



香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

COMP 4901B

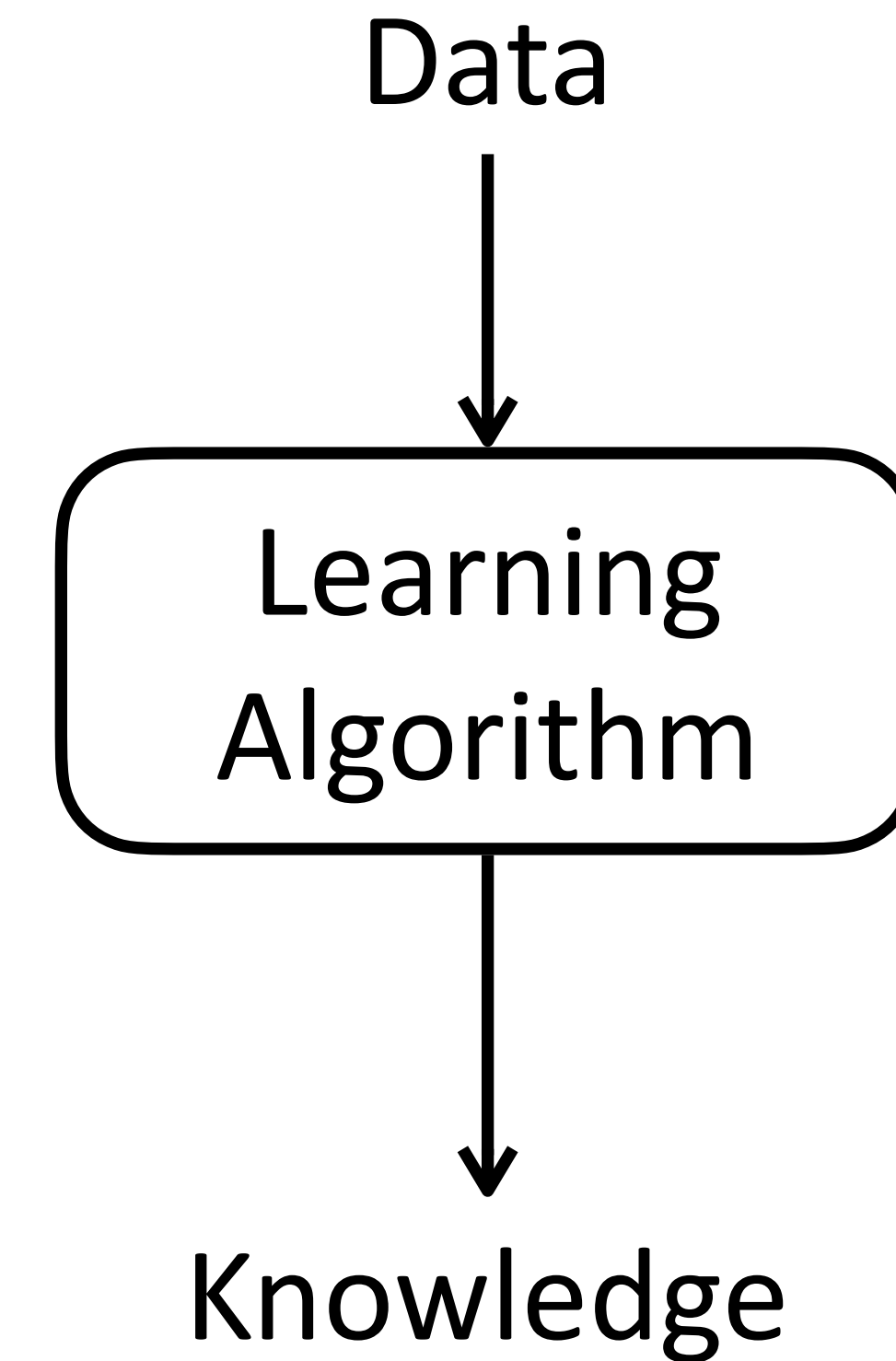
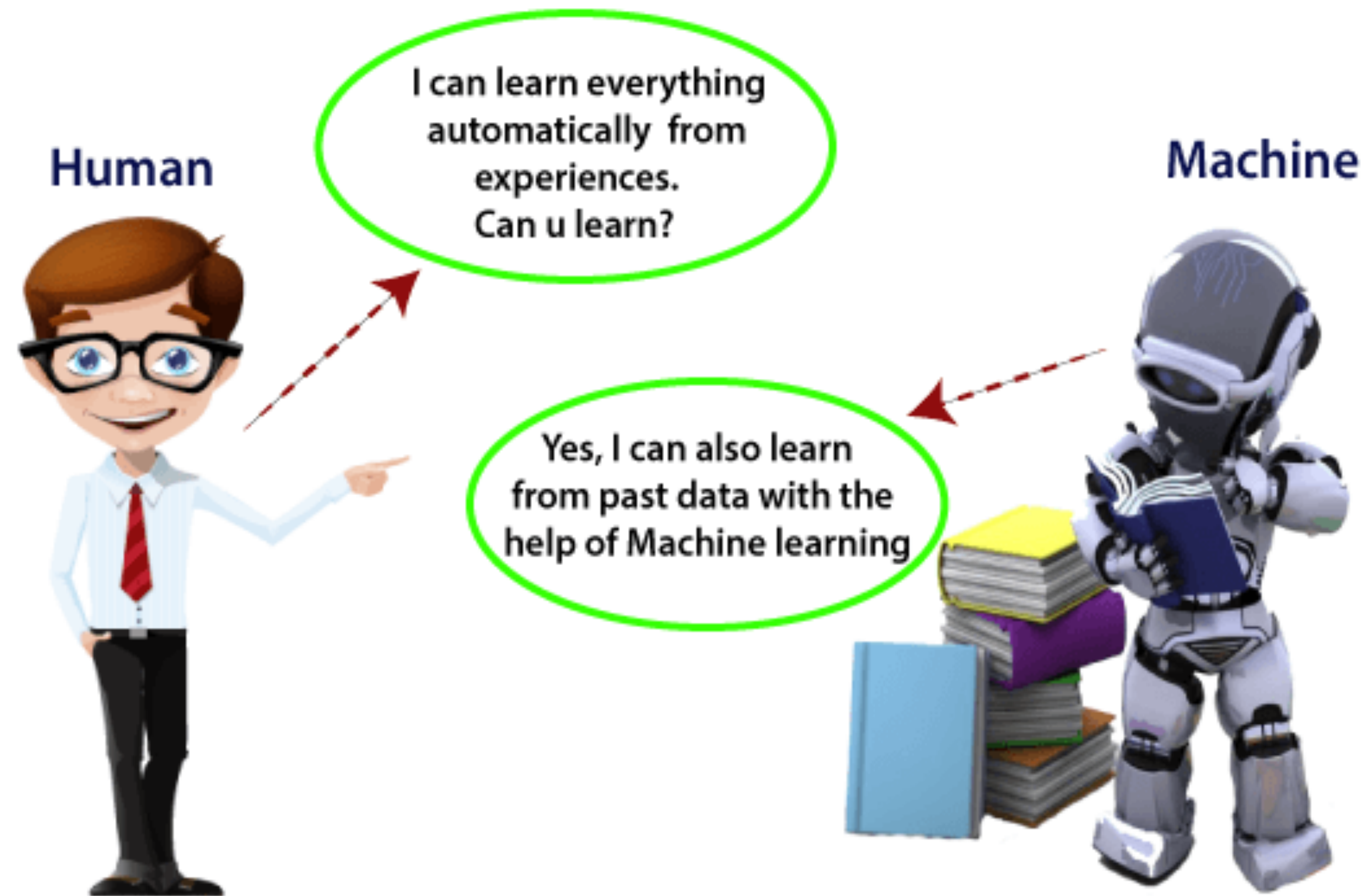
Large Language Models

Machine Learning Basics

Junxian He

Sep 5, 2025

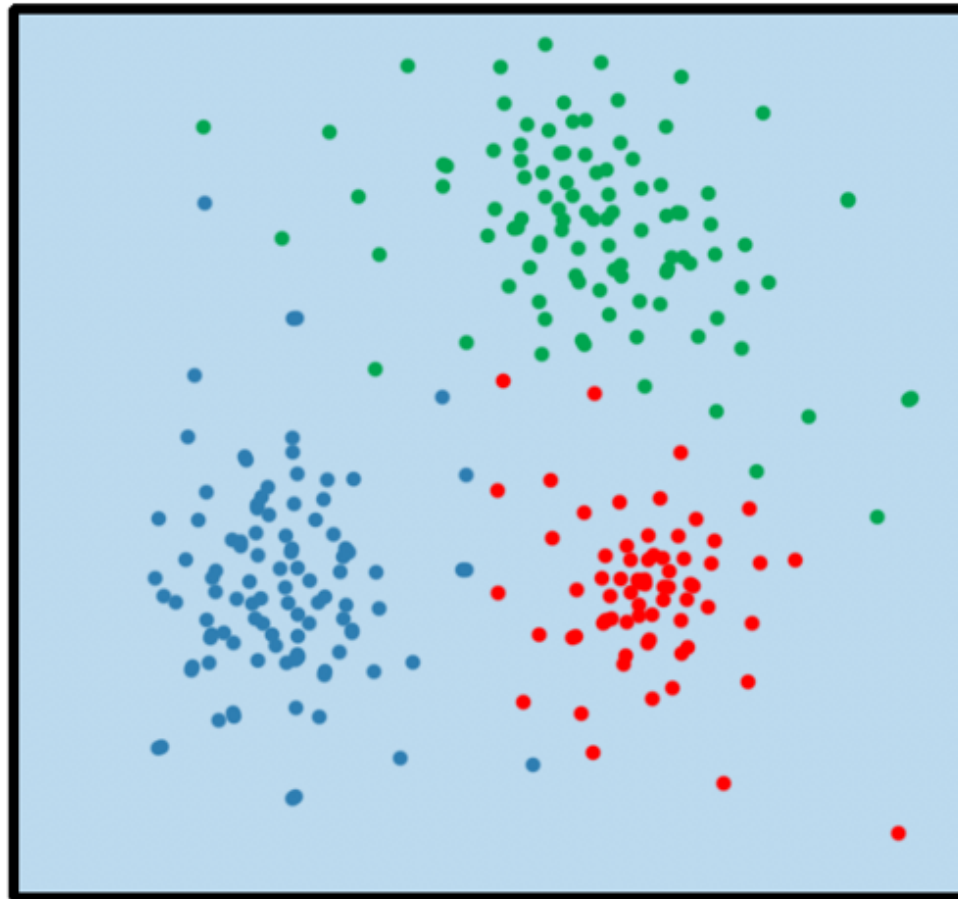
What is Machine Learning



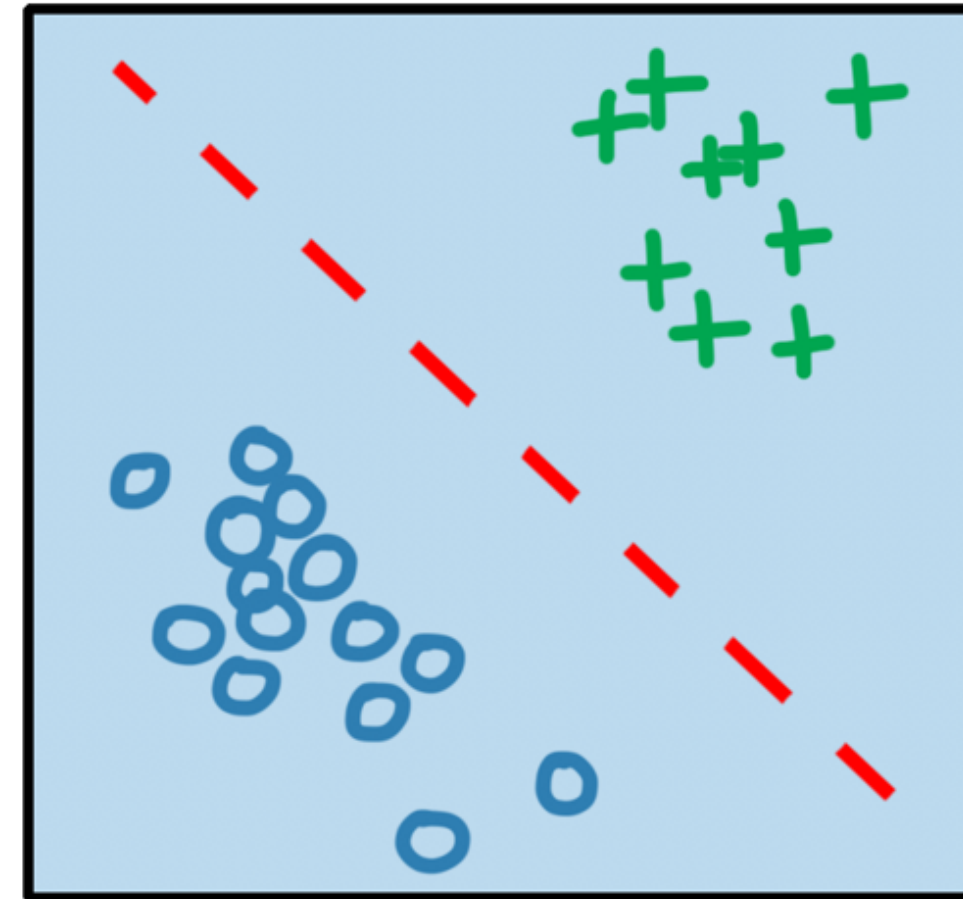
Taxonomy of Machine Learning

machine learning

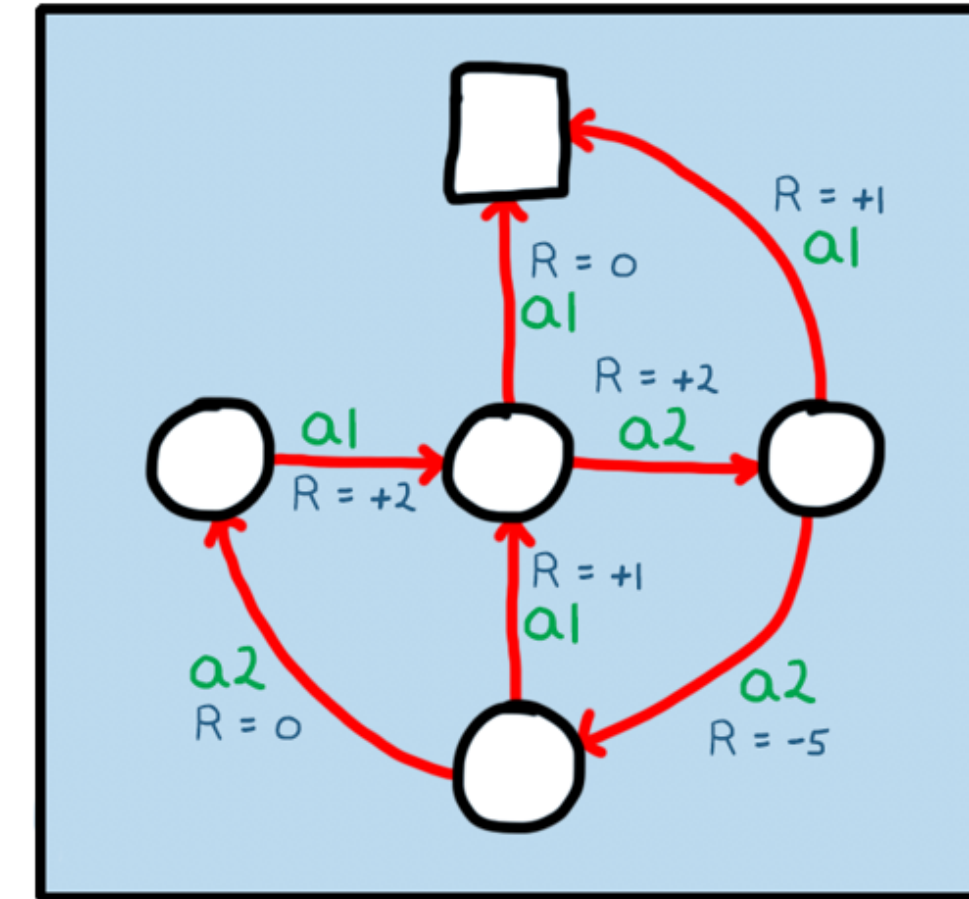
unsupervised
learning



supervised
learning



reinforcement
learning



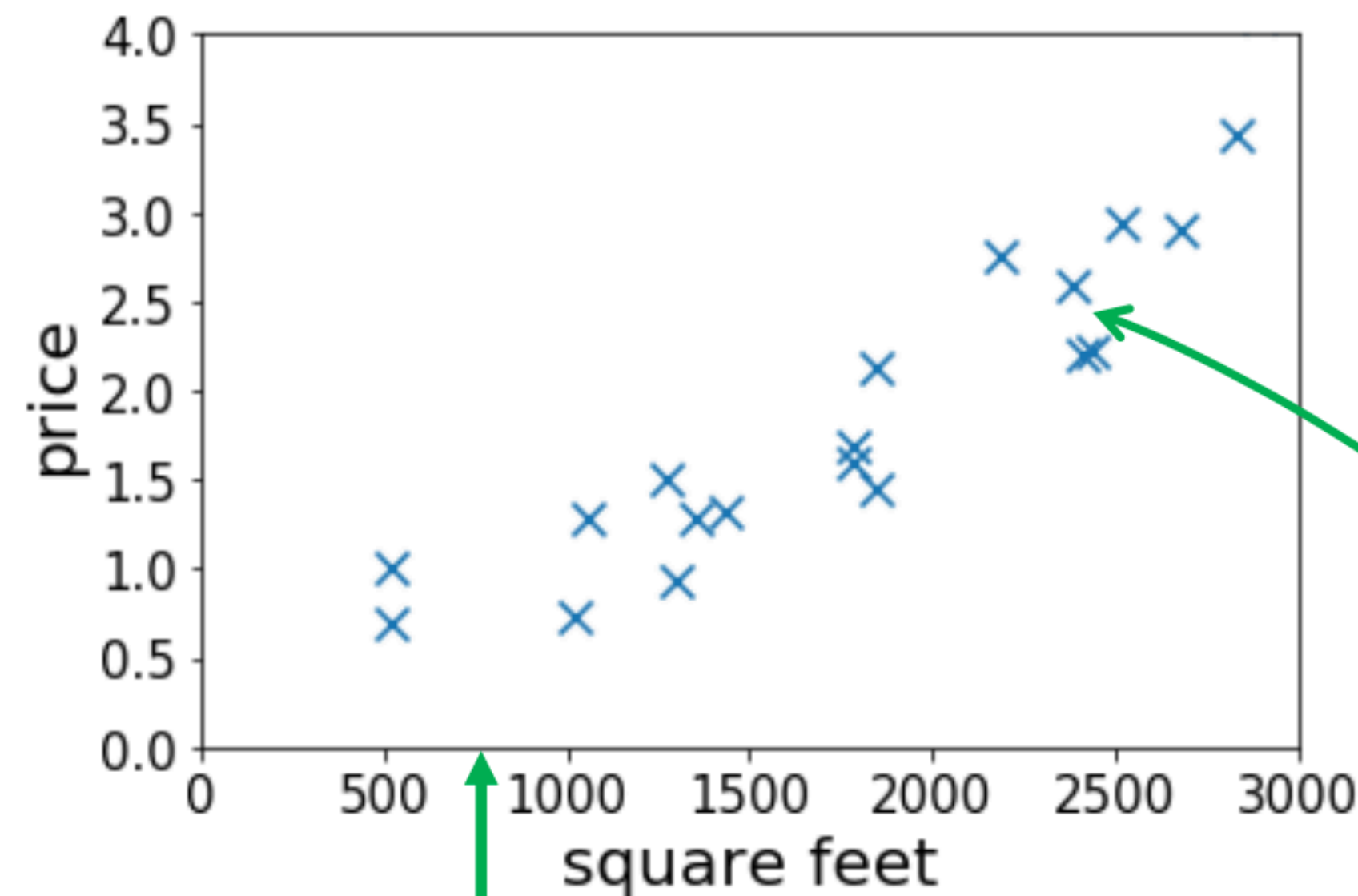
Supervised Learning

Housing Price Prediction

- Given: a dataset that contains n samples

$$(x^{(1)}, y^{(1)}), \dots (x^{(n)}, y^{(n)})$$

- Task: if a residence has x square feet, predict its price?



15th sample
 $(x^{(15)}, y^{(15)})$

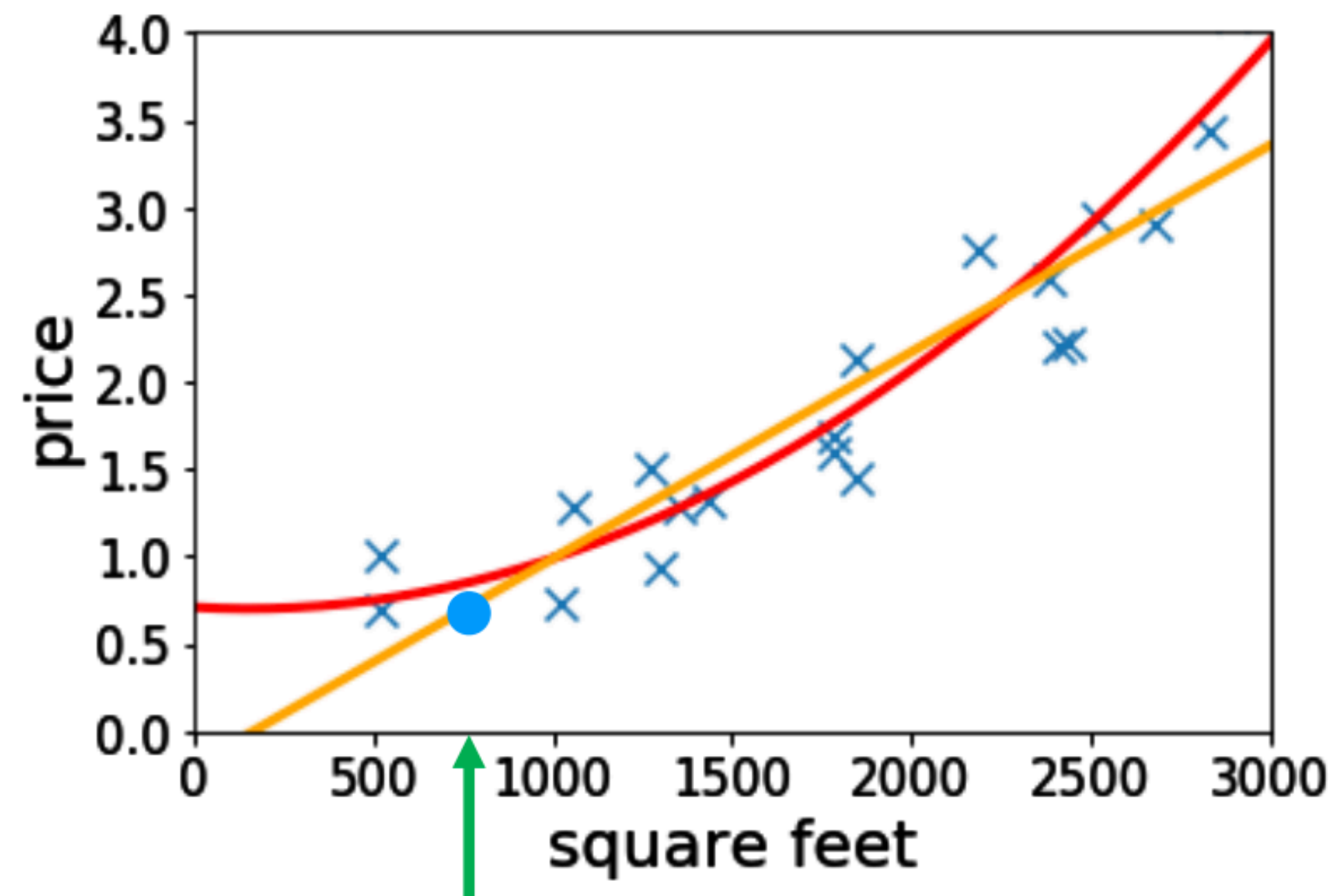
$x = 800$
 $y = ?$

Housing Price Prediction

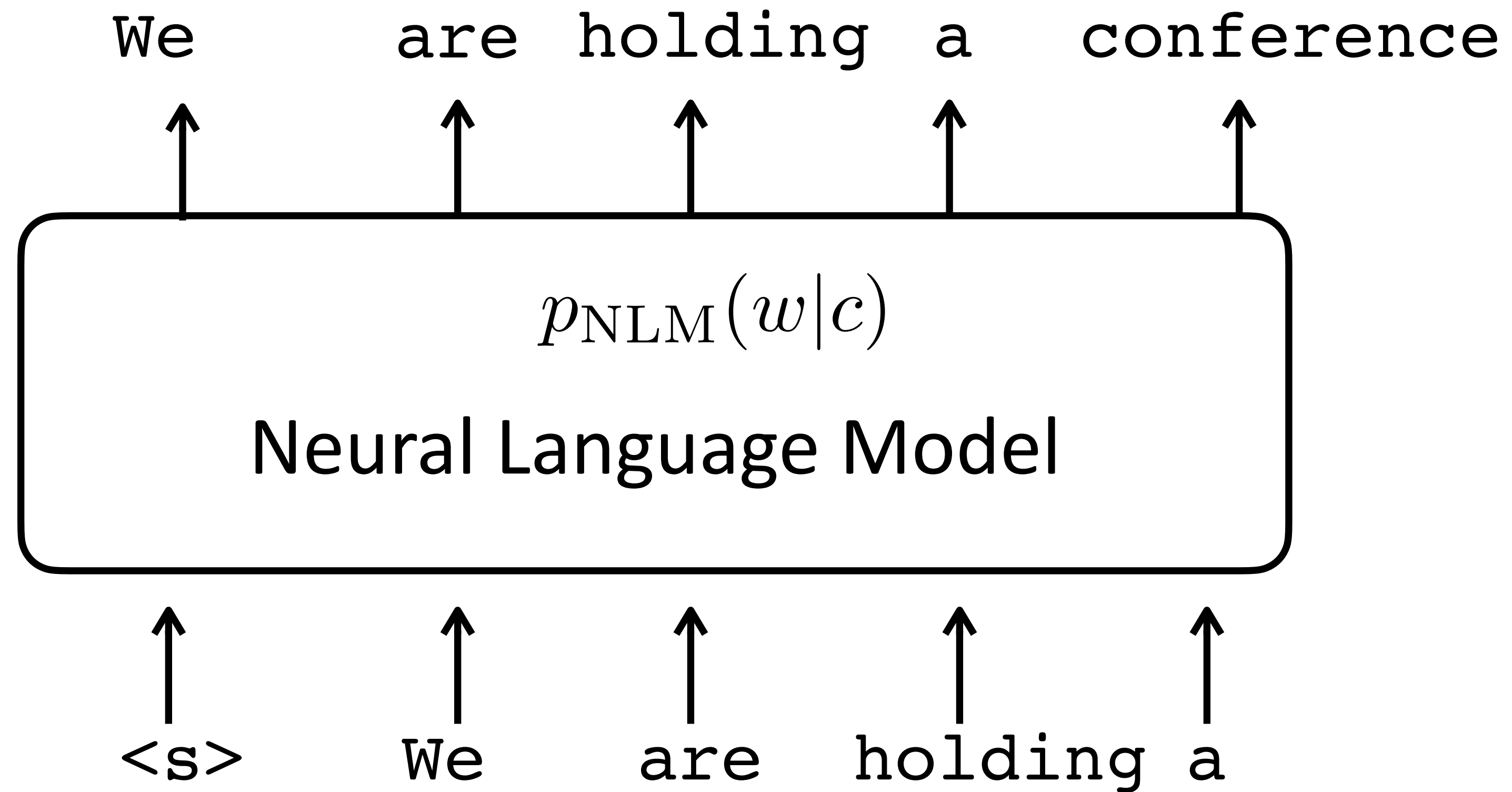
- Given: a dataset that contains n samples

$$(x^{(1)}, y^{(1)}), \dots (x^{(n)}, y^{(n)})$$

- Task: if a residence has x square feet, predict its price?



Language Models



Large Language Models

SYSTEM PROMPT
(HUMAN-WRITTEN)

In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

MODEL COMPLETION
(MACHINE-WRITTEN,
10 TRIES)

The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

Large Language Models

Context →	Please unscramble the letters into a word, and write that word: taefed =
Target Completion →	defeat
Context →	L'analyse de la distribution de fréquence des stades larvaires d'I. verticalis dans une série d'étangs a également démontré que les larves mâles étaient à des stades plus avancés que les larves femelles. =
Target Completion →	Analysis of instar distributions of larval I. verticalis collected from a series of ponds also indicated that males were in more advanced instars than females.
Context →	Q: What is 95 times 45? A:
Target Completion →	4275

Classification



CAT

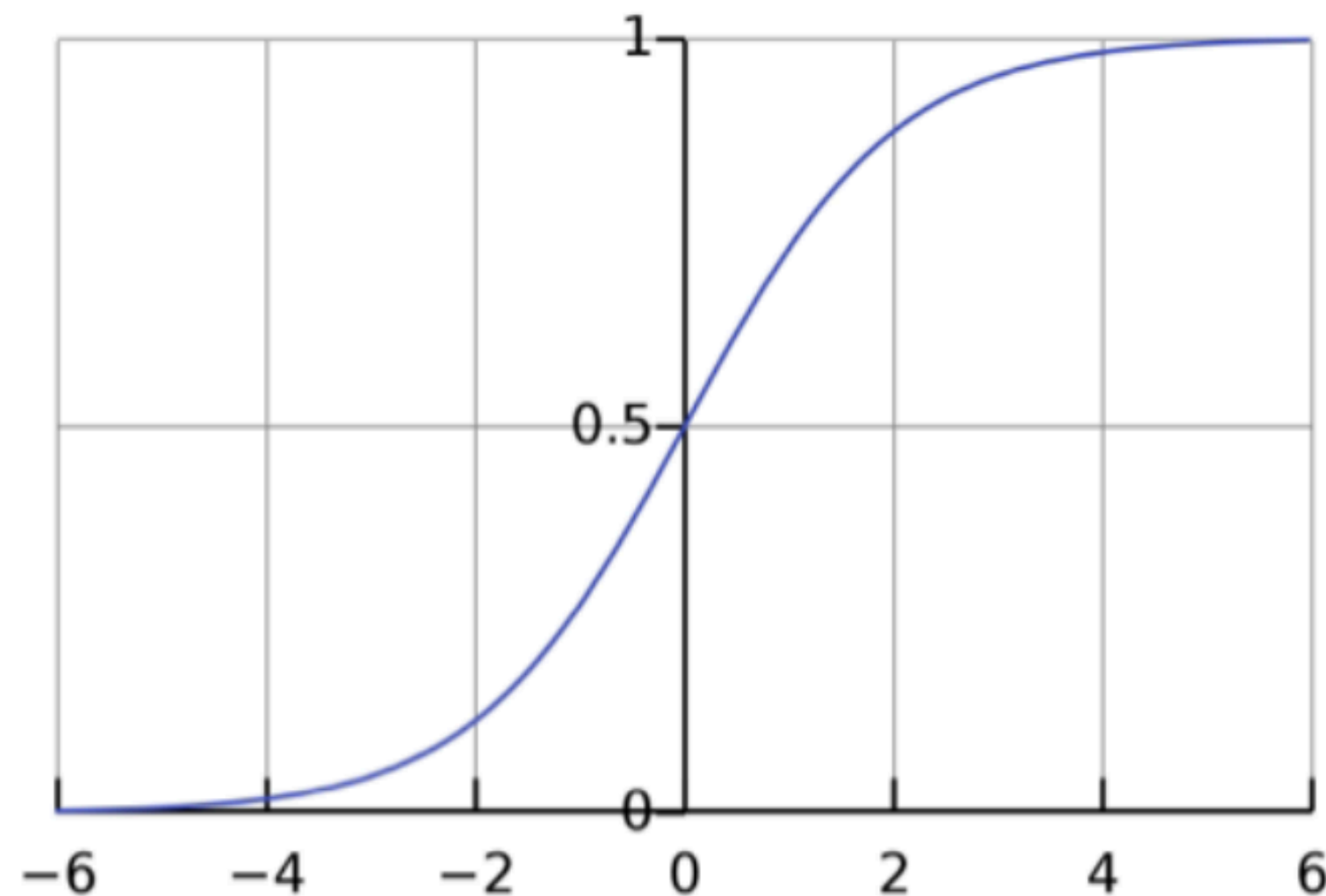
Labels are discrete

Logistic Regression

Given a training set $\{(x^{(i)}, y^{(i)}) \text{ for } i = 1, \dots, n\}$ let $y^{(i)} \in \{0, 1\}$.
Want $h_{\theta}(x) \in [0, 1]$. Let's pick a smooth function:

$$h_{\theta}(x) = g(\theta^T x) \quad \text{Link Function}$$

There are many options of g



$$g(z) = \frac{1}{1 + e^{-z}} \cdot \quad \begin{array}{l} \text{Logistic Function} \\ \text{Sigmoid Function} \end{array}$$

How do we interpret $h_{\theta}(x)$?

$$P(y = 1 \mid x; \theta) = h_{\theta}(x)$$

$$P(y = 0 \mid x; \theta) = 1 - h_{\theta}(x)$$

Logistic Regression

Let's write the Likelihood function. Recall:

$$P(y = 1 \mid x; \theta) = h_{\theta}(x)$$

$$P(y = 0 \mid x; \theta) = 1 - h_{\theta}(x)$$

Then,

$$\begin{aligned} L(\theta) = P(y \mid X; \theta) &= \prod_{i=1}^n p(y^{(i)} \mid x^{(i)}; \theta) \quad \text{We want to express "if-then" logics, how?} \\ &= \prod_{i=1}^n h_{\theta}(x^{(i)})^{y^{(i)}} (1 - h_{\theta}(x^{(i)}))^{1-y^{(i)}} \end{aligned}$$

Taking logs to compute the log likelihood $\ell(\theta)$ we have:

$$\ell(\theta) = \log L(\theta) = \sum_{i=1}^n y^{(i)} \log h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \quad \text{Maximum likelihood estimation}$$

Multi-Label Classification

Given a training set $\{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$, $y^{(i)} \in \{1, 2, \dots, k\}$,
we aim to model the distribution $p(y | x; \theta)$

Categorical distribution, $p(y = k | x; \theta) = \phi_k$

$$\text{s.t. } \sum_{i=1}^k \phi_i = 1$$

$$\phi_i = \theta_i^T x ?$$

Softmax Function

$$\text{Softmax}: \mathbb{R}^k \rightarrow \mathbb{R}^k$$

$$\text{softmax}(t_1, \dots, t_k) = \begin{bmatrix} \frac{\exp(t_1)}{\sum_{j=1}^k \exp(t_j)} \\ \vdots \\ \frac{\exp(t_k)}{\sum_{j=1}^k \exp(t_j)} \end{bmatrix}$$

The denominator is a normalization constant

Multi-Label Classification

$$\text{Let } (t_1, \dots, t_k) = (\theta_1^\top x, \dots, \theta_k^\top x)$$

$$\begin{bmatrix} P(y = 1 \mid x; \theta) \\ \vdots \\ P(y = k \mid x; \theta) \end{bmatrix} = \text{softmax}(t_1, \dots, t_k) = \begin{bmatrix} \frac{\exp(\theta_1^\top x)}{\sum_{j=1}^k \exp(\theta_j^\top x)} \\ \vdots \\ \frac{\exp(\theta_k^\top x)}{\sum_{j=1}^k \exp(\theta_j^\top x)} \end{bmatrix}$$

$$P(y = i \mid x; \theta) = \phi_i = \frac{\exp(t_i)}{\sum_{j=1}^k \exp(t_j)} = \frac{\exp(\theta_i^\top x)}{\sum_{j=1}^k \exp(\theta_j^\top x)}$$

Multi-Label Classification

$$-\log p(y \mid x, \theta) = -\log \left(\frac{\exp(t_y)}{\sum_{j=1}^k \exp(t_j)} \right) = -\log \left(\frac{\exp(\theta_y^\top x)}{\sum_{j=1}^k \exp(\theta_j^\top x)} \right)$$

$$\ell(\theta) = \sum_{i=1}^n -\log \left(\frac{\exp(\theta_{y^{(i)}}^\top x^{(i)})}{\sum_{j=1}^k \exp(\theta_j^\top x^{(i)})} \right) \quad \text{Negative log likelihood}$$

Cross-entropy loss $\ell_{\text{ce}} : \mathbb{R}^k \times \{1, \dots, k\} \rightarrow \mathbb{R}_{\geq 0}$

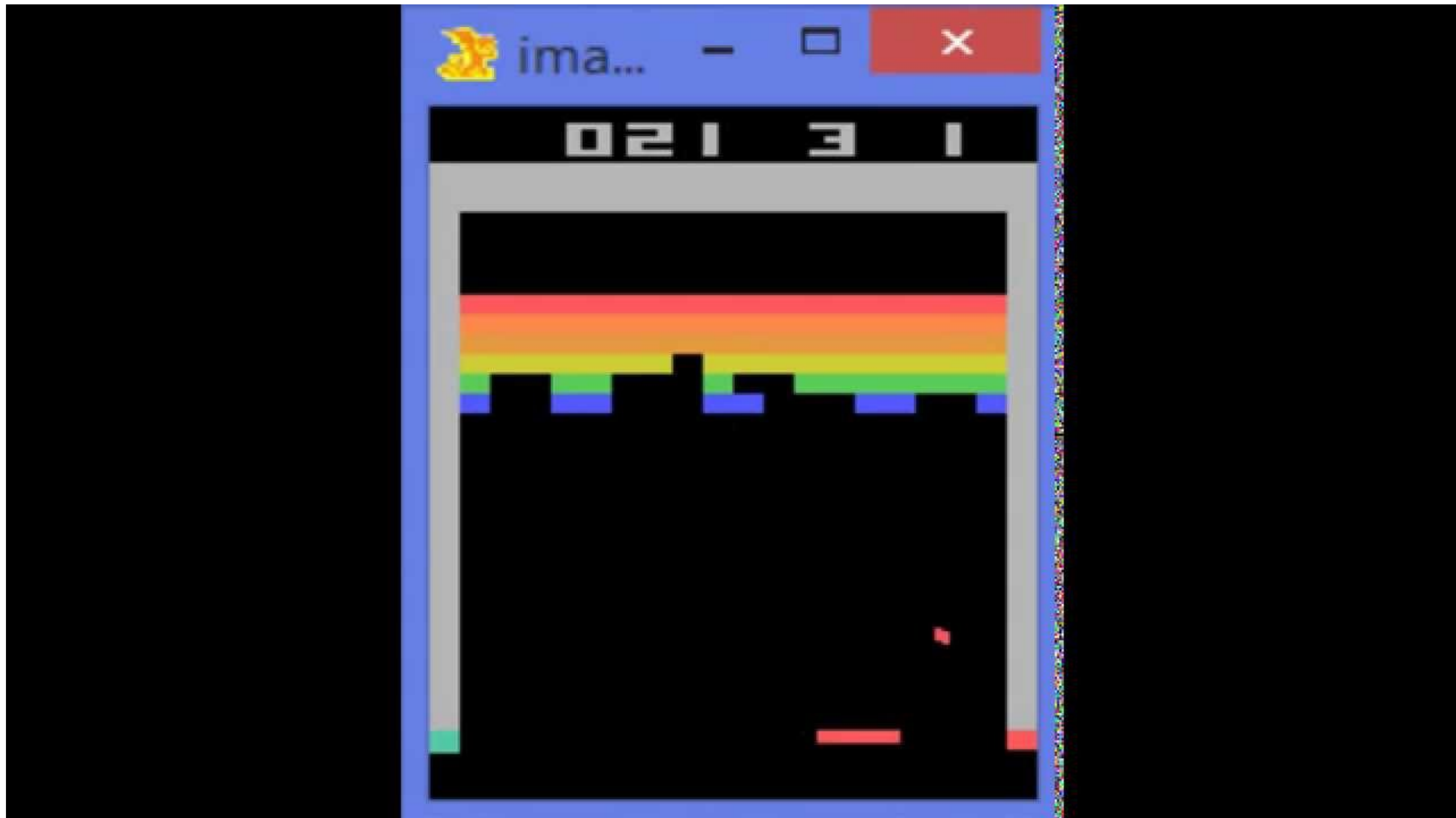
$$\ell_{\text{ce}}((t_1, \dots, t_k), y) = -\log \left(\frac{\exp(t_y)}{\sum_{j=1}^k \exp(t_j)} \right) \quad \ell(\theta) = \sum_{i=1}^n \ell_{\text{ce}}((\theta_1^\top x^{(i)}, \dots, \theta_k^\top x^{(i)}), y^{(i)})$$

Reinforcement Learning

AlphaGo

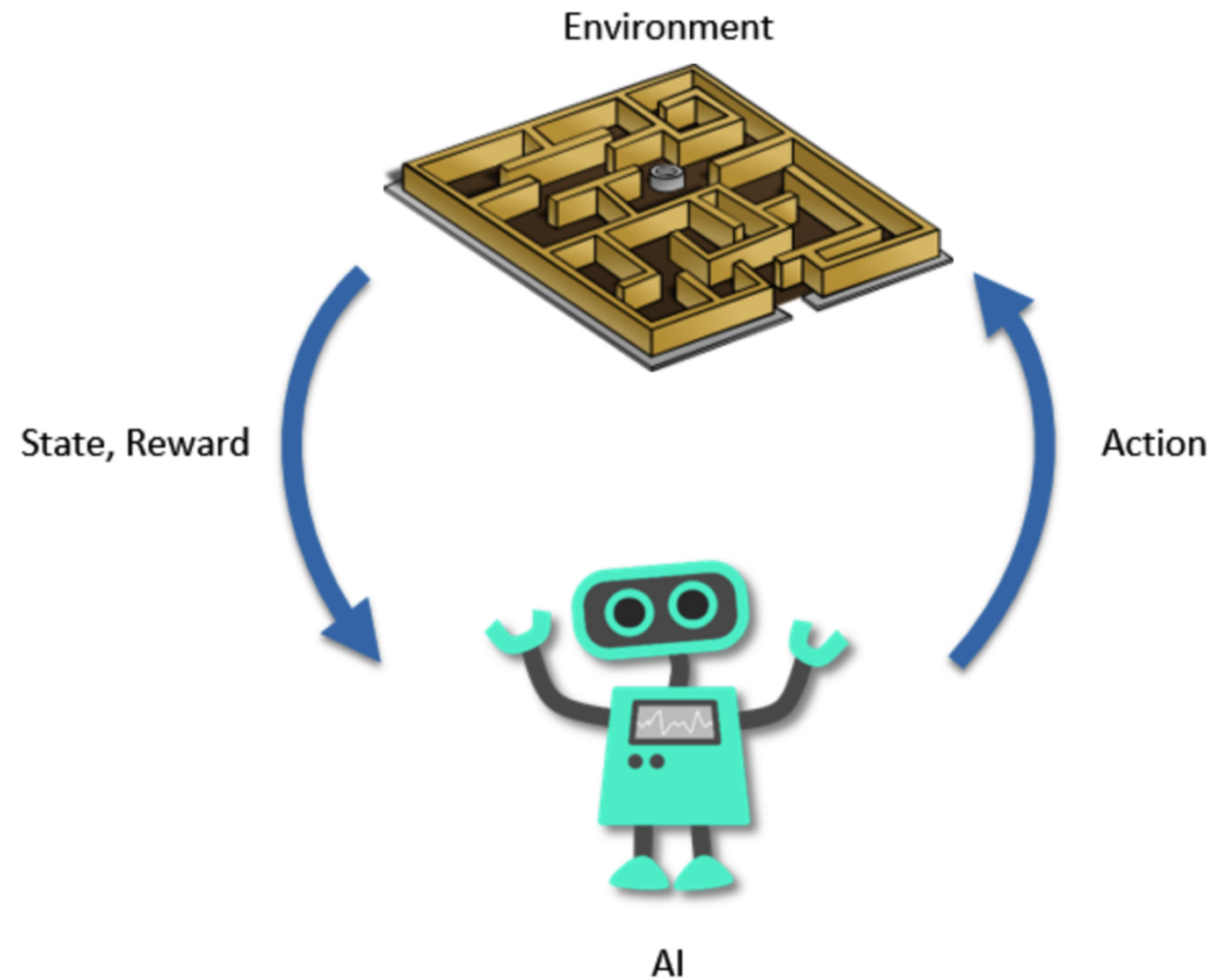


Atari Breakout Game



Reinforcement Learning

- RL can collect data interactively



Train, Validation, Test

Training data is the data we see and use during model development

Validation dataset is another set of pairs $\{(\hat{x}^{(1)}, \hat{y}^{(1)}), \dots, (\hat{x}^{(m)}, \hat{y}^{(m)})\}$

Does not overlap with training dataset

Test dataset is another set of pairs $\{(\tilde{x}^{(1)}, \tilde{y}^{(1)}), \dots, (\tilde{x}^{(L)}, \tilde{y}^{(L)})\}$

Does not overlap with training and validation dataset

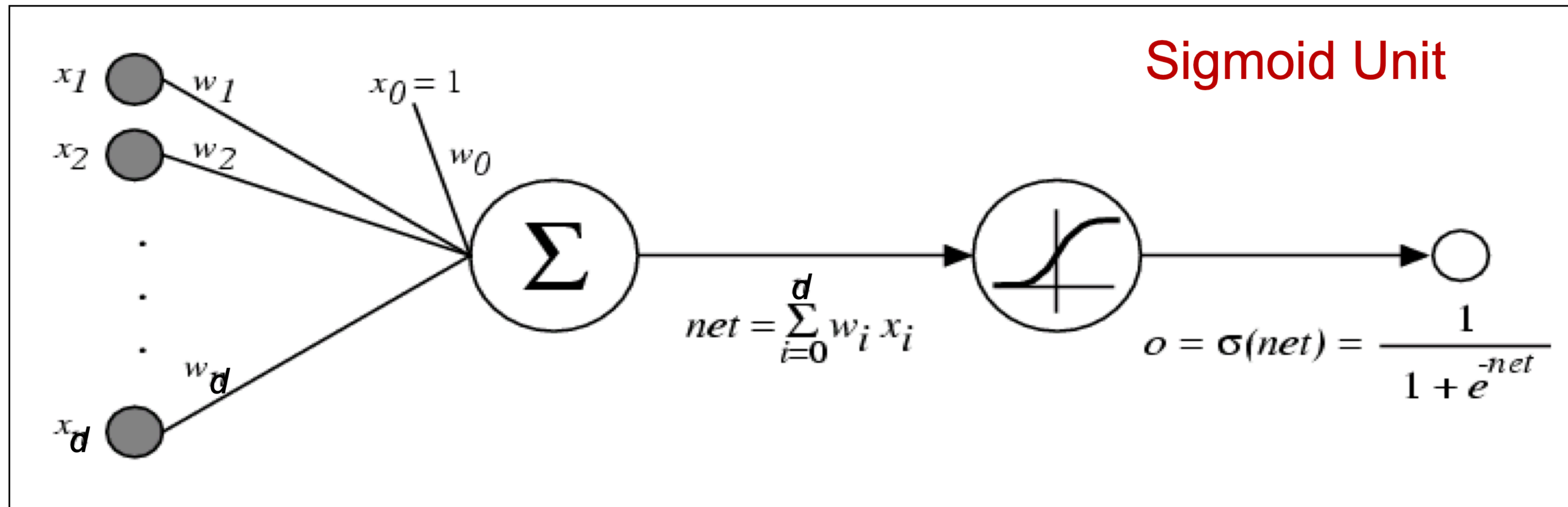
Completely unseen before deployment

Realistic setting

Neural Networks, Backpropagation

Logistic Function as a Graph

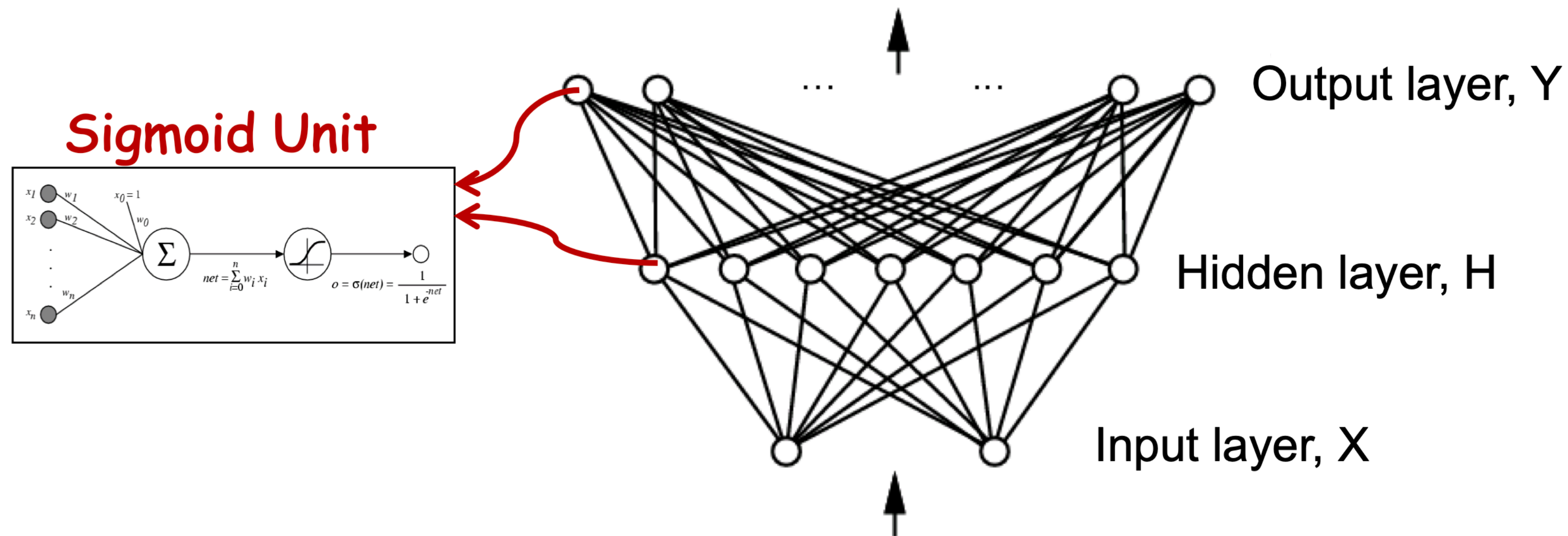
$$\text{Output, } o(\mathbf{x}) = \sigma(w_0 + \sum_i w_i X_i) = \frac{1}{1 + \exp(-(w_0 + \sum_i w_i X_i))}$$



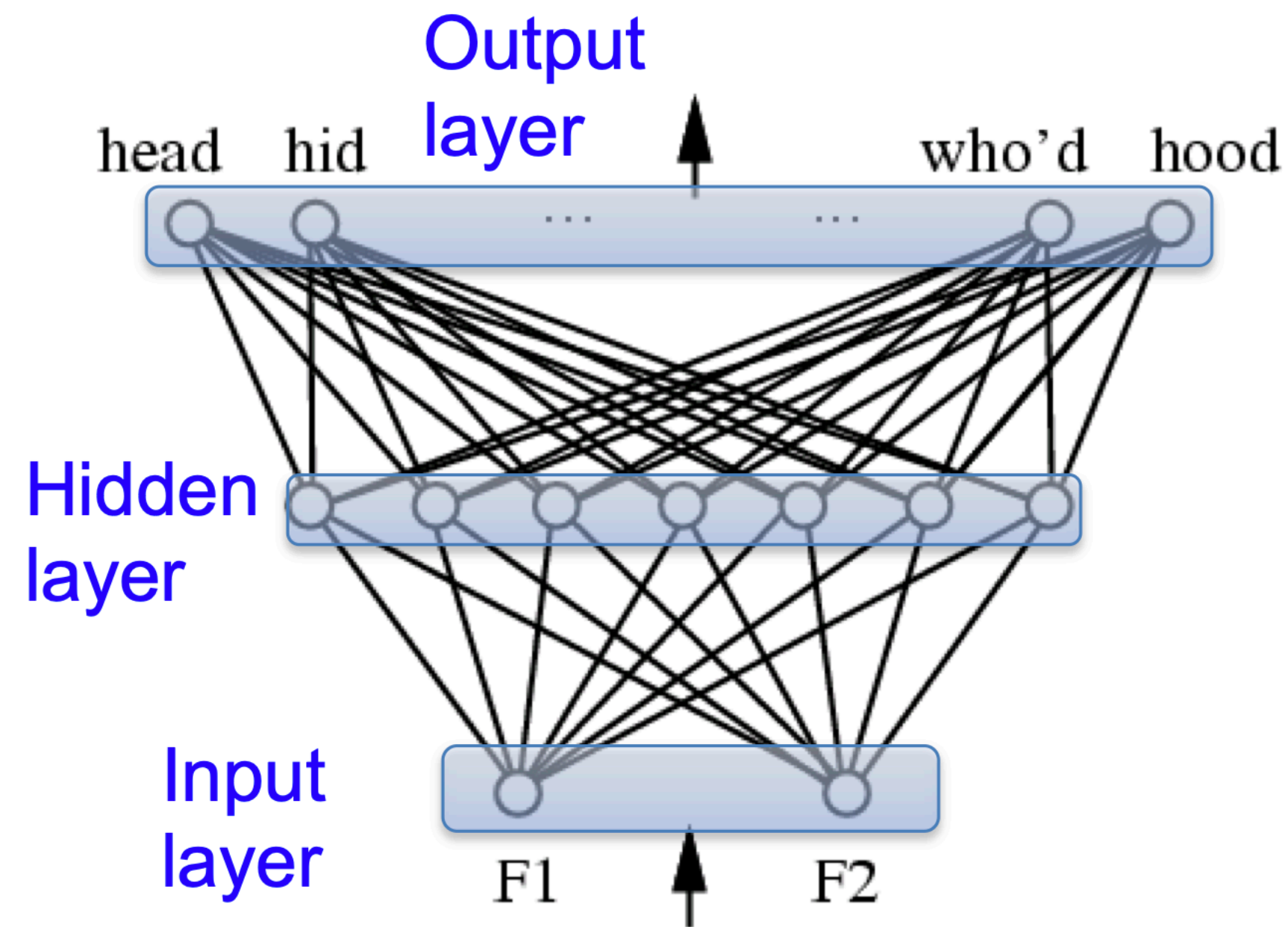
Computation Graph

Neural Networks

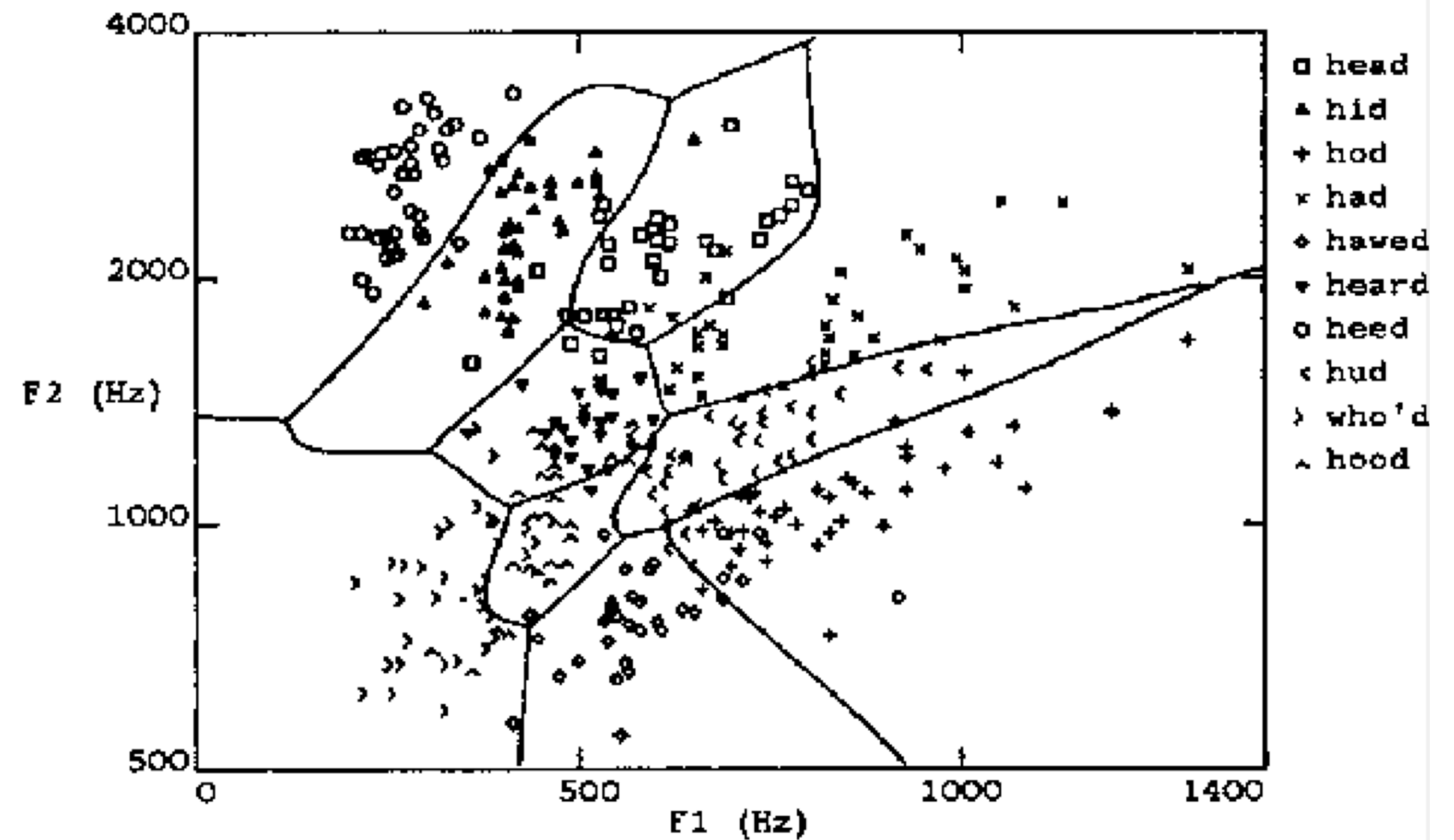
- f can be a **non-linear** function
- X (vector of) continuous and/or discrete variables
- Y (**vector** of) continuous and/or discrete variables
- Neural networks - Represent f by network of sigmoid (more recently ReLU – next lecture) units :



Multilayer Networks of Sigmoid Units



Two layers of logistic units



Highly non-linear decision surface

Expressive Capabilities of ANNs

Continuous functions:

- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989]
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988].

Prediction using Neural Networks

Prediction – Given neural network (hidden units and weights), use it to predict the label of a test point

Forward Propagation –

Start from input layer

For each subsequent layer, compute output of sigmoid unit

Sigmoid unit:

$$o(\mathbf{x}) = \sigma(w_0 + \sum_i w_i x_i)$$

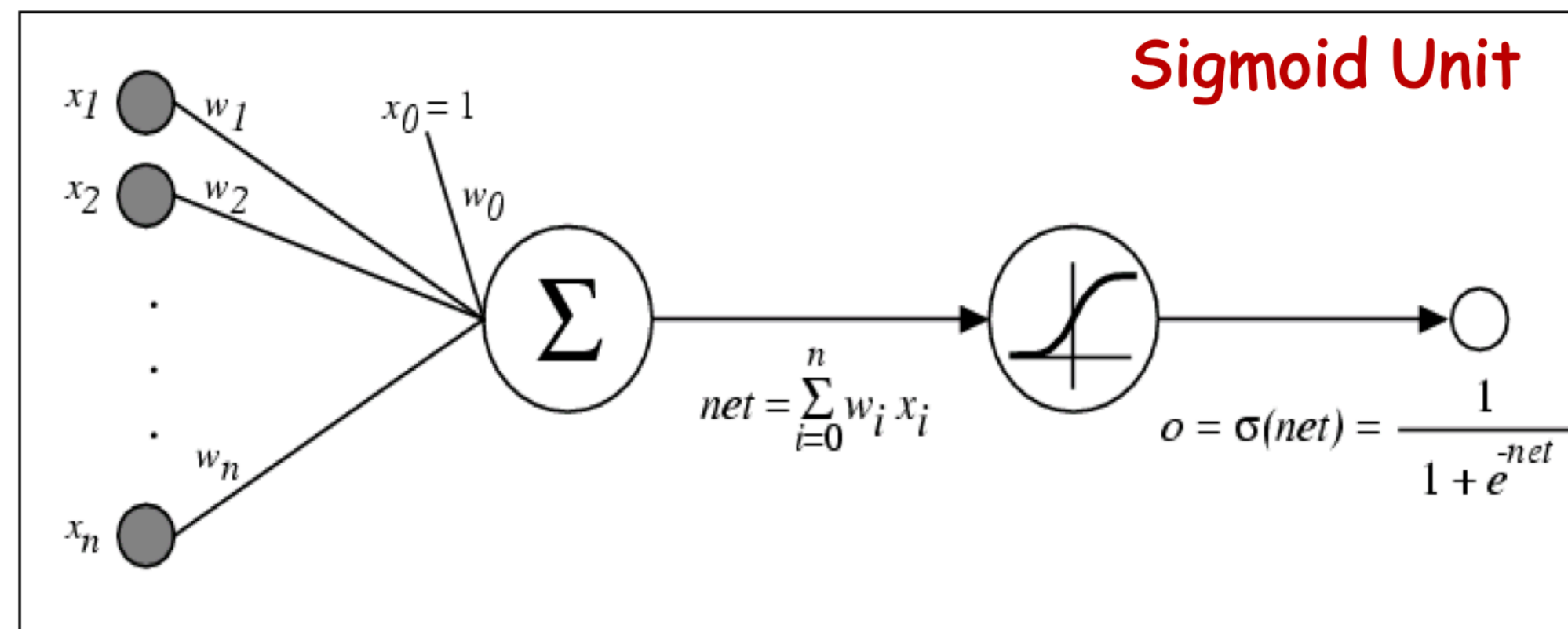
1-Hidden layer,
1 output NN:

$$o(\mathbf{x}) = \sigma \left(w_0 + \sum_h w_h \underbrace{\sigma \left(w_0^h + \sum_i w_i^h x_i \right)}_{o_h} \right)$$

Gradient descent for training NNs

$$w \leftarrow w - \alpha \cdot \frac{\partial L}{\partial w}$$

Gradient decent for 1 node:



$$\frac{\partial o}{\partial w_i} = \frac{\partial o}{\partial net} \cdot \frac{\partial net}{\partial w_i} = o(1 - o)x_i$$

Chain rule

Univariate Chain Rule

- We've already been using the univariate Chain Rule.
- Recall: if $f(x)$ and $x(t)$ are univariate functions, then

$$\frac{d}{dt}f(x(t)) = \frac{df}{dx} \frac{dx}{dt}.$$

Example:

$$z = wx + b$$

$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

Let's compute the loss derivatives.

Example of Chain Rule

$$\begin{aligned}\mathcal{L} &= \frac{1}{2}(\sigma(wx + b) - t)^2 \\ \frac{\partial \mathcal{L}}{\partial w} &= \frac{\partial}{\partial w} \left[\frac{1}{2}(\sigma(wx + b) - t)^2 \right] \\ &= \frac{1}{2} \frac{\partial}{\partial w} (\sigma(wx + b) - t)^2 \\ &= (\sigma(wx + b) - t) \frac{\partial}{\partial w} (\sigma(wx + b) - t) \\ &= (\sigma(wx + b) - t) \sigma'(wx + b) \frac{\partial}{\partial w} (wx + b) \\ &= (\sigma(wx + b) - t) \sigma'(wx + b) x\end{aligned}$$

Using Chain Rules

Computing the loss:

$$z = wx + b$$

$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

Computing the derivatives:

$$\frac{d\mathcal{L}}{dy} = y - t$$

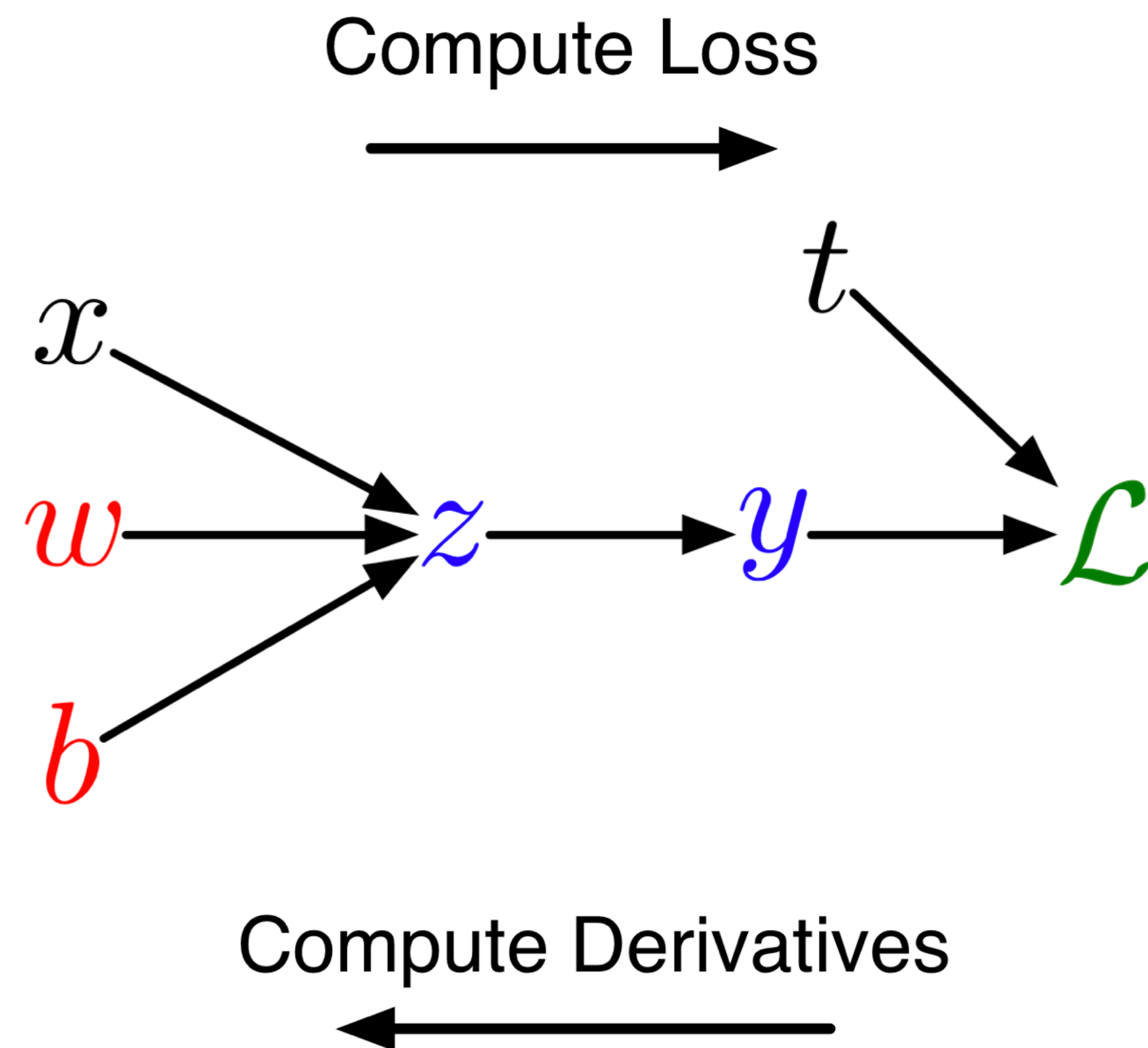
$$\frac{d\mathcal{L}}{dz} = \frac{d\mathcal{L}}{dy} \sigma'(z)$$

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{d\mathcal{L}}{dz} x$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{d\mathcal{L}}{dz}$$

The goal isn't to obtain closed-form solutions, but to be able to write a program that efficiently computes the derivatives

Univariate Chain Rule



A Slightly More Convenient Notation

Use \bar{y} to denote the derivative $d\mathcal{L}/dy$, sometimes called the **error signal**

Computing the loss:

$$z = wx + b$$

$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

Computing the derivatives:

$$\bar{y} = y - t$$

$$\bar{z} = \bar{y} \sigma'(z)$$

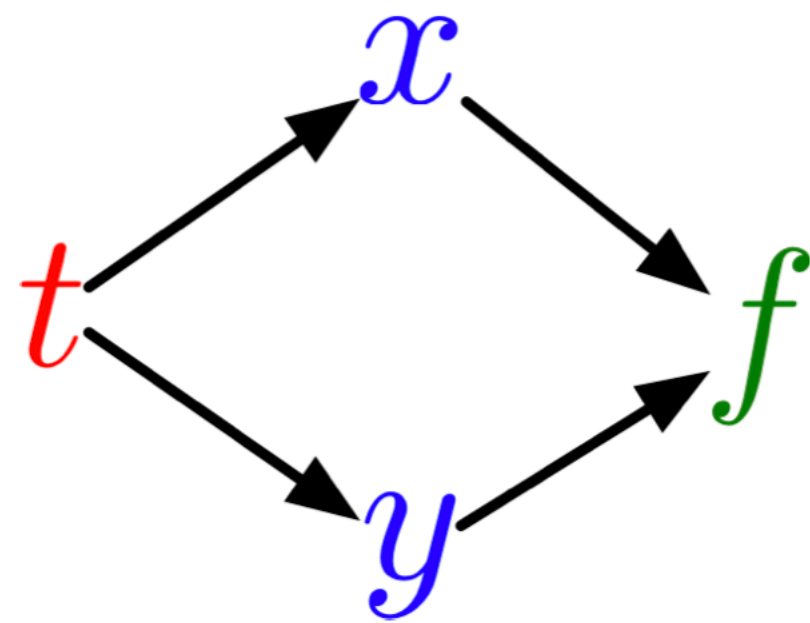
$$\bar{w} = \bar{z} x$$

$$\bar{b} = \bar{z}$$

Multivariate Chain Rule

Problem: what if the computation graph has **fan-out** > 1 ?

This requires the **multivariate Chain Rule**!



$$\frac{d}{dt} f(x(t), y(t)) = \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt}$$

Example:

$$f(x, y) = y + e^{xy}$$

$$x(t) = \cos t$$

$$y(t) = t^2$$

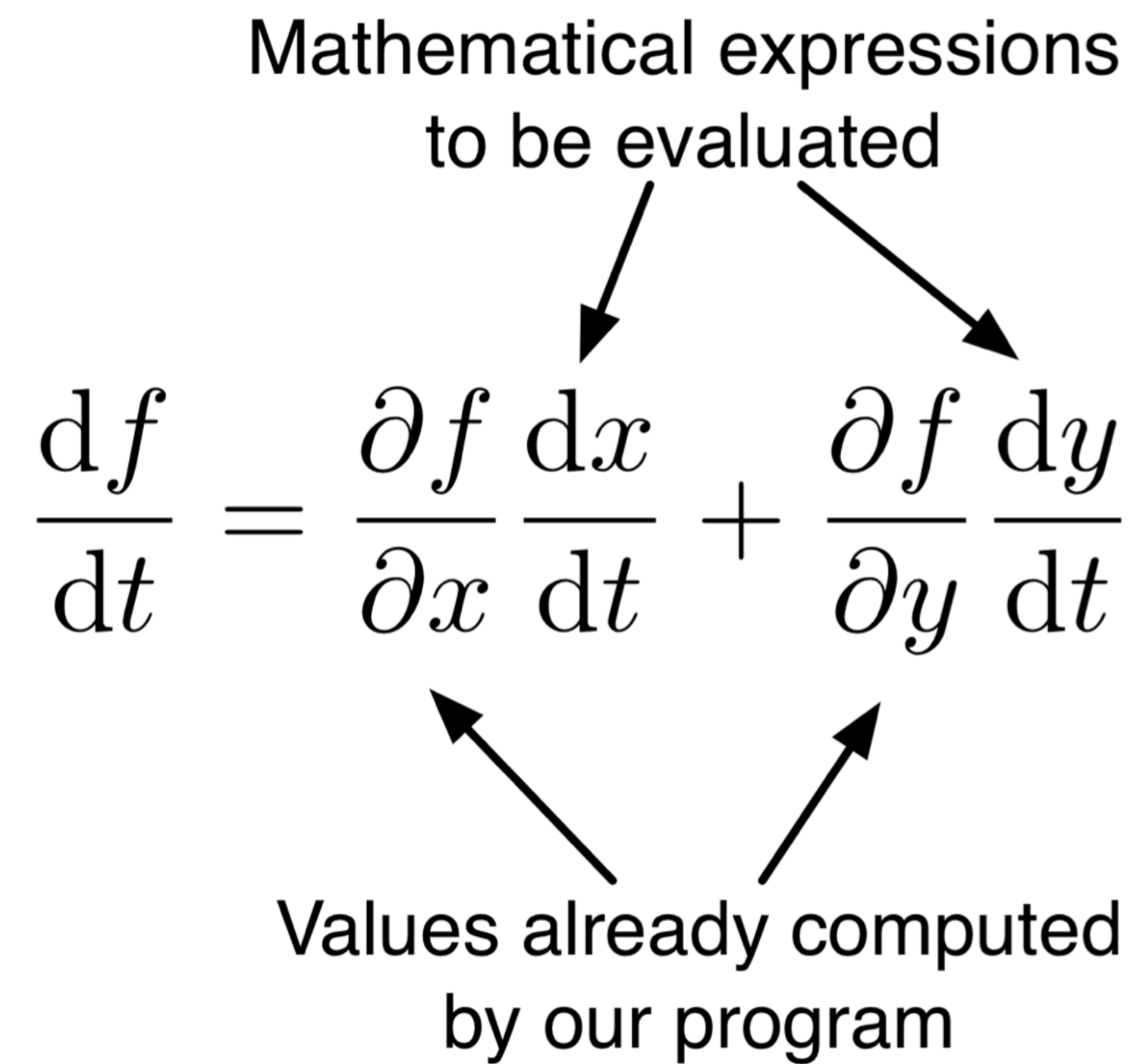
$$\begin{aligned} \frac{df}{dt} &= \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt} \\ &= (ye^{xy}) \cdot (-\sin t) + (1 + xe^{xy}) \cdot 2t \end{aligned}$$

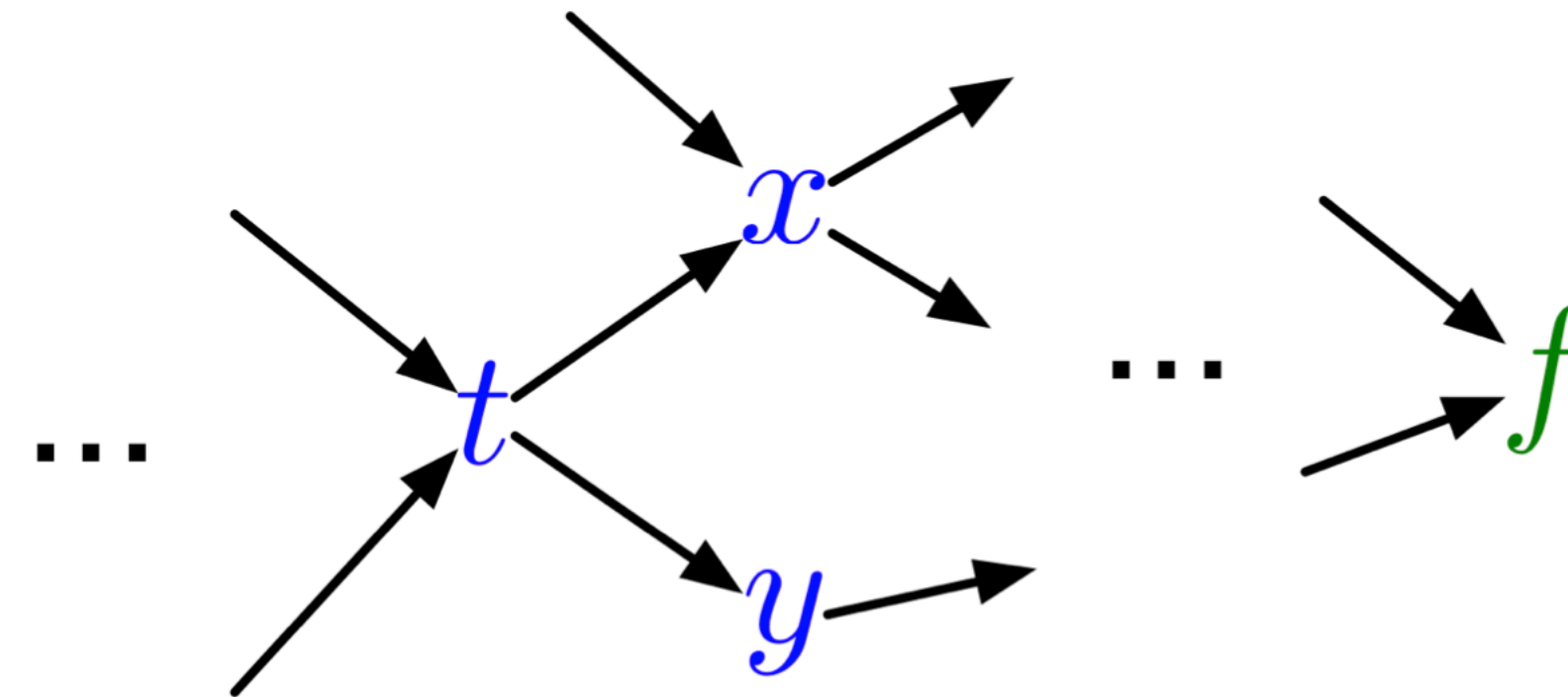
Multivariate Chain Rule

Mathematical expressions
to be evaluated

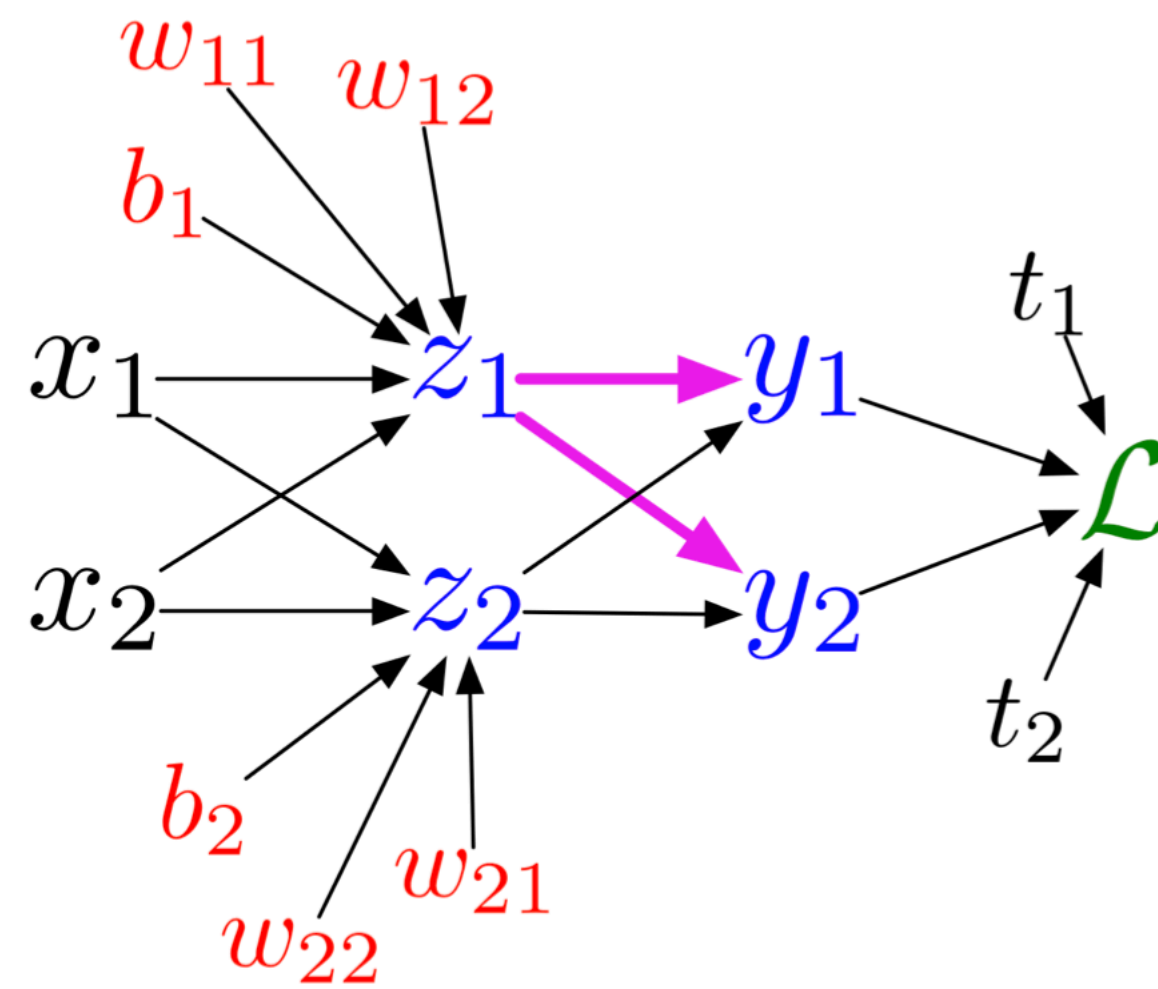
$$\frac{df}{dt} = \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt}$$

Values already computed
by our program





Another Example



$$z_\ell = \sum_j w_{\ell j} x_j + b_\ell$$

$$y_k = \frac{e^{z_k}}{\sum_\ell e^{z_\ell}}$$

$$\mathcal{L} = - \sum_k t_k \log y_k$$

Backpropagation

Let v_1, \dots, v_N be a **topological ordering** of the computation graph (i.e. parents come before children.)

v_N denotes the variable we're trying to compute derivatives of (e.g. loss).

forward pass

For $i = 1, \dots, N$

Compute v_i as a function of $\text{Pa}(v_i)$

backward pass

$\overline{v_N} = 1$

For $i = N - 1, \dots, 1$

$$\overline{v_i} = \sum_{j \in \text{Ch}(v_i)} \overline{v_j} \frac{\partial v_j}{\partial v_i}$$

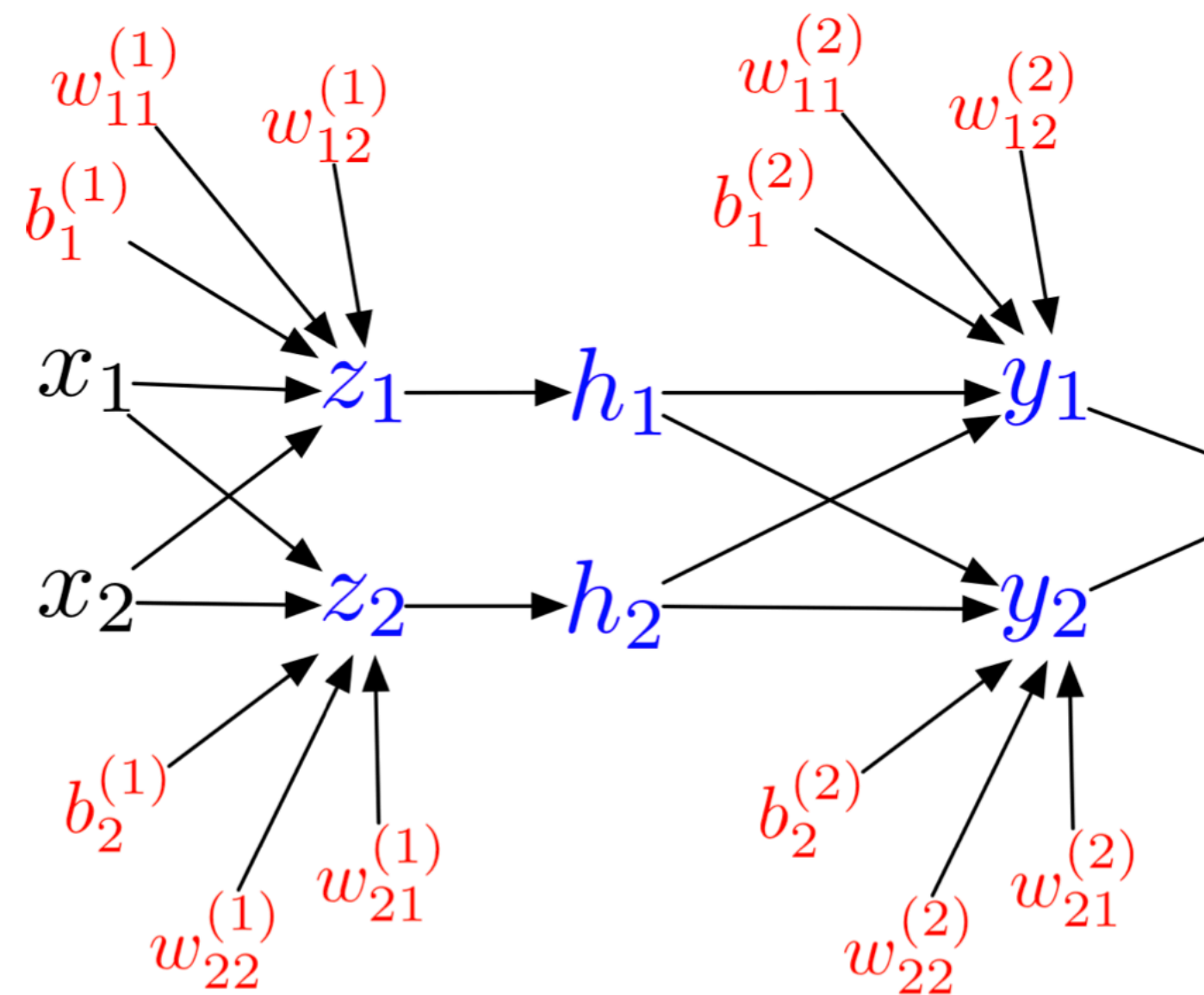
[1] David Rumelhart, Geoffrey Hinton, Ronald Williams. Learning representations by back-propagating errors. Nature. 1986

Backpropagation

Multilayer Perceptron (multiple outputs):

Backward pass:

Forward pass:



$$z_i = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$$

$$h_i = \sigma(z_i)$$

$$y_k = \sum_i w_{ki}^{(2)} h_i + b_k^{(2)}$$

$$\mathcal{L} = \frac{1}{2} \sum_k (y_k - t_k)^2$$

$$\bar{\mathcal{L}} = 1$$

$$\bar{y}_k = \bar{\mathcal{L}} (y_k - t_k)$$

$$\bar{w}_{ki}^{(2)} = \bar{y}_k h_i$$

$$\bar{b}_k^{(2)} = \bar{y}_k$$

$$\bar{h}_i = \sum_k \bar{y}_k w_{ki}^{(2)}$$

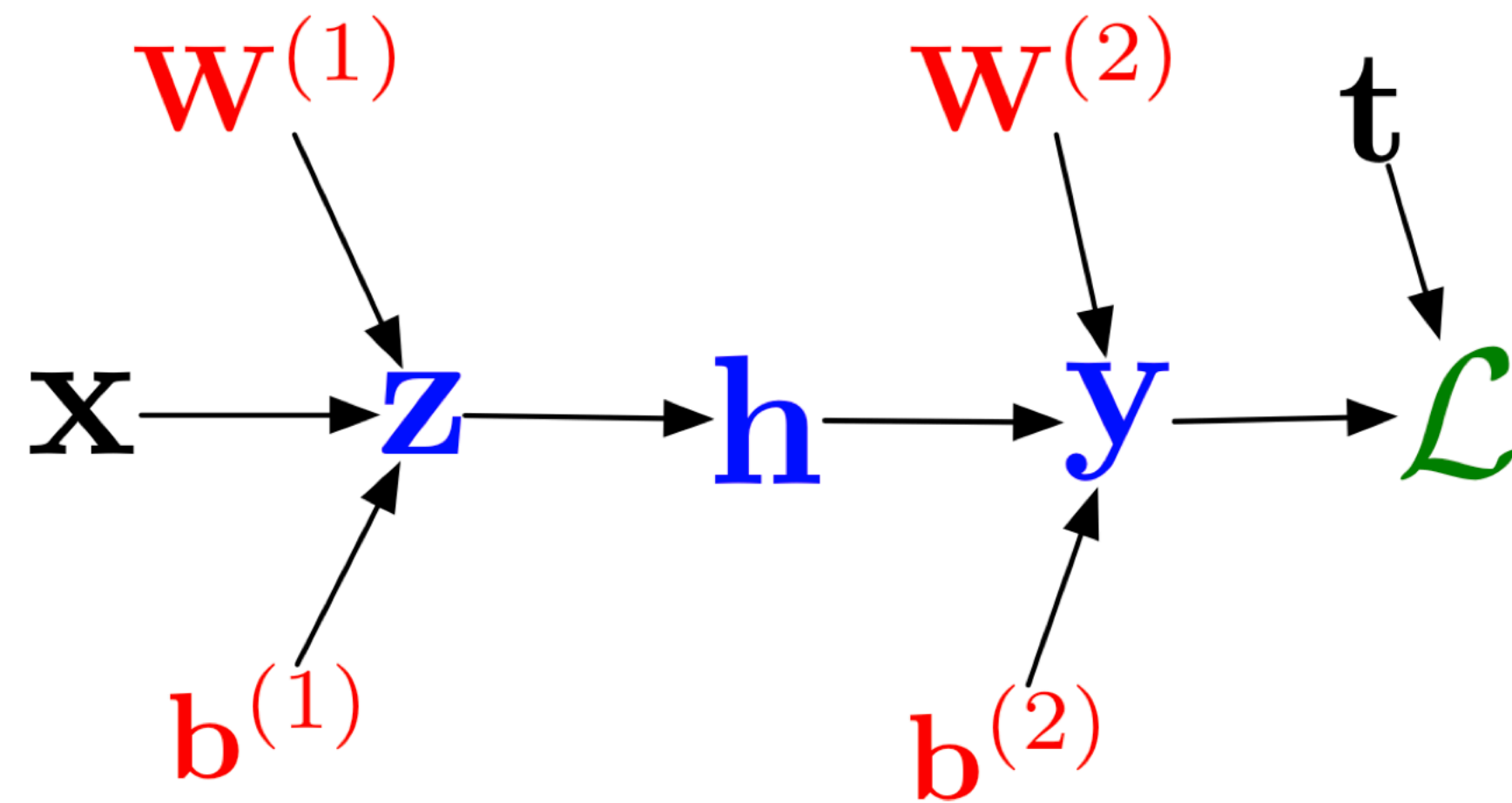
$$\bar{z}_i = \bar{h}_i \sigma'(z_i)$$

$$\bar{w}_{ij}^{(1)} = \bar{z}_i x_j$$

$$\bar{b}_i^{(1)} = \bar{z}_i$$

Backpropagation

In vectorized form:



Forward pass:

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$

$$\mathbf{h} = \sigma(\mathbf{z})$$

$$\mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\mathcal{L} = \frac{1}{2} \|\mathbf{t} - \mathbf{y}\|^2$$

Backward pass:

$$\bar{\mathcal{L}} = 1$$

$$\bar{\mathbf{y}} = \bar{\mathcal{L}} (\mathbf{y} - \mathbf{t})$$

$$\overline{\mathbf{W}^{(2)}} = \bar{\mathbf{y}}\mathbf{h}^\top$$

$$\overline{\mathbf{b}^{(2)}} = \bar{\mathbf{y}}$$

$$\bar{\mathbf{h}} = \mathbf{W}^{(2)\top} \bar{\mathbf{y}}$$

$$\bar{\mathbf{z}} = \bar{\mathbf{h}} \circ \sigma'(\mathbf{z})$$

$$\overline{\mathbf{W}^{(1)}} = \bar{\mathbf{z}}\mathbf{x}^\top$$

$$\overline{\mathbf{b}^{(1)}} = \bar{\mathbf{z}}$$

Backpropagation

- Backprop is used to train the overwhelming majority of neural nets today.
 - Even optimization algorithms much fancier than gradient descent (e.g. second-order methods) use backprop to compute the gradients.
- Despite its practical success, backprop is believed to be neurally implausible.
 - No evidence for biological signals analogous to error derivatives.
 - All the biologically plausible alternatives we know about learn much more slowly (on computers).
 - So how on earth does the brain learn?

Stochastic Gradient Descent

Vanilla backpropagation training is slow with lot of data and lot of weights

Denote the loss of a single data example x_i as $l(x_i)$, the training loss L is:

$$L = \mathbb{E}_{x \sim p_{data}} l(x) \approx \frac{1}{N} \sum_{i=1}^N l(x_i) \quad \text{N is the size of the entire training dataset}$$

This is slow on the entire training dataset, thus we approximate it:

$$\nabla L = \nabla \mathbb{E}_{x \sim p_{data}} l(x) \approx \nabla \frac{1}{n} \sum_{i=1}^n l(x_i) \quad \begin{array}{l} n \text{ is the size of a} \\ \text{random minibatch} \\ \text{(batch size)} \end{array} \quad \text{n can be as small as one}$$

Background

A Recipe for Machine Learning

1. Given training data:

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function

$$\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x}_i)$$

– Loss function

$$\ell(\hat{\mathbf{y}}, \mathbf{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

4. Train with SGD:

(take small steps
opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

Thank You!