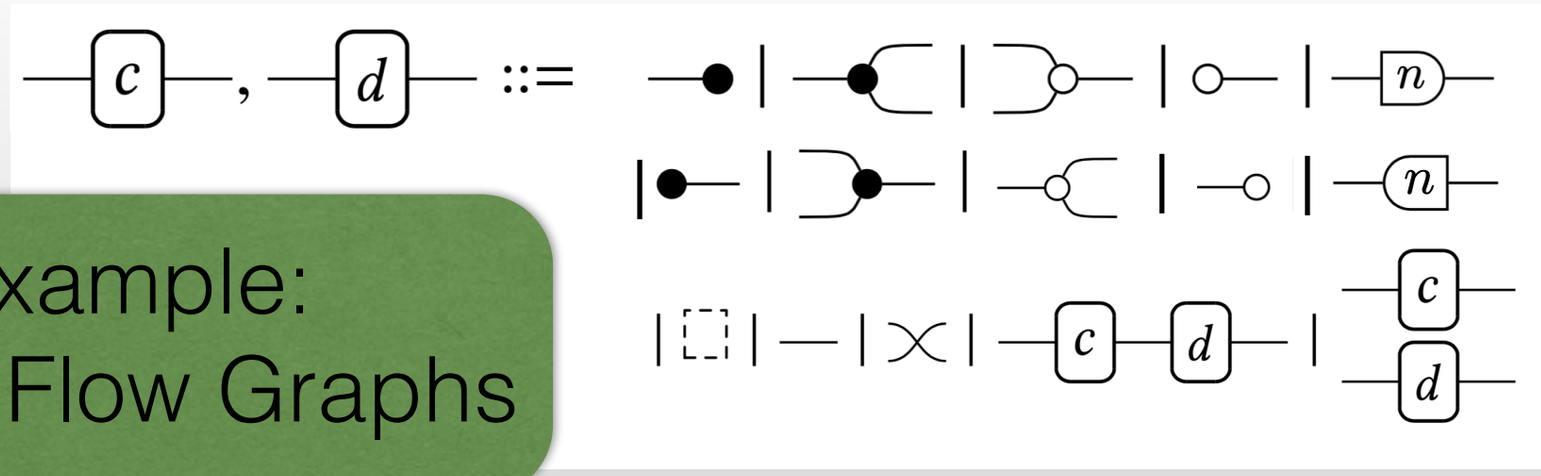


? ? R_1 ; R_2 = R

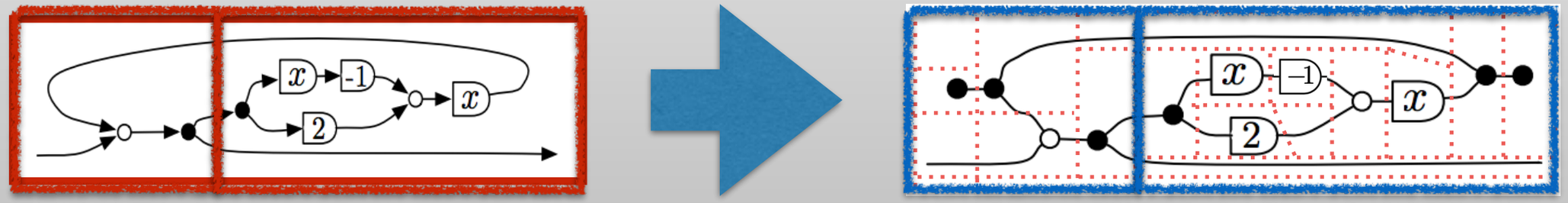
Key Ideas

Graphical notation becomes **formal syntax**

and it is given **compositional** semantics



Example:
Signal Flow Graphs

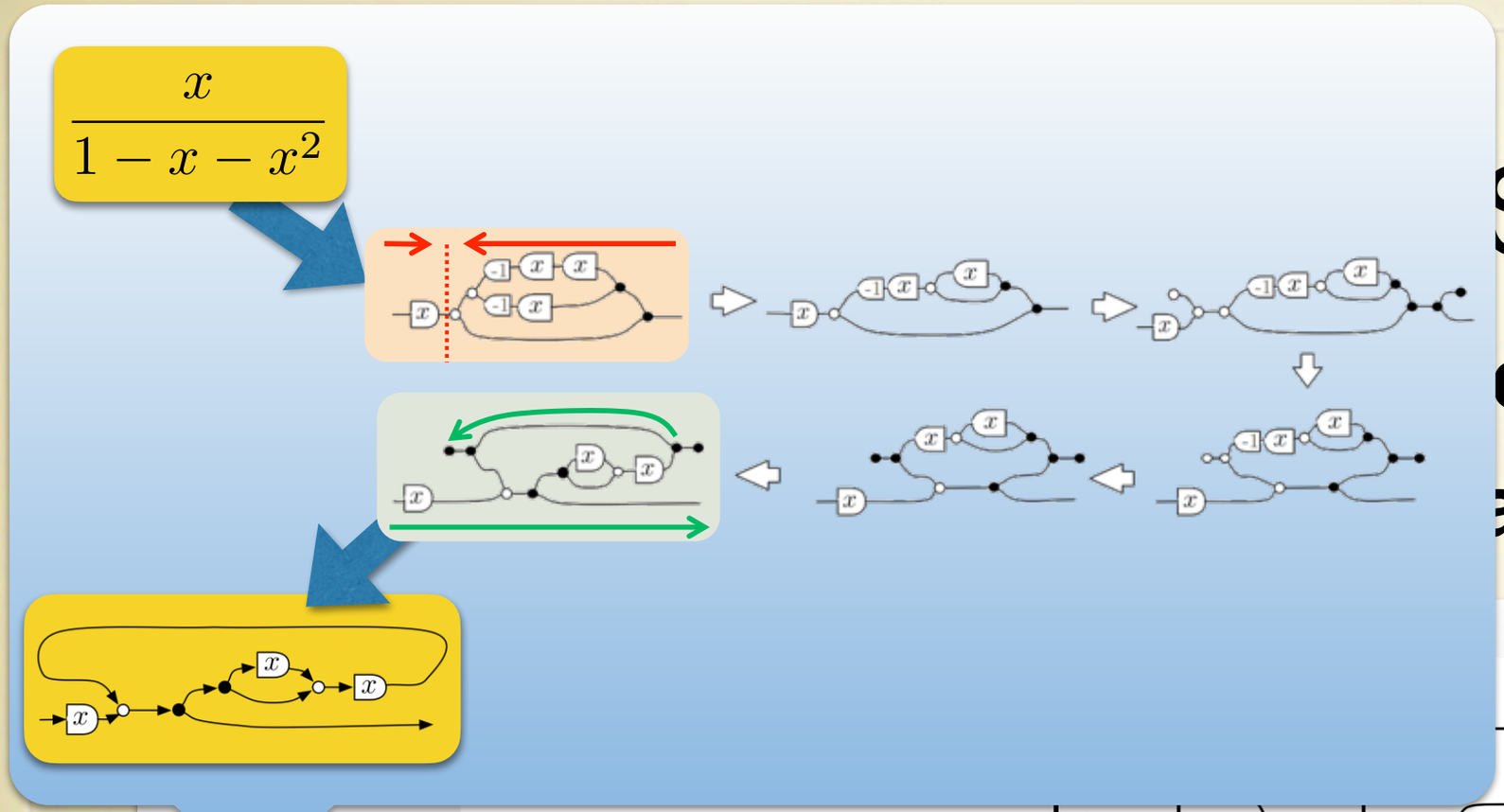


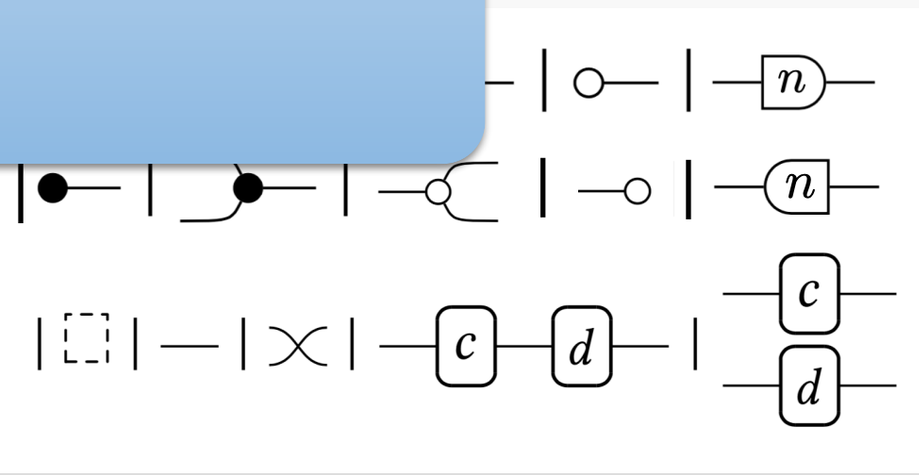
? ? R_1 ; R_2 = R

S

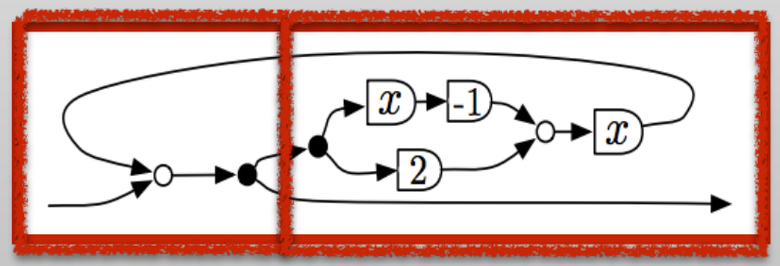
ormal syntax

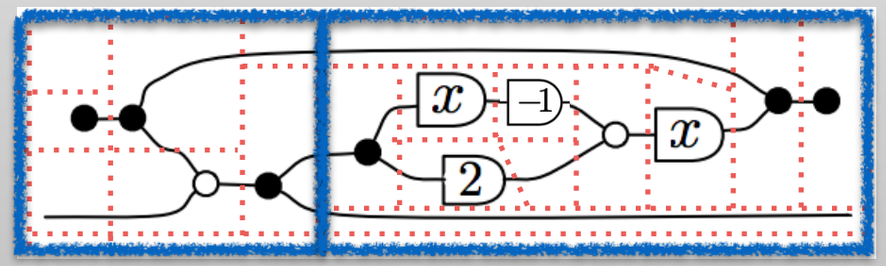
al semantics





Example:
Signal Flow Graphs



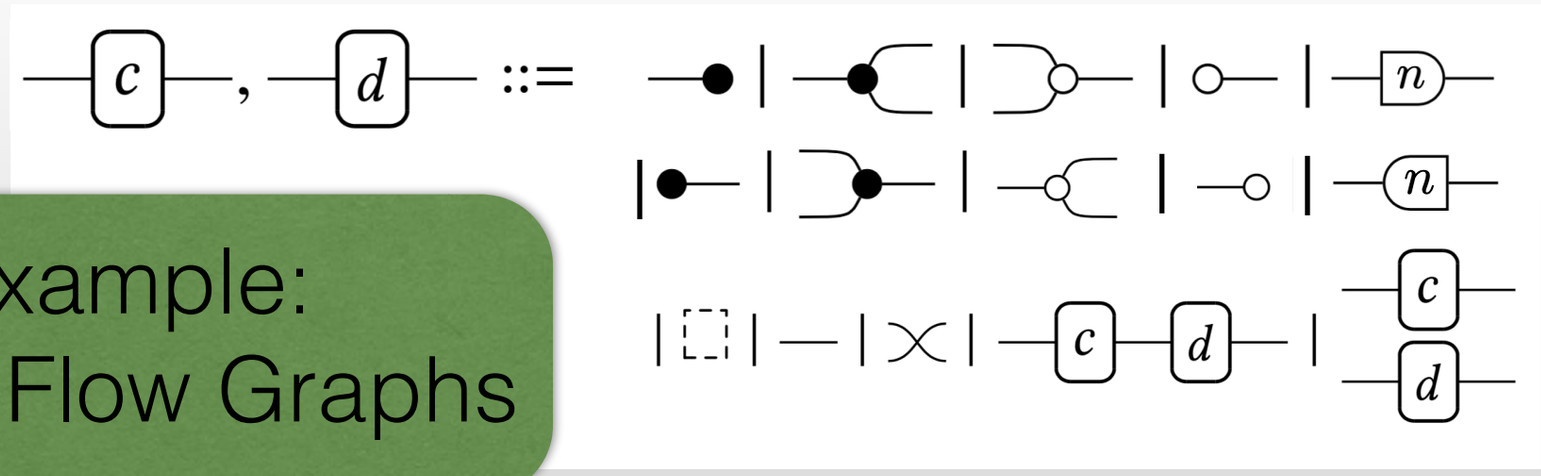


? ? R_1 ; R_2 = R

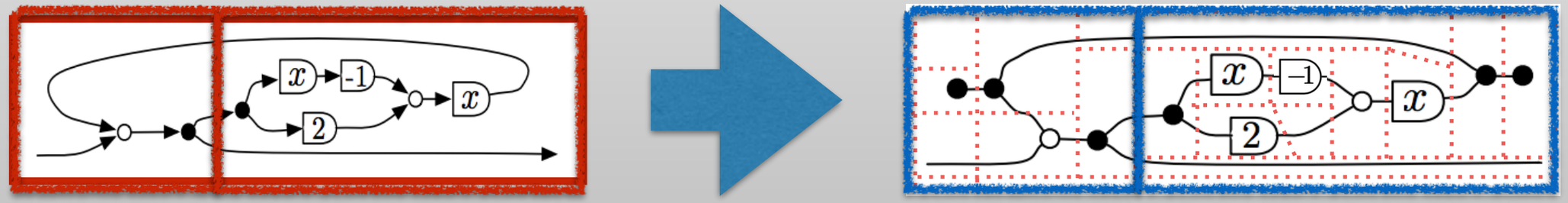
Key Ideas

Graphical notation becomes **formal syntax**

and it is given **compositional** semantics

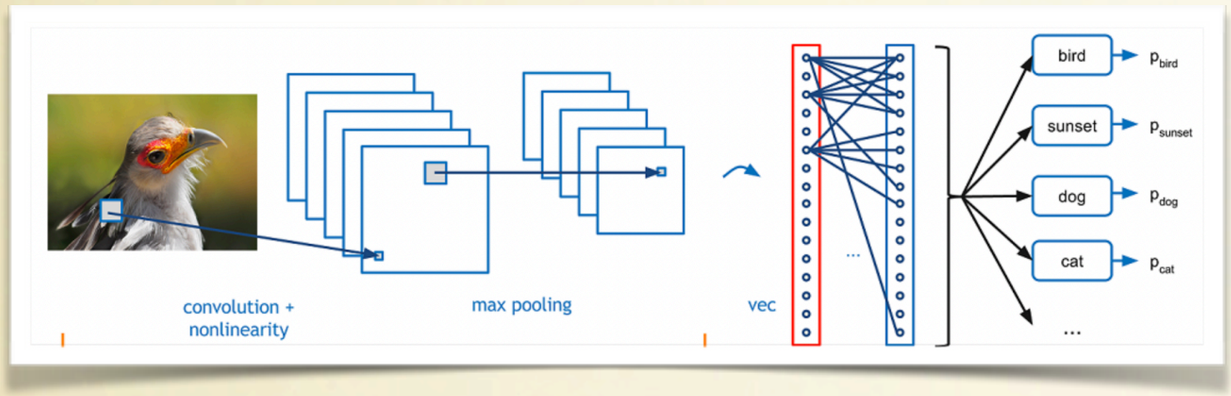


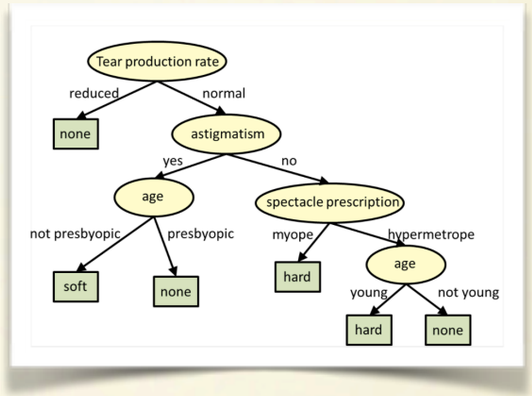
Example:
Signal Flow Graphs

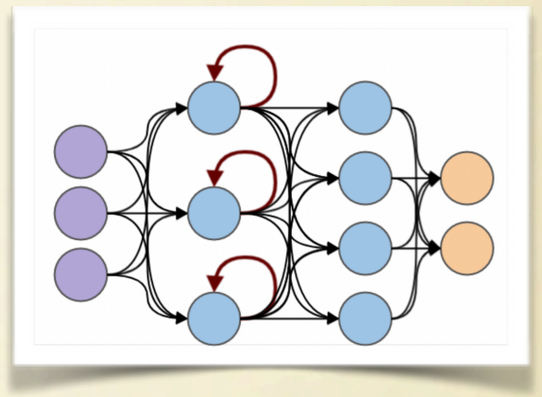


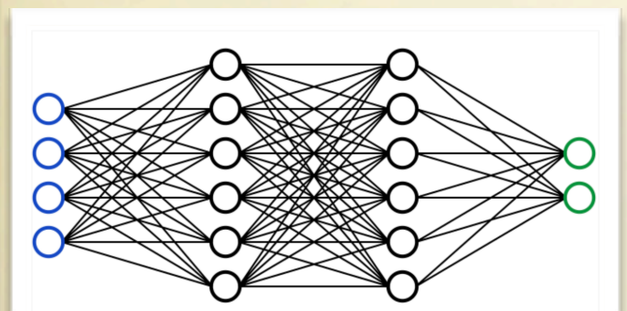
? ? R_1 ; R_2 = R

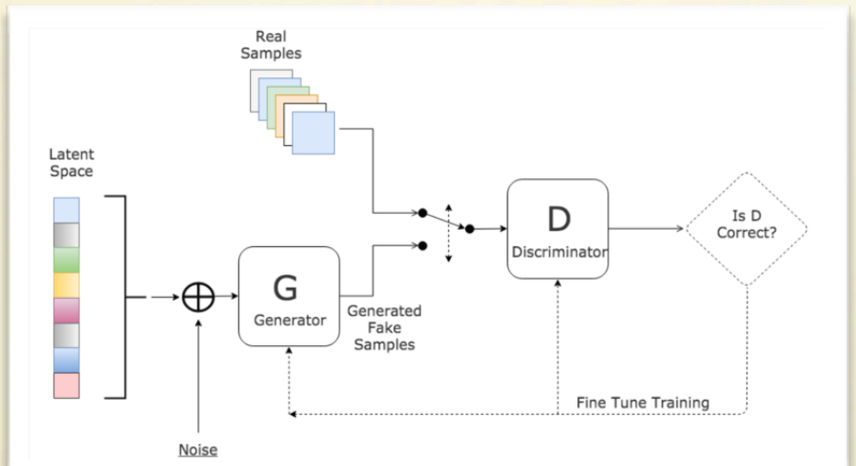
Categorical Foundations of Machine Learning?

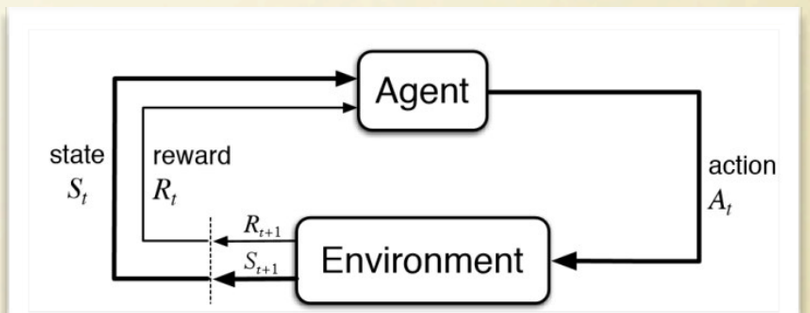




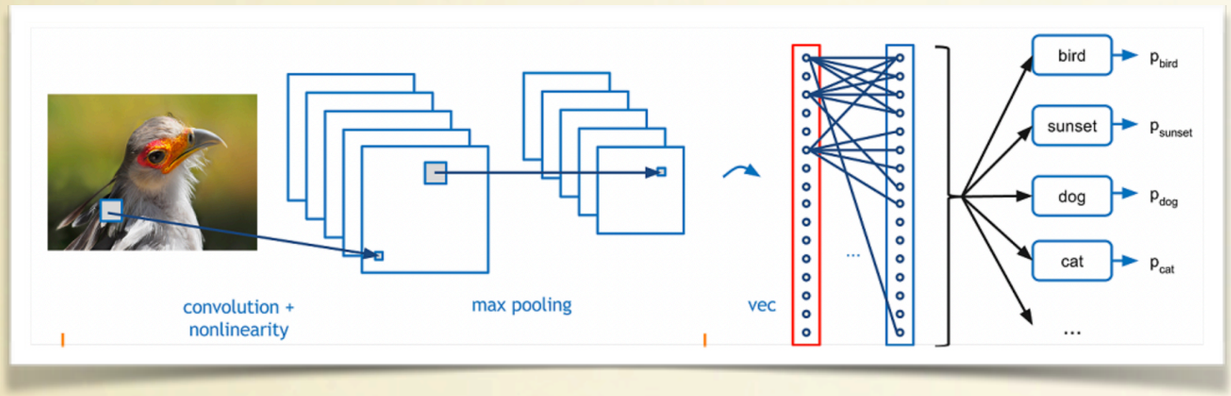


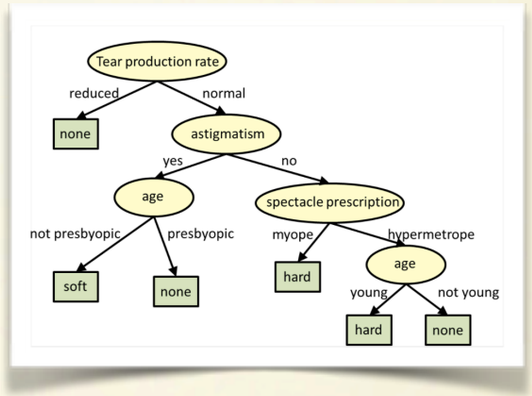


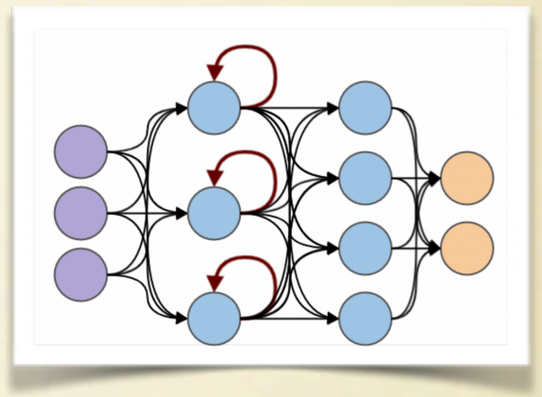


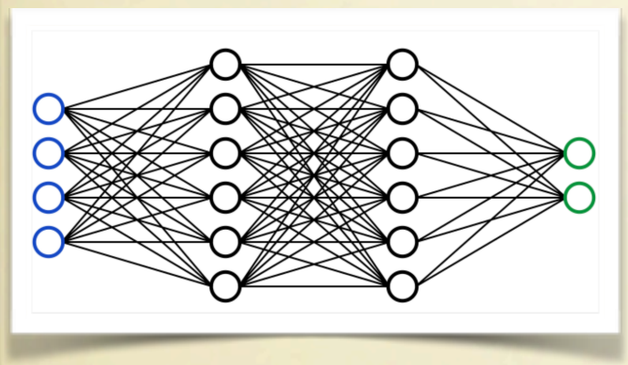


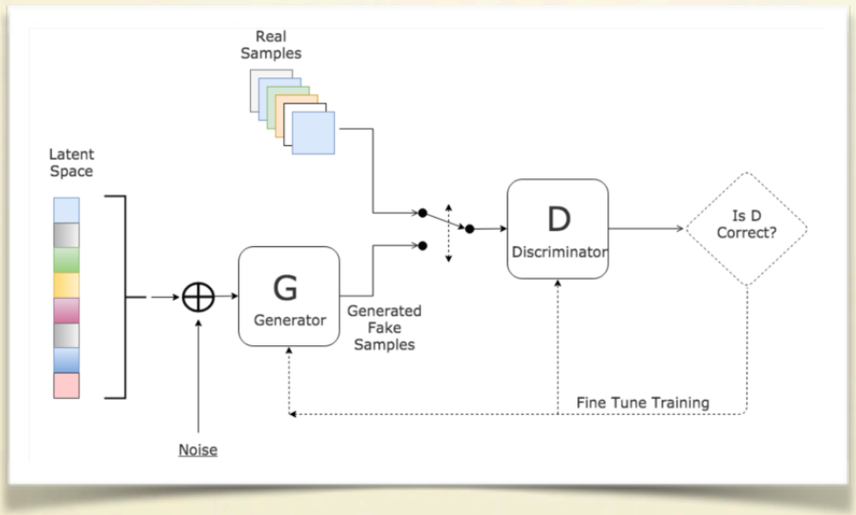
Categorical Foundations of Machine Learning?

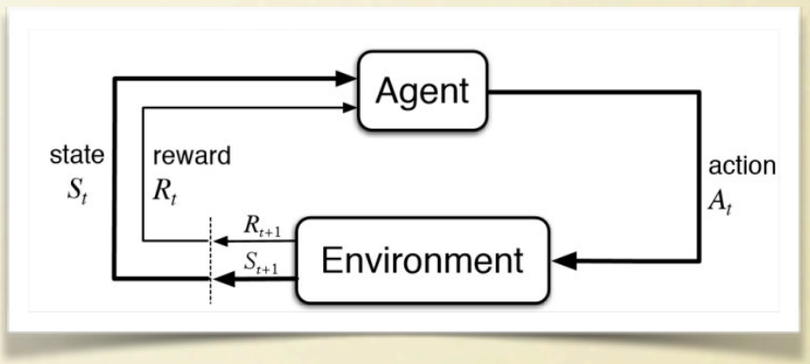












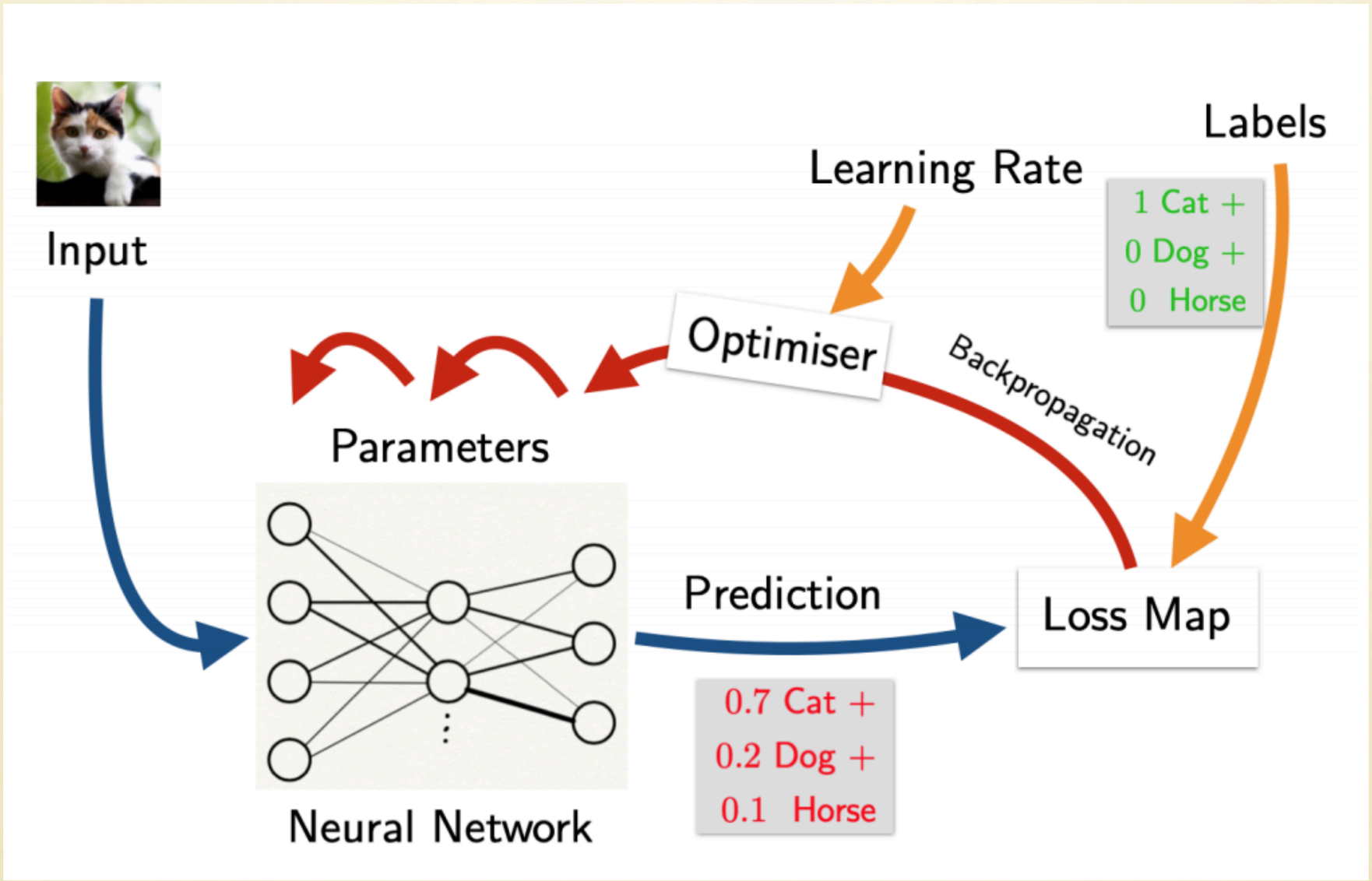
Uniformity

Explainability

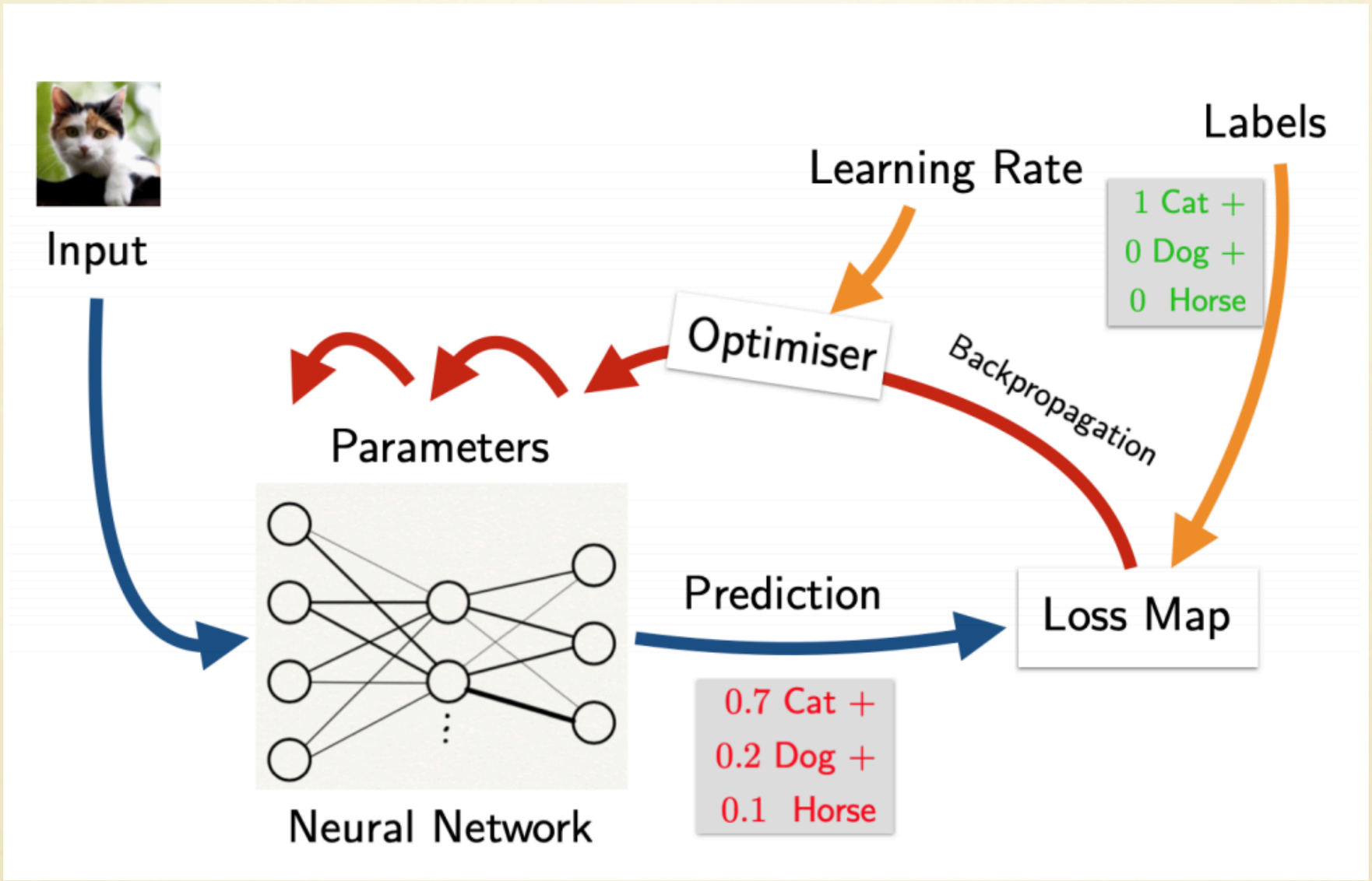
Prog. Principles

Trustworthiness

Case study: Gradient-Based Learning

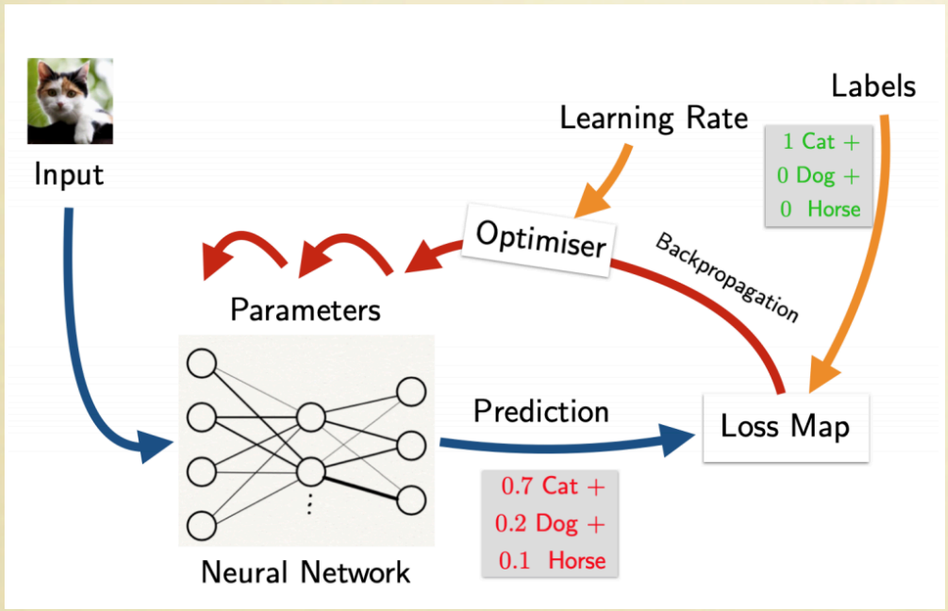


Case study: Gradient-Based Learning

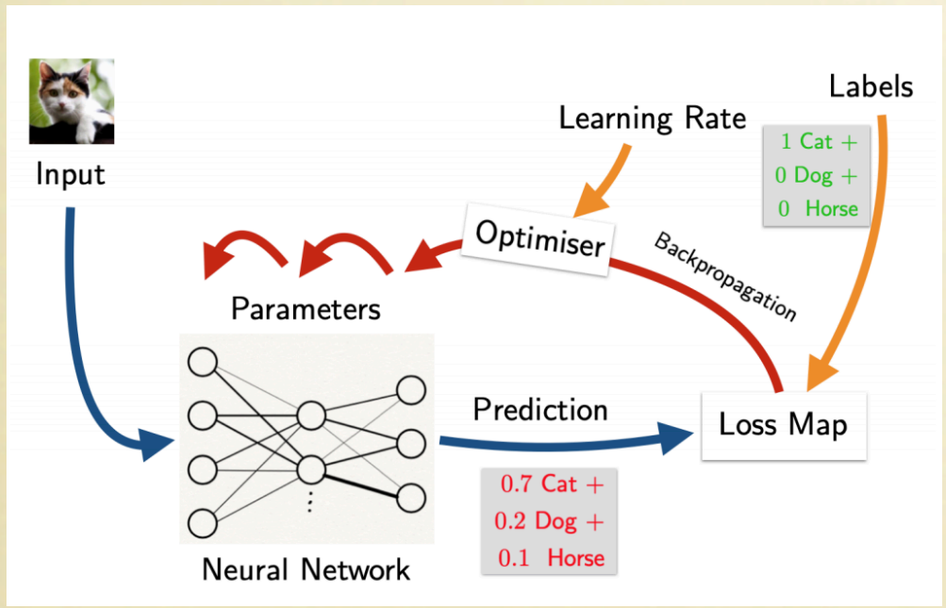


What are the fundamental mathematical structures underpinning gradient-based learning?

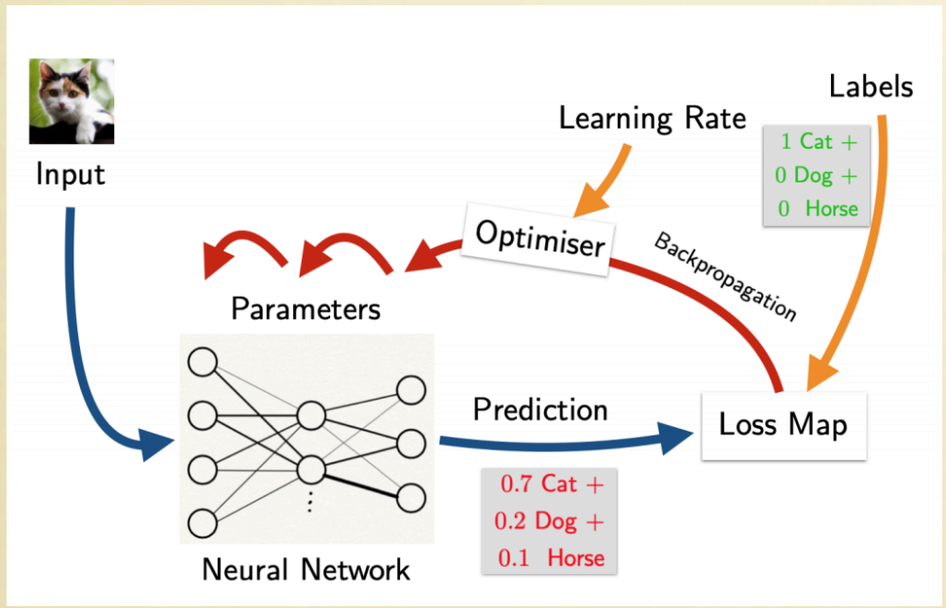
Case study: Gradient-Based Learning

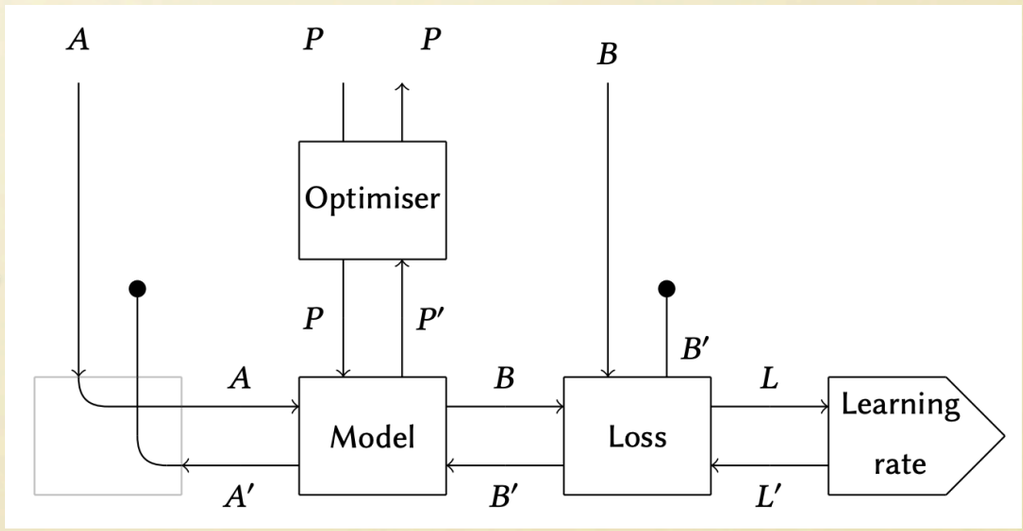


Roadmap


Input
Learning Rate
Labels
1 Cat +
0 Dog +
0 Horse
Optimiser
Backpropagation
Parameters
Prediction
Loss Map
0.7 Cat +
0.2 Dog +
0.1 Horse
Neural Network

Roadmap


Input
Learning Rate
Labels
1 Cat +
0 Dog +
0 Horse
Optimiser
Backpropagation
Parameters
Prediction
Loss Map
0.7 Cat +
0.2 Dog +
0.1 Horse
Neural Network


A
P
P
B
Optimiser
P
P′
A
B
B′
L
Learning rate
Model
Loss
A′
B′
L′

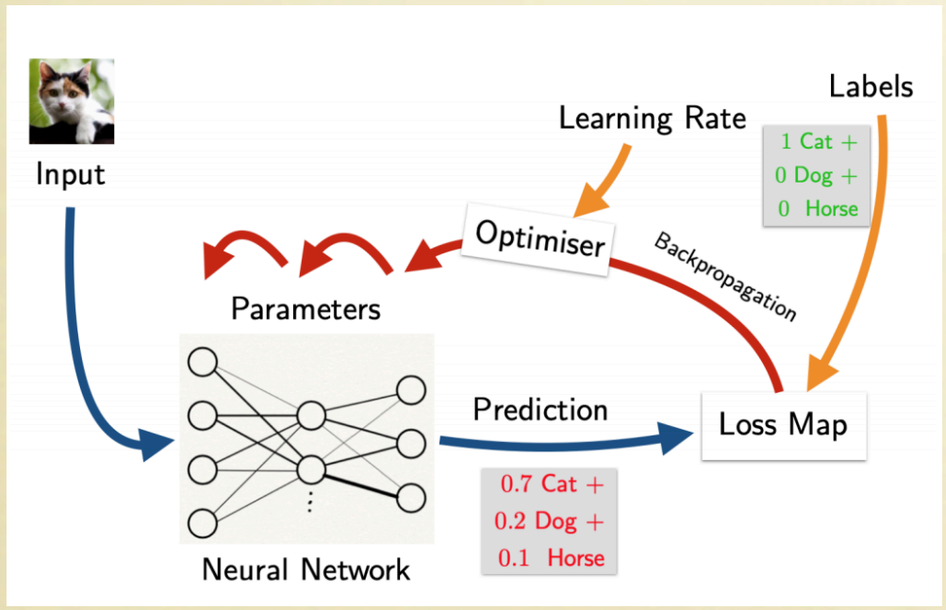
Roadmap

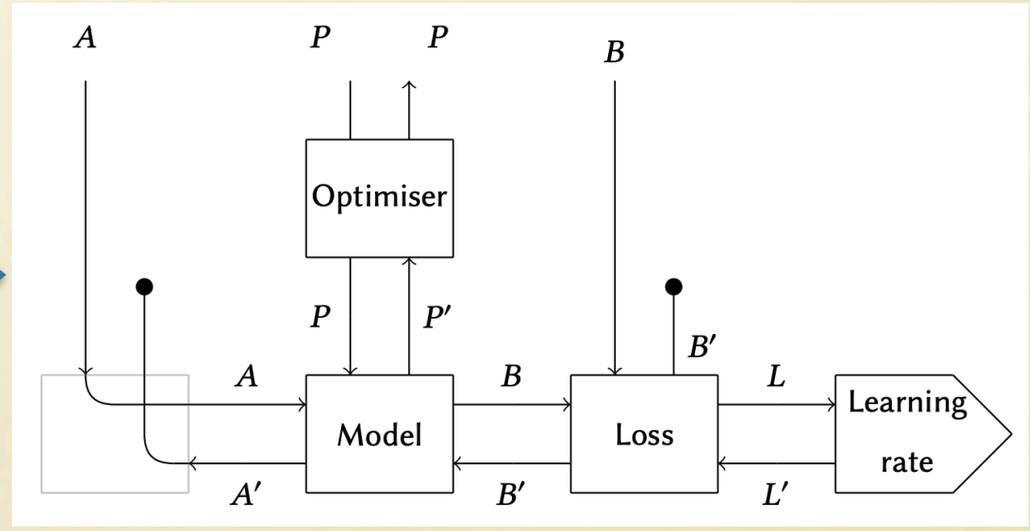
Plan:

Deconstruct learning into simple,
uniform components, called **parametric lenses**

(We express parametric lenses
as ***string diagrams*** in a suitable category)

Learning will be defined by ***lens composition.***





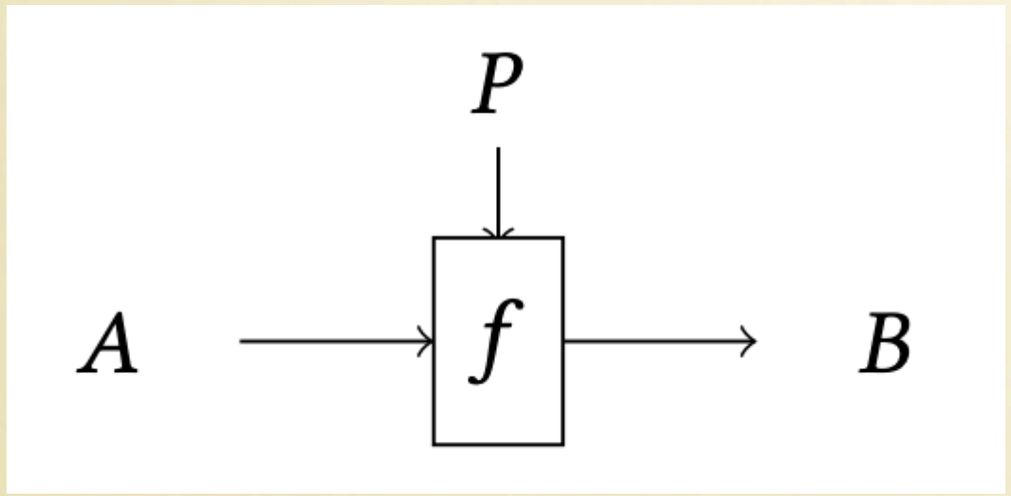
Parametric Lenses

Parametric lens = parametric map with lens structure

Parametric Lenses

Parametric lens = parametric map with lens structure

$$f : P \times A \to B$$

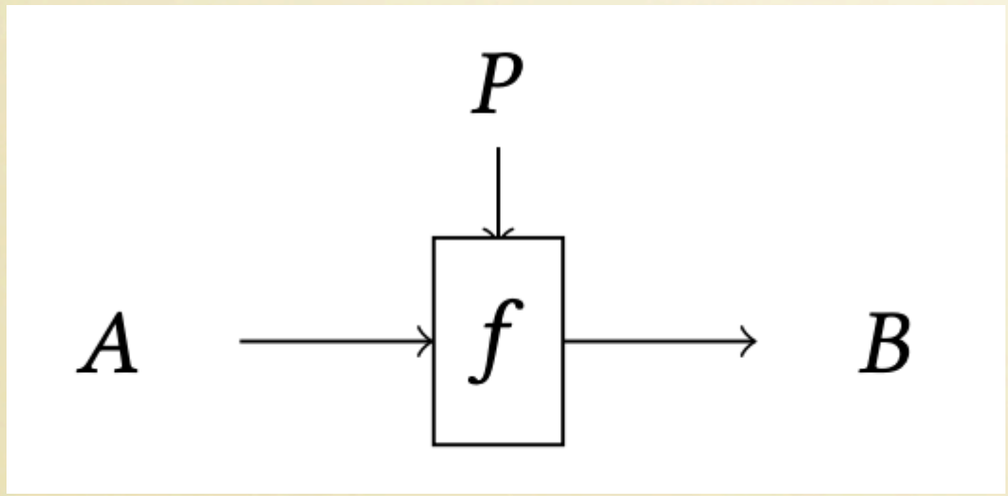


Para(C)

Parametric Lenses

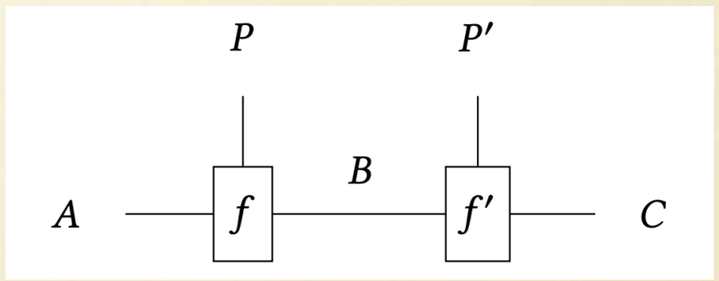
Parametric lens = parametric map with lens structure

$$f : P \times A \to B$$

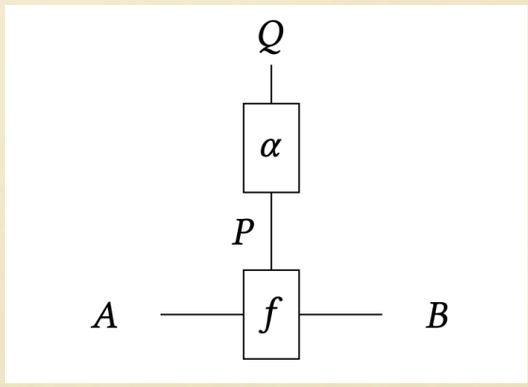


Para(C)

Composition

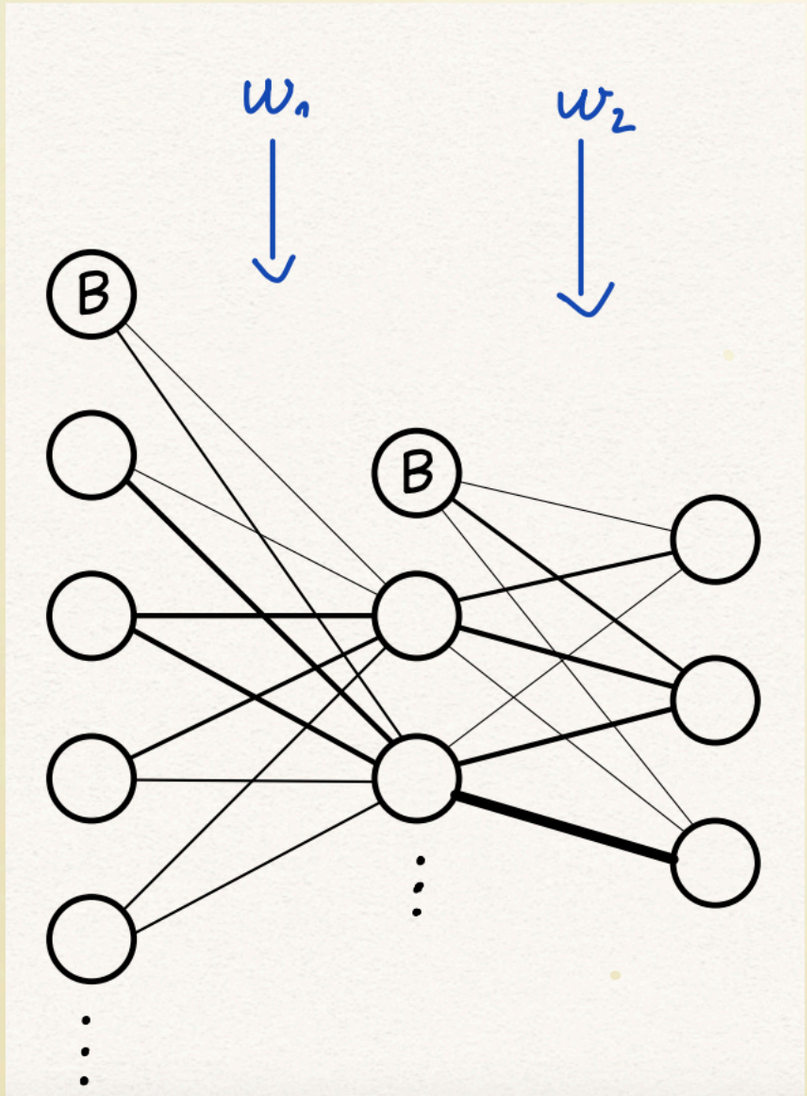


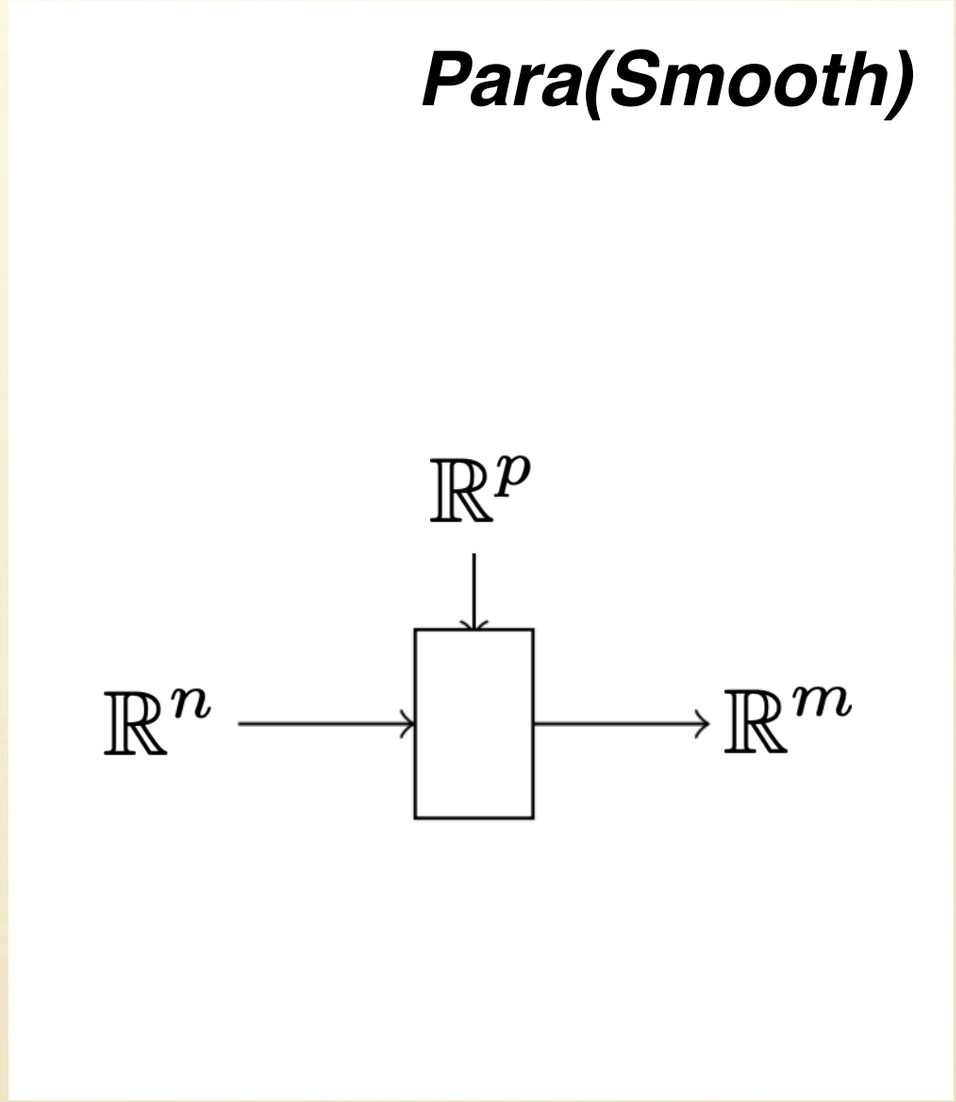
Reparametrisation



Parametric Maps

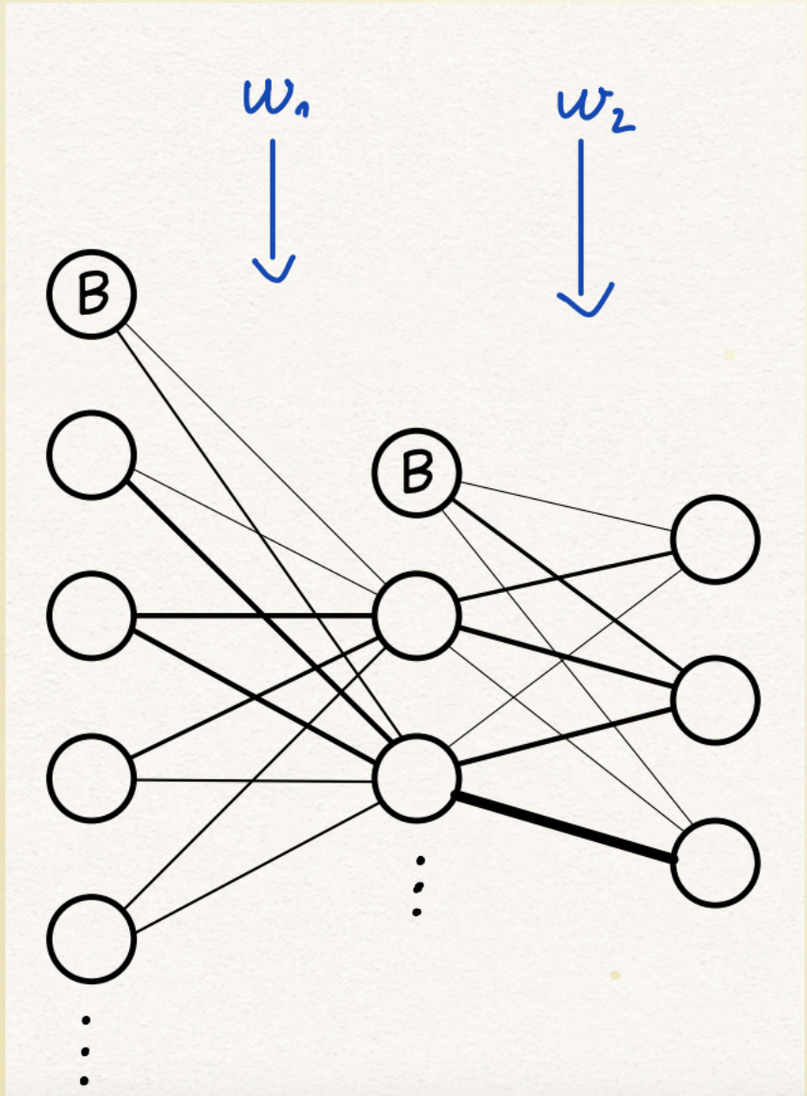
A neural network is a parametric map



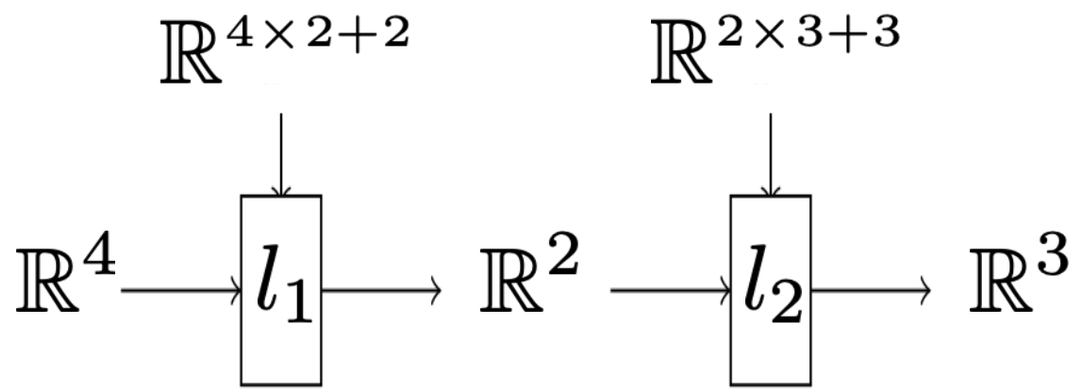


Parametric Maps

A neural network is a parametric map



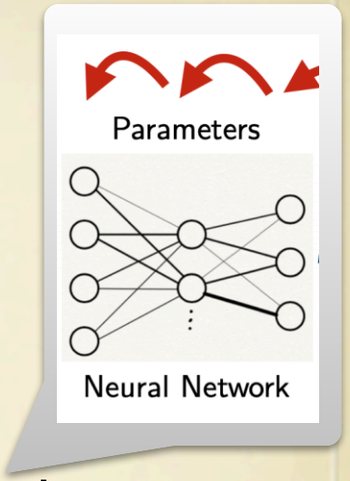
Para(Smooth)



Lenses

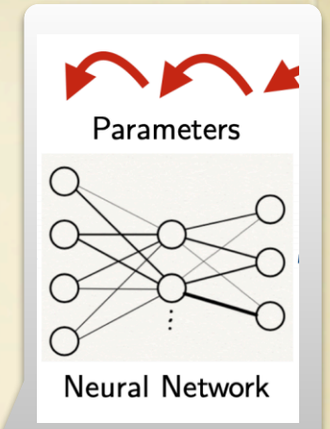
Parametric lens = parametric map with lens structure

Lenses



Parametric lens = parametric map with lens structure

Lenses

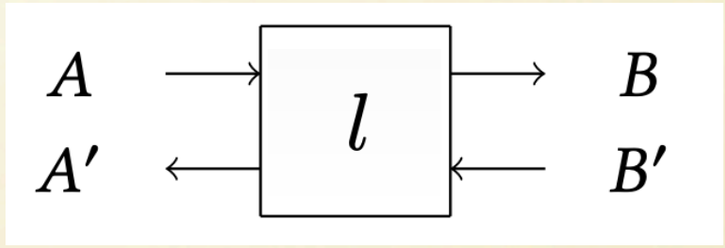


Parametric lens = parametric map with lens structure

$$l : (A, A') \to (B, B')$$

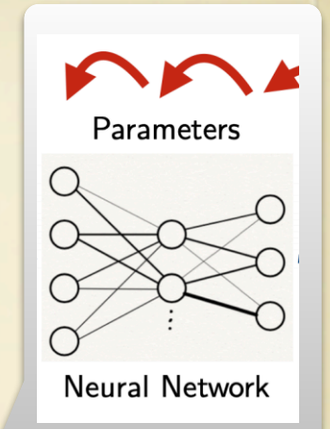
$$\frac{}{\mathsf{get}_l : A \to B}$$

$$\mathsf{put}_l : A \times B' \to A'$$



Lens(C)

Lenses

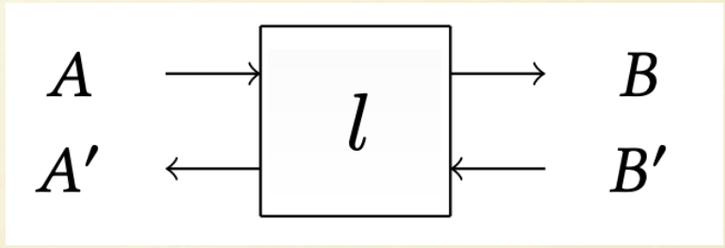


Parametric lens = parametric map with lens structure

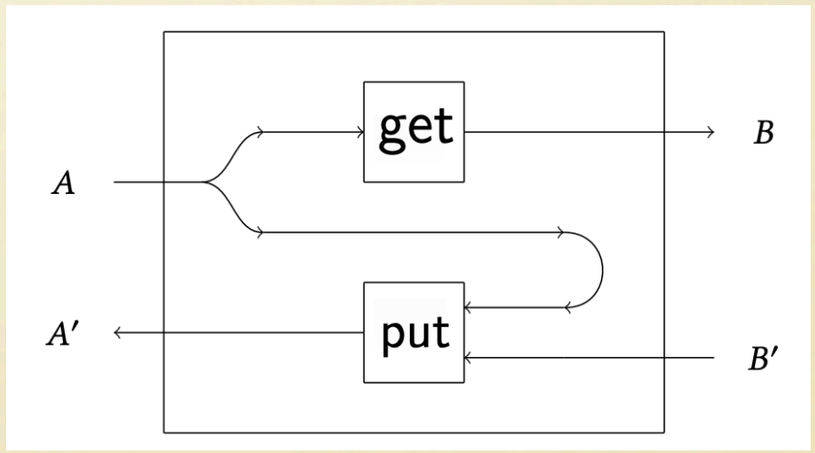
$$l : (A, A') \to (B, B')$$

$$\frac{}{\mathsf{get}_l : A \to B}$$

$$\mathsf{put}_l : A \times B' \to A'$$



`Opening the box':

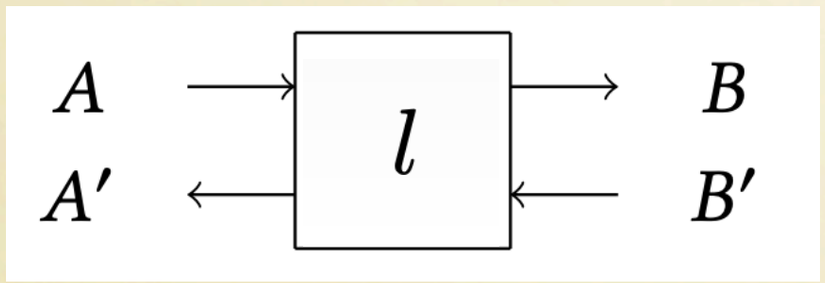


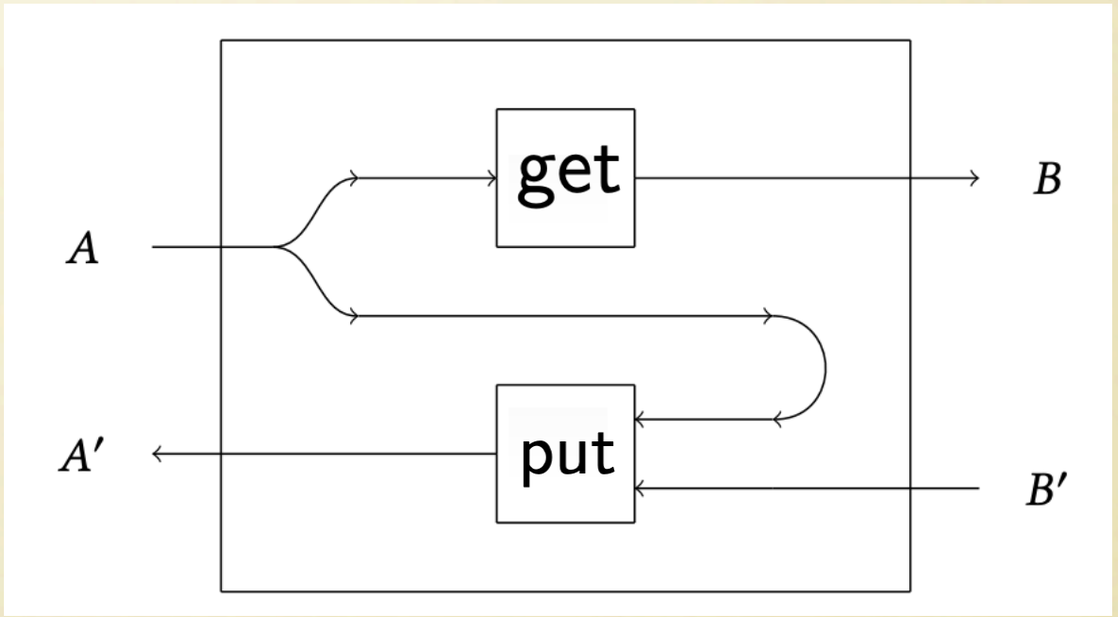
Lens(C)

String Diagrams for Lenses

We use string diagrams in $\mathbf{Tamb}(\mathcal{C})$

Faithful embedding of $\mathbf{Lens}(\mathcal{C})$ in $[\mathbf{Lens}(\mathcal{C})^{op}, \mathbf{Set}] \cong \mathbf{Tamb}(\mathcal{C})$





Lenses

A neural network is a lens in **Smooth**

$$l\colon (A, A') \to (B, B')$$

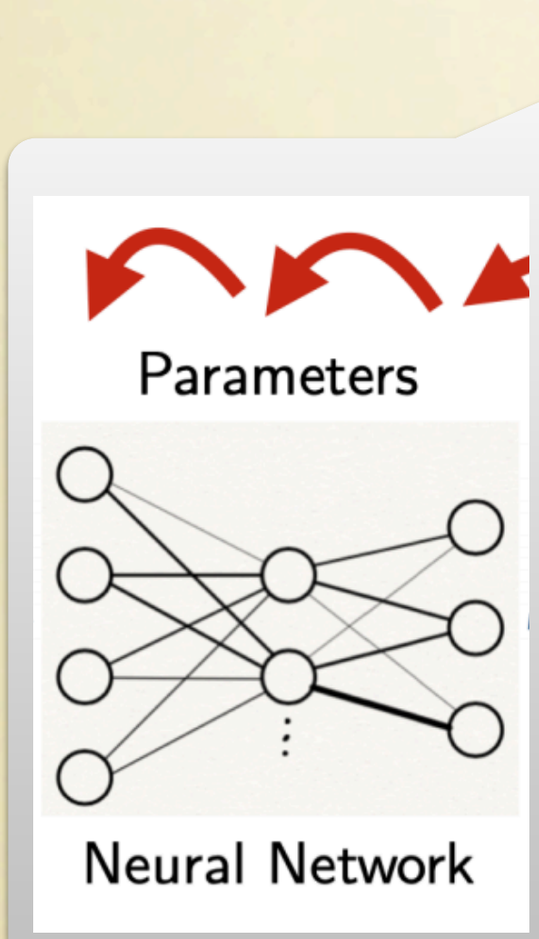
$$f\colon A \to B$$

$$R[f]\colon A \times B' \to A'$$

$$(\vec{x}, \vec{y}) \mapsto J_f(\vec{x})^T \cdot \vec{y}$$

Lenses

A neural network is a lens in **Smooth**



$$l\colon (A, A') \to (B, B')$$

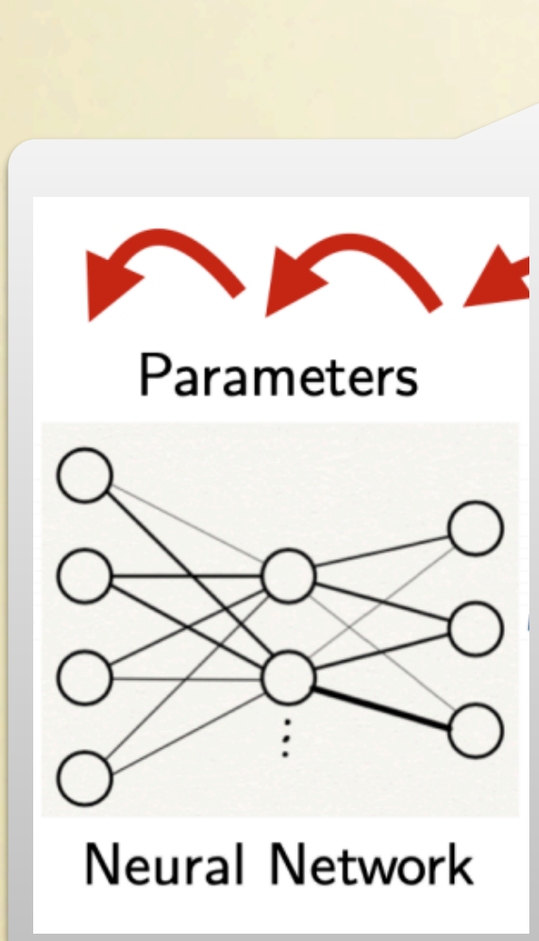
$$\frac{}{f\colon A \to B}$$

$$R[f]\colon A \times B' \to A'$$

$$(\vec{x}, \vec{y}) \mapsto J_f(\vec{x})^T \cdot \vec{y}$$

Lenses

A neural network is a lens in **Smooth**



$$l\colon (A, A') \to (B, B')$$

$$\frac{}{f\colon A \to B}$$

$$R[f]\colon A \times B' \to A'$$

$$(\vec{x}, \vec{y}) \mapsto J_f(\vec{x})^T \cdot \vec{y}$$

Reverse derivative of *f*

Reverse derivative categories *

Robin Cockett
University of Calgary, Department of Computer Science, Canada
robin@ucalgary.ca

Geoffrey Cruttwell
Mount Allison University, Department of Mathematics and Computer Science, Ca
gcruttwell@mta.ca

Jonathan Gallagher
Dalhousie University, Department of Mathematics and Statistics, Canada
jonathan.gallagher@dal.ca

Jean-Simon Pacaud Lemay
University of Oxford, Department of Computer Science, UK
jean-simon.lemay@kellogg.ox.ac.uk

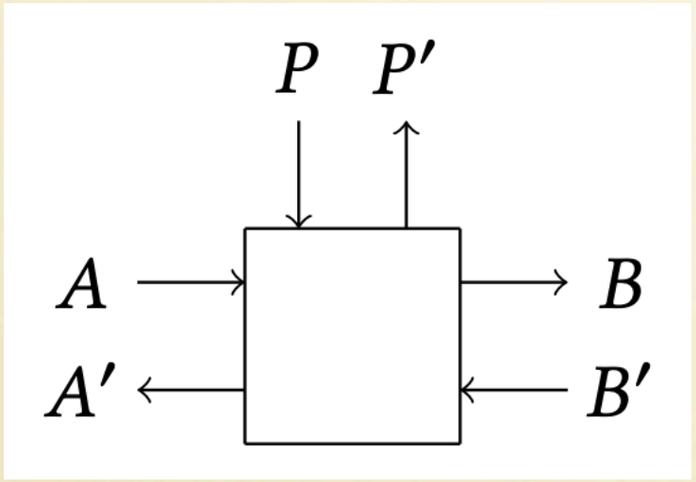
Benjamin MacAdam
University of Calgary, Department of Computer Science, Canada
benjamin.macadam@ucalgary.ca

Gordon Plotkin
Google Research
gdp@inf.ed.ac.uk

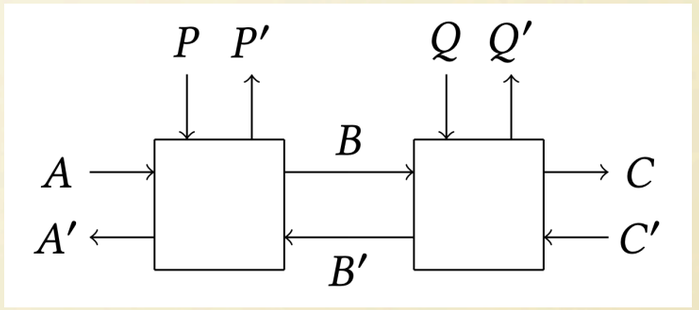
Dorette Pronk
Dalhousie University, Department of Mathematics and Statistics, Canada

Parametric Lenses

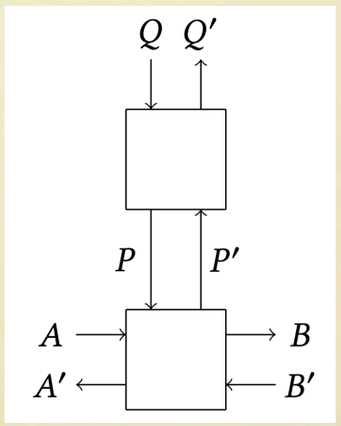
Parametric lens = parametric map with lens structure



Composition



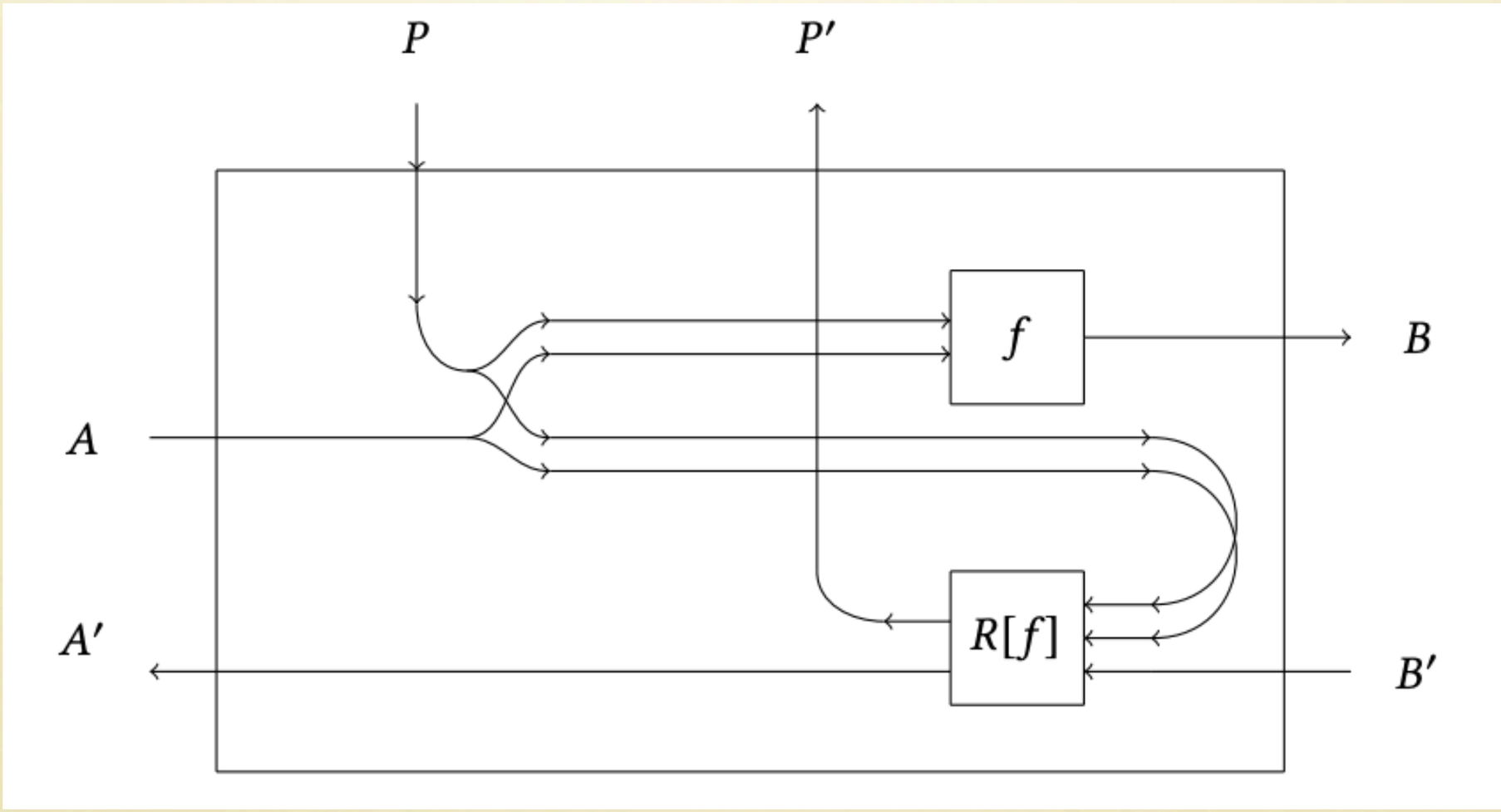
Reparametrisation



Para(Lens(C))

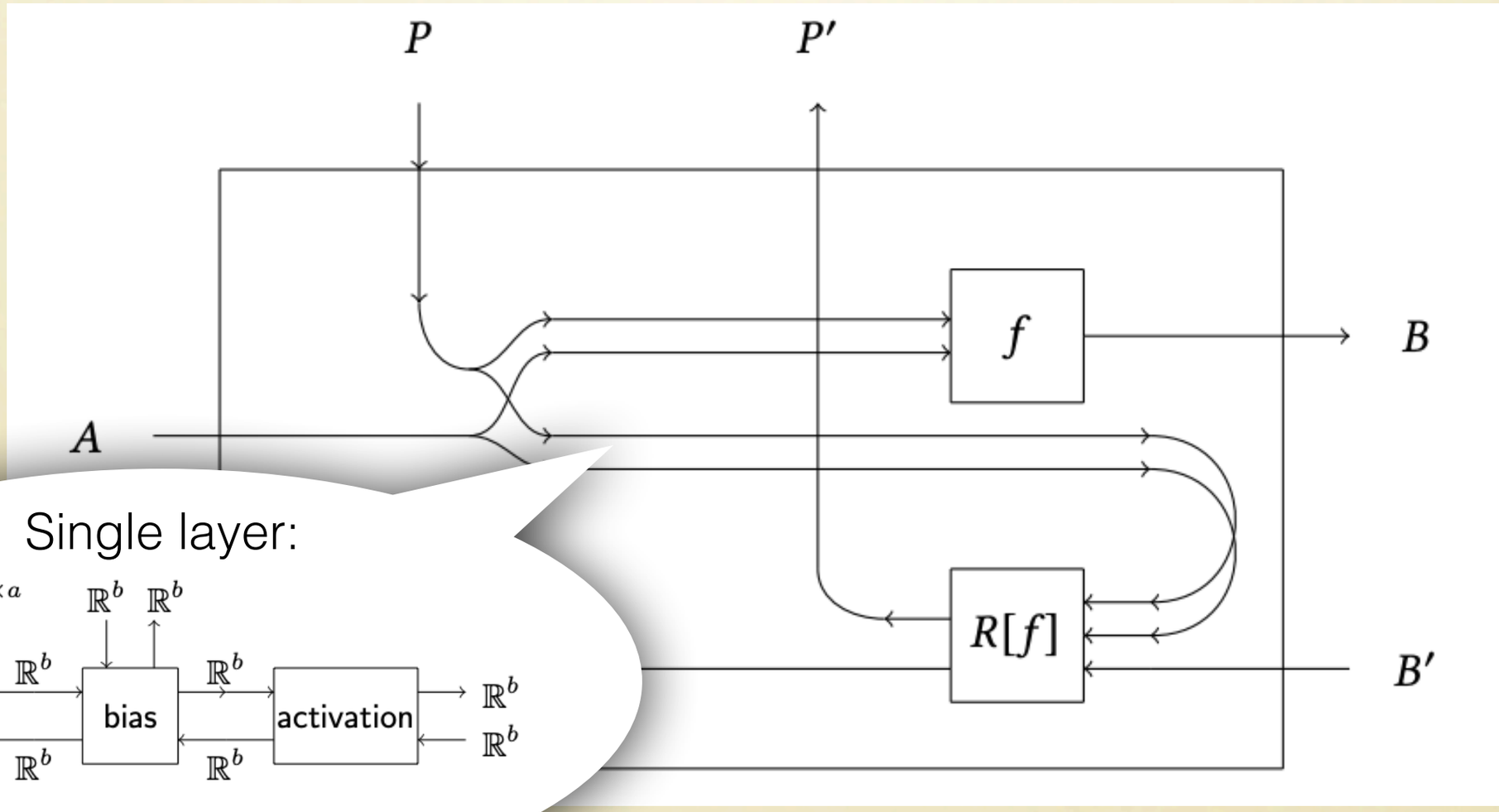
Parametric Lenses

A neural network is a parametric lens



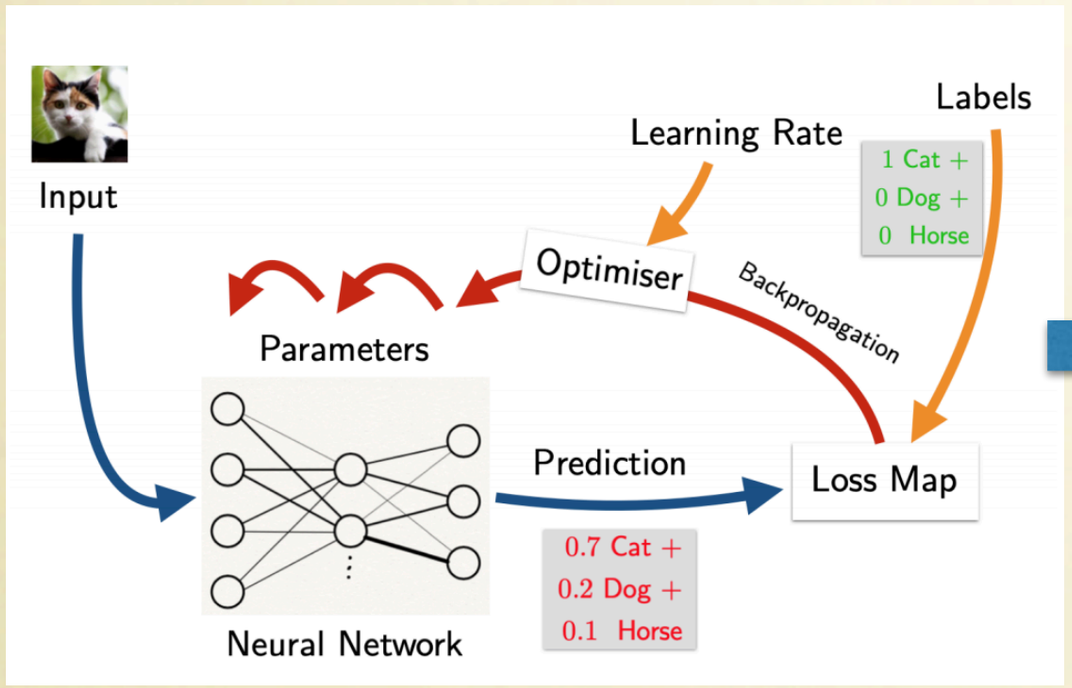
Parametric Lenses

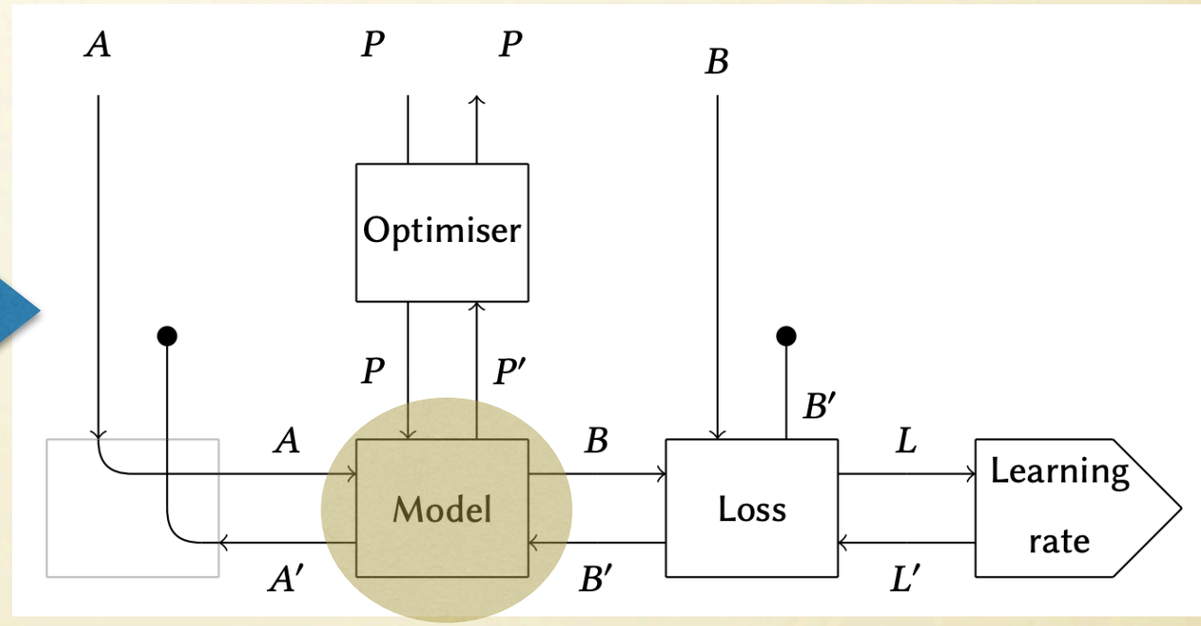
A neural network is a parametric lens



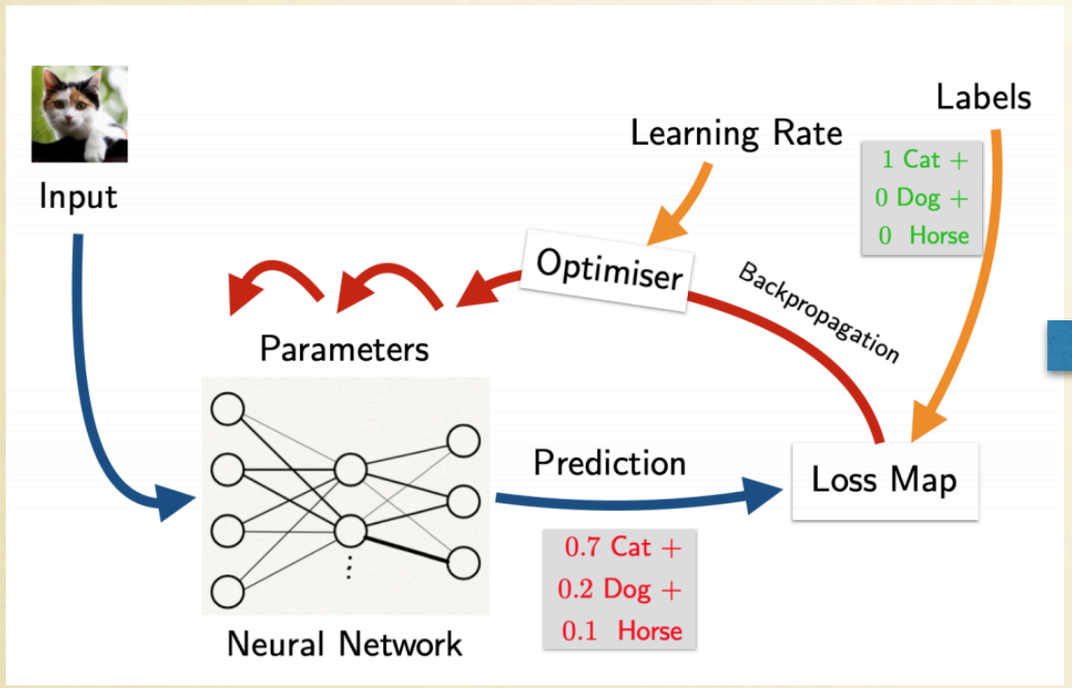
Single layer:

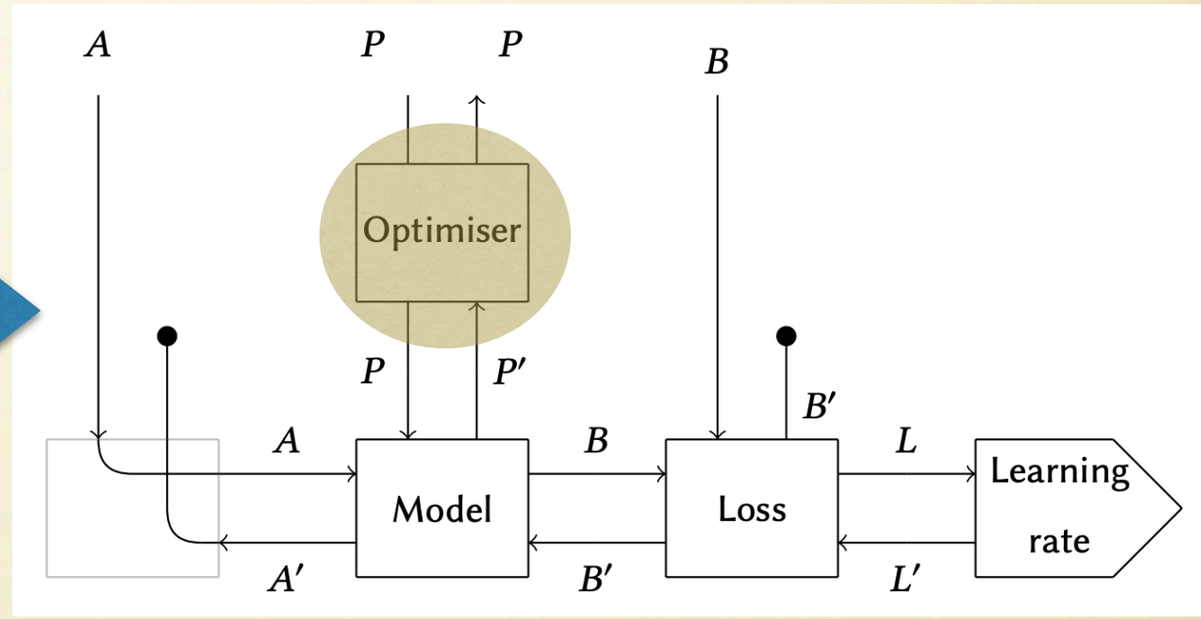
Where we are, so far



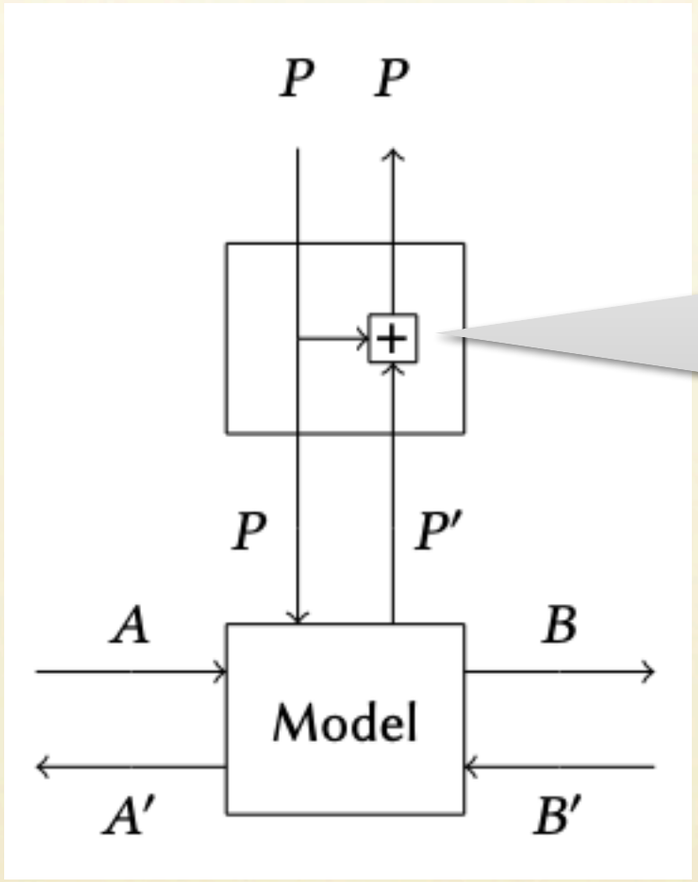


Where we are, so far





Optimisers are reparametrisations



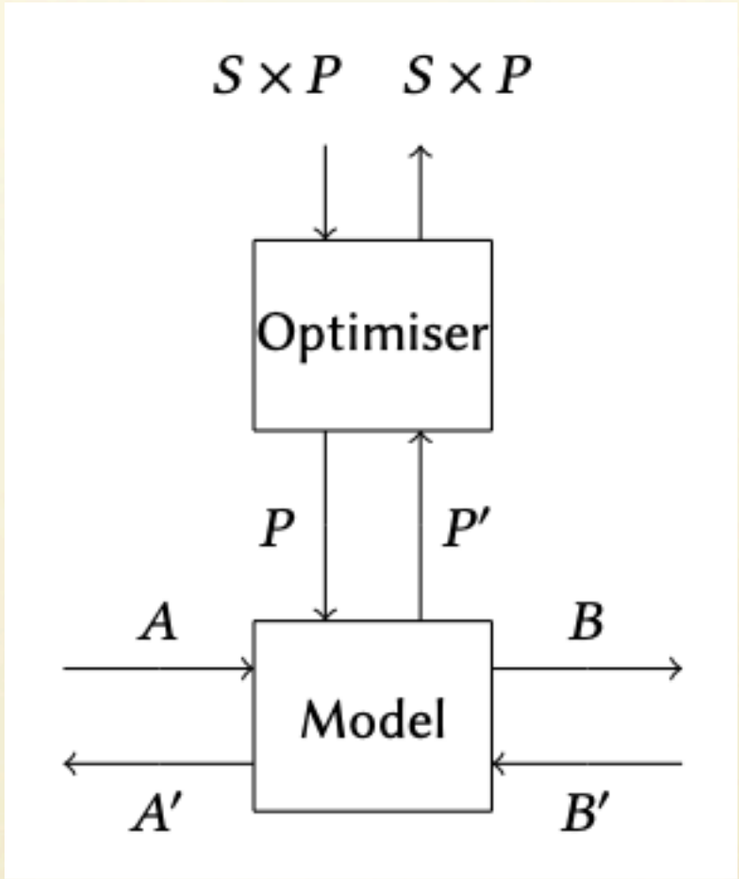
Lens(Smooth)

$$\frac{\mathsf{gd}\colon (P,P) \to (P,P')}{\mathsf{get}_{gd}\colon p \mapsto p}$$

$$\mathsf{put}_{\mathsf{gd}}\colon (p,p') \mapsto p + p'$$

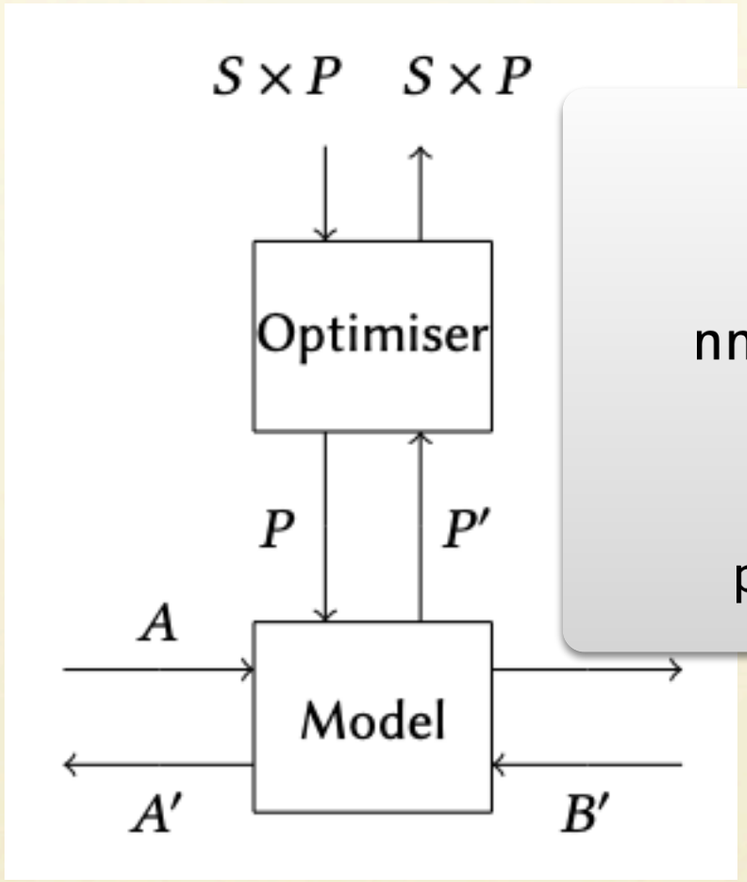
Basic gradient descent

Optimisers are reparametrisations



`Stateful' optimiser such as **Adagrad**, **Adam**, **Nesterov**, **Momentum**, etc. can be all formalised uniformly as reparametrisations.

Optimisers are reparametrisations

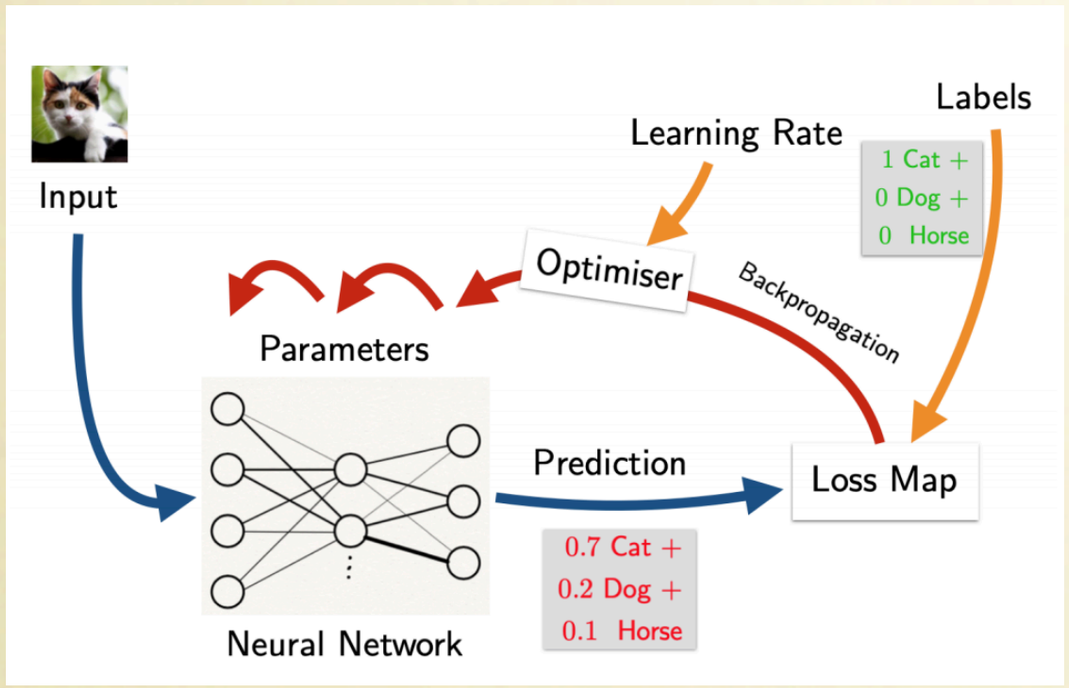


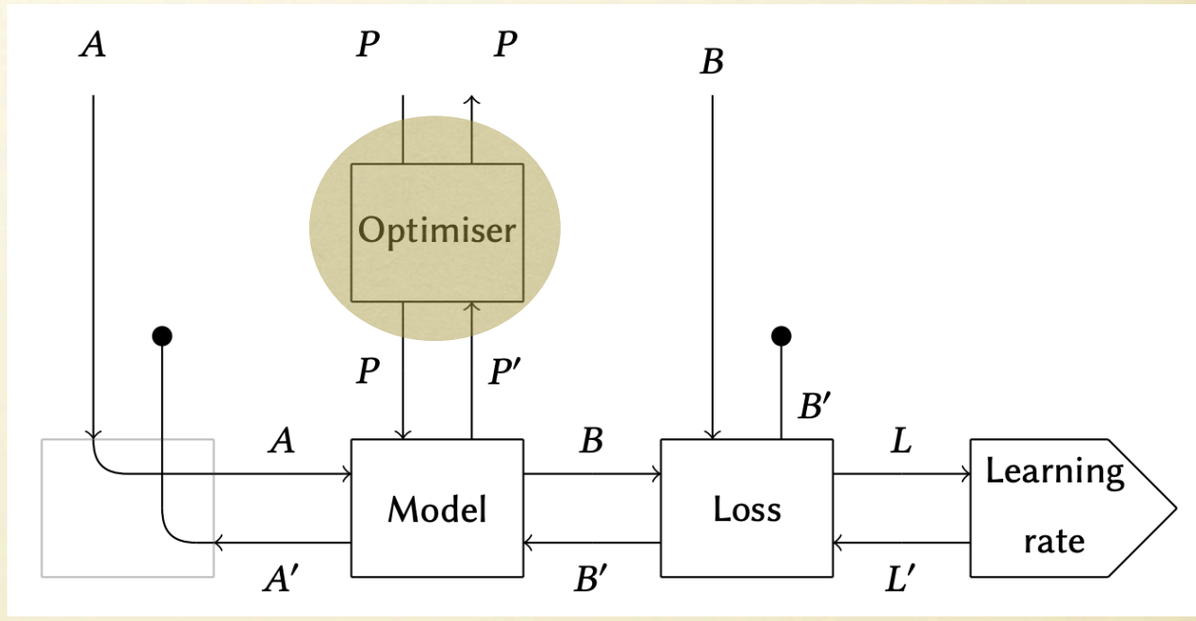
Lens(Smooth)

$$\mathsf{nm} : \frac{(S \times P, S \times P) \to (P, P')}{\mathsf{get}_{\mathsf{nm}} : (s, p) \to p - \gamma s \qquad \mathsf{put}_{nm} : (s, p, p') \mapsto (\gamma s + p', p - s)}$$

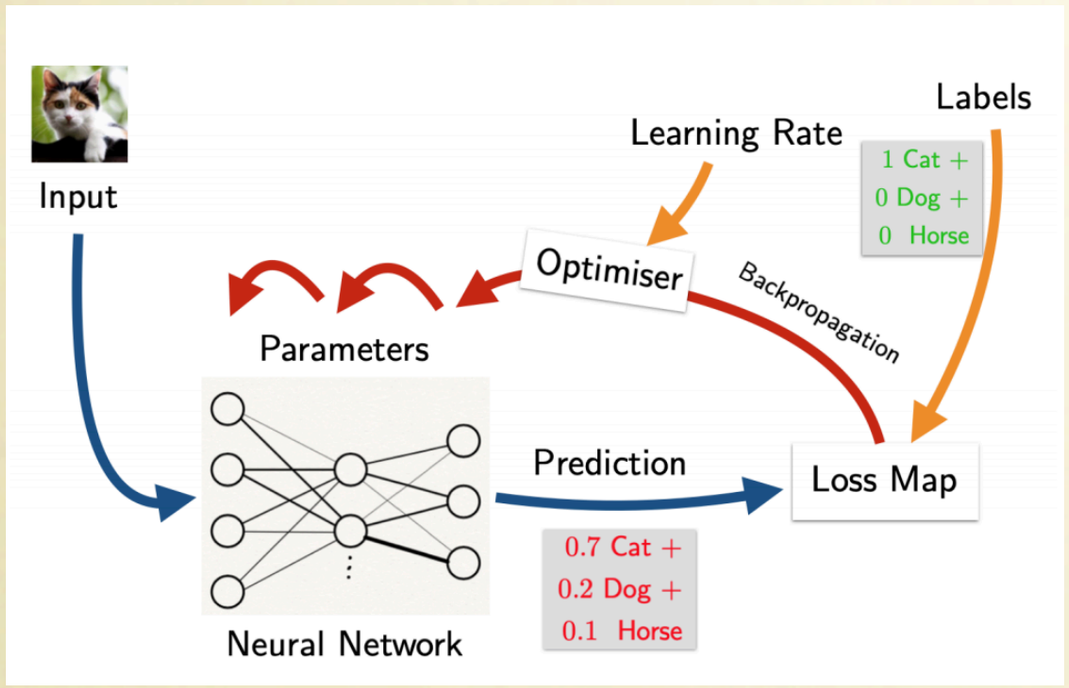
`Stateful' optimiser such as **Adagrad**, **Adam**, **Nesterov**, **Momentum**, etc. can be all formalised uniformly as reparametrisations.

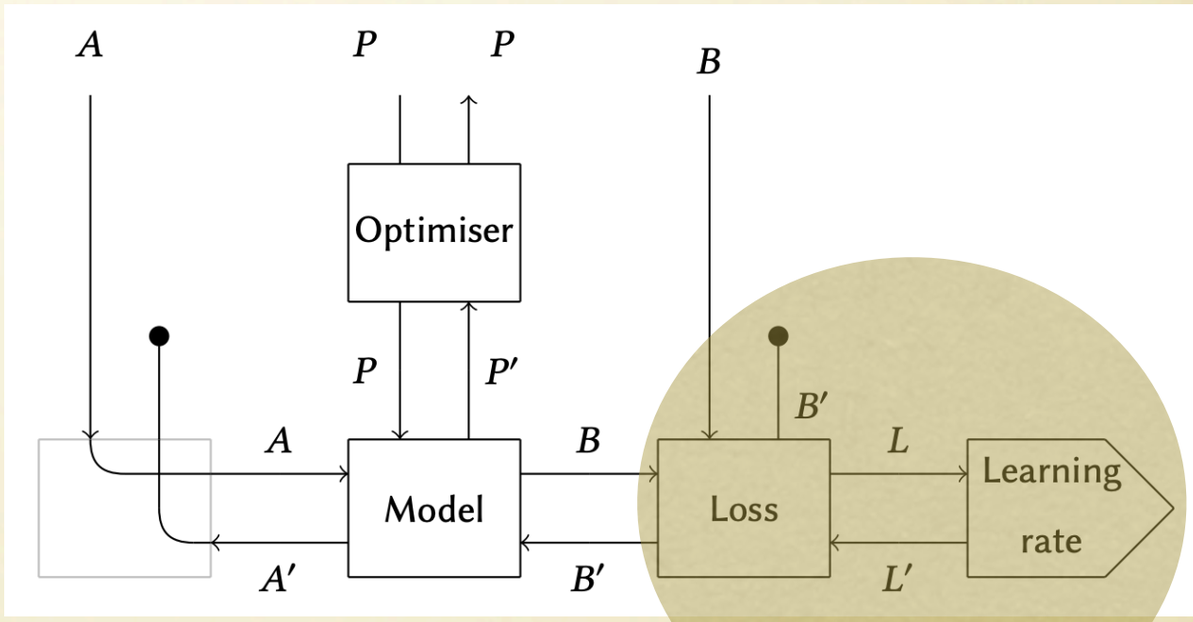
Where we are, so far



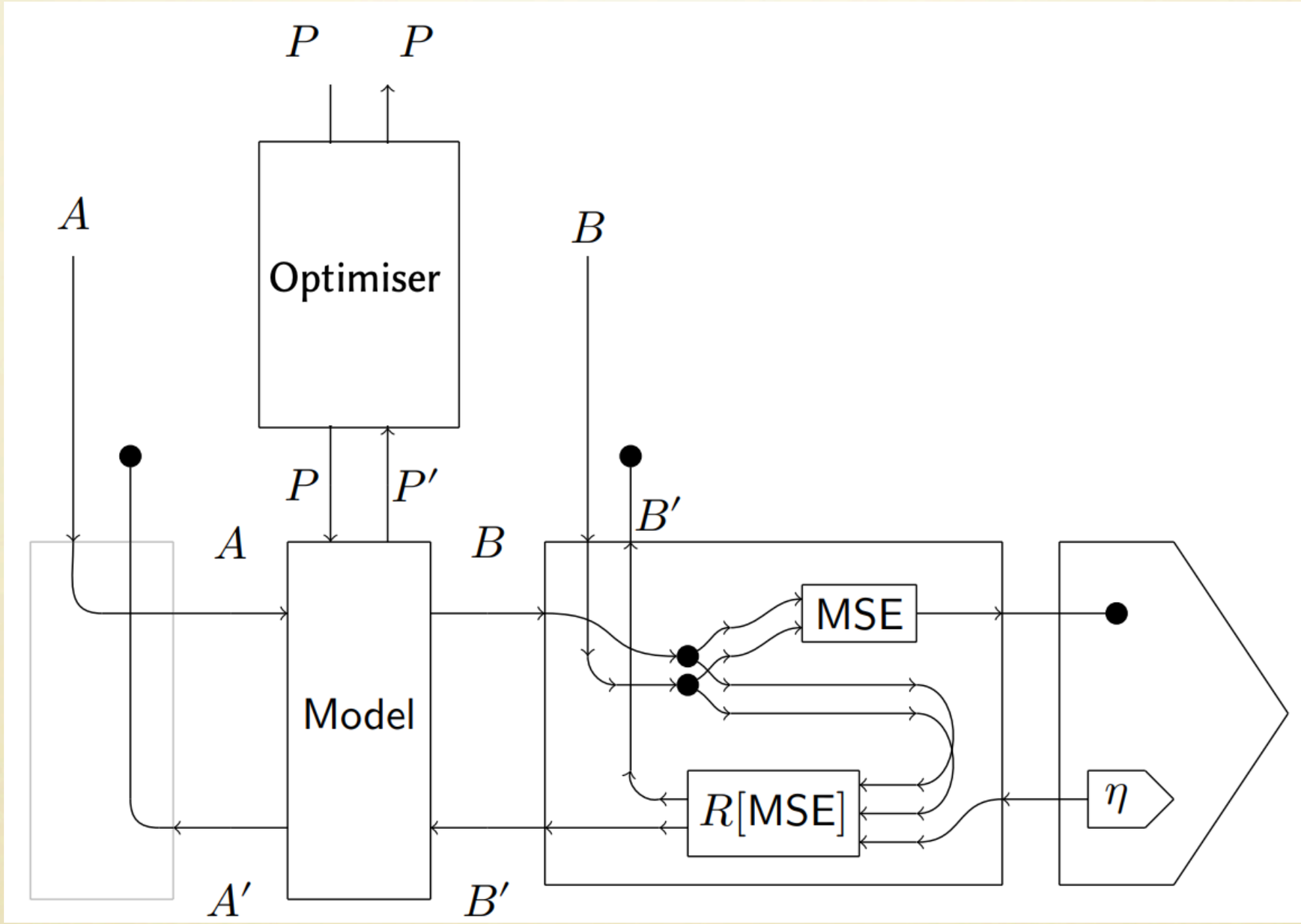


Where we are, so far

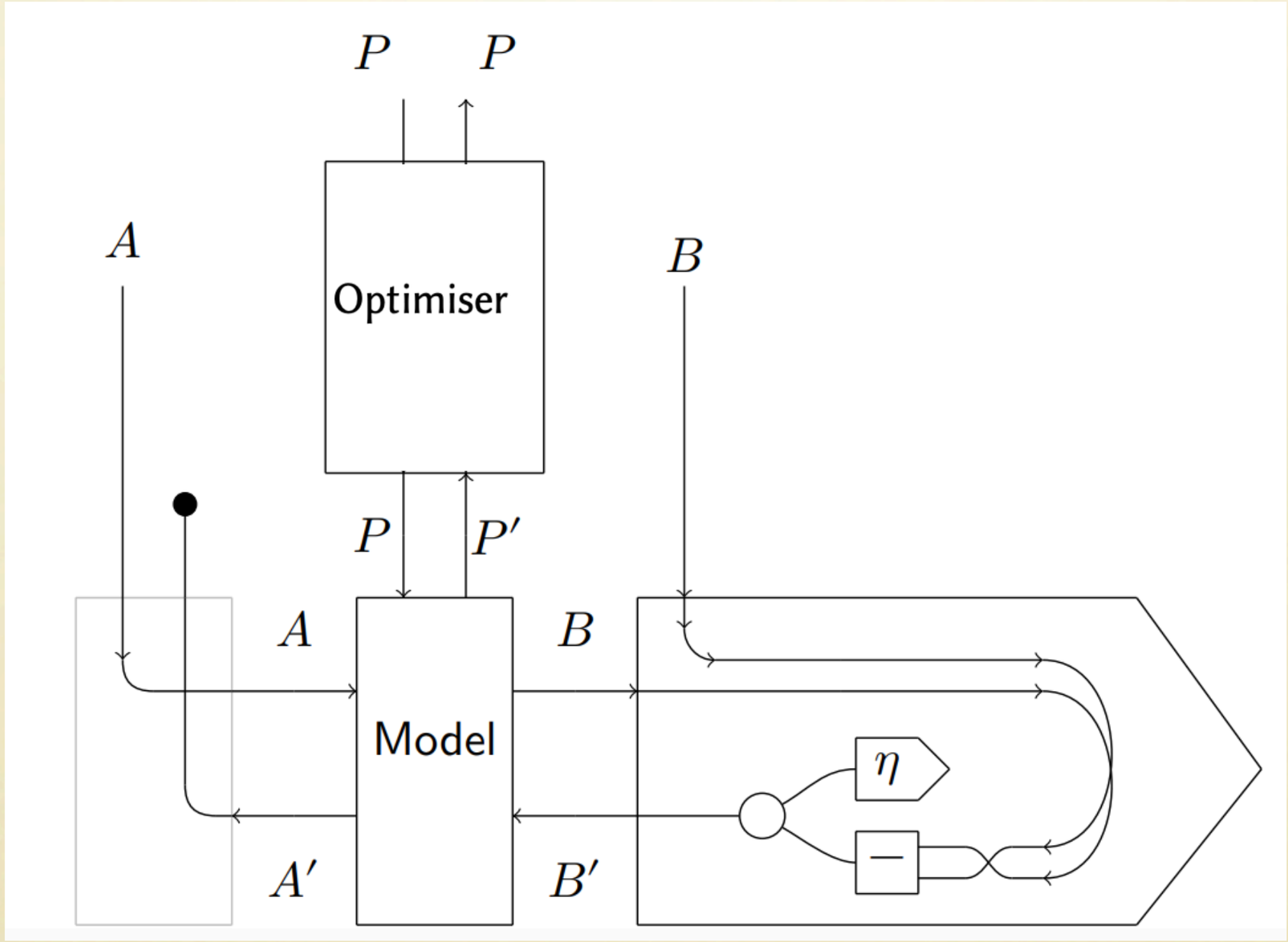




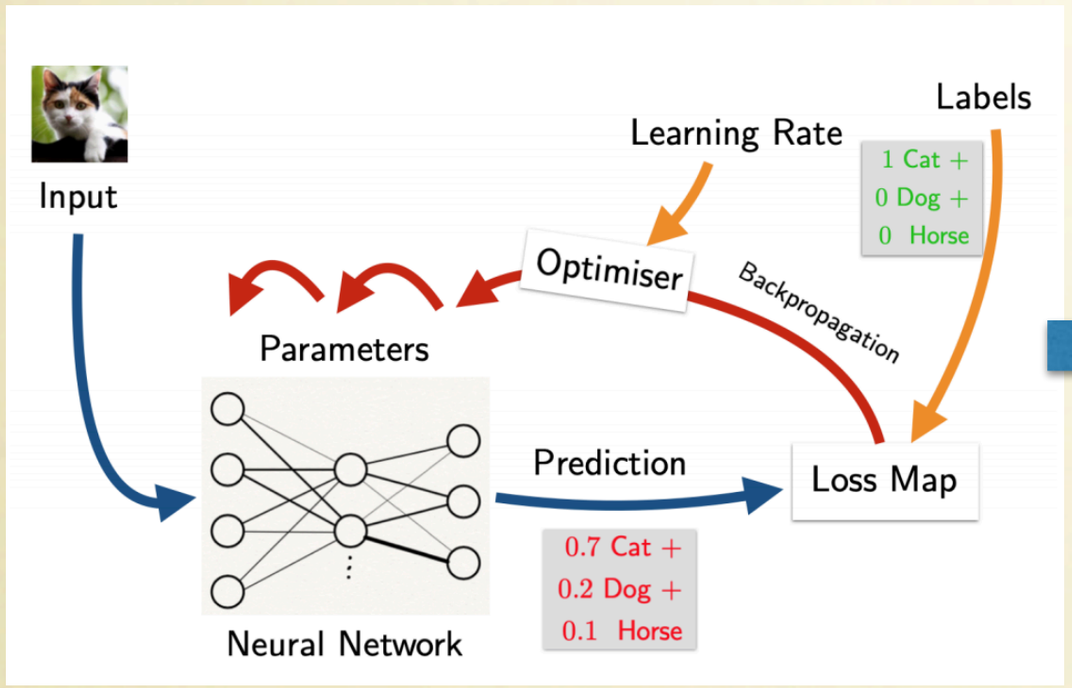
Loss map as parametric lens

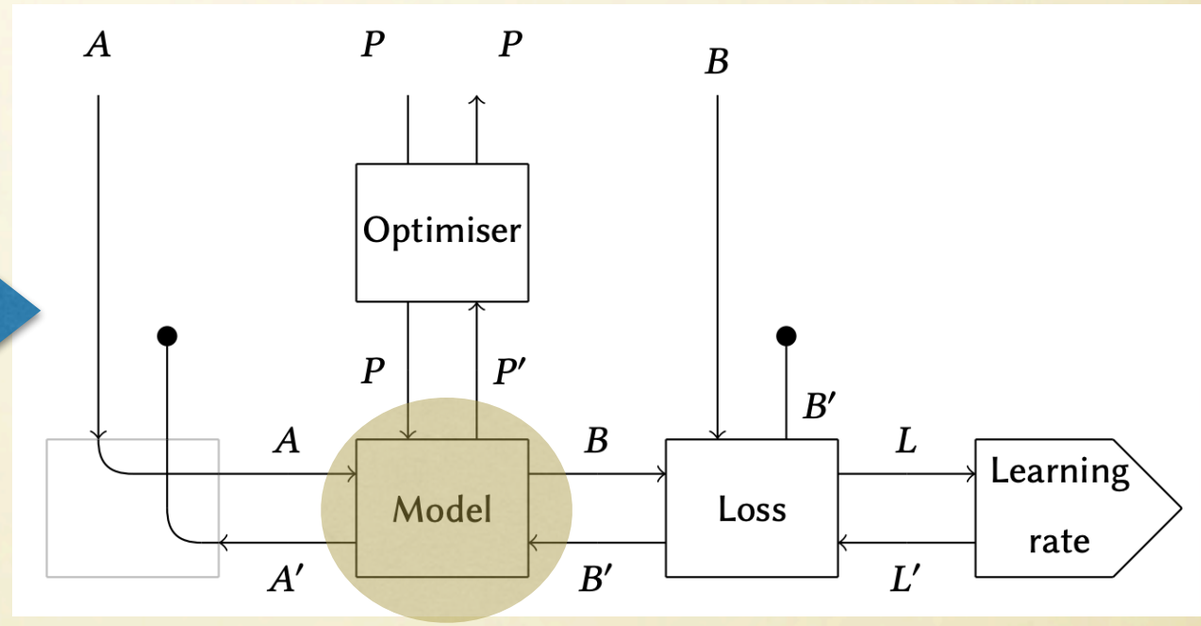


Loss map as parametric lens

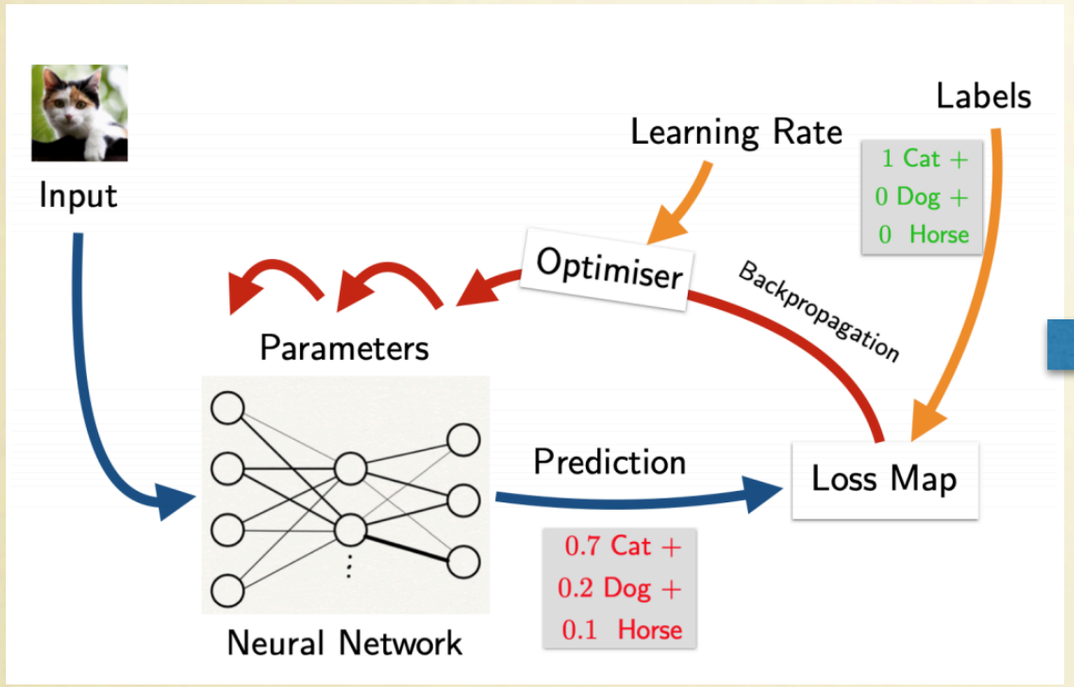


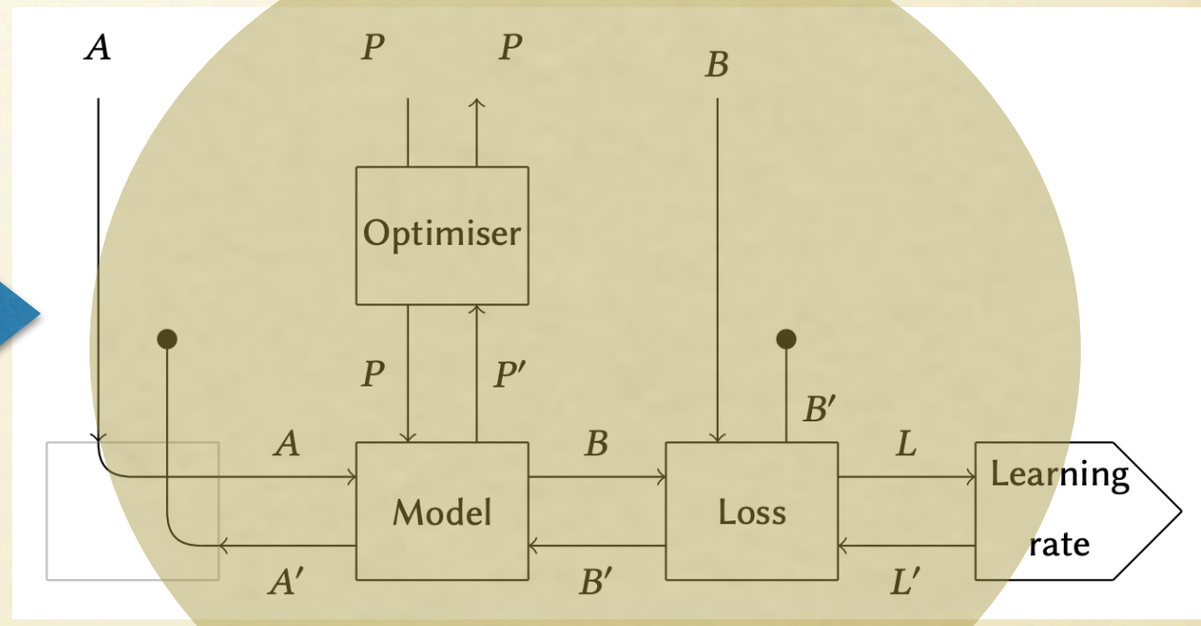
Where we are, so far





Where we are, so far





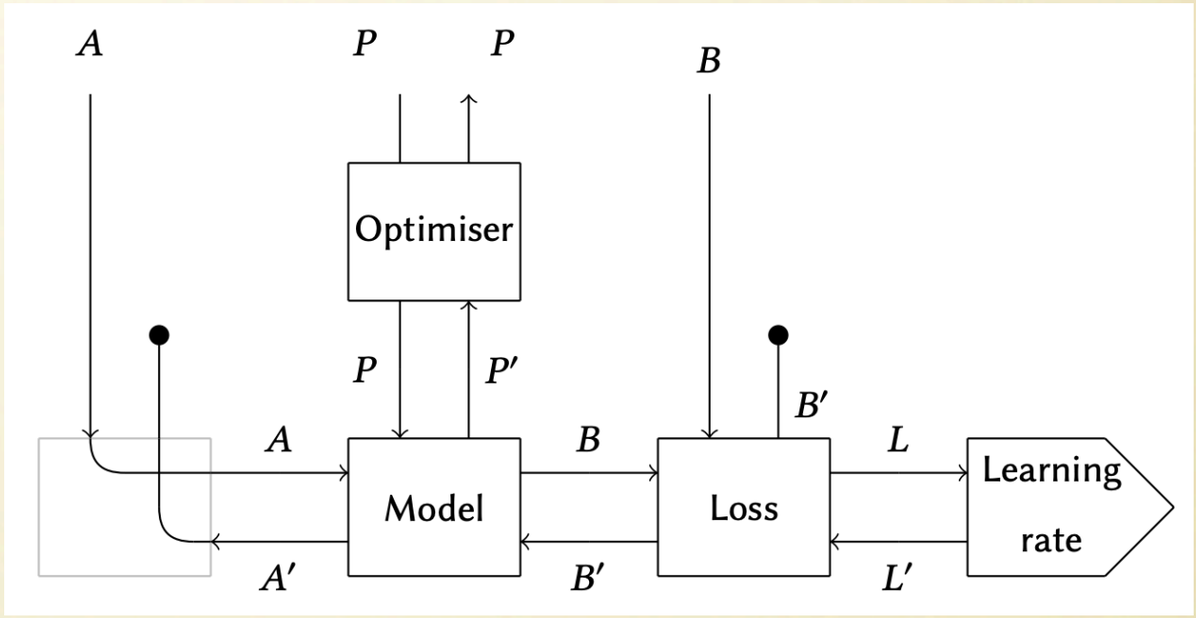
Putting it all together

A parametric lens of type

$$(1,1) \to (1,1)$$

A lens of type

$$(A \times P \times B, A' \times P' \times B') \to (1,1)$$



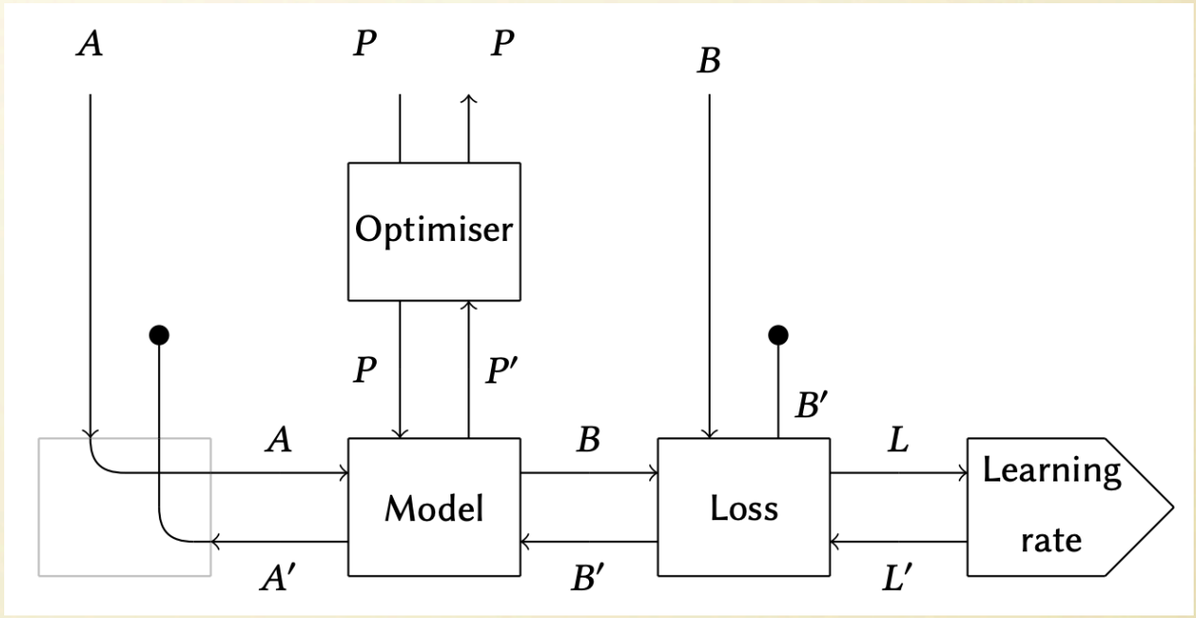
Putting it all together

A parametric lens of type

$$(1,1) \to (1,1)$$

A lens of type

$$(A \times P \times B, A' \times P' \times B') \to (1,1)$$

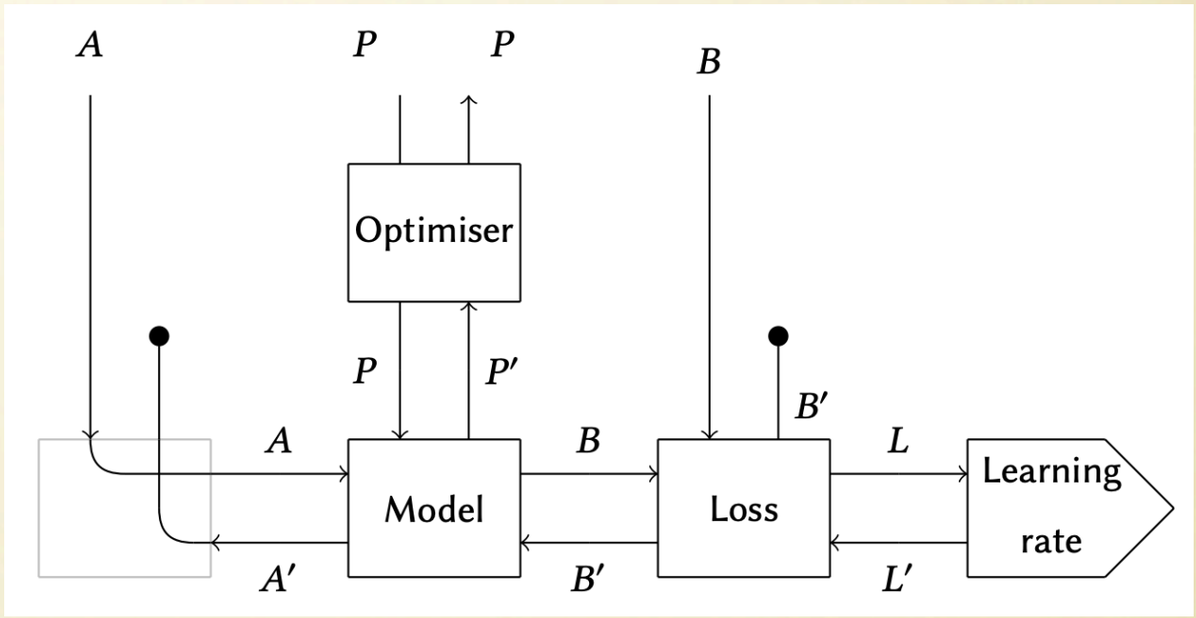


$$\textsf{put} \quad A \times P \times B \to A' \times P' \times B'$$

$$\textsf{put}(a, p, b_t) = (a', p', b'_t)$$

$$b_p = f(p, a) \qquad (b'_t, b'_p) = R[\textsf{loss}](b_t, b_p, \alpha(\textsf{loss}(b_t, b_p))) \qquad (p', a') = R[f](p, a, b'_p)$$

Learning as iterative process

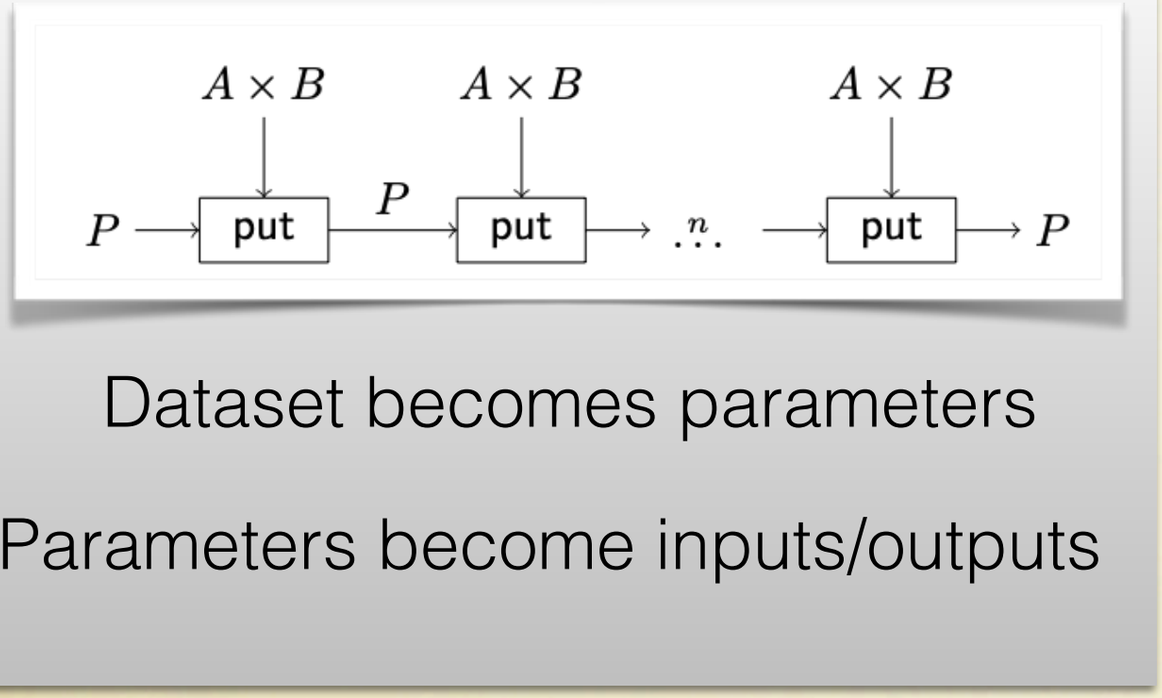


$$\textsf{put} \quad A \times P \times B \to A' \times P' \times B'$$

$$\textsf{put}(a, p, b_t) = (a', p', b'_t)$$

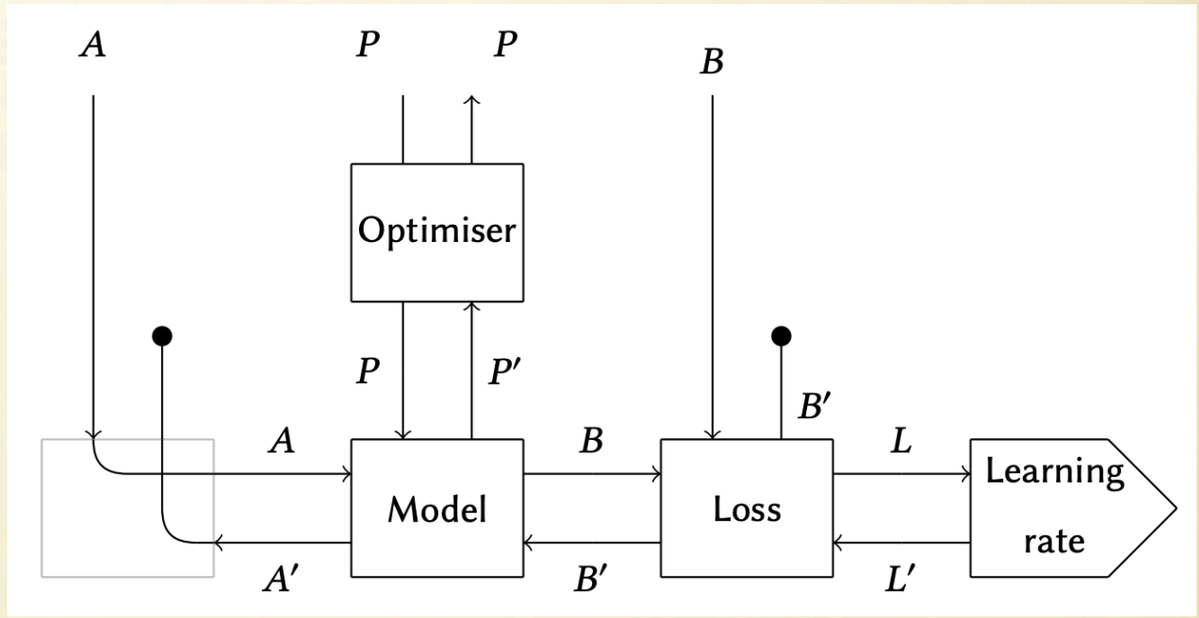
$$b_p = f(p, a) \qquad (b'_t, b'_p) = R[\textsf{loss}](b_t, b_p, \alpha(\textsf{loss}(b_t, b_p))) \qquad (p', a') = R[f](p, a, b'_p)$$

Learning as iterative process



Dataset becomes parameters

Parameters become inputs/outputs

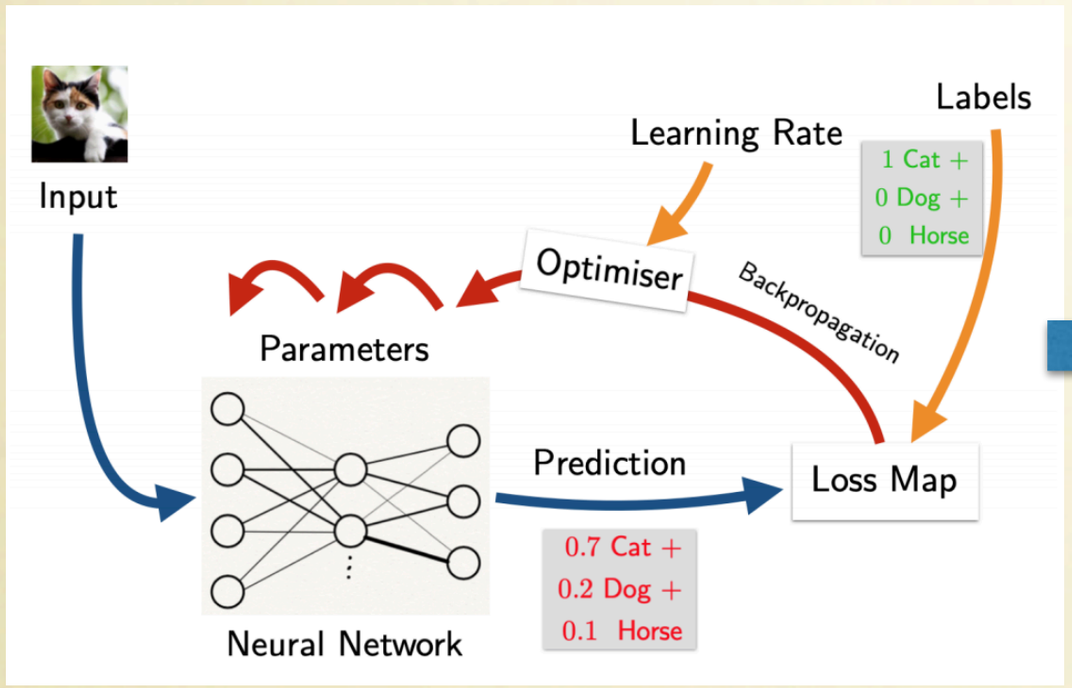


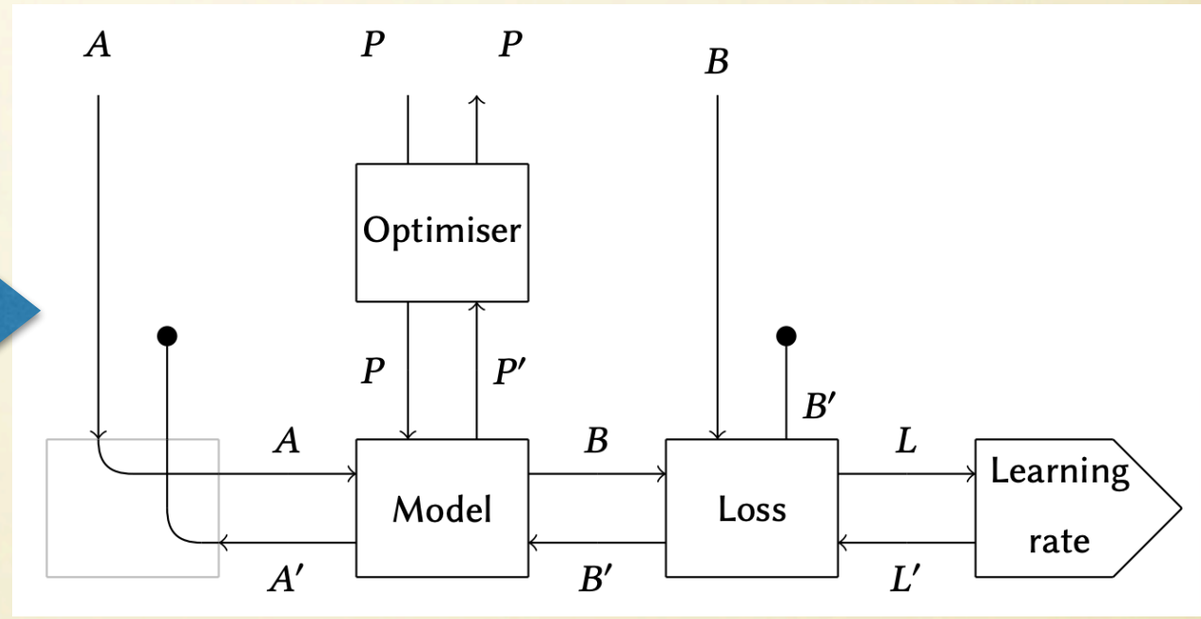
$$\textsf{put} \quad A \times P \times B \to A' \times P' \times B'$$

$$\textsf{put}(a, p, b_t) = (a', p', b'_t)$$

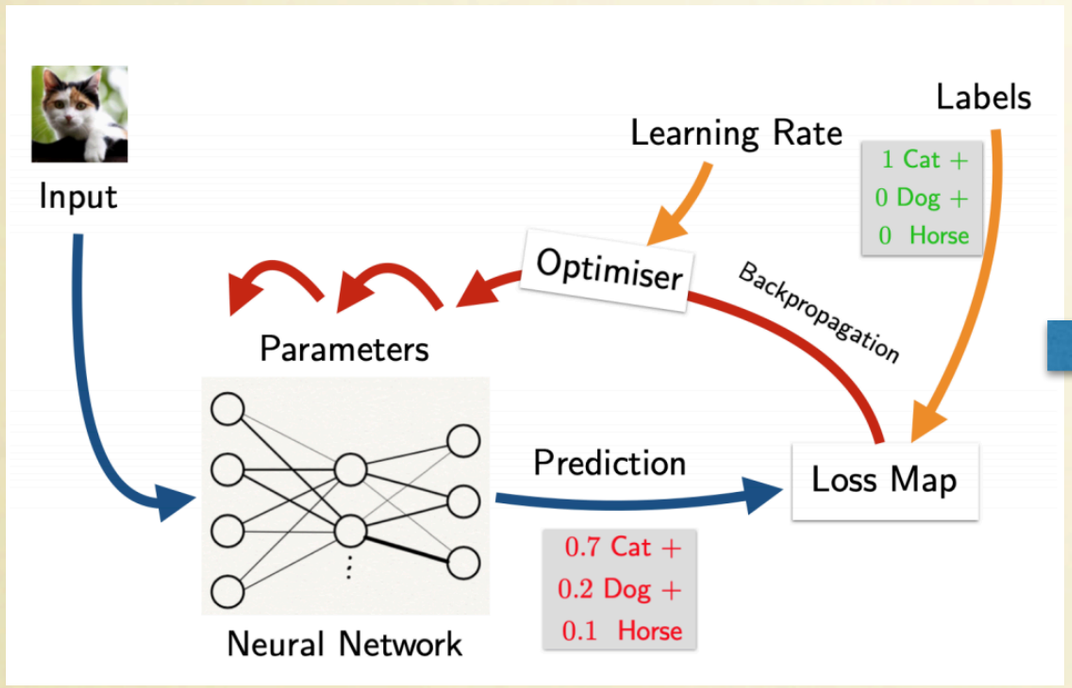
$$b_p = f(p, a) \qquad (b'_t, b'_p) = R[\textsf{loss}](b_t, b_p, \alpha(\textsf{loss}(b_t, b_p))) \qquad (p', a') = R[f](p, a, b'_p)$$

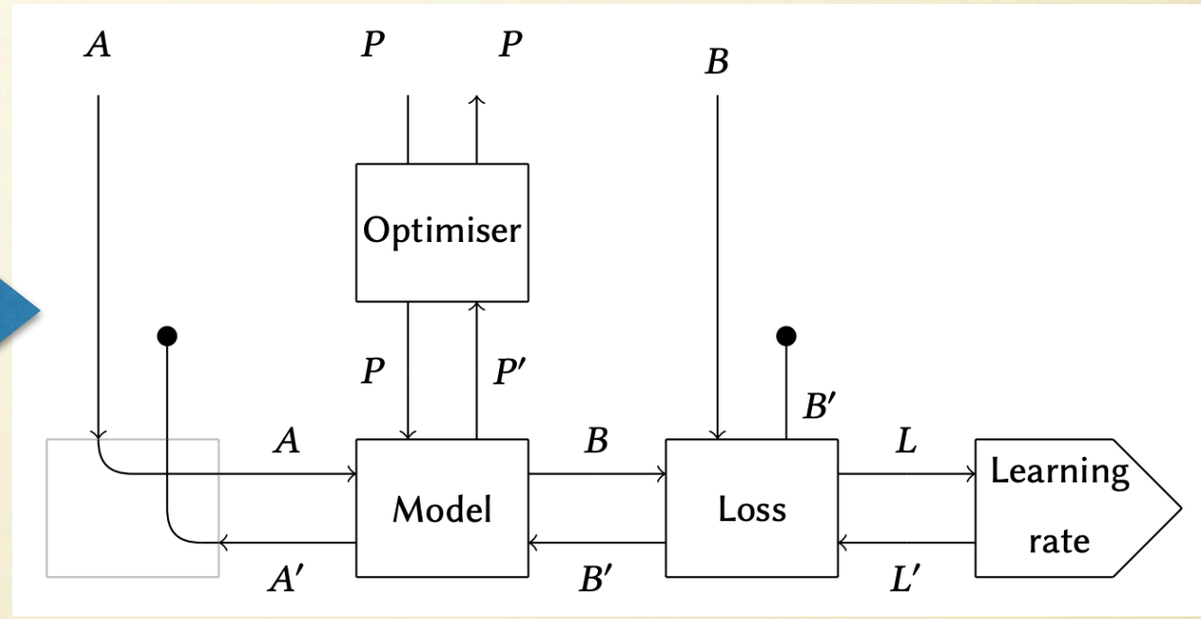
Where we are, so far





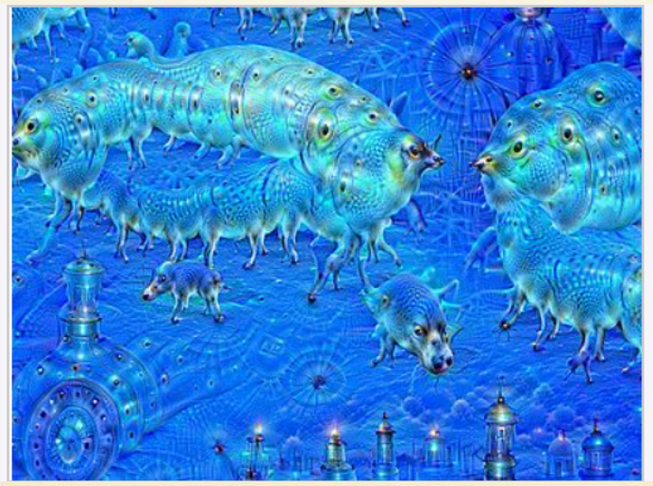
Where we are, so far





Variations I: Deep Dreaming

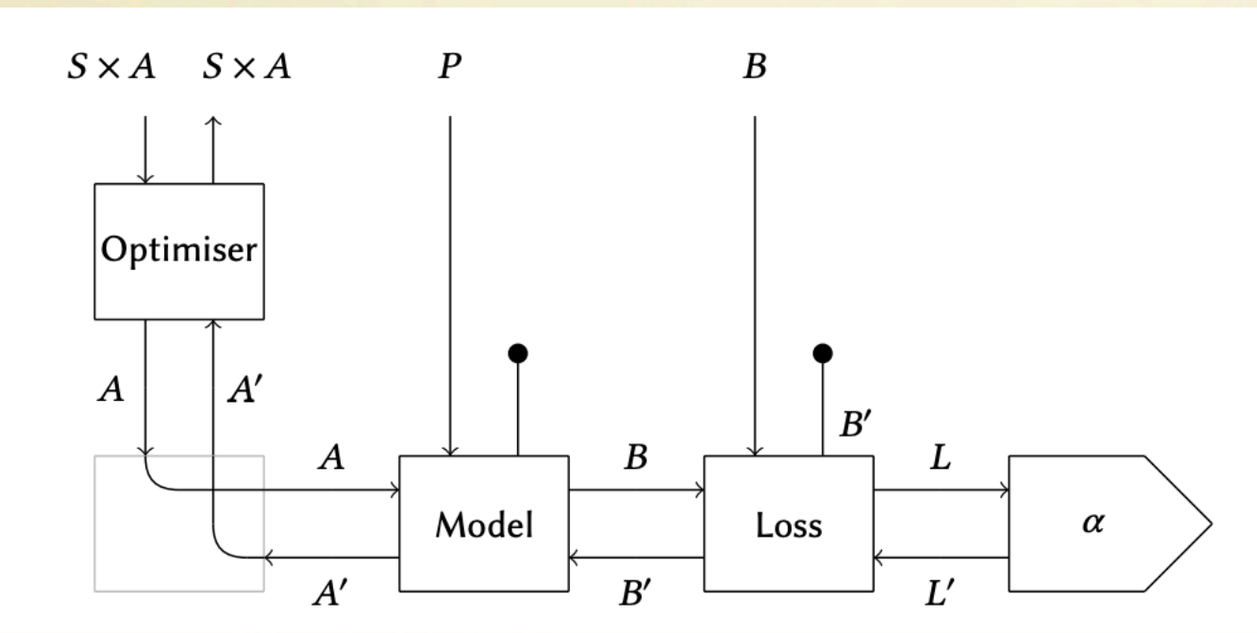
Input Image



``Over-processed'' Image

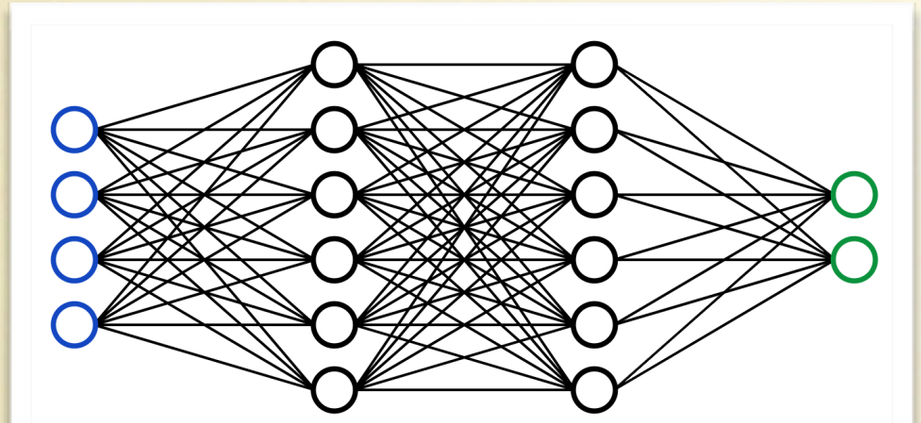
Variations I: Deep Dreaming

Parameter Learning

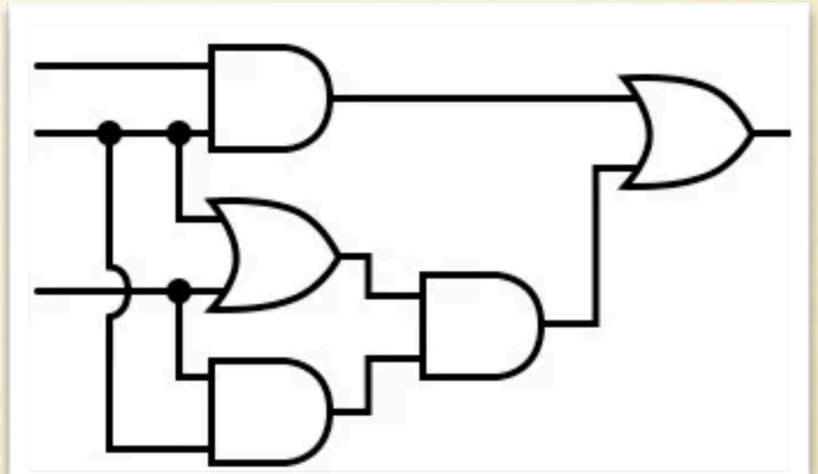


Deep dreaming

Variations II: Learning Boolean circuits



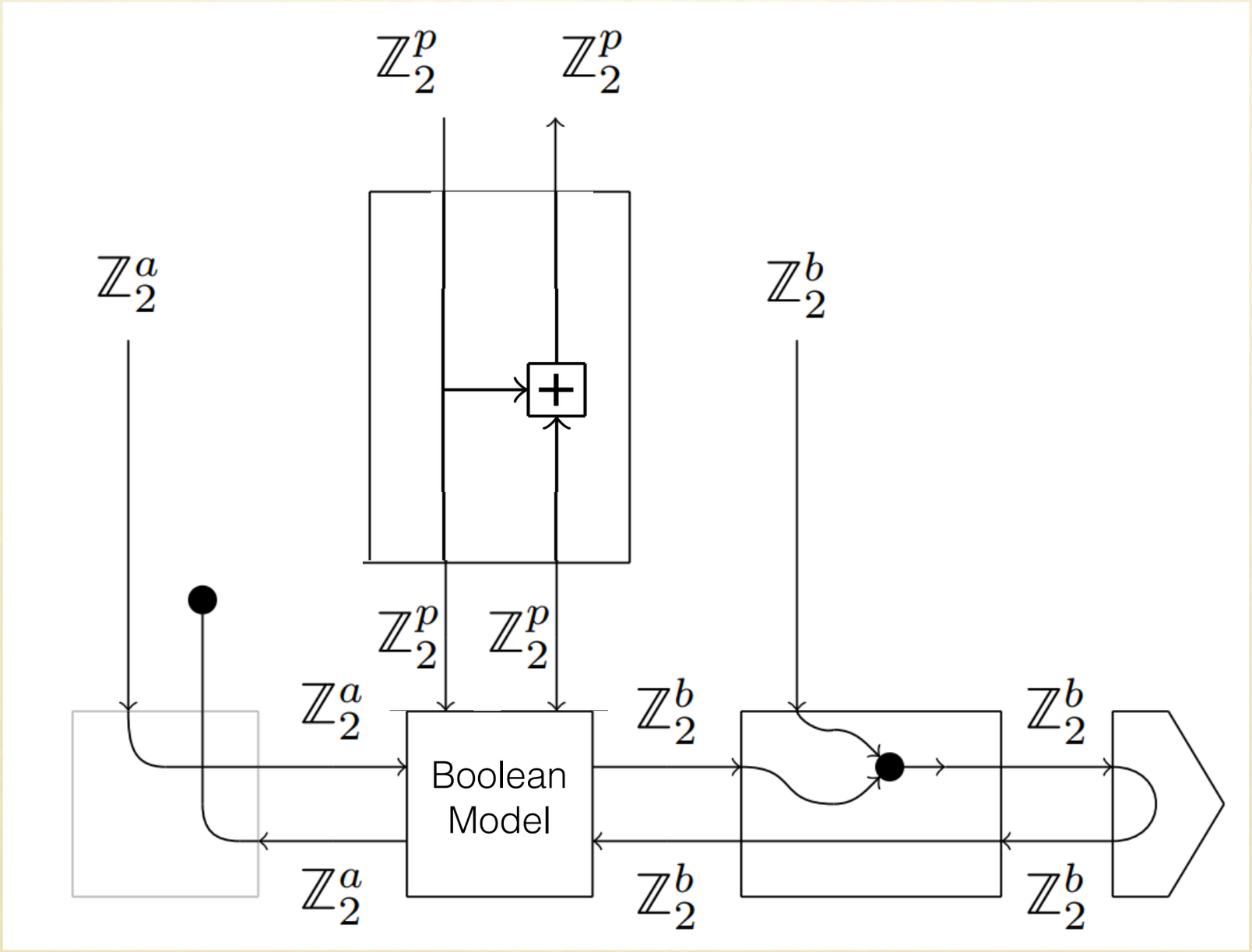
Neural networks



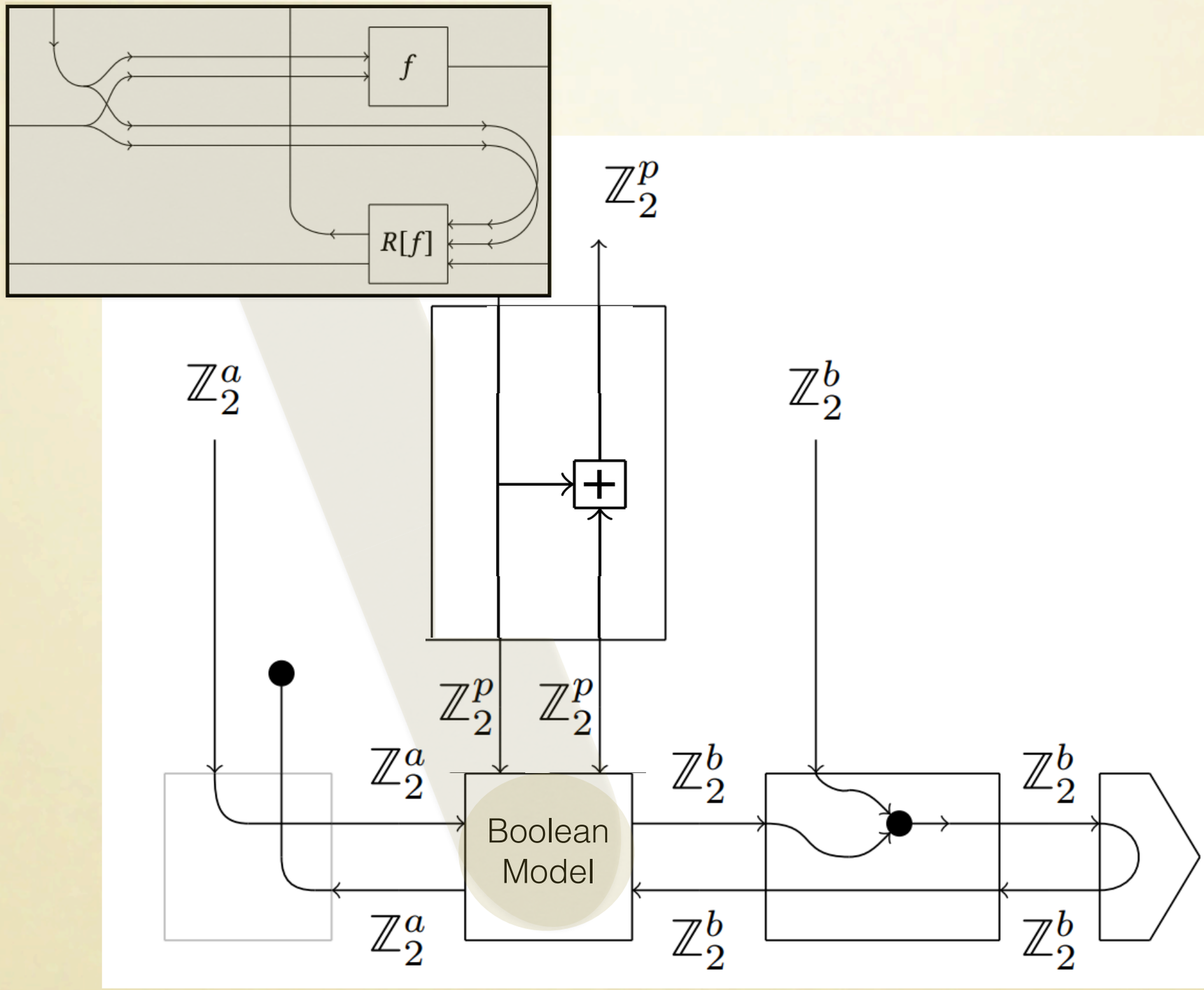
Boolean circuits

Instead of ***Smooth*** we work in ***Bool***

Variations II: Learning Boolean circuits

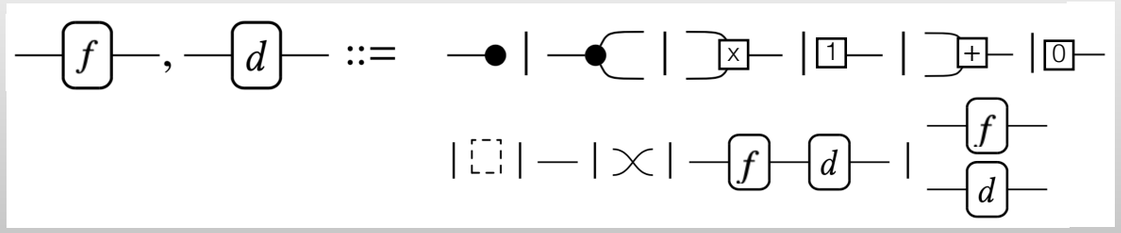


Variations II: Learning Boolean circuits



Variations II: Learning Boolean circuits

R[f] is defined **inductively** on circuit syntax:



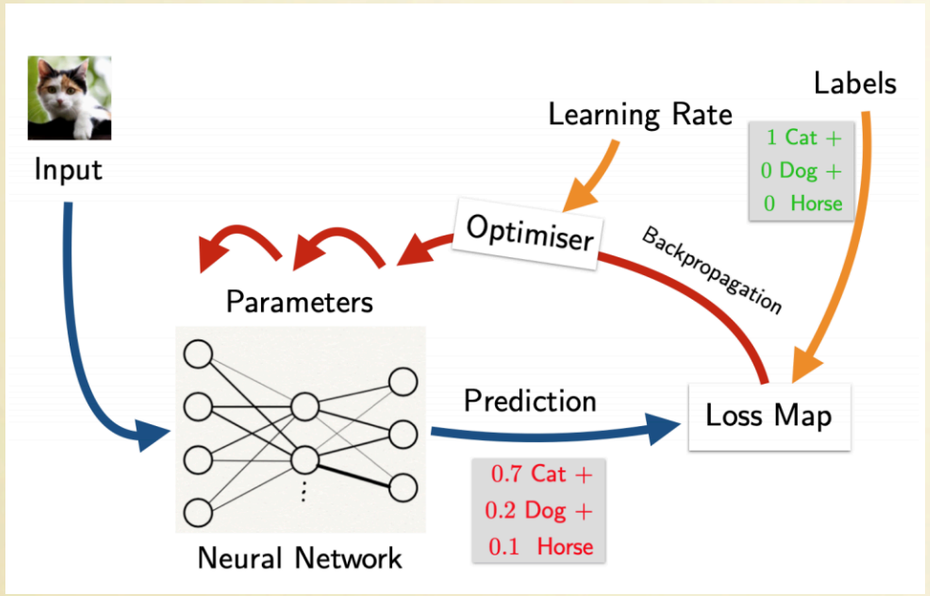
… and more

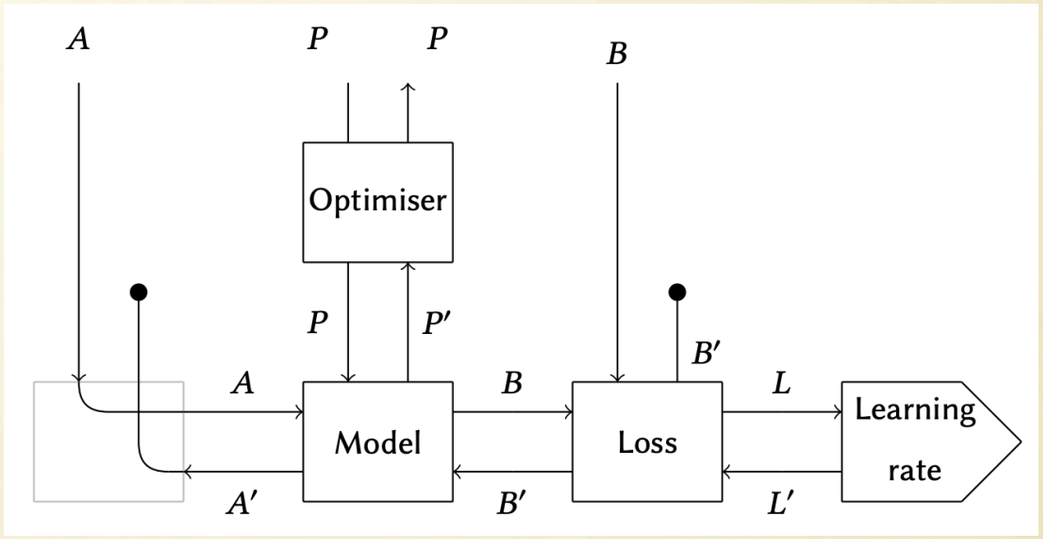
Generative Adversarial Networks

Convolutional Neural Networks

…

Discussion





A lens-theoretic semantics for gradient-based learning.

Discussion

Uniformity

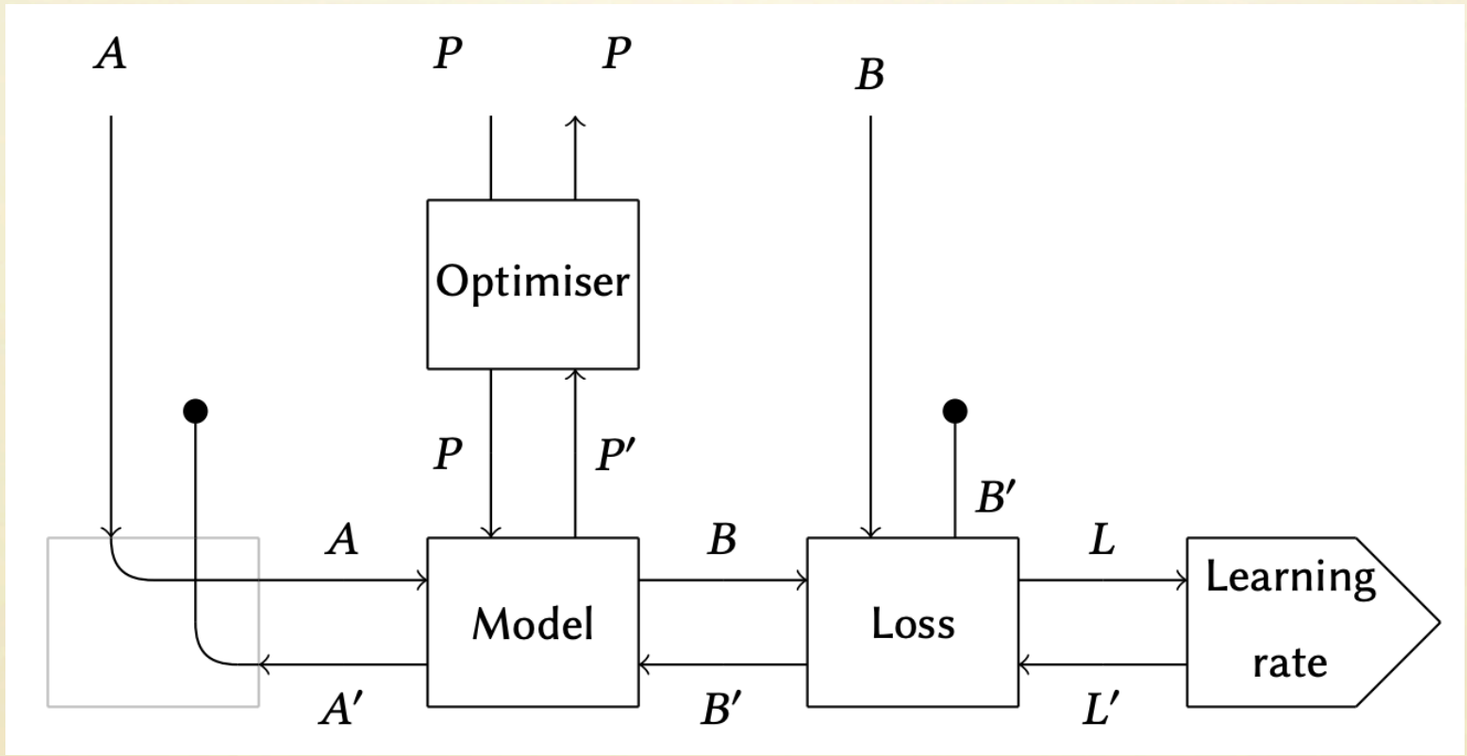
All components at work in the learning process are reduced to a single concept: parametric lenses.

Abstraction

A wide range of optimisers, models, loss maps, etc. are instances of the framework.

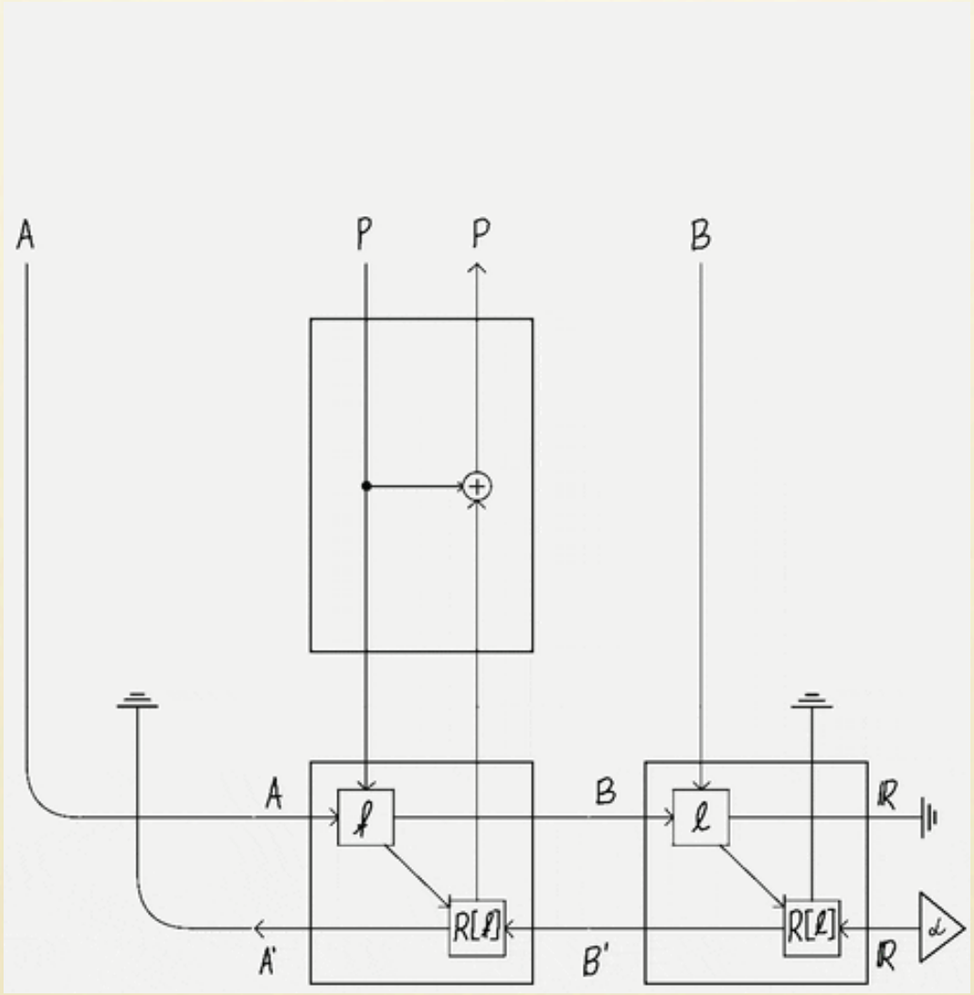
Discussion

Compositional reasoning



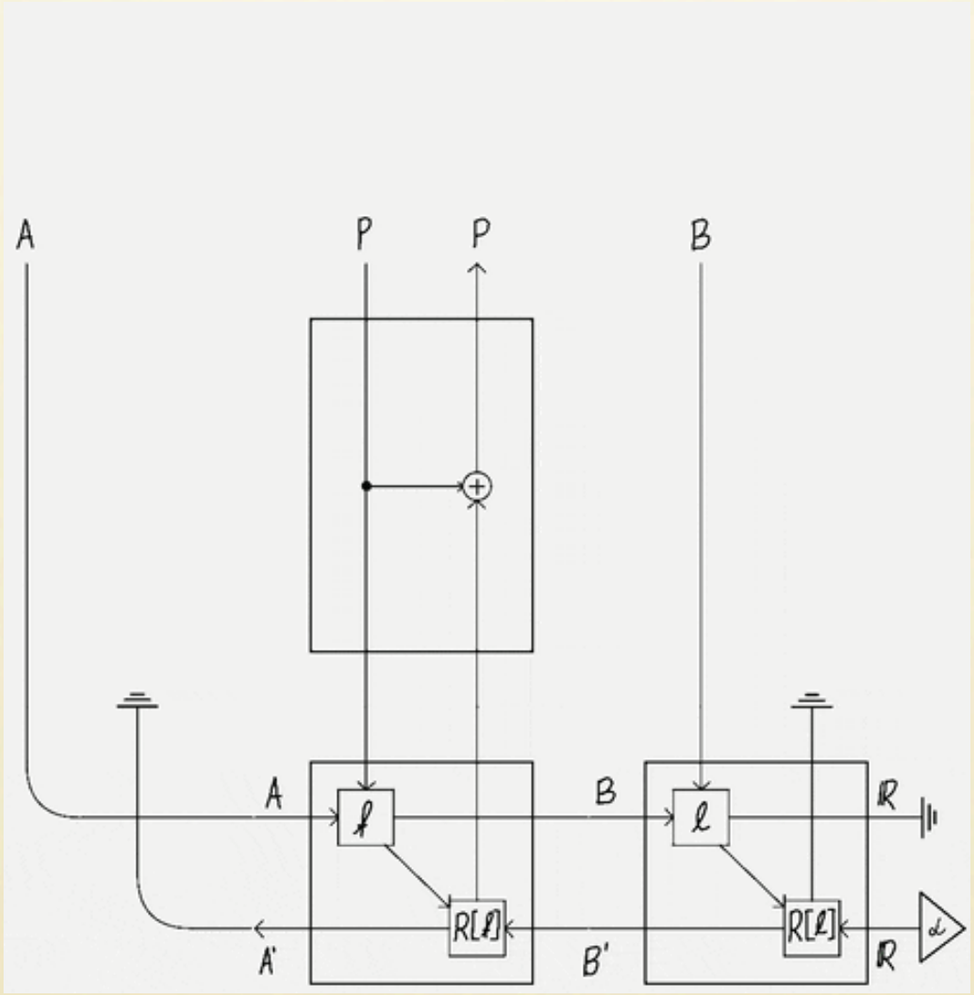
Discussion

Resource-sensitive Formalism



Discussion

Resource-sensitive Formalism



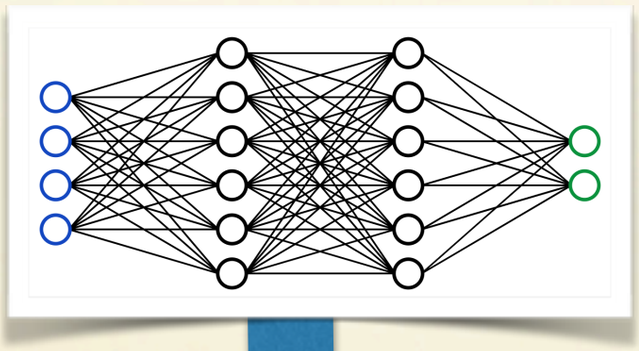
Going Forward

Going Forward

Learning over simpler models

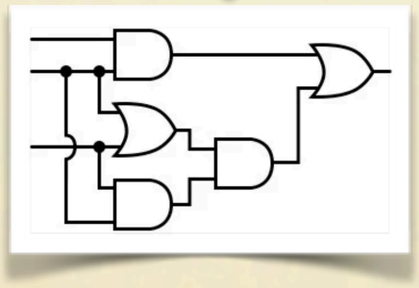
Performance

Accuracy



Performance

Accuracy



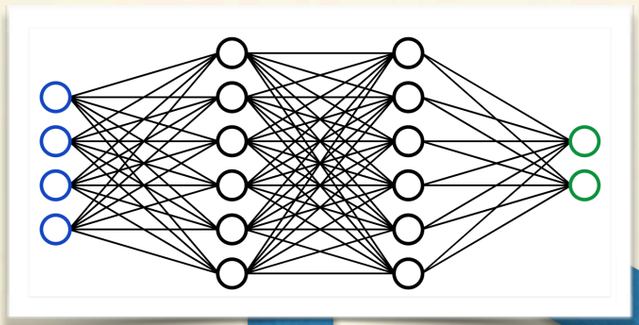
Boolean circuits

Going Forward

Learning over simpler models

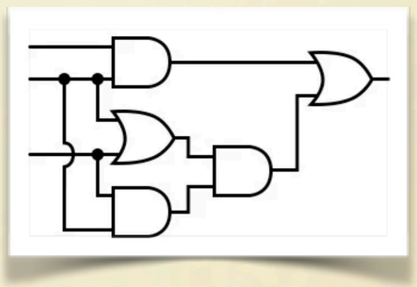
Performance

Accuracy



Performance

Accuracy

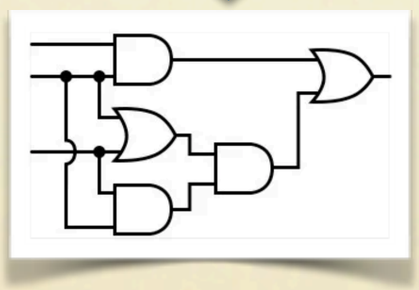


Polynomial circuits

cf. saturation arithmetic, zero-knowledge proofs

Performance

Accuracy



Boolean circuits

Going Forward

Programming with lenses?

https://github.com/statusfailed/numeric-optics-python/

http://catgrad.com/p/reverse-derivative-ascent

Going Forward

Programming with lenses?

https://github.com/statusfailed/numeric-optics-python/

http://catgrad.com/p/reverse-derivative-ascent

```
def dense(a, b, activation):
  return linear(a, b) >> bias(b) >> activation
```

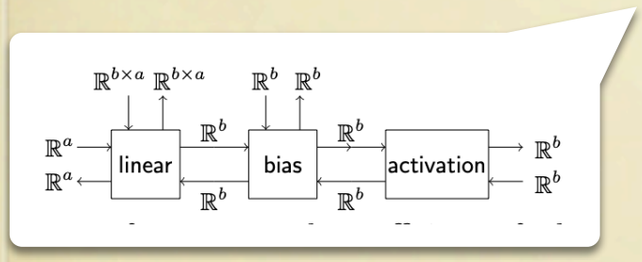
Going Forward

Programming with lenses?

https://github.com/statusfailed/numeric-optics-python/

http://catgrad.com/p/reverse-derivative-ascent

```
def dense(a, b, activation):
  return linear(a, b) >> bias(b) >> activation
```



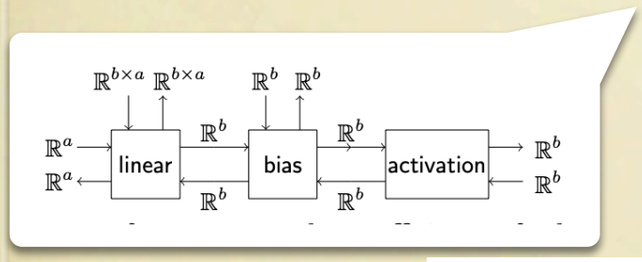
Going Forward

Programming with lenses?

https://github.com/statusfailed/numeric-optics-python/

http://catgrad.com/p/reverse-derivative-ascent

```
def dense(a, b, activation):
  return linear(a, b) >> bias(b) >> activation
```




```
apply_update(basic_update, dense) >> loss >> learning_rate(ϵ)
```

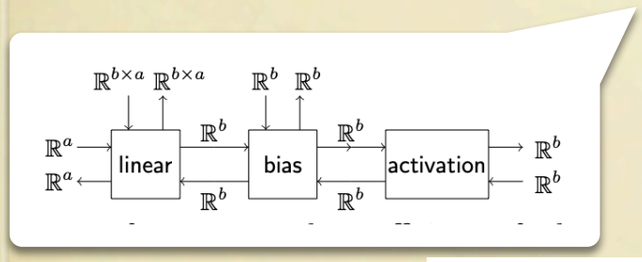
Going Forward

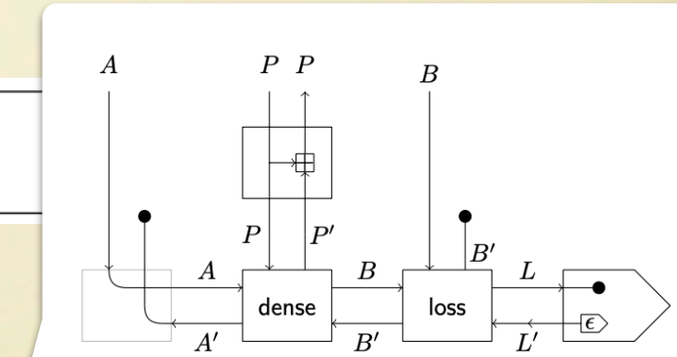
Programming with lenses?

https://github.com/statusfailed/numeric-optics-python/

http://catgrad.com/p/reverse-derivative-ascent

```
def dense(a, b, activation):
  return linear(a, b) >> bias(b) >> activation
```






```
apply_update(basic_update, dense) >> loss >> learning_rate(ϵ)
```

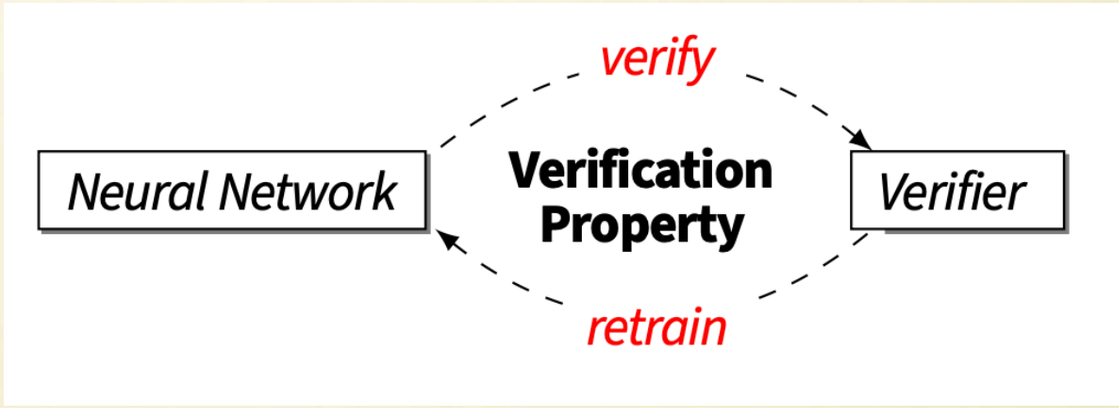
Going Forward

Formal verification with lenses

Going Forward

Formal verification with lenses

Continuous verification:
Loss function encapsulates the property to verify

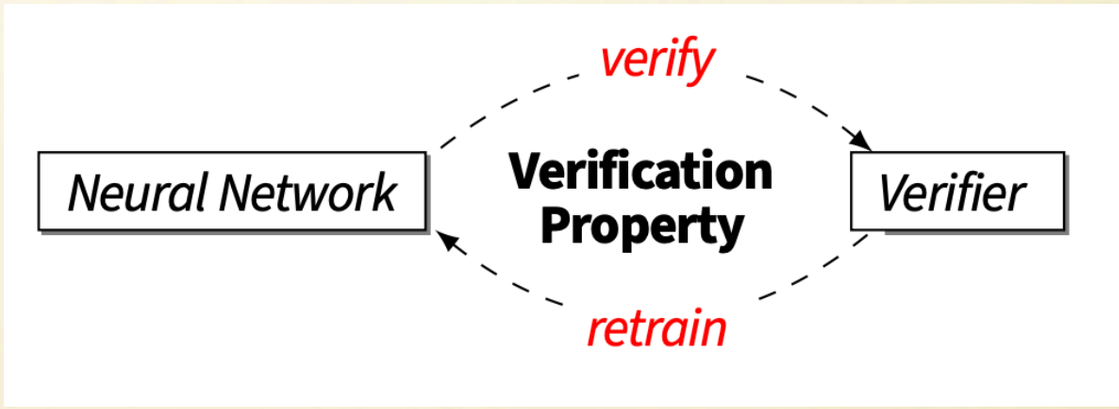


Going Forward

Formal verification with lenses

Continuous verification:
Loss function encapsulates the property to verify

Specification language:
Differentiable logics

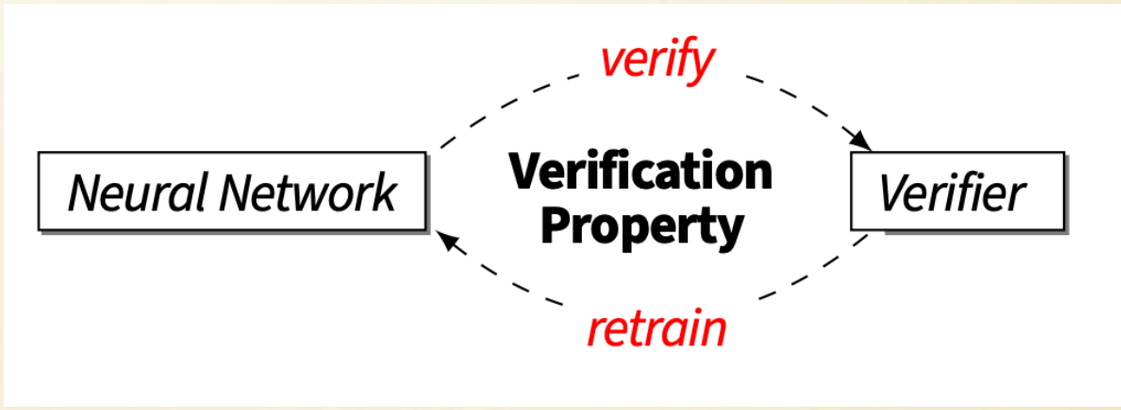


Going Forward

Formal verification with lenses

Continuous verification:
Loss function encapsulates the property to verify

Specification language:
Differentiable logics



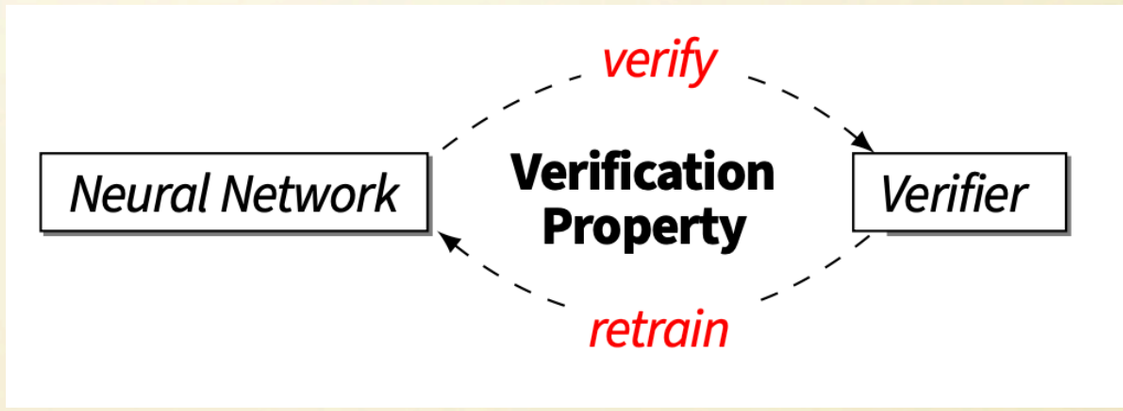
Lens-based semantics
of the continuous verification cycle

Going Forward

Formal verification with lenses

Continuous verification:
Loss function encapsulates the property to verify

Specification language:
Differentiable logics



Lens-based semantics
of the continuous verification cycle

Lens-based specification language for structural properties to be verified

References

Implementation

https://github.com/statusfailed/numeric-optics-python/

http://catgrad.com/p/reverse-derivative-ascent

https://yarrow.id/ (Ethereum Foundation Grant)

(Partial) bibliography

G. Cruttwell, B. Gavranovic, N. Ghani, P. Wilson, F. Zanasi - *Categorical Foundations of Gradient-Based Learning*, ESOP 2022

P. Wilson, F. Zanasi - *Reverse Derivative Ascent: a Categorical Approach to Learning Boolean Circuits*, ACT 2020

P. Wilson, F. Zanasi - *Categories of Differentiable Polynomial Circuits for Machine Learning*, ICGT 2022