Machine Learning and AI in experiments and theory

PhyNuBe4 Summer School
Aussois, France
(100% generated by a human)

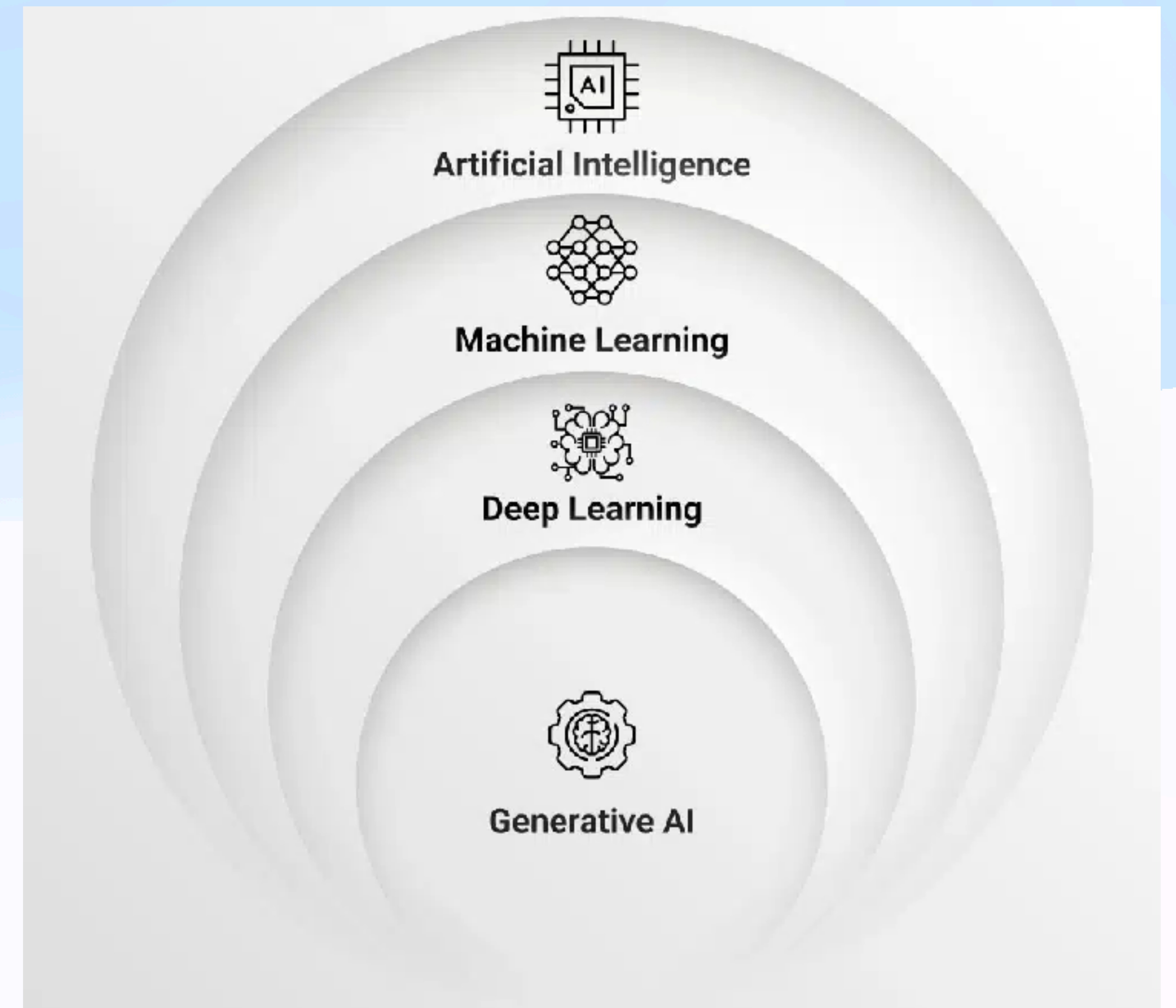
Antonin Vacheret

Learning outcomes

- This week you will :
  - Get a (brief) overview of Machine Learning and Deep Learning in Science
  - Learn about supervised learning and neural networks
  - Learn basics of pytorch implementation of a NN model and its training
  - Learn to use realistic nuclear physics datasets with NN
  - Understand how to implement and evaluate the most common ML tasks

What is Machine Learning and Artificial Intelligence ?

- Machine Learning is a type of Artificial Intelligence :
  - It aims to teach a machine to perform a certain task and provide accurate results by identifying patterns present in training example.
  - It is mainly directed towards automation of tasks without explicitly providing rules like in a program.
    - It replaces preset rules with a transformation that has learned the data.

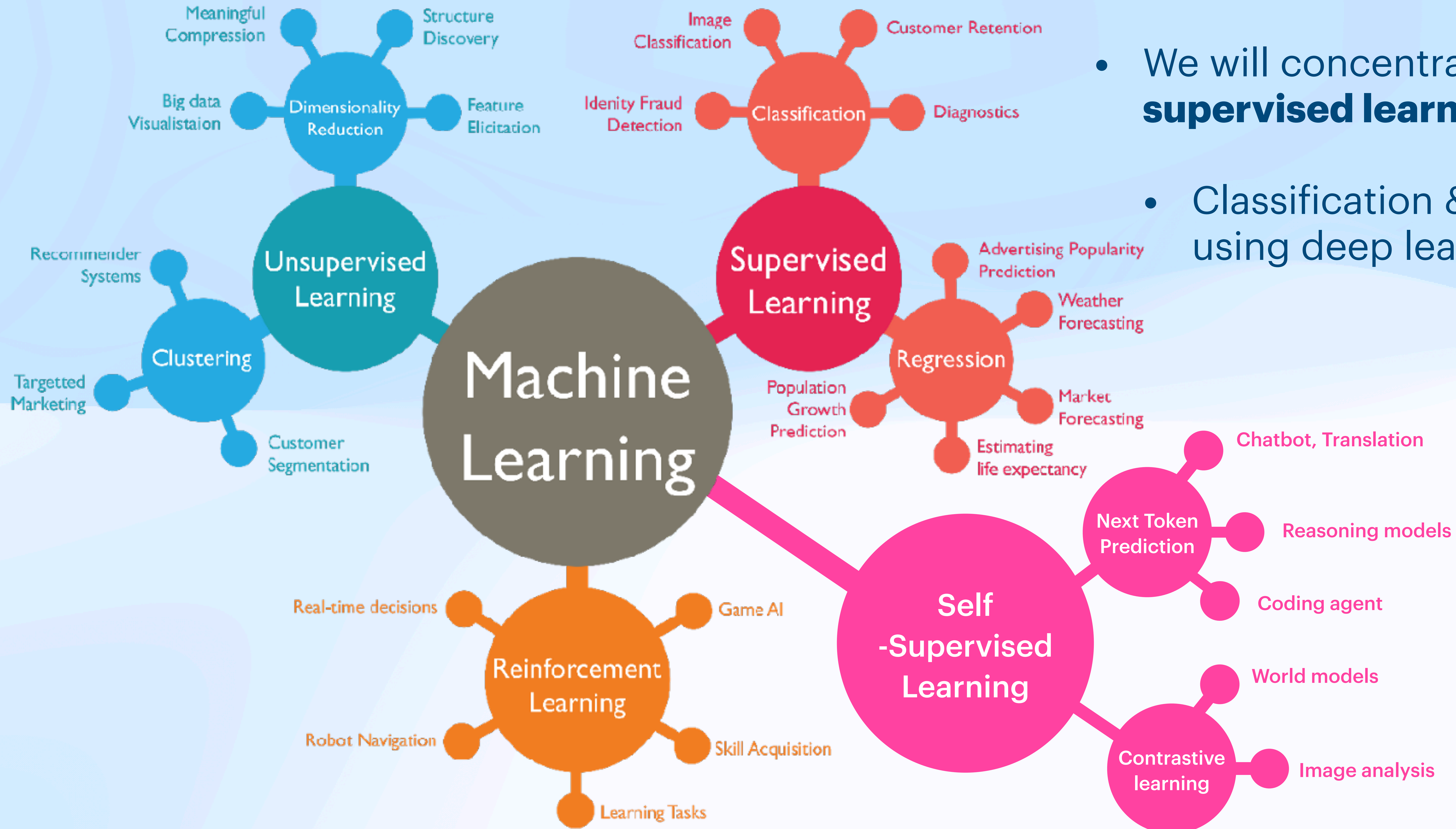


Antonin Vacheret

What is Machine Learning and Artificial Intelligence ?

- **Artificial Intelligence** has a broader goal to create programs that exhibits “intelligent behavior”.
  - Until very recently, AI was limited to narrow automation of tasks in the form of machine learning.
  - Since the arrival of Large Language Models (LLMs) as an “engine” for assisting work, “reasoning” and conducting task, AI has now permeated pretty much every fields and is available to anybody to help with writing, consulting, coding, scientific work etc…

Machine learning paradigms



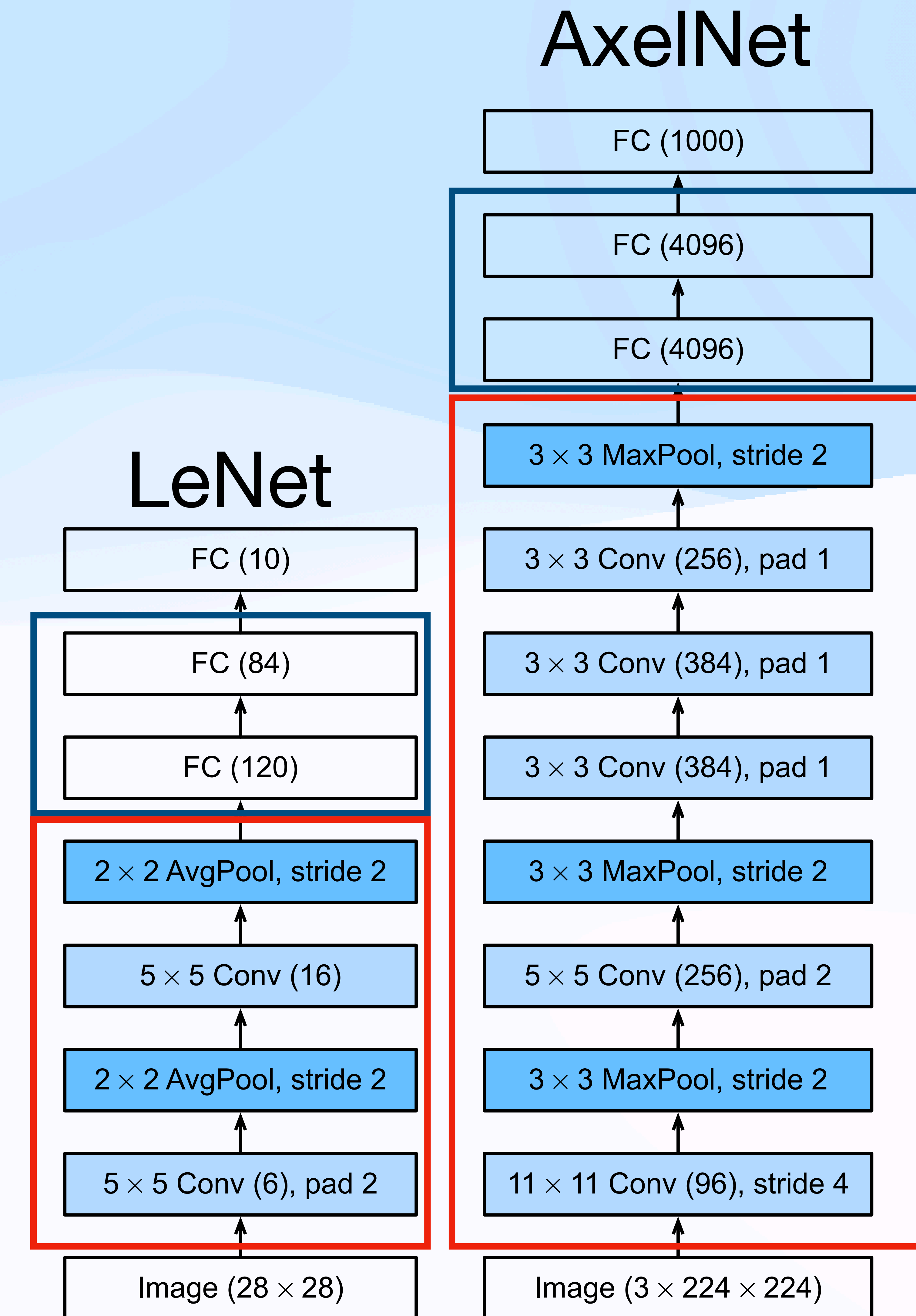
- We will concentrate this week to **supervised learning**
  - Classification & regression tasks using deep learning models

Antonin Vacheret

September 2012 - The deep learning Revolution

- Challenge to classify 1000 categories of images
- Main achievement is to use neural network-based method
- Work at scale : millions of parameters
- Learn features from examples

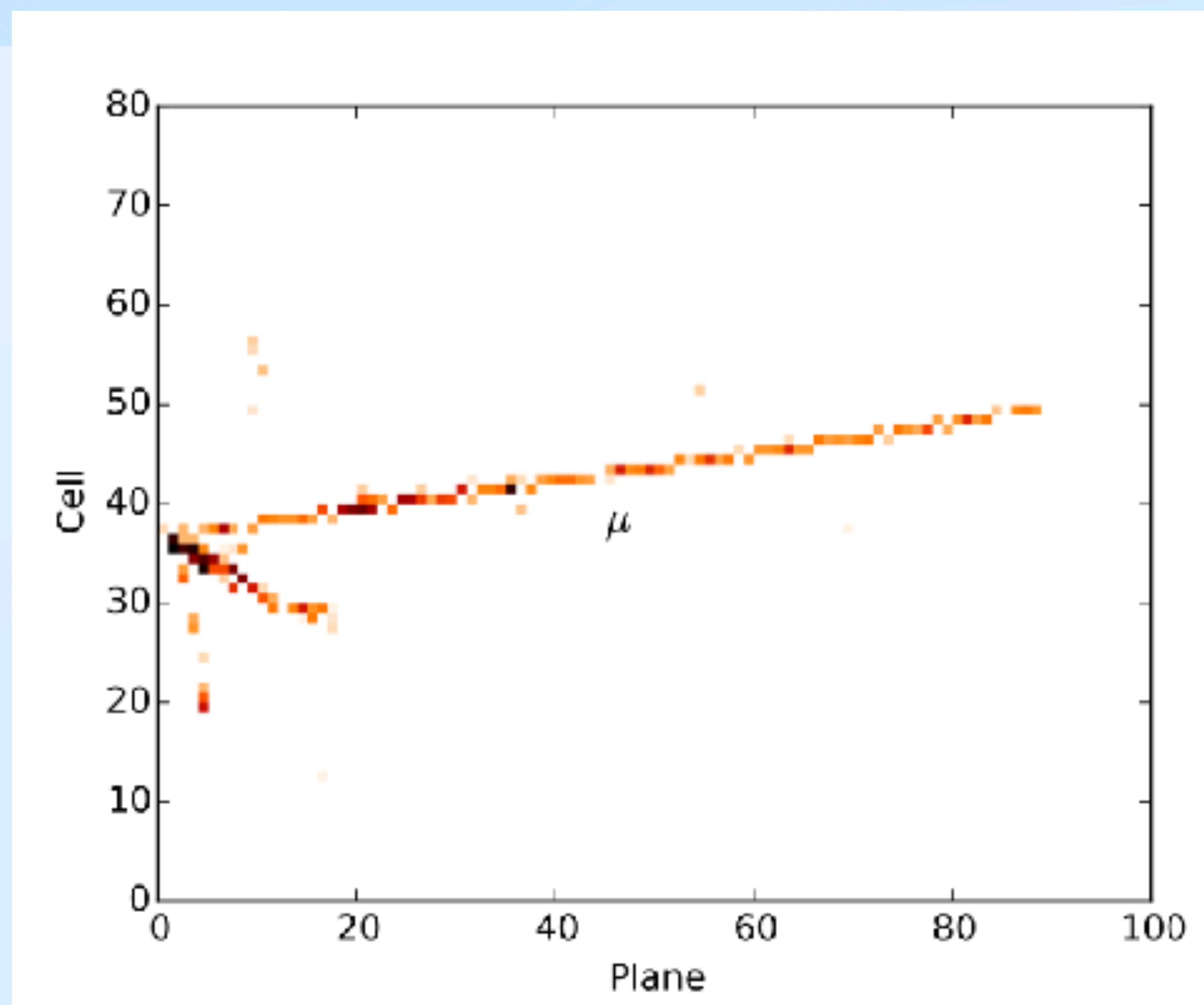
Compression + Abstraction



Antonin Vacheret

2016 AlexNet in Neutrino physics (NOvA)

- CNN classification transferable to non- natural images
- Deep learning is applicable to low level physics datasets !



A Convolutional Neural Network Neutrino Event Classifier

:1604.01444v3 [hep-ex] 12 Aug 2016

A. Aurisano,[a,1] A. Radovic,[b,1] D. Rocco,[c,1] A. Himmel,[d] M.D. Messier,[e] E. Niner,[d] G. Pawloski,[c] F. Psihas,[e] A. Sousa[a] and P. Vahle[b]

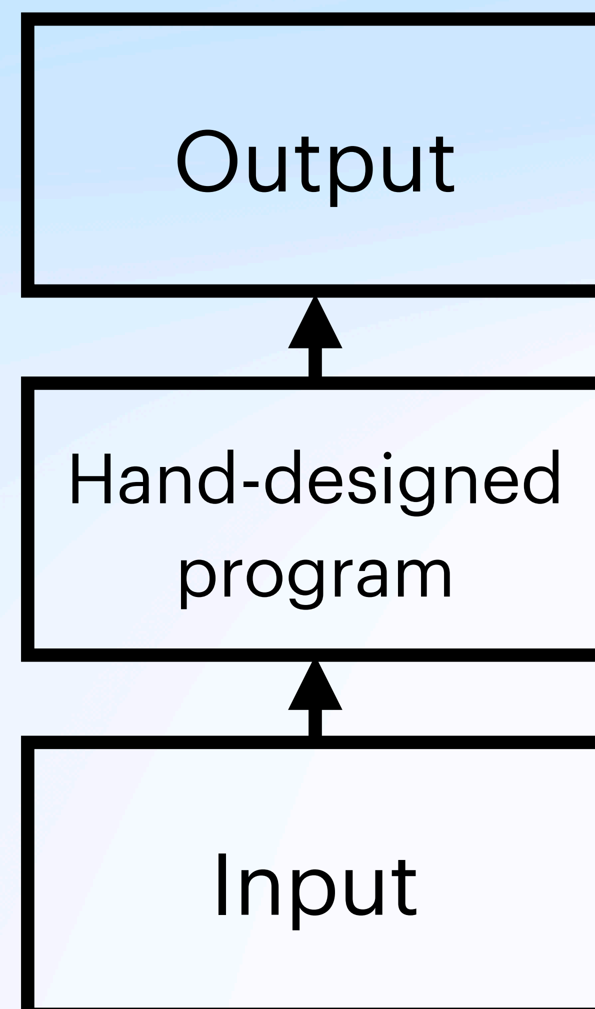
[a]*University of Cincinnati, Cincinnati, Ohio 45221, USA*
[b]*College of William & Mary, Williamsburg, Virginia 23187, USA*
[c]*University of Minnesota, Minneapolis, Minnesota 55455, USA*
[d]*Fermi National Accelerator Laboratory, Batavia, Illinois 60510, USA*
[e]*Indiana University, Bloomington, Indiana 47405, USA*

E-mail: aurisaam@ucmail.uc.edu, aradovic@wm.edu, rocco@physics.umn.edu

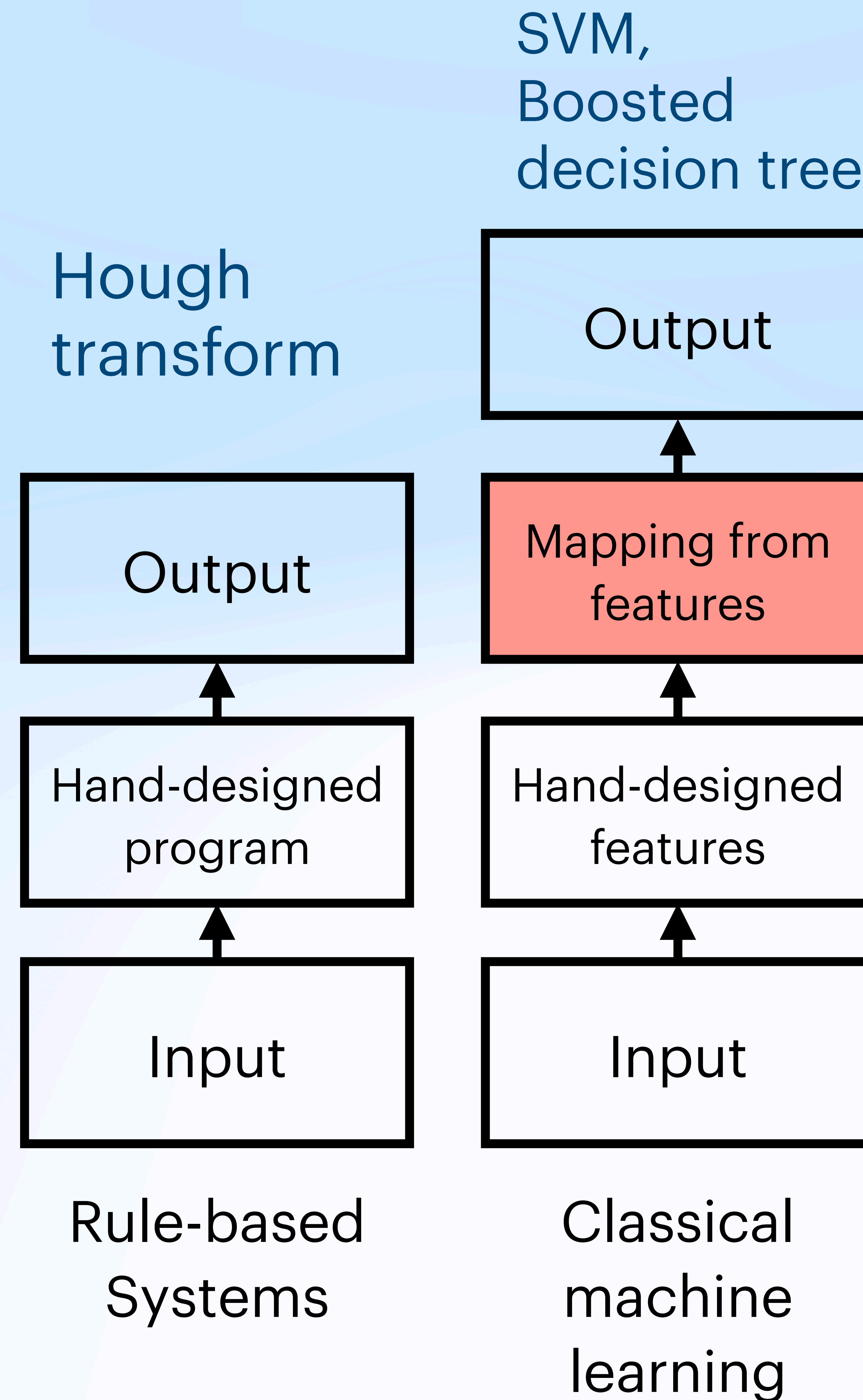
ABSTRACT: Convolutional neural networks (CNNs) have been widely applied in the computer vision community to solve complex problems in image recognition and analysis. We describe an application of the CNN technology to the problem of identifying particle interactions in sampling calorimeters used commonly in high energy physics and high energy neutrino physics in particular. Following a discussion of the core concepts of CNNs and recent innovations in CNN architectures related to the field of deep learning, we outline a specific application to the NOvA neutrino detector. This algorithm, CVN (Convolutional Visual Network) identifies neutrino interactions based on their topology without the need for detailed reconstruction and outperforms algorithms currently

Antonin Vacheret

The deep learning paradigm

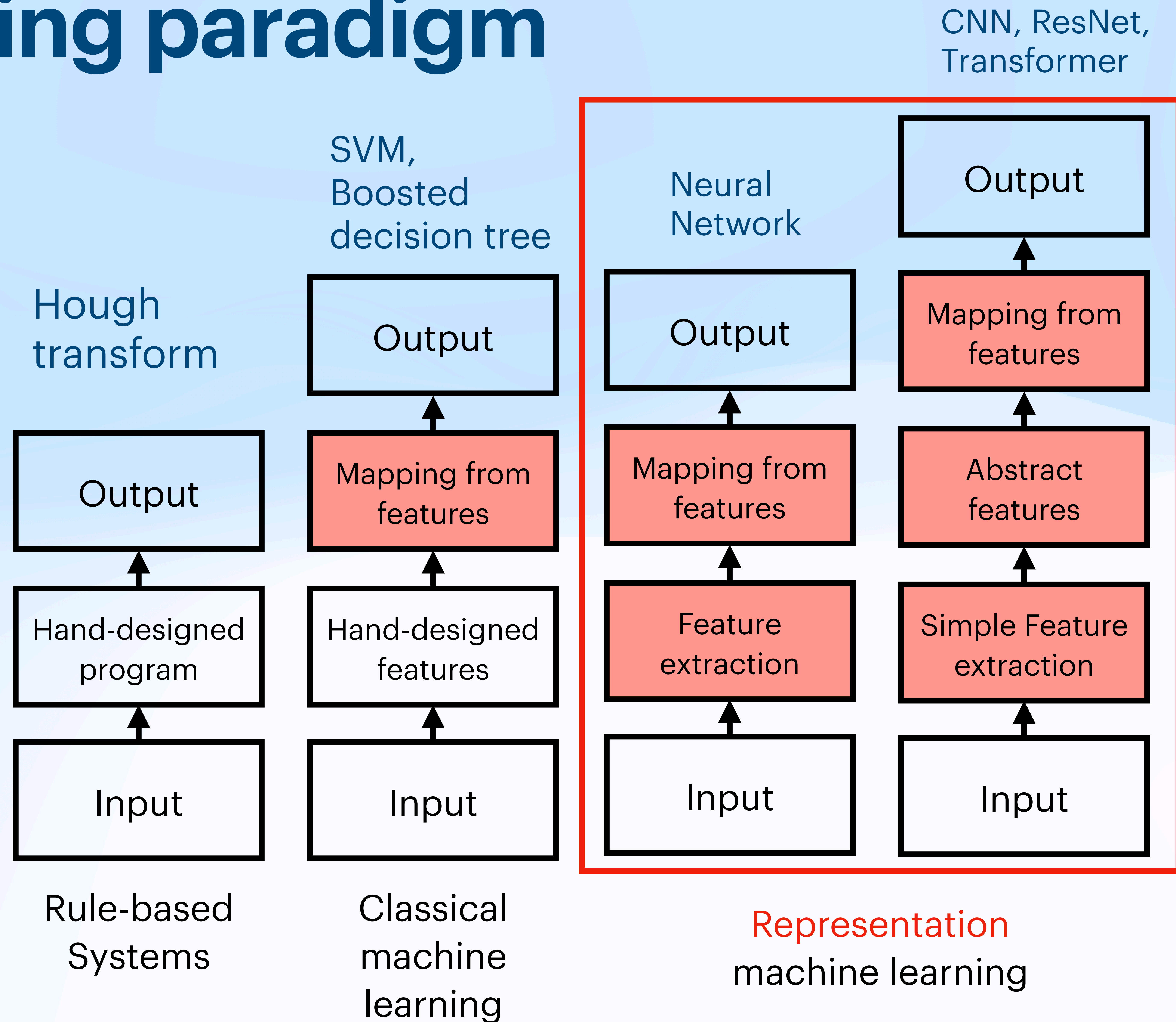


The deep learning paradigm



Antonin Vacheret

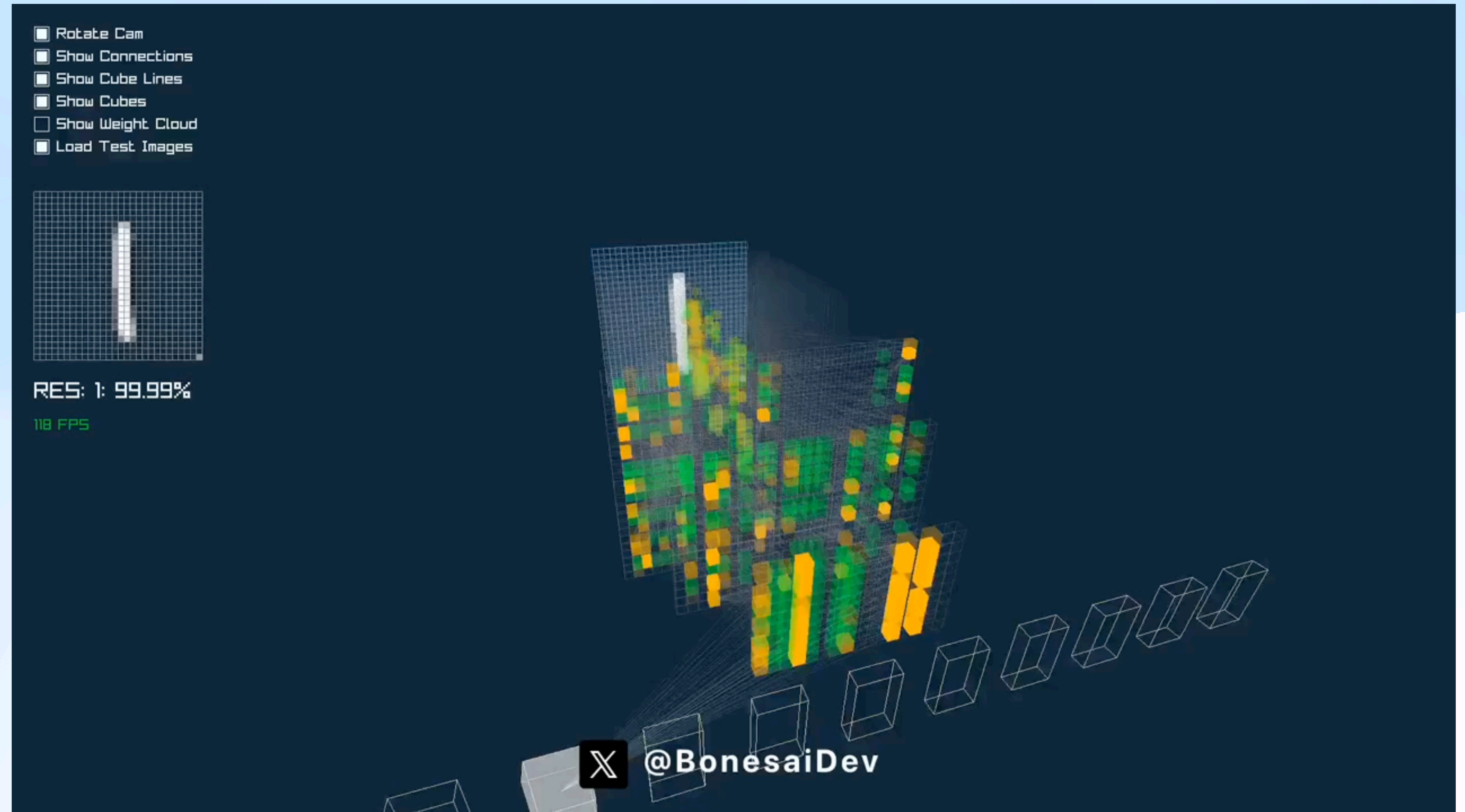
The deep learning paradigm



Antonin Vacheret

Learning from the data

- Simple DL model on MNIST dataset
- Using image features to classify the numbers



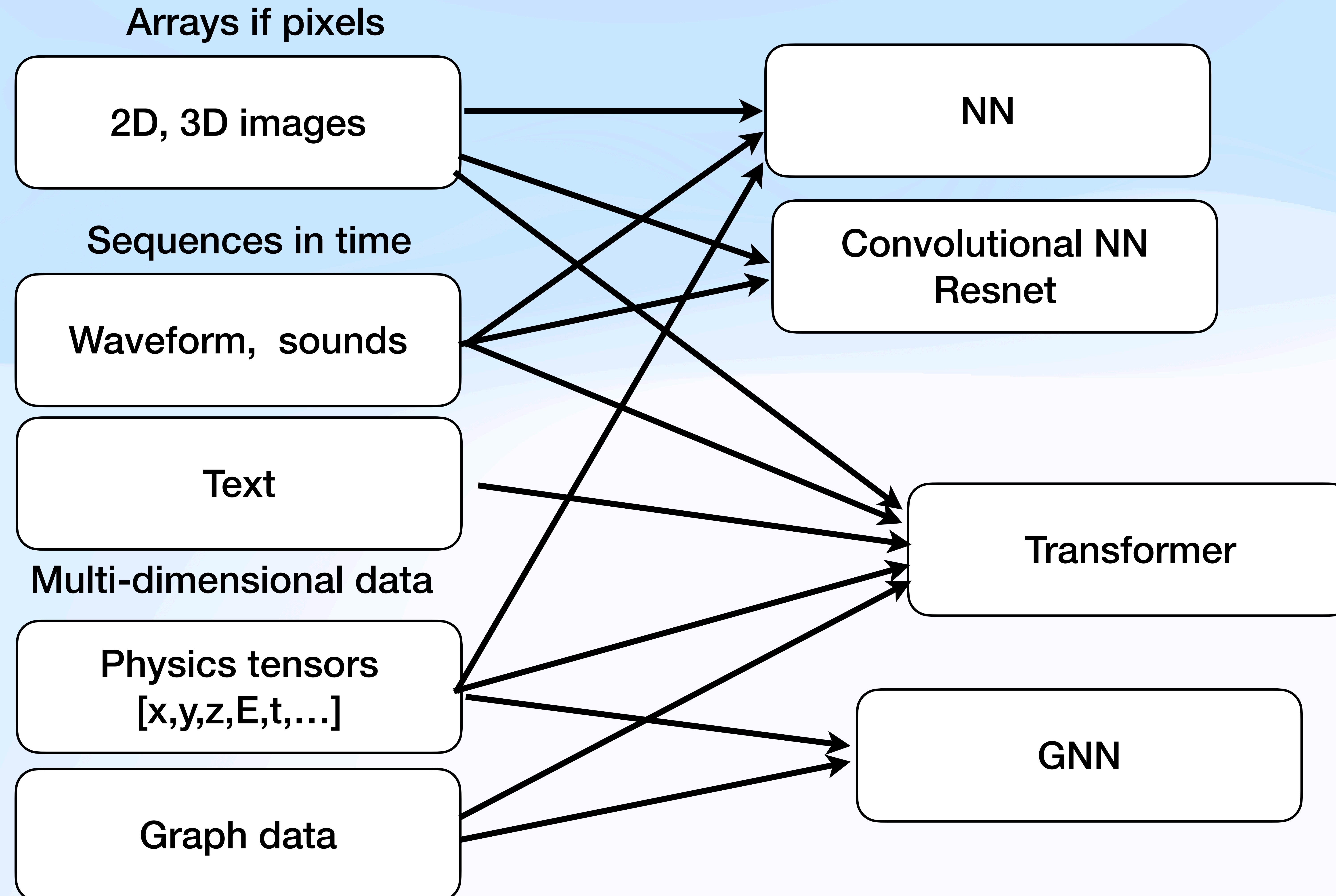
Data representation in the model

- Visual features are learned from the data

Antonin Vacheret

Data modalities and model architecture

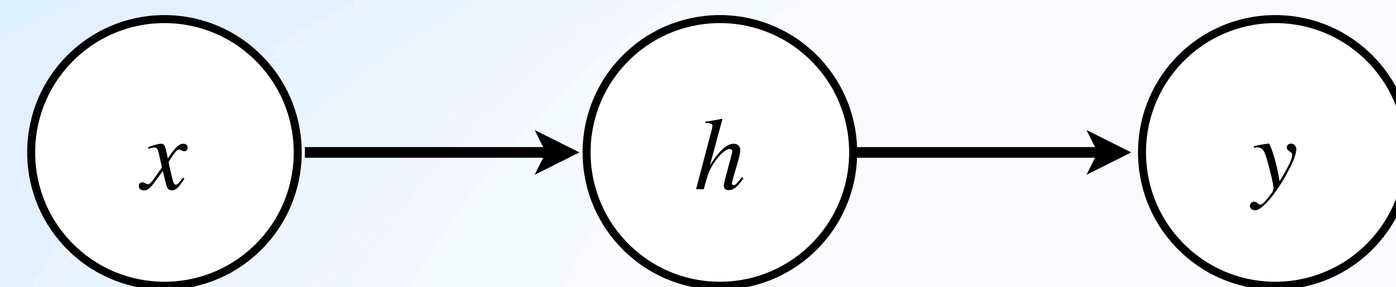
Many models but not tailored to all modalities



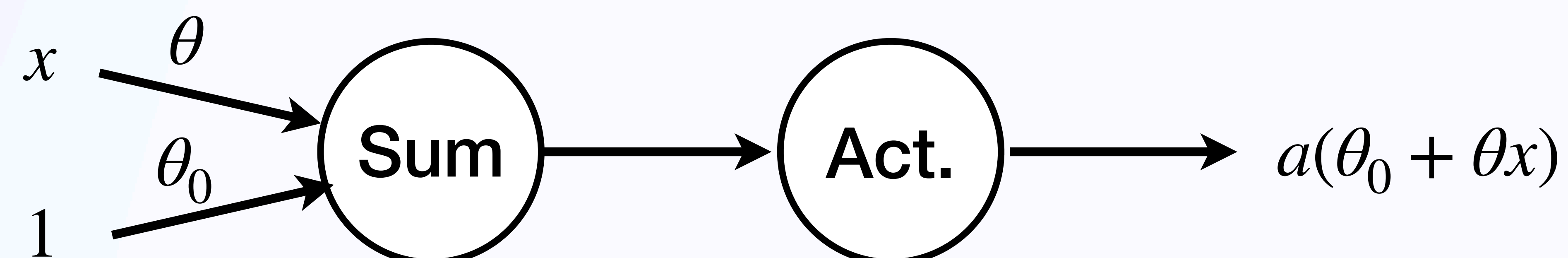
The Neural network

Functional unit of a neural network

- In supervised learning we want a model (transformation) that take input **x** and output a prediction **y**
- **x** and **y** are considered here as vectors
- A neural network is based on a single unit called the "neuron" or hidden unit h



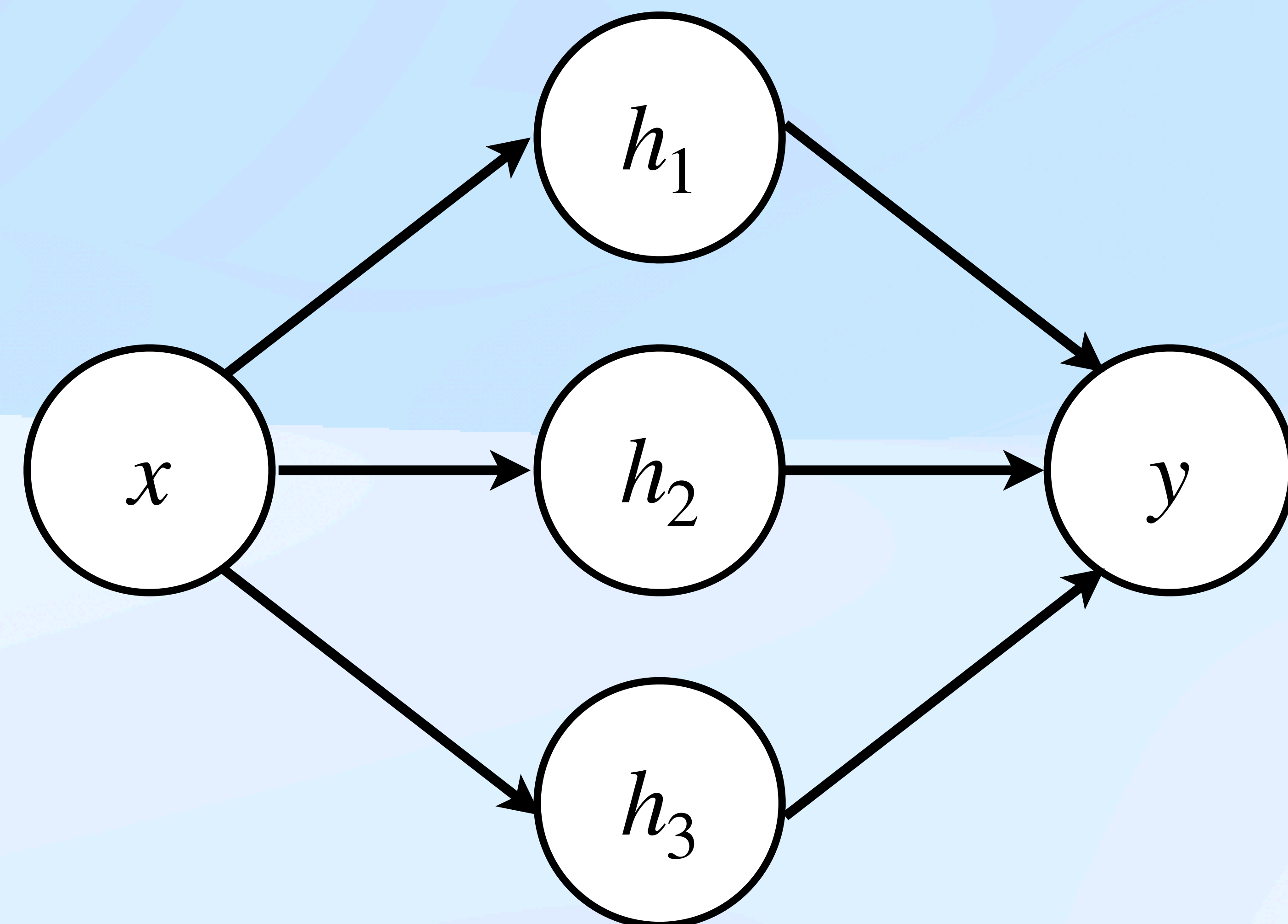
- h is based on a linear function followed by an "activation" :



Building a Neural Network model

Example of a NN with 1 input, 3 hidden units and 1 output

$$y = \phi_0 + \phi_1 \mathrm{a}[\theta_{10} + \theta_{11}x] + \phi_2 \mathrm{a}[\theta_{20} + \theta_{21}x] + \phi_3 \mathrm{a}[\theta_{30} + \theta_{31}x].$$

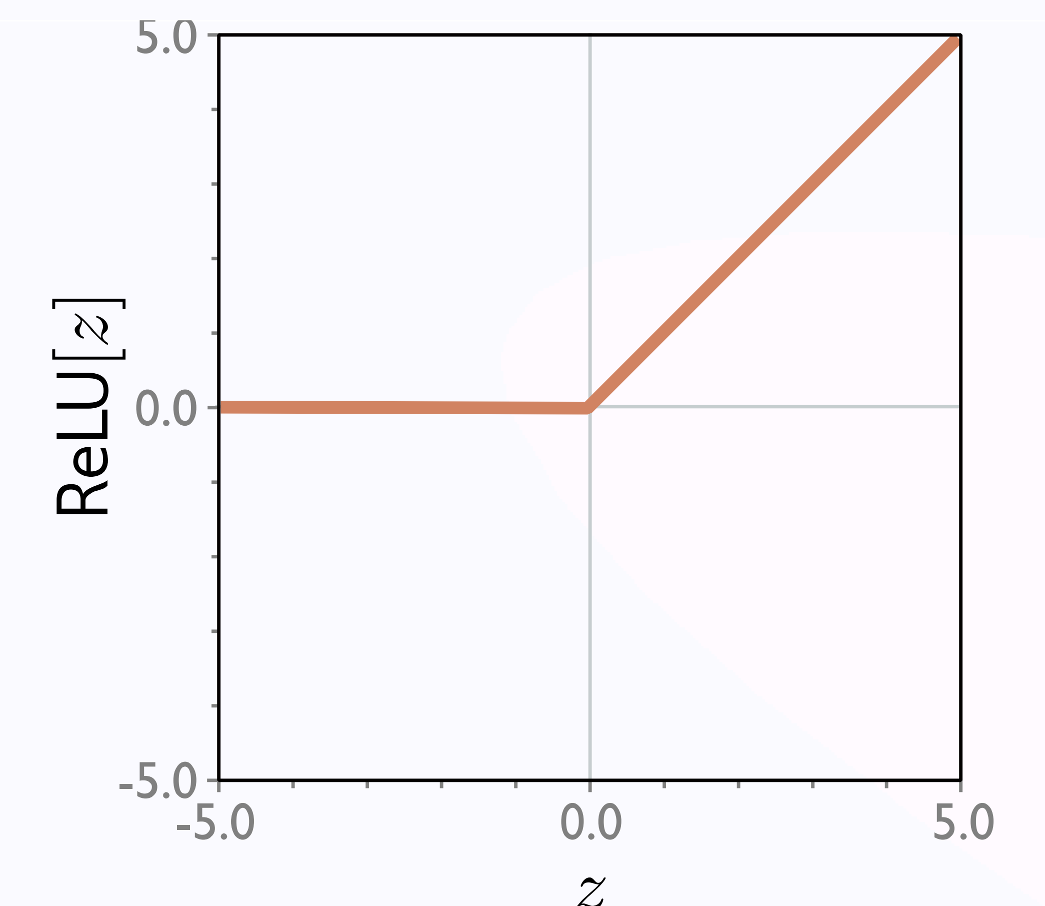


$$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x]$$
$$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x]$$
$$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x]$$

- Each neuron takes the input x and output y
- We choose the activation function as ReLU

$$\mathrm{a}[z] = \mathrm{ReLU}[z] = \begin{cases} 0 & z < 0 \\ z & z \geq 0 \end{cases}.$$

Rectified Linear Unit
(particular kind of activation function)

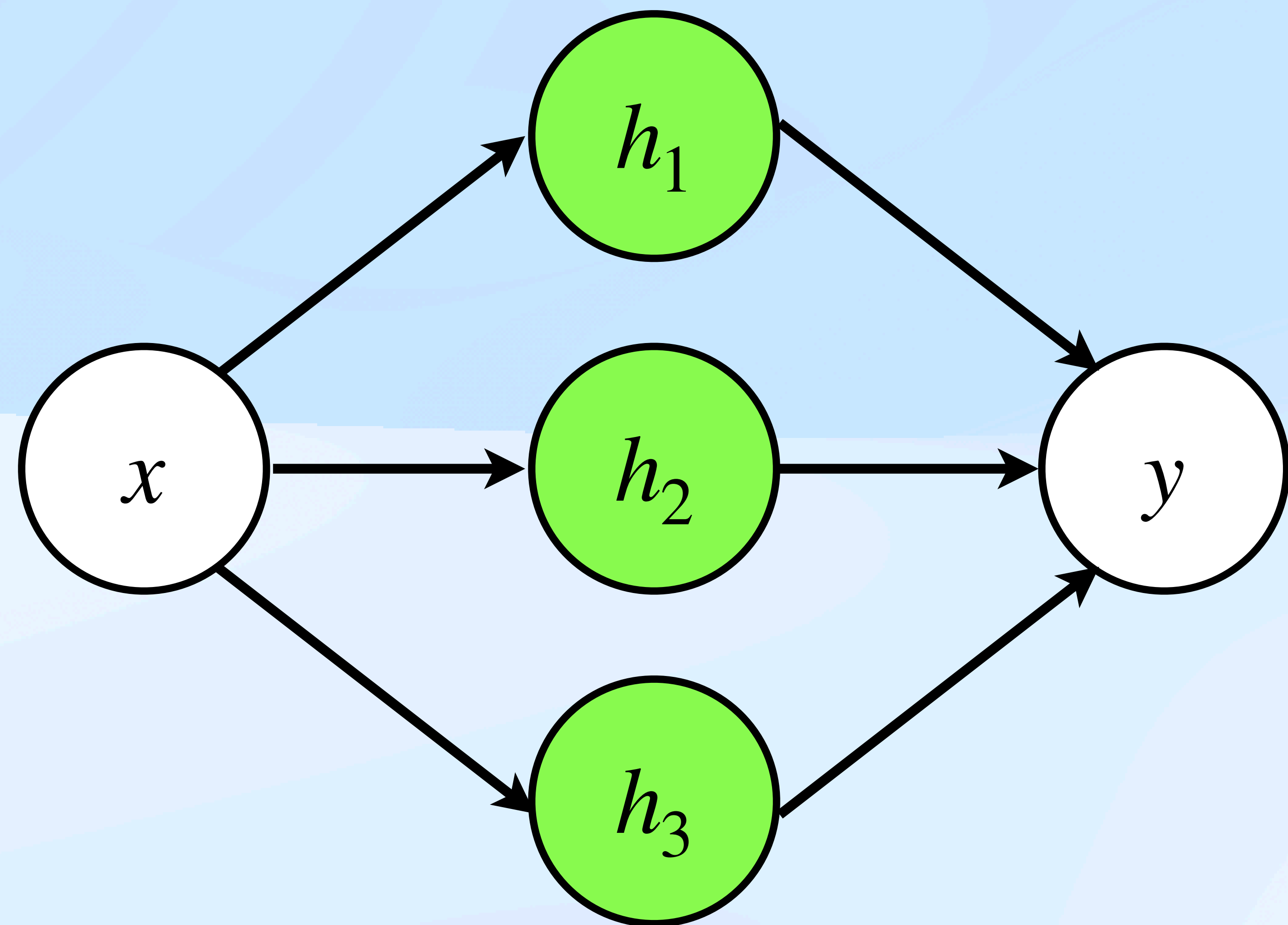


Antonin Vacheret

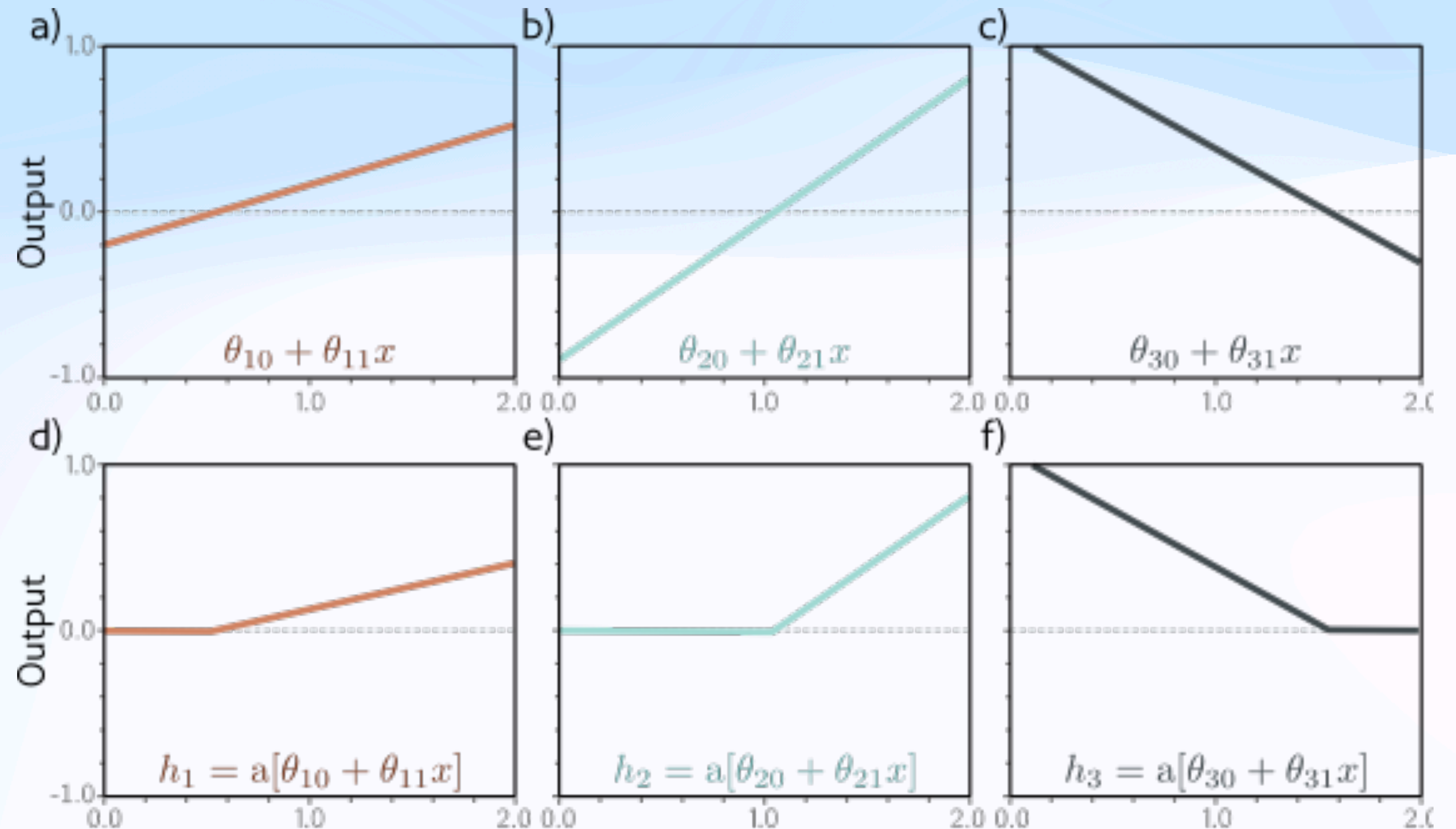
Building a Neural network model

Example of a NN with 1 input, 3 hidden units and 1 output

$$y = \phi_0 + \phi_1 \mathrm{a}[\theta_{10} + \theta_{11}x] + \phi_2 \mathrm{a}[\theta_{20} + \theta_{21}x] + \phi_3 \mathrm{a}[\theta_{30} + \theta_{31}x].$$



$$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x]$$
$$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x]$$
$$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x]$$

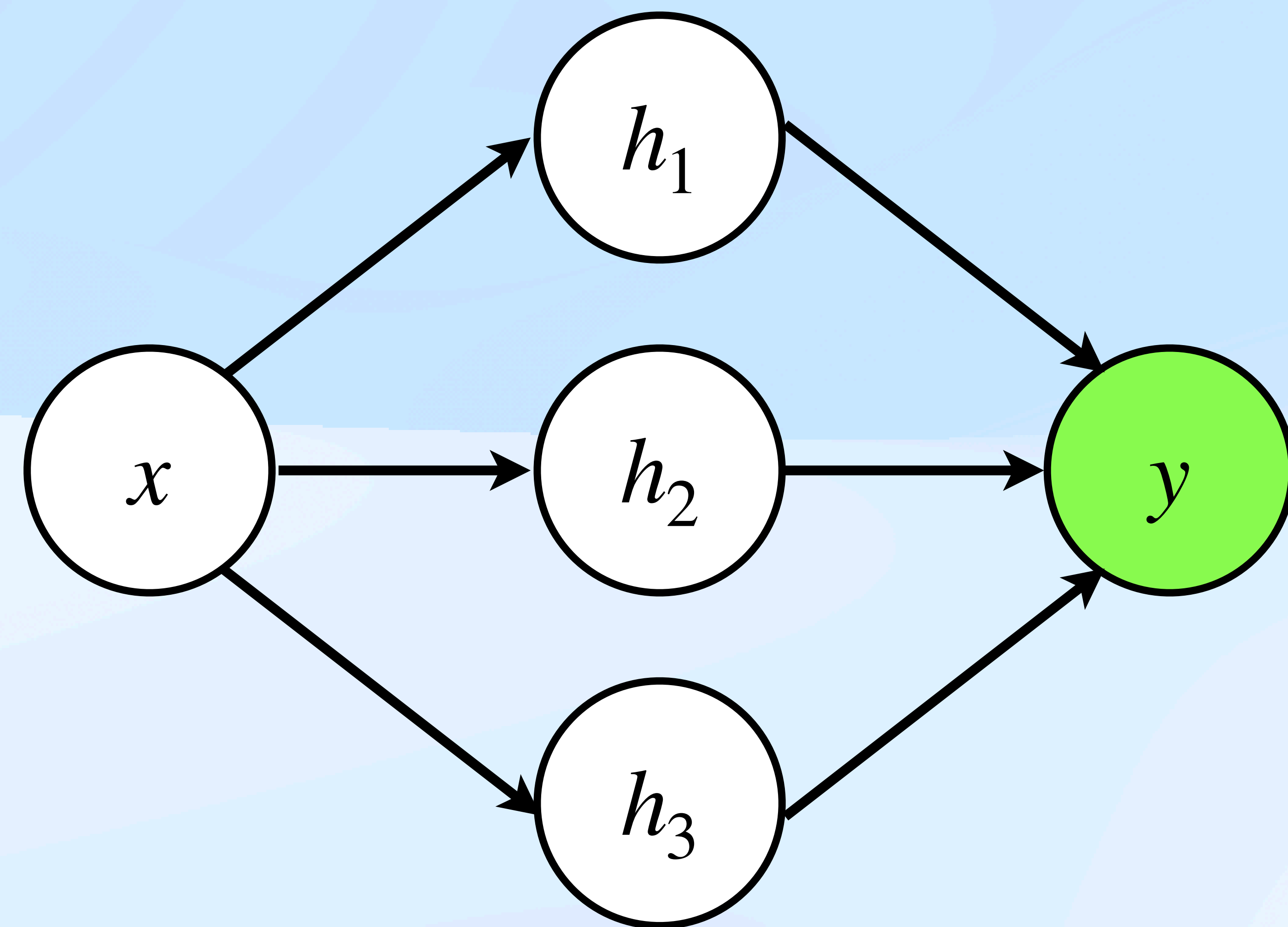


Antonin Vacheret

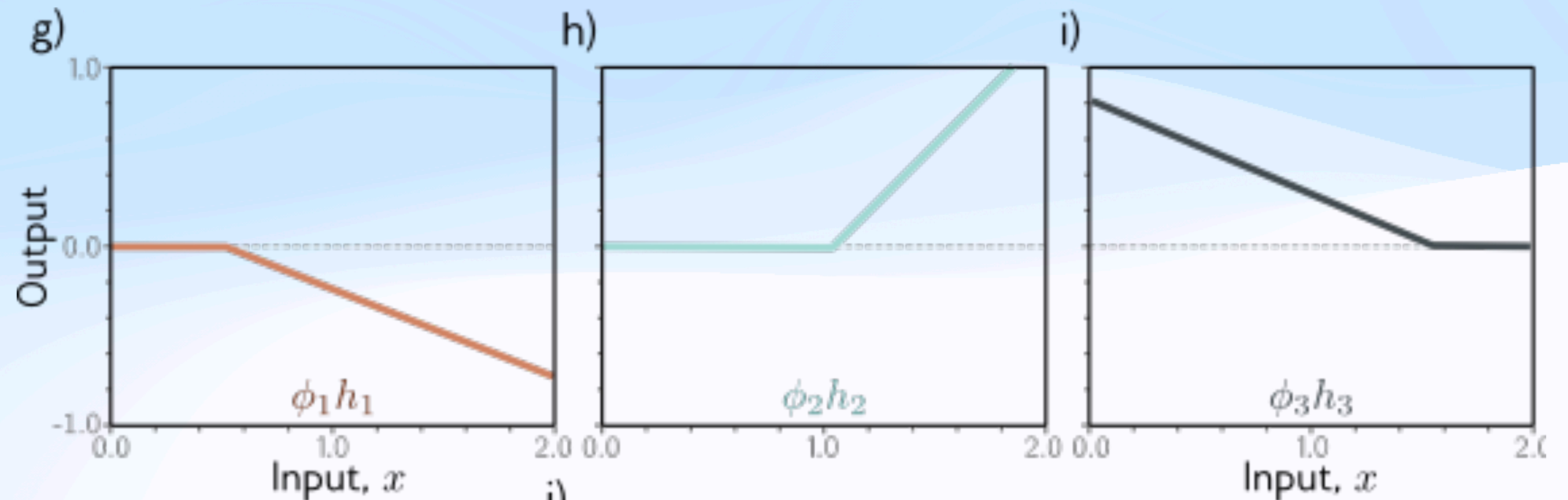
Building a Neural network model

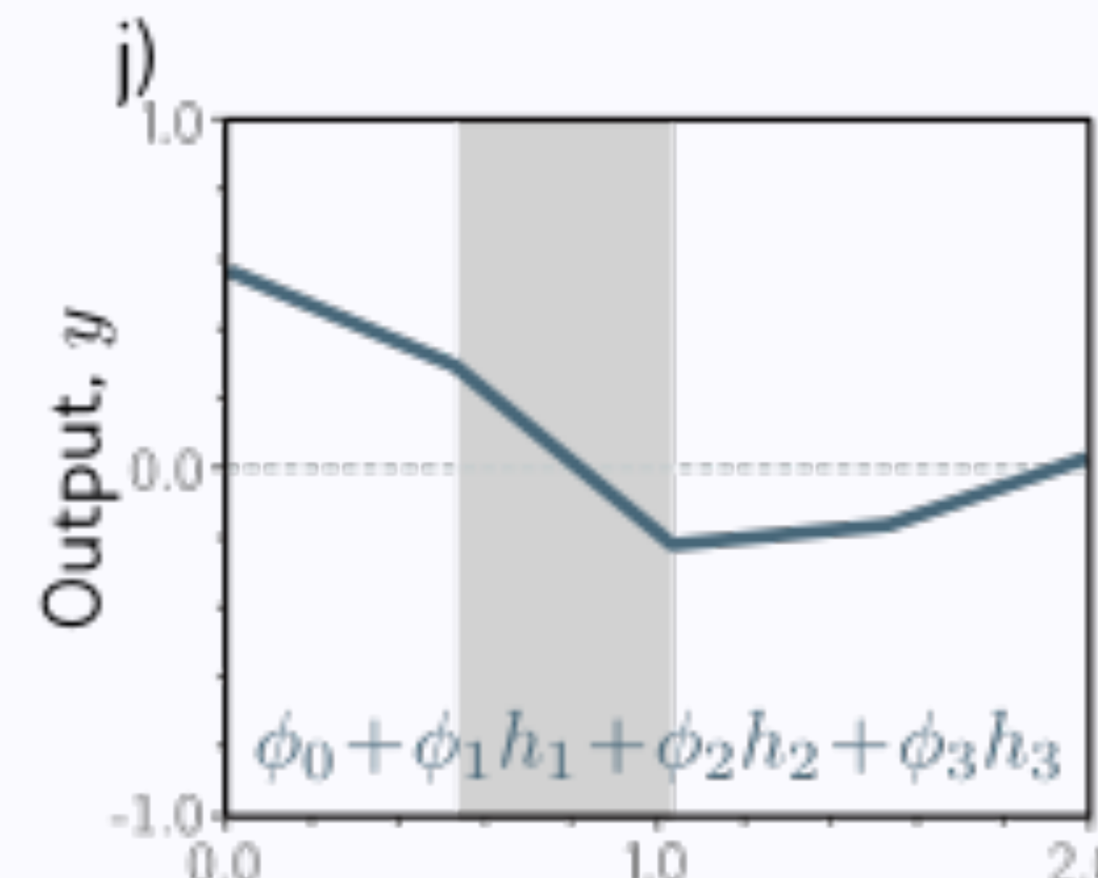
Example of a NN with 1 input, 3 hidden units and 1 output

$$y = \phi_0 + \phi_1 \mathrm{a}[\theta_{10} + \theta_{11}x] + \phi_2 \mathrm{a}[\theta_{20} + \theta_{21}x] + \phi_3 \mathrm{a}[\theta_{30} + \theta_{31}x].$$



$$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x]$$
$$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x]$$
$$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x]$$





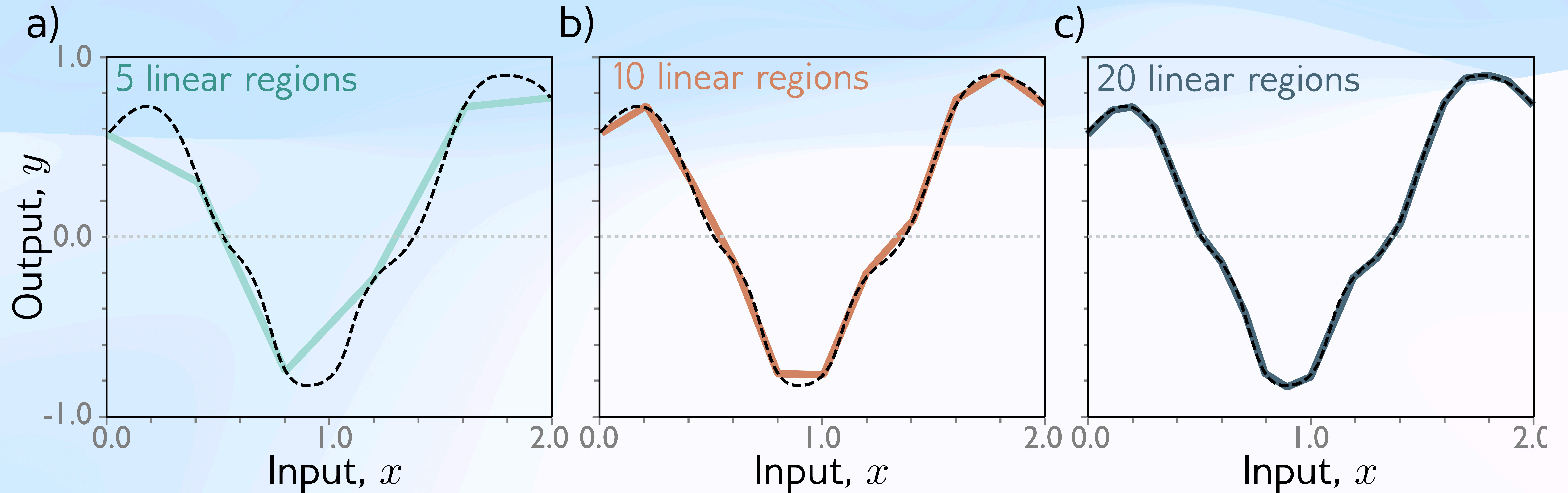
Shaded region:
- Unit 1 active
- Unit 2 inactive
- Unit 3 active

Antonin Vacheret

Shallow Neural networks

With enough hidden units...

... we can describe any 1D function to arbitrary accuracy

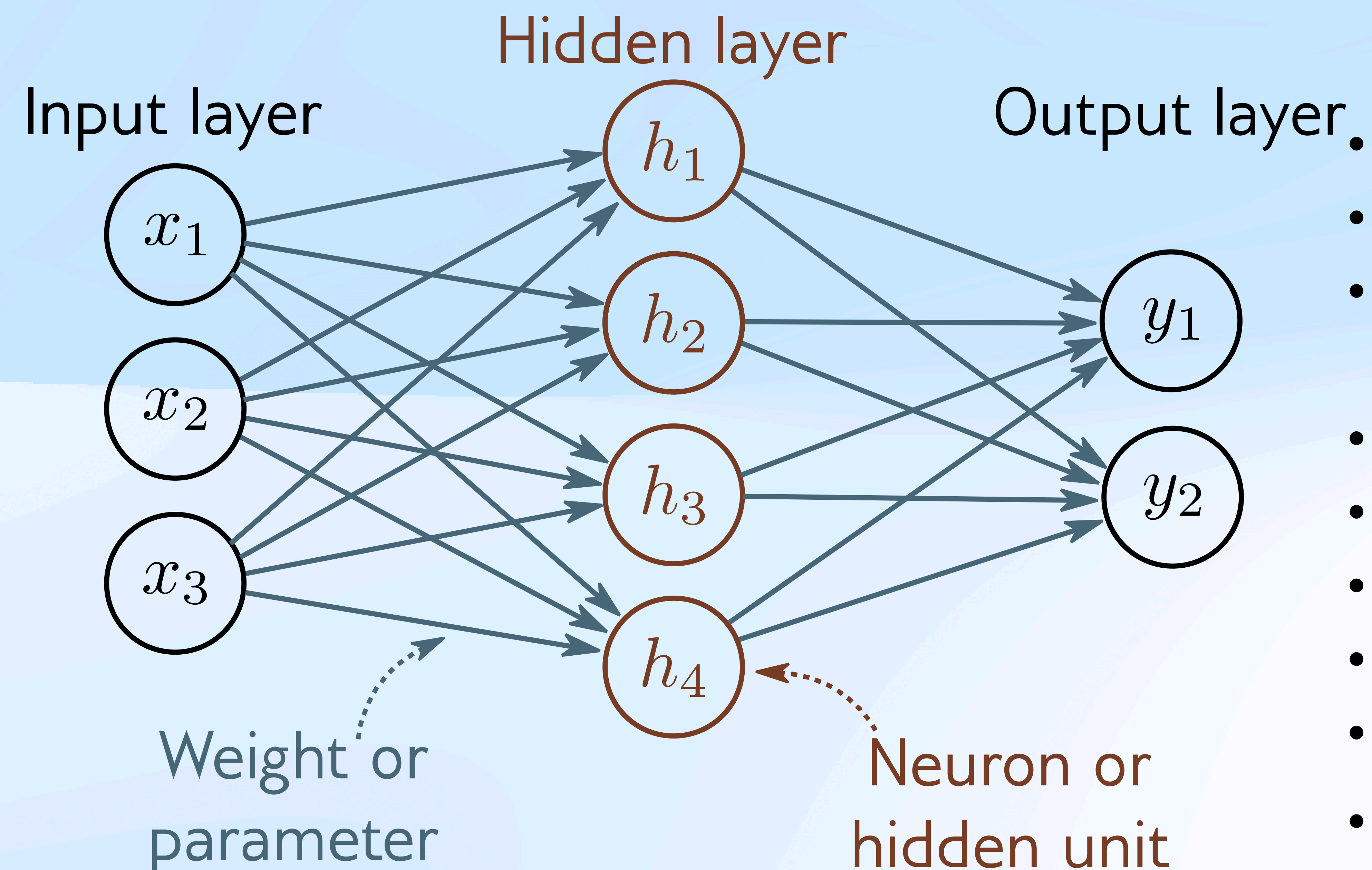


Antonin Vacheret

Universal approximation theorem

"a formal proof that, with enough hidden units, a shallow neural network can describe any continuous function on a compact subset of $\mathbb{R}^D$ to arbitrary precision"

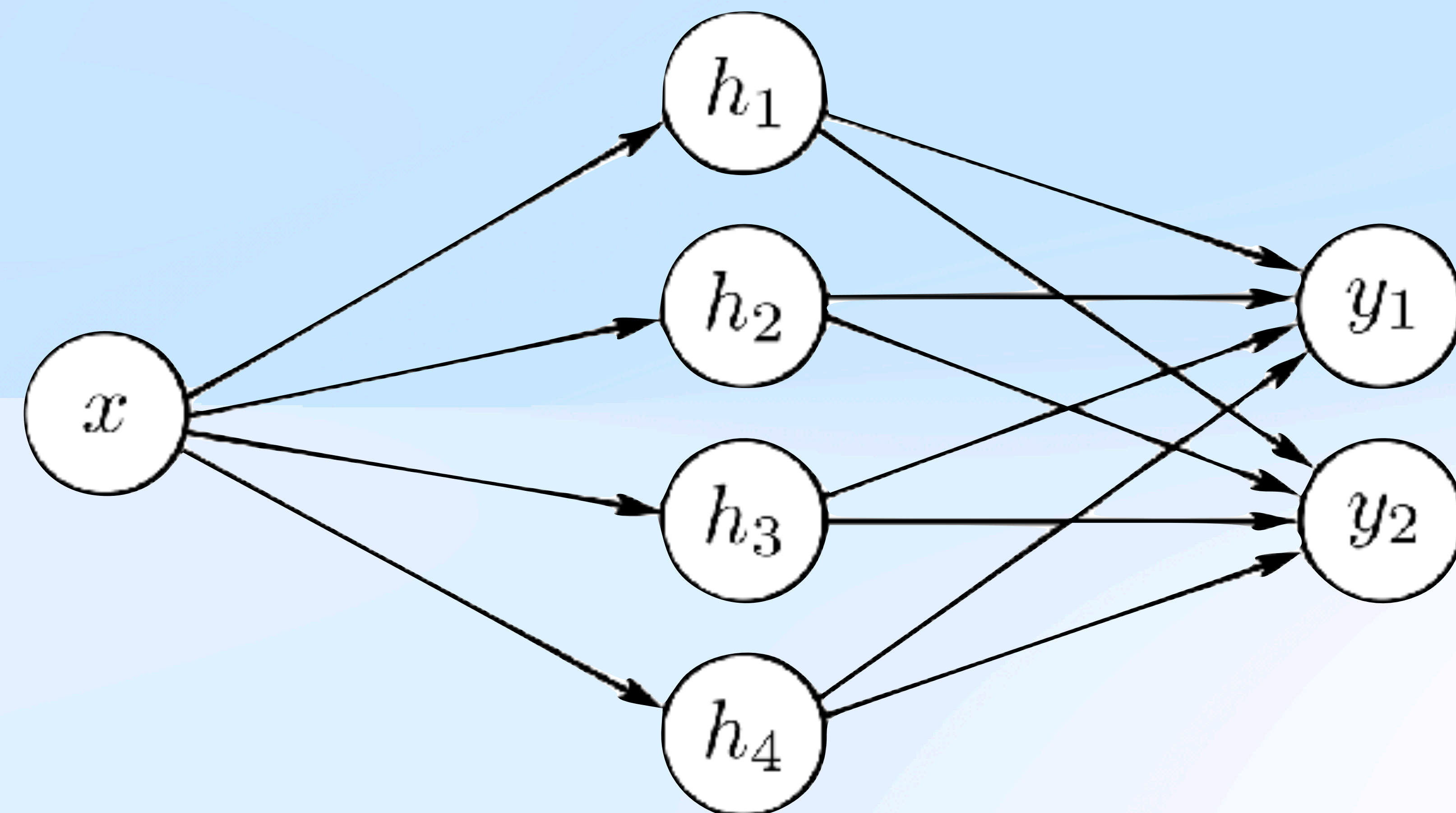
Neural Network : nomenclature



- Y-offsets = biases
- Slopes = weights
- Everything in one layer connected to everything in the next = Fully Connected Network
- No loops = Feedforward network
- Values after ReLU (activation functions) = activations
- Values before ReLU = pre-activations
- One hidden layer = shallow neural network
- More than one hidden layer = deep neural network
- Number of hidden units $\approx$ capacity

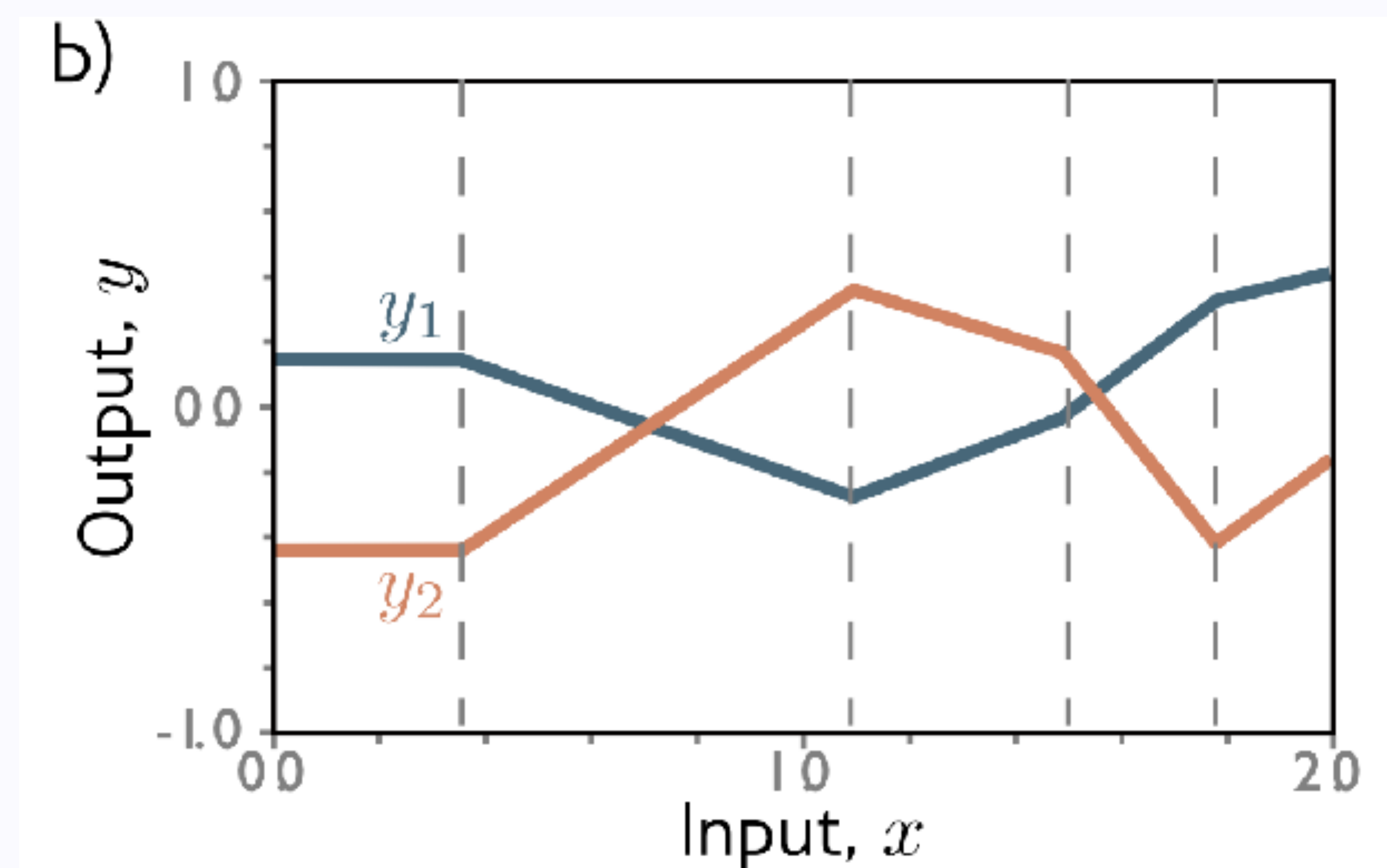
More outputs

We can construct networks with more than 1 input and output
example of 2 outputs :



$$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x]$$
$$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x]$$
$$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x]$$
$$h_4 = \mathrm{a}[\theta_{40} + \theta_{41}x]$$

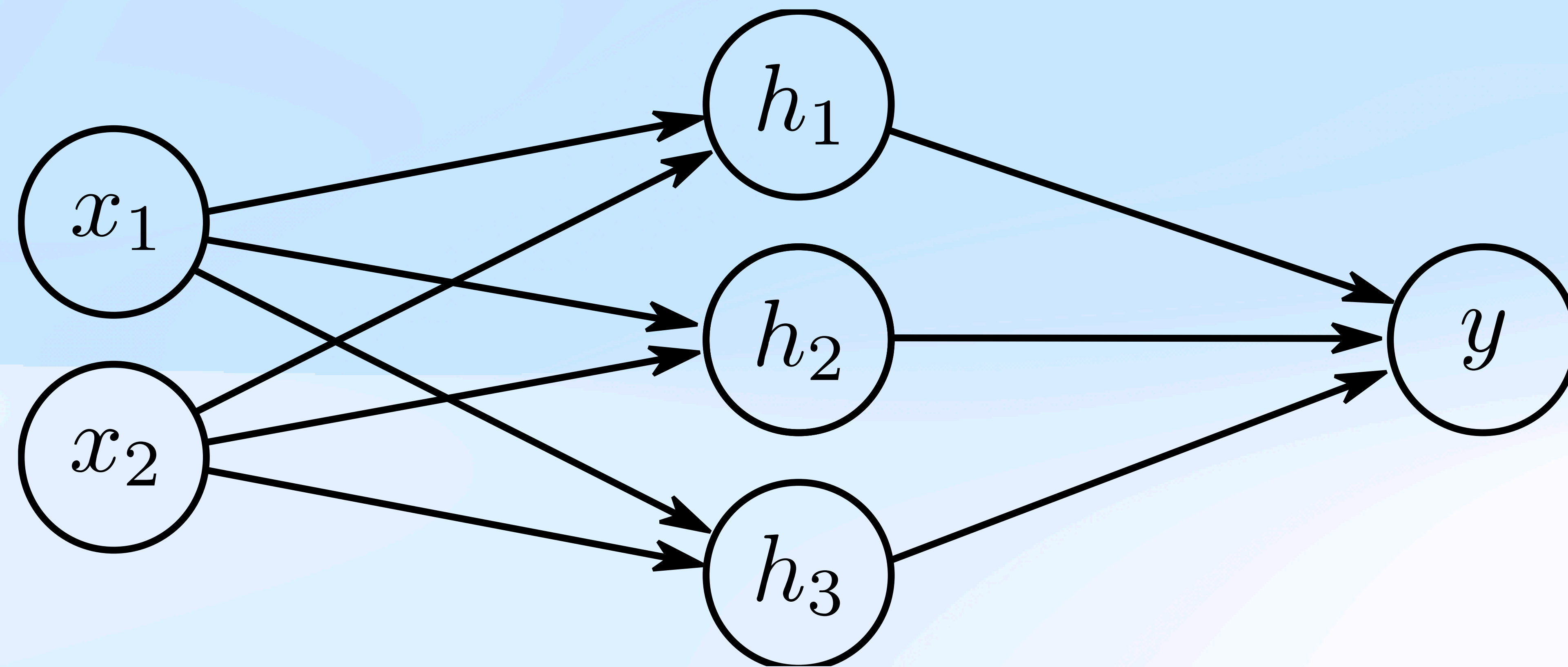
$$y_1 = \phi_{10} + \phi_{11}h_1 + \phi_{12}h_2 + \phi_{13}h_3 + \phi_{14}h_4$$
$$y_2 = \phi_{20} + \phi_{21}h_1 + \phi_{22}h_2 + \phi_{23}h_3 + \phi_{24}h_4$$



Antonin Vacheret

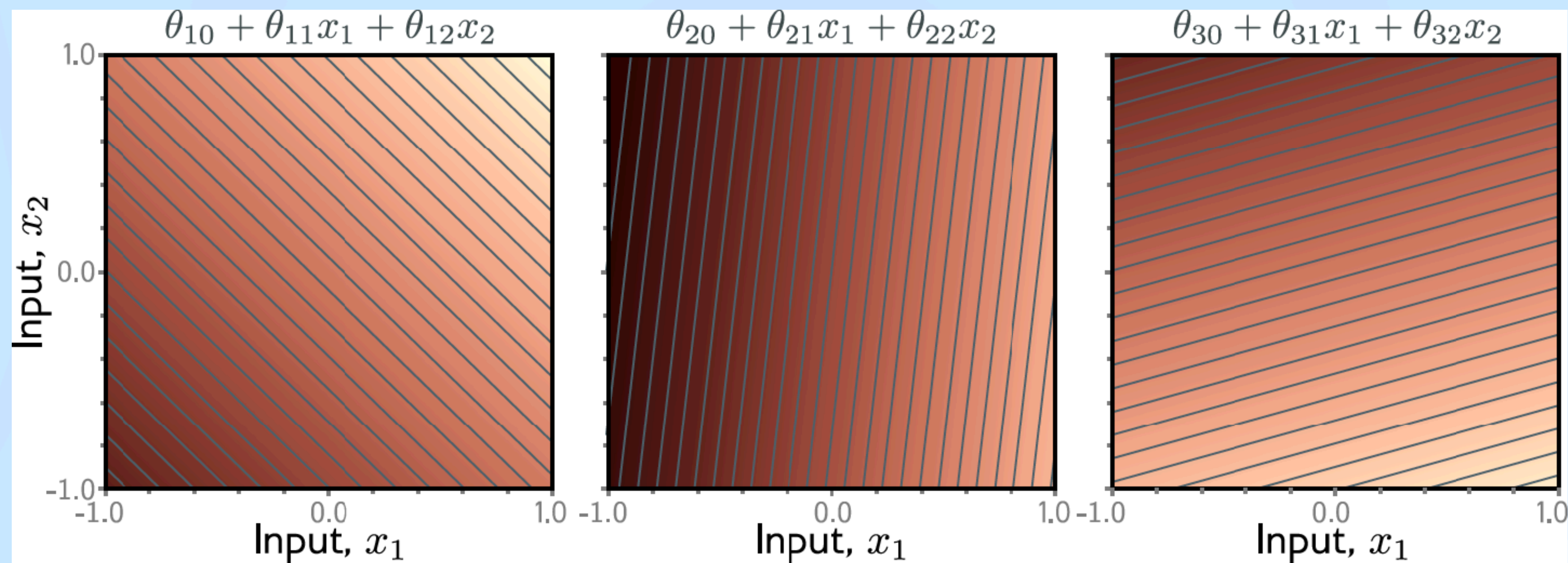
More inputs

We can construct networks with more than 1 input and outputs : example of 2 inputs

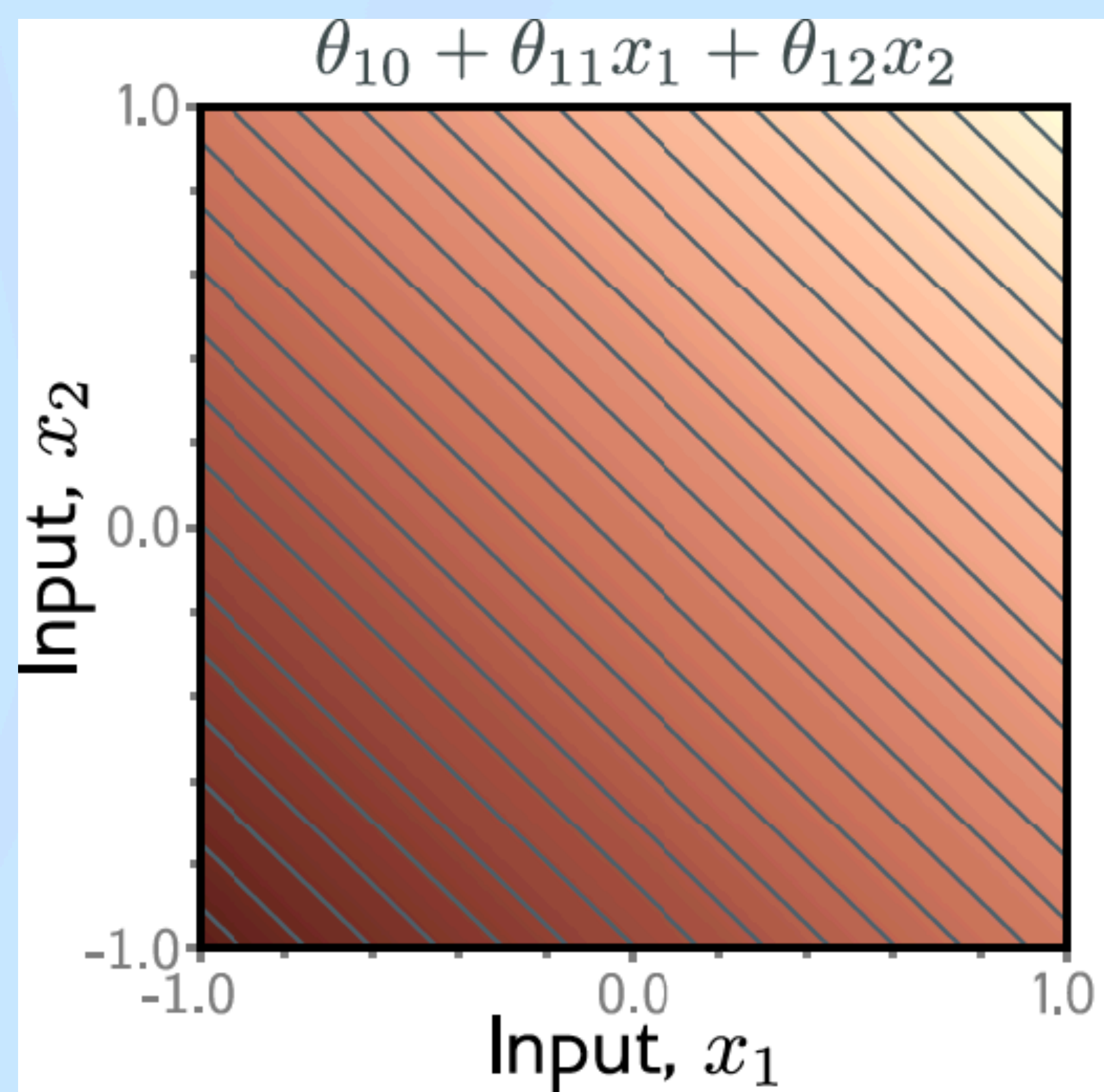


$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$

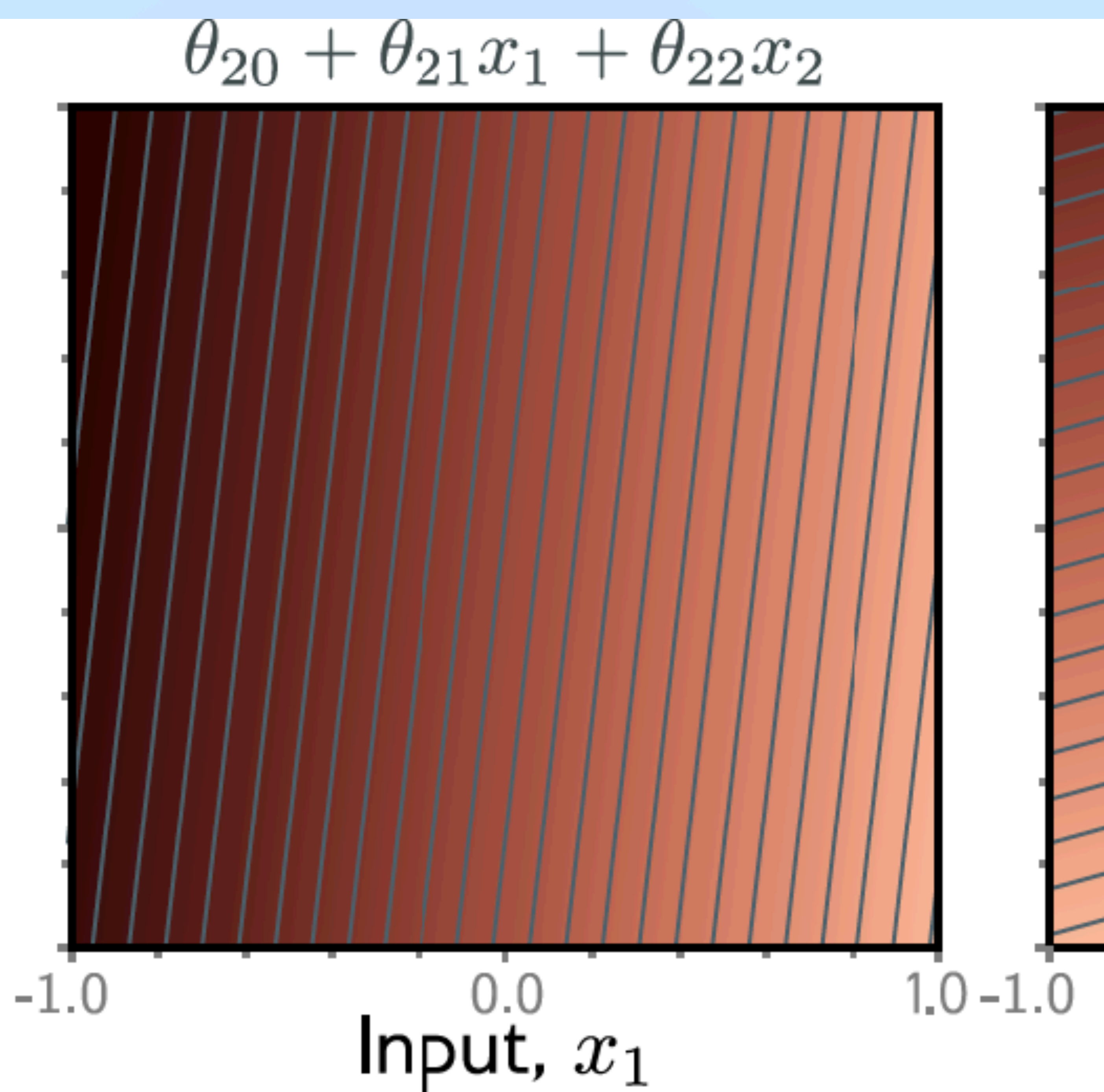
$$h_1 = \text{a}[\theta_{10} + \theta_{11} x_1 + \theta_{12} x_2]$$
$$h_2 = \text{a}[\theta_{20} + \theta_{21} x_1 + \theta_{22} x_2]$$
$$h_3 = \text{a}[\theta_{30} + \theta_{31} x_1 + \theta_{32} x_2]$$



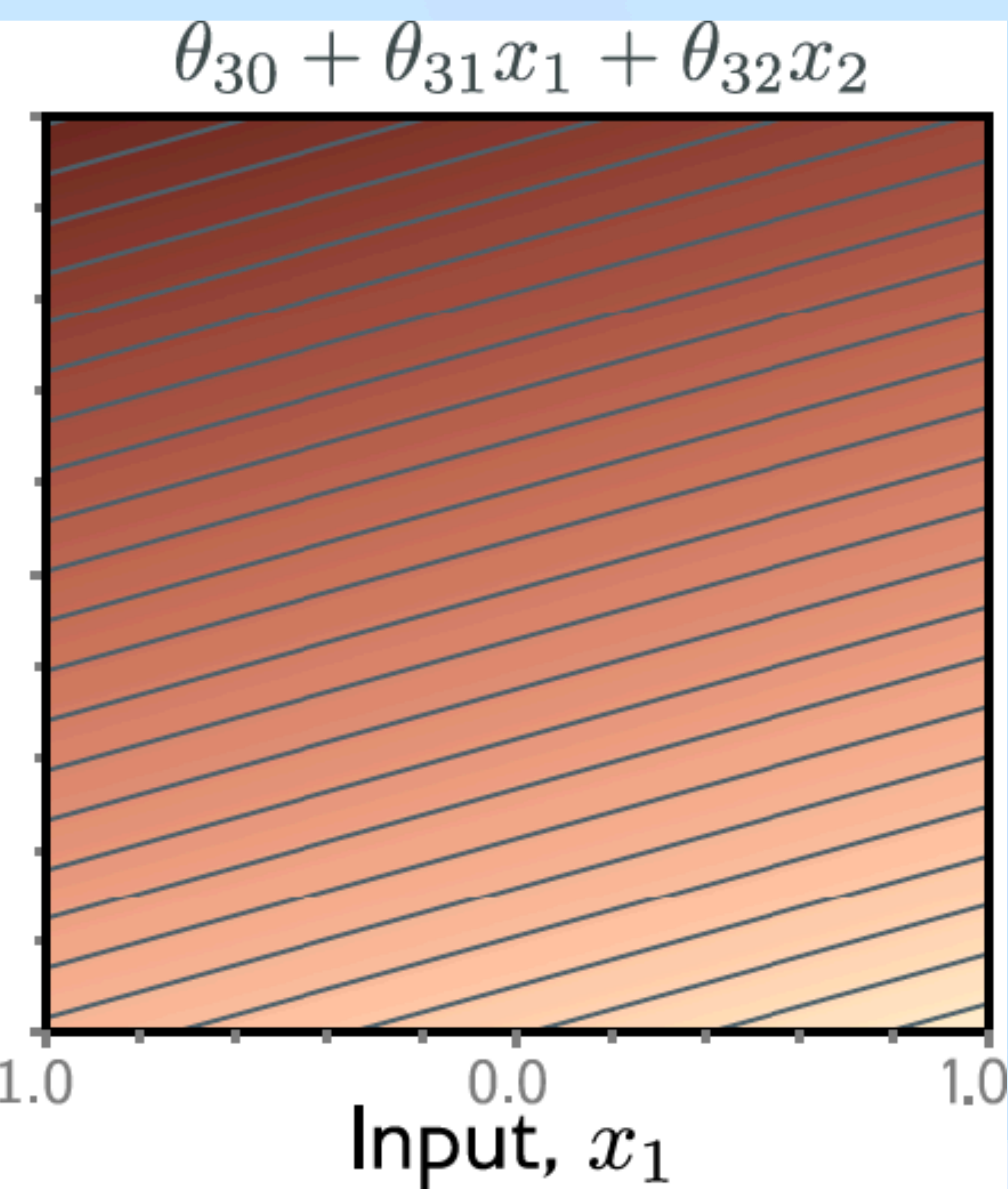
$\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2$
$\theta_{20} + \theta_{21}x_1 + \theta_{22}x_2$
$\theta_{30} + \theta_{31}x_1 + \theta_{32}x_2$
Input, x_2
1.0
0.0
-1.0
-1.0
0.0
1.0
Input, x_1
Input, x_1
Input, x_1



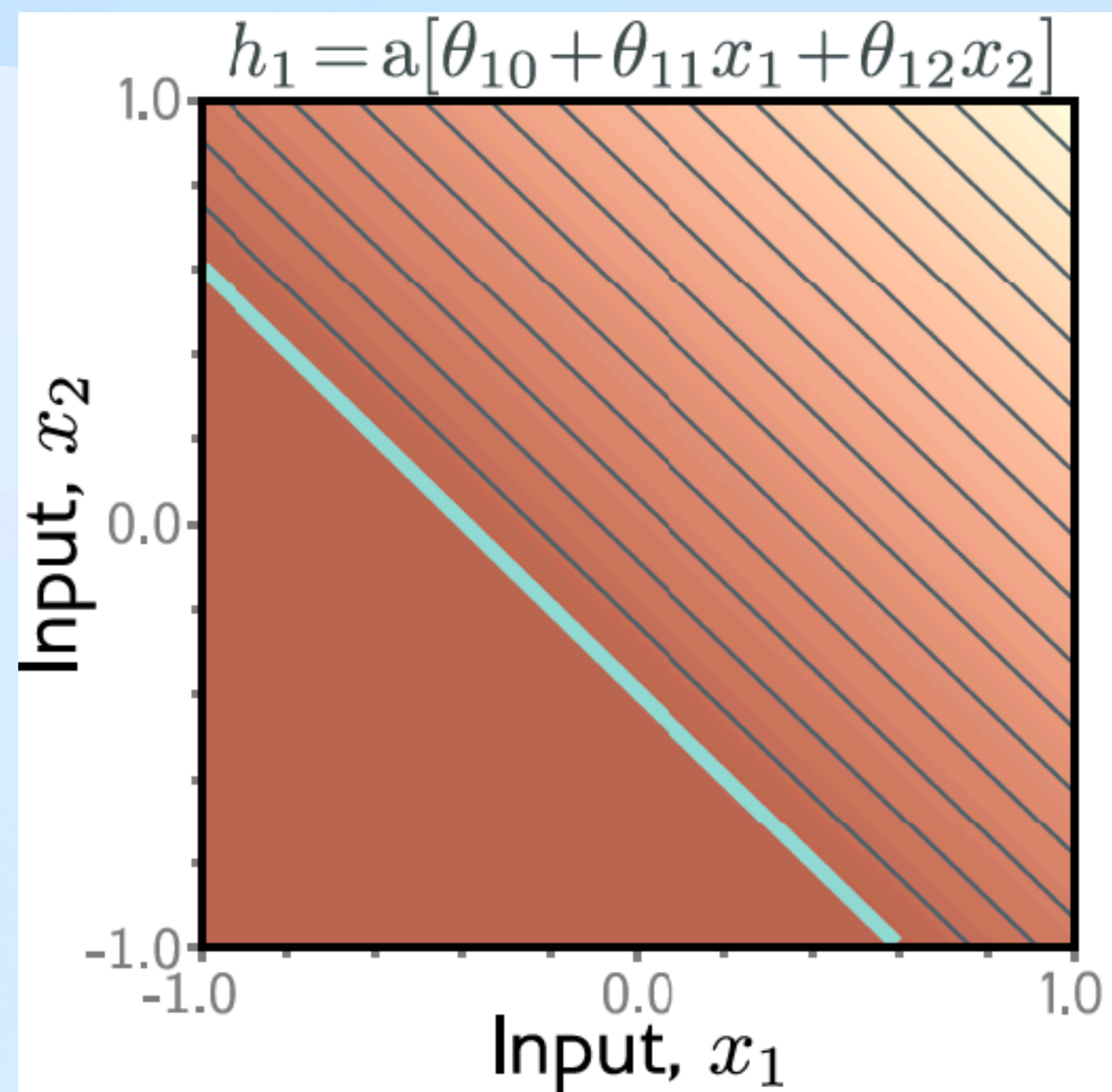
$\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2$
1.0
0.0
-1.0
Input, x_2
-1.0
0.0
1.0
Input, x_1



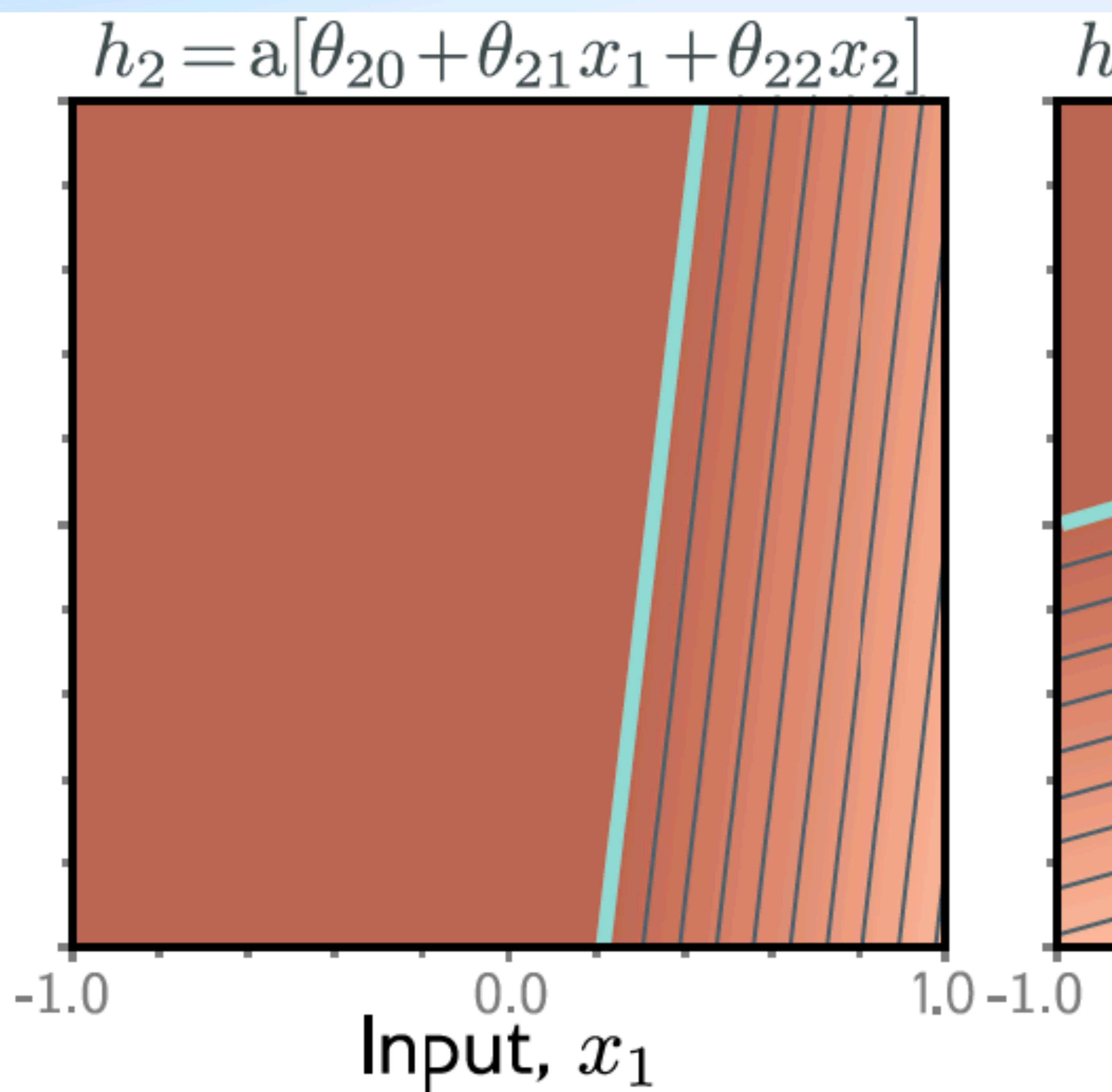
$\theta_{20} + \theta_{21}x_1 + \theta_{22}x_2$
-1.0
0.0
1.0
Input, x_1



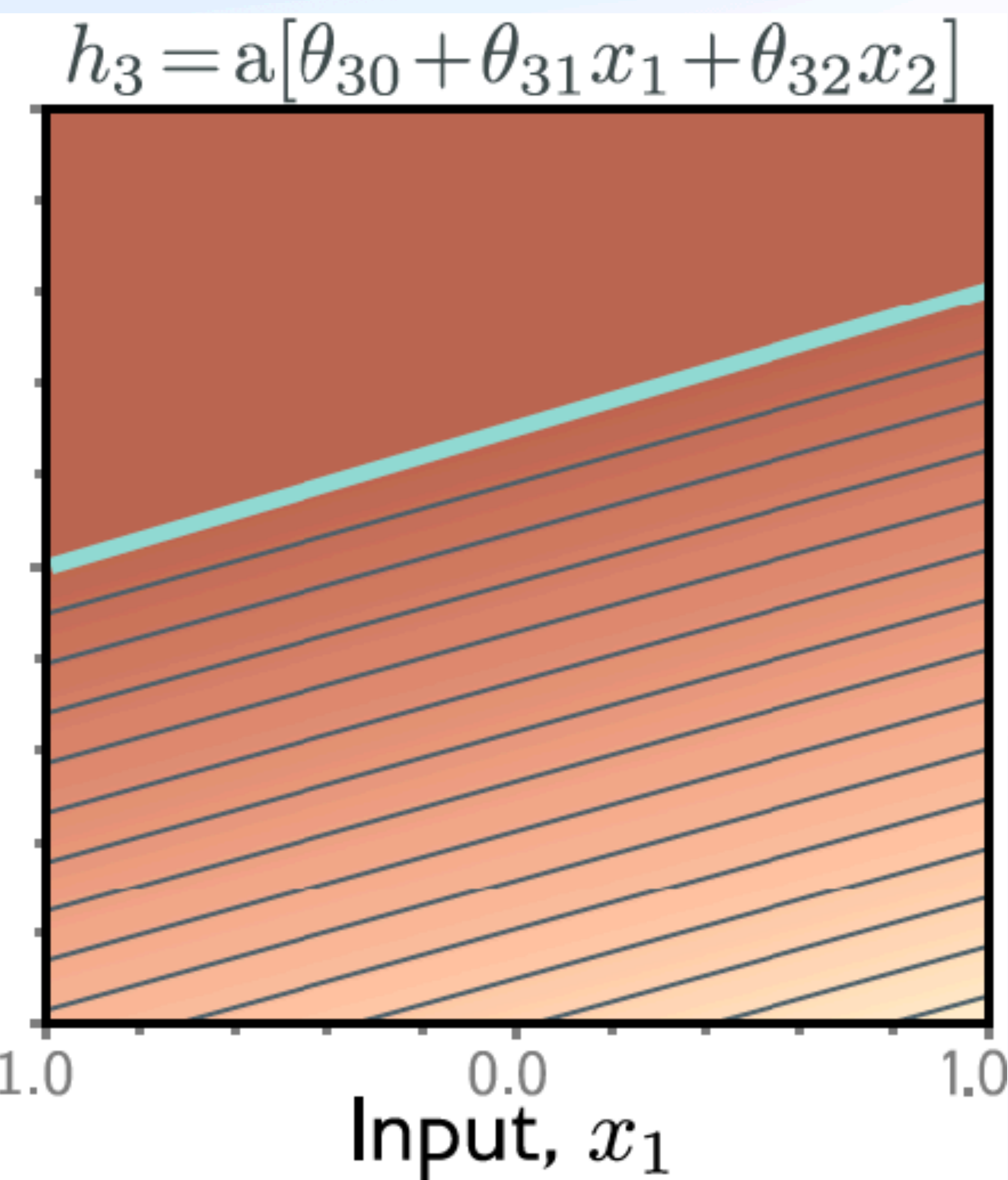
$\theta_{30} + \theta_{31}x_1 + \theta_{32}x_2$
-1.0
0.0
1.0
Input, x_1



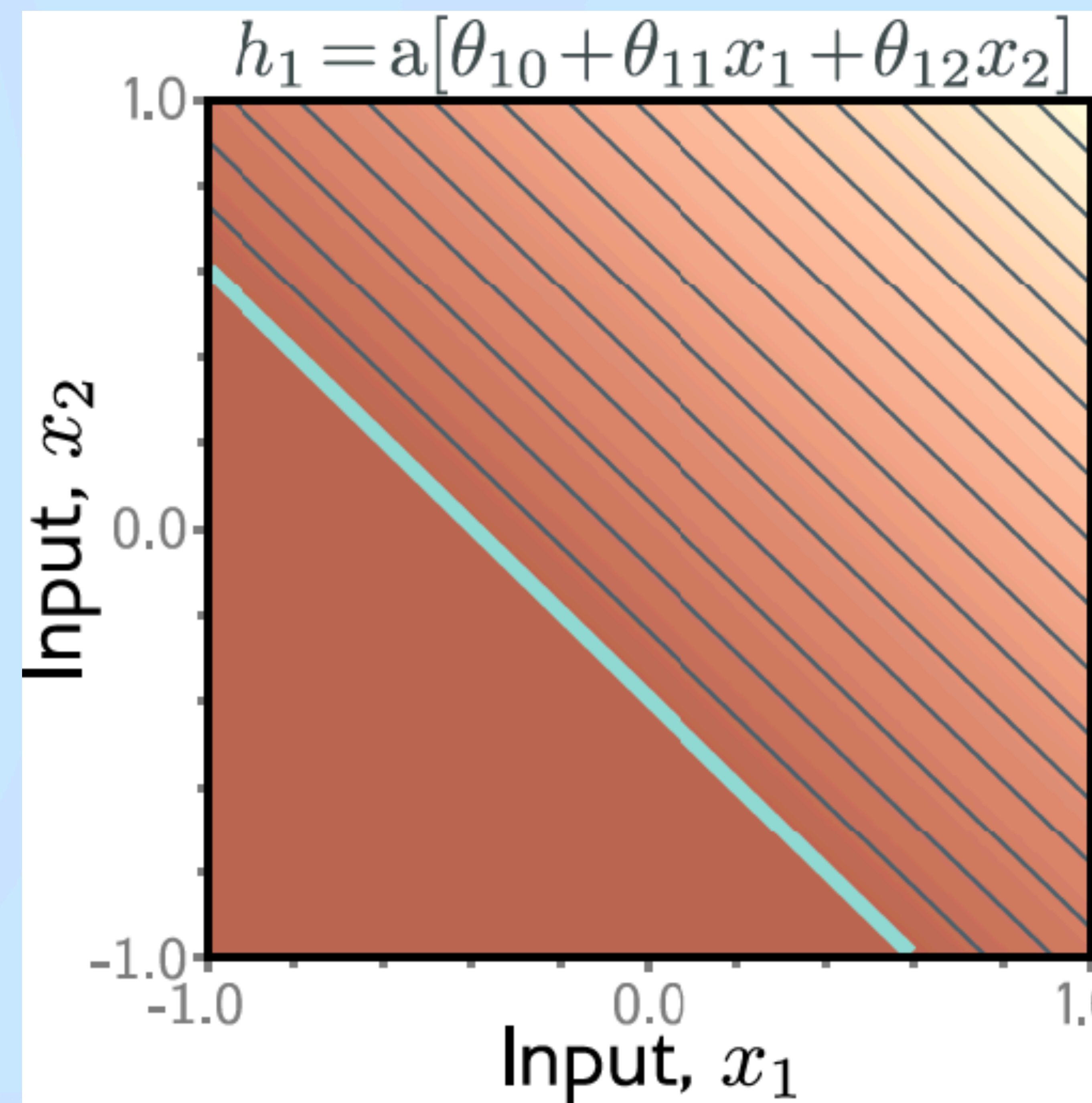
$h_1 = \mathrm{a}[\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2]$
1.0
0.0
-1.0
Input, x_2
-1.0
0.0
1.0
Input, x_1

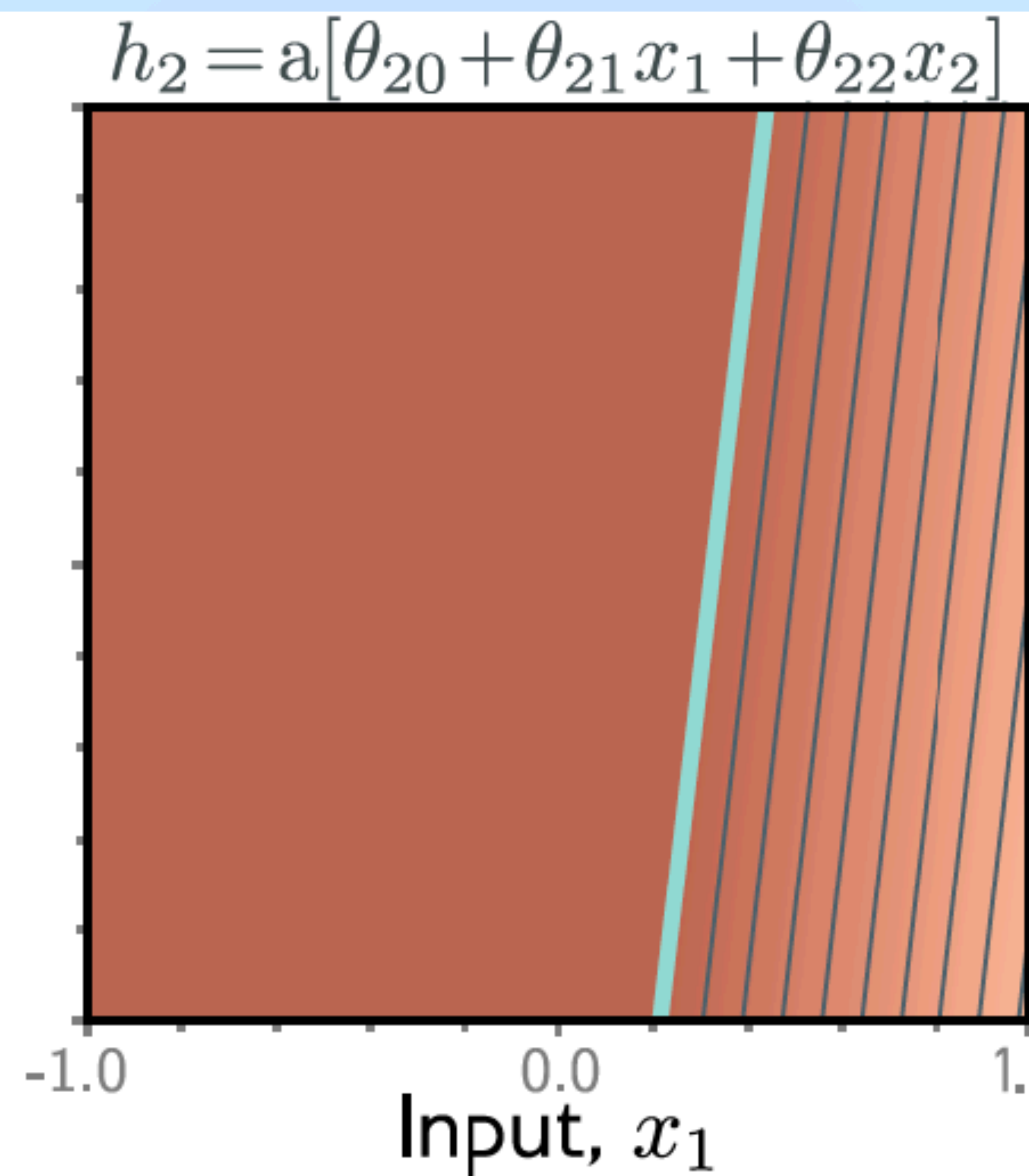


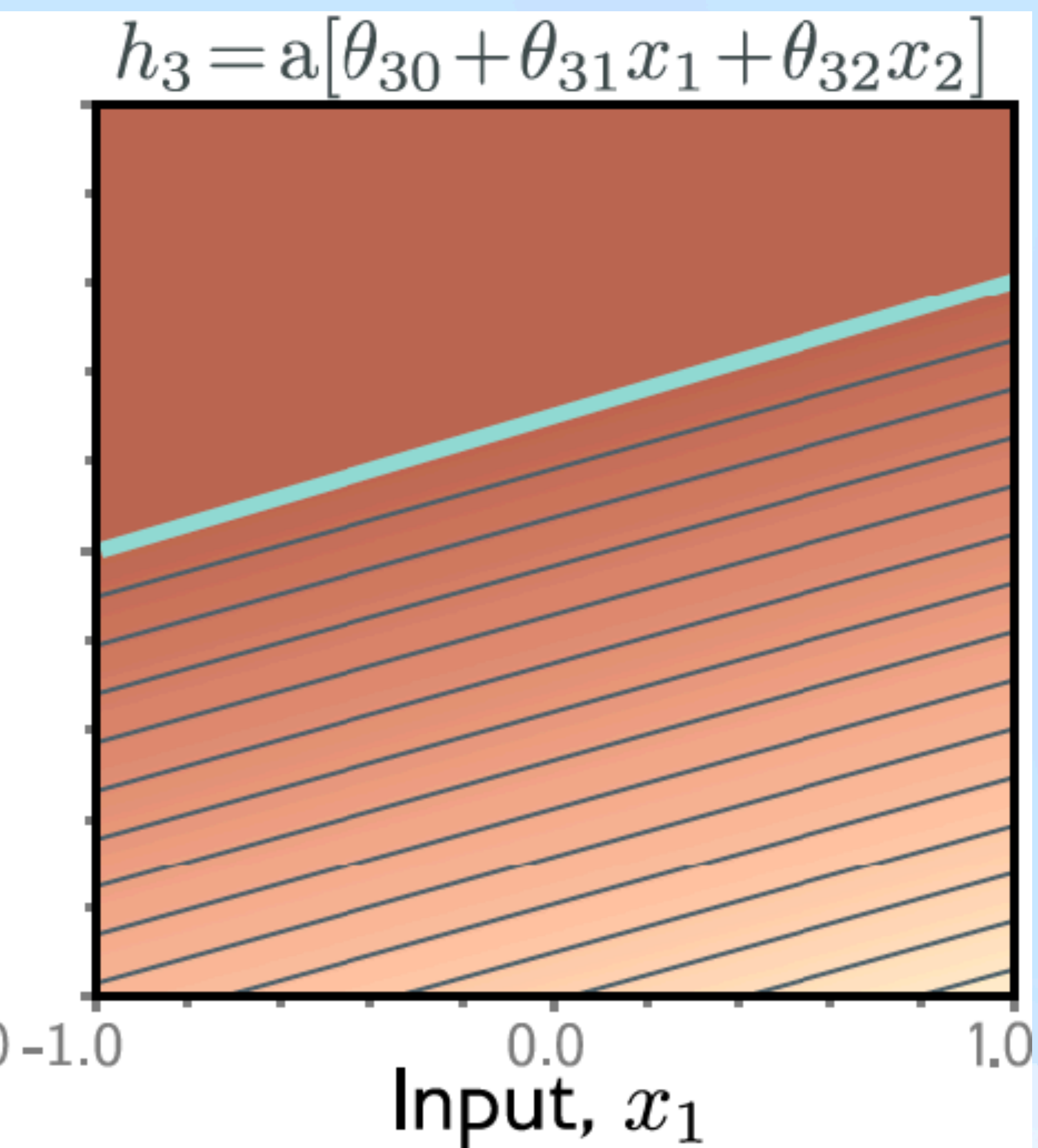
$h_2 = \mathrm{a}[\theta_{20} + \theta_{21}x_1 + \theta_{22}x_2]$
-1.0
0.0
1.0
Input, x_1

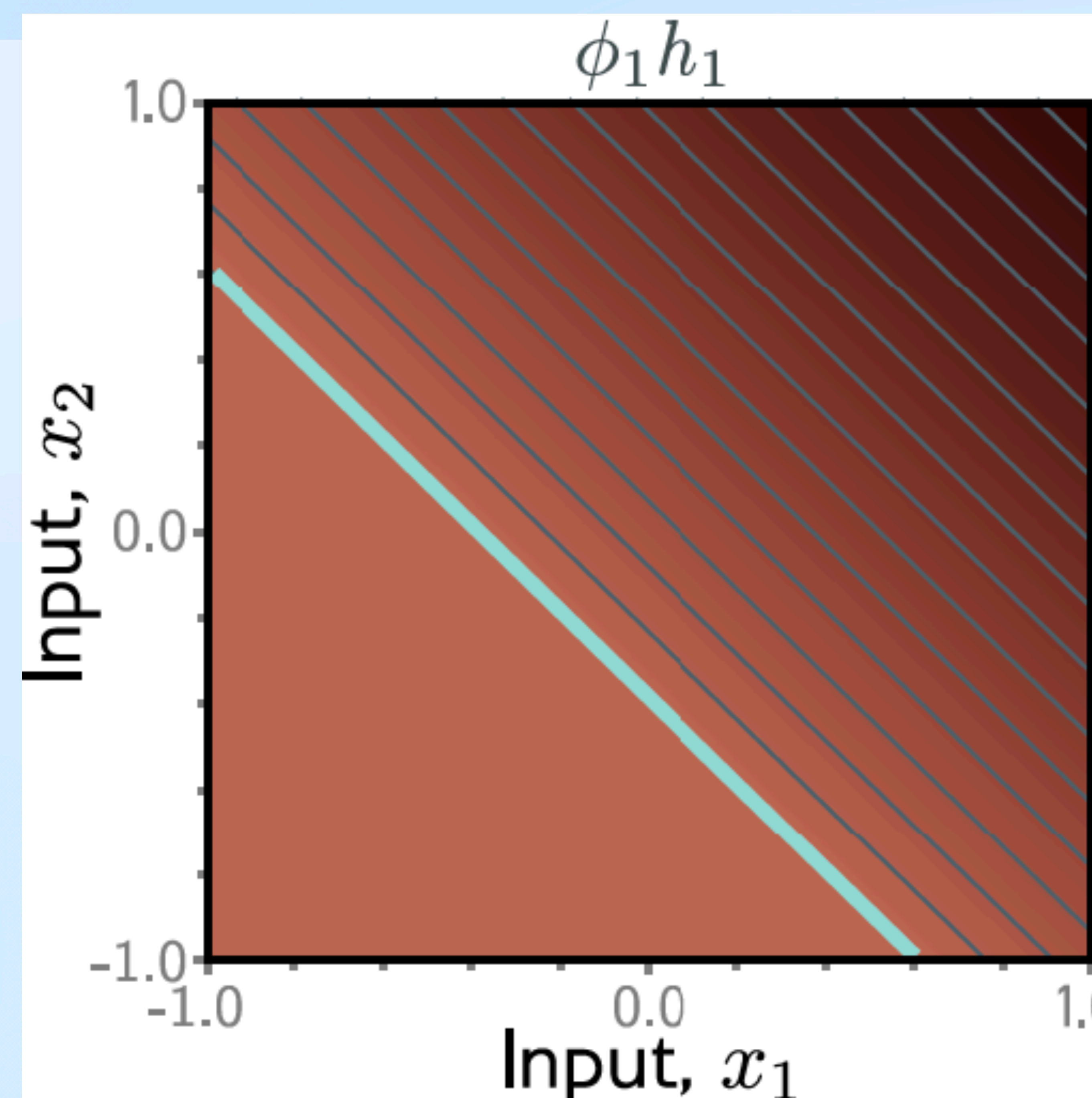


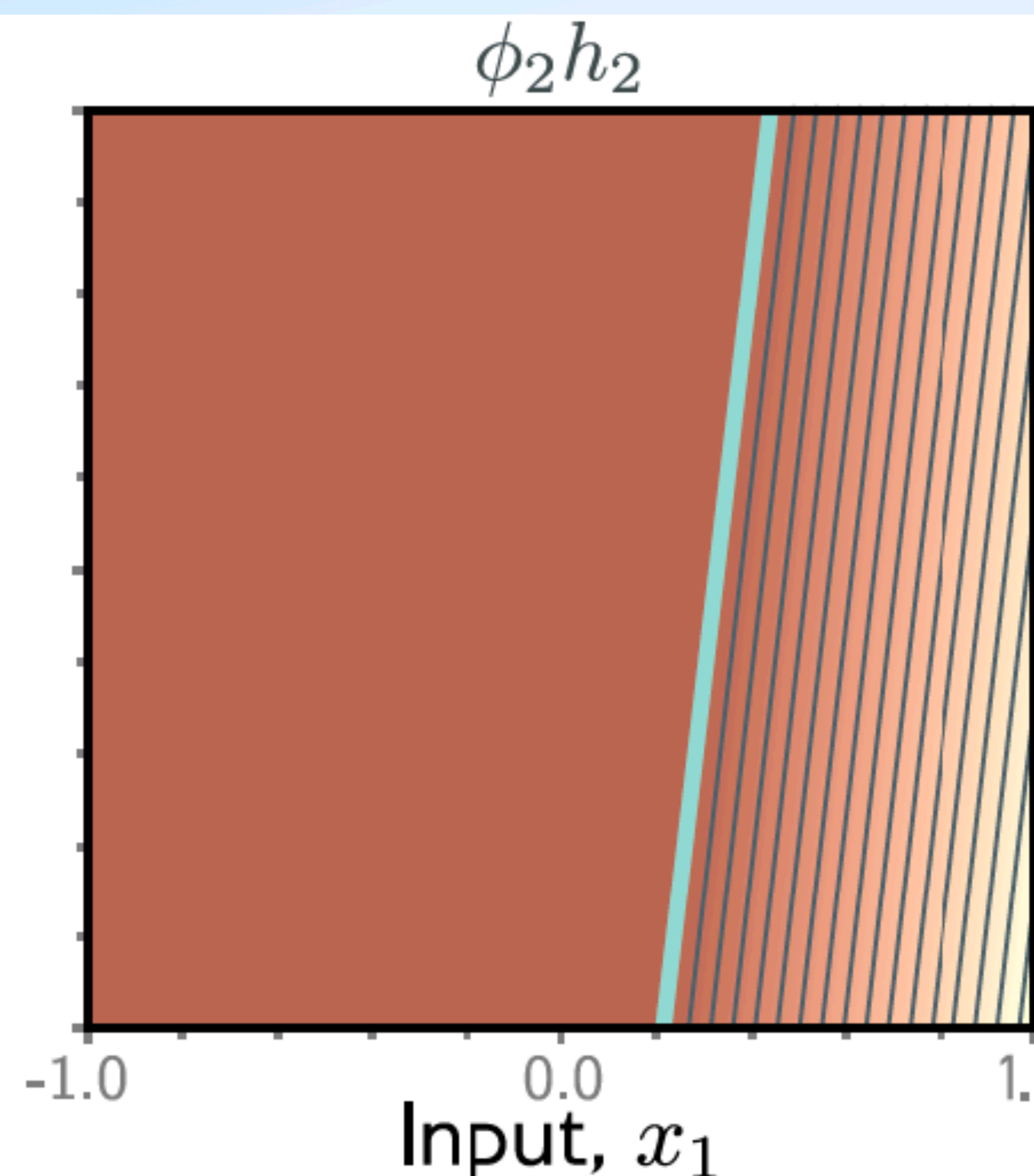
$h_3 = \mathrm{a}[\theta_{30} + \theta_{31}x_1 + \theta_{32}x_2]$
-1.0
0.0
1.0
Input, x_1

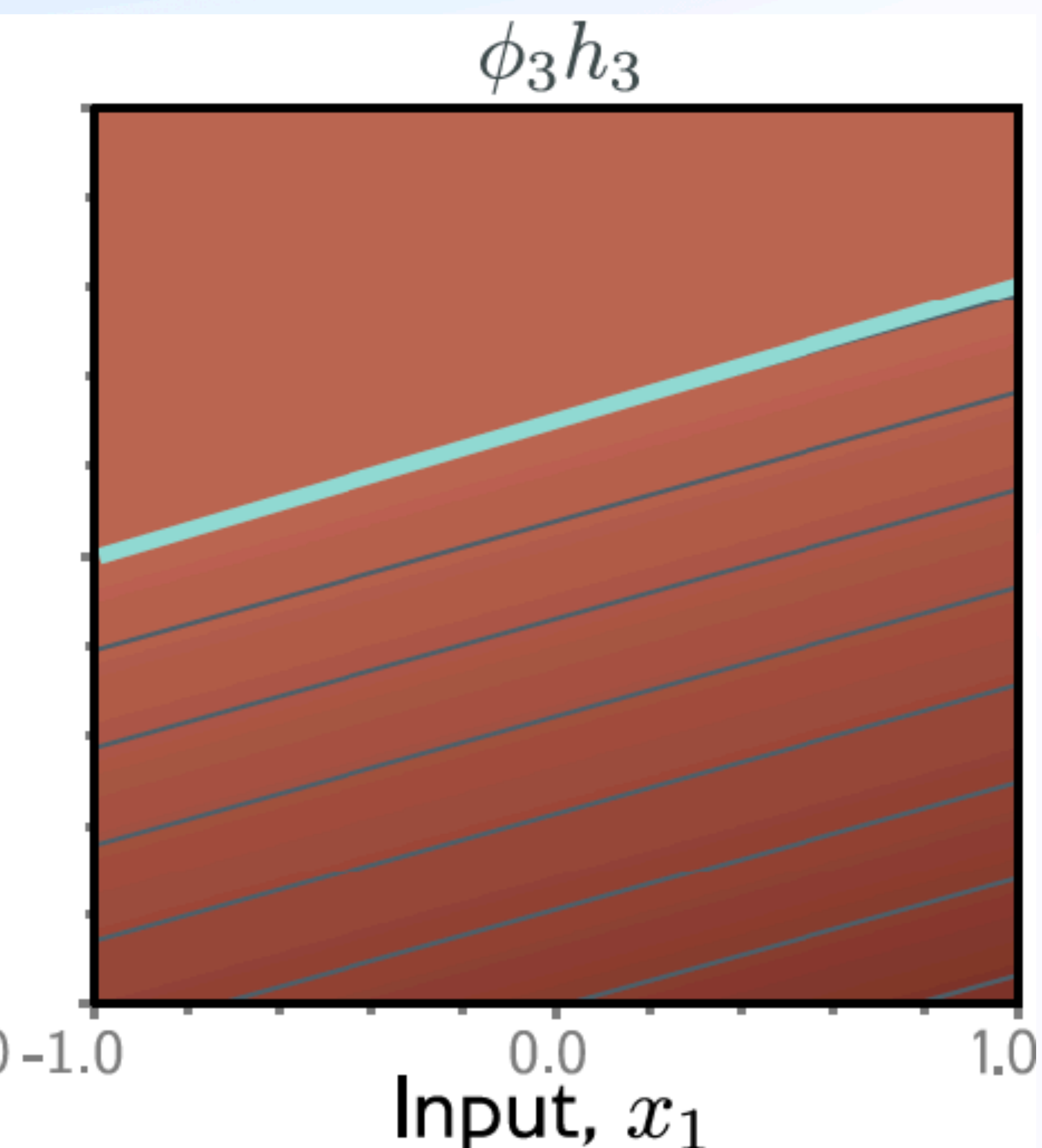

$h_1 = a[\theta_{10} + \theta_{11}x_1 + \theta_{12}x_2]$
Input, x_2
Input, x_1
1.0
0.0
-1.0


$h_2 = a[\theta_{20} + \theta_{21}x_1 + \theta_{22}x_2]$
Input, x_1
-1.0
0.0
1.0


$h_3 = a[\theta_{30} + \theta_{31}x_1 + \theta_{32}x_2]$
Input, x_1
-1.0
0.0
1.0


$\phi_1 h_1$
Input, x_2
Input, x_1
1.0
0.0
-1.0

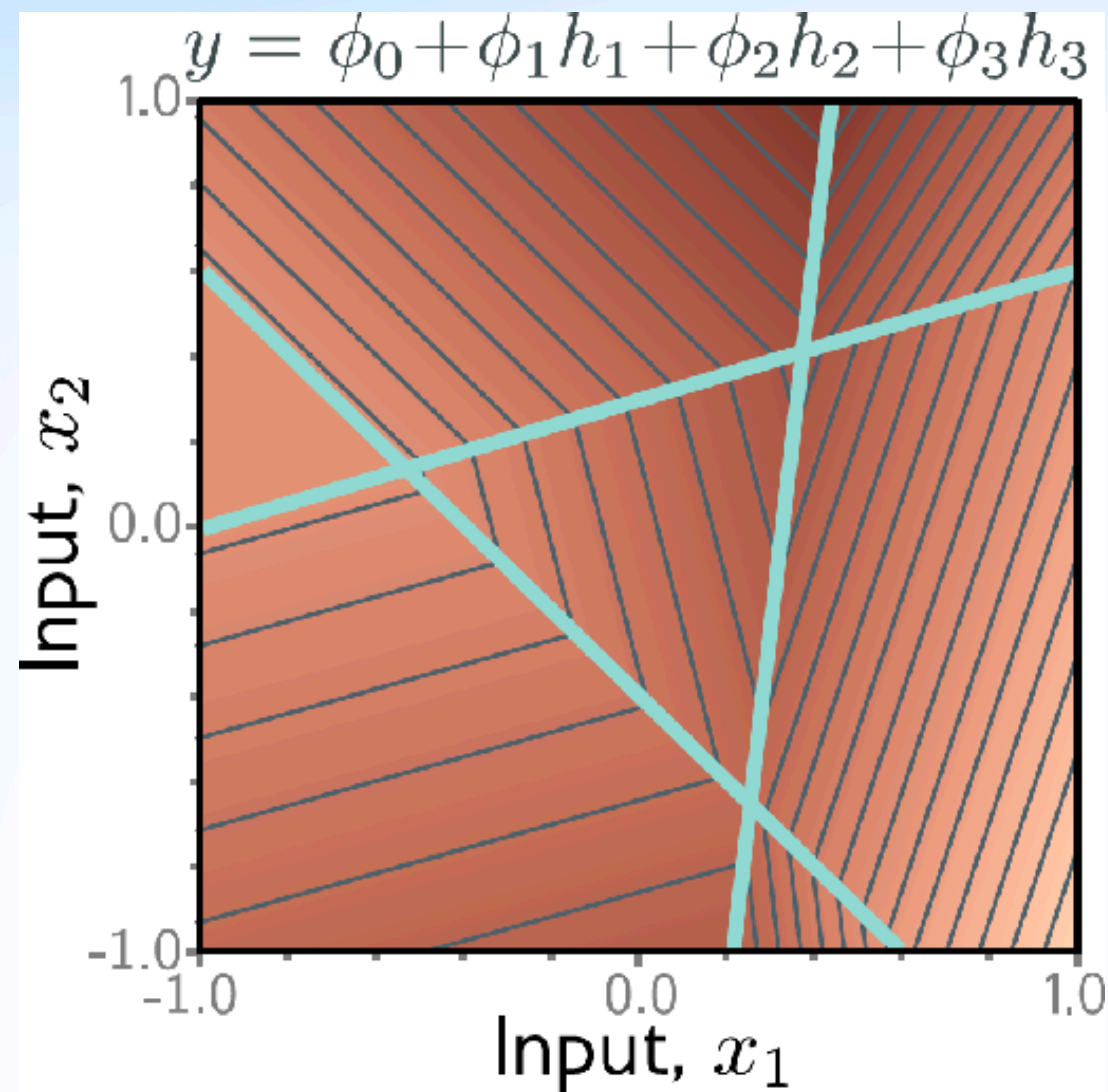

$\phi_2 h_2$
Input, x_1
-1.0
0.0
1.0


$\phi_3 h_3$
Input, x_1
-1.0
0.0
1.0

$\phi_1 h_1$ $\phi_2 h_2$ $\phi_3 h_3$

Input, x_2

Input, x_1 Input, x_1 Input, x_1



Convex polygons

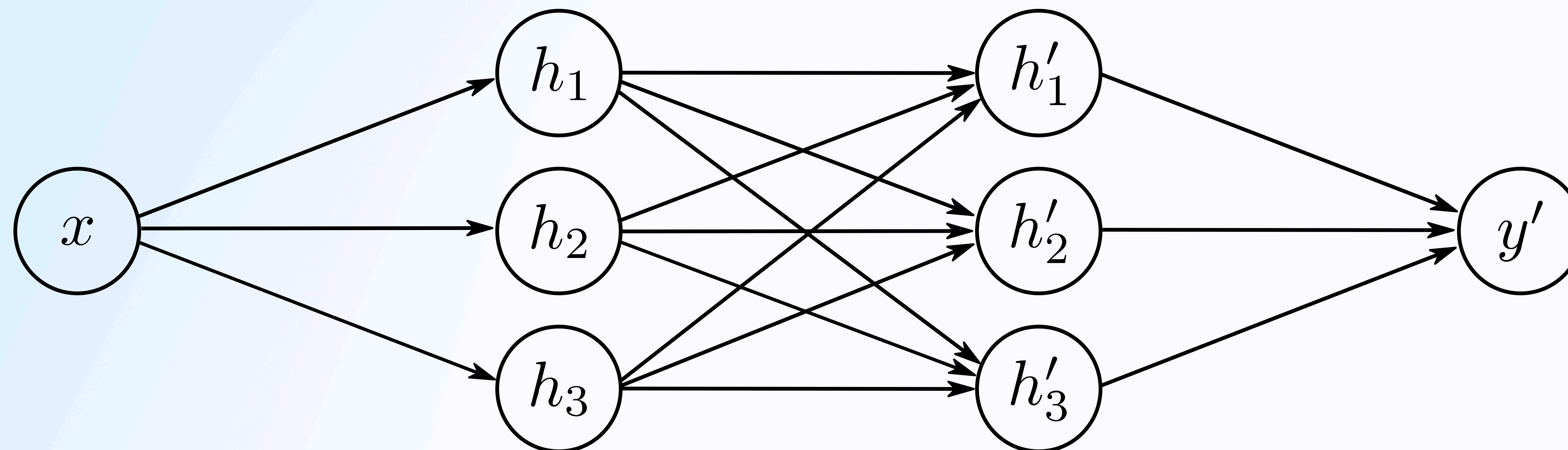
Interactive visualisation [here]

Deep Neural networks : Adding hidden layers

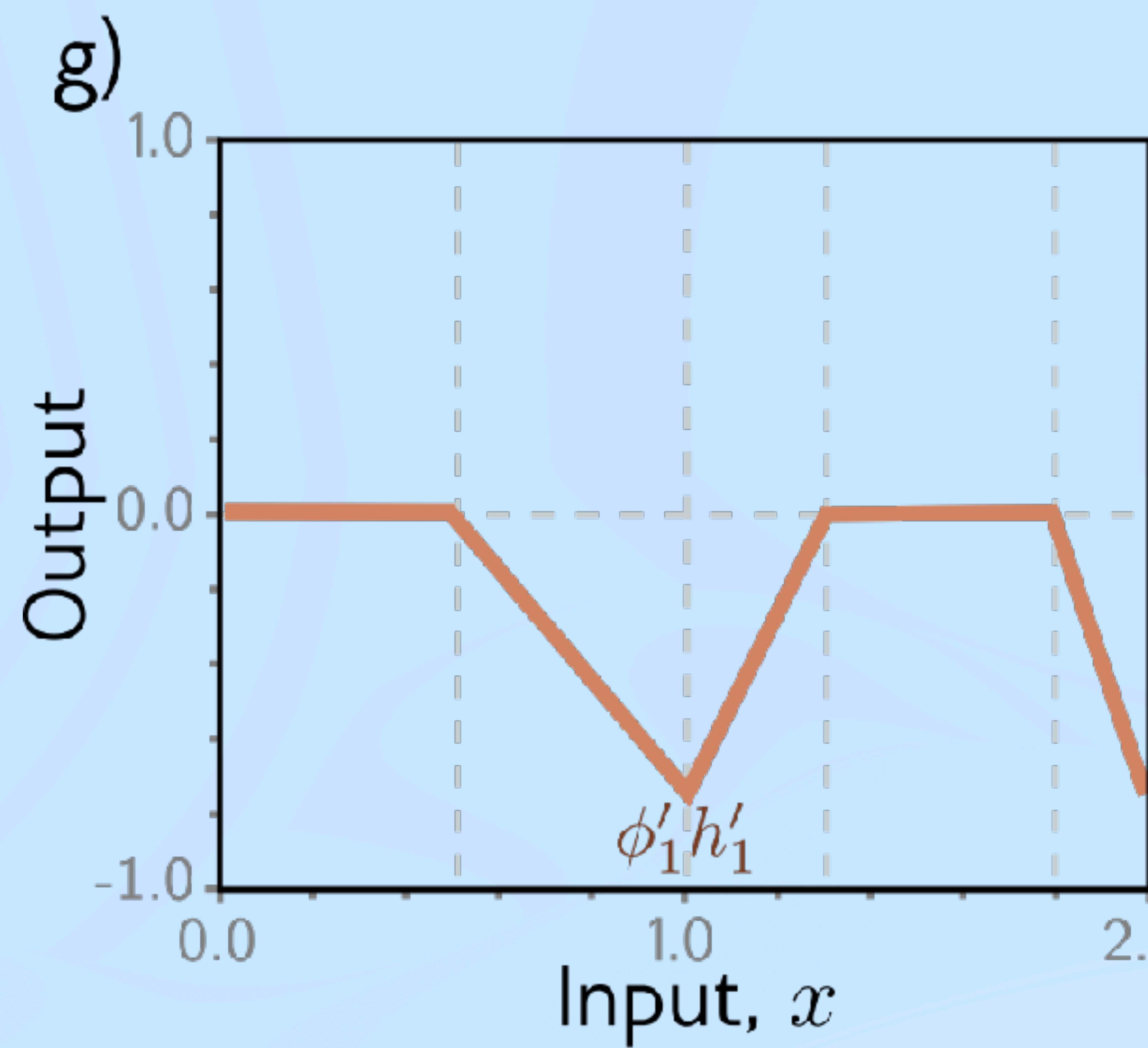
$$h_1 = \text{a}[\theta_{10} + \theta_{11}x]$$
$$h_2 = \text{a}[\theta_{20} + \theta_{21}x]$$
$$h_3 = \text{a}[\theta_{30} + \theta_{31}x]$$

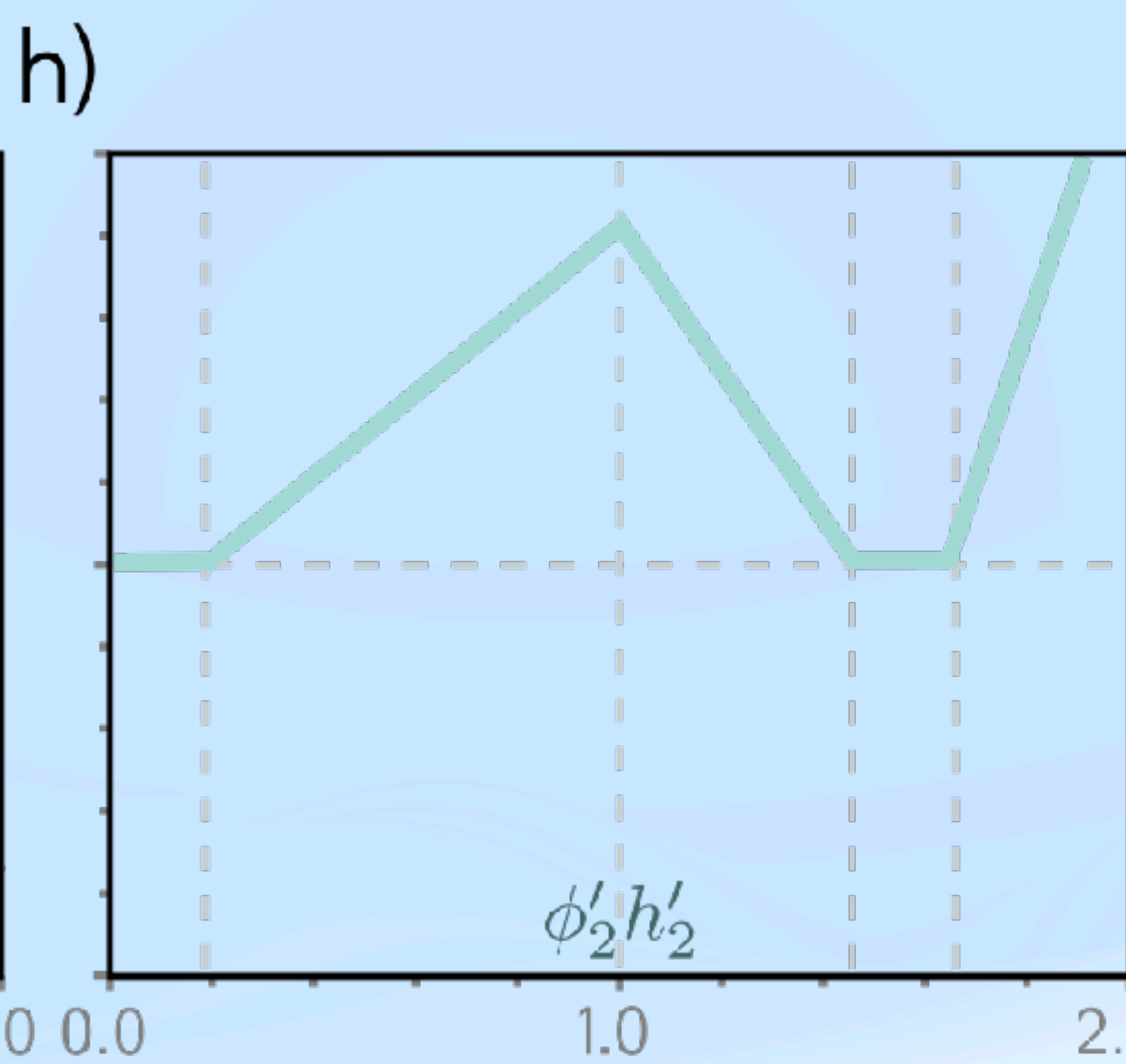
$$h'_1 = \text{a}[\psi_{10} + \psi_{11}h_1 + \psi_{12}h_2 + \psi_{13}h_3]$$
$$h'_2 = \text{a}[\psi_{20} + \psi_{21}h_2 + \psi_{22}h_2 + \psi_{23}h_3]$$
$$h'_3 = \text{a}[\psi_{30} + \psi_{31}h_2 + \psi_{32}h_2 + \psi_{33}h_3]$$

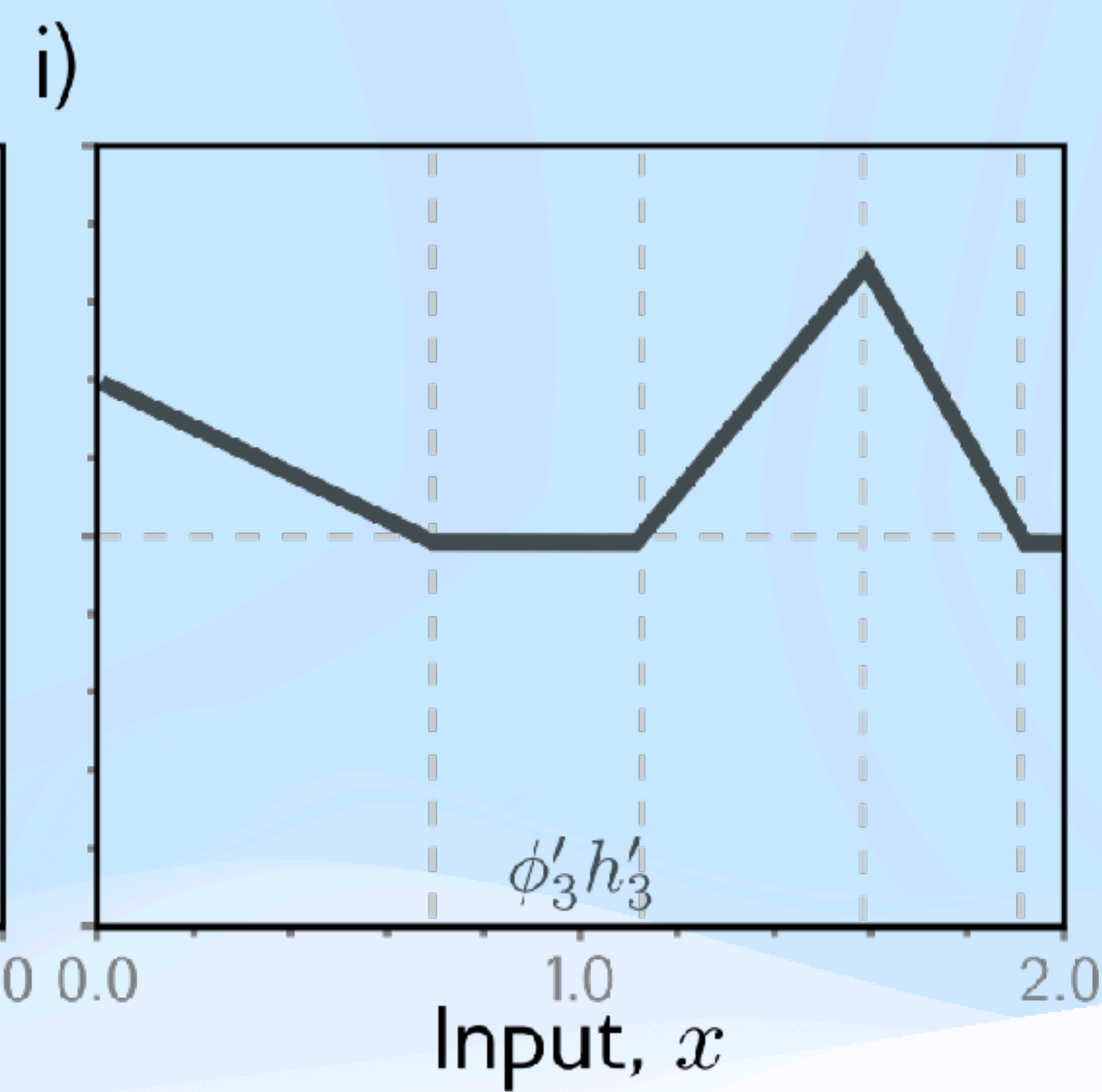
$$y' = \phi'_0 + \phi'_1 h'_1 + \phi'_2 h'_2 + \phi'_3 h'_3$$

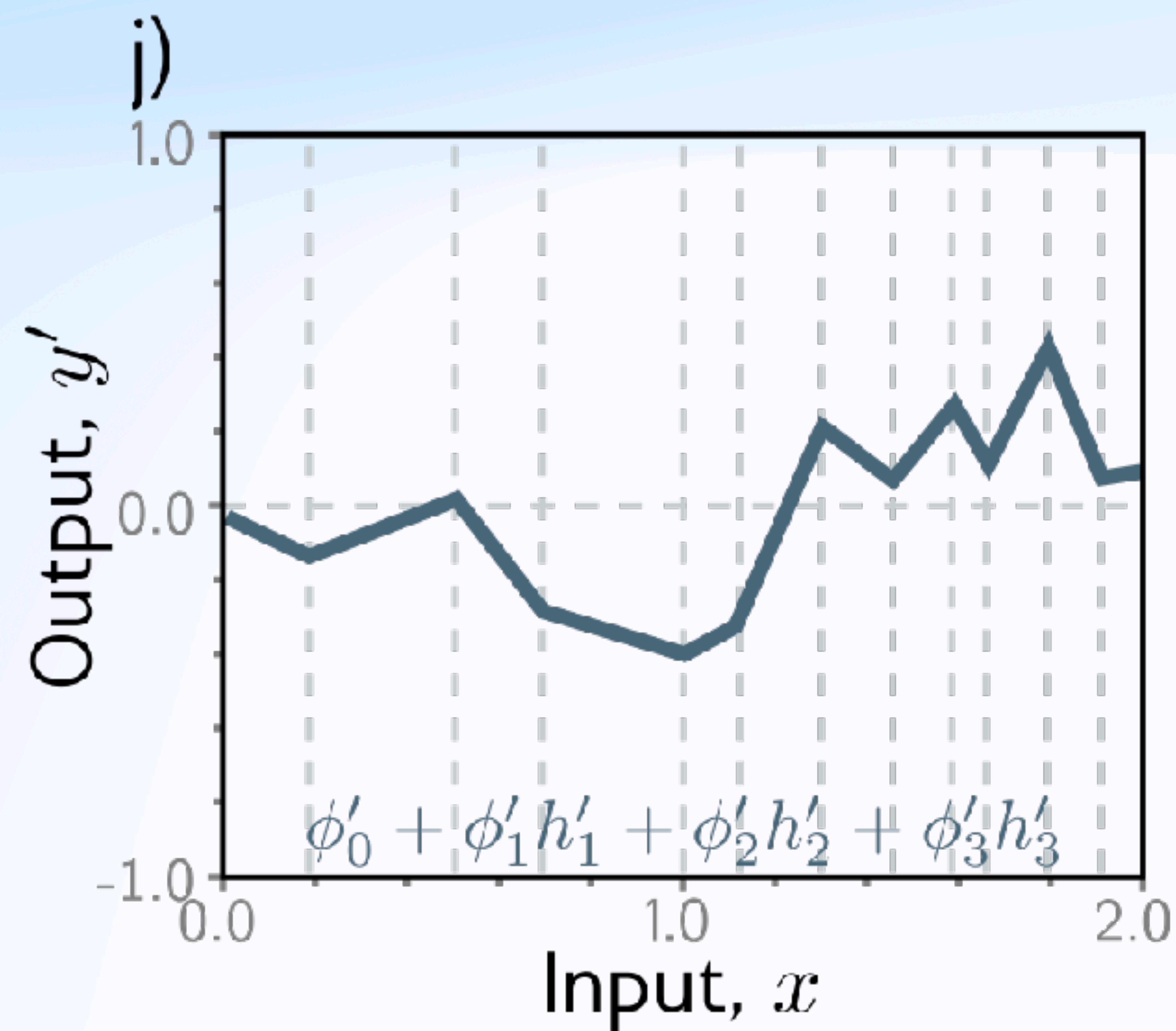


Antonin Vacheret


g)
1.0
0.0
-1.0
Output
$\phi'_1 h'_1$
0.0
1.0
2.0
Input, x


h)
$\phi'_2 h'_2$
0.0
1.0
2.0

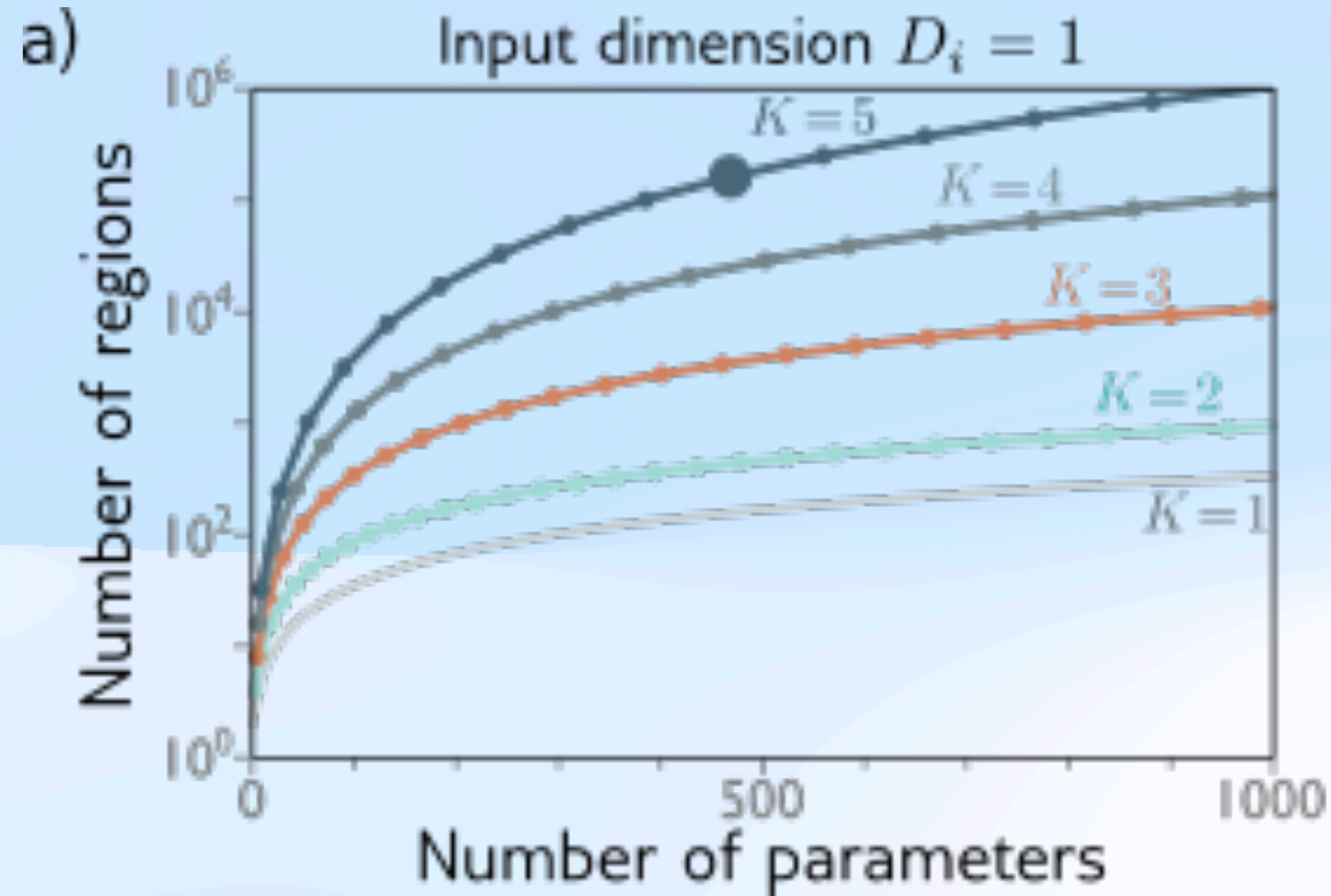

i)
$\phi'_3 h'_3$
0.0
1.0
2.0
Input, x

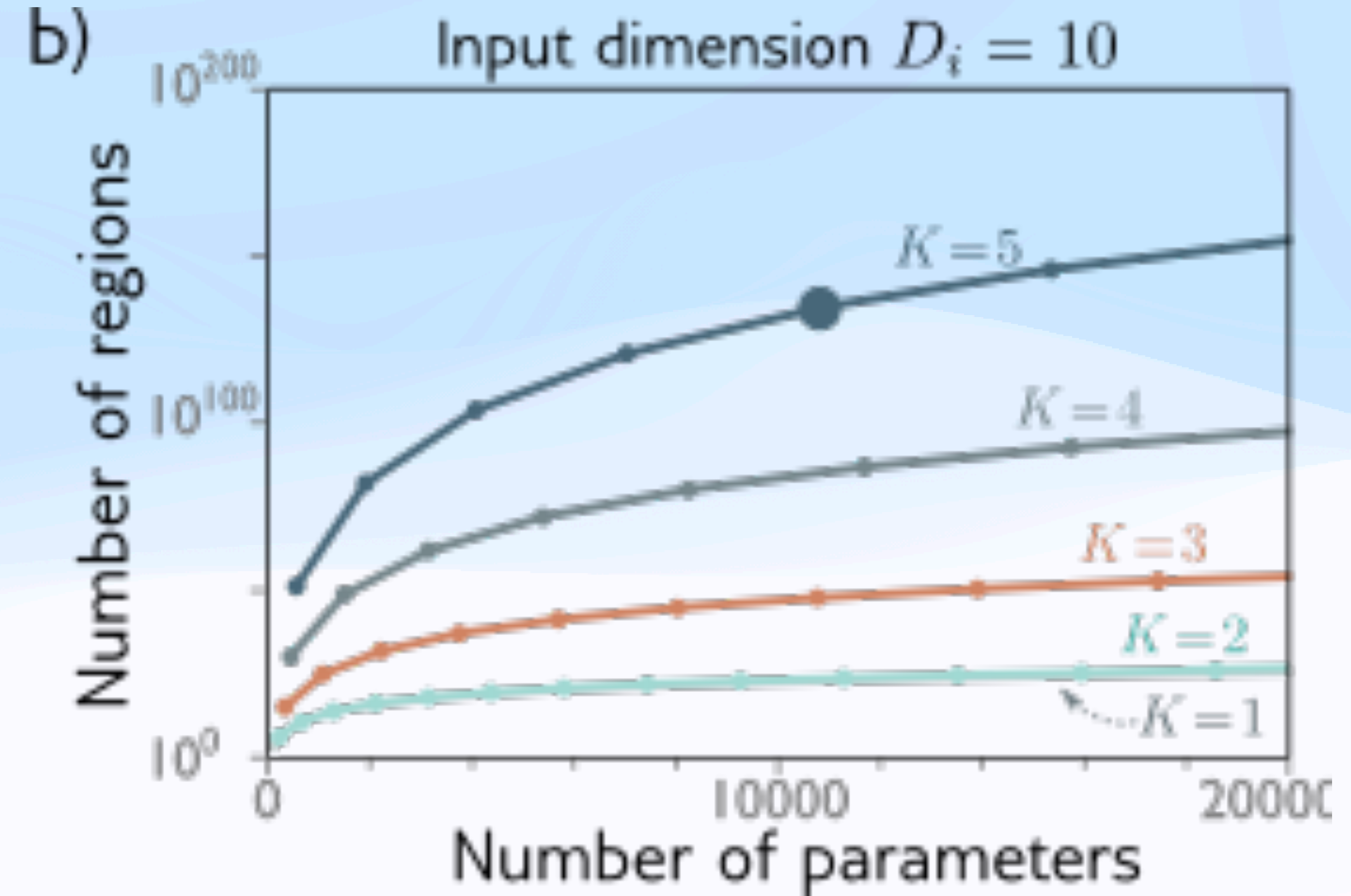

j)
1.0
0.0
-1.0
Output, y'
$\phi'_0 + \phi'_1 h'_1 + \phi'_2 h'_2 + \phi'_3 h'_3$
0.0
1.0
2.0
Input, x

Visualising the piecewise representation of a deep network

- Structure of data emerges as function of N layers (3 in the middle)
- Bright colour represents high activation of layer units or neutrons
  - https://blog.janestreet.com/visualizing-piecewise-linear-neural-networks/ & arXiv 1903.08778

Number of linear regions per parameter

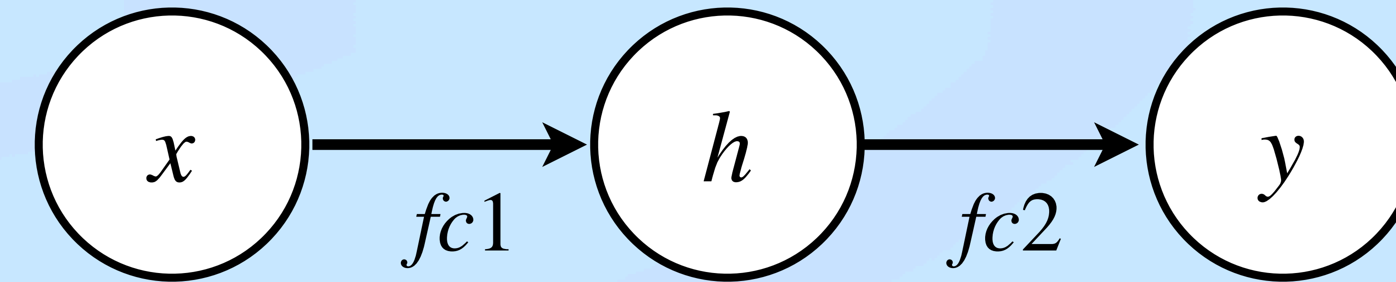




5 layers
10 hidden units per layer
471 parameters
161,501 linear regions

5 layers
50 hidden units per layer
10,801 parameters
$>10^{134}$ linear regions

NN implementation in Pytorch




```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class ShallowNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(ShallowNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# Example usage
input_size = 10
hidden_size = 5
output_size = 2
model = ShallowNN(input_size, hidden_size, output_size)
print(model)
```

How many layers ?

```python
class DeepNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(DeepNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, hidden_size)
        self.fc3 = nn.Linear(hidden_size, hidden_size)
        self.fc4 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.fc4(x)
        return x

# Example usage
model = DeepNN(input_size, hidden_size, output_size)
print(model)
```

Supervised learning training

Supervised learning example tasks

- Many inverse problems can be solved using supervised learning :
  - Event reconstruction tasks
    - Reconstruct the energy of particles using detector information on dEdx
    - Parameter inference
- Classification tasks
  - Event topology recognition
  - Particule identification

Supervised learning workflow

- Supervised learning requires training data
- A forward pass is called inference
- A backward pass is used to modify the weights of the hidden units
- We need
  - An objective function called **the loss** to provide a criterion for training.
  - An algorithm that compute the gradient for each step and guide the training minimising the loss by updating the weights of NN: this is called an **optimiser** and the method is **gradient descent**.
  - Additional **metrics** (if needed) to decide when to stop the training loop
    - For example we may want to use accuracy for classification task which counts the number of good "signal" classification

Supervised learning workflow

Task and metrics defined

Datasets with labels

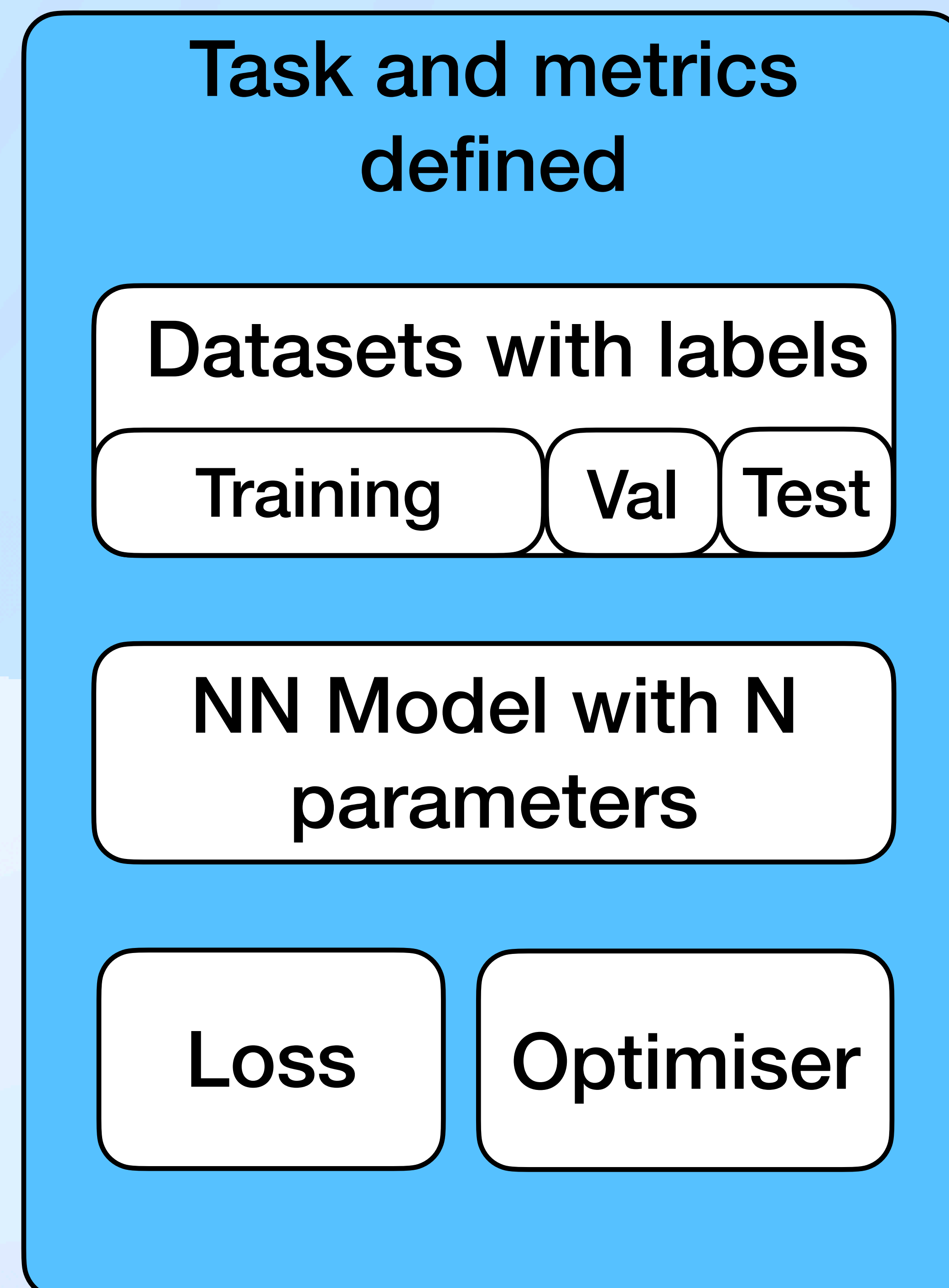
Training | Val | Test

NN Model with N parameters

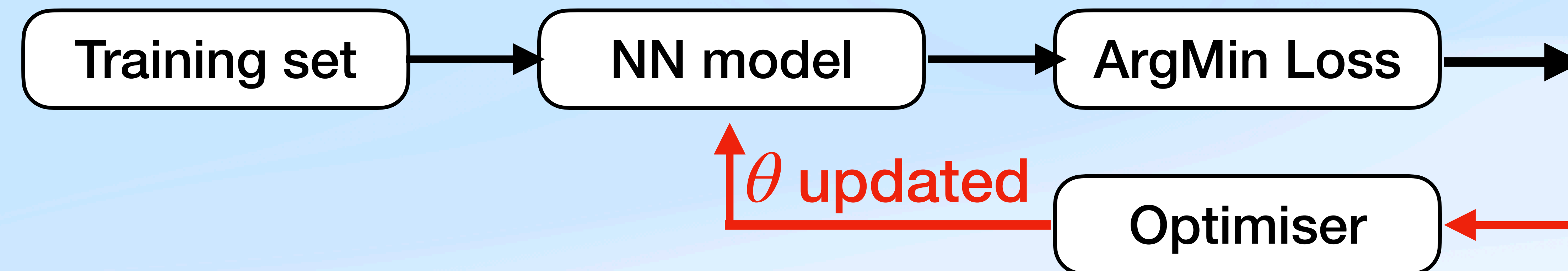
Loss

Optimiser

Supervised learning workflow

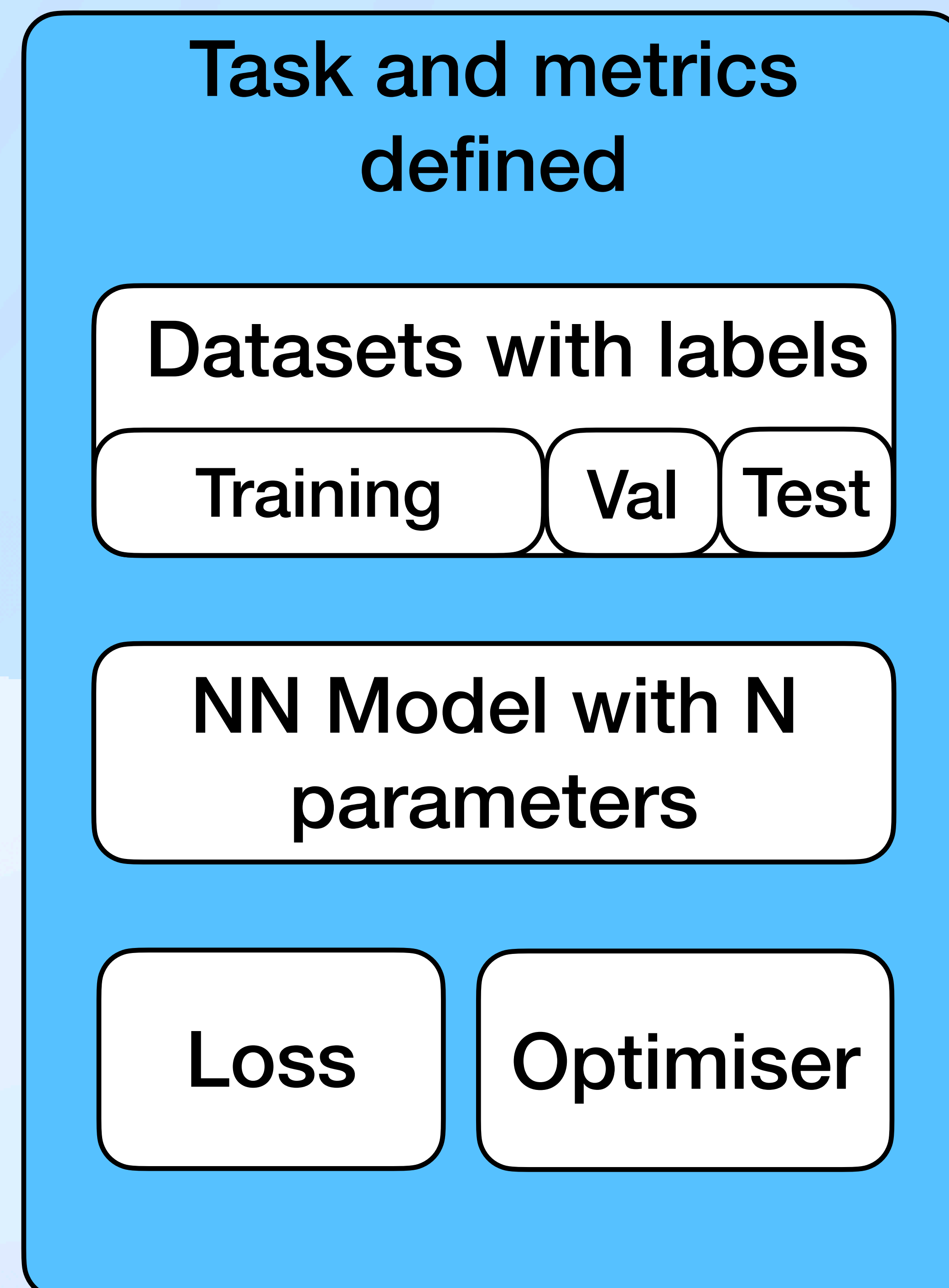


Training phase : forward and backward pass to minimise the loss and find the parameters θ that fits best the label

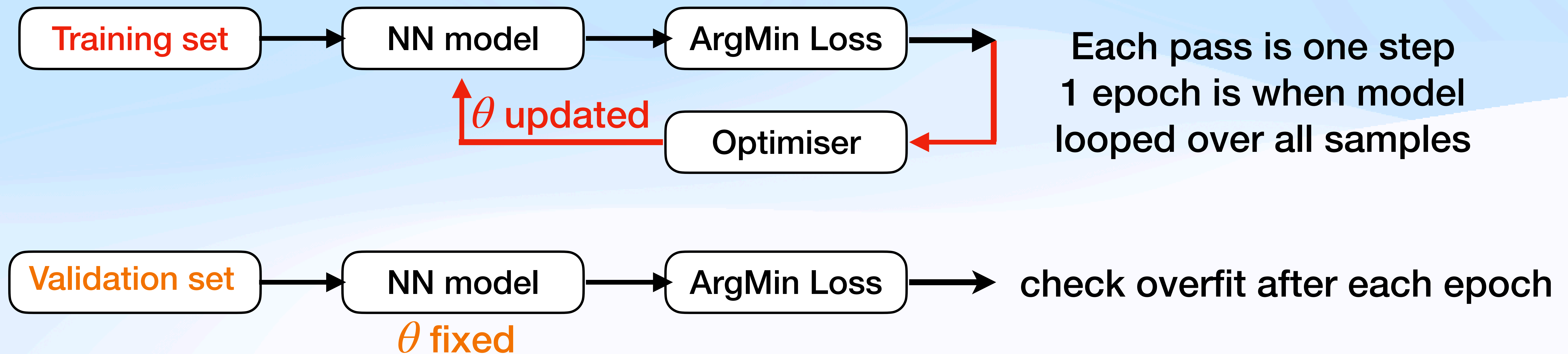


Each pass is one step
1 epoch is when model looped over all samples

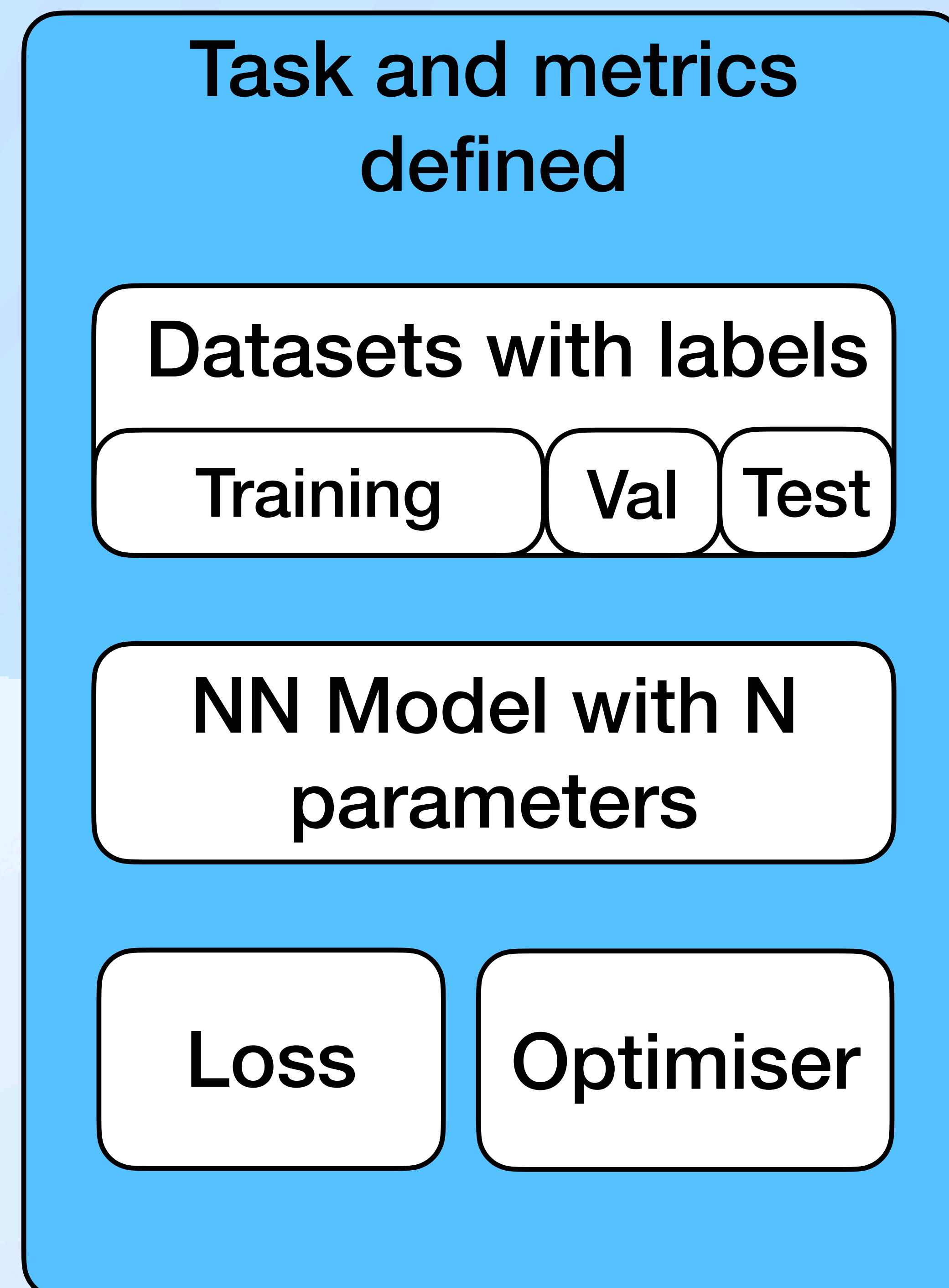
Supervised learning workflow



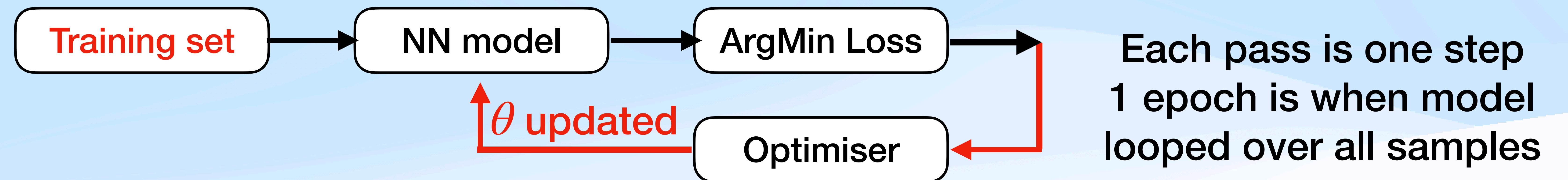
Training phase : forward and backpropagation to minimise the loss and find the parameters θ that fits best the label



Supervised learning workflow



Training phase : forward and backpropagation to minimise the loss and find the parameters θ that fits best the label



Testing phase : θ parameters are fixed, evaluates the model on unseen data

- Every time data diverge from training data, the model needs to be retrained !

Loss functions

- ~~Model predicts output *y* given input *x*~~
- Model predicts a conditional probability distribution:

$$Pr(\mathbf{y}|\mathbf{x})$$

over outputs y given inputs x.

- **Loss function aims to make the outputs have high probability**

Recipe for loss functions

1. Choose a suitable probability distribution $Pr(\mathbf{y}|\boldsymbol{\theta})$ that is defined over the domain of the predictions $\mathbf{y}$ and has distribution parameters $\boldsymbol{\theta}$.

Recipe for loss functions

1. Choose a suitable probability distribution $Pr(\mathbf{y}|\boldsymbol{\theta})$ that is defined over the domain of the predictions $\mathbf{y}$ and has distribution parameters $\boldsymbol{\theta}$.
2. Set the machine learning model $\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ to predict one or more of these parameters so $\boldsymbol{\theta} = \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ and $Pr(\mathbf{y}|\boldsymbol{\theta}) = Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}])$.

Recipe for loss functions

1. Choose a suitable probability distribution $Pr(\mathbf{y}|\boldsymbol{\theta})$ that is defined over the domain of the predictions $\mathbf{y}$ and has distribution parameters $\boldsymbol{\theta}$.
2. Set the machine learning model $\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ to predict one or more of these parameters so $\boldsymbol{\theta} = \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ and $Pr(\mathbf{y}|\boldsymbol{\theta}) = Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}])$.
3. To train the model, find the network parameters $\hat{\boldsymbol{\phi}}$ that minimize the negative log-likelihood loss function over the training dataset pairs $\{\mathbf{x}_i, \mathbf{y}_i\}$:

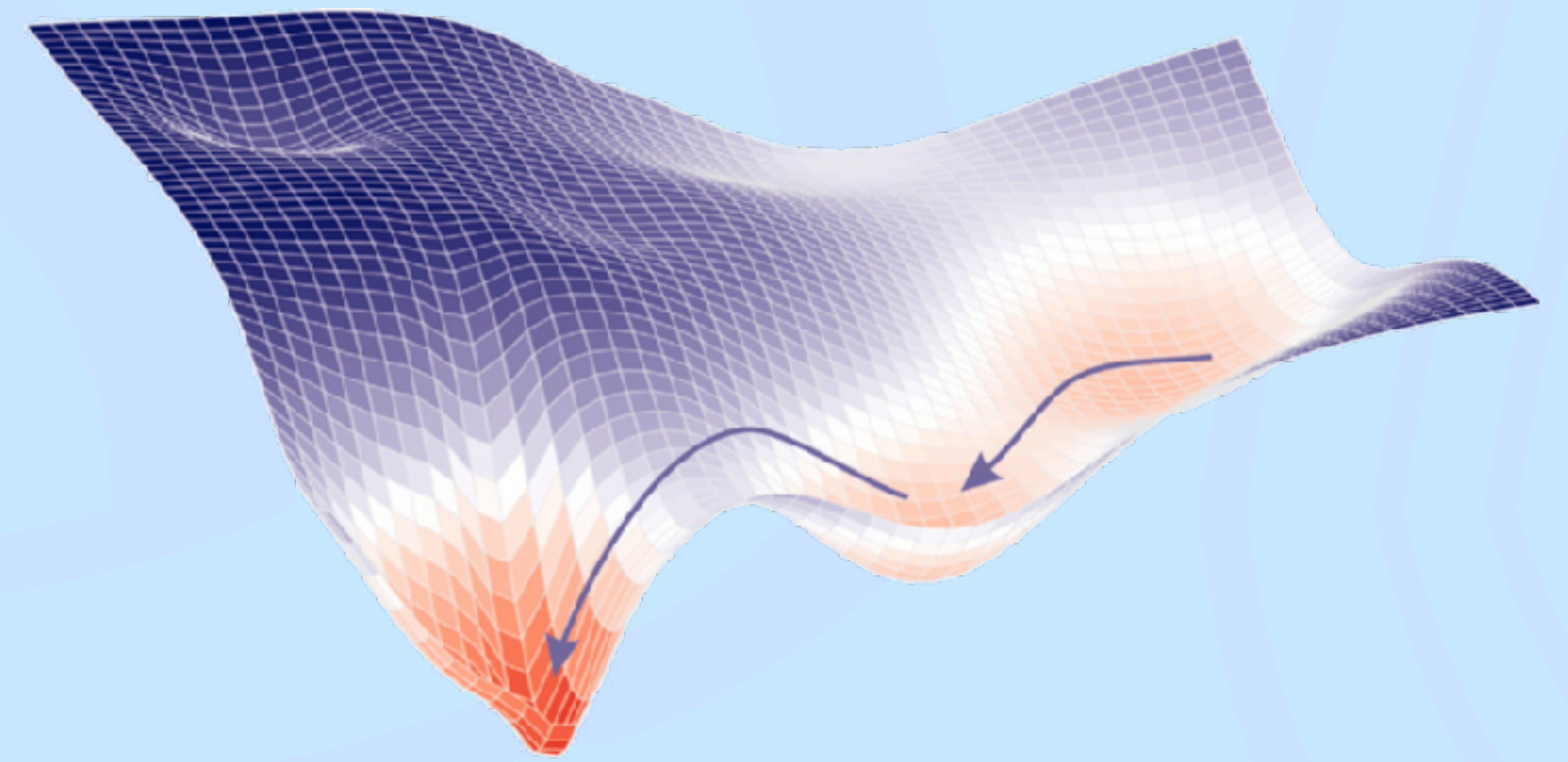
$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[L[\boldsymbol{\phi}] \right] = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log \left[Pr(\mathbf{y}_i | \mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}]) \right] \right]. \tag{5.7}$$

- In practice all we have to do is to choose the correct loss function for the task
  - But in some cases we have to modify or develop a new one

Common Loss functions

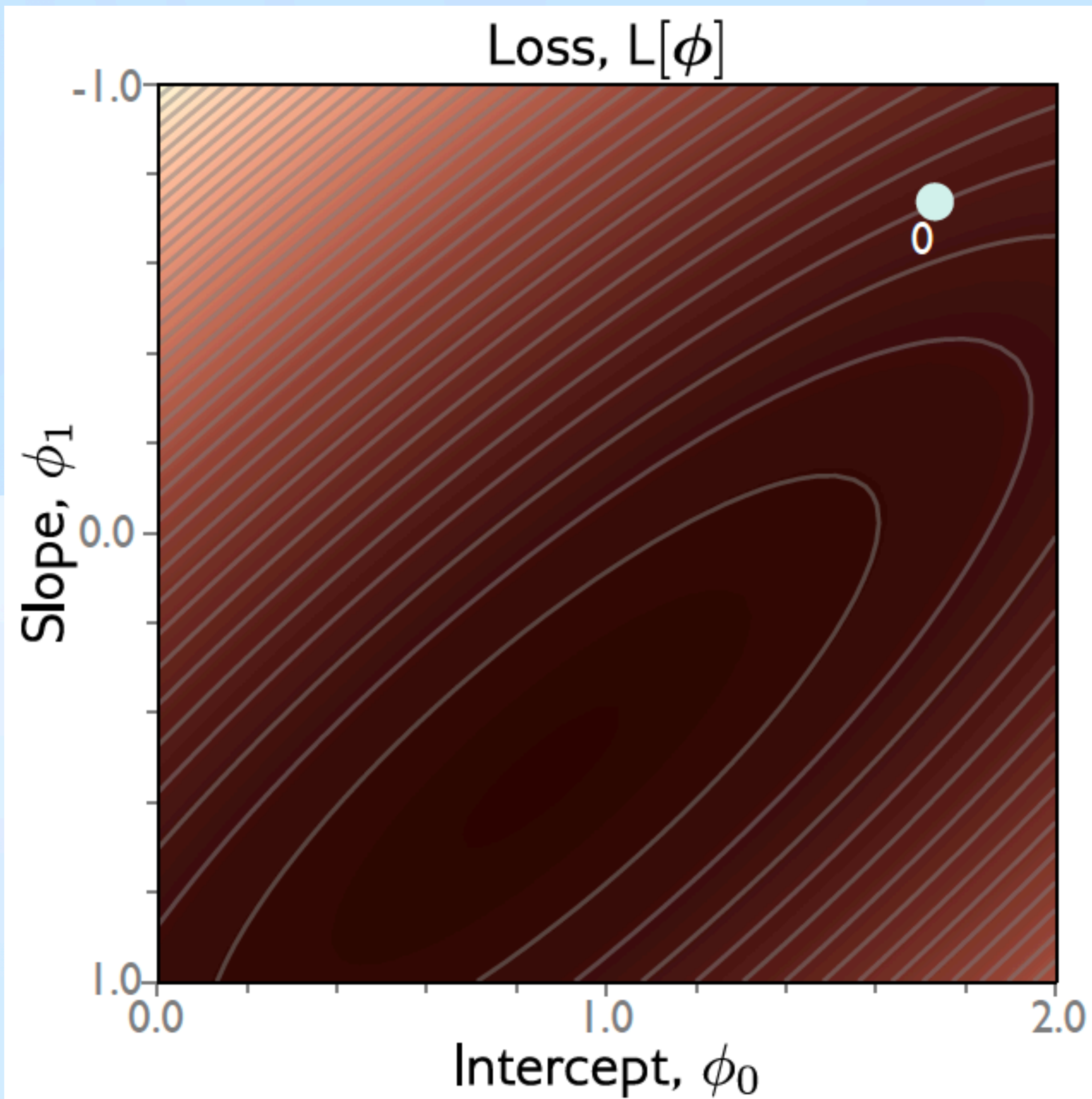
- For classification
  - we use the Bernoulli distribution which gives the binary cross Entropy loss
  - For multiple category classification we use the categorical distribution
    - A softmax function is used to normalise all probabilities
- For Regression typical Loss function are :
  - Mean Square Error (MSE), Mean absolute error (MAE).
  - Von Mises-Fisher (vMF) for directionality / pointing reconstruction
  - And many more …

Gradient descent



- Gradient descent is a technique used to find minima in the multidimensional loss landscape.
- The optimiser is the algorithm that will do it after each N forward pass (1 batch of N samples)
- Gradient Descent in Pytorch is hidden under the hood
  - Very important you understand how it works but not so useful to program it yourself (except if you research new optimisers…)
    - Optimiser used for gradient descent defined in model PyTorch lightning module configure_optimizers() method and gradient descent performed in the Trainer.fit() method
    - You can set the optimizer as an argument and choose at run time
    - Large library of optimisers available

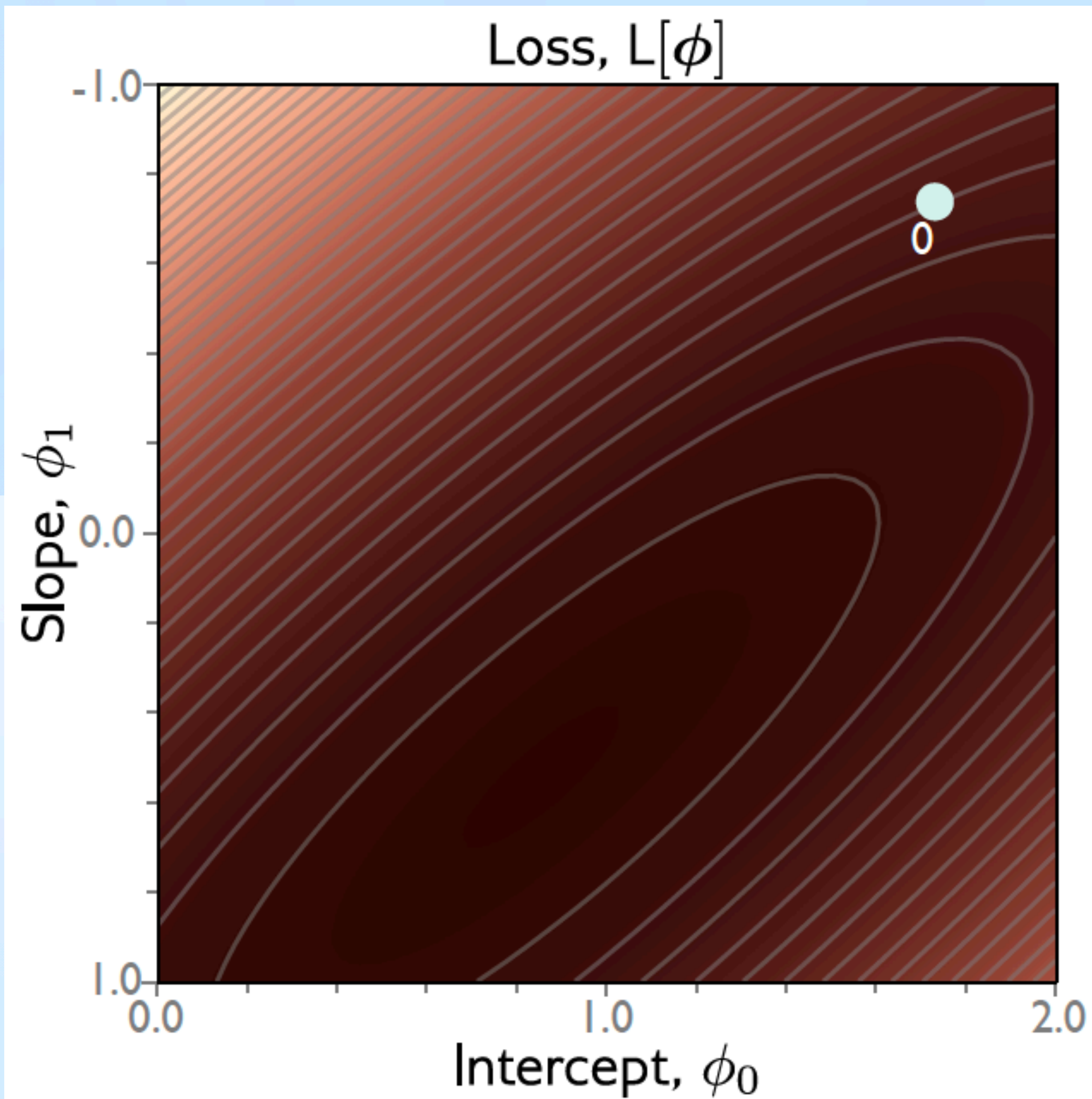
Gradient descent : case of 2 parameters-space



Step 1: Compute derivatives (slopes of function) with Respect to the parameters

$$
\begin{aligned}
L[\phi] &= \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \left(\mathrm{f}[x_i, \phi] - y_i\right)^2 \\
&= \sum_{i=1}^{I} \left(\phi_0 + \phi_1 x_i - y_i\right)^2
\end{aligned}
$$

Gradient descent

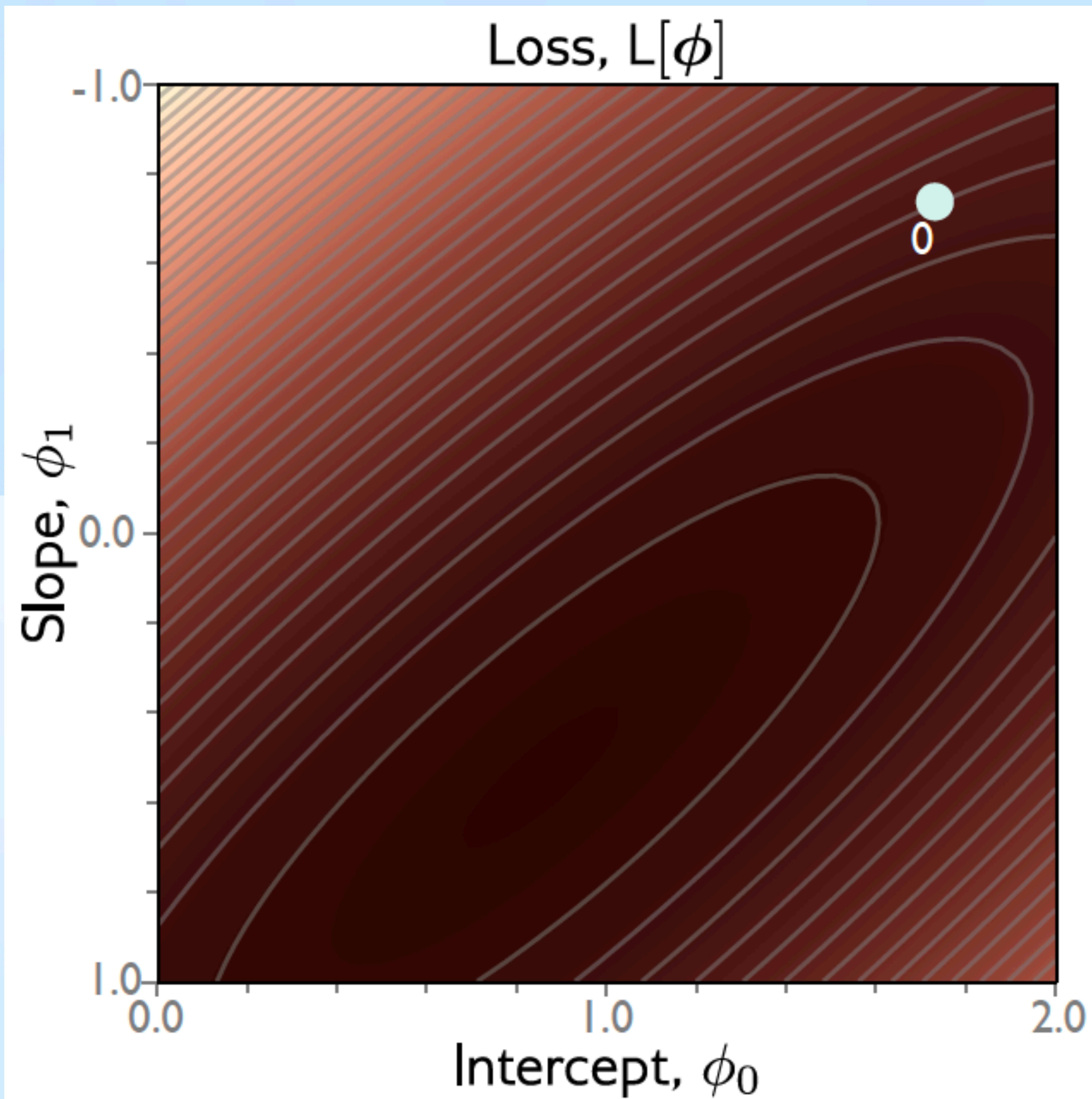


Step 1: Compute derivatives (slopes of function) with Respect to the parameters

$$
\begin{aligned}
L[\phi] &= \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} (\mathrm{f}[x_i, \phi] - y_i)^2 \\
&= \sum_{i=1}^{I} (\phi_0 + \phi_1 x_i - y_i)^2
\end{aligned}
$$

$$
\frac{\partial L}{\partial \phi} = \frac{\partial}{\partial \phi} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \phi}
$$

Gradient descent



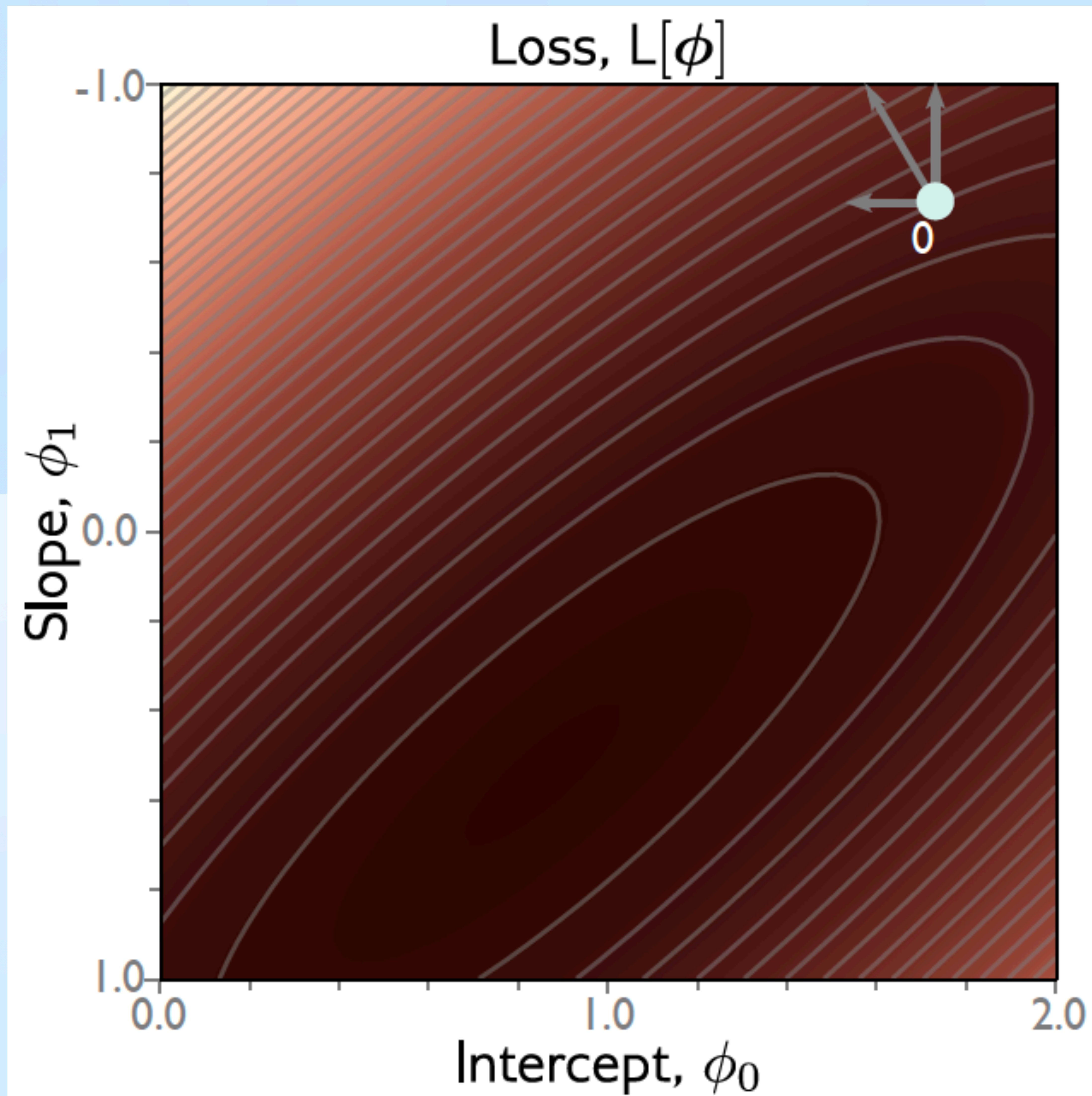
Step 1: Compute derivatives (slopes of function) with Respect to the parameters

$$L[\phi] = \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} (\mathrm{f}[x_i, \phi] - y_i)^2$$
$$= \sum_{i=1}^{I} (\phi_0 + \phi_1 x_i - y_i)^2$$

$$\frac{\partial L}{\partial \phi} = \frac{\partial}{\partial \phi} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \phi}$$

$$\frac{\partial \ell_i}{\partial \phi} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

Gradient descent

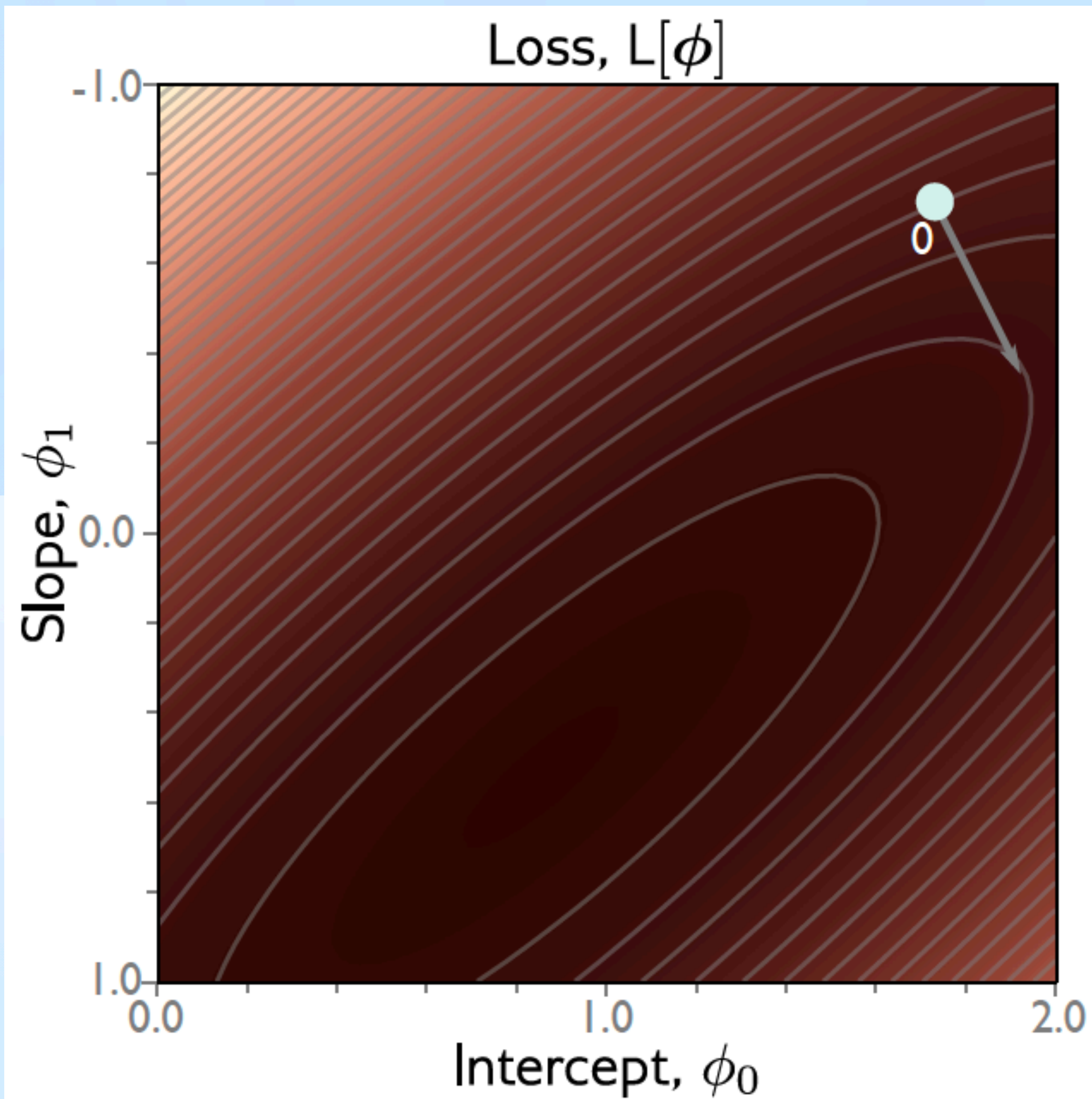


Step 1: Compute derivatives (slopes of function) with Respect to the parameters

$$\frac{\partial L}{\partial \phi} = \frac{\partial}{\partial \phi} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \phi}$$

$$\frac{\partial \ell_i}{\partial \phi} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

Gradient descent



Step 1: Compute derivatives (slopes of function) with Respect to the parameters

$$\frac{\partial L}{\partial \phi} = \frac{\partial}{\partial \phi}\sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \phi}$$

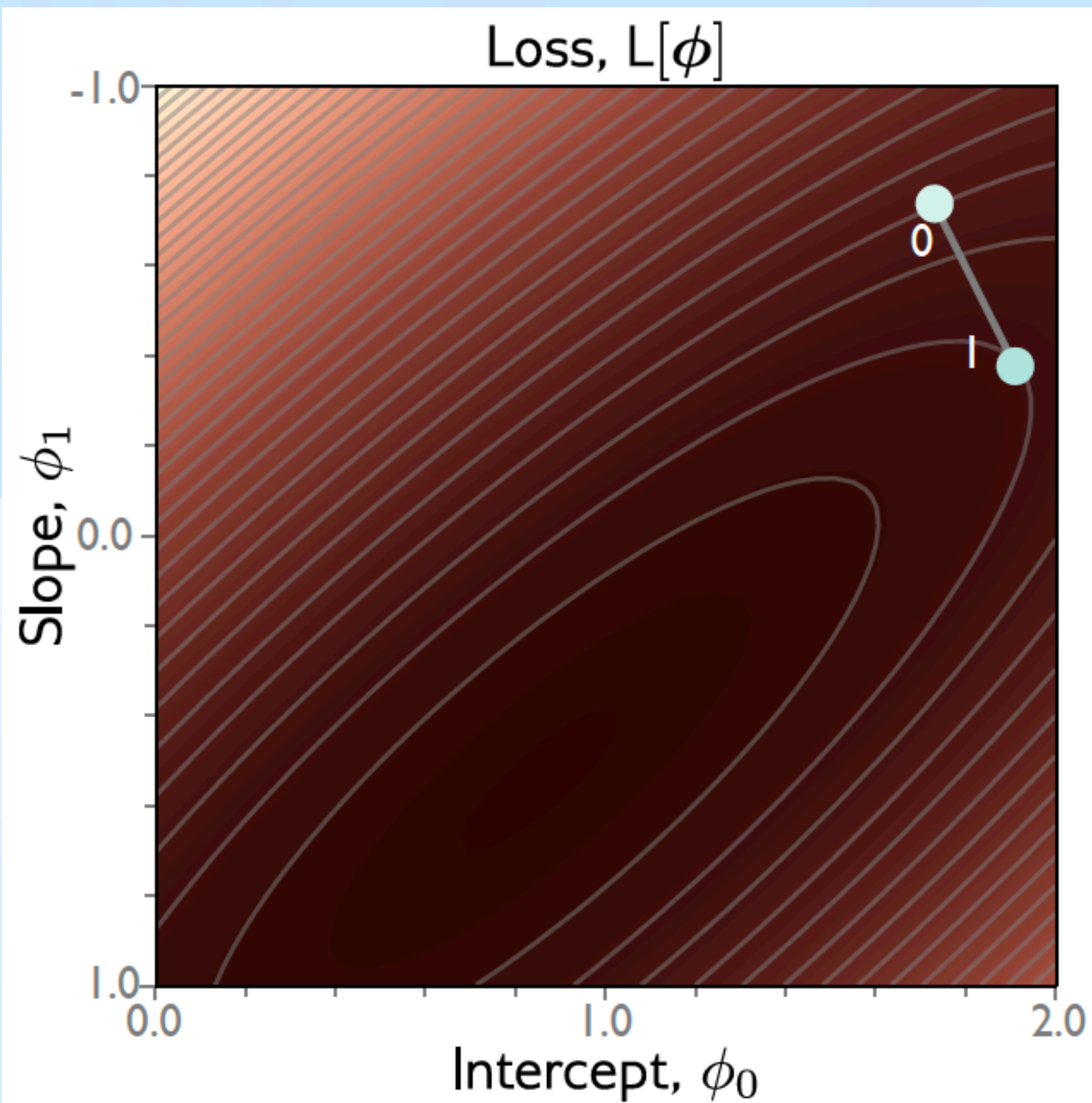
$$\frac{\partial \ell_i}{\partial \phi} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

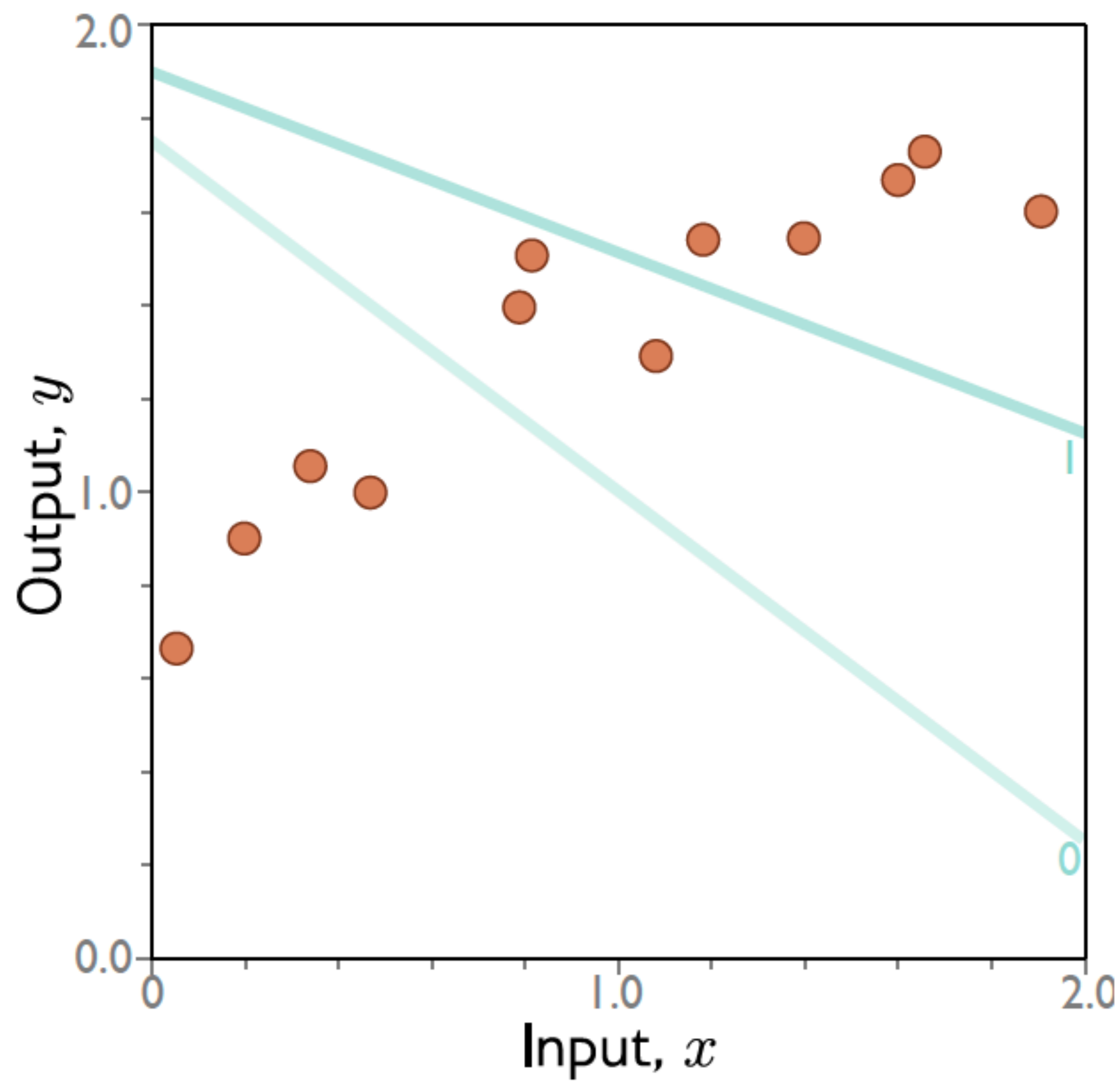
Step 2: Update parameters according to rule

$$\phi \longleftarrow \phi - \alpha \frac{\partial L}{\partial \phi}$$

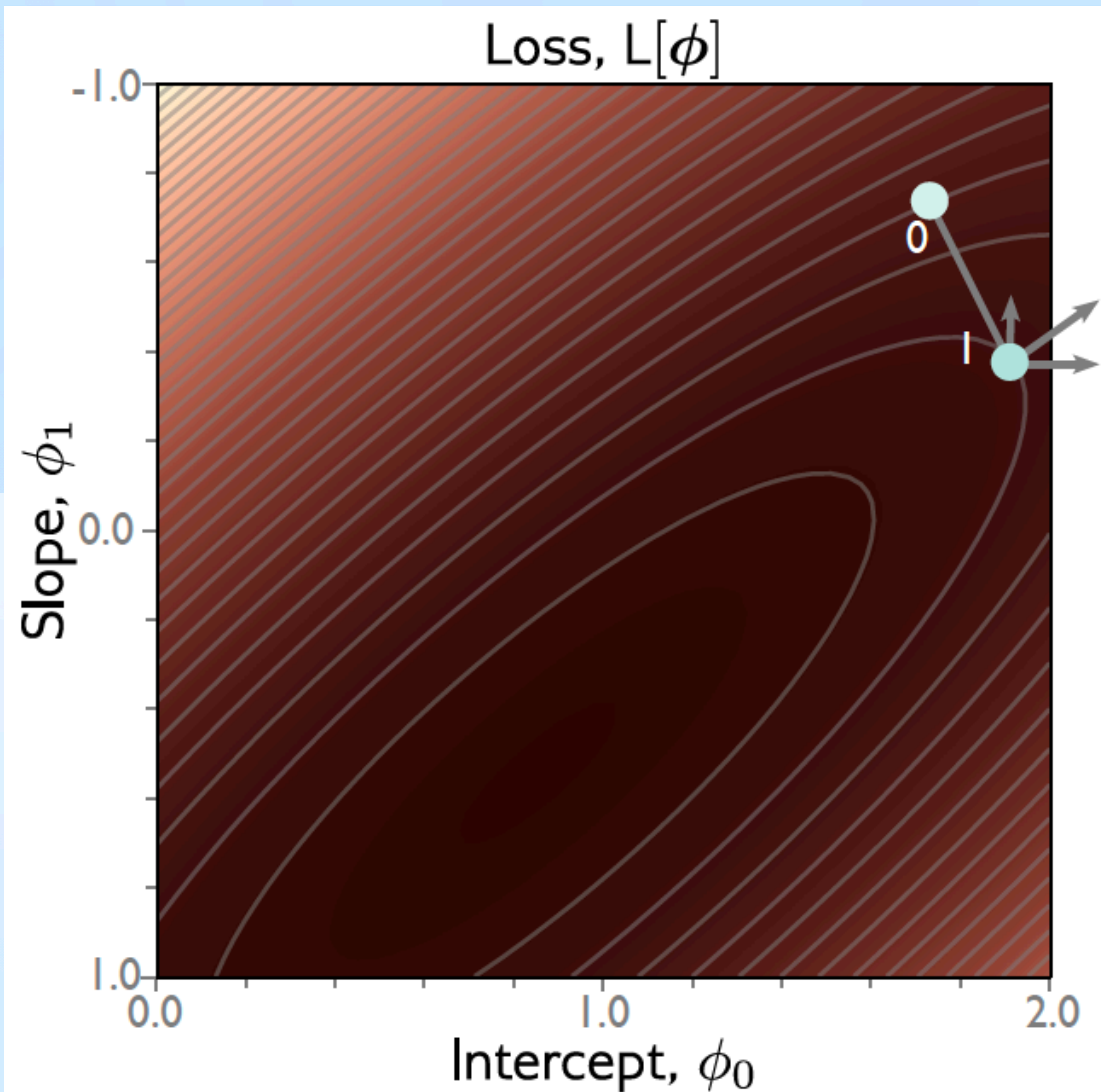
α = step size or learning rate if fixed

Gradient descent





Gradient descent



Step 1: Compute derivatives (slopes of function) with Respect to the parameters

$$\frac{\partial L}{\partial \phi} = \frac{\partial}{\partial \phi} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \phi}$$

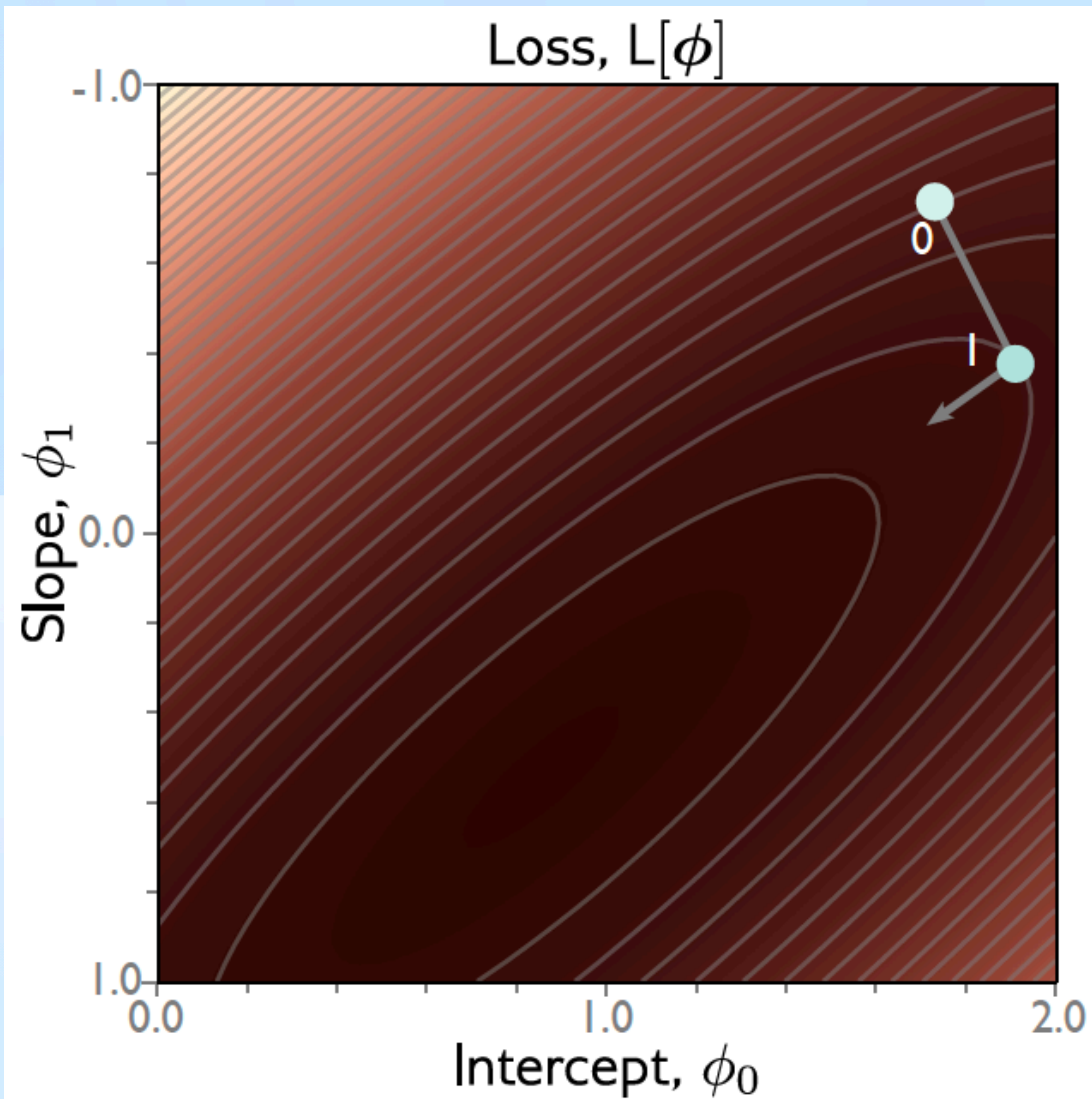
$$\frac{\partial \ell_i}{\partial \phi} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

Step 2: Update parameters according to rule

$$\phi \longleftarrow \phi - \alpha \frac{\partial L}{\partial \phi}$$

α = step size

Gradient descent



Step 1: Compute derivatives (slopes of function) with Respect to the parameters

$$\frac{\partial L}{\partial \phi} = \frac{\partial}{\partial \phi} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \phi}$$

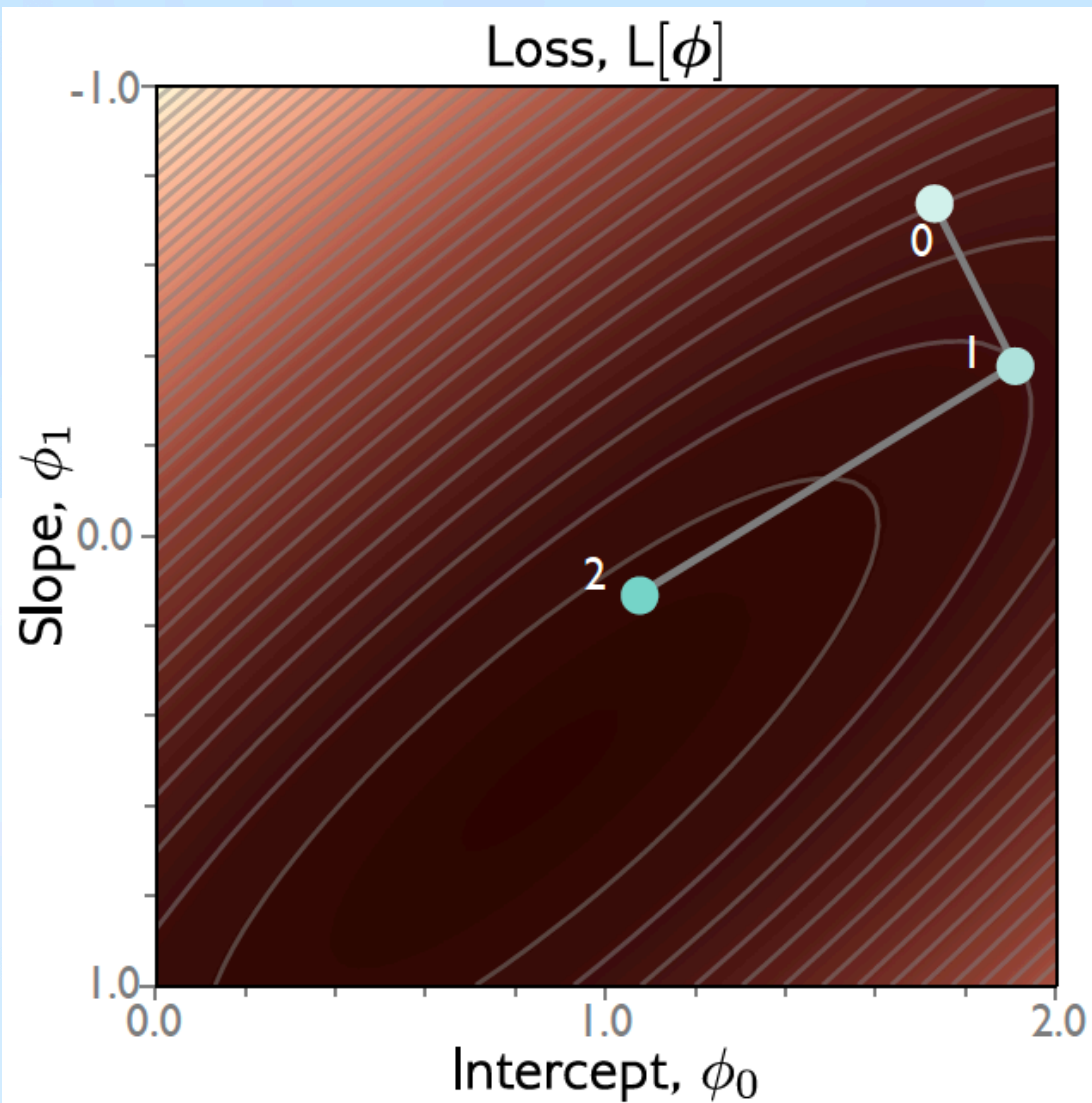
$$\frac{\partial \ell_i}{\partial \phi} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

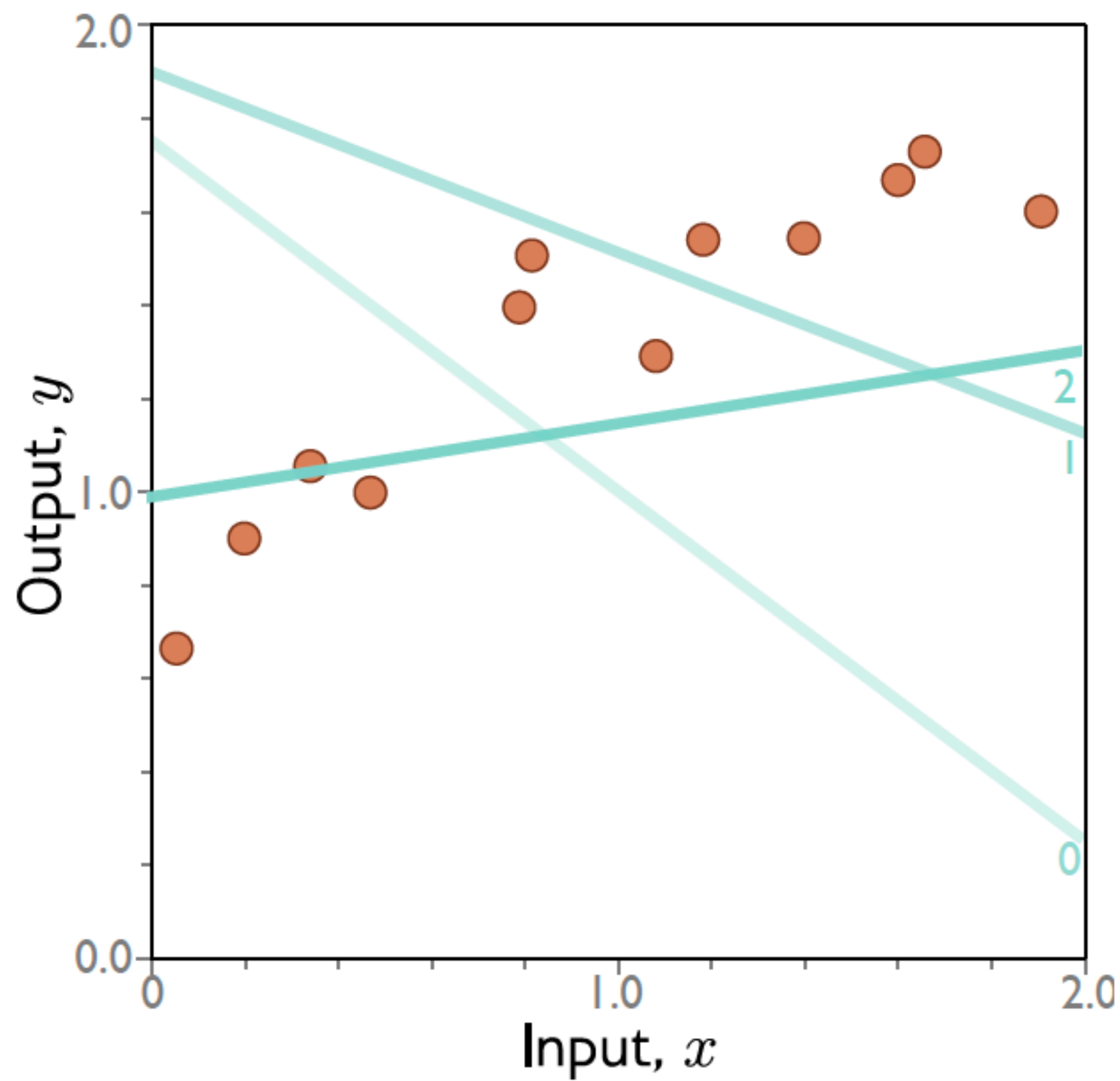
Step 2: Update parameters according to rule

$$\phi \longleftarrow \phi - \alpha \frac{\partial L}{\partial \phi}$$

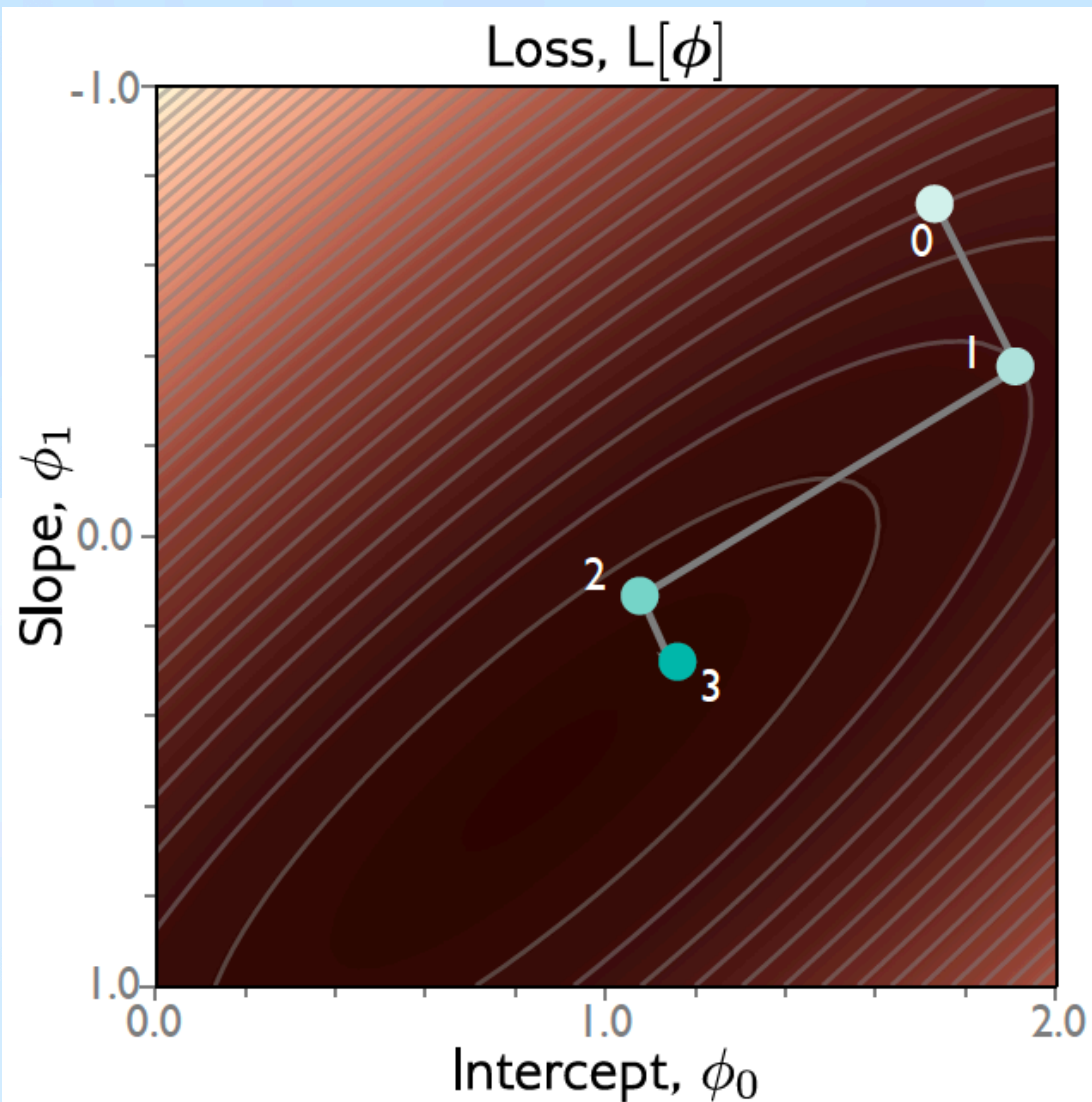
α = step size or learning rate if fixed

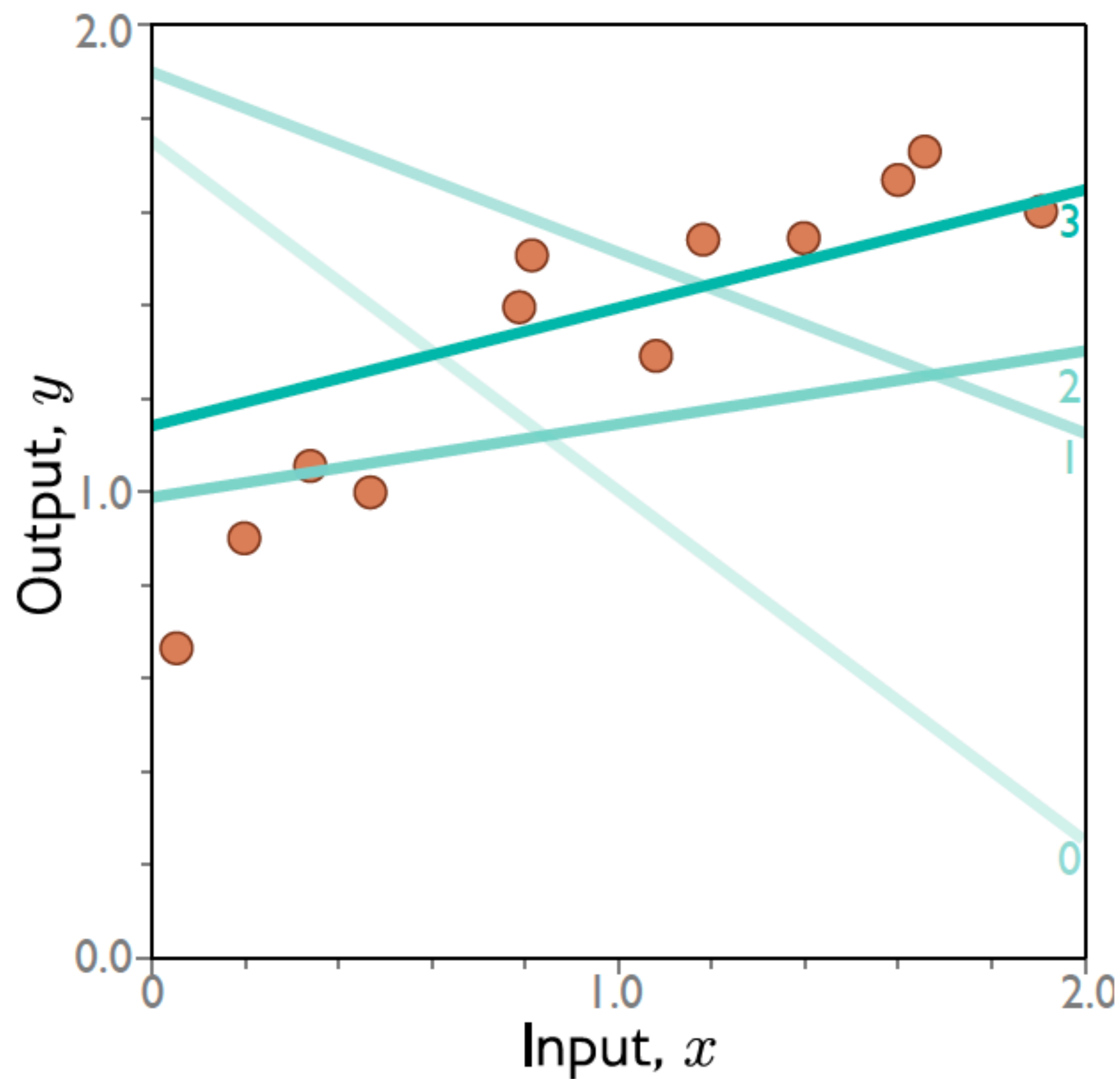
Gradient descent



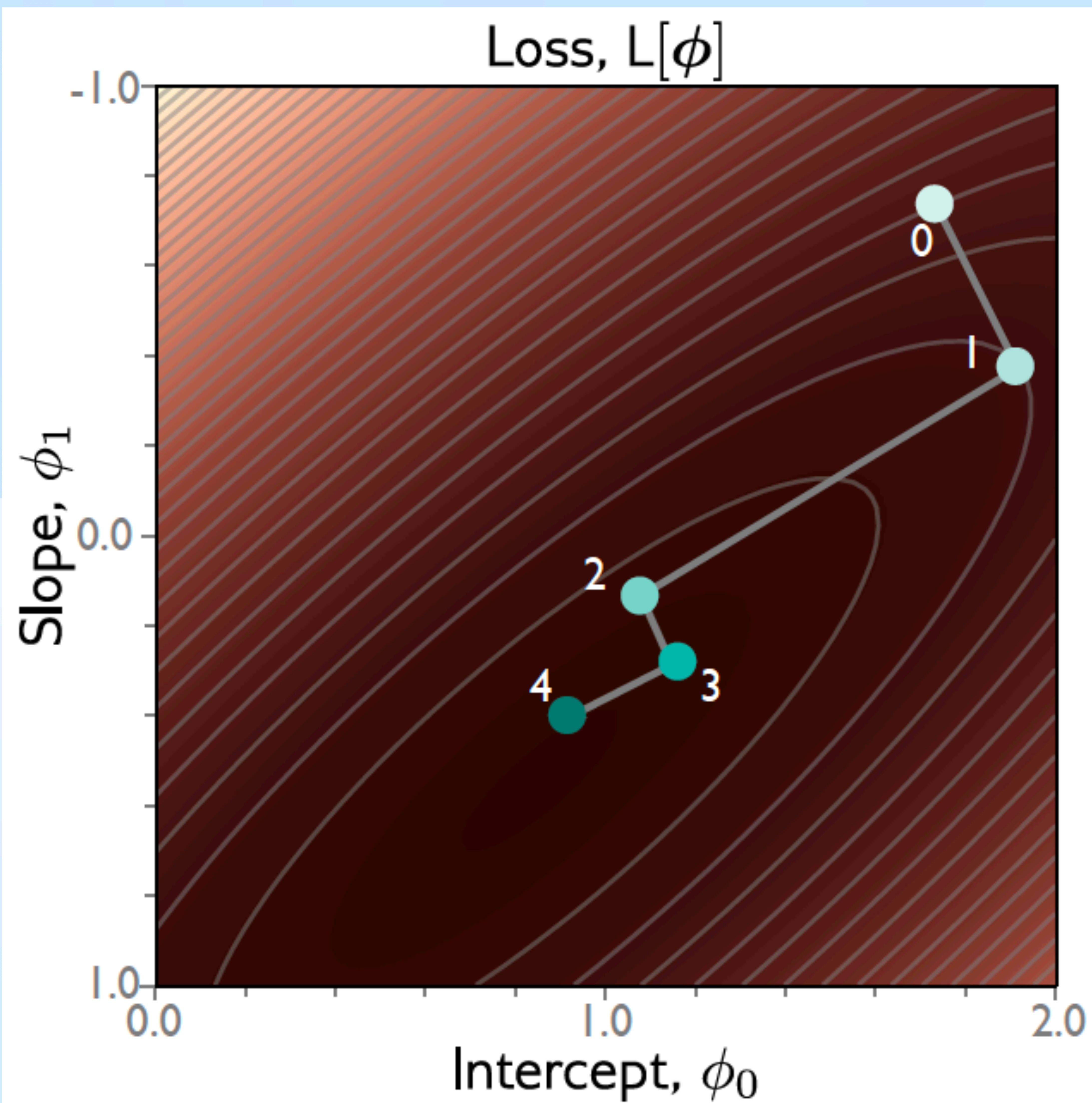


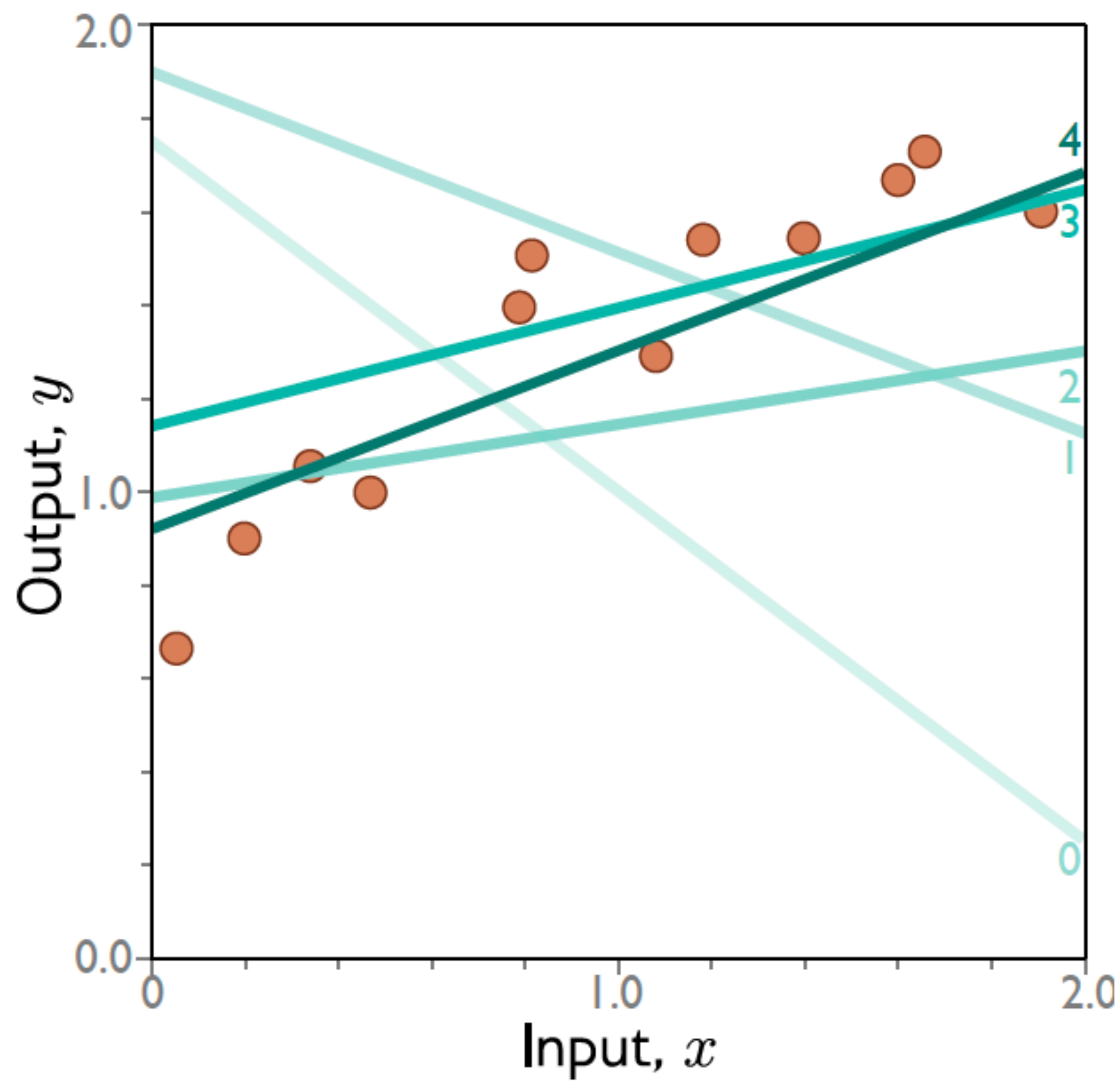
Gradient descent





Gradient descent





a) Loss, $L[\phi]$

b) Loss, $L[\phi]$

22.5

2.5

ϕ_1

-10 0 10

ϕ_0

1

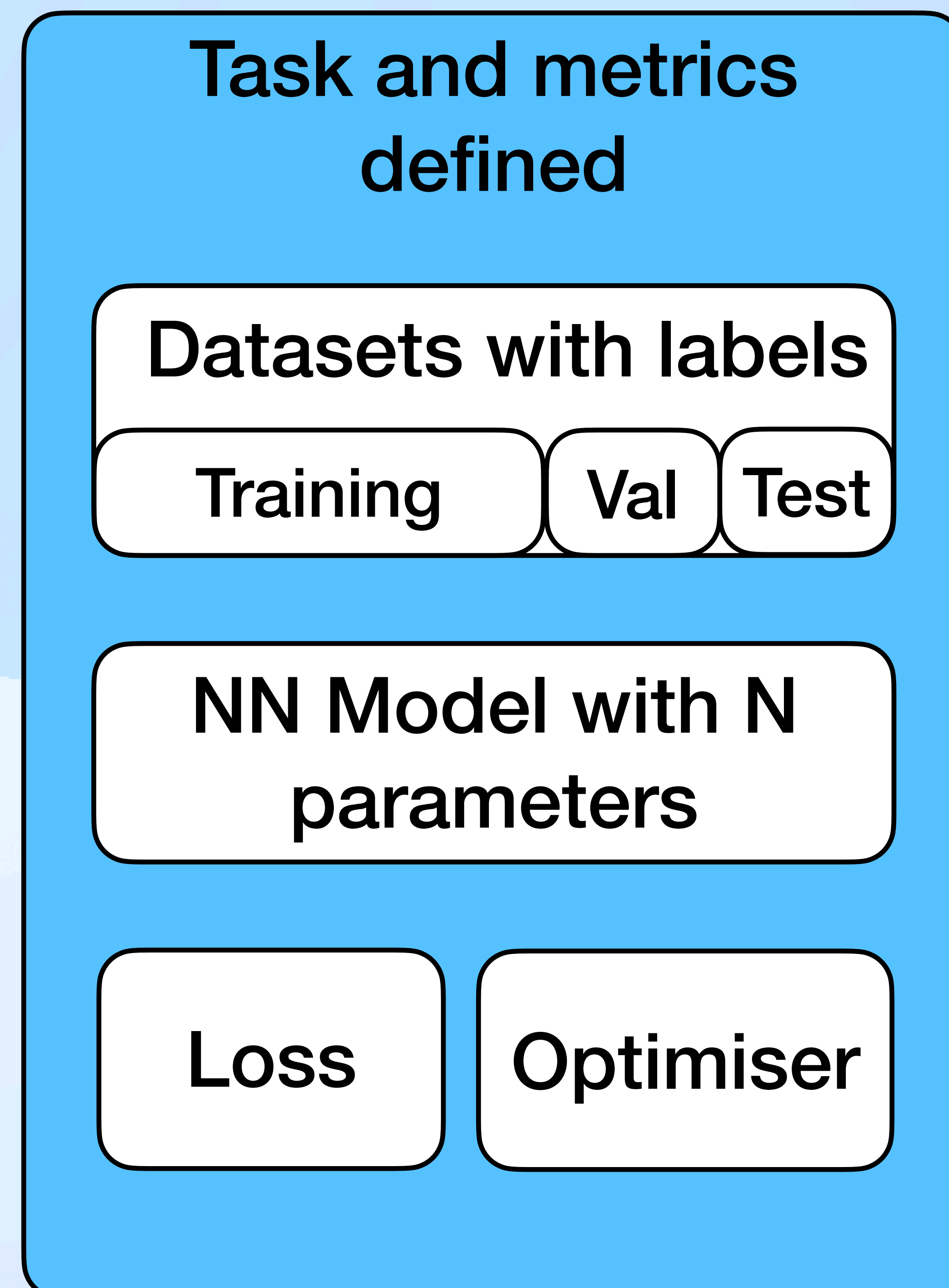
2

3

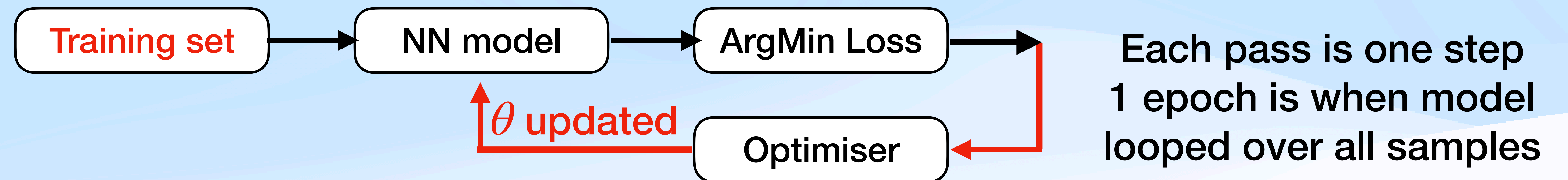
Gradient descent

Stochastic gradient descent

Recap supervised learning workflow



Training phase : forward and backpropagation to minimise the loss and find the parameters θ that fits best the label



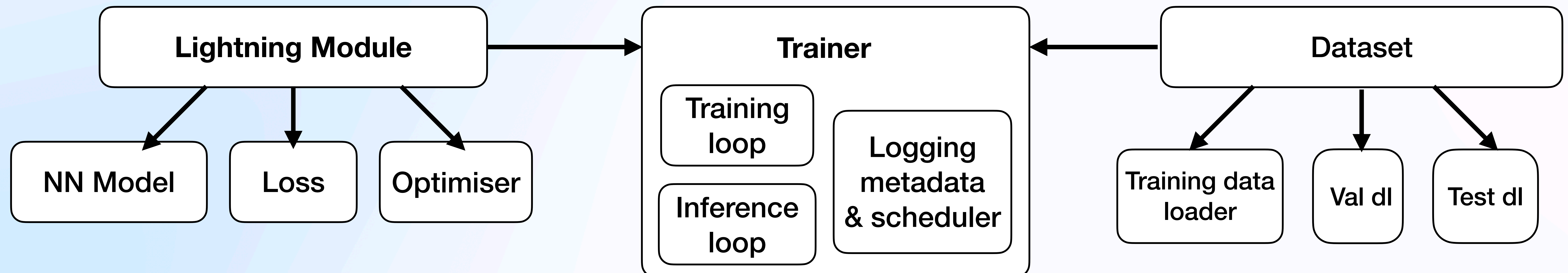
Each pass is one step
1 epoch is when model looped over all samples

Testing phase : θ parameters are fixed, evaluates the model on unseen data

Evaluate metric & deploy

An Abstracted training loop with Pytorch Lightning

- Lightning is a high level framework built on top of pytorch
  - It simplifies the code needed to run model training loop and automate it (training loop, moving the data to GPU, logging, checkpointing, parameters sweep)
- It provides key modules like lightning module, trainer, and Dataset.

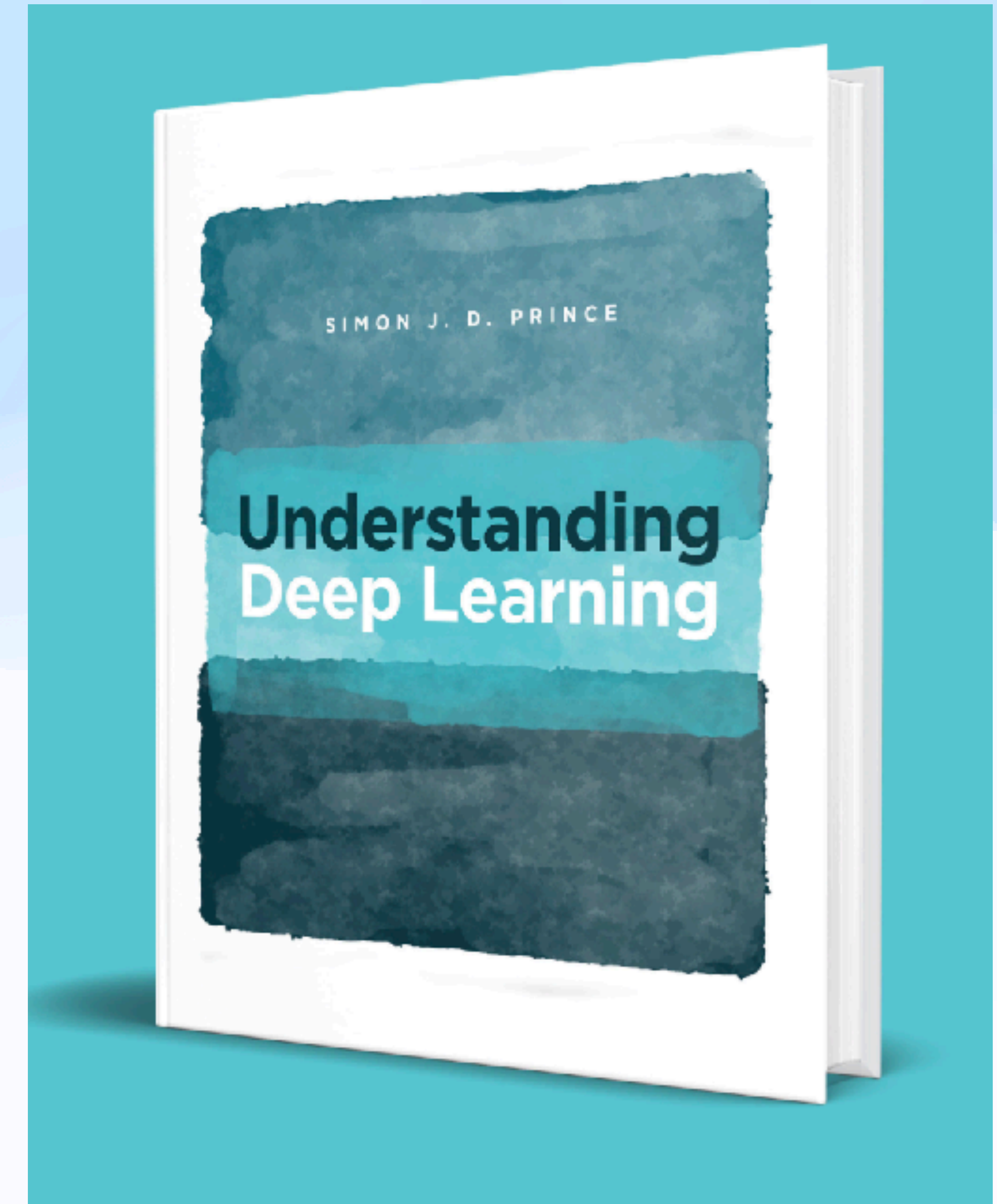


Current challenges of applying AI on physics data

- Supervised learning means data needs to be a subset of input domain used for training
- Quantify uncertainties of the data and the models
- Data and Monte Carlo differences
- More interpretable models / explainable performance could be critical
- We are still responsible to choose the best representation for the problem
  - Know your data !

Materials used for this course (and to go further)

- Many of the slides are based on this book : **Understand deep learning**
    - https://udlbook.github.io/udlbook/
- Other useful books to explore models and ML principles :
- **The little book of deep learning**, F. Fleuret (2024) - free
    - https://fleuret.org/public/lbdl.pdf
- **Deep Learning**, Y. Bengio (2015)
- **Deep Learning**, C. Bishop (2023) - free to browse
- **Deep Learning with Pytorch**, E. Stevens (2020)
- **Generative Deep Learning**, 2nd edition D. Foster (2023)
- **Machine learning Design patterns**, V. Lakshmana (2021)



Antonin Vacheret

Questions ?