

# The surprising effectiveness of lenses in machine learning

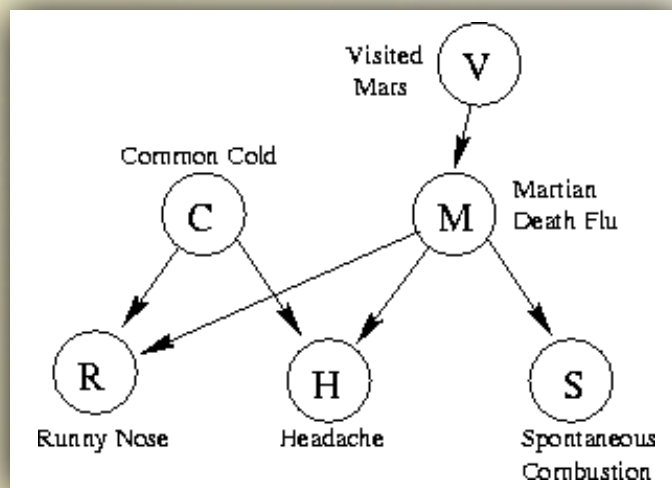
Fabio Zanasi



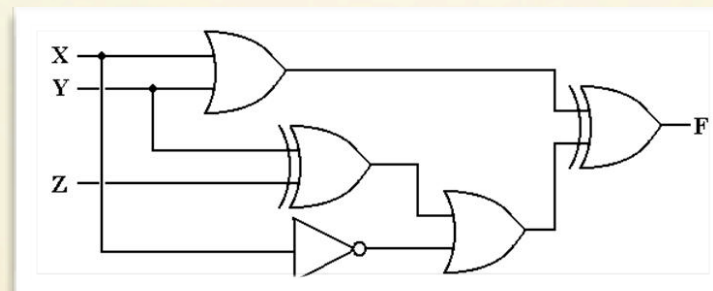
Based on joint work with  
Geoff Cruttwell, Bruno Gavranovic, Neil Ghani, Paul Wilson

# Background: Categorical Semantics of Component-based systems

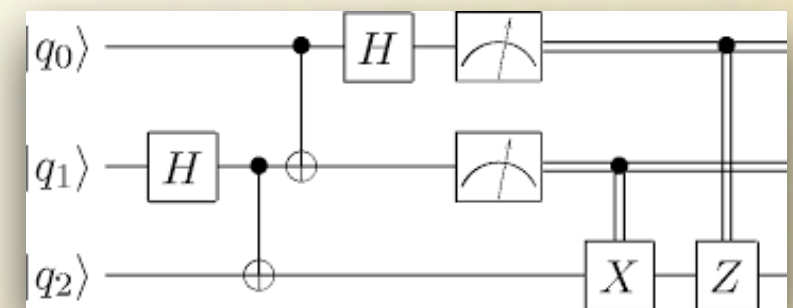
## Bayesian networks



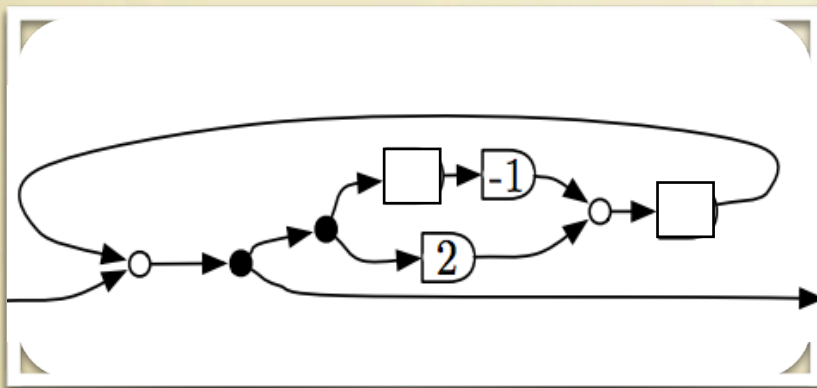
## Digital Circuits



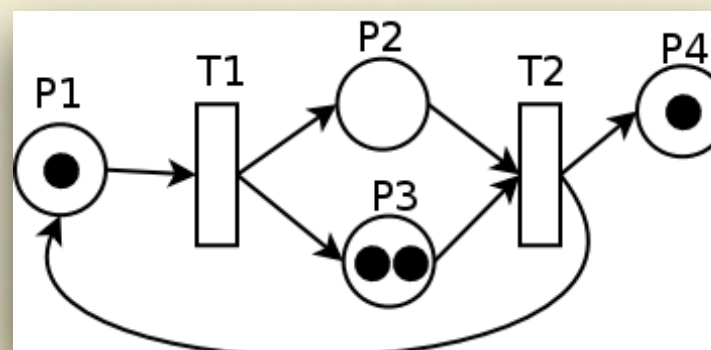
## Quantum circuits



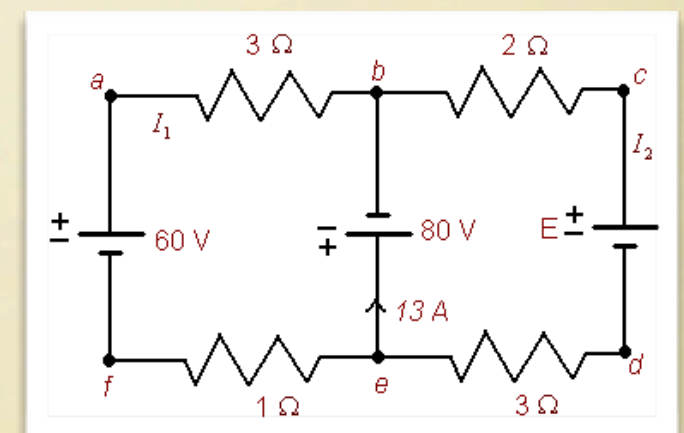
## Signal flow graphs



## Petri nets



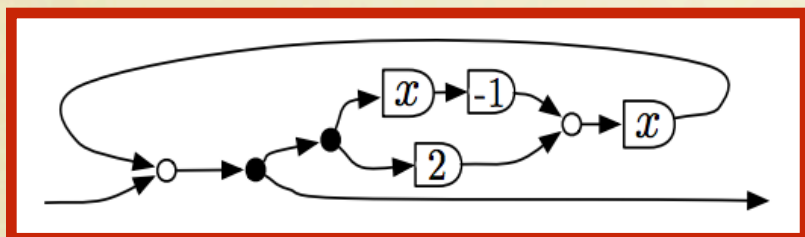
## Electrical Circuits



# Key Ideas

Graphical notation becomes **formal syntax**  
and it is given **compositional** semantics

Example:  
Signal Flow Graphs

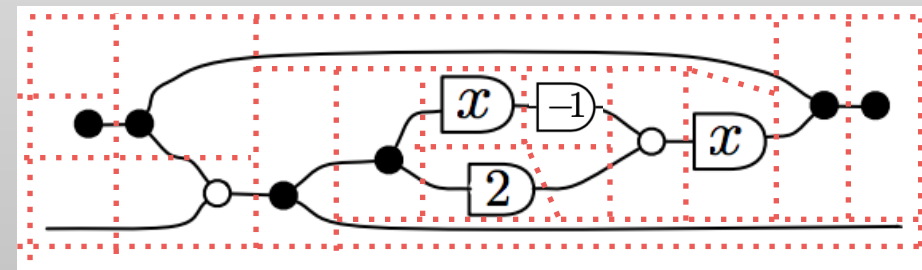
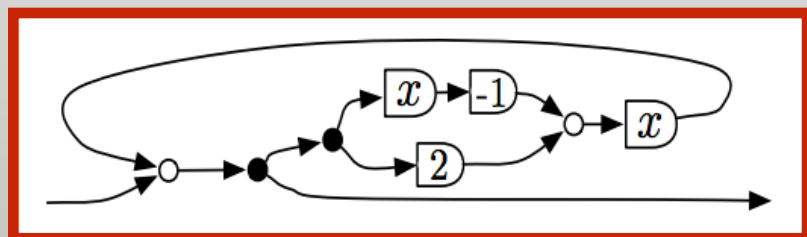
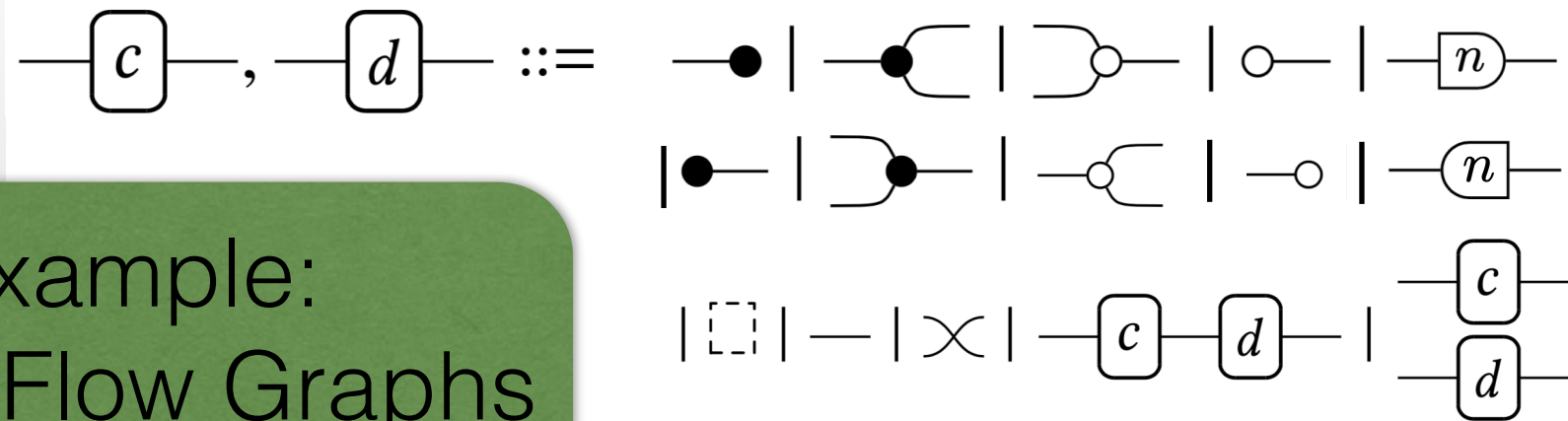




# Key Ideas

Graphical notation becomes **formal syntax**  
and it is given **compositional** semantics

Example:  
Signal Flow Graphs

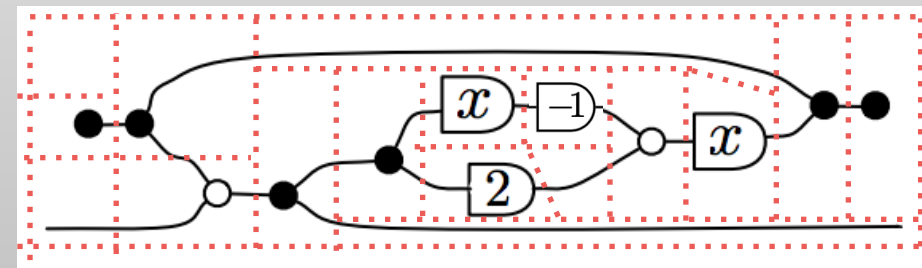
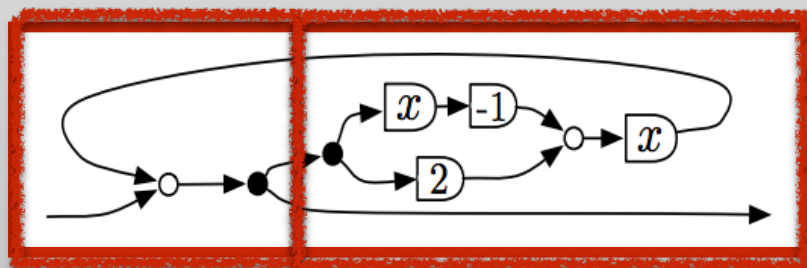
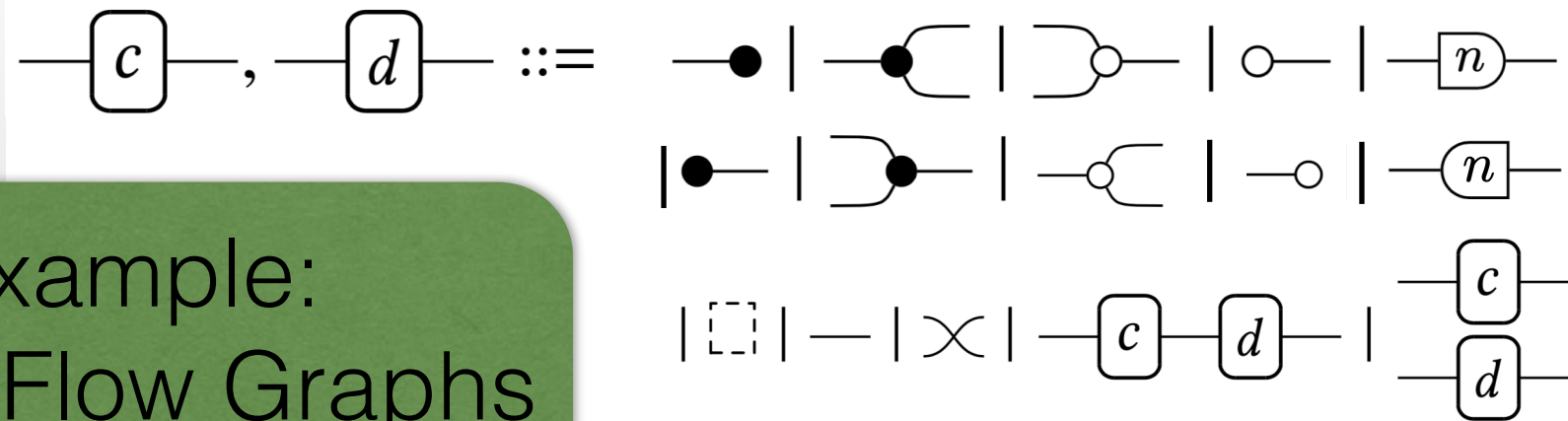




# Key Ideas

Graphical notation becomes **formal syntax**  
and it is given **compositional** semantics

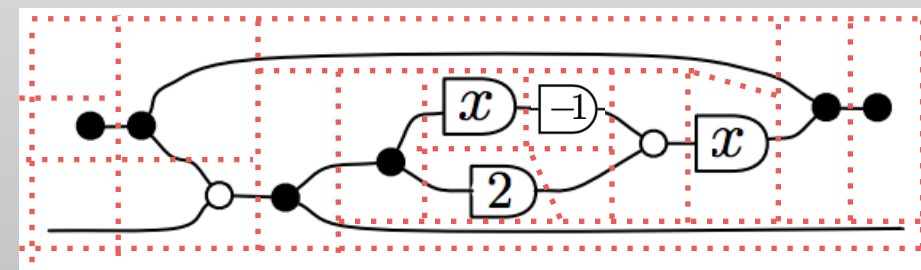
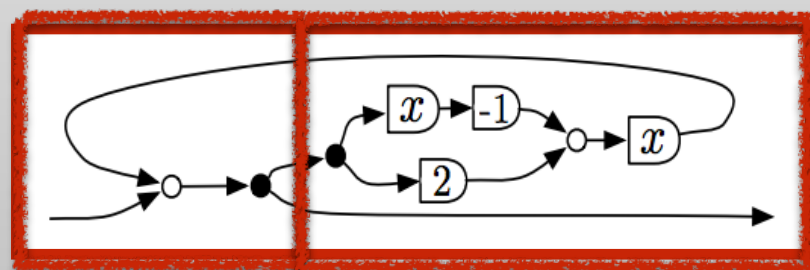
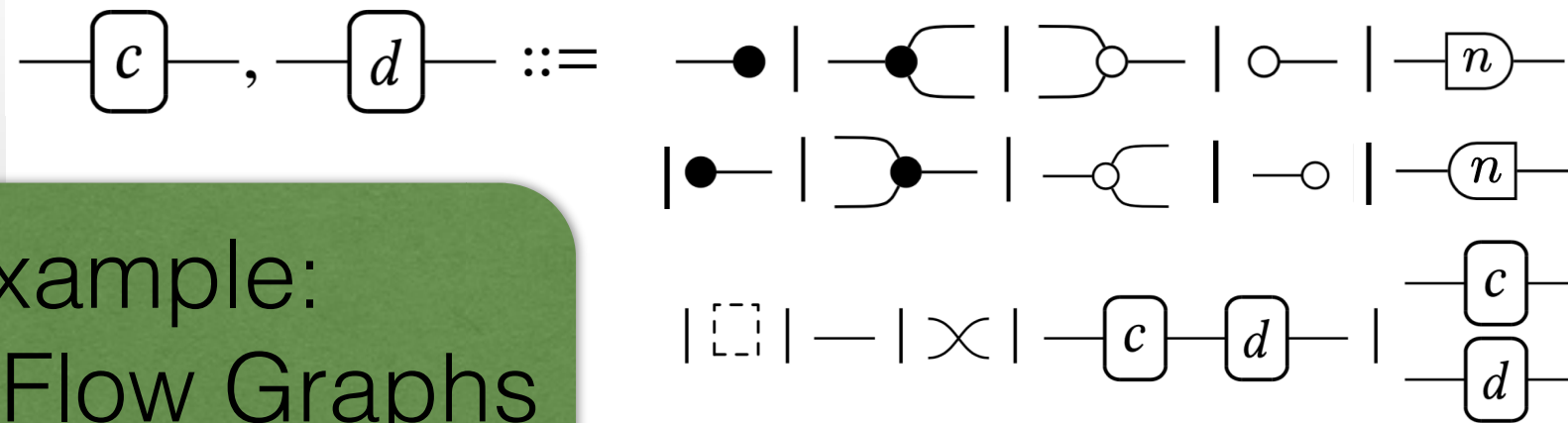
Example:  
Signal Flow Graphs



# Key Ideas

Graphical notation becomes **formal syntax**  
and it is given **compositional** semantics

Example:  
Signal Flow Graphs



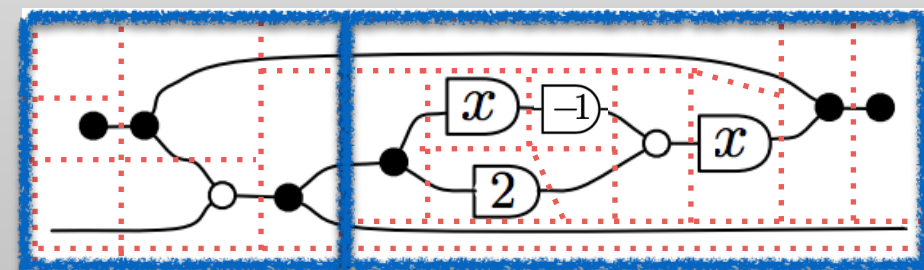
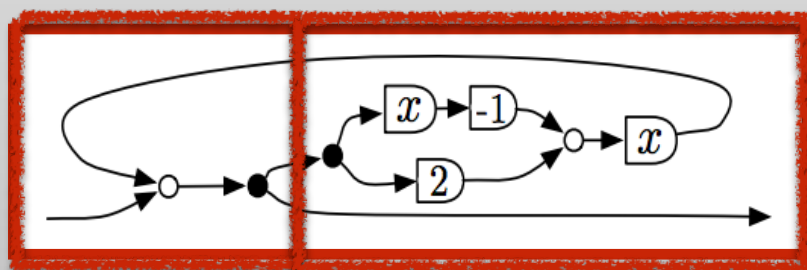
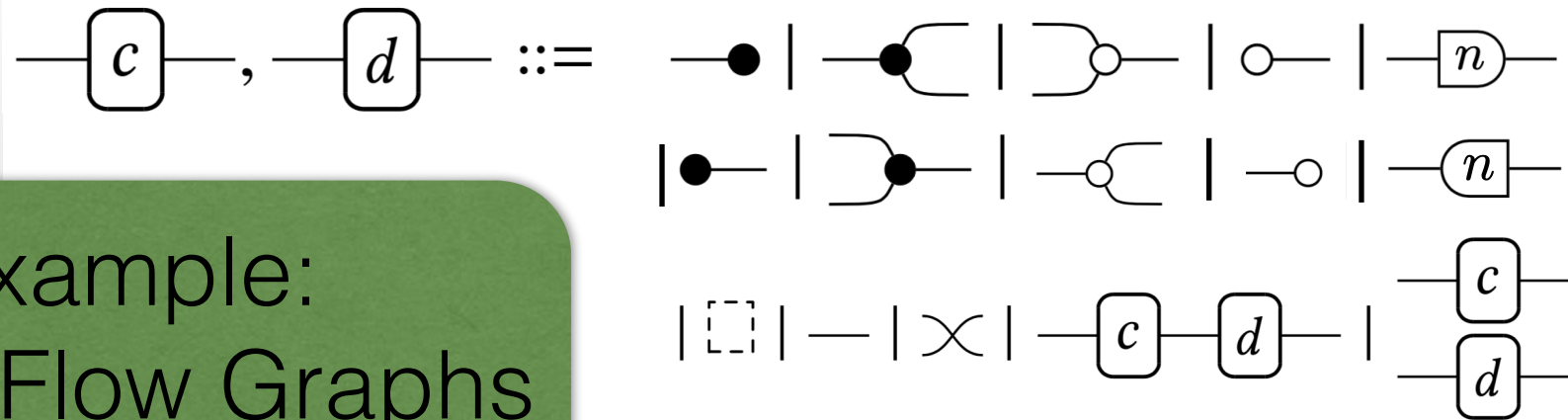
↑  
↓  
?

↑  
↓  
?

# Key Ideas

Graphical notation becomes **formal syntax**  
and it is given **compositional** semantics

Example:  
Signal Flow Graphs



↑  
↓  
?

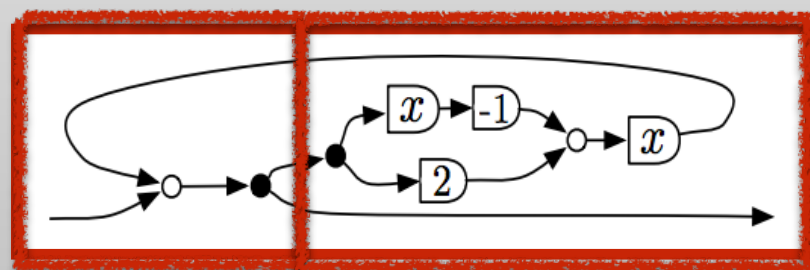
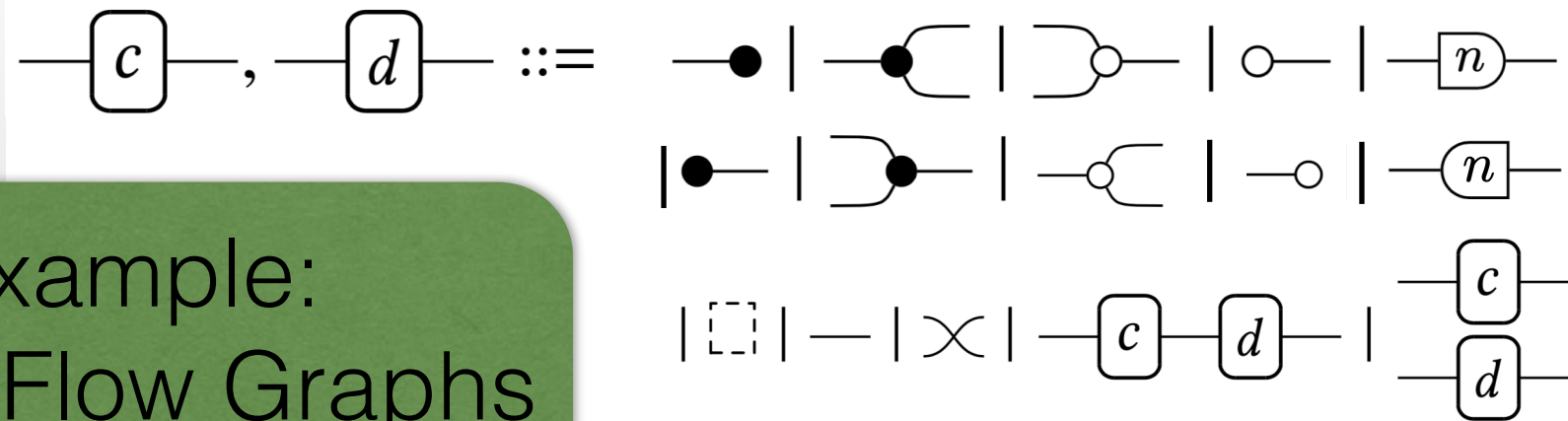
↑  
↓  
?



# Key Ideas

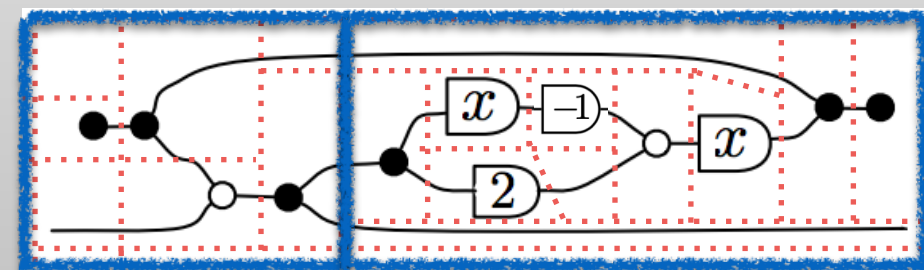
Graphical notation becomes **formal syntax**  
and it is given **compositional** semantics

Example:  
Signal Flow Graphs



↕  
?

↕  
?



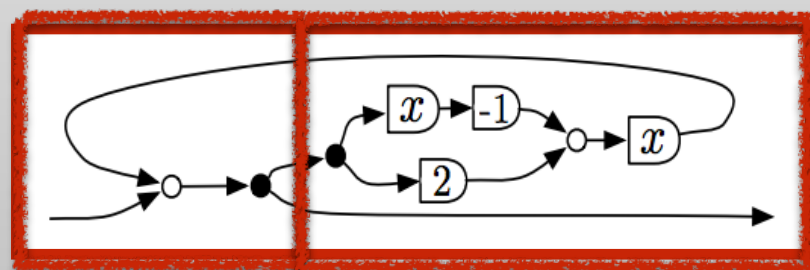
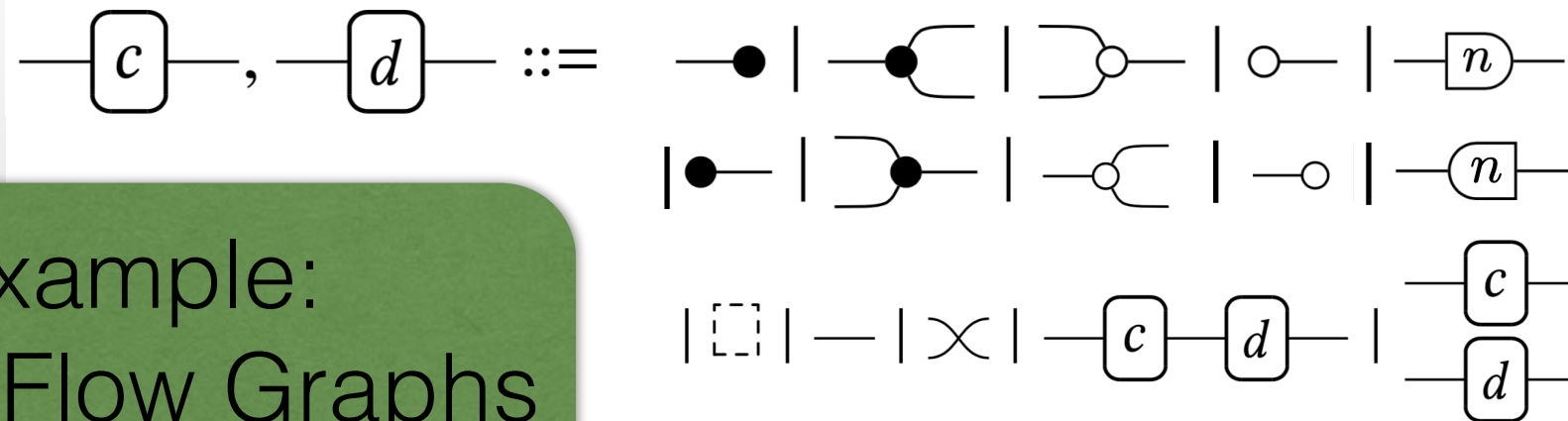
↕  
 $R_1$

↕  
 $R_2$

# Key Ideas

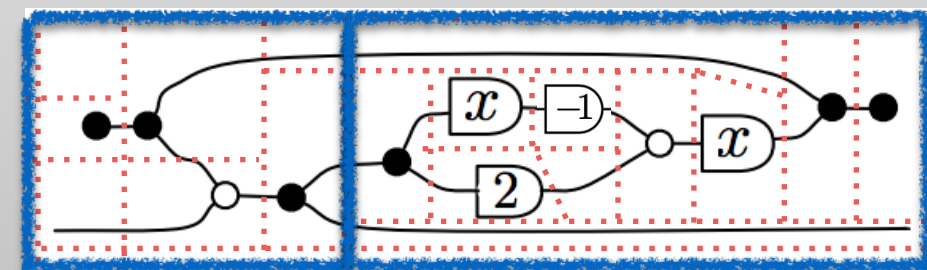
Graphical notation becomes **formal syntax**  
and it is given **compositional** semantics

Example:  
Signal Flow Graphs



↕  
?

↕  
?



↕  
 $R_1$

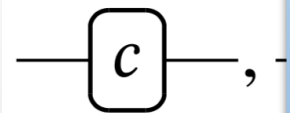
;

↕  
 $R_2$

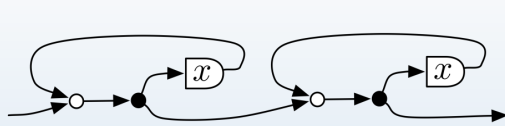
=

$R$

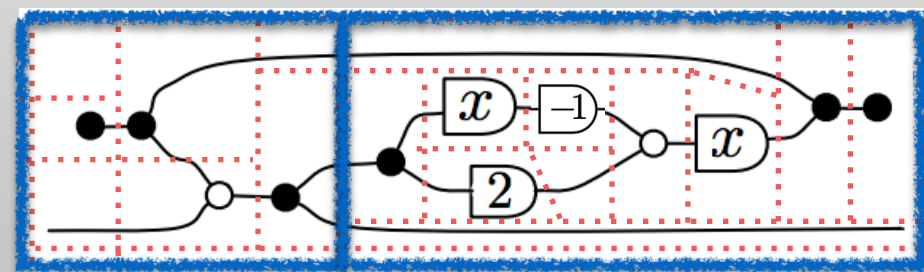
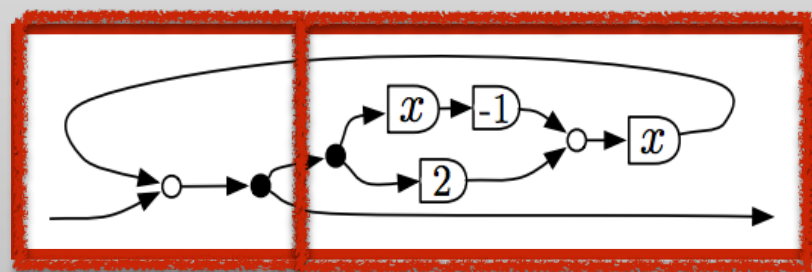
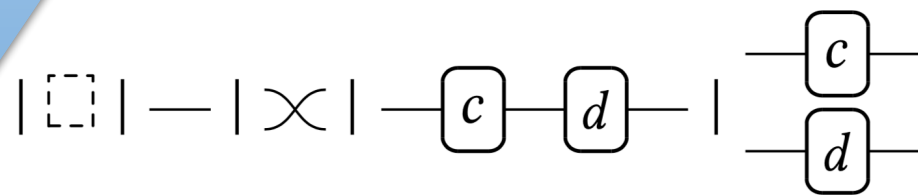
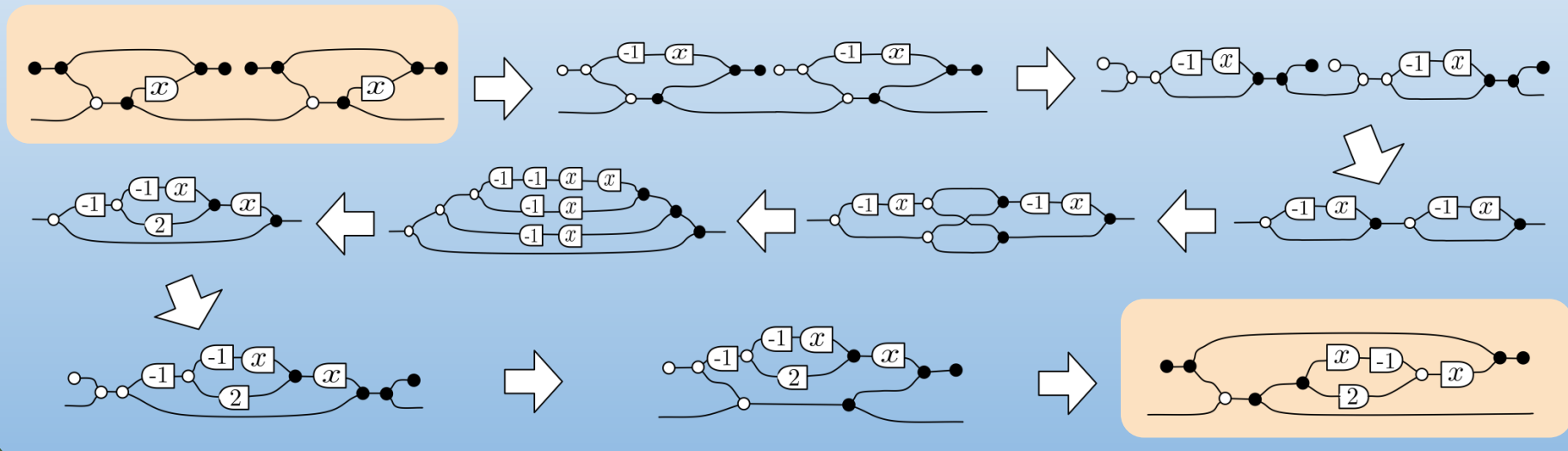
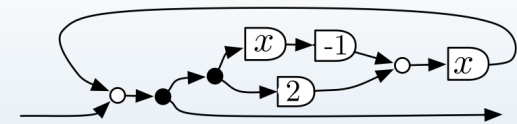
Graphical  
and it is



Example:  
Signal Flow Graphs



?



?

?

$R_1$

;

$R_2$

=

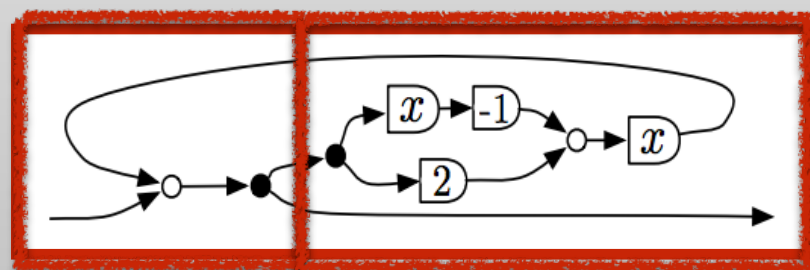
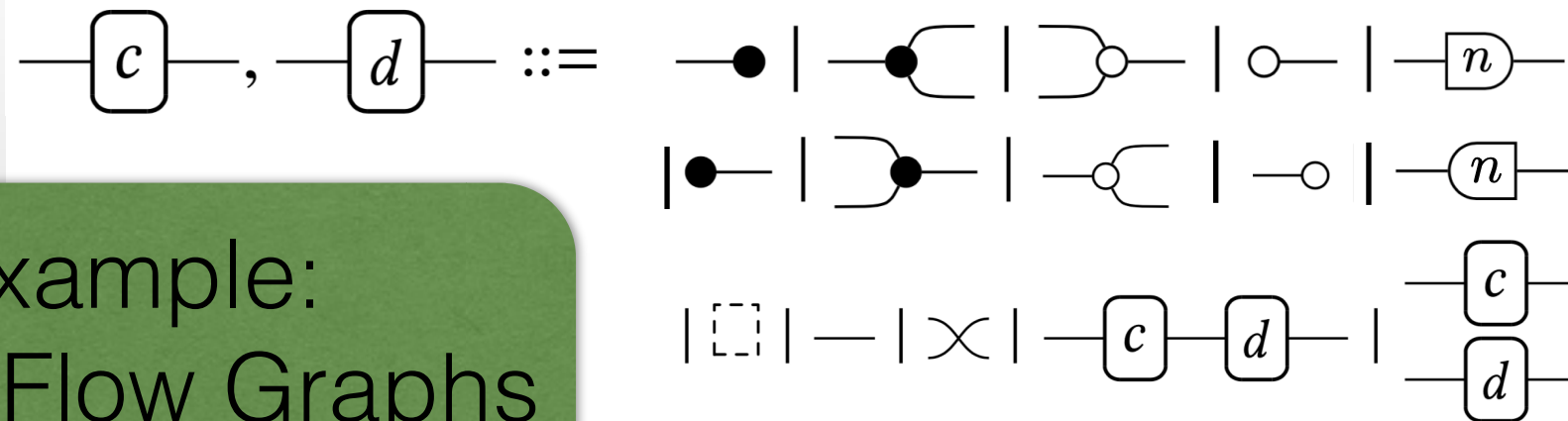
$R$



# Key Ideas

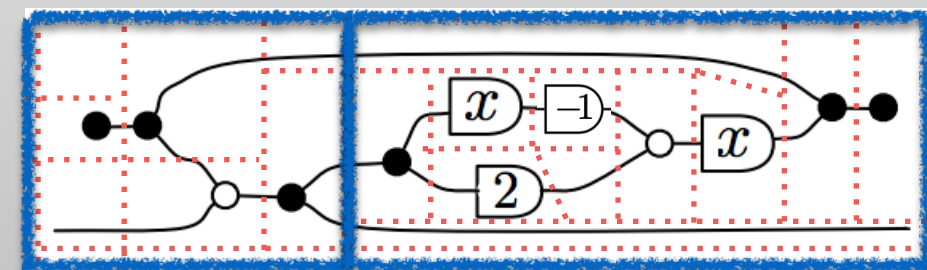
Graphical notation becomes **formal syntax**  
and it is given **compositional** semantics

Example:  
Signal Flow Graphs



↕  
?

↕  
?



↕  
 $R_1$

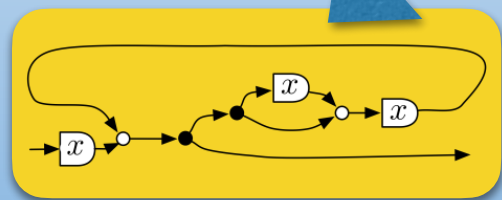
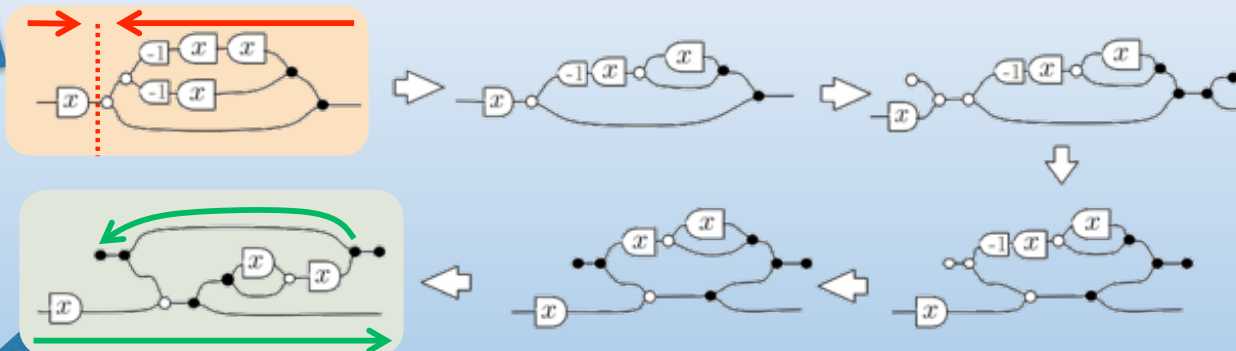
;

↕  
 $R_2$

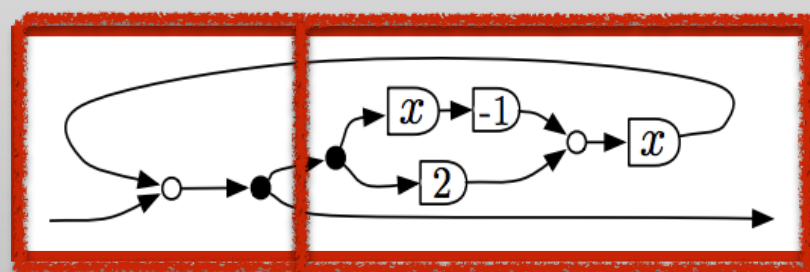
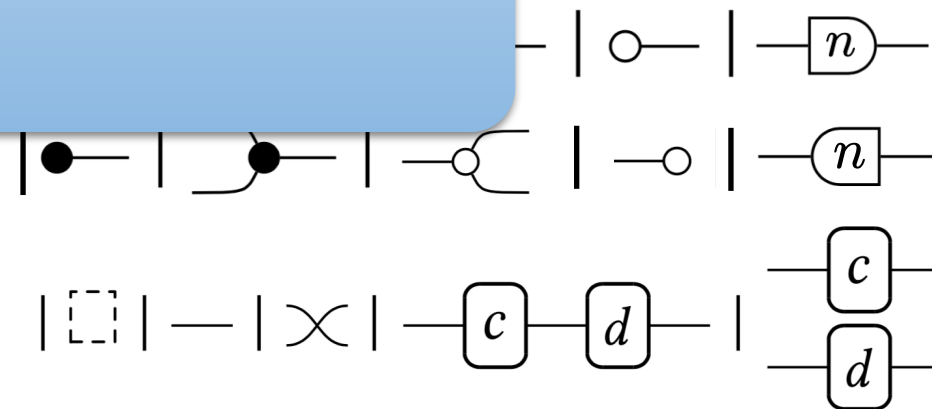
=

$R$

$$\frac{x}{1 - x - x^2}$$

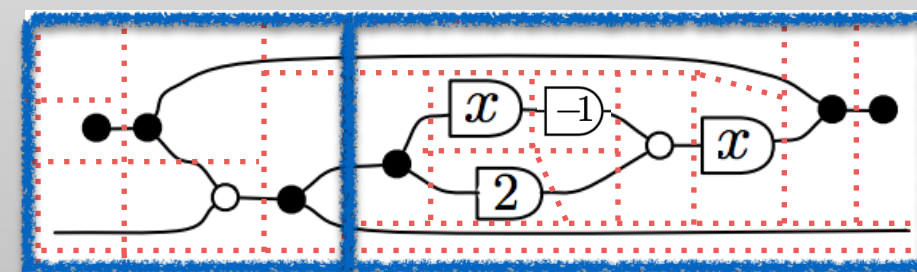


Example:  
Signal Flow Graphs



↕  
?

↕  
?



↕  
 $R_1$

;

↕  
 $R_2$

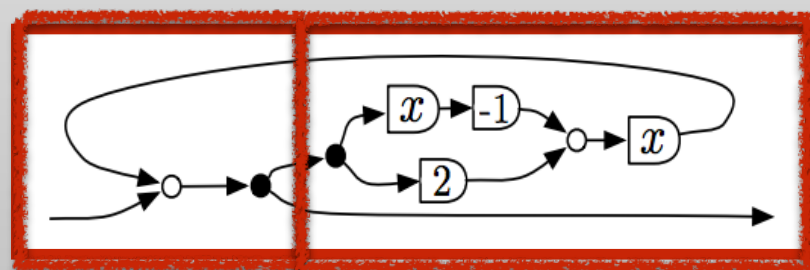
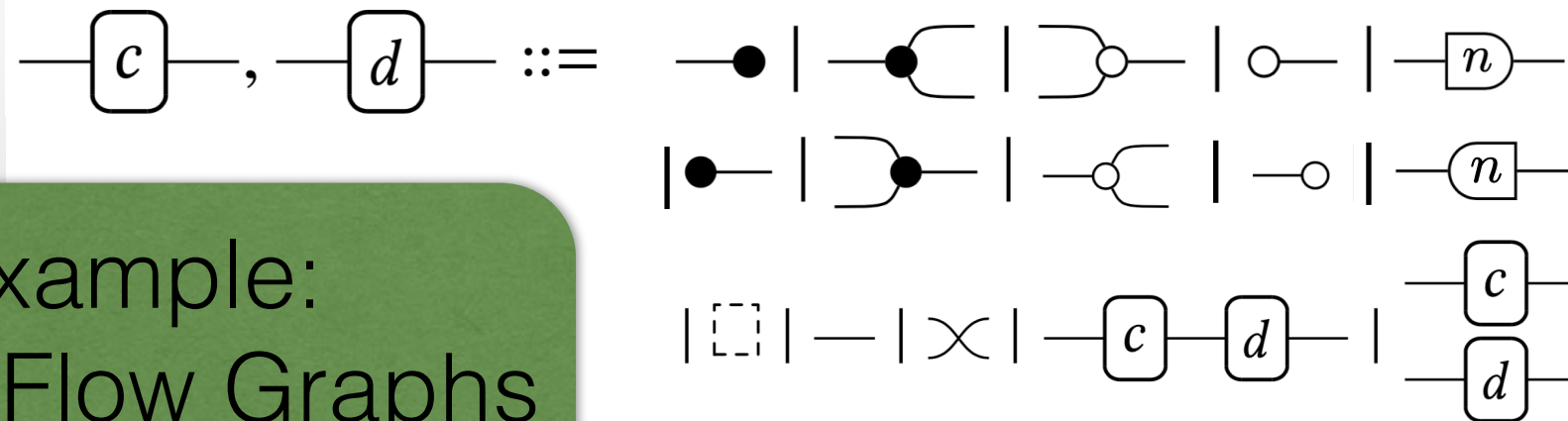
=

$R$

# Key Ideas

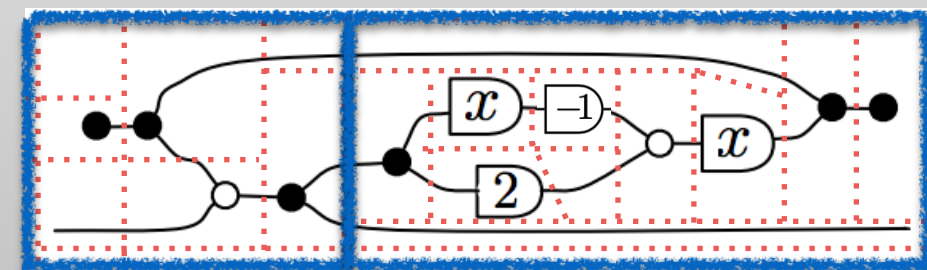
Graphical notation becomes **formal syntax**  
and it is given **compositional** semantics

Example:  
Signal Flow Graphs



↕  
?

↕  
?



↕  
 $R_1$

;

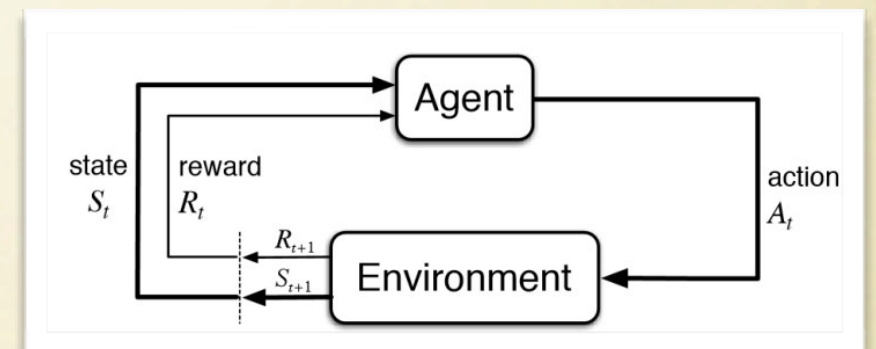
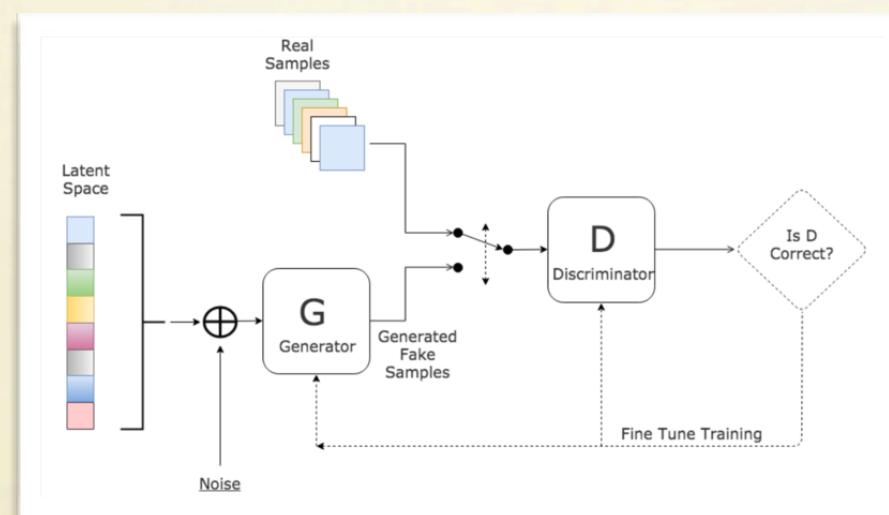
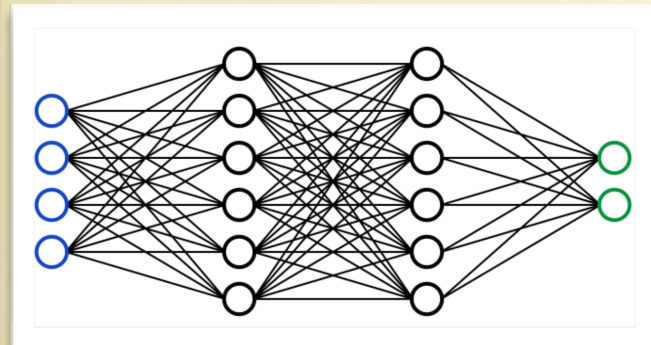
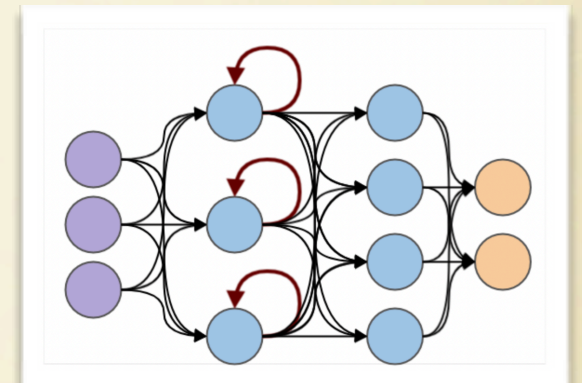
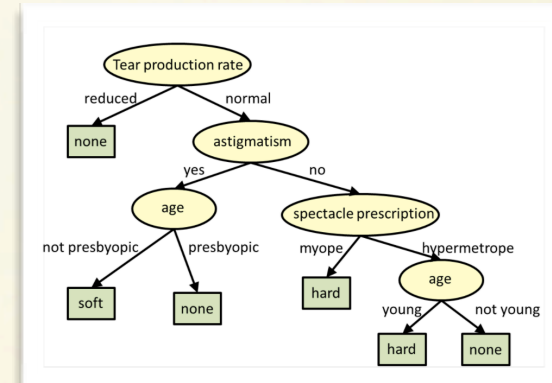
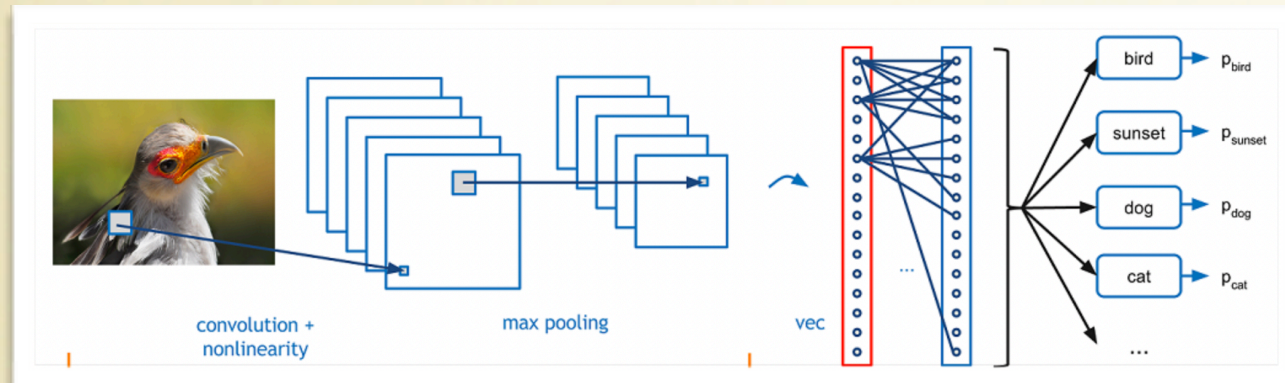
↕  
 $R_2$

=

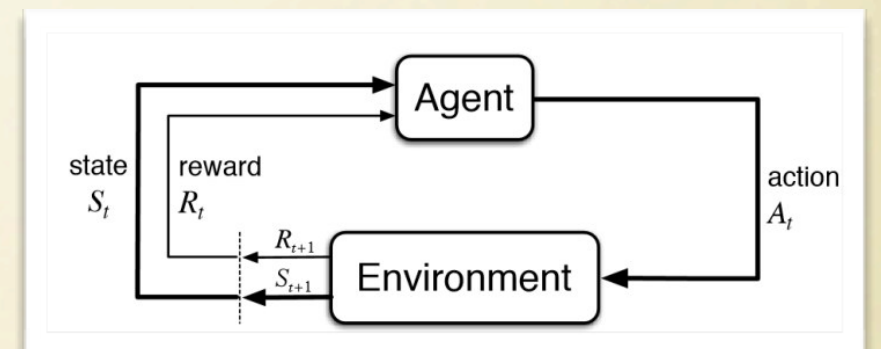
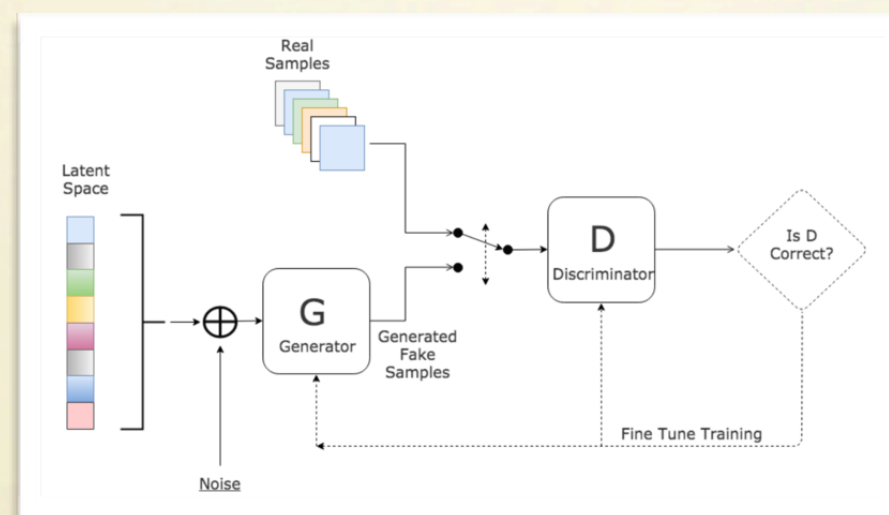
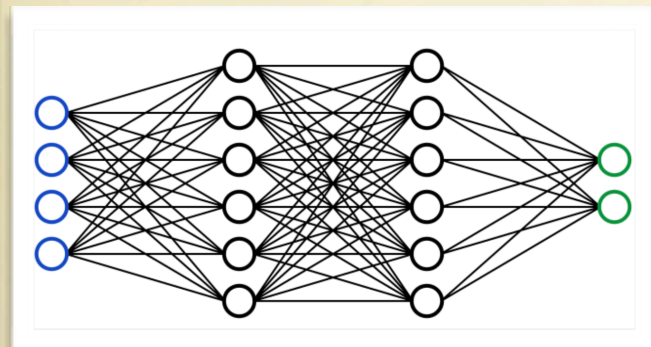
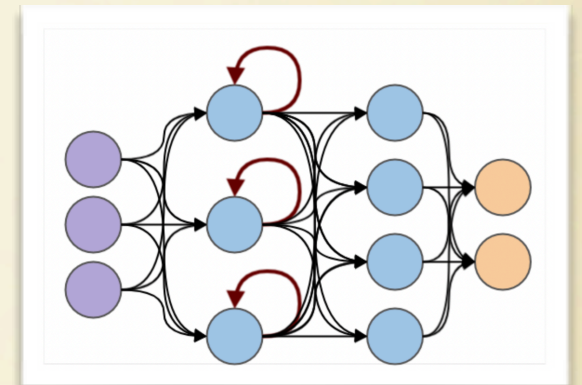
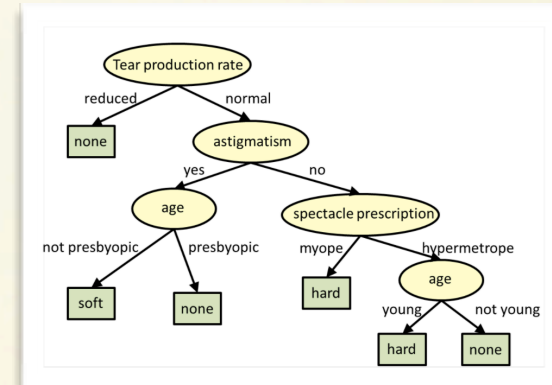
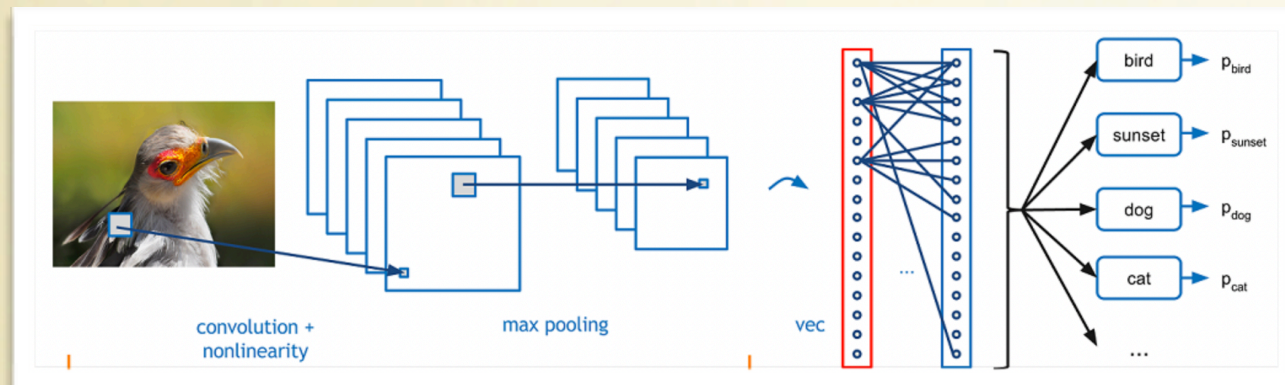
$R$



# Categorical Semantics of Machine Learning: what for?



# Categorical Semantics of Machine Learning: what for?



Uniformity

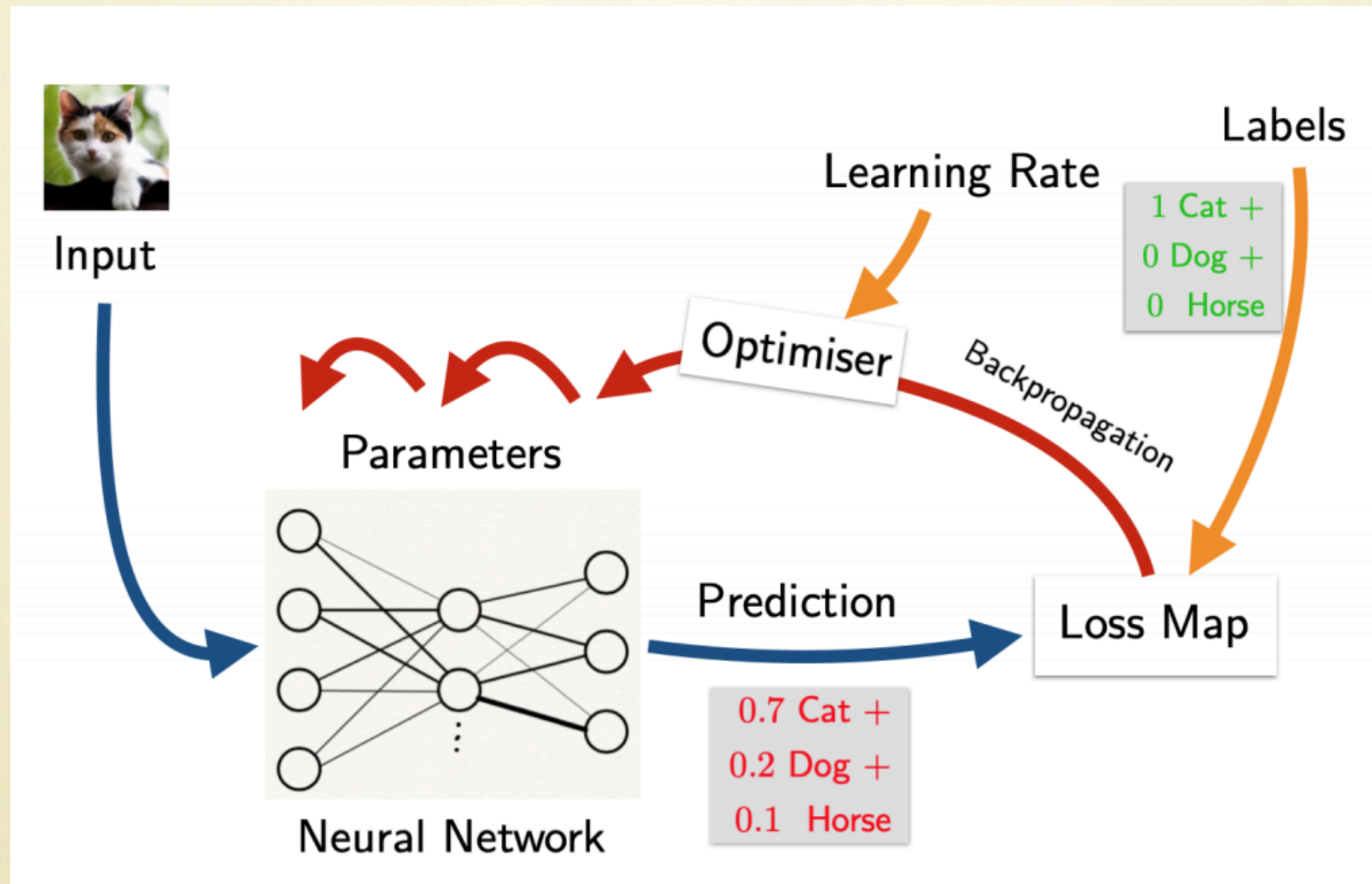
Explainability

Prog. Principles

Trustworthiness

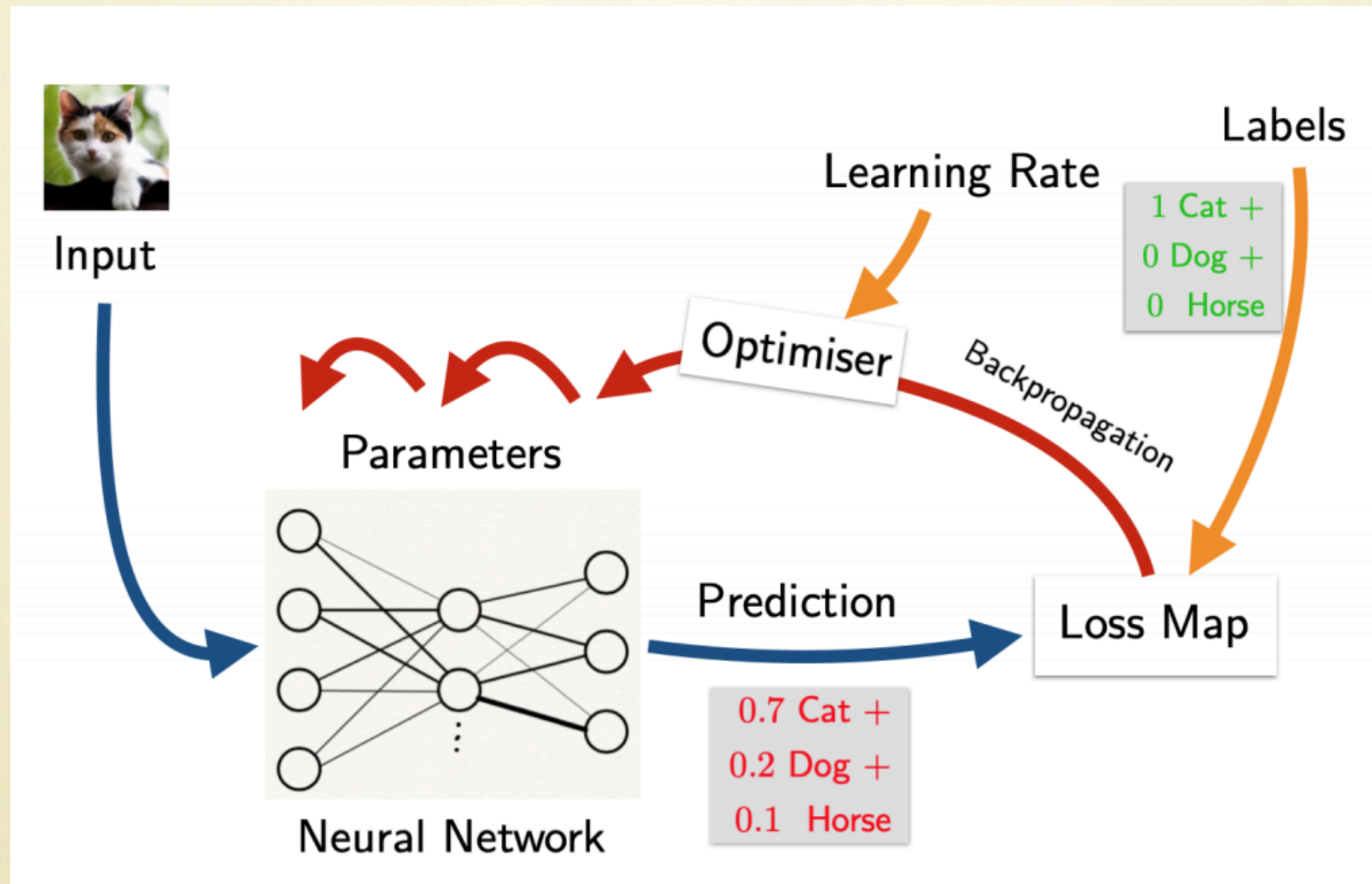


# Case study: Gradient-Based Learning



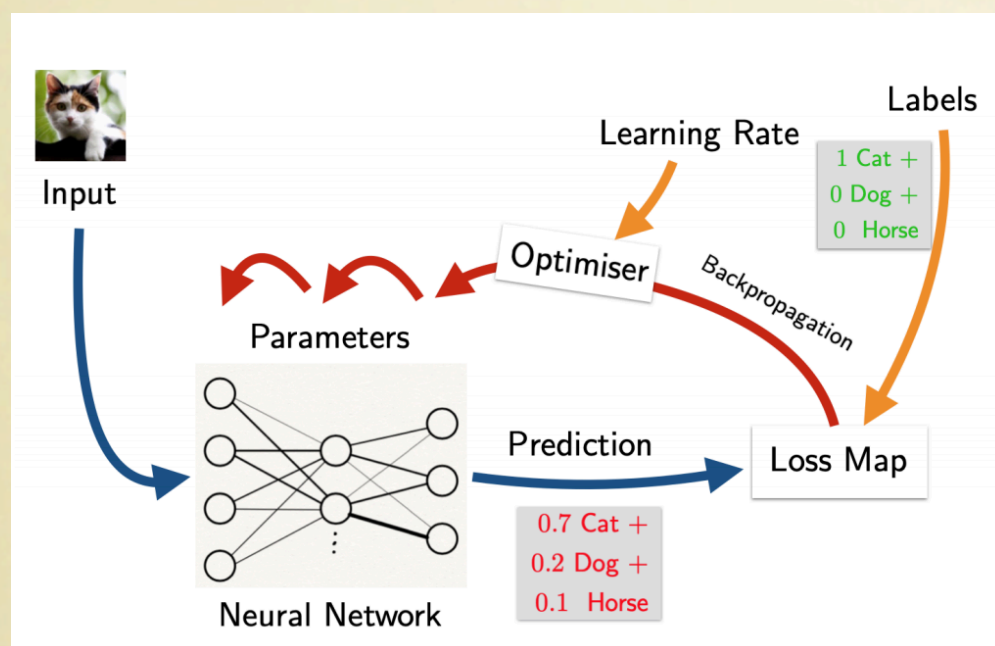


# Case study: Gradient-Based Learning

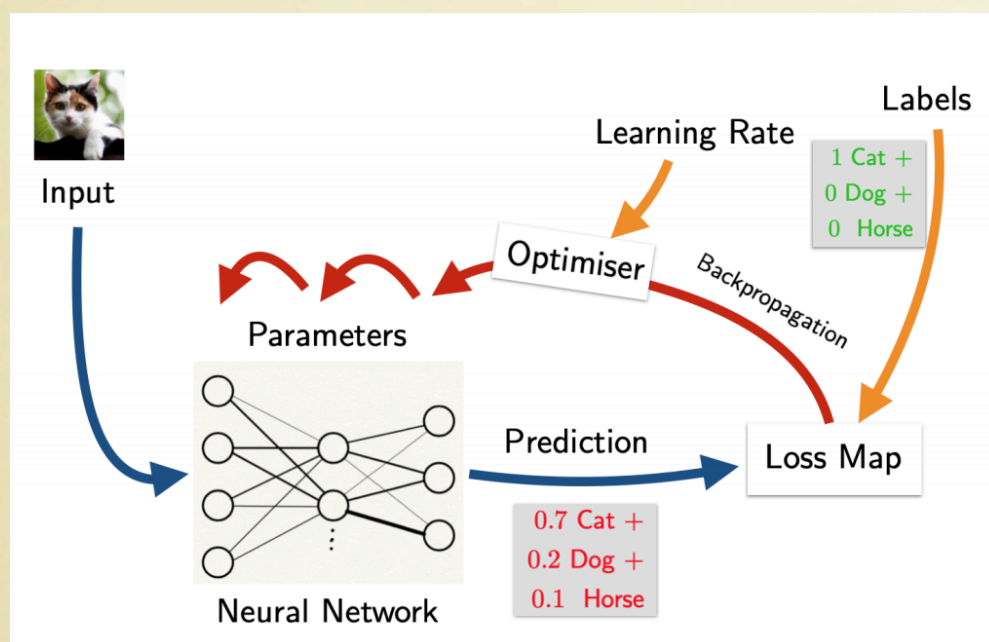


What are the fundamental mathematical structures underpinning gradient-based learning?

# Case study: Gradient-Based Learning

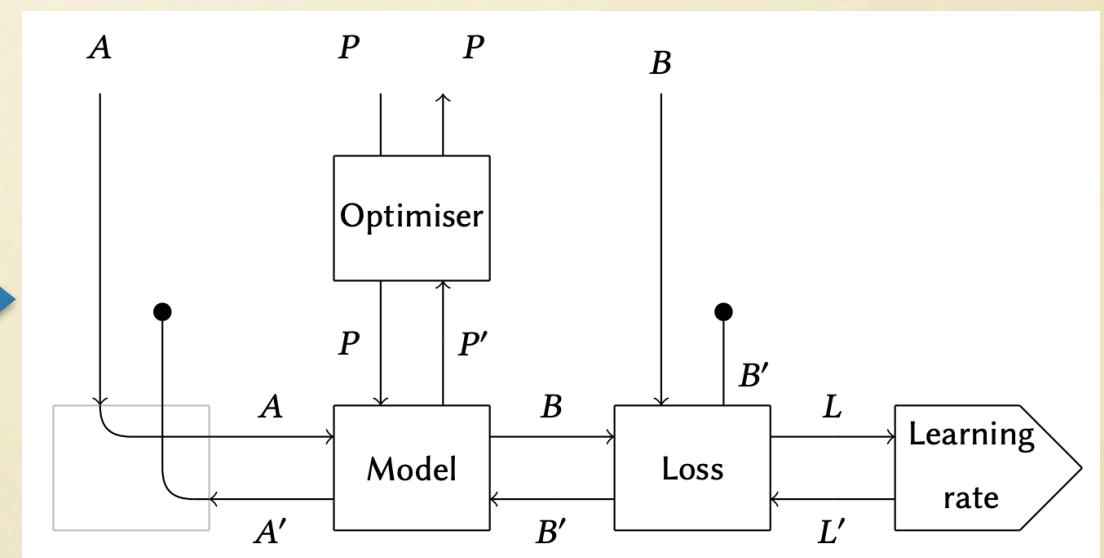
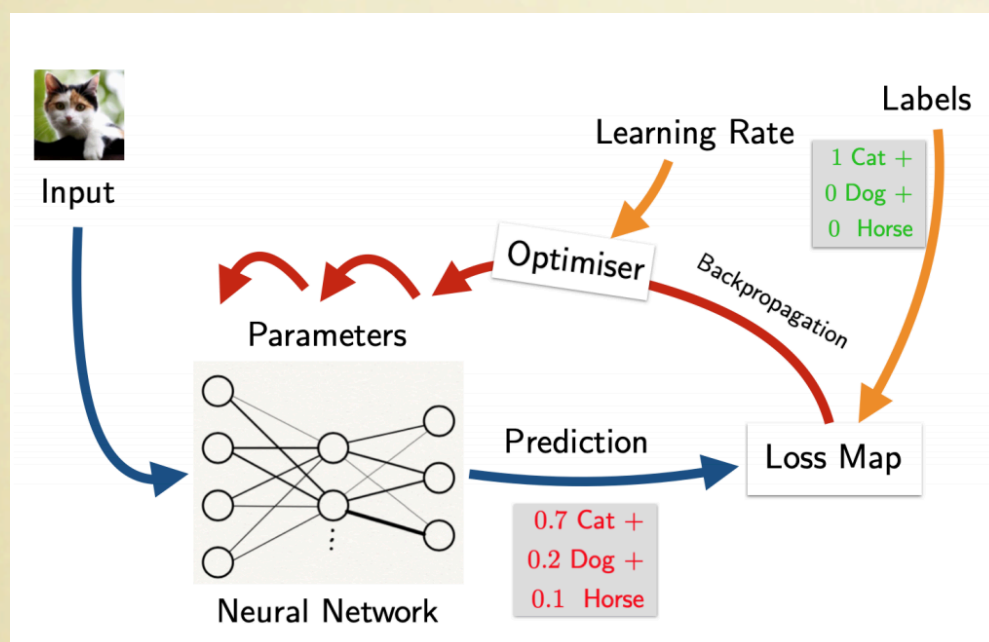


# Roadmap





# Roadmap



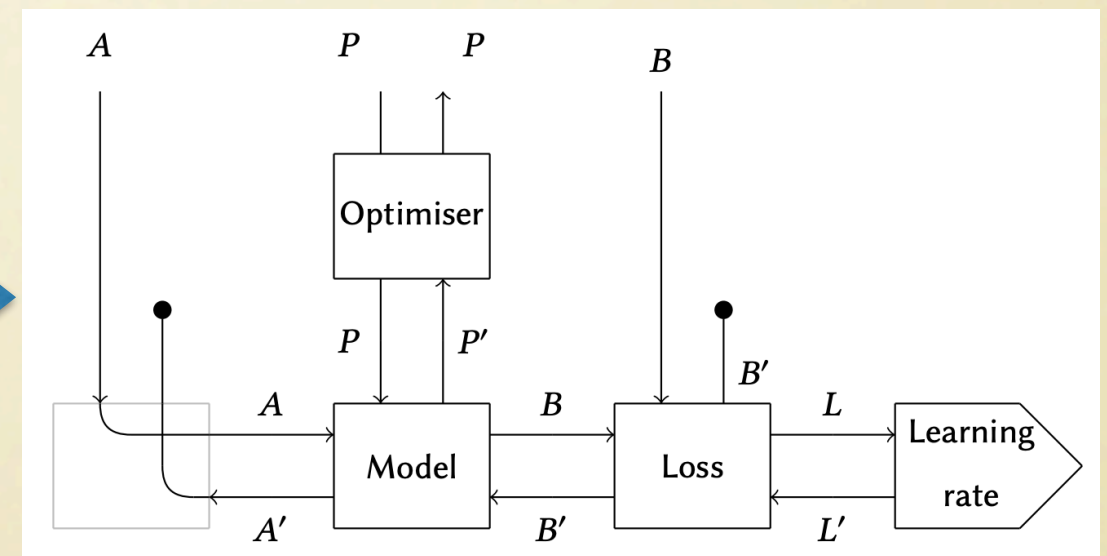
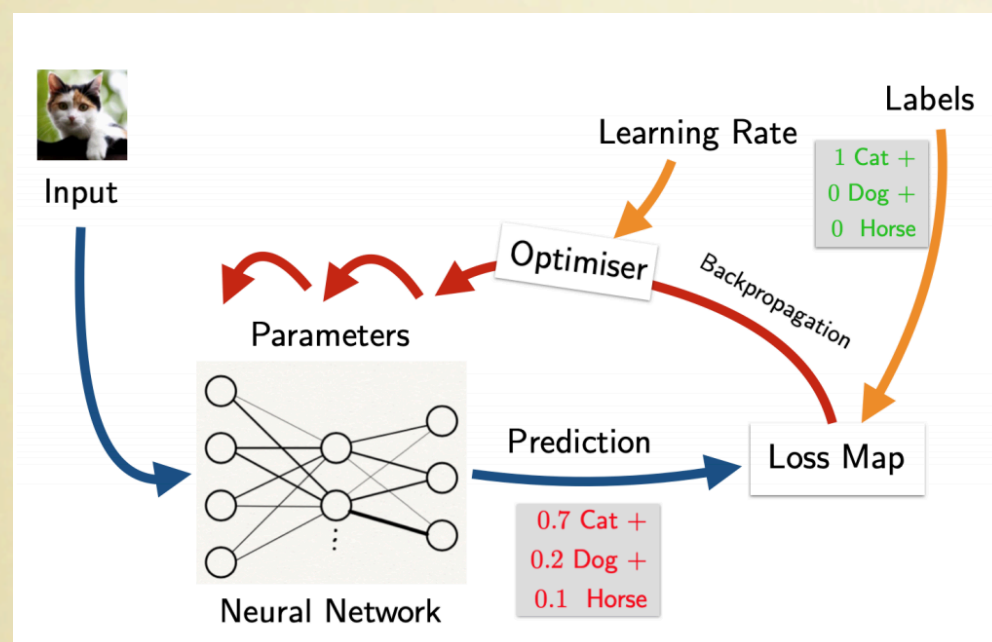
# Roadmap

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

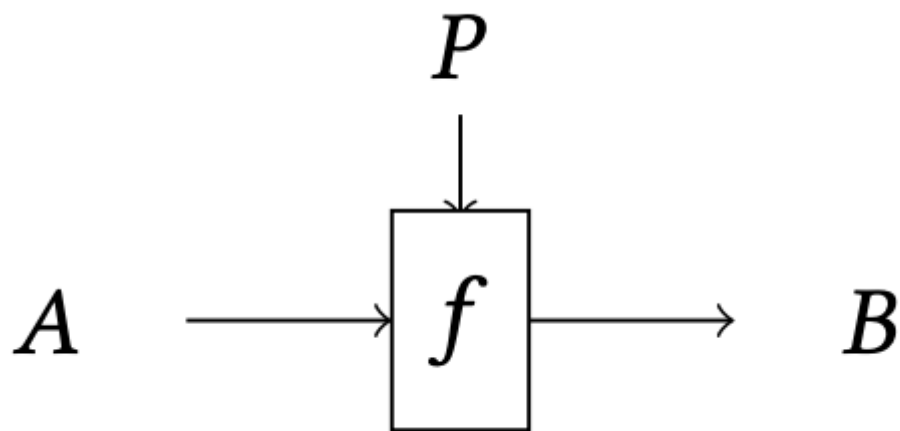
Parametric lens = parametric map with lens structure



# Parametric Lenses

Parametric lens = parametric map with lens structure

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

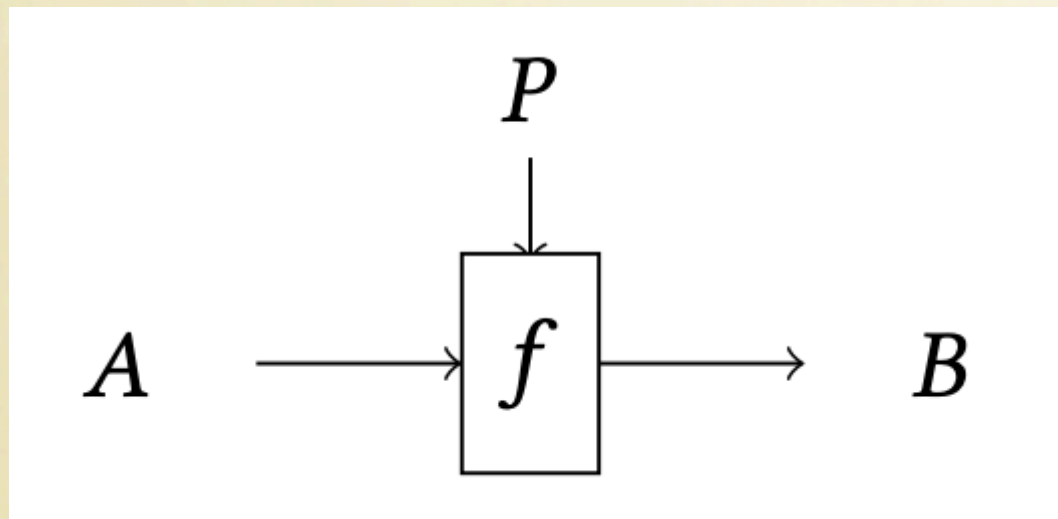


***Para(C)***

# Parametric Lenses

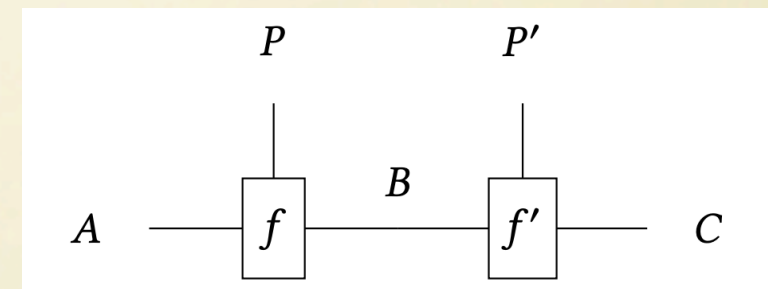
Parametric lens = parametric map with lens structure

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

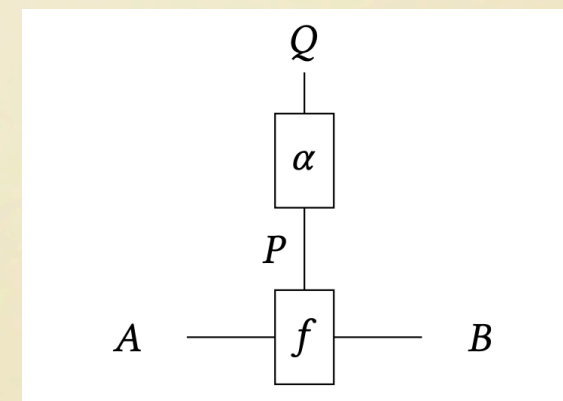


***Para(C)***

Composition



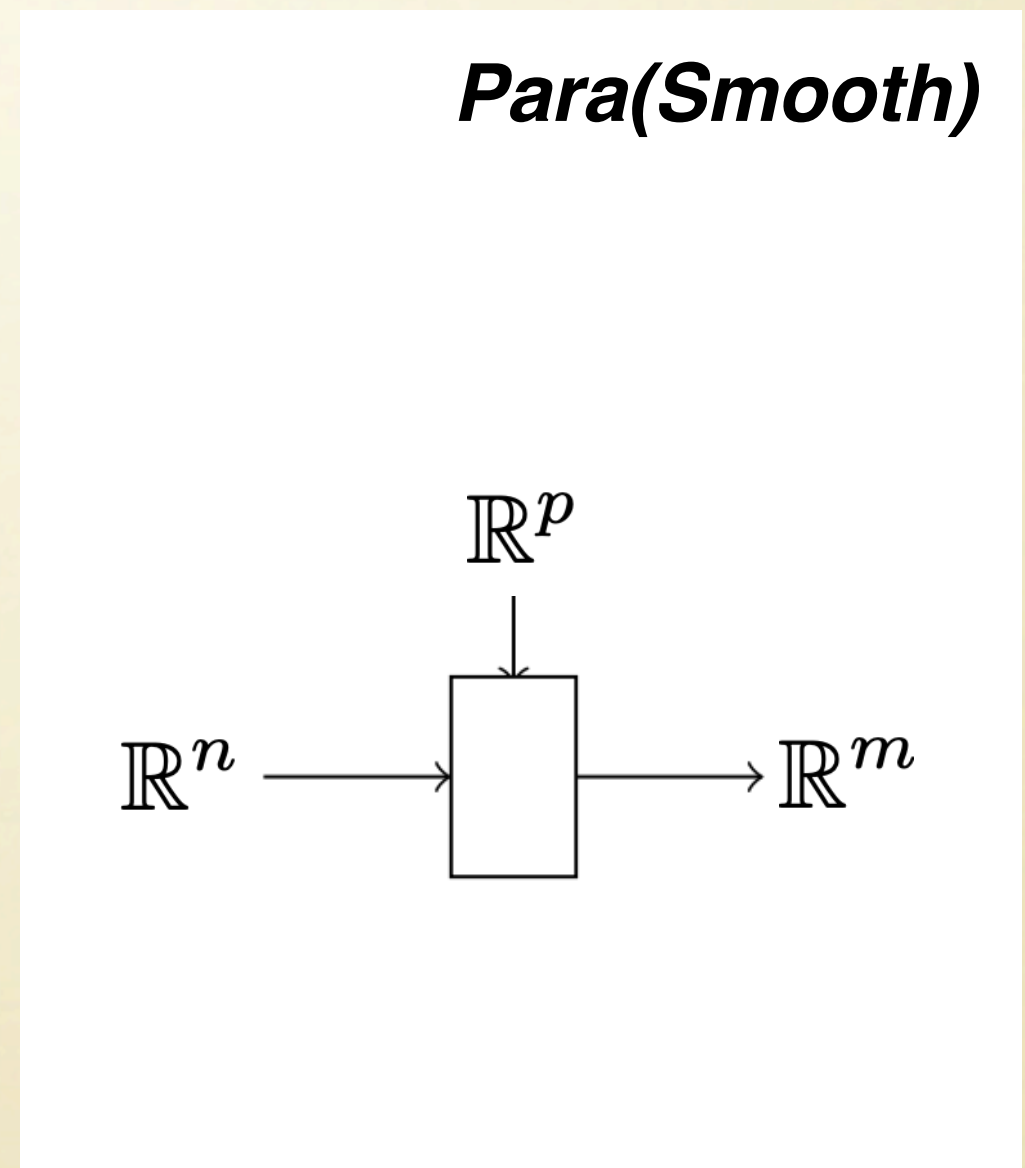
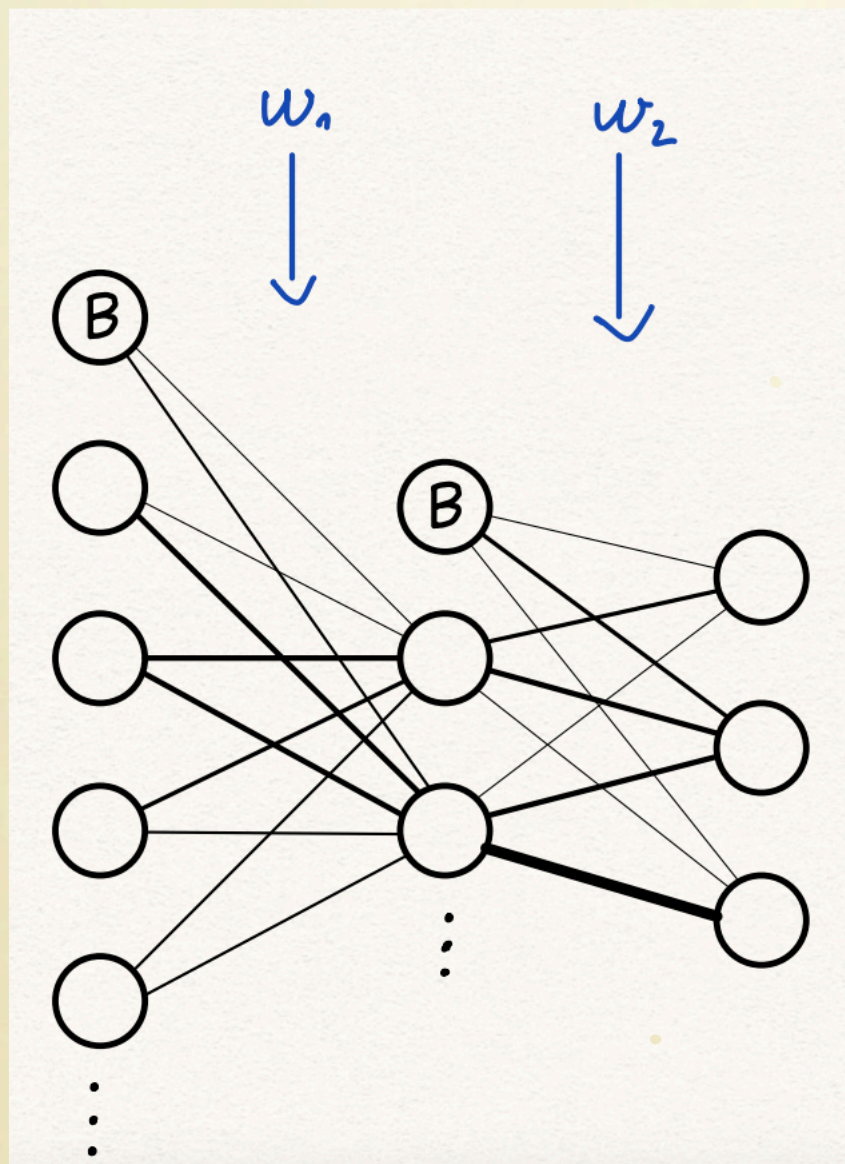
Reparametrisation





# Parametric Maps

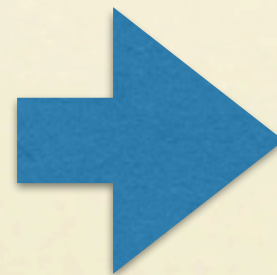
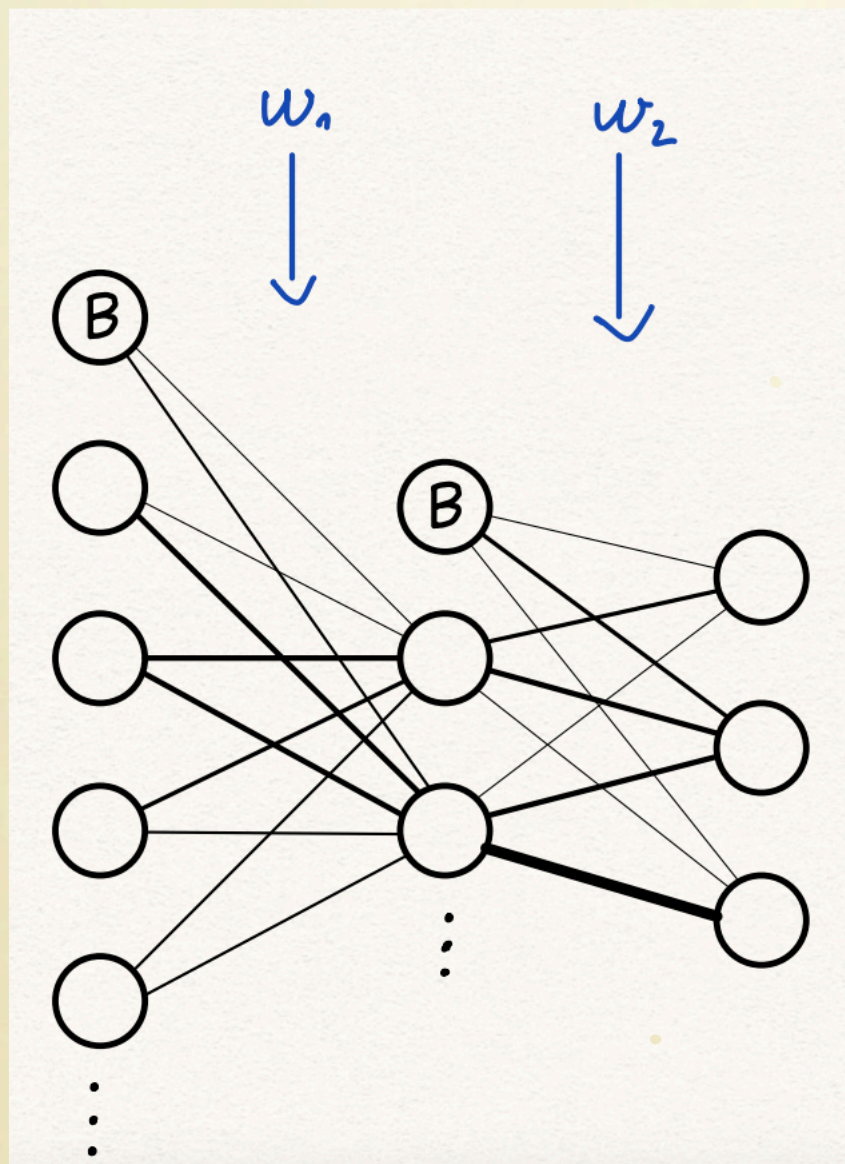
A neural network is a parametric map



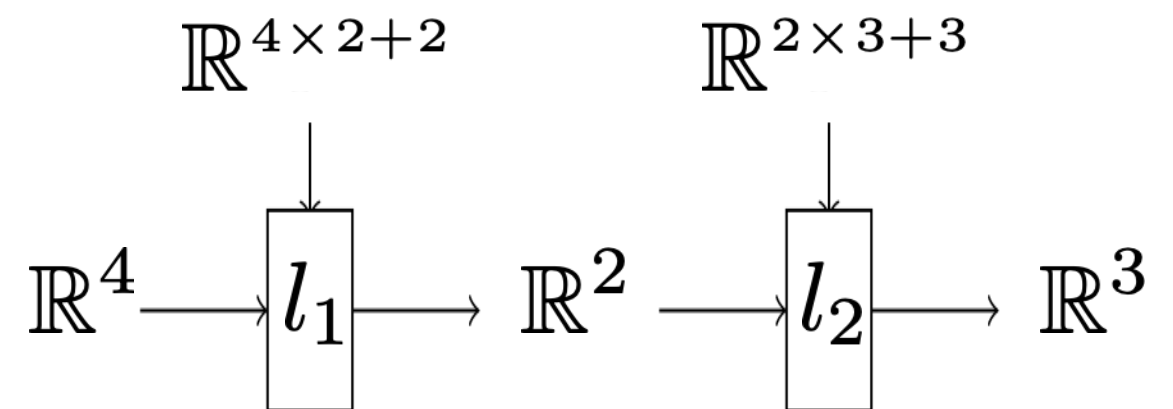


# Parametric Maps

A neural network is a parametric map



***Para(Smooth)***

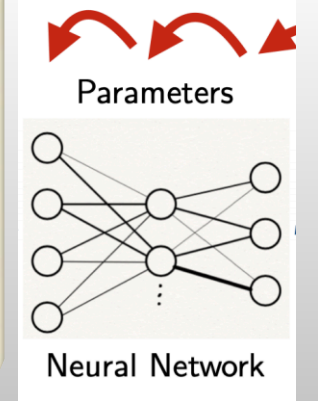


# Lenses

Parametric lens = parametric map with lens structure



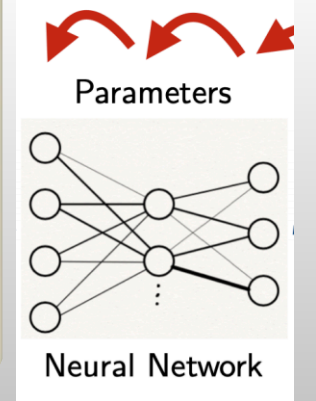
# Lenses



Parametric lens = parametric map with lens structure



# Lenses



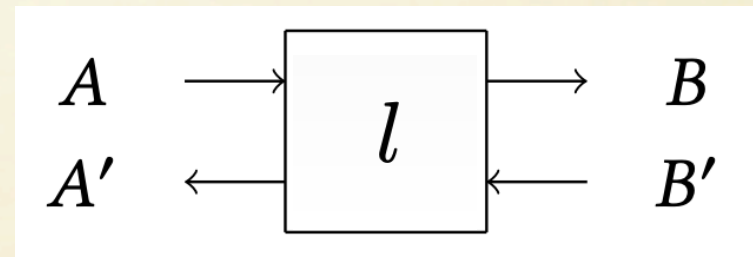
Parametric lens = parametric map with lens structure

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

---

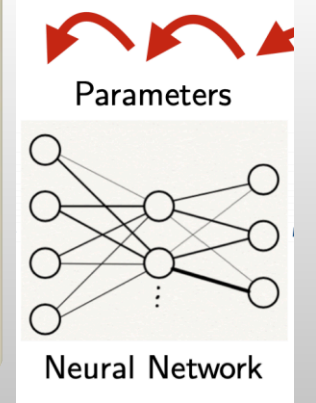
$$\text{get}_l: A \rightarrow B$$

$$\text{put}_l: A \times B' \rightarrow A'$$



***Lens(C)***

# Lenses



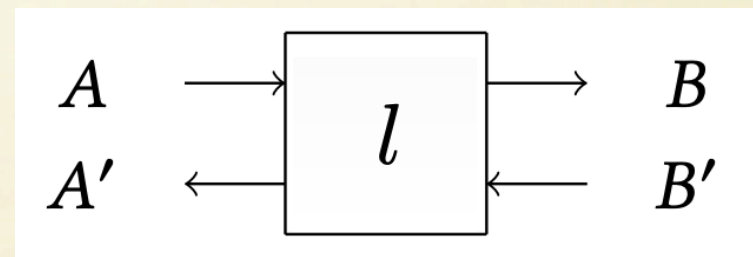
Parametric lens = parametric map with lens structure

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

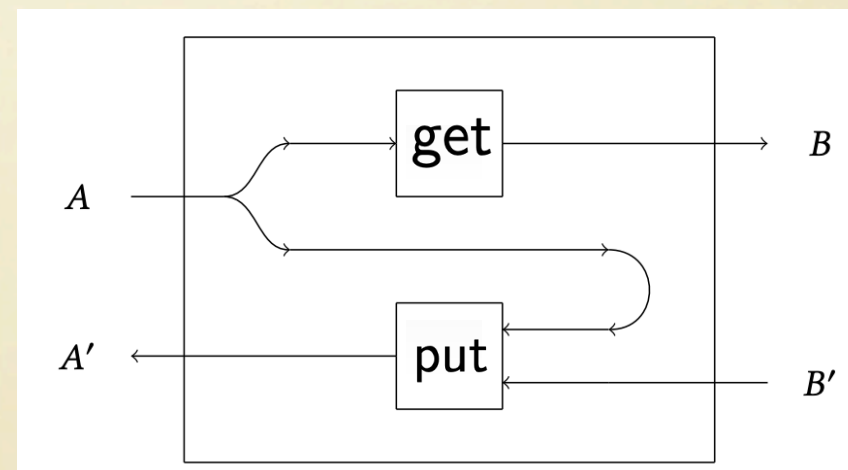
---


$$\text{get}_l: A \rightarrow B$$

$$\text{put}_l: A \times B' \rightarrow A'$$



‘Opening the box’:



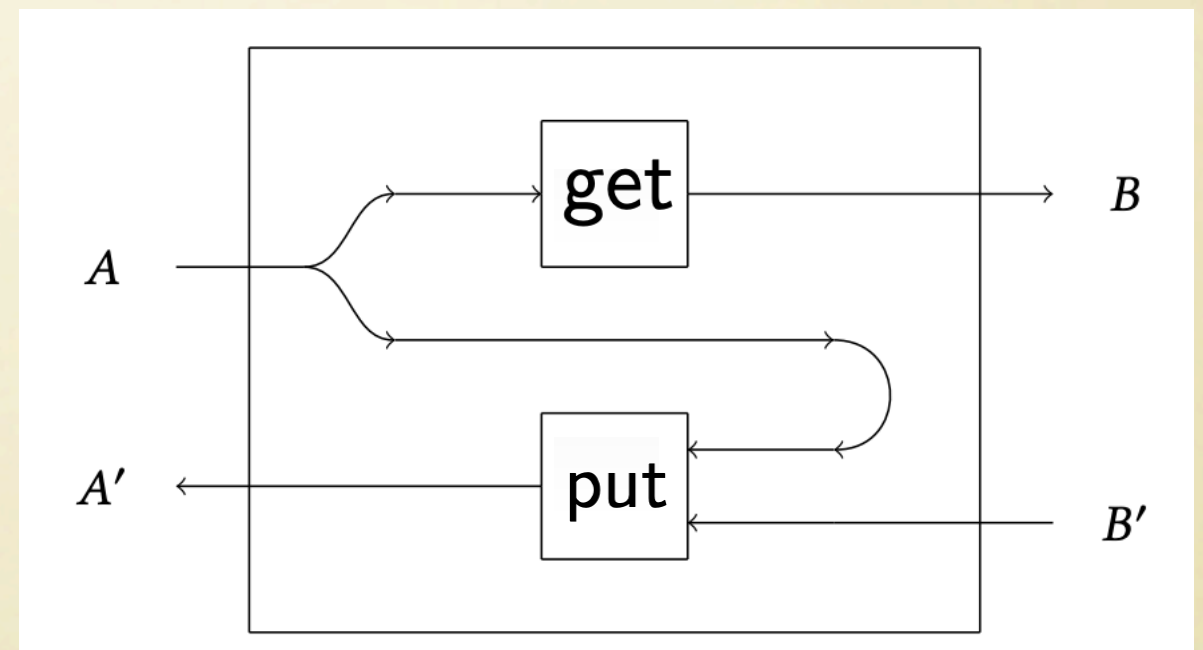
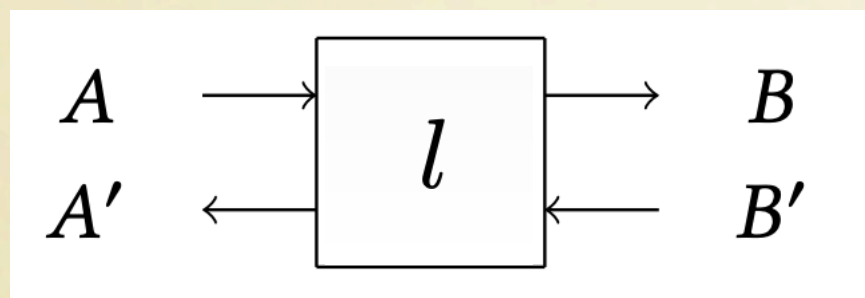
***Lens(C)***



# 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**

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

---

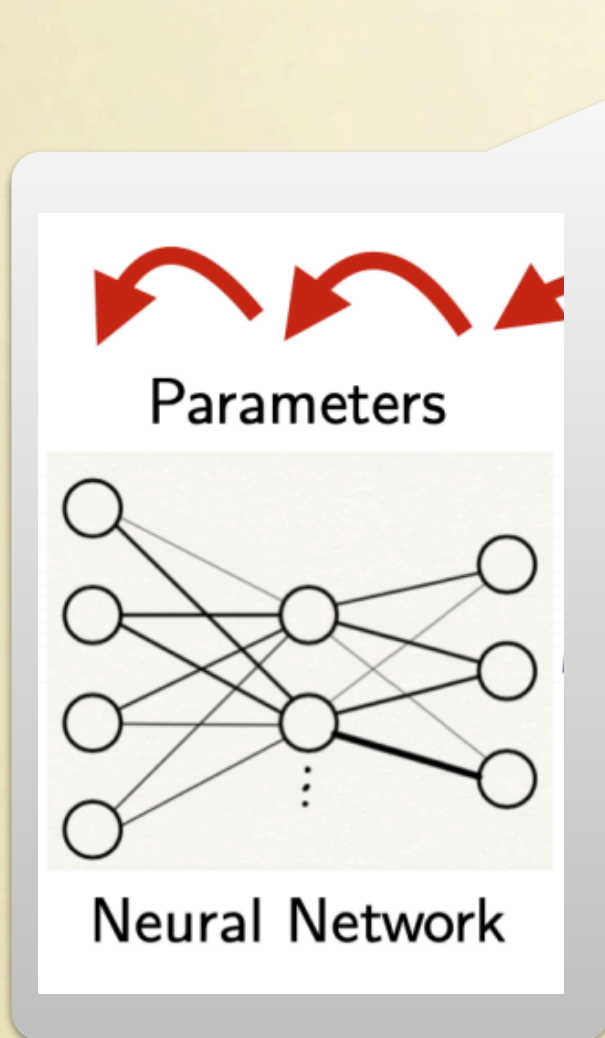
$$f: A \rightarrow B$$

$$R[f]: A \times B' \rightarrow A'$$

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

# Lenses

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



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

$$\frac{}{f: A \rightarrow B}$$

$$R[f]: A \times B' \rightarrow A'$$

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



# Lenses

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

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

---


$$f: A \rightarrow B$$

$$R[f]: A \times B' \rightarrow A'$$

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

## 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, Canada  
gcruttwell@mta.ca

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

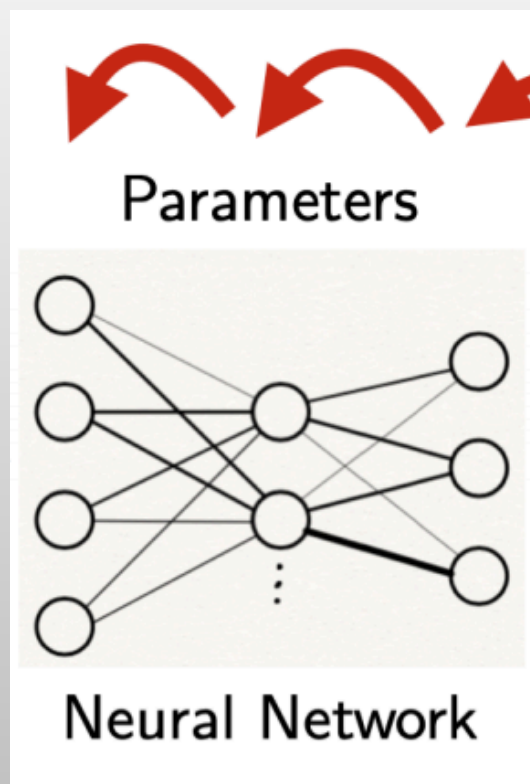
**Jean-Simon Pacaud Lemay**  
University of Oxford, Department of Computer Science, UK  
jean-simon.lemay@kellogg.ox.ac.uk

**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  
dorette.pronk@dal.ca

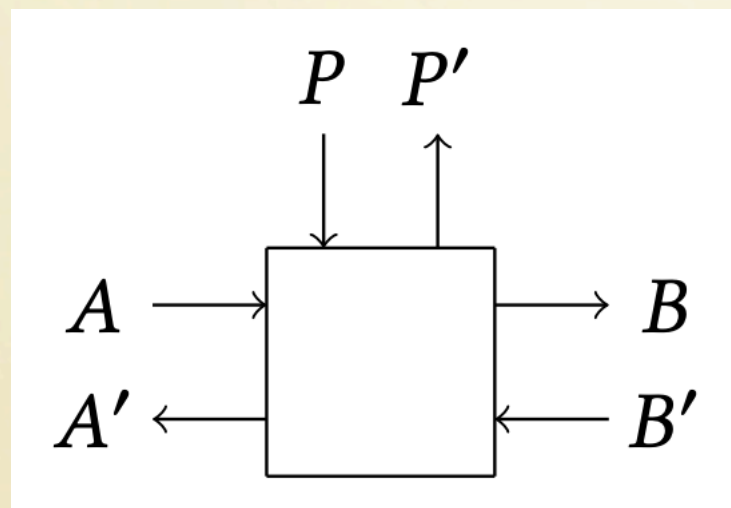
Reverse  
derivative  
of  $f$





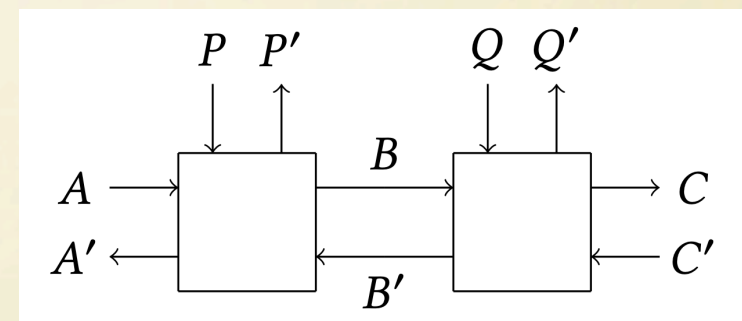
# Parametric Lenses

Parametric lens = parametric map with lens structure

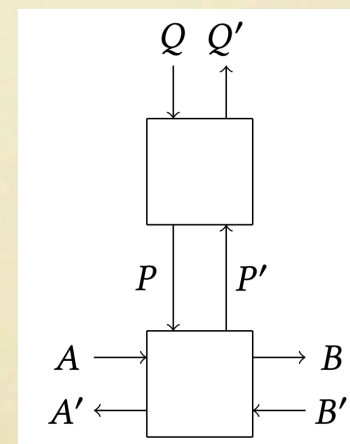


***Para(Lens(C))***

Composition

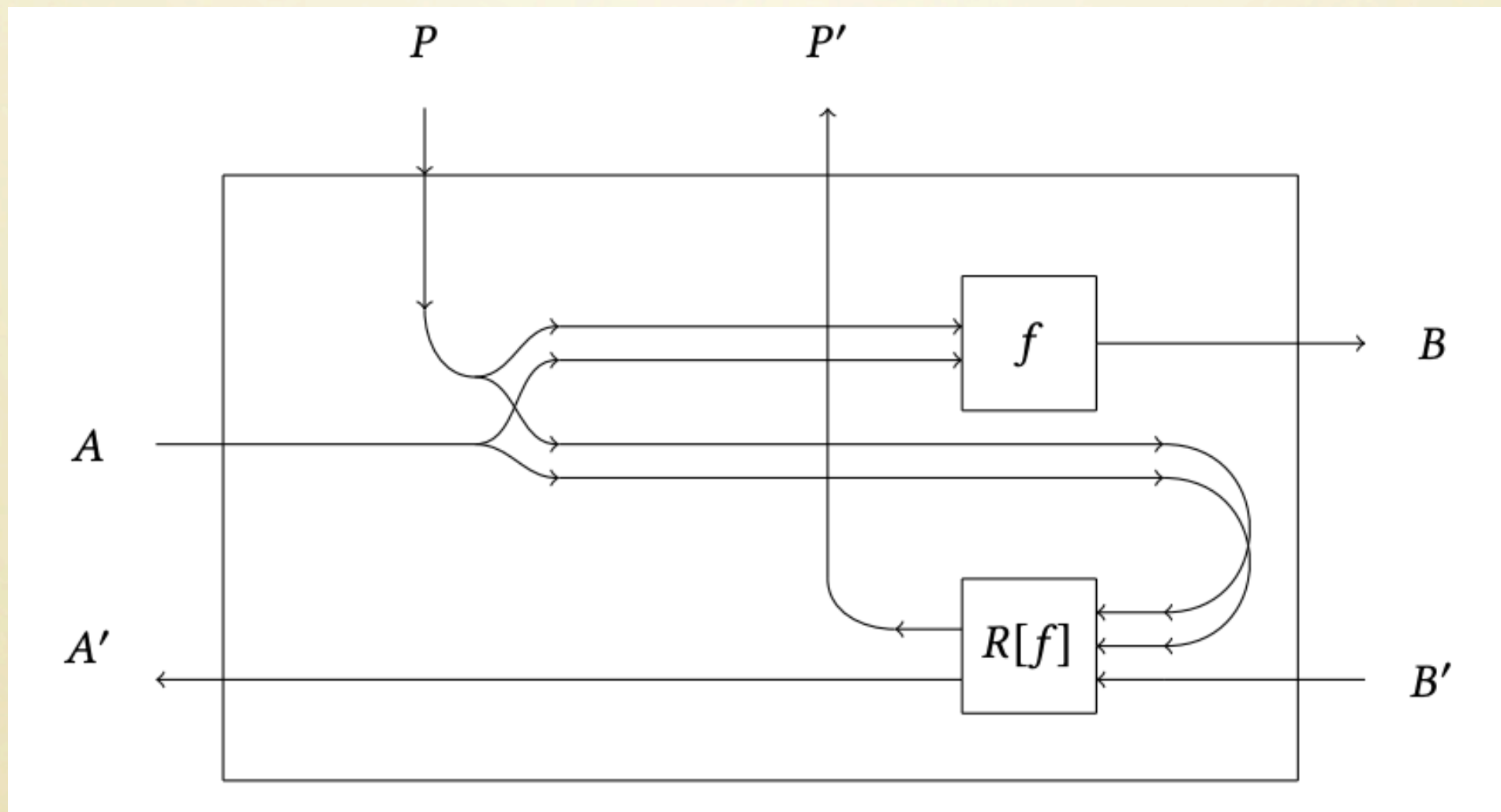


Reparametrisation



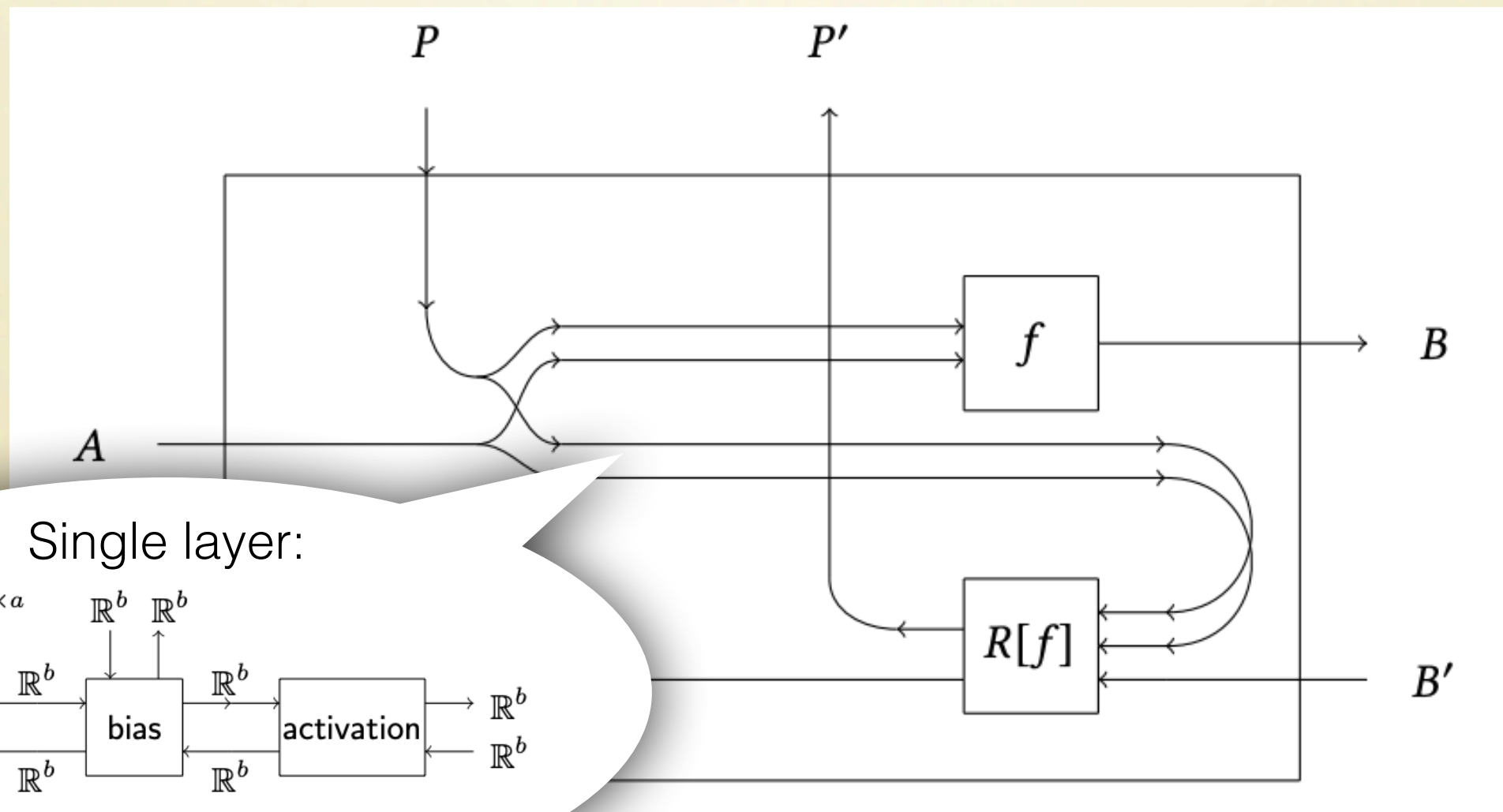
# Parametric Lenses

A neural network is a parametric lens



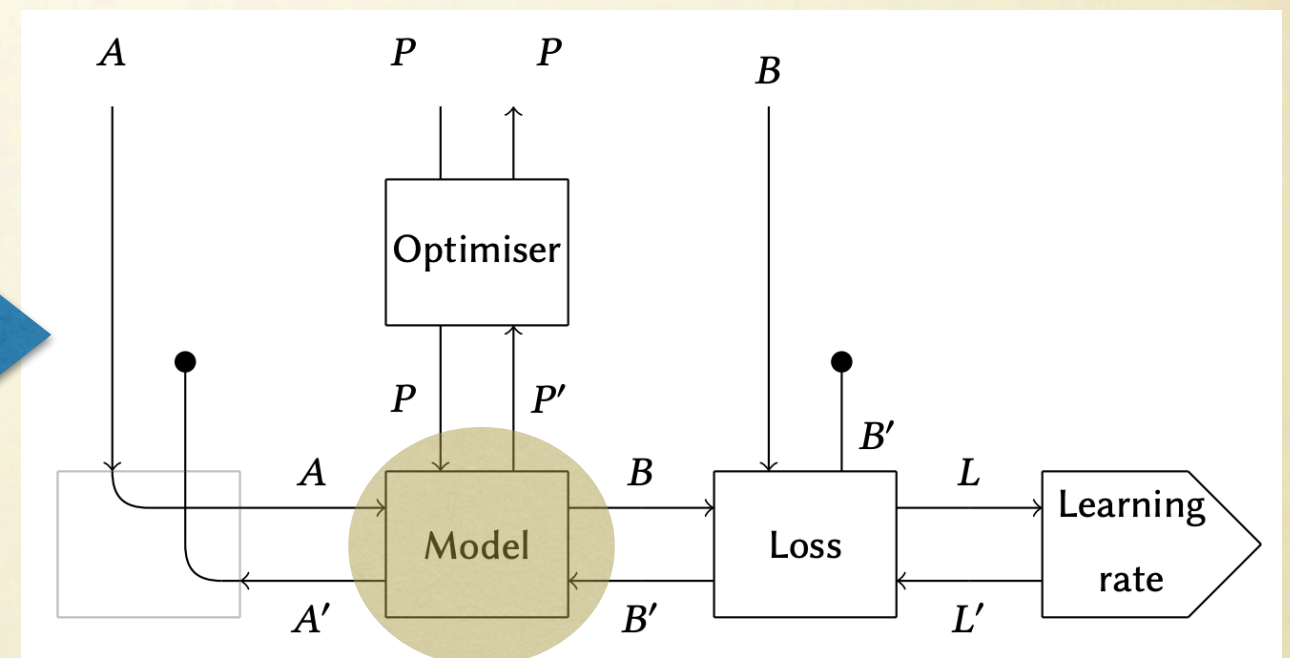
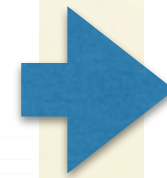
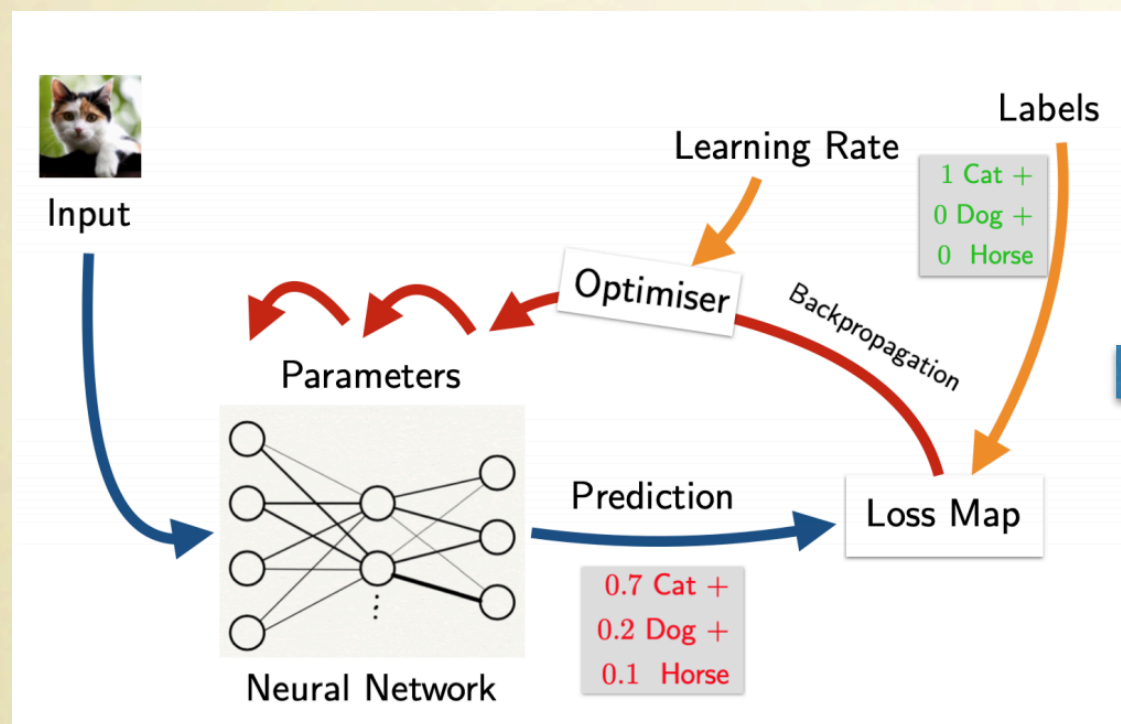
# Parametric Lenses

A neural network is a parametric lens

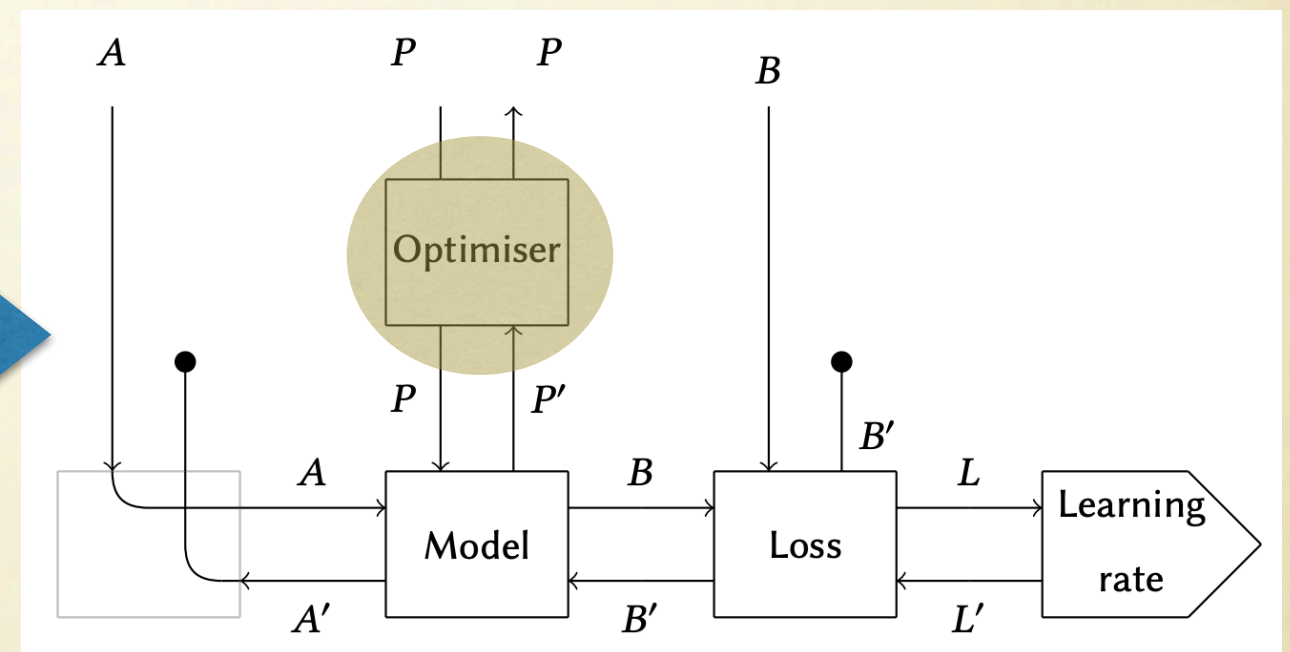
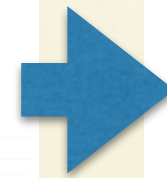
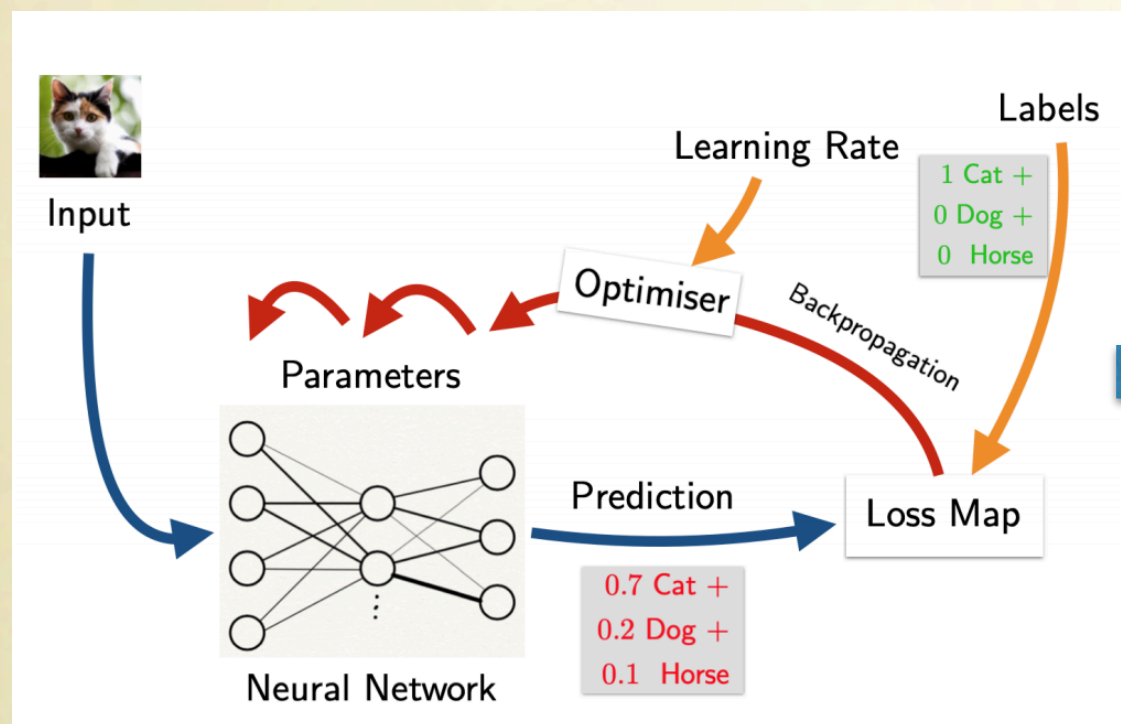




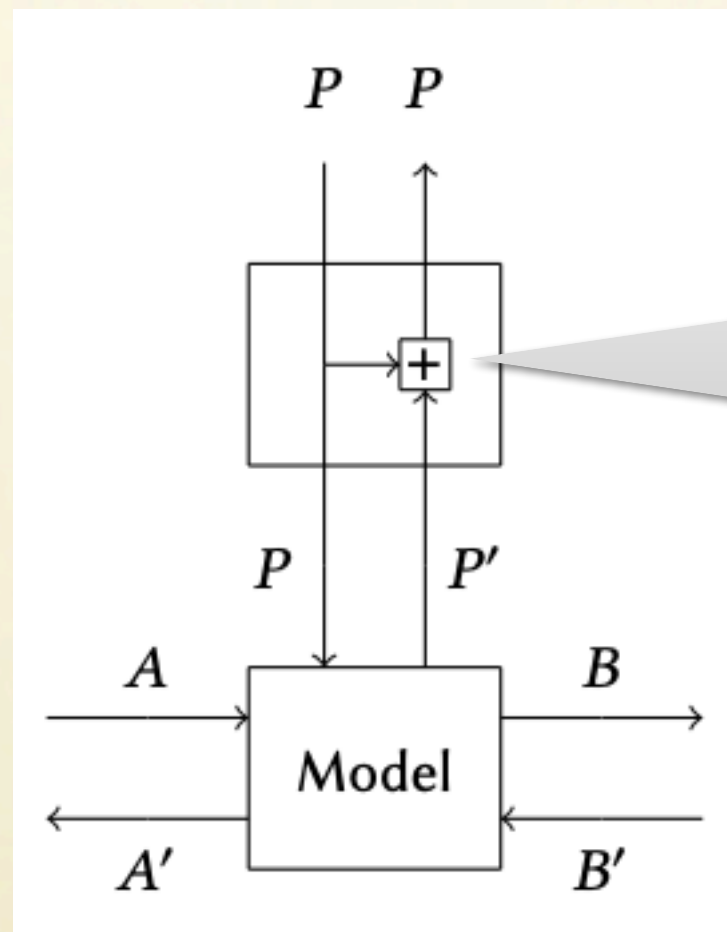
# Where we are, so far



# Where we are, so far



# Optimisers are reparametrisations



***Lens(Smooth)***

$gd: (P, P) \rightarrow (P, P')$

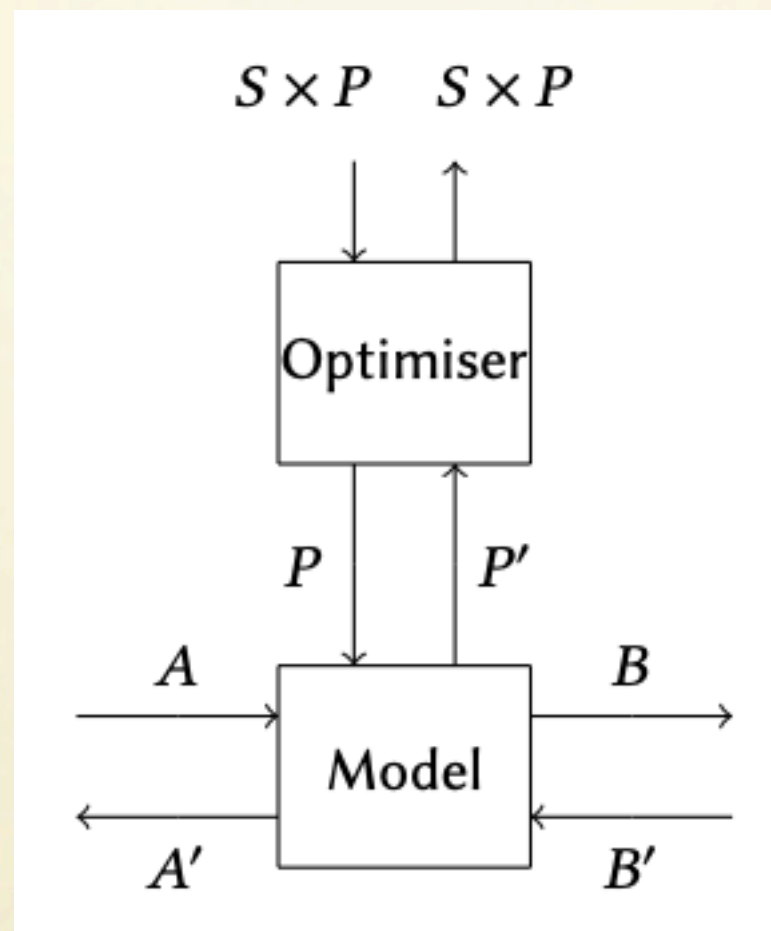
$get_{gd}: p \mapsto p$

$put_{gd}: (p, p') \mapsto p + p'$

Basic gradient descent

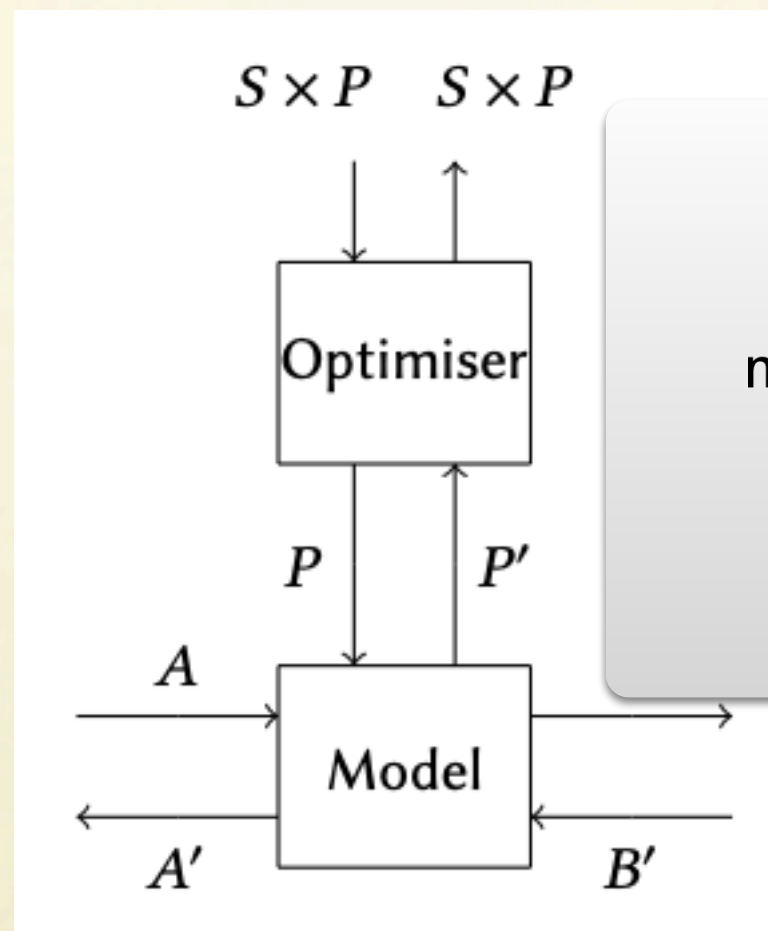


# Optimisers are reparametrisations



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

# Optimisers are reparametrisations



## ***Lens(Smooth)***

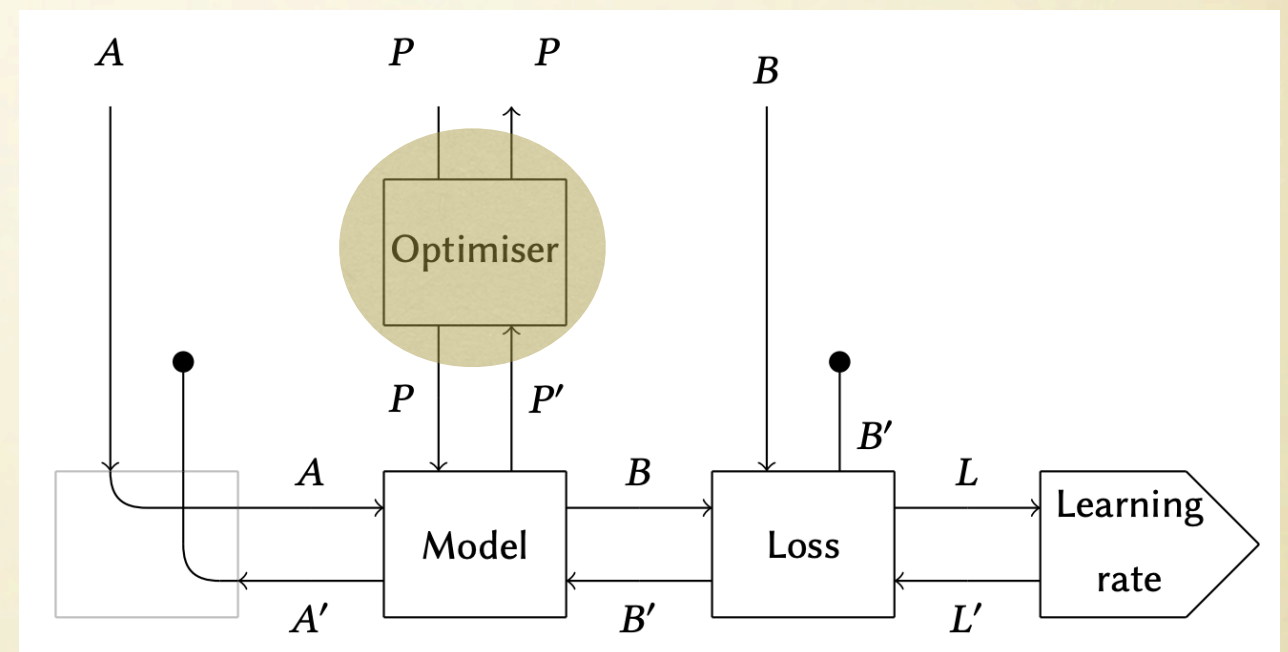
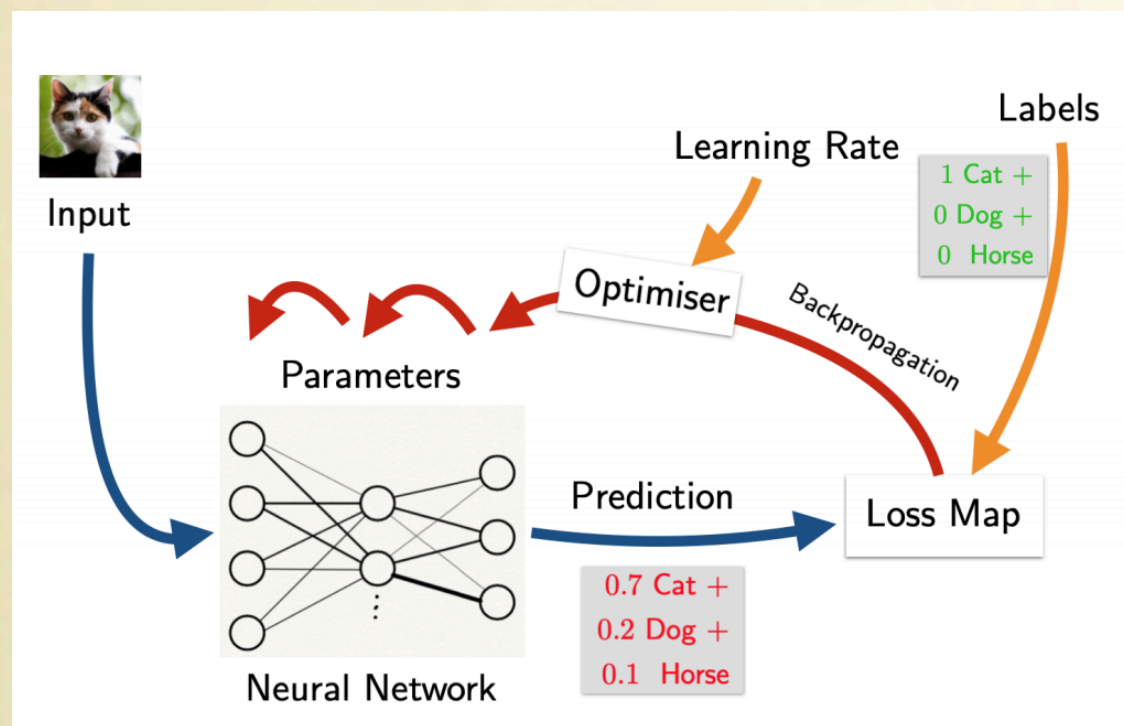
$$\text{nm} : (S \times P, S \times P) \rightarrow (P, P')$$

$$\text{get}_{\text{nm}} : (s, p) \rightarrow p - \gamma s$$

$$\text{put}_{\text{nm}} : (s, p, p') \mapsto (\gamma s + p', p - s)$$

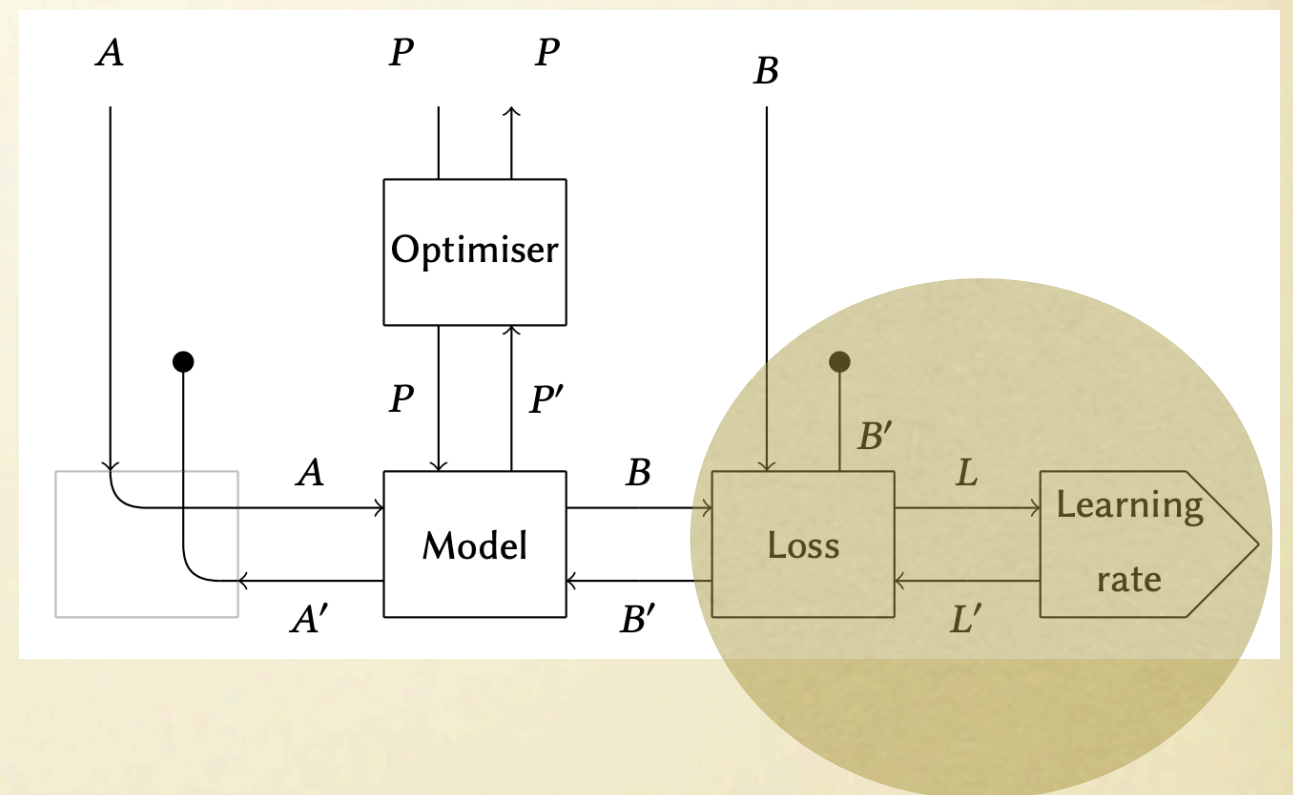
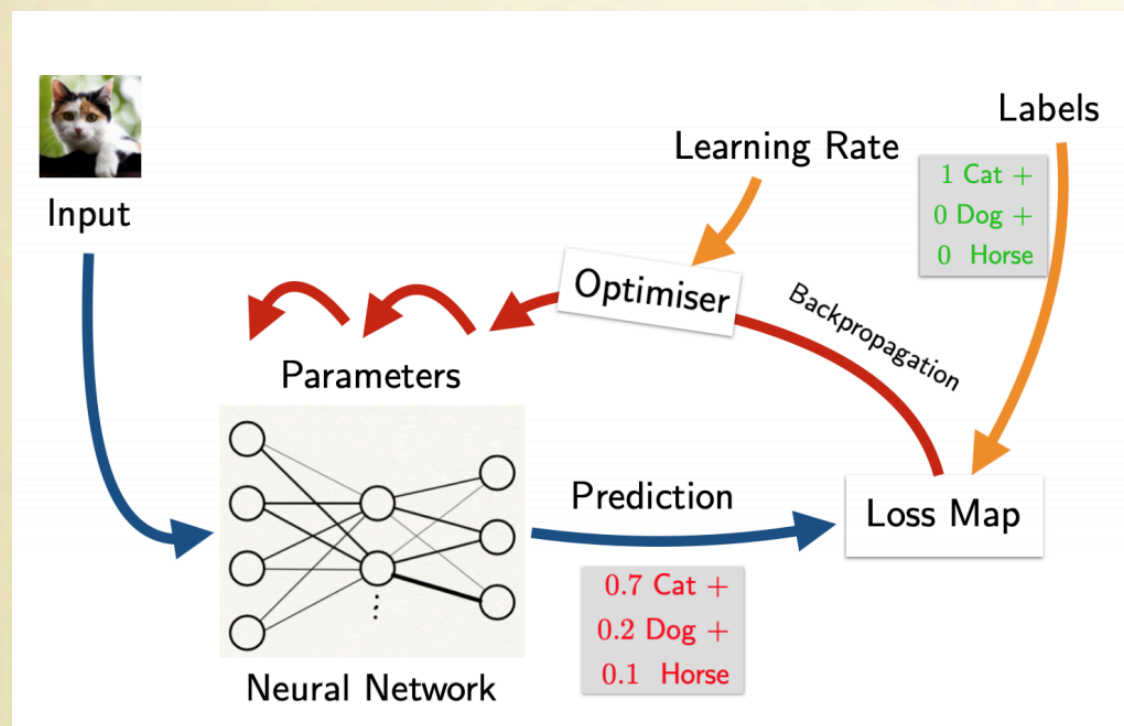
‘Stateful’ optimiser such as **Adagrad**, **Adam**, **Nesterov**, **Momentum**, etc. can be all formalised uniformly as reparametrisations.

# Where we are, so far

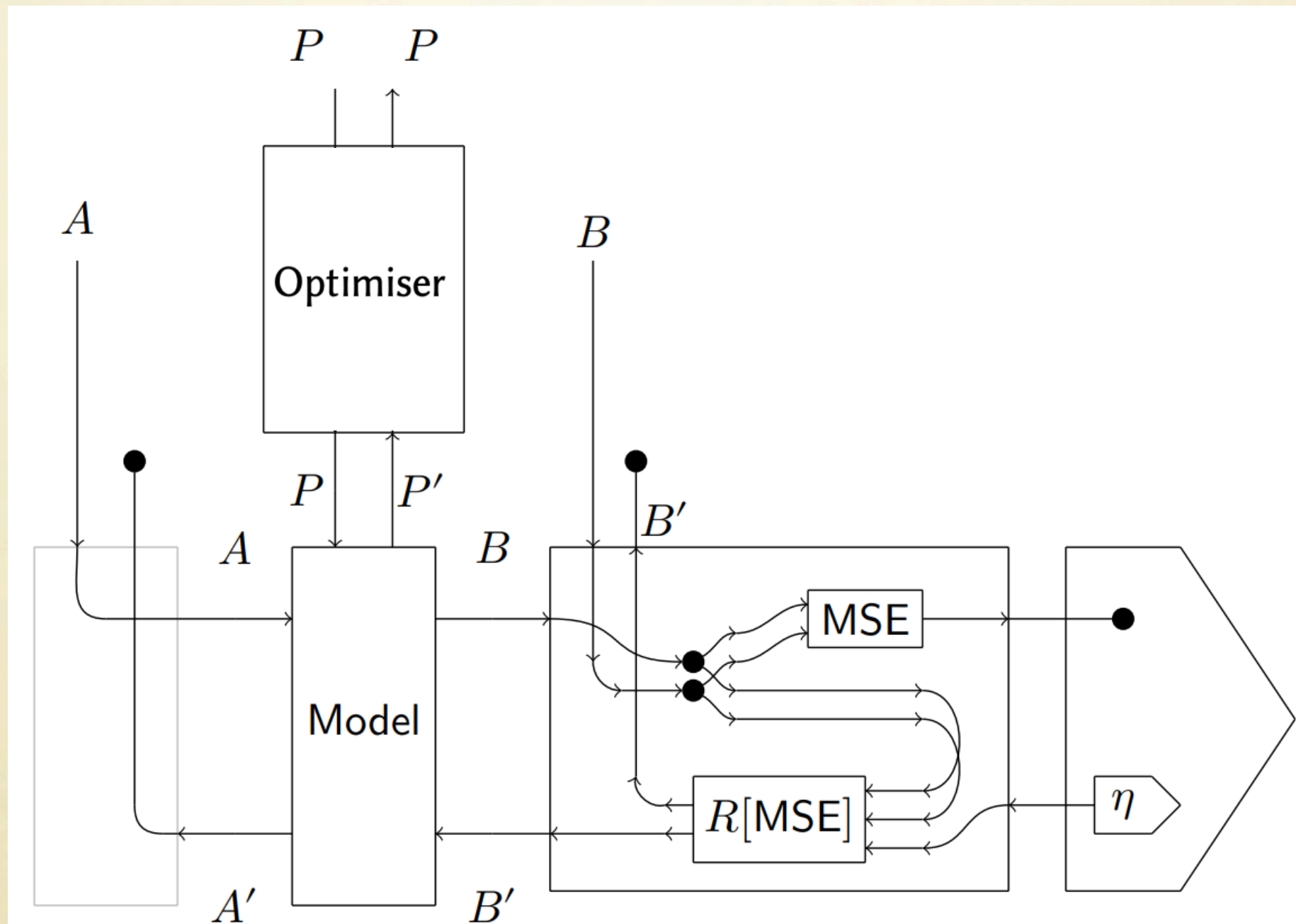




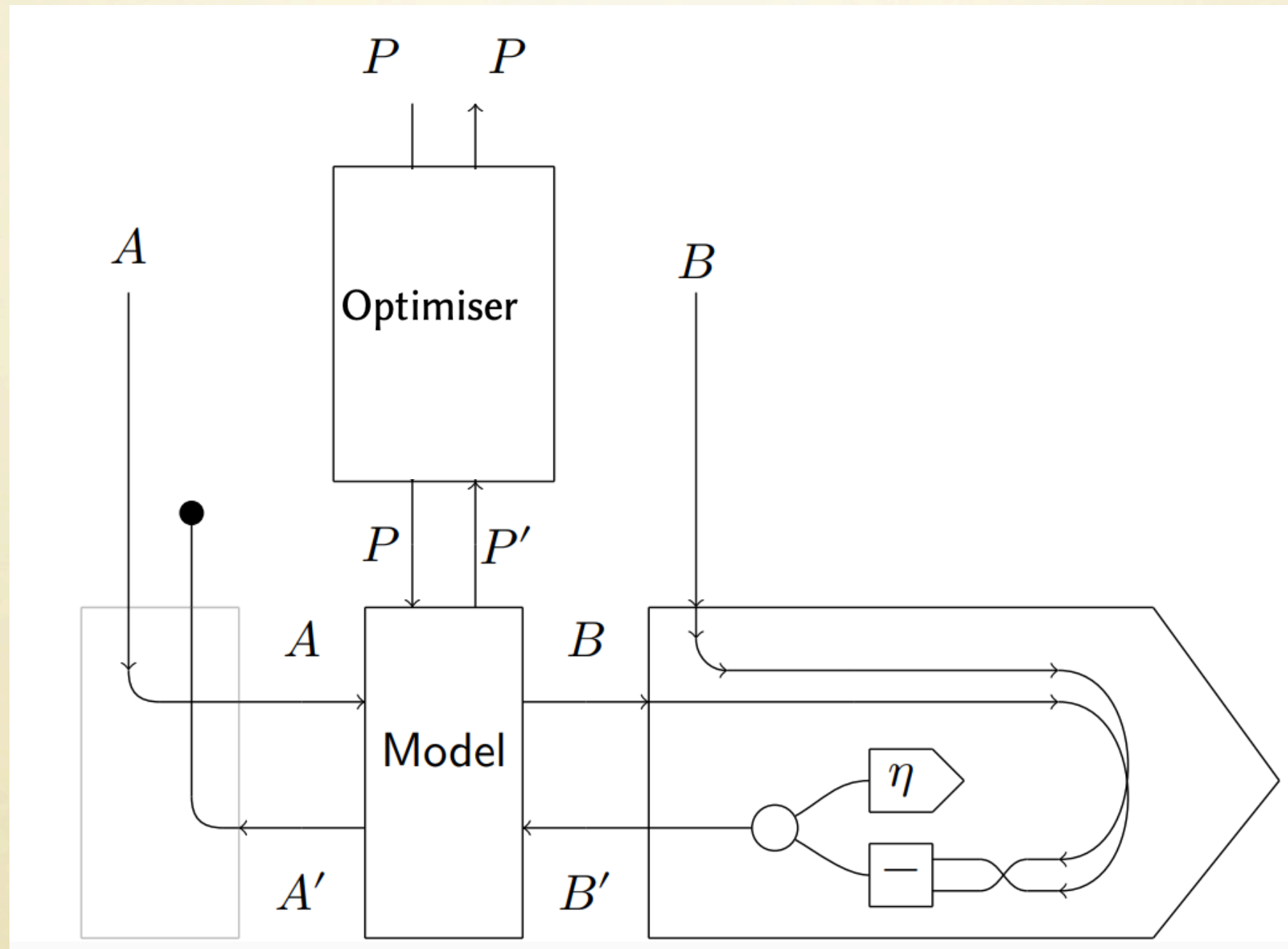
# Where we are, so far



# Loss map as parametric lens

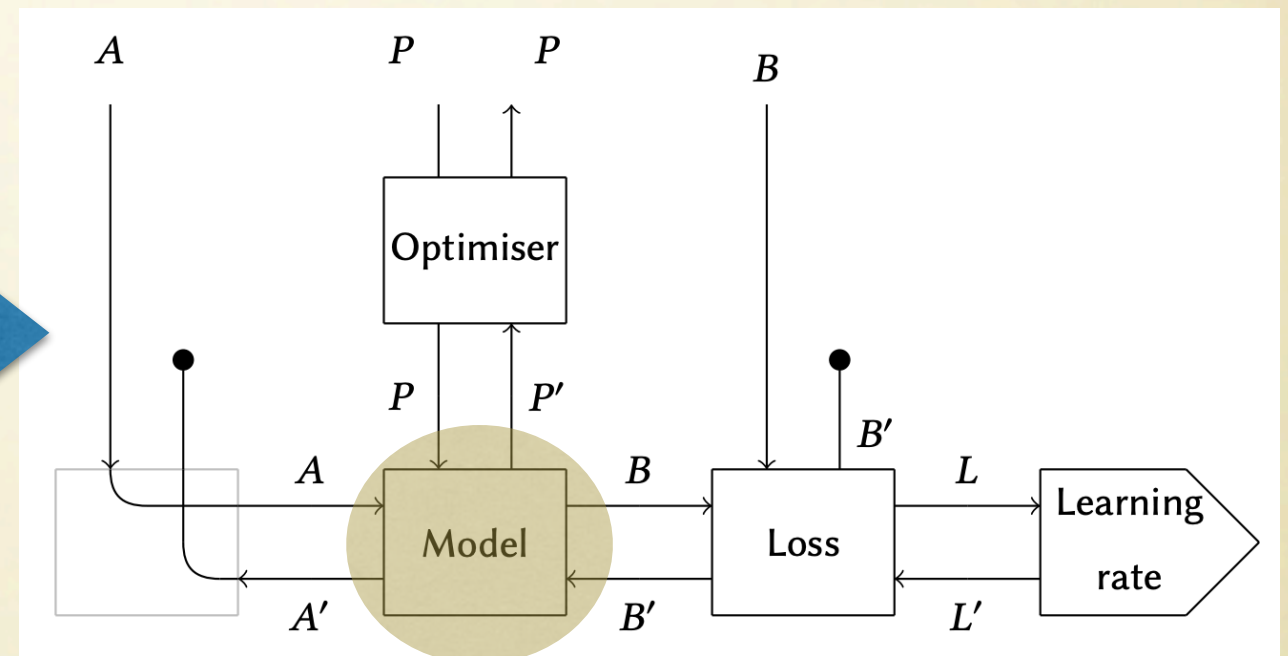
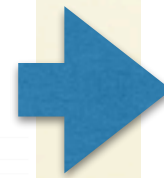
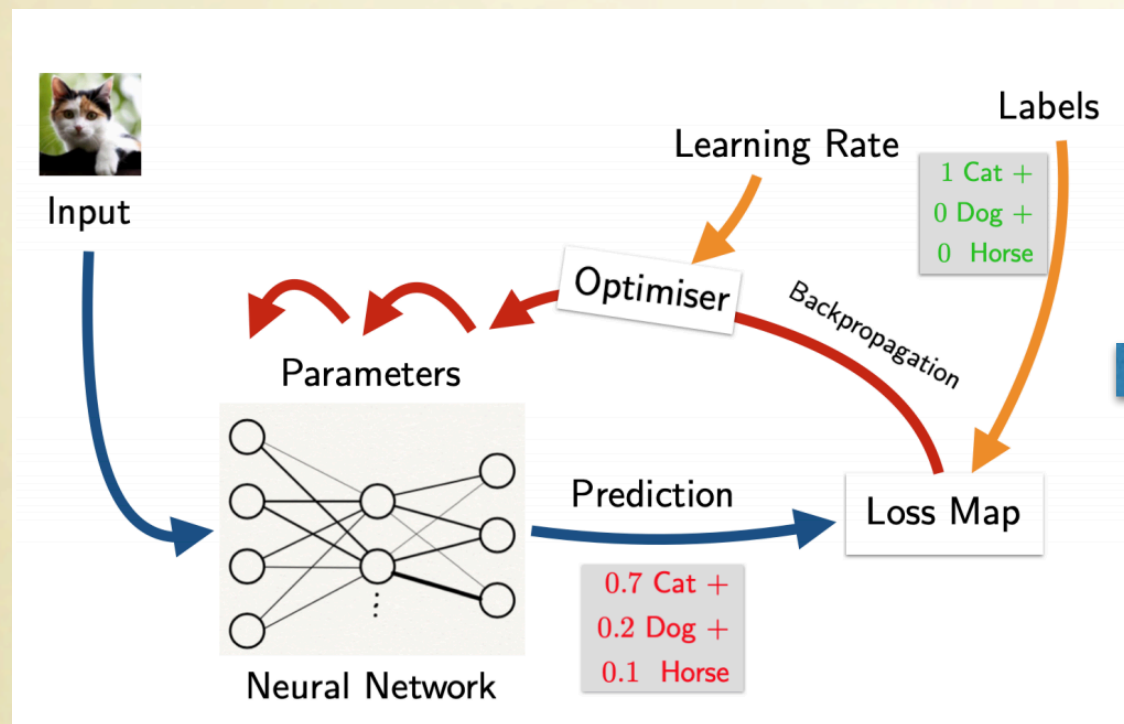


# Loss map as parametric lens

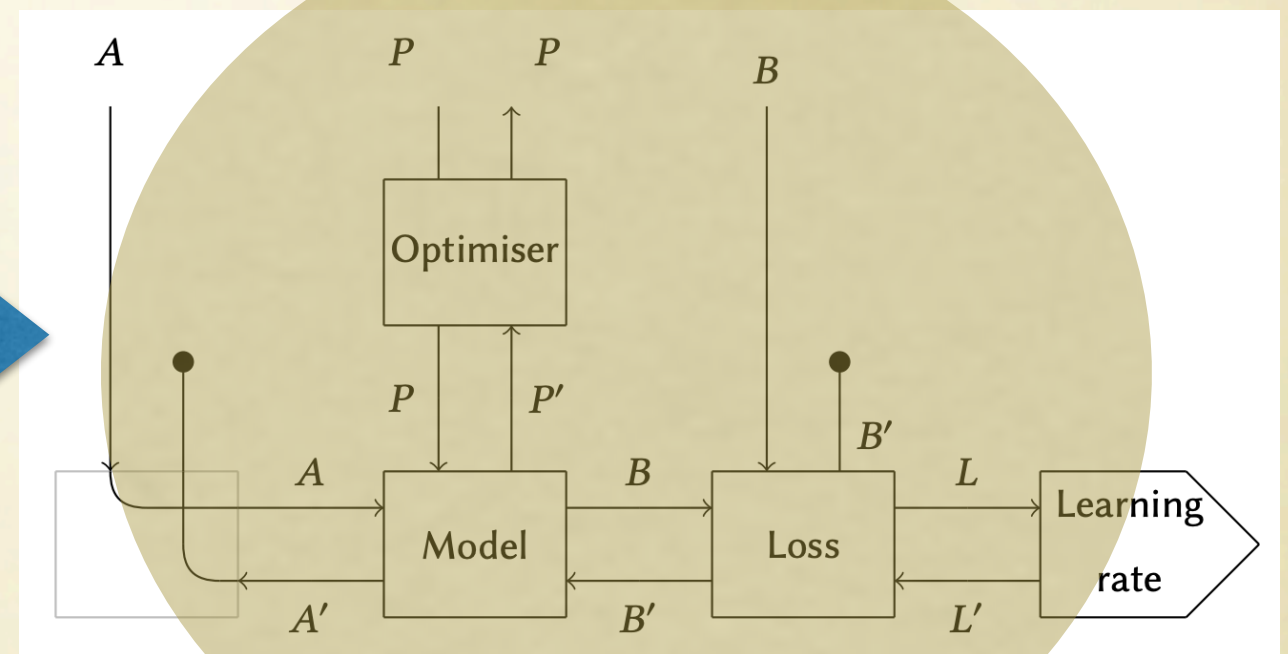
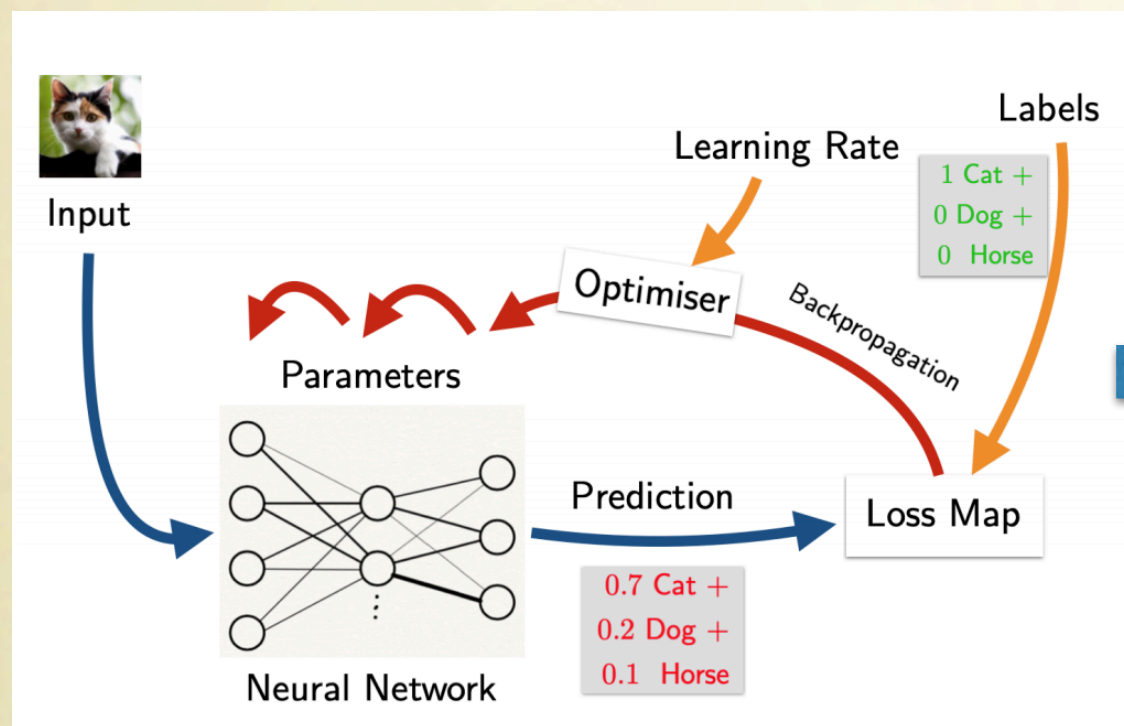




# Where we are, so far



# Where we are, so far



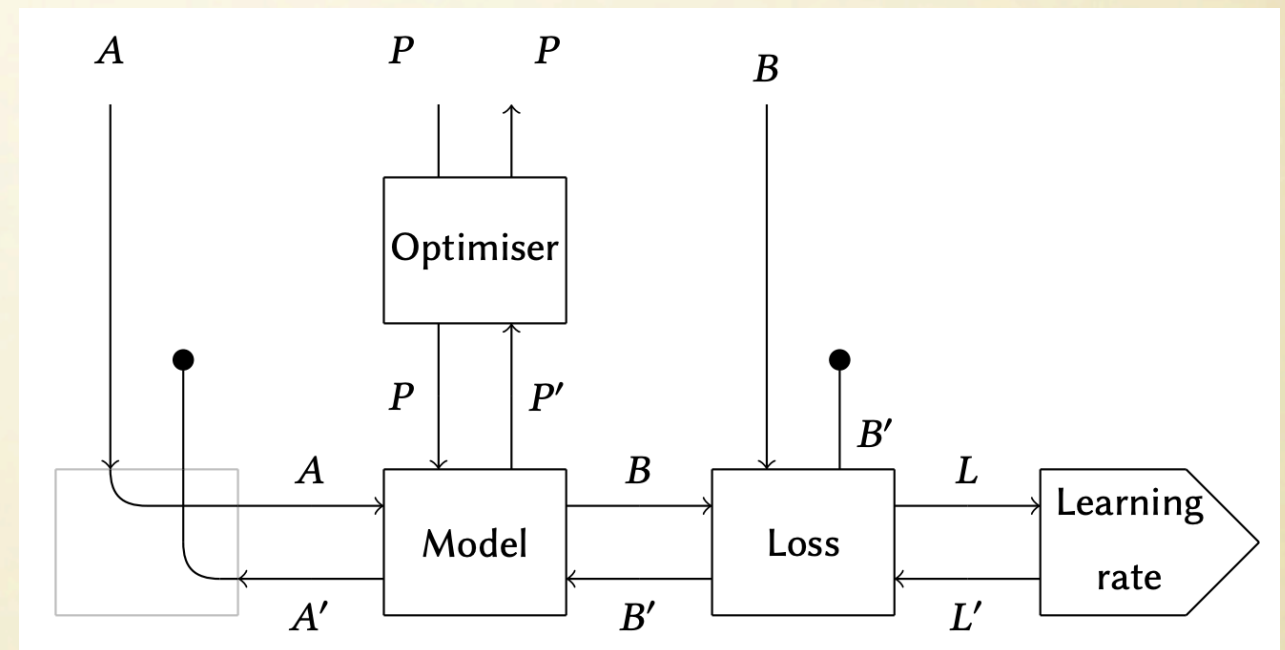
# Putting it all together

A parametric lens of type

$$(1, 1) \rightarrow (1, 1)$$

A lens of type

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





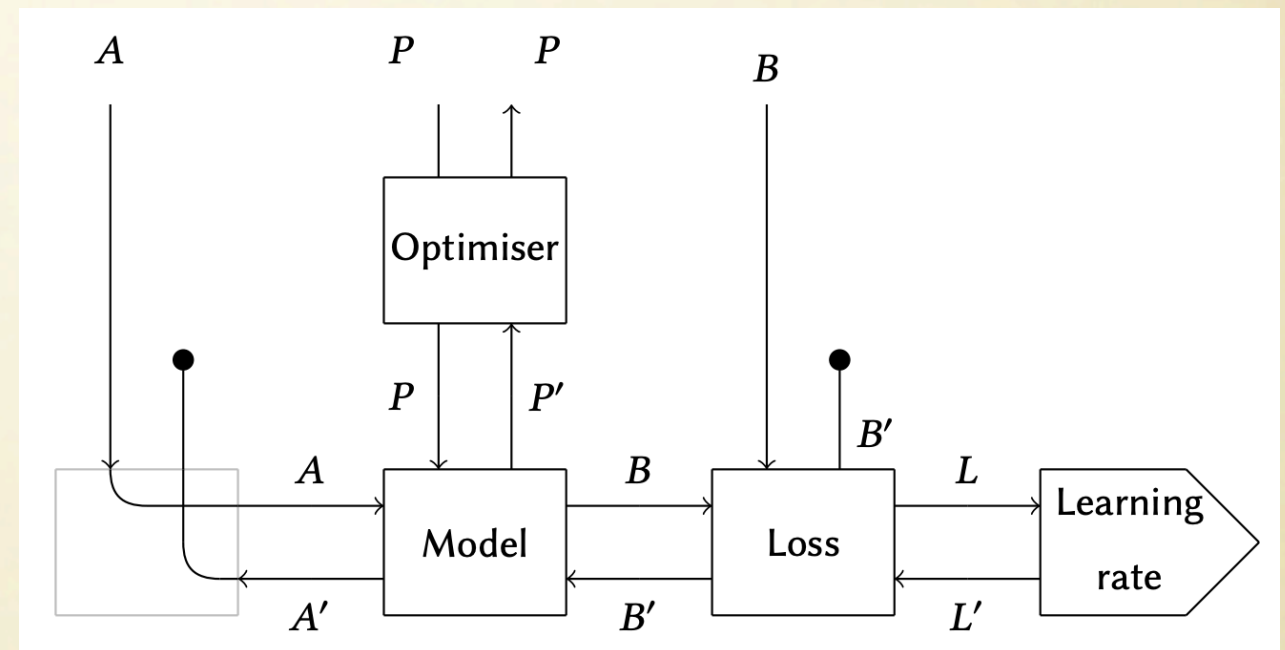
# Putting it all together

A parametric lens of type

$$(1, 1) \rightarrow (1, 1)$$

A lens of type

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

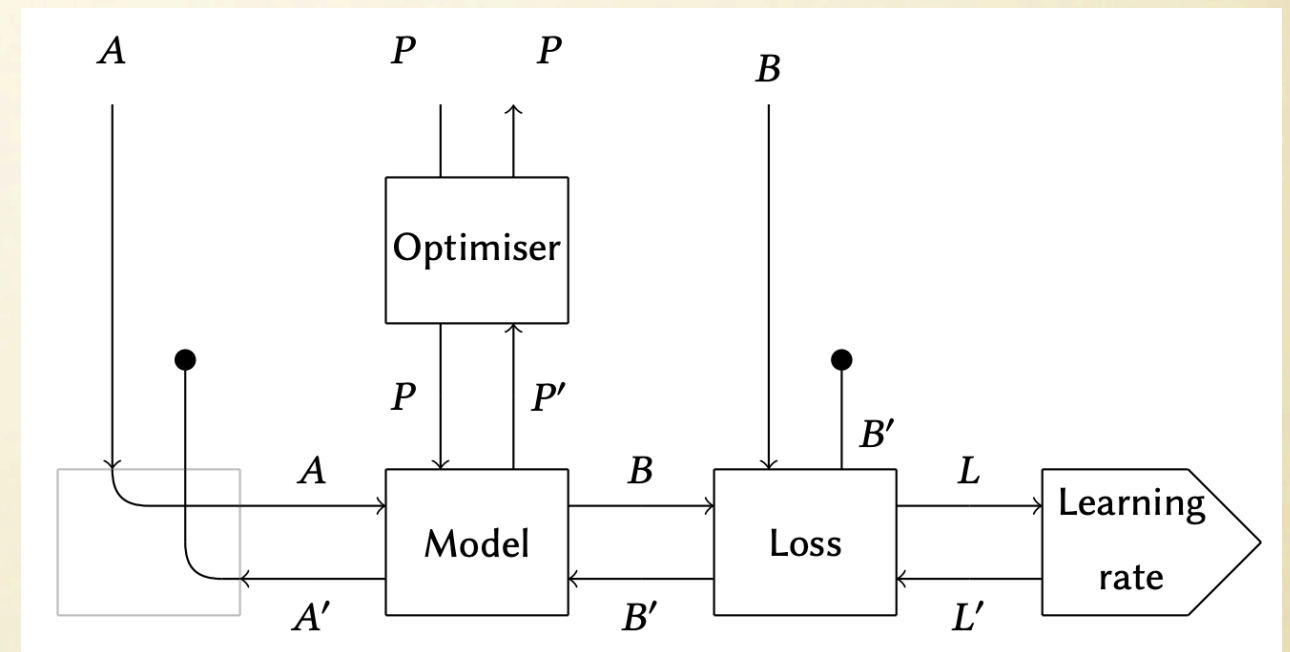


put  $A \times P \times B \rightarrow A' \times P' \times B'$

put  $(a, p, b_t) = (a', p', b'_t)$

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

# Learning as iterative process

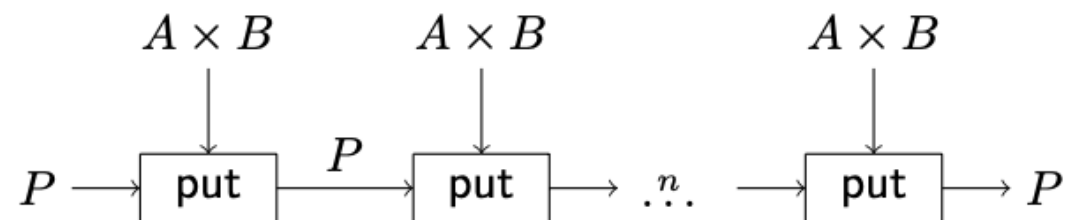


put  $A \times P \times B \rightarrow A' \times P' \times B'$

put  $(a, p, b_t) = (a', p', b'_t)$

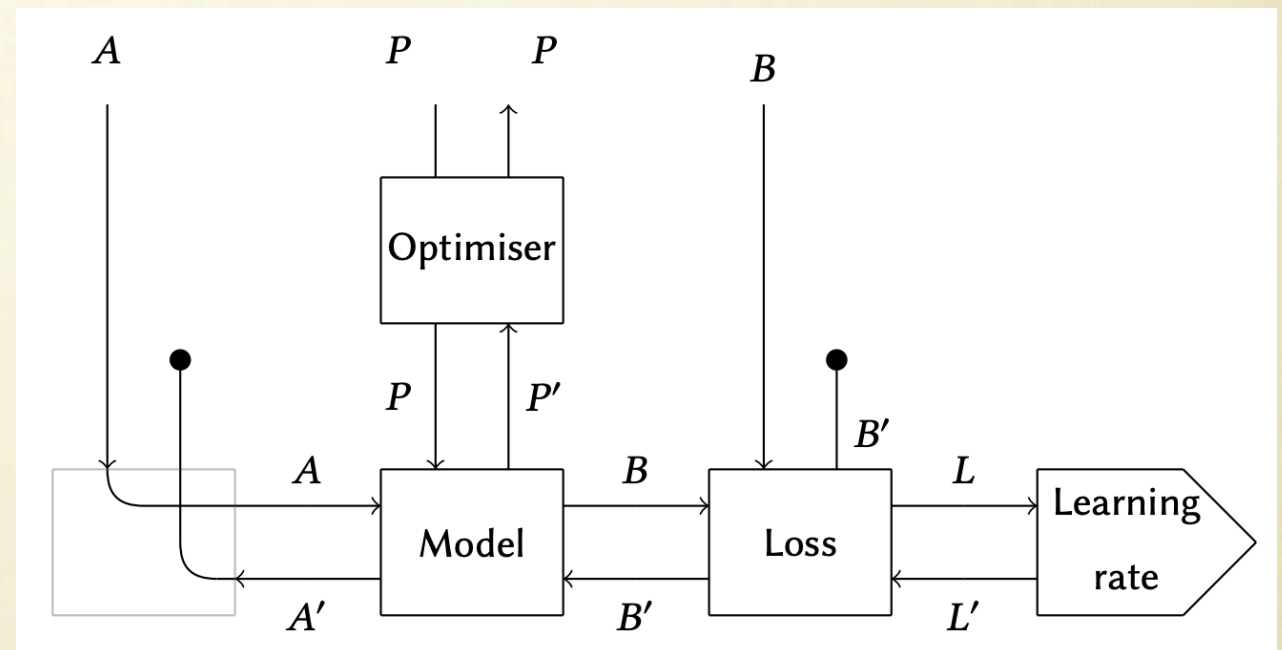
$b_p = f(p, a)$        $(b'_t, b'_p) = R[\text{loss}](b_t, b_p, \alpha(\text{loss}(b_t, b_p)))$        $(p', a') = R[f](p, a, b'_p)$

# Learning as iterative process



Dataset becomes parameters

Parameters become inputs/outputs



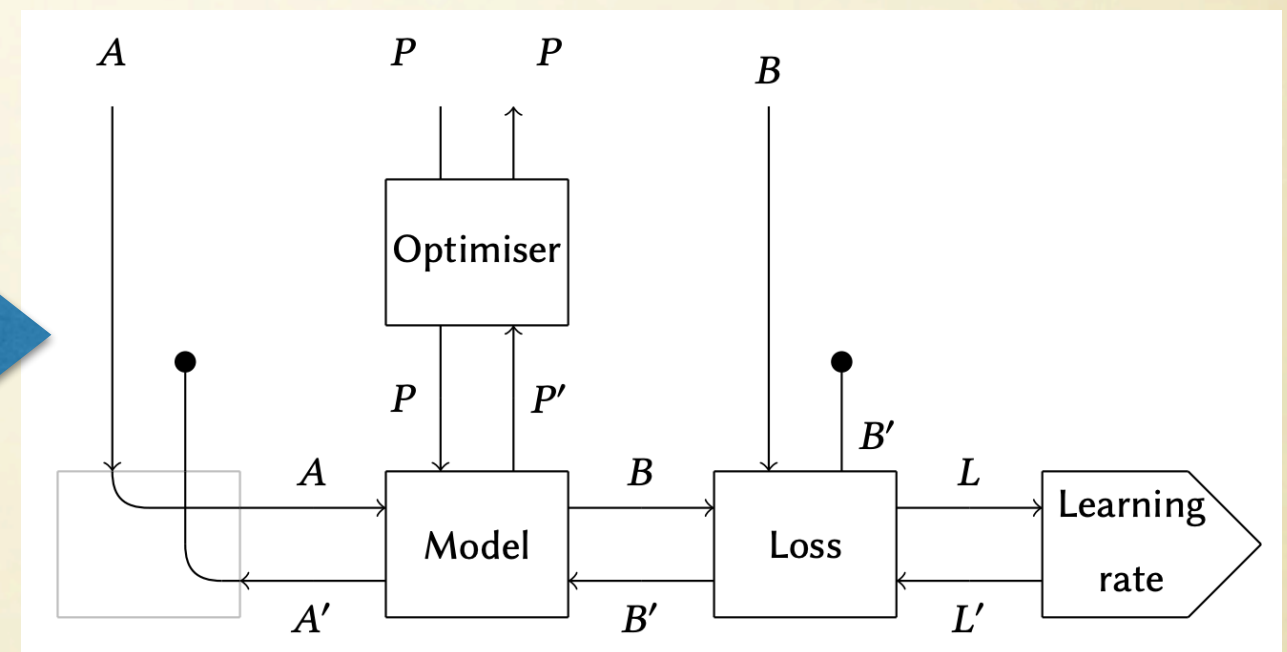
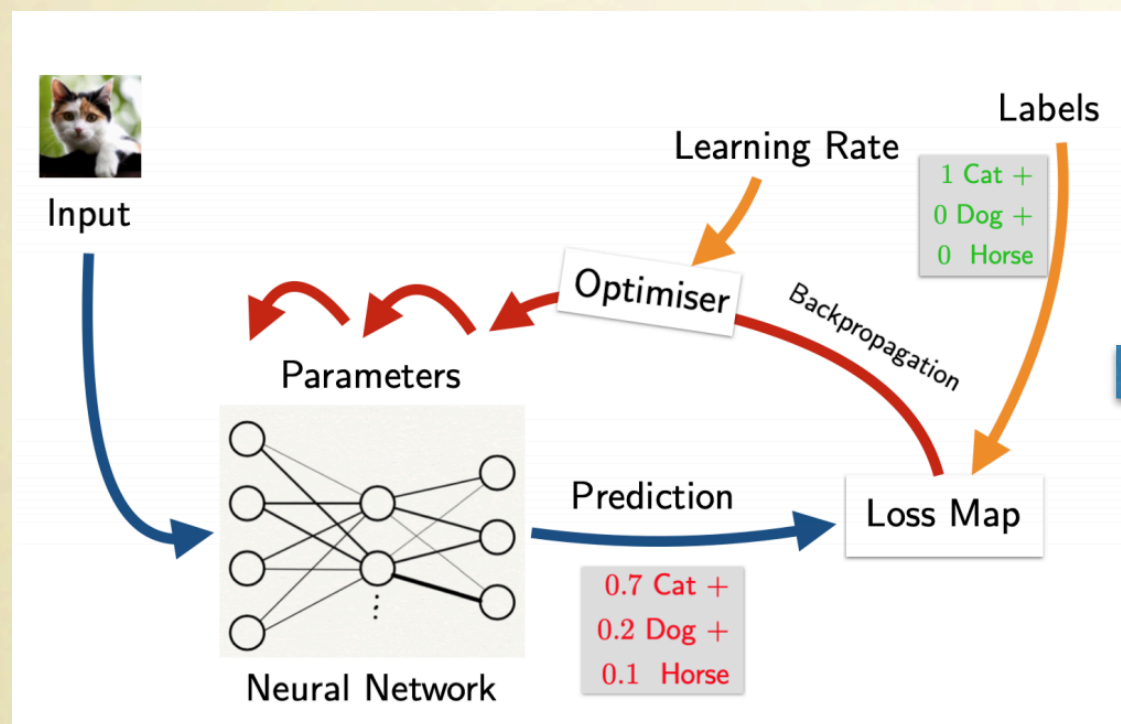
$$\text{put } A \times P \times B \rightarrow A' \times P' \times B'$$

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

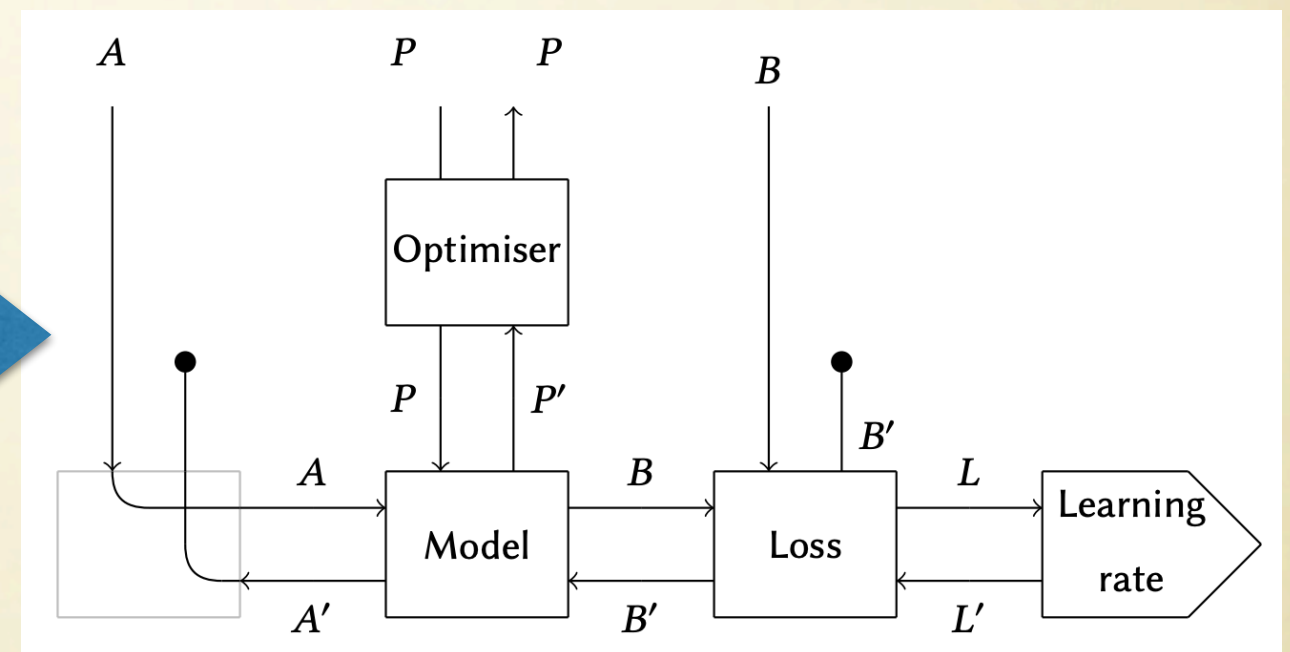
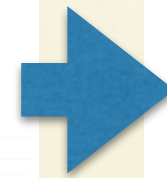
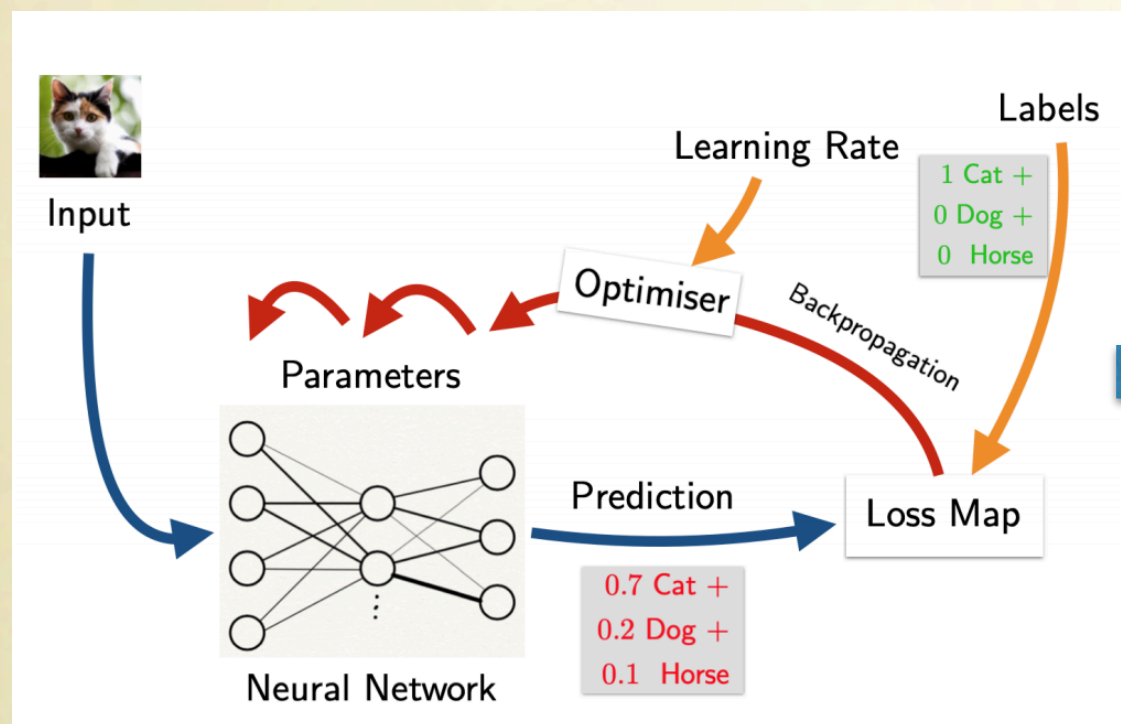
$$b_p = f(p, a) \quad (b'_t, b'_p) = R[\text{loss}](b_t, b_p, \alpha(\text{loss}(b_t, b_p))) \quad (p', a') = R[f](p, a, b'_p)$$



# Where we are, so far



# Where we are, so far

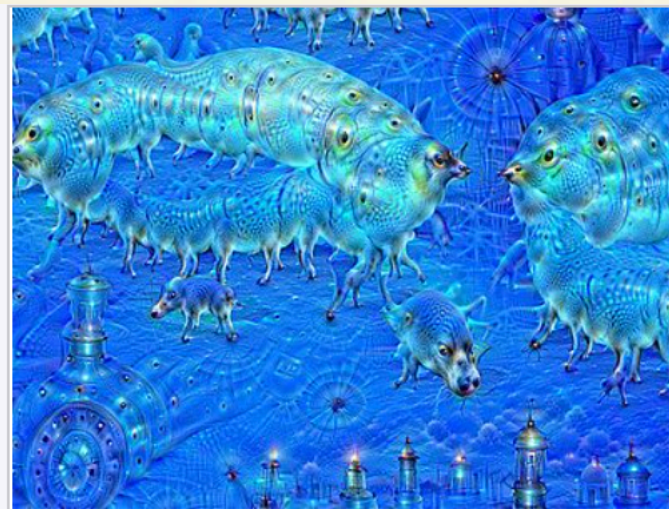




# Variations I: Deep Dreaming



Input  
Image

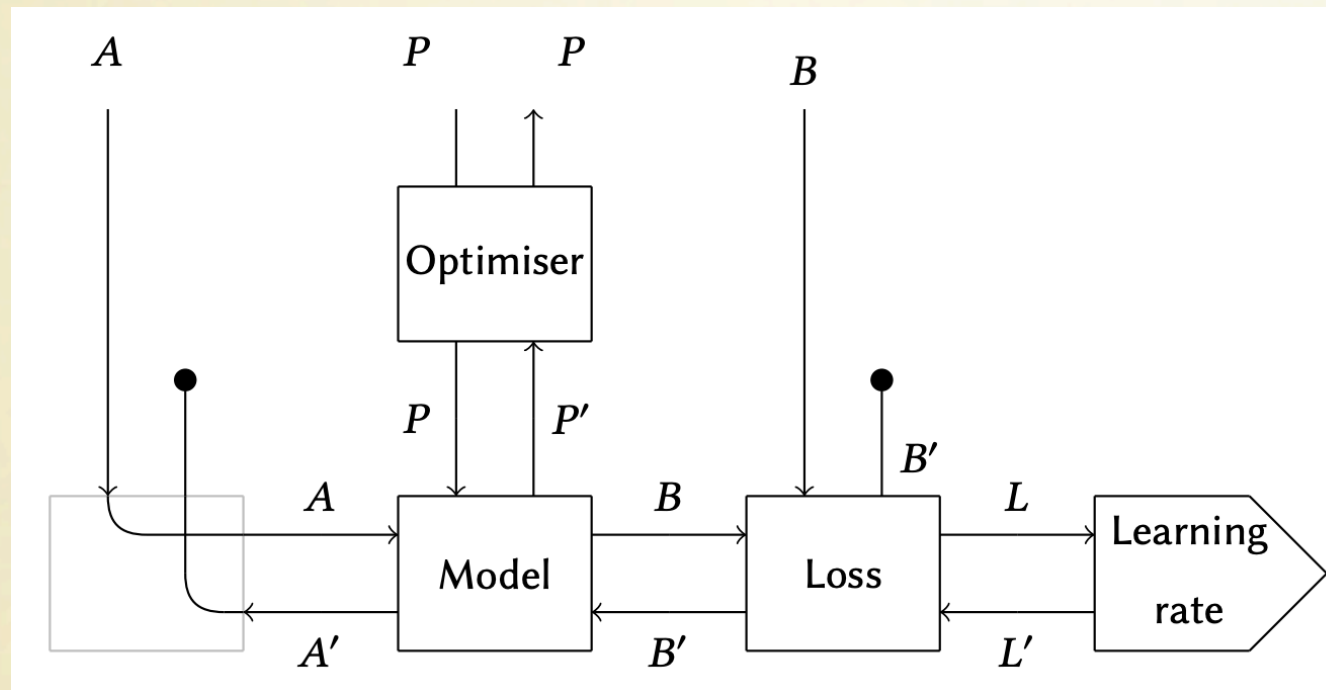


``Over-  
processed"  
Image

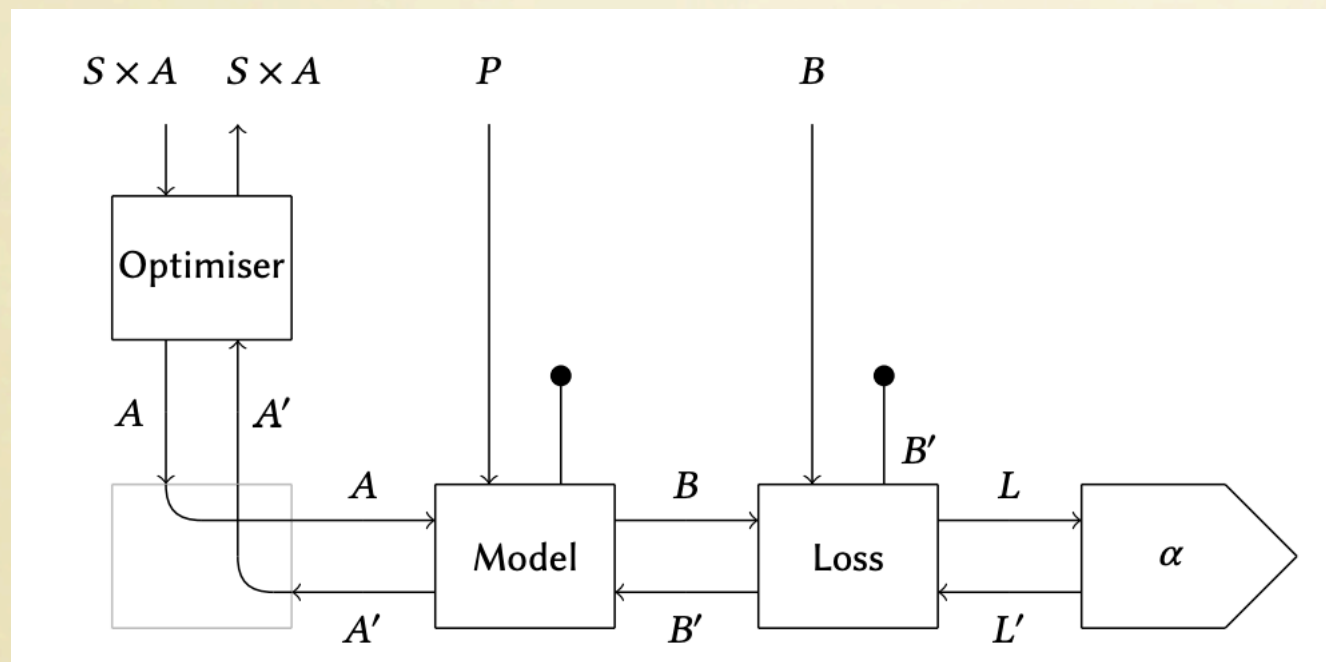




# Variations I: Deep Dreaming

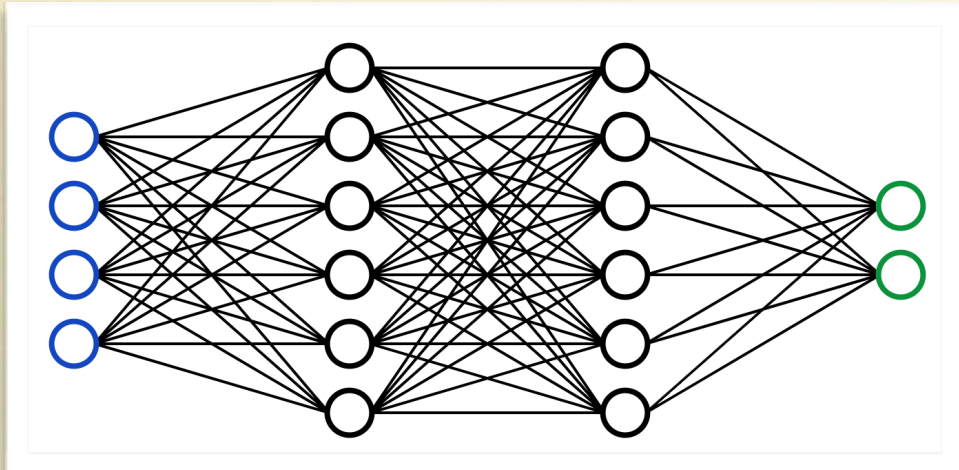


Parameter  
Learning

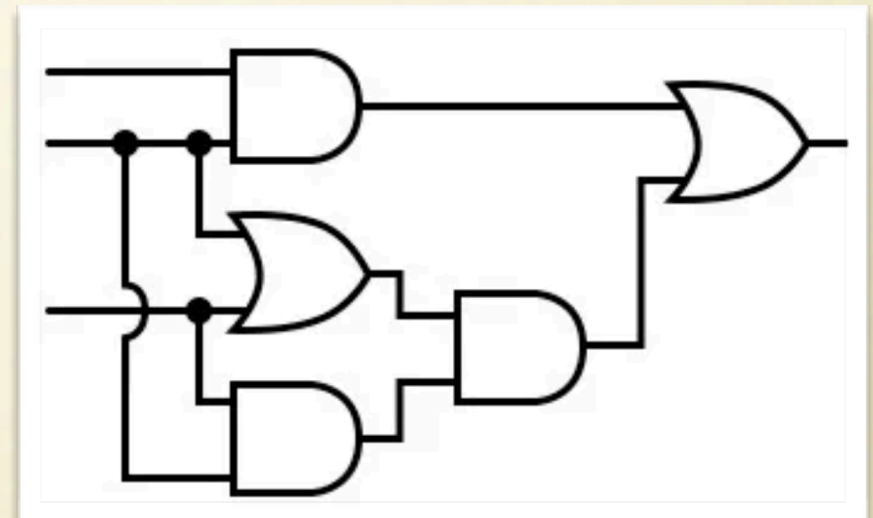


Deep  
dreaming

# Variations II: Learning Boolean circuits



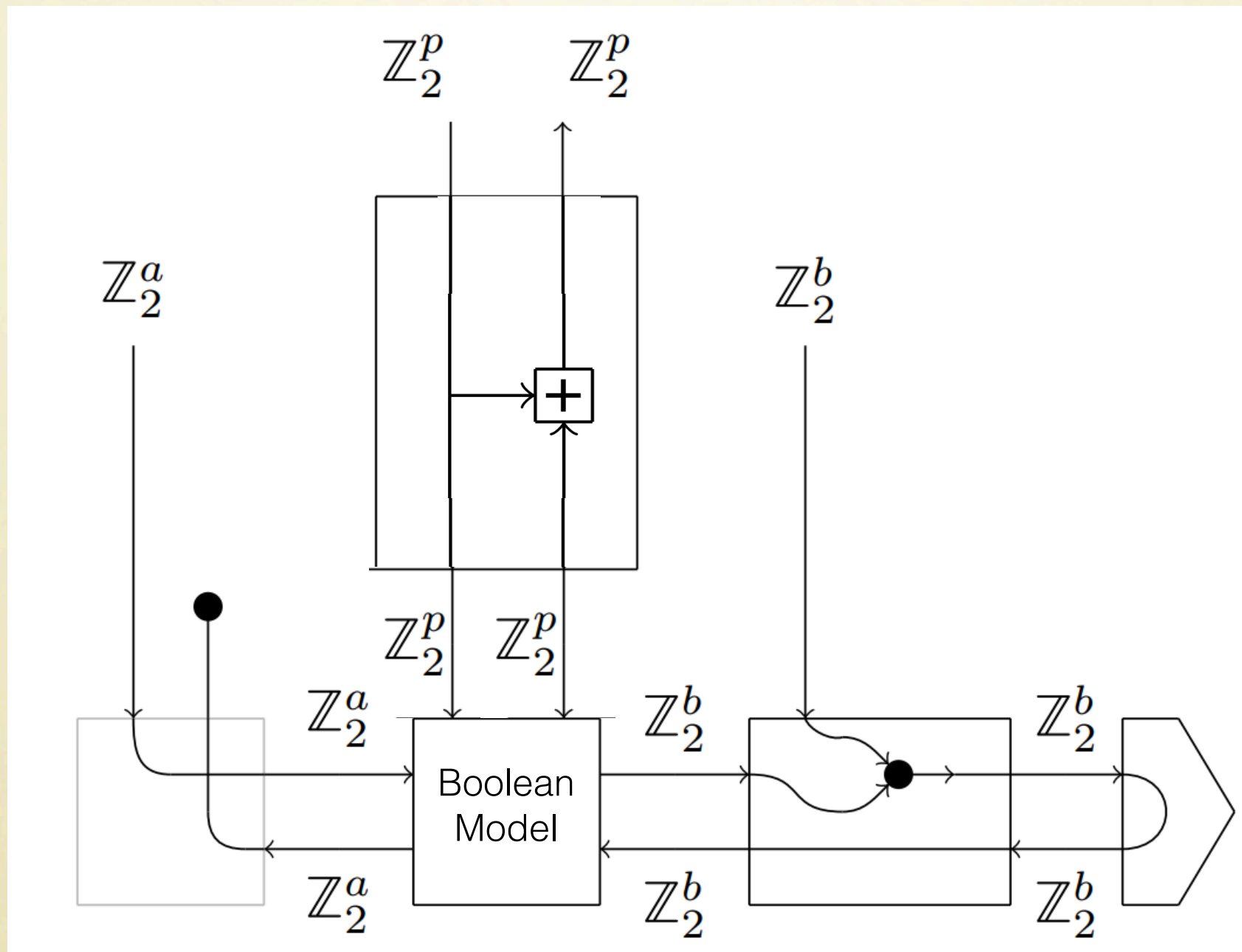
Neural networks



Boolean circuits

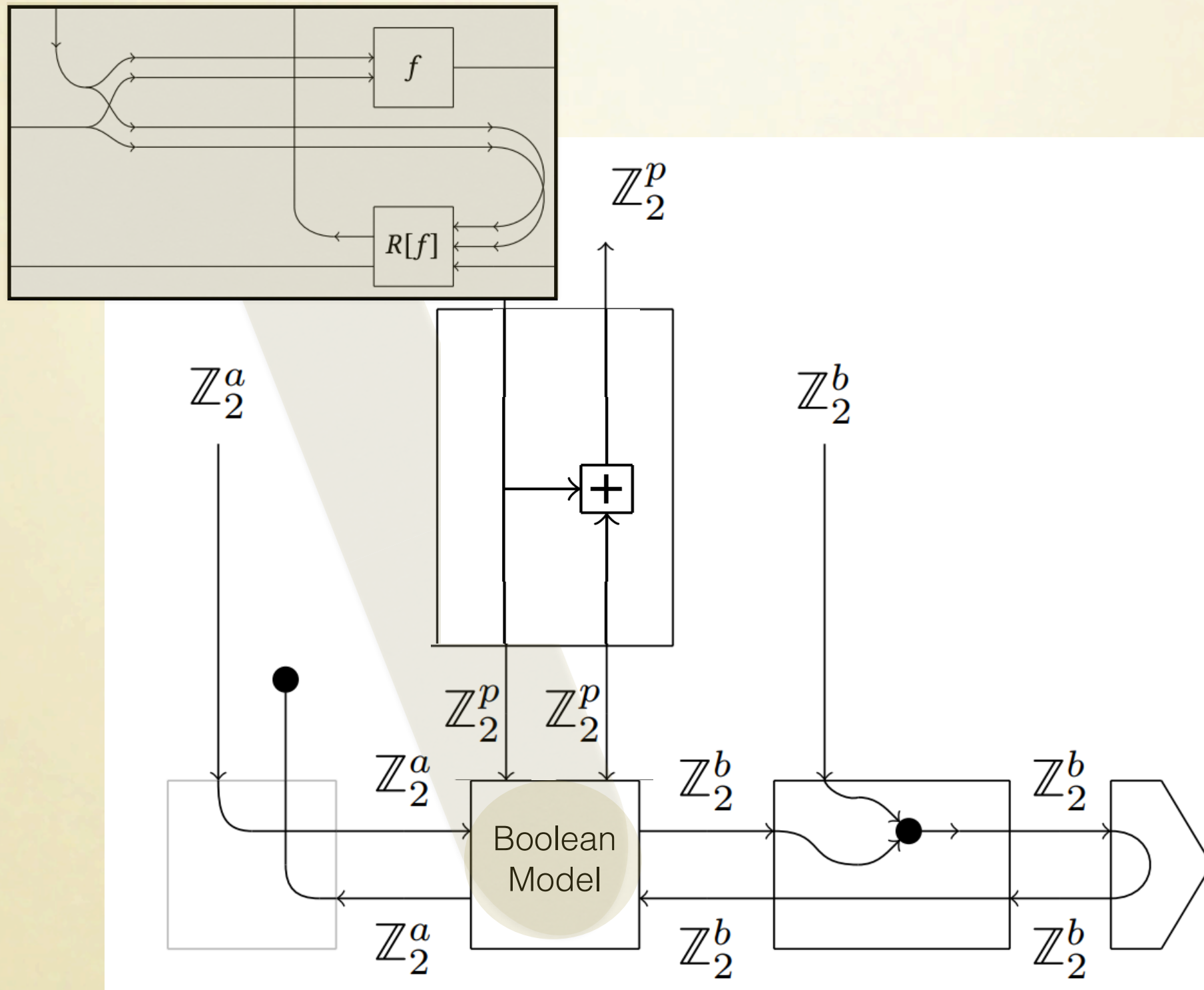
Instead of ***Smooth*** we work in ***Bool***

# Variations II: Learning Boolean circuits

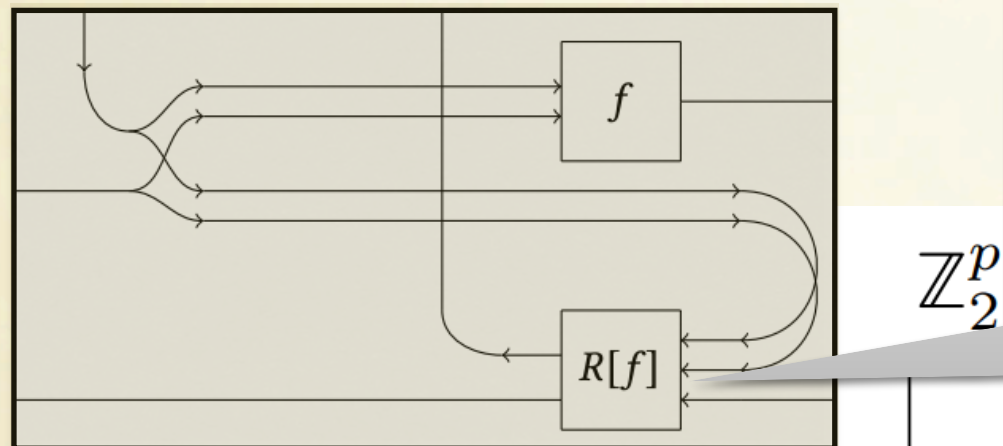




# Variations II: Learning Boolean circuits

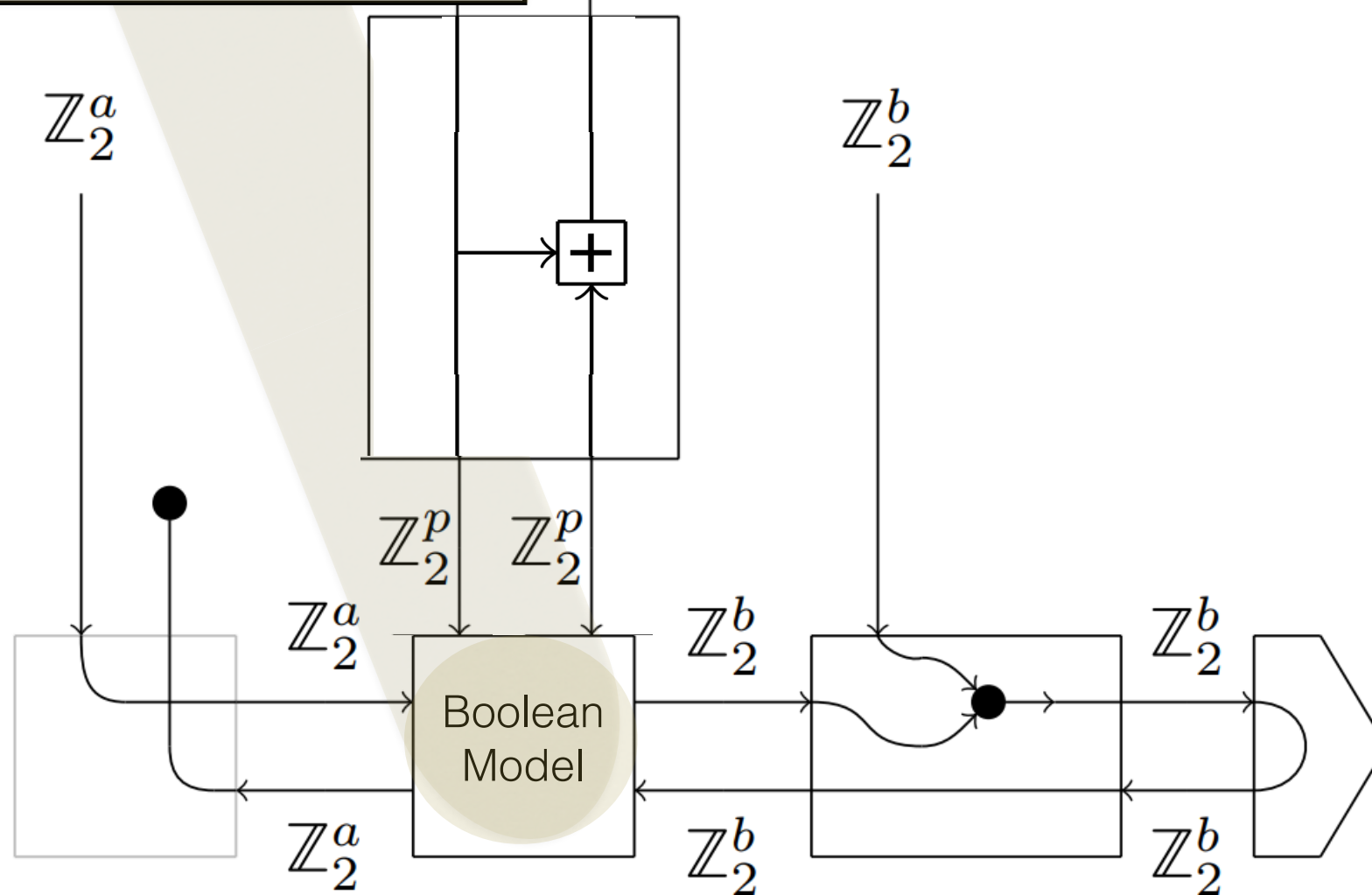


# Variations II: Learning Boolean circuits



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

$$\begin{aligned} \text{---}[f]\text{---}, \text{---}[d]\text{---} ::= & \text{---}\bullet\text{---} \mid \text{---}\bullet\text{---} \mid \text{---}\boxed{x}\text{---} \mid \text{---}\boxed{1}\text{---} \mid \text{---}\boxed{+}\text{---} \mid \text{---}\boxed{0}\text{---} \\ & \mid \text{---}\boxed{\phantom{x}}\text{---} \mid \text{---}\times\text{---} \mid \text{---}[f]\text{---}[d]\text{---} \mid \begin{array}{c} \text{---}[f]\text{---} \\ \text{---}[d]\text{---} \end{array} \end{aligned}$$



... and more

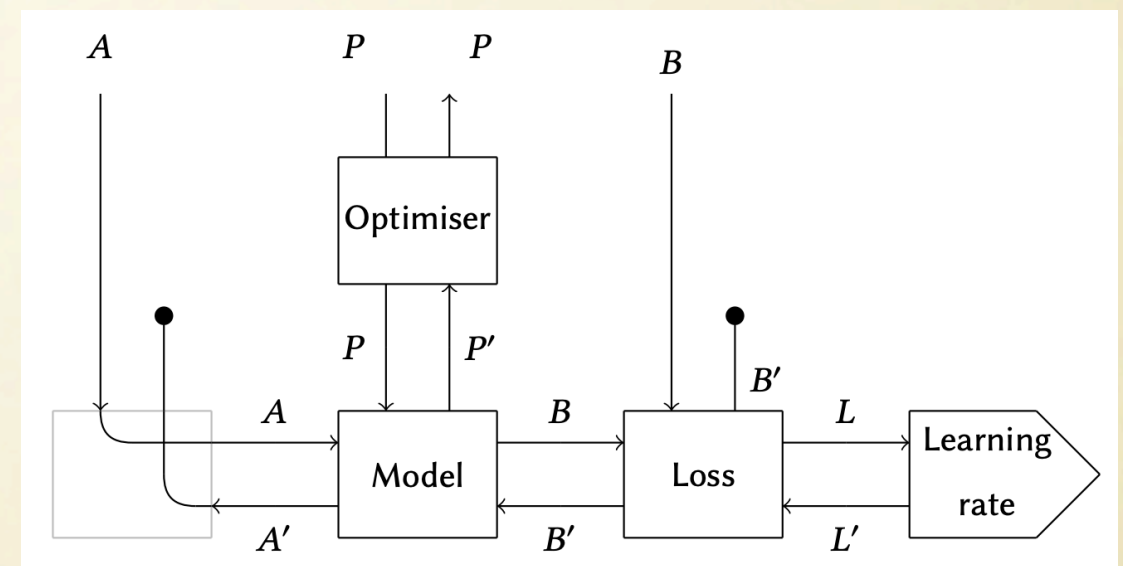
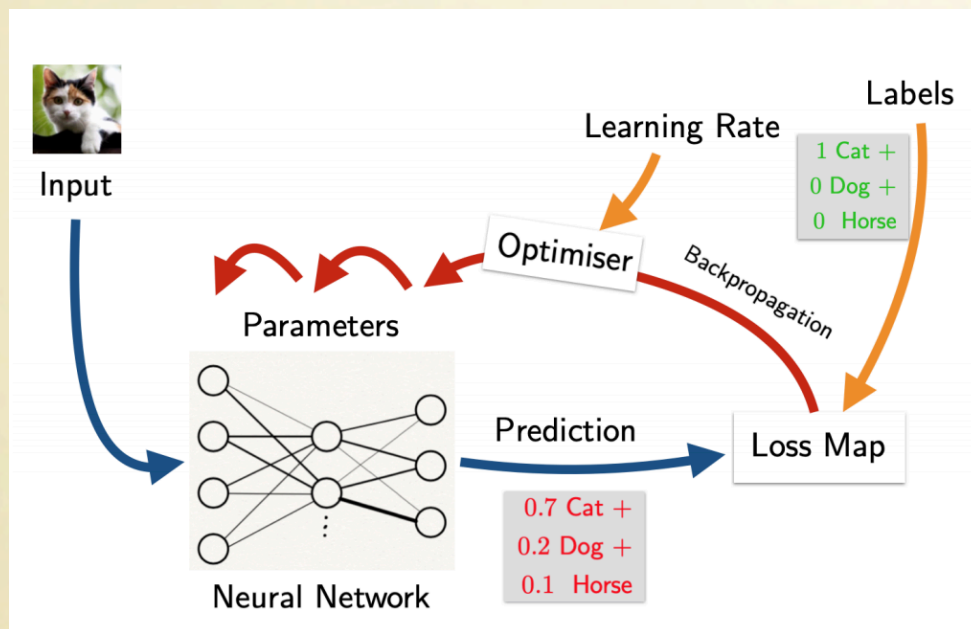
Generative Adversarial Networks

Convolutional Neural Networks

...



# Discussion



A lens-theoretic semantics for gradient-based learning.

# Discussion

## **Uniformity**

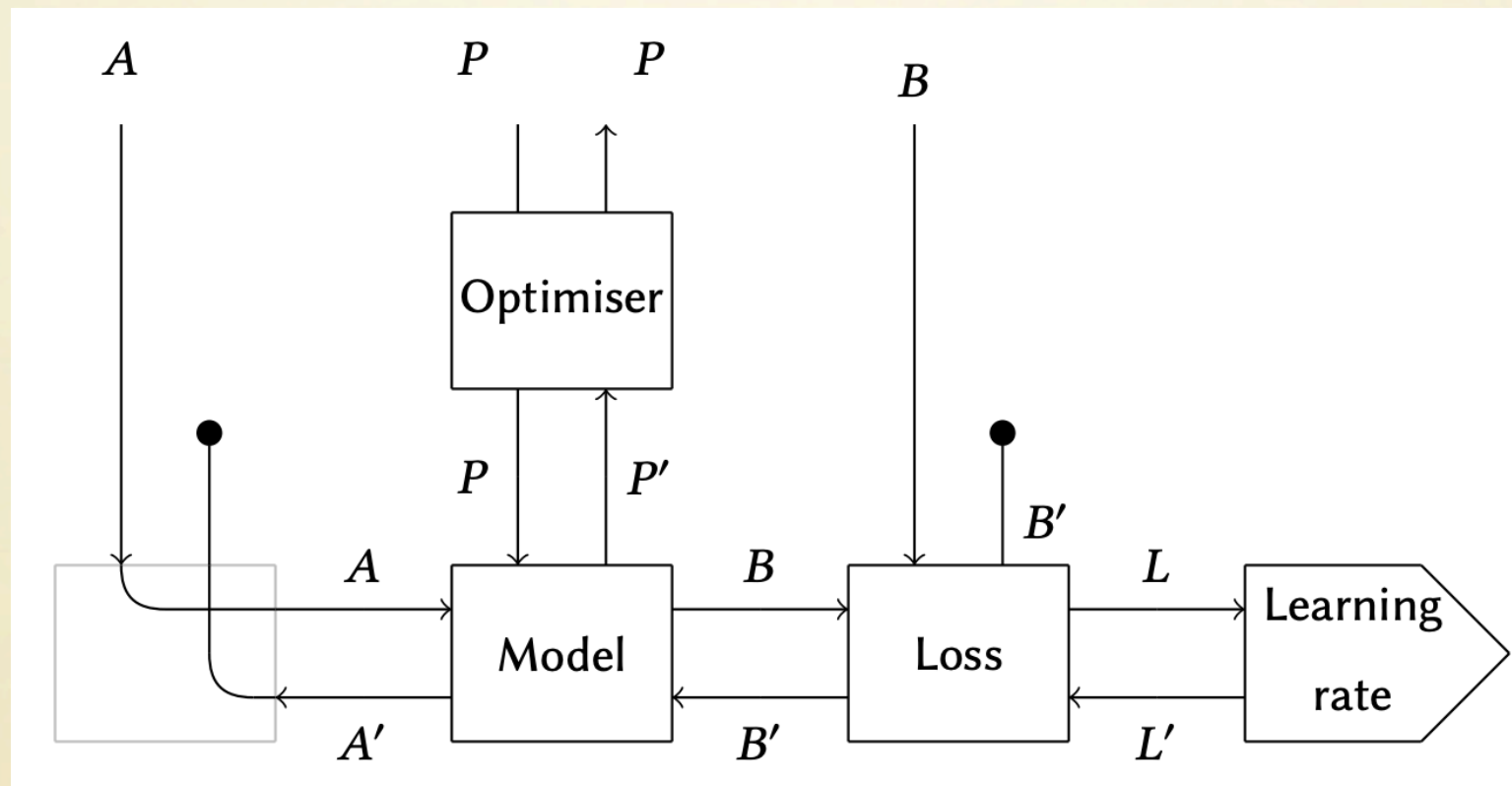
All components at work in the learning process are reduced to a single concept: parametric lenses.

## **Abstraction**

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

# Discussion

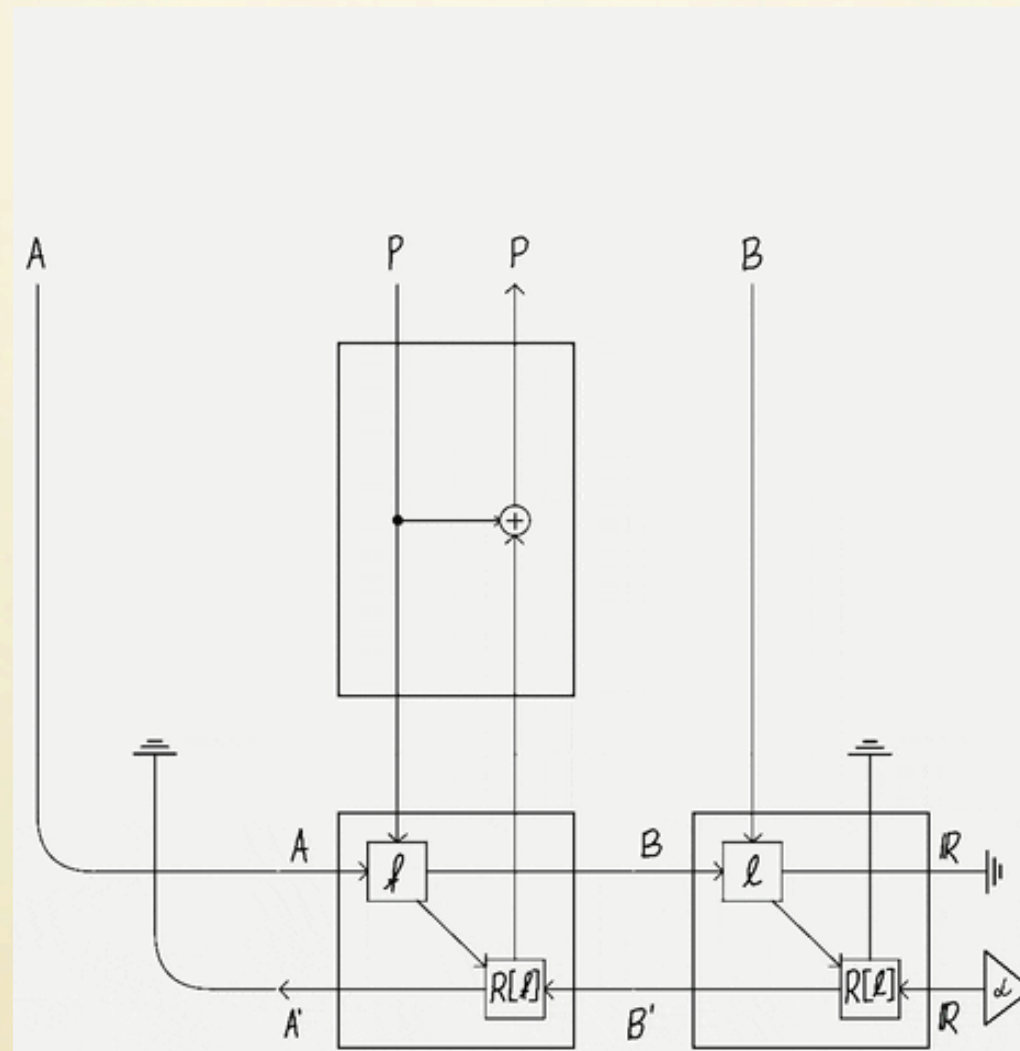
## Compositional reasoning





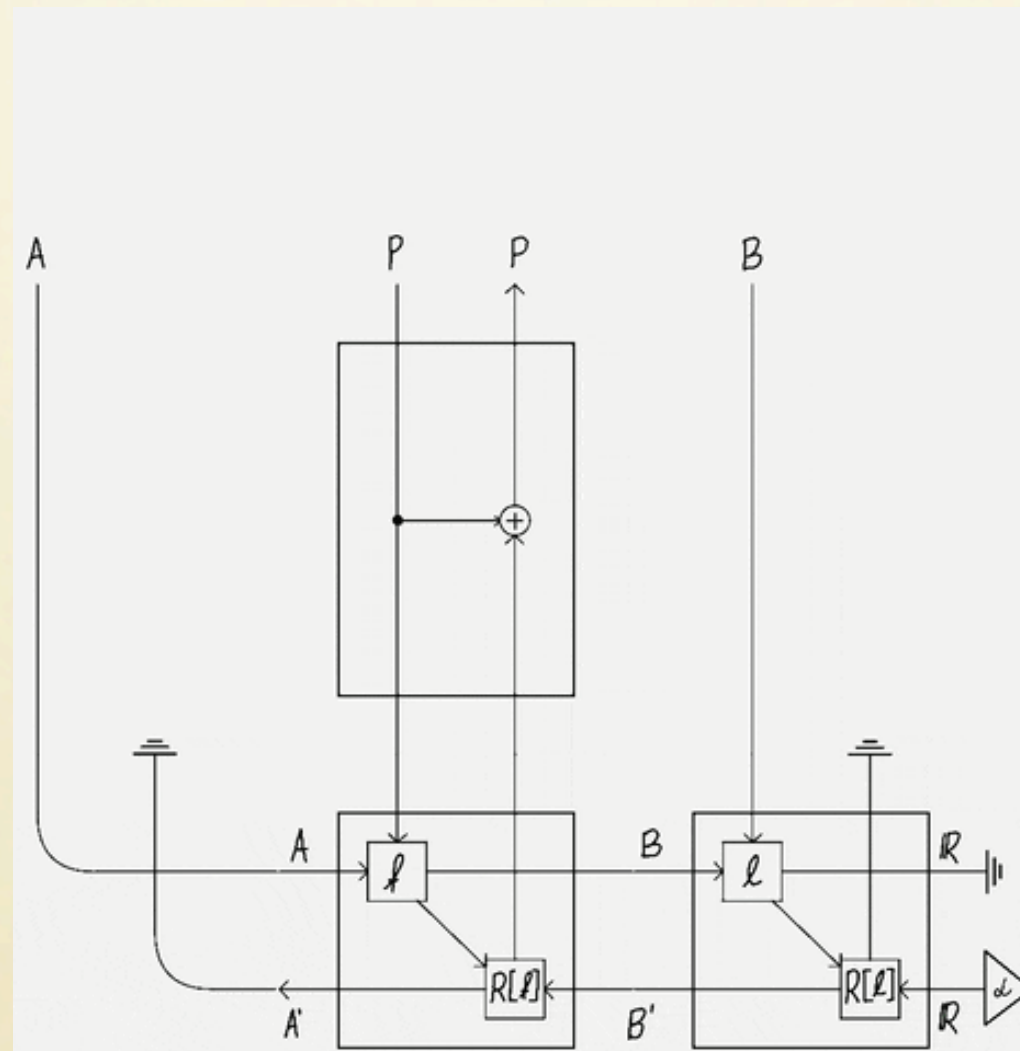
# Discussion

## Resource-sensitive Formalism



# Discussion

## Resource-sensitive Formalism



# Going Forward



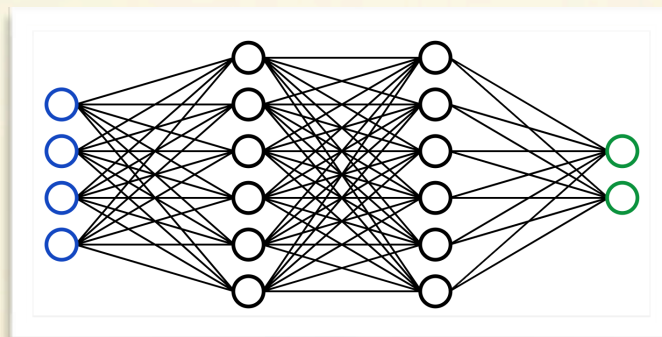
# Going Forward

Learning over simpler models

Performance



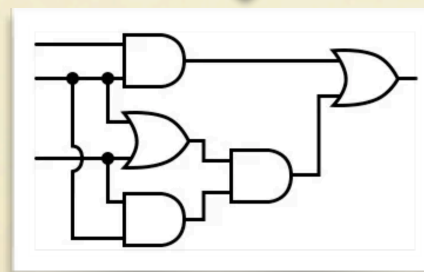
Accuracy



Performance



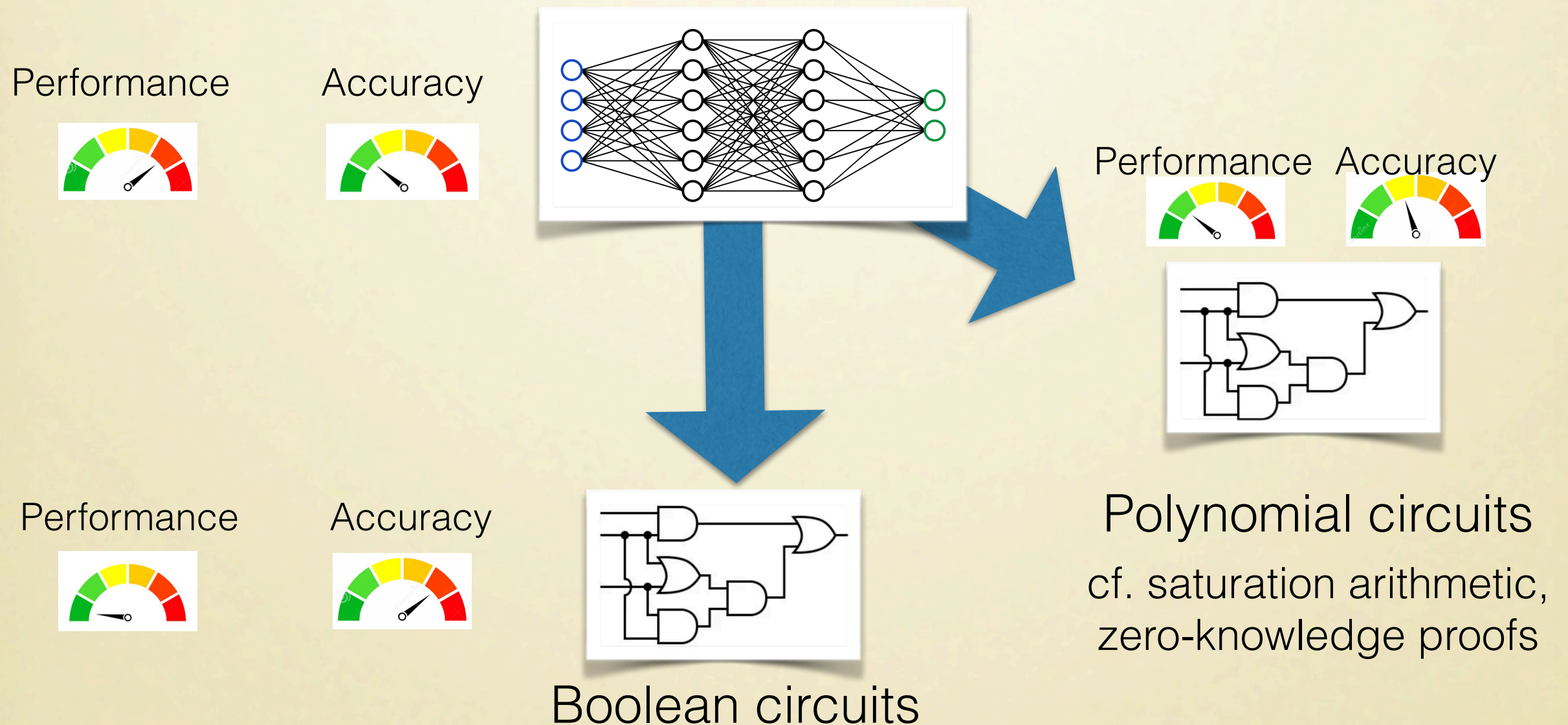
Accuracy



Boolean circuits

# Going Forward

Learning over simpler models





# 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
```

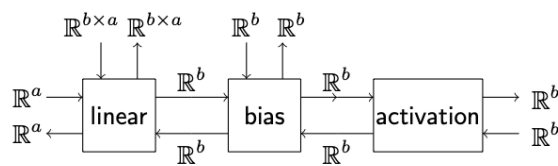
# Going Forward

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
```



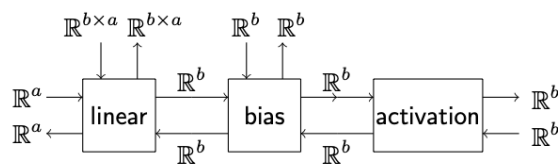
# Going Forward

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( $\epsilon$ )
```



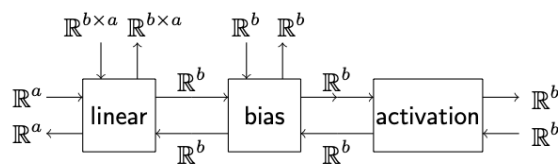
# Going Forward

Programming with lenses?

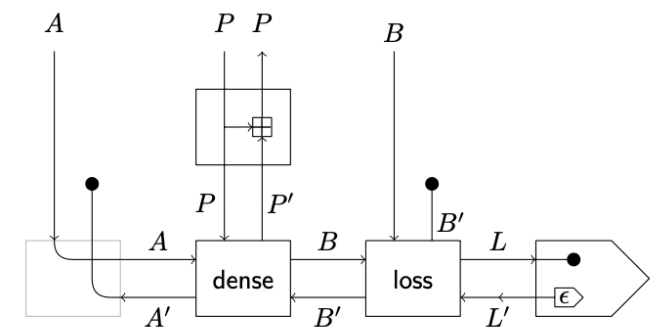
<https://github.com/statusfailed/numeric-optics-python/>

<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( $\epsilon$ )
```



# 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