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

COMP 5212
Machine Learning
Lecture 18

PGM $\longrightarrow$ HMM

Neural Networks, Backpropagation

Junxian He
Nov 7, 2024

Logistic Function as a Graph

Computation Graph

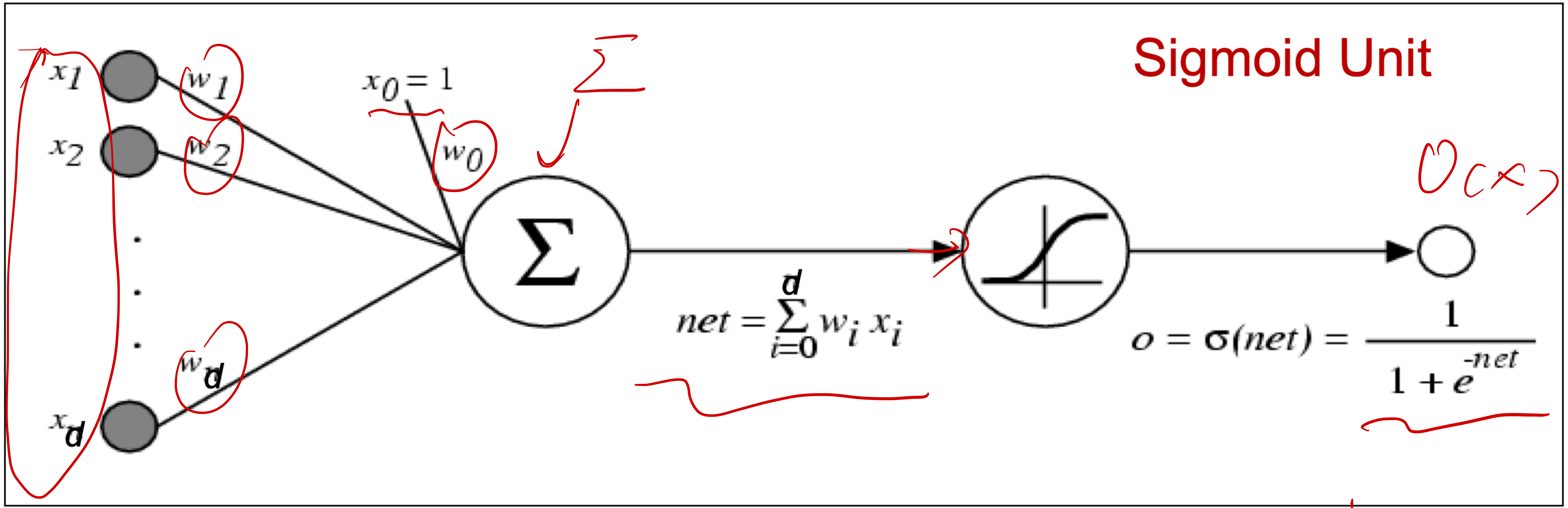
Logistic Function as a Graph

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

$exp(-\sum_i w_i x_i + w_0)$

Logistic Function as a Graph

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



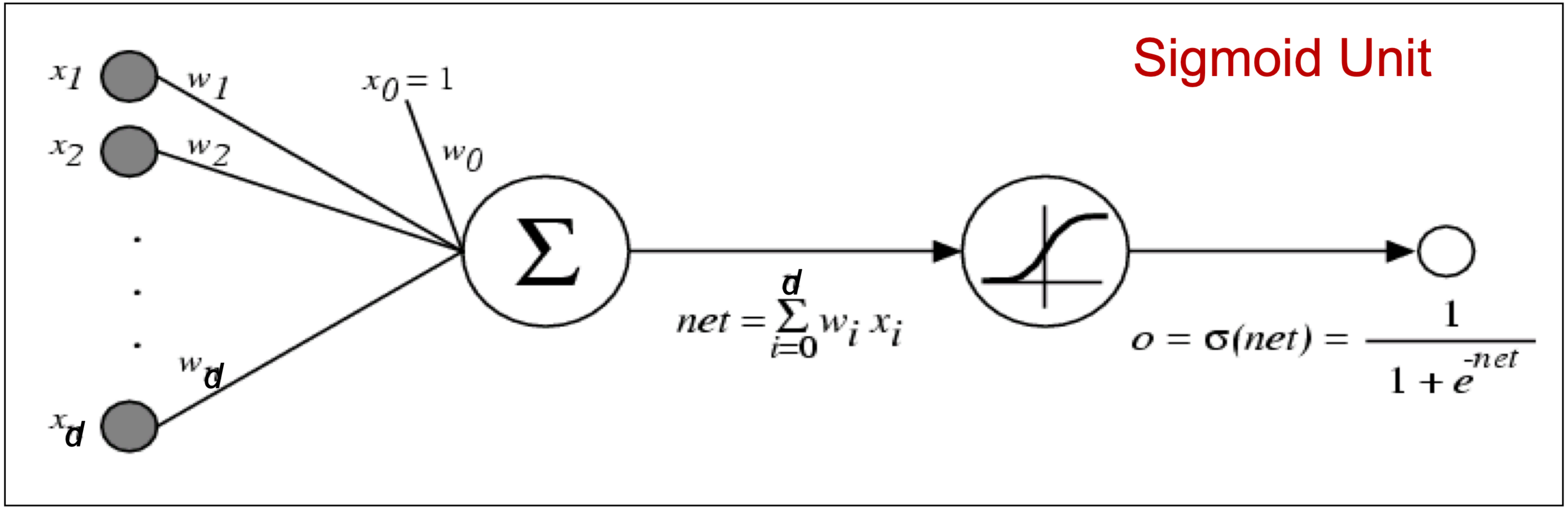
Σ

$O(x)$

$\dfrac{1}{1+e^{-x}}$

Logistic Function as a Graph

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

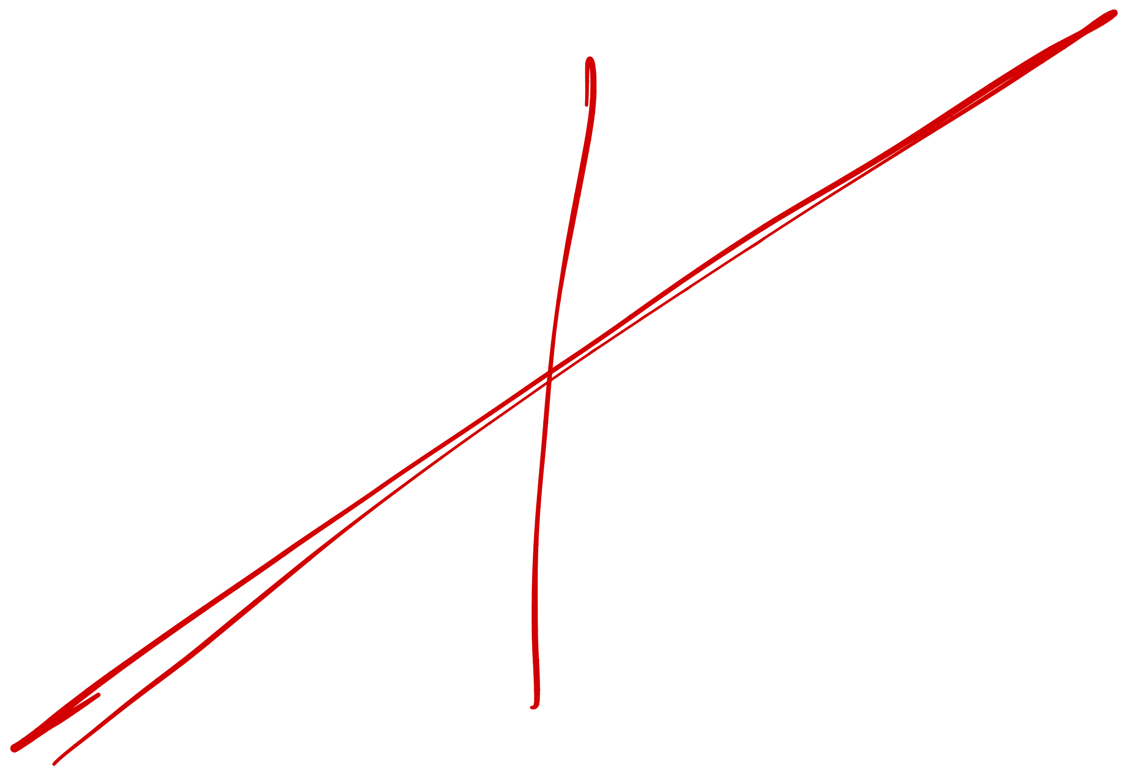


Computation Graph

Neural Networks

Neural Networks

$\frac{1}{1+e^{-x}}$



- 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

PGM:

random variable

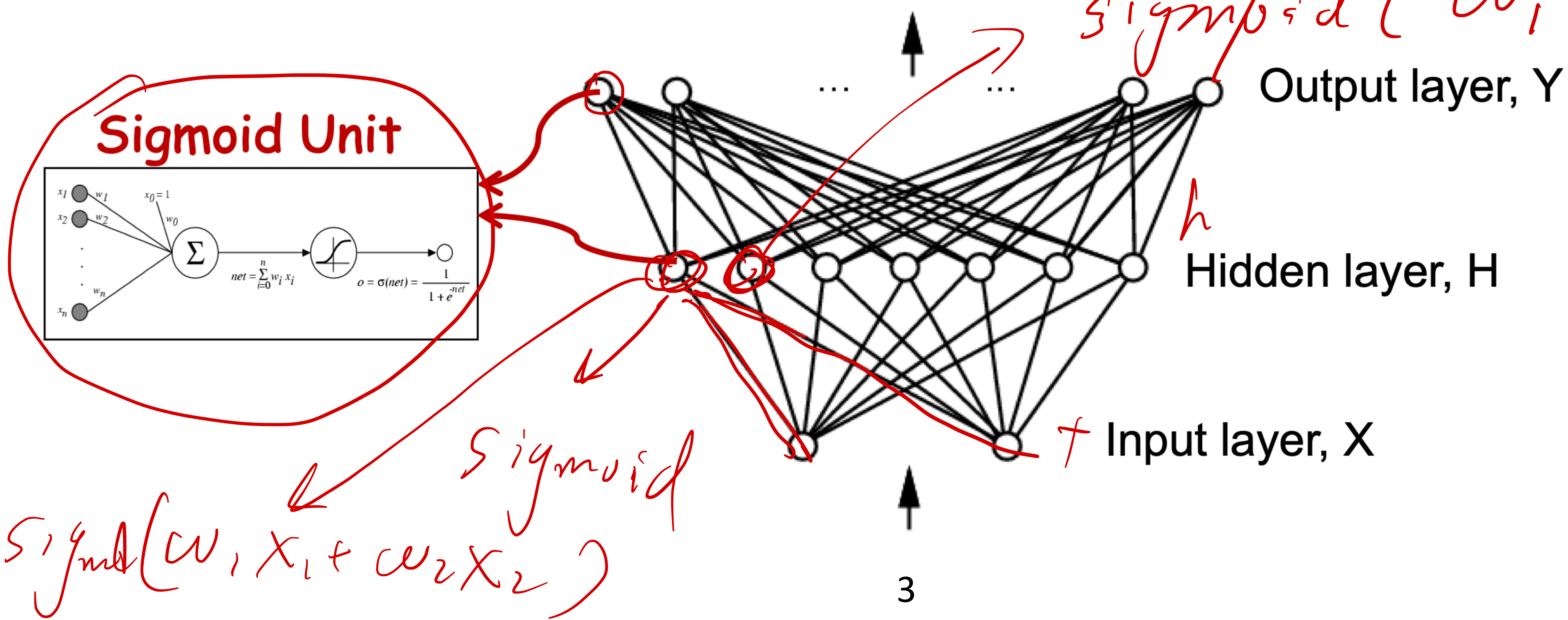
$e(P(h|X))$

deterministic

- f can be a **non-linear** function $h = g(x)$
- X (vector of) continuous and/or discrete variables
- Y (**vector** of) continuous and/or discrete variables

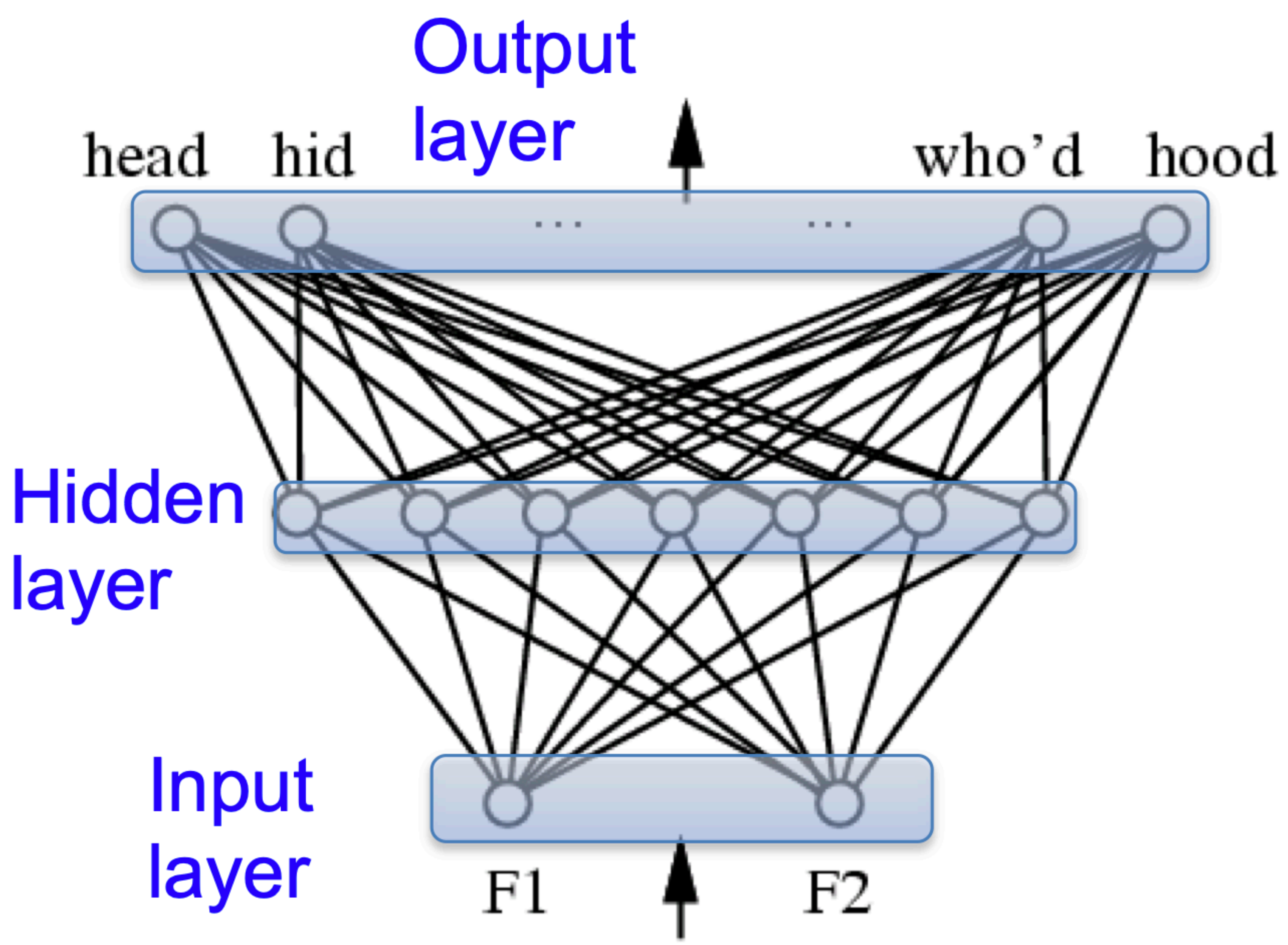
  - Neural networks - Represent f by ***network*** of sigmoid (more recently ReLU – next lecture) units :

$\text{sigmoid}(w_1^{(2)} x_1 + w_2^{(2)} x_2)$



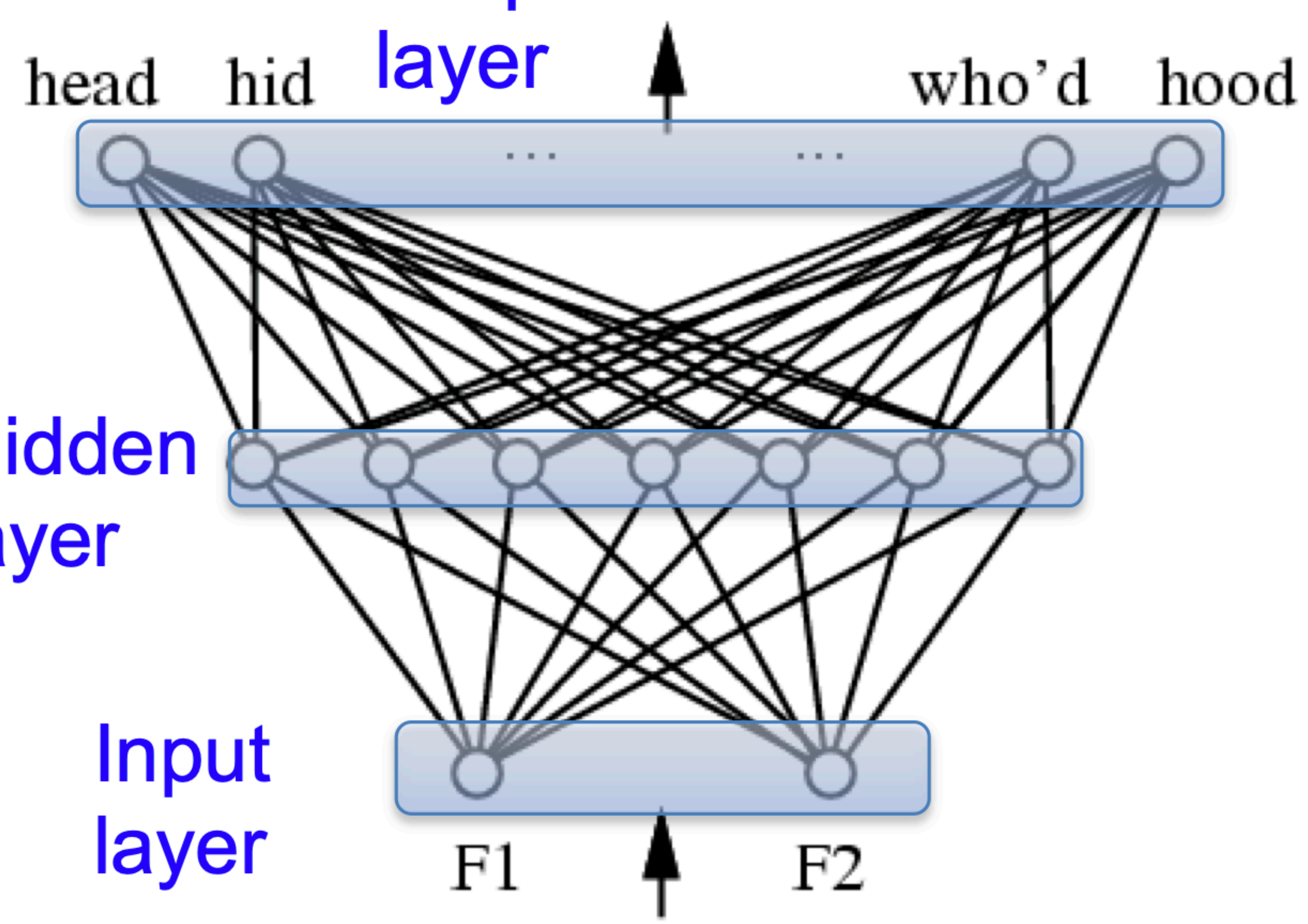
Multilayer Networks of Sigmoid Units

Multilayer Networks of Sigmoid Units



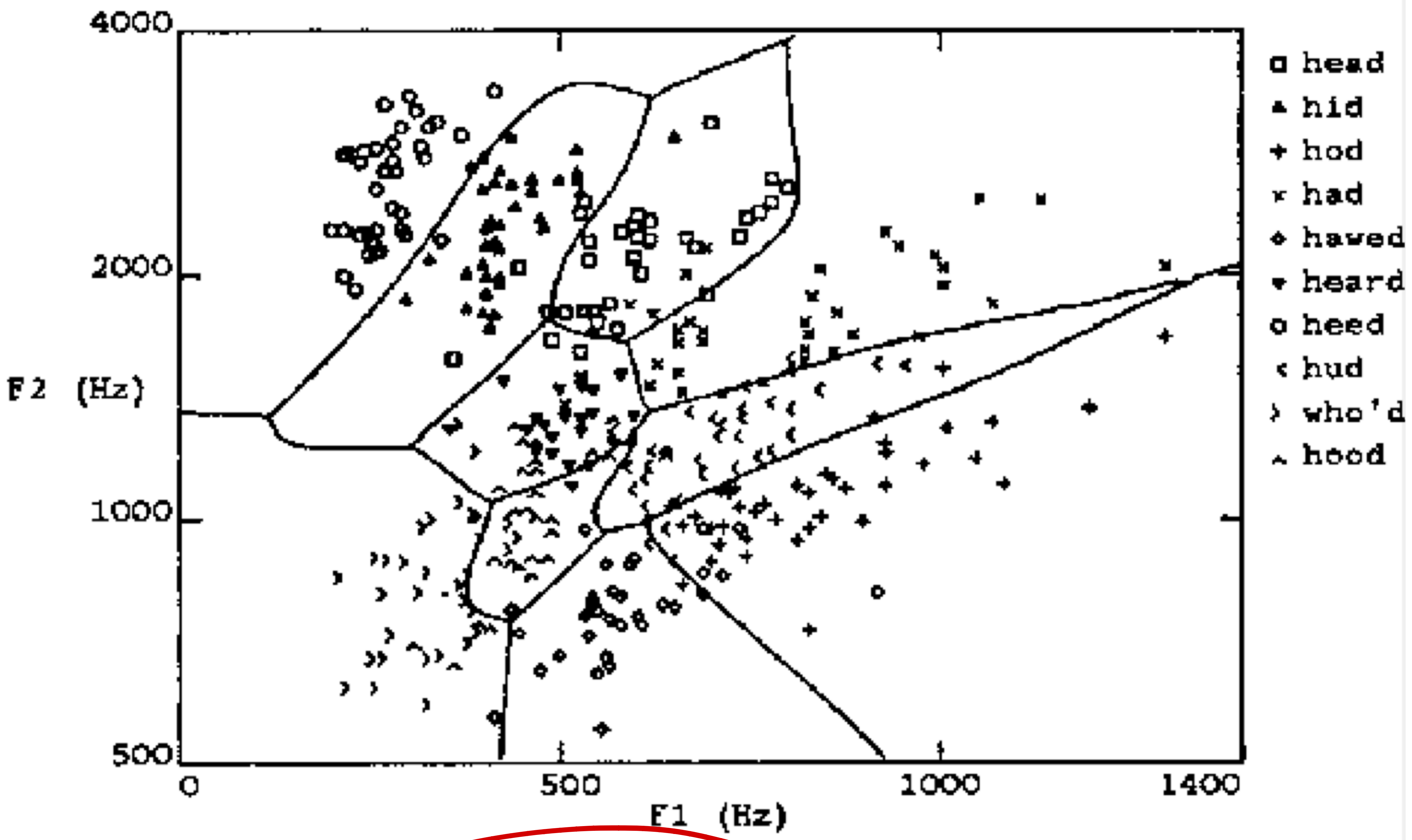
Two layers of logistic units

Multilayer Networks of Sigmoid Units



Two layers of logistic units

Universal function approximators



Highly non-linear decision surface

More Applications

Neural Network trained to drive a car!

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 using Neural Networks

Parameters

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

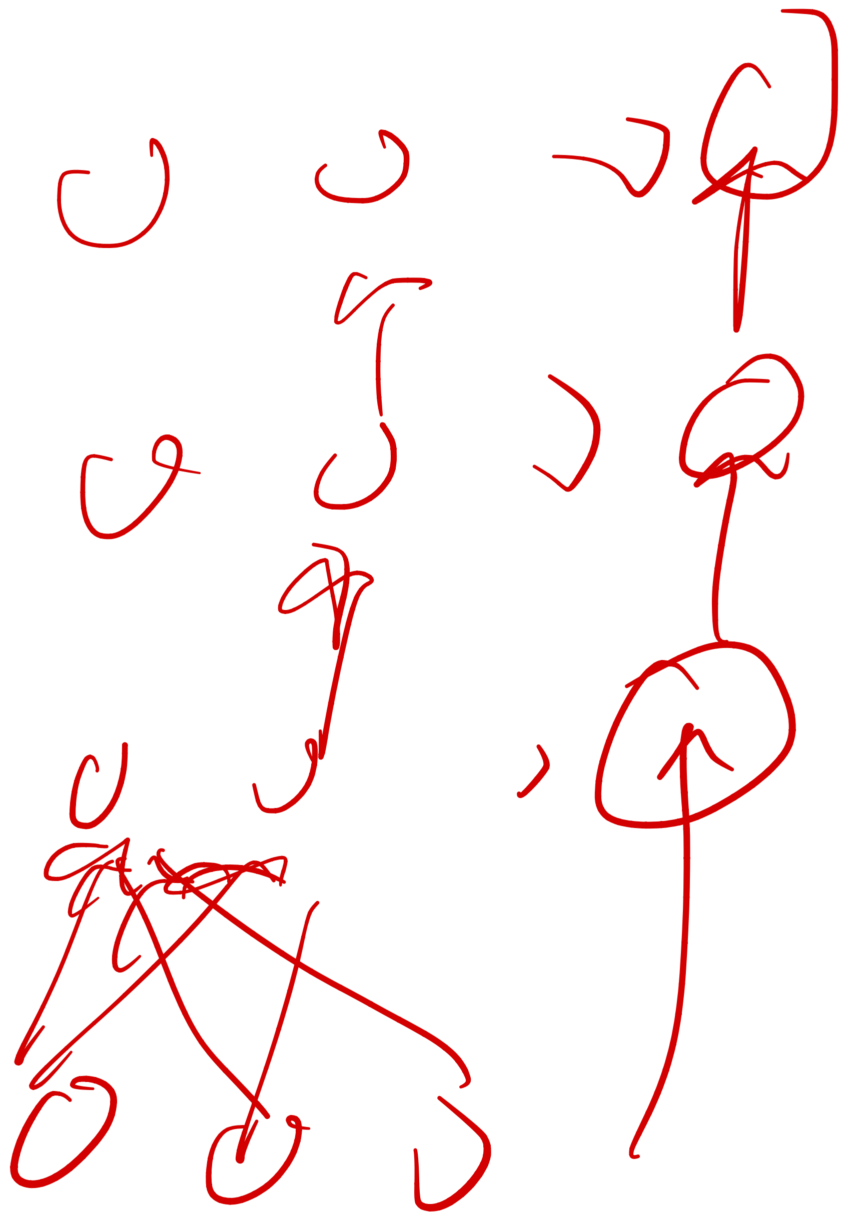
Prediction using Neural Networks

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:

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



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:

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

Objective Functions for NNs

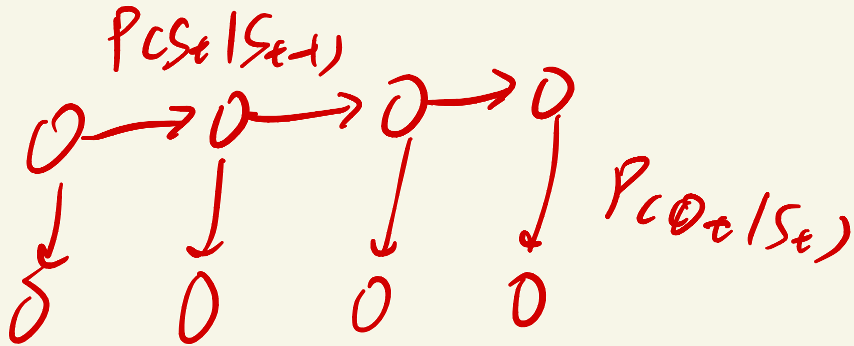
- Regression:
  - Use the same objective as Linear Regression
  - Quadratic loss (i.e. mean squared error)

MSE

- Classification:
  - Use the same objective as Logistic Regression
  - Cross-entropy (i.e. negative log likelihood)
  - This requires probabilities, so we add an additional "softmax" layer at the end of our network

MLE

parametric function



$$P(S_t|S_{t-1}) = NN^{(1)}(S_{t-1}, O_t)$$

$$P(O_t|S_t) = NN^{(2)}(S_t)$$

Neural HMM

Gradient descent for training NNs

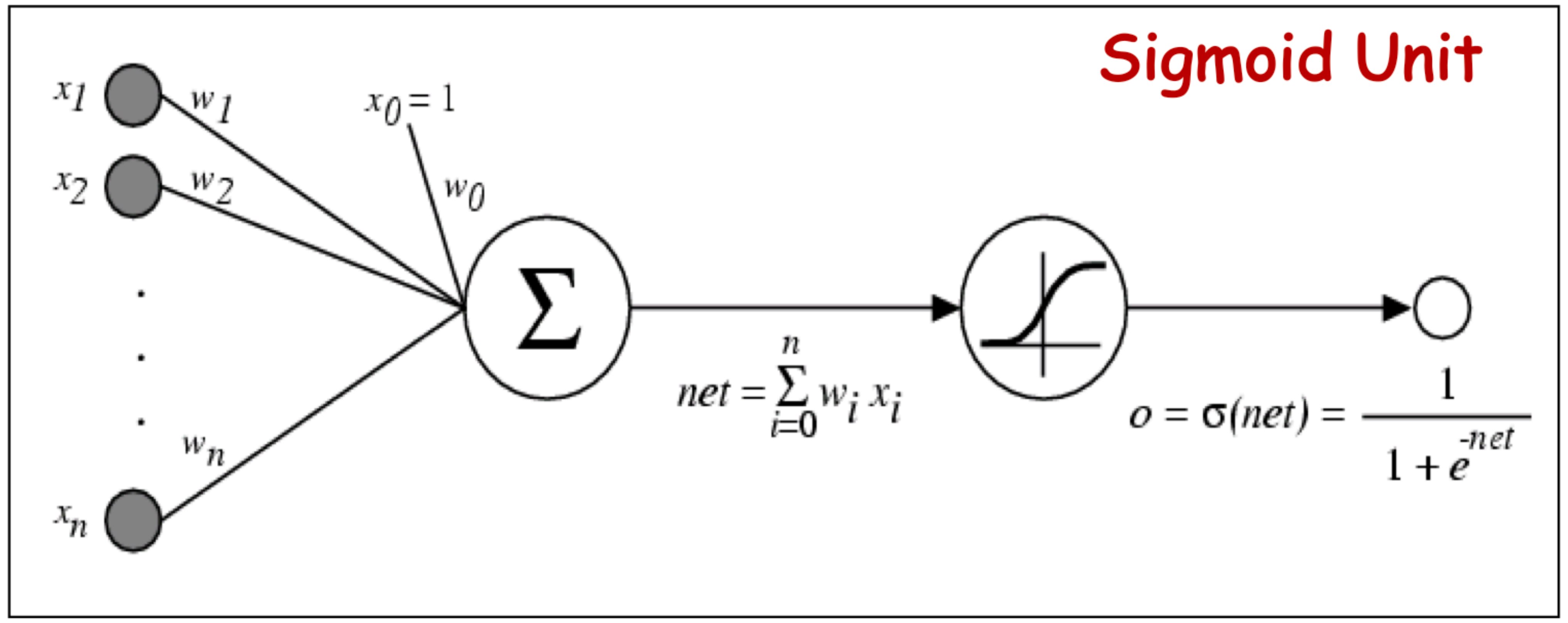
Gradient descent for training NNs

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

Gradient descent for training NNs

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

Gradient decent for 1 node:



Gradient descent for training NNs

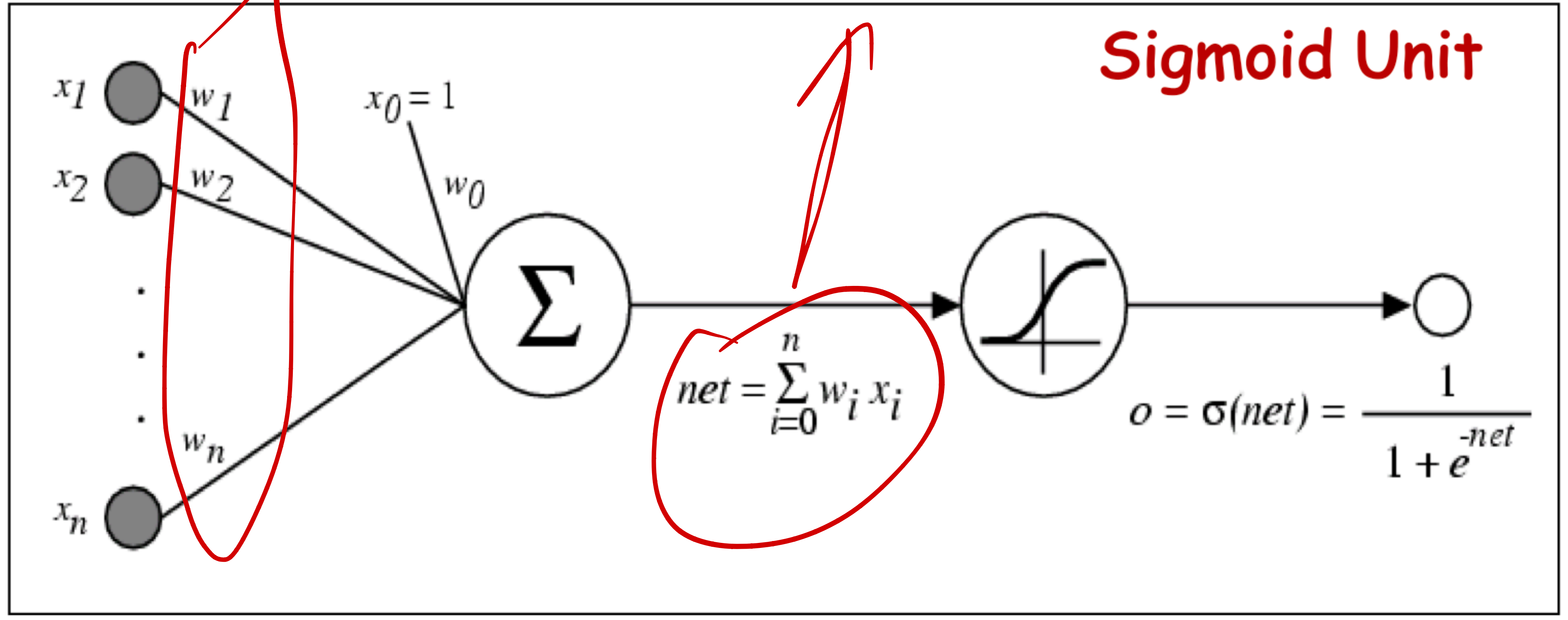
backward()

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

$$o = \frac{1}{1+e^{-net}}$$

Gradient decent for 1 node:

net



$$\frac{\partial o}{\partial w} = \frac{\partial o}{\partial net} \cdot \frac{\partial net}{\partial w_i}$$

chain rule

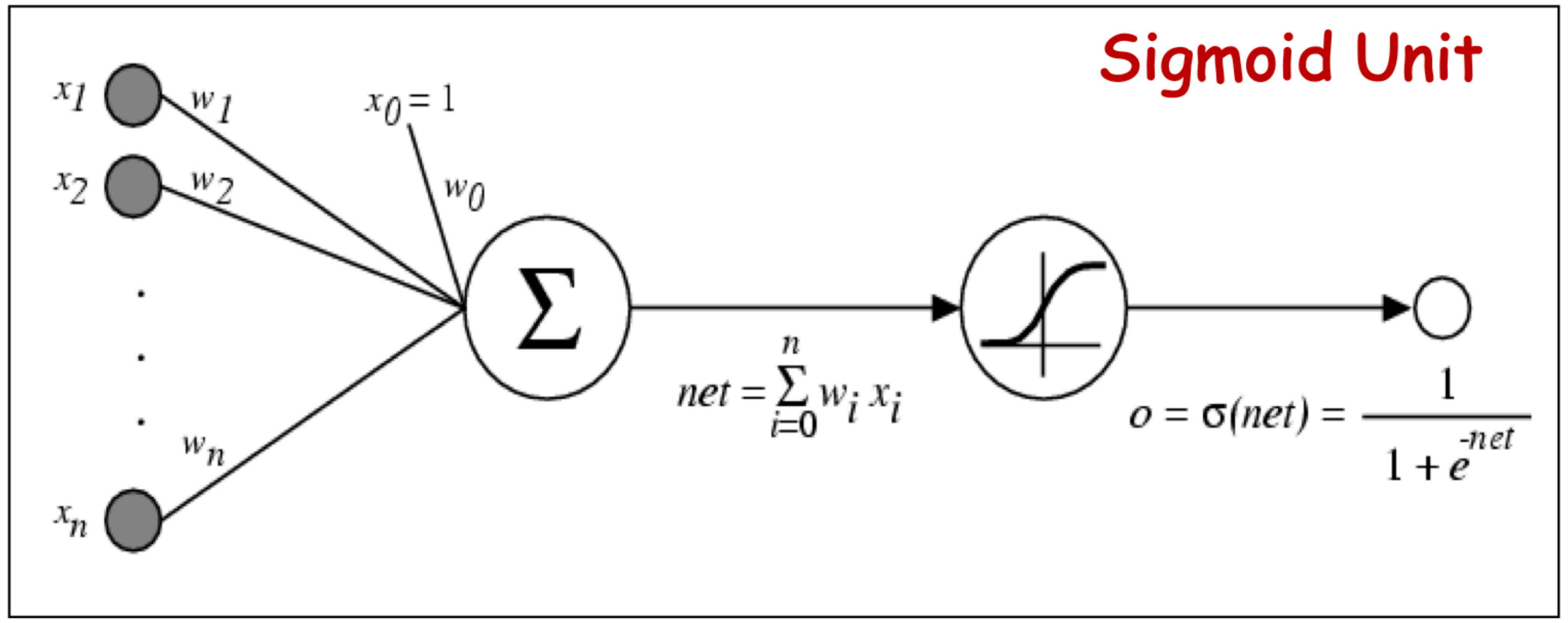
$$\frac{\partial o}{\partial w_i} = \frac{\partial o}{\partial net}\frac{\partial net}{\partial w_i} = o(1-o)x_i$$

chain rule

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

$$\frac{\mathrm{d}}{\mathrm{d}t} f(x(t)) = \frac{\mathrm{d}f}{\mathrm{d}x} \frac{\mathrm{d}x}{\mathrm{d}t}.$$

$$\frac{df}{dx} \cdot \frac{dx}{dt}$$

$$f(x(t))$$

$$f(x(y(z(t))))$$

$$\frac{df}{dt}$$

Univariate Chain Rule

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

$$\frac{\mathrm{d}}{\mathrm{d}t} f(x(t)) = \frac{\mathrm{d}f}{\mathrm{d}x}\frac{\mathrm{d}x}{\mathrm{d}t}.$$

Example:

input

hidden

$$z = wx + b$$

hidden

$$y = \sigma(z)$$

loss

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

Let's compute the loss derivatives.

Example of Chain Rule

$y = \sigma(wx+b) - t$

$L = y^2$

$$\mathcal{L} = \frac{1}{2}(\sigma(wx+b)-t)^2$$

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial}{\partial w}\left[\frac{1}{2}(\sigma(wx+b)-t)^2\right]$$

$$= \frac{1}{2}\frac{\partial}{\partial w}(\sigma(wx+b)-t)^2$$

y

$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial w}$

$$= (\sigma(wx+b)-t)\frac{\partial}{\partial w}(\sigma(wx+b)-t)$$

$$= (\sigma(wx+b)-t)\sigma'(wx+b)\frac{\partial}{\partial w}(wx+b)$$

$$= (\sigma(wx+b)-t)\sigma'(wx+b)x$$

Using Chain Rules

Computing the loss:

$$z = wx + b$$

$$y = \sigma(z)$$

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

Computing the derivatives:

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

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y}\,\sigma'(z)$$

$$\frac{\partial\mathcal{L}}{\partial w} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}\,x$$

$$\frac{\partial\mathcal{L}}{\partial b} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}$$

Using Chain Rules

Computing the loss:

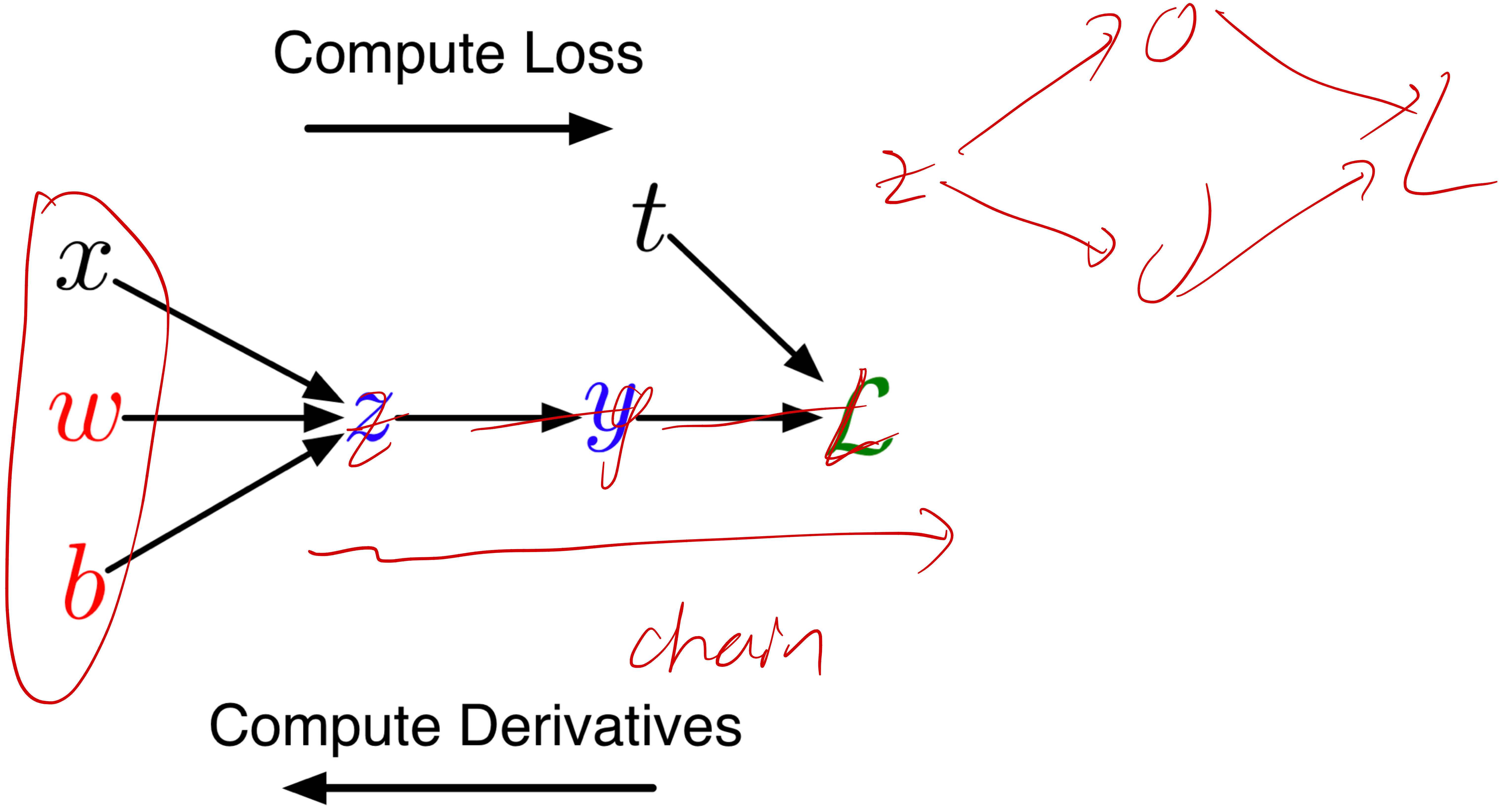
$$
\begin{aligned}
z &= wx + b \\
y &= \sigma(z) \\
\mathcal{L} &= \frac{1}{2}(y - t)^2
\end{aligned}
$$

Computing the derivatives:

$$
\begin{aligned}
\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y} &= y - t \\
\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z} &= \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y}\,\sigma'(z) \\
\frac{\partial\mathcal{L}}{\partial w} &= \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}\,x \\
\frac{\partial\mathcal{L}}{\partial b} &= \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}
\end{aligned}
$$

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

A Slightly More Convenient Notation

Use $\overline{y}$ to denote the derivative $\mathrm{d}\mathcal{L}/\mathrm{d}y$, sometimes called the error signal

A Slightly More Convenient Notation

Use $\overline{y}$ to denote the derivative $\mathrm{d}\mathcal{L}/\mathrm{d}y$, 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:

$\frac{dL}{dy}$

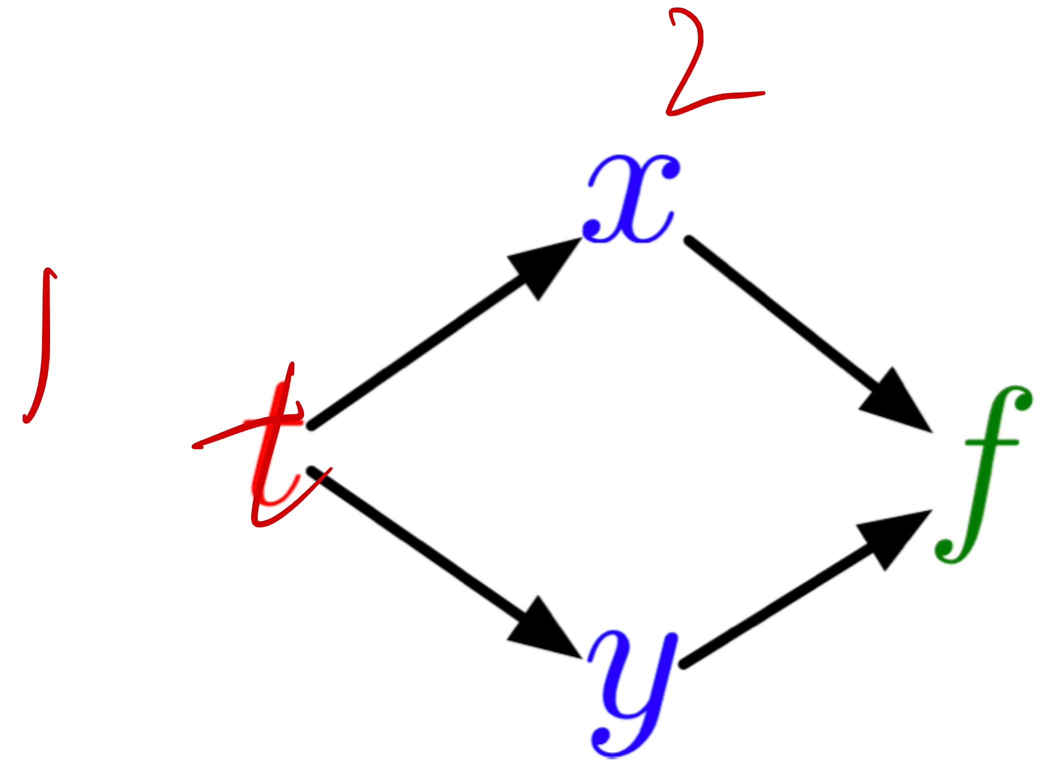
$$\overline{y} = y - t$$
$$\overline{z} = \overline{y}\,\sigma'(z)$$
$$\overline{w} = \overline{z}\,x$$
$$\overline{b} = \overline{z}$$

$\frac{dL}{dz}$

$\frac{dL}{\partial x} = \overline{x}$

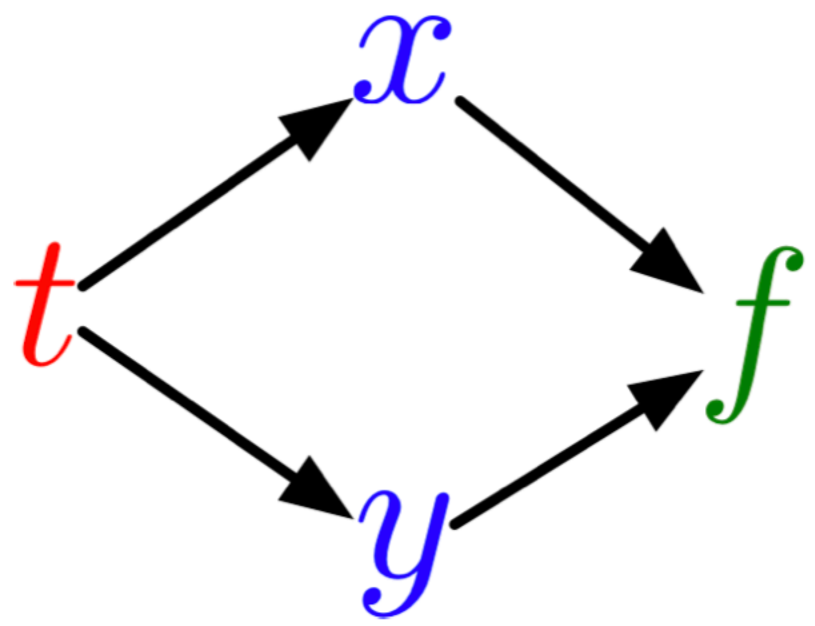
Multivariate Chain Rule

Problem: what if the computation graph has fan-out > 1?
This requires the multivariate Chain Rule!



Multivariate Chain Rule

Problem: what if the computation graph has fan-out > 1?
This requires the multivariate Chain Rule!

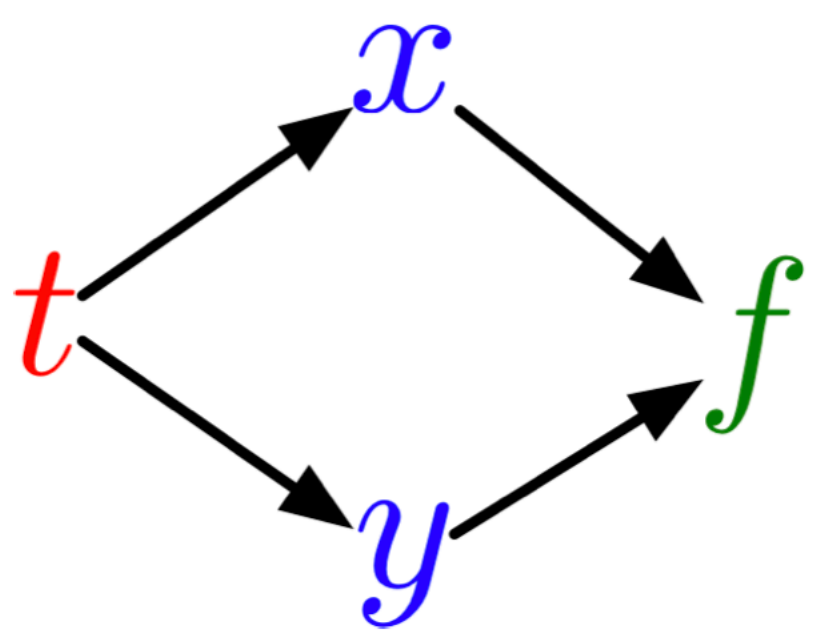


$$\frac{\mathrm{d}}{\mathrm{d}t} f(x(t), y(t)) = \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t}$$

$$\frac{df(x(t), y(t))}{dt}$$

Multivariate Chain Rule

Problem: what if the computation graph has fan-out > 1?
This requires the multivariate Chain Rule!



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

Example:

$$f(x, y) = y + e^{xy}$$
$$x(t) = \cos t$$
$$y(t) = t^2$$

Multivariate Chain Rule

Problem: what if the computation graph has fan-out > 1?
This requires the multivariate Chain Rule!

$$\begin{tikzcd} & x \arrow[dr] & \\ t \arrow[ur] \arrow[dr] & & f \\ & y \arrow[ur] & \end{tikzcd}$$

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

$f(x^{(t)}, y^{(t)}, z^{(t)}, a, b, c)$ $\qquad \frac{df}{dt}$

Example:

$$f(x, y) = y + e^{xy}$$
$$x(t) = \cos t$$
$$y(t) = t^2$$

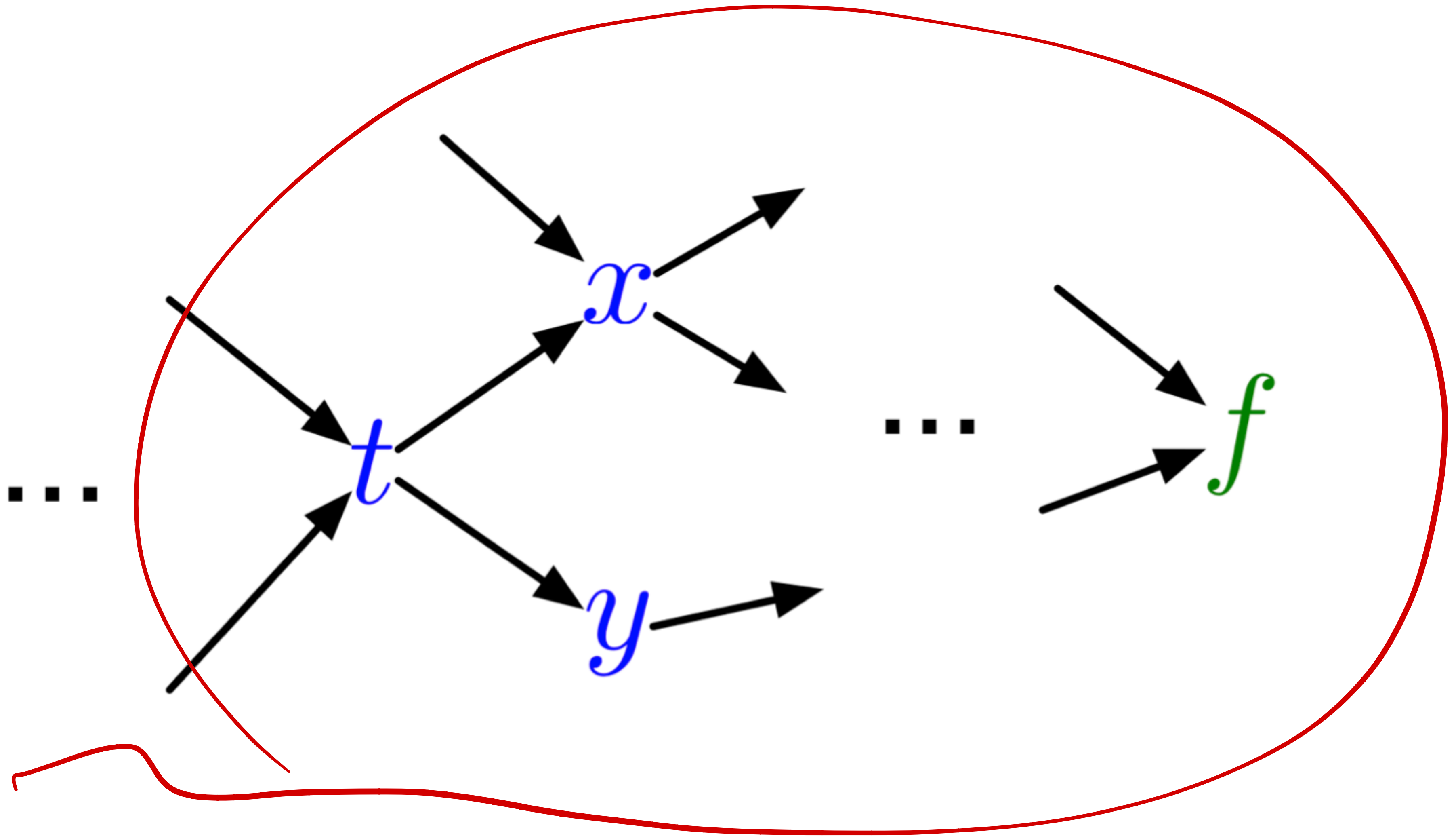
$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t}$$
$$= (ye^{xy}) \cdot (-\sin t) + (1 + xe^{xy}) \cdot 2t$$

Multivariate Chain Rule

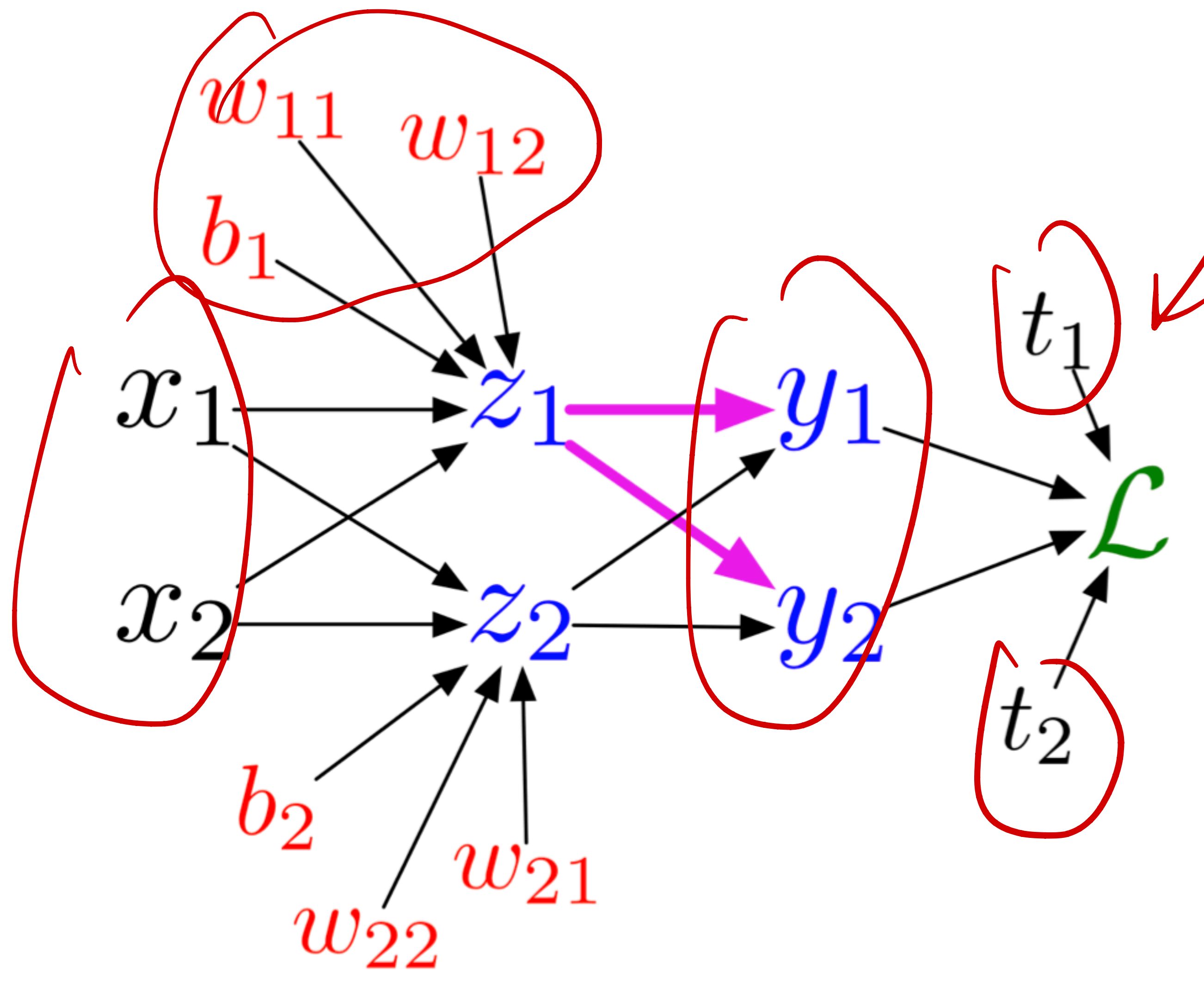
Mathematical expressions
to be evaluated

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t}$$

Values already computed
by our program



Another Example



NN

$$\boxed{\frac{dL}{w_{11}}}$$

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

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

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

cross-entropy

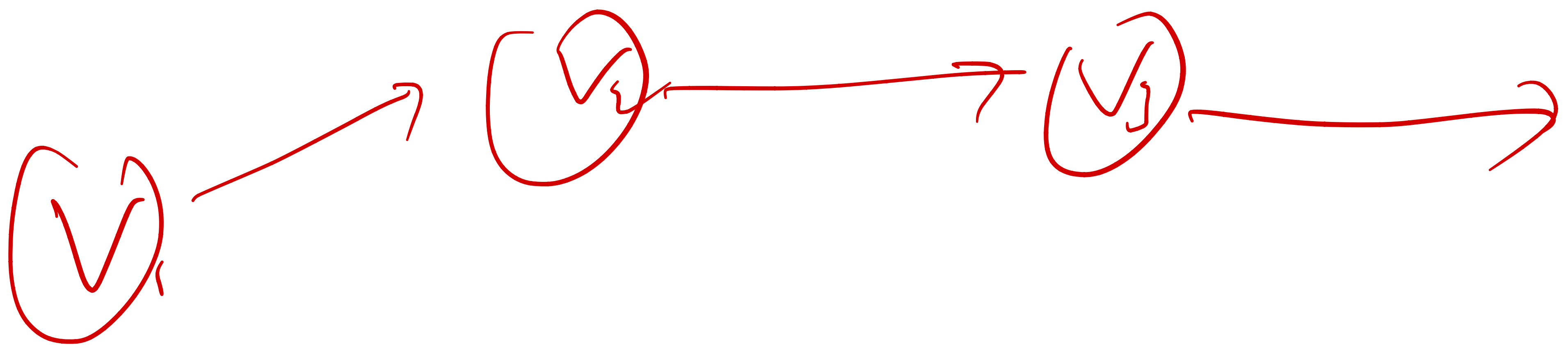
z_1, y_1 y_2

related w_{11}

Backpropagation

Let $v_1, \ldots, v_N$ 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).



$V_N \to loss$

$$\frac{dV_N}{dV}$$

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

Backpropagation

Let $v_1, \ldots, v_N$ 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

$$\text{For } i = 1, \ldots, N$$

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

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

Backpropagation

Let $v_1, \ldots, v_N$ 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 $i = 1, \ldots, N$

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

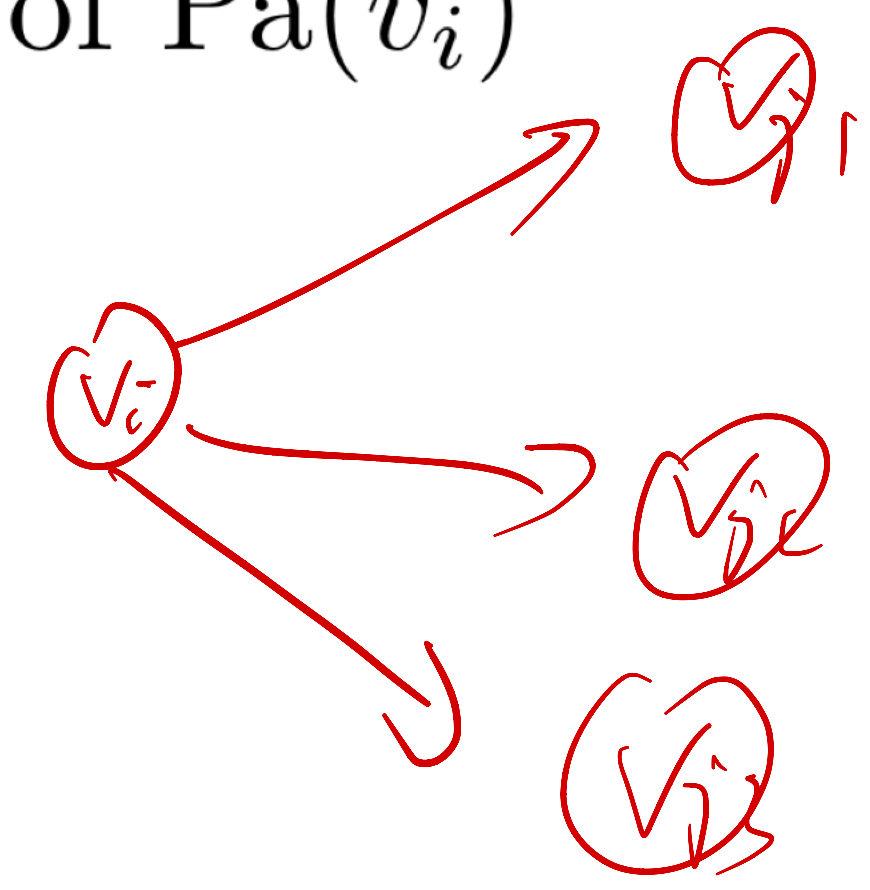
backward pass

$\overline{v_N} = 1$

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

$$\frac{dL}{dv_i} \leftarrow \overline{v_i} = \sum_{j \in \mathrm{Ch}(v_i)} \overline{v_j} \frac{\partial v_j}{\partial v_i}$$

$Ch(v_i)$



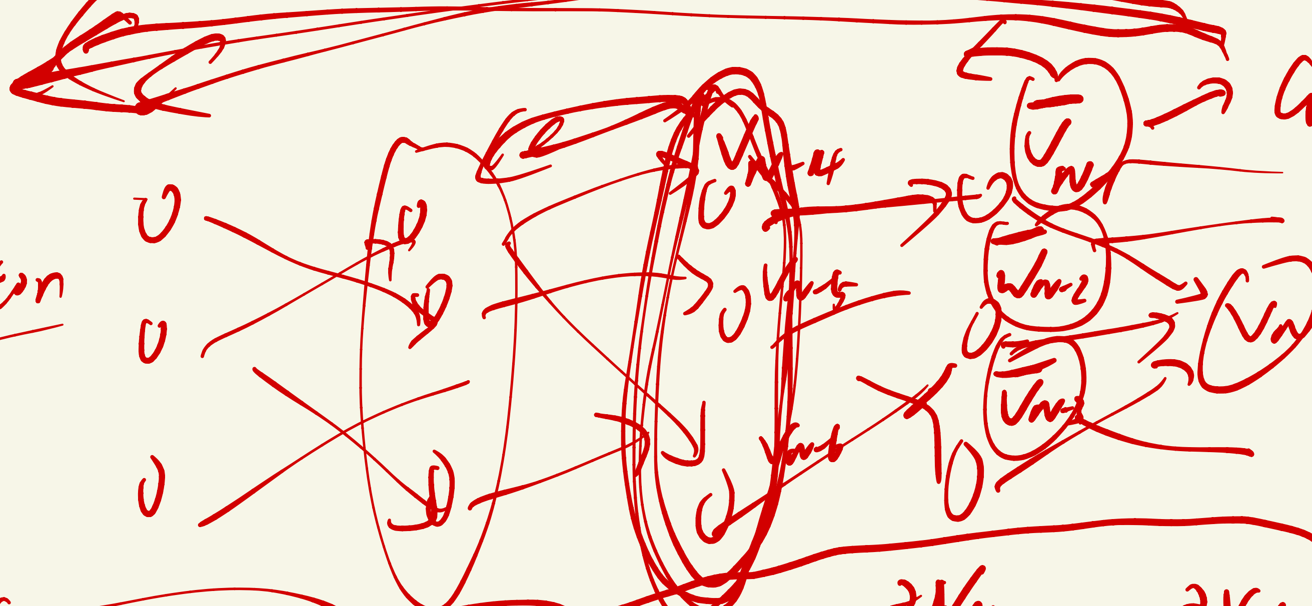
$$\overline{V_i} = \sum_{j \in Ch(v_i)} \overline{V_j} \cdot \frac{\partial V_j}{\partial V_i}$$

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

back propagation

multivariant chain-rule

Hinton



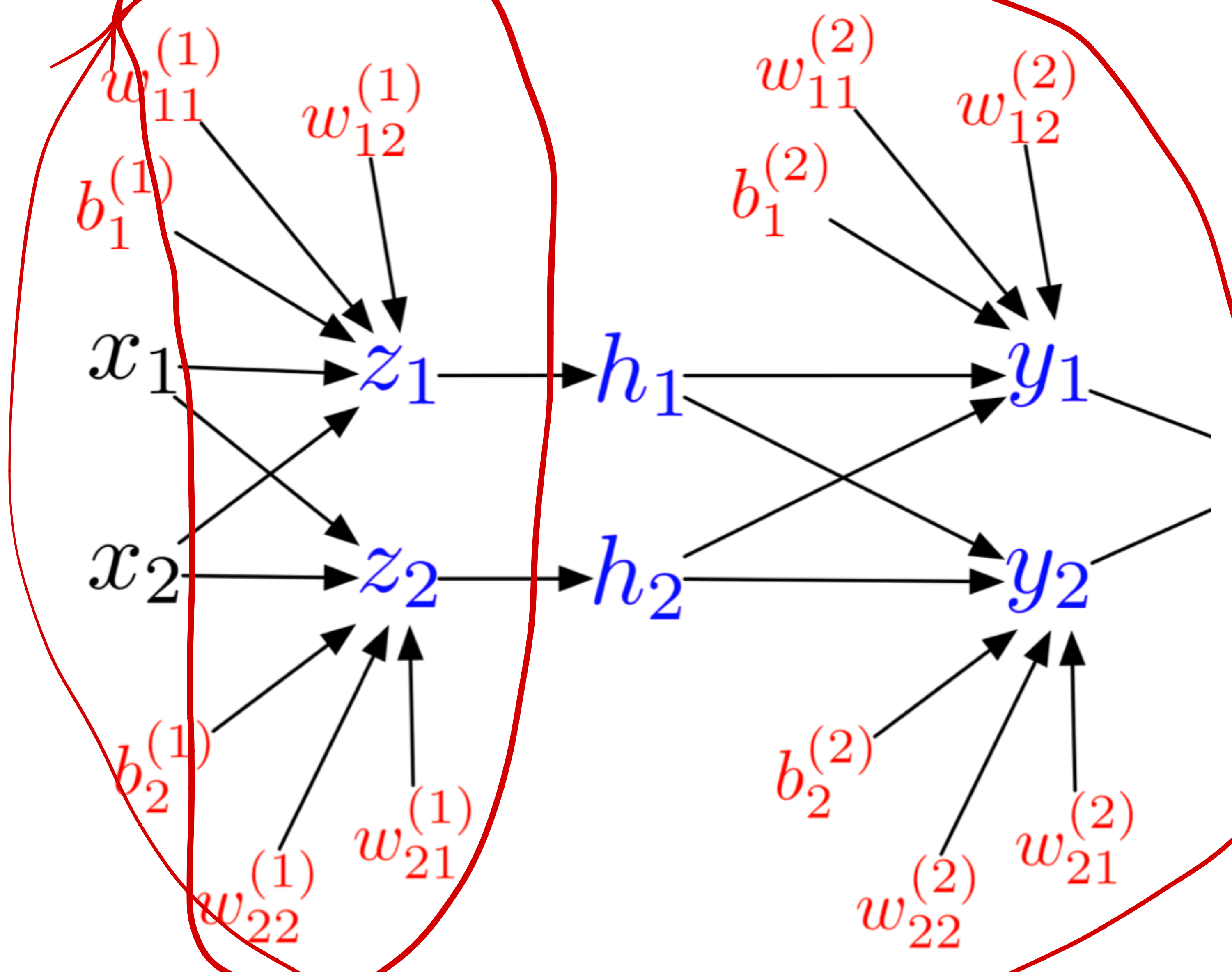
GPU

$$\frac{\partial v_n}{\partial v_{n-1}} \qquad \frac{\partial v_n}{\partial v_{n-2}}$$

$$\bar{v}_{n-4} = \bar{v}_{n-1} \cdot \frac{\partial v_{n-1}}{\partial v_{n-4}} + \bar{v}_{n-2} \cdot \frac{\partial v_{n-2}}{v_{n-4}}$$

Backpropagation

Multilayer Perceptron (multiple outputs):



W

$x \quad z \quad h \quad y$

Forward pass:

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

$$h_i = \sigma(z_i)$$

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

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

backward

Backward pass: $\frac{\partial L}{\partial y_k}$

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

$$\overline{y_k} = \overline{\mathcal{L}}\,(y_k - t_k)$$

$$\overline{w_{ki}^{(2)}} = \overline{y_k}\, h_i$$

$$\overline{b_k^{(2)}} = \overline{y_k}$$

$$\overline{h_i} = \sum_k \overline{y_k}\, w_{ki}^{(2)}$$

$$\overline{z_i} = \overline{h_i}\, \sigma'(z_i)$$

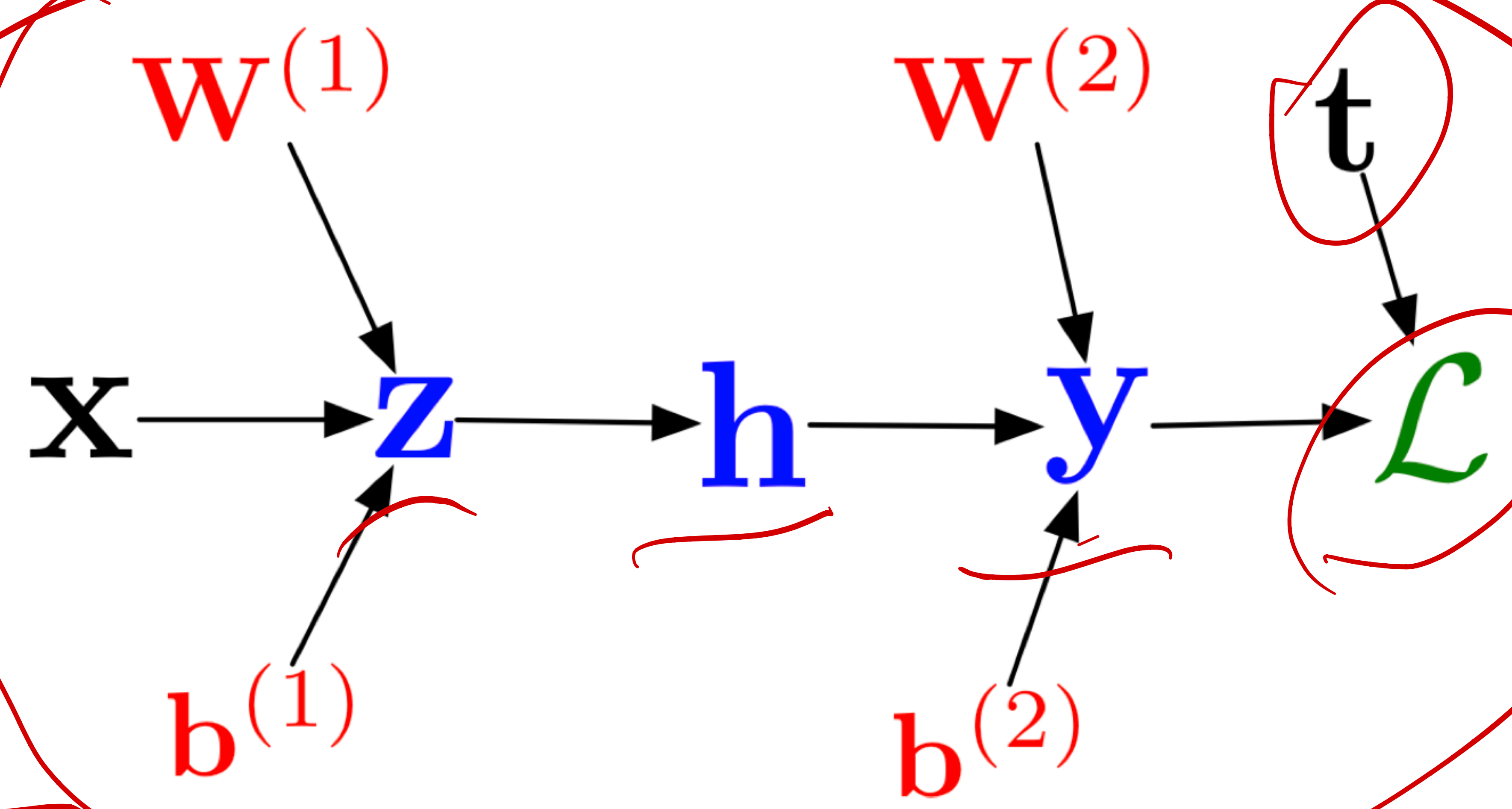
$$\overline{w_{ij}^{(1)}} = \overline{z_i}\, x_j$$

$$\overline{b_i^{(1)}} = \overline{z_i}$$

Backpropagation

W matrix

In vectorized form:



scalar

Forward pass:

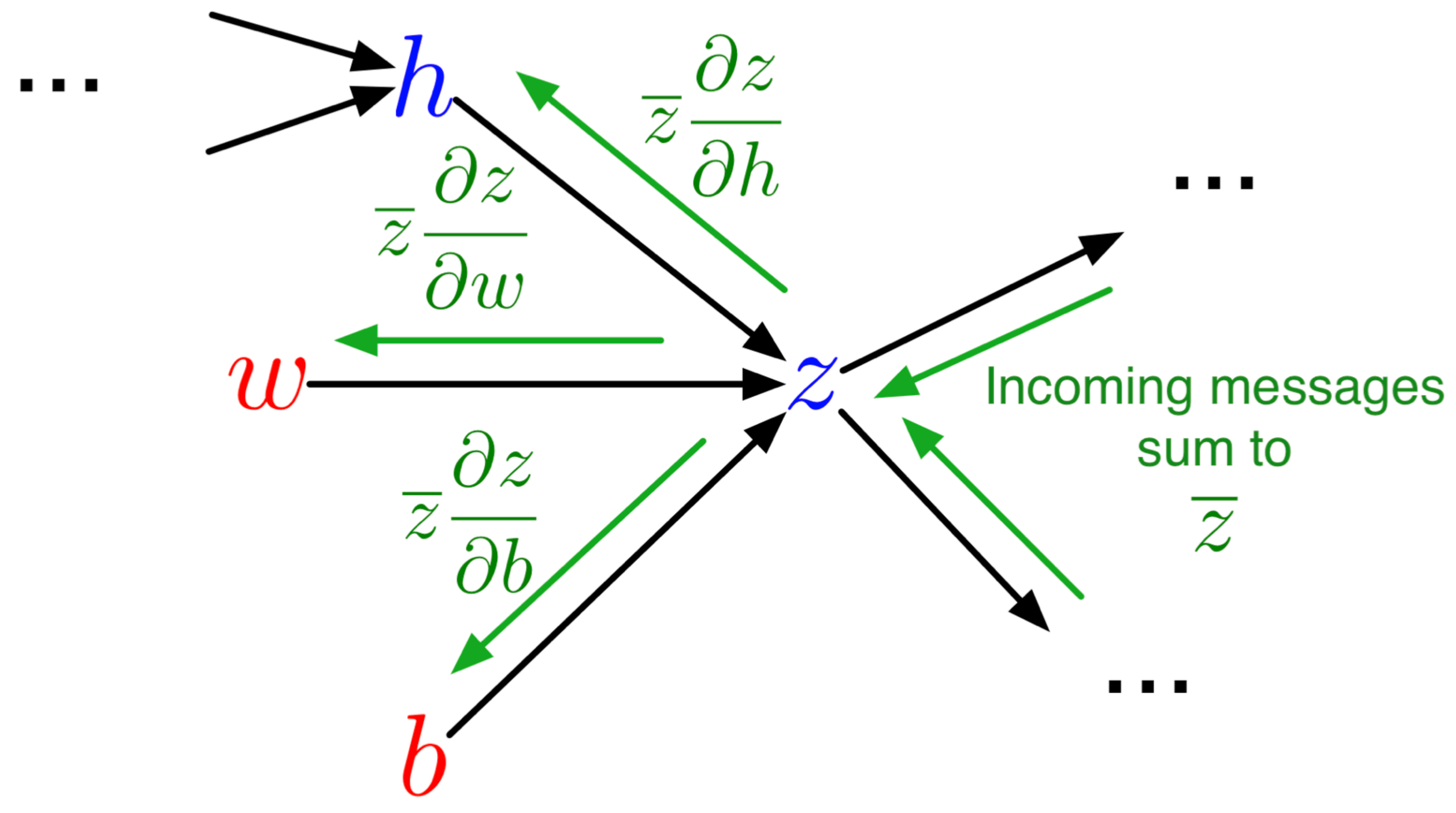
$W^{(i)}$ 2×2

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$
$$\mathbf{h} = \sigma(\mathbf{z})$$
$$\mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$
$$\mathcal{L} = \frac{1}{2}\|\mathbf{t} - \mathbf{y}\|^2$$

Backward pass:

$$\overline{\mathcal{L}} = 1$$
$$\overline{\mathbf{y}} = \overline{\mathcal{L}}\,(\mathbf{y} - \mathbf{t})$$
$$\overline{\mathbf{W}^{(2)}} = \overline{\mathbf{y}}\mathbf{h}^\top$$
$$\overline{\mathbf{b}^{(2)}} = \overline{\mathbf{y}}$$
$$\overline{\mathbf{h}} = \mathbf{W}^{(2)\top}\overline{\mathbf{y}}$$
$$\overline{\mathbf{z}} = \overline{\mathbf{h}} \circ \sigma'(\mathbf{z})$$
$$\overline{\mathbf{W}^{(1)}} = \overline{\mathbf{z}}\mathbf{x}^\top$$
$$\overline{\mathbf{b}^{(1)}} = \overline{\mathbf{z}}$$

Backpropagation as Message Passing

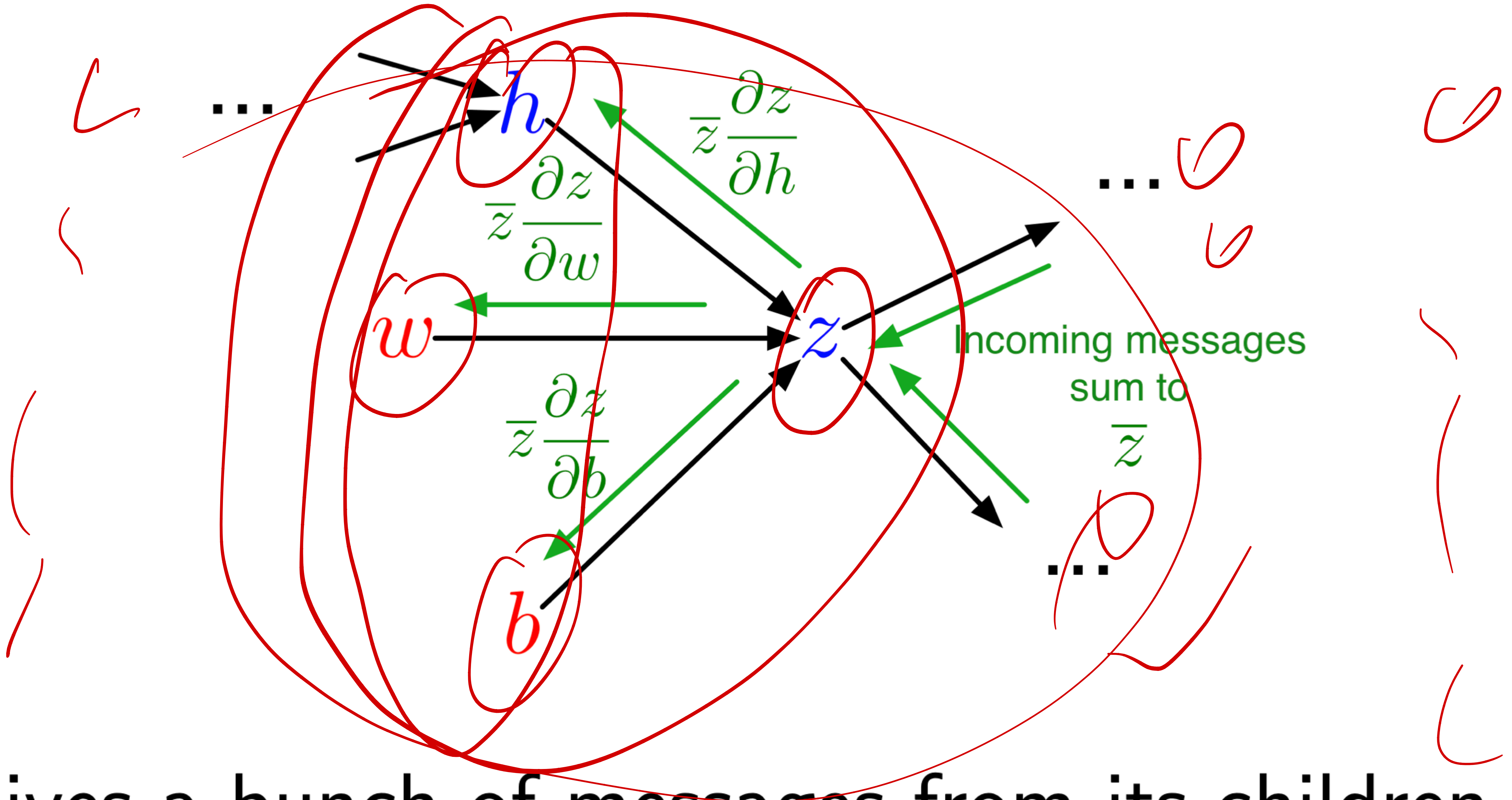


signal neurons

- Each node receives a bunch of messages from its children, which it aggregates to get its error signal. It then passes messages to its parents.

fam elimination for
belief propagation

Backpropagation as Message Passing



- Each node receives a bunch of messages from its children, which it aggregates to get its error signal. It then passes messages to its parents.

Each node only has to know how to compute derivatives with respect to its arguments, and doesn't have to know anything about the rest of the graph

Computational Cost

Computational Cost

- Computational cost of forward pass: one add-multiply operation per weight

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

Computational Cost

- Computational cost of forward pass: one add-multiply operation per weight

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

- Computational cost of backward pass: two add-multiply operations per weight

$$\overline{w_{ki}^{(2)}} = \overline{y_k}\, h_i$$

$$\overline{h_i} = \sum_k \overline{y_k}\, w_{ki}^{(2)}$$

gradient for both
hidden h
parameter w

learning parameter

Computational Cost

- Computational cost of forward pass: one add-multiply operation per weight

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

- Computational cost of backward pass: two add-multiply operations per weight

$$\overline{w_{ki}^{(2)}} = \overline{y_k}\, h_i$$
$$\overline{h_i} = \sum_k \overline{y_k} w_{ki}^{(2)}$$

The backward pass is about as expensive as two forward passes

Computational Cost

- Computational cost of forward pass: one add-multiply operation per weight

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

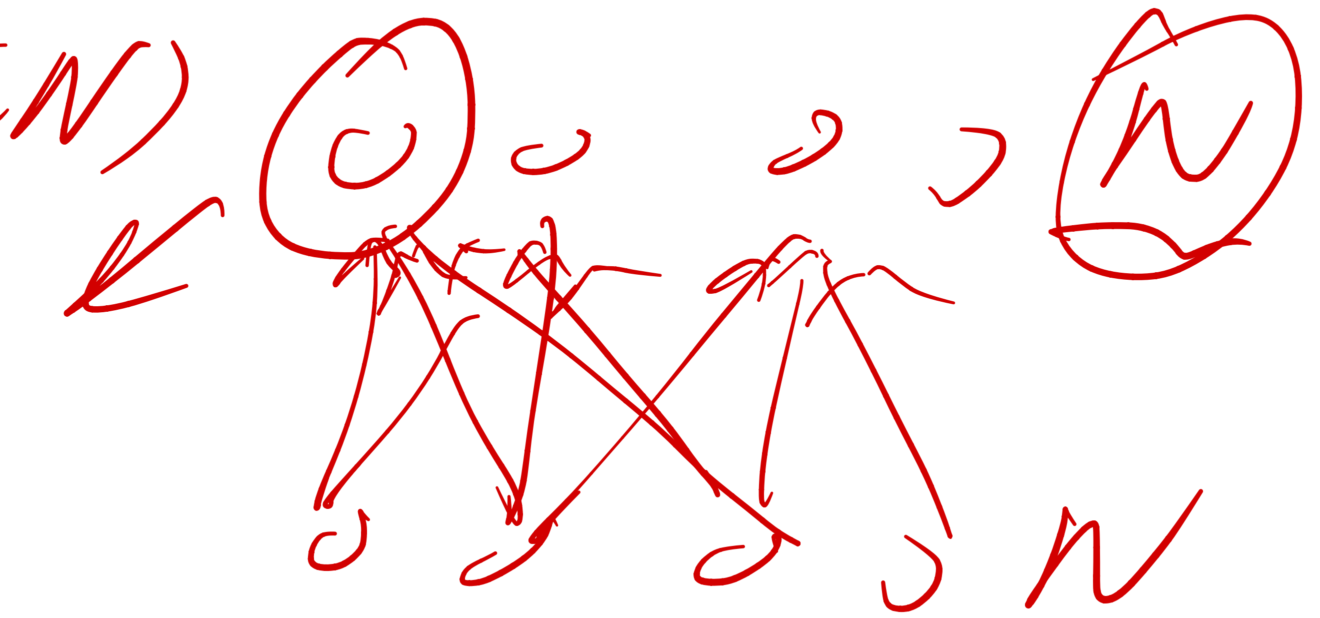
$N \cdot N$

- Computational cost of backward pass: two add-multiply operations per weight

$$\overline{w_{ki}^{(2)}} = \overline{y_k}\, h_i$$

$$\overline{h_i} = \sum_k \overline{y_k}\, w_{ki}^{(2)}$$

$O(N)$



The backward pass is about as expensive as two forward passes

For a multilayer perceptron, this means the cost is linear in the number of layers, quadratic in the number of units per layer

Backpropagation

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.

adam

gradient-based metho

Backpropagation

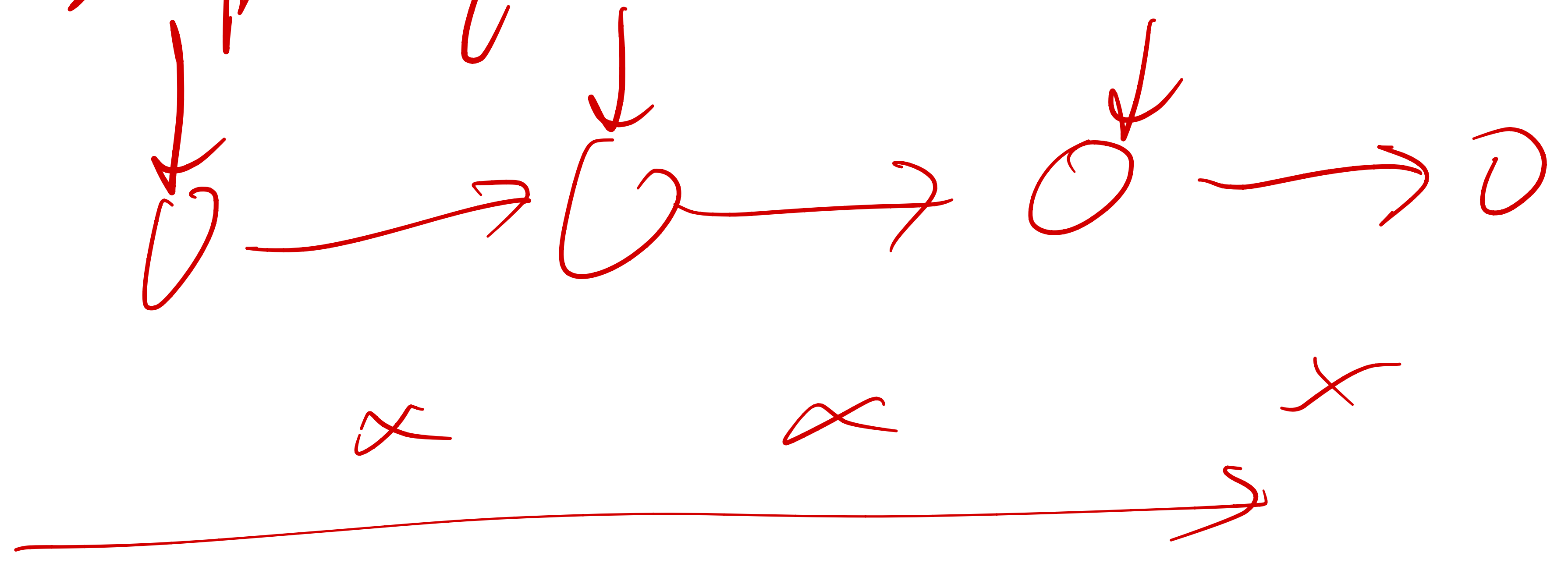
- Backprop is used to train the overwhelming majority of neural nets today.
  - Even optimization algorithms much fancier than gradient descent (e.g. second-order methods) use backprop to compute the gradients.

- 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?

Backpropagation

- By now, we've seen three different ways of looking at gradients:
  - **Geometric:** visualization of gradient in weight space
  - **Algebraic:** mechanics of computing the derivatives
  - **Implementational:** efficient implementation on the computer

Simple equation



Stochastic Gradient Descent

Stochastic Gradient Descent SGD

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

$$\frac{\sum_{i}^{N} \frac{\partial L}{\partial w}}{N}$$ for all data

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:

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:

$$L = \mathbb{E}_{x \sim p_{data}} l(x) \approx \frac{1}{N} \sum_{i=1}^{N} l(x_i)$$

Monte Carlo

N can be small

$$E_{x \sim p_{data}} l(x) = \frac{1}{n} \sum_{i=1}^{n} l(x_i)$$

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:

$$L = \mathbb{E}_{x \sim p_{data}} l(x) \approx \frac{1}{N} \sum_{i=1}^{N} l(x_i)$$

N is the size of the entire training dataset

Monte Carlo

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:

$$L = \mathbb{E}_{x \sim p_{data}} l(x) \approx \frac{1}{N} \sum_{i=1}^{N} l(x_i)$$

N is the size of the entire training dataset

This is slow on the entire training dataset, thus we use MCMC to approximate:

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:

$$L = \mathbb{E}_{x \sim p_{data}} l(x) \approx \frac{1}{N} \sum_{i=1}^{N} l(x_i)$$

N is the size of the entire training dataset

This is slow on the entire training dataset, thus we use MCMC to approximate:

$$\nabla L = \nabla \mathbb{E}_{x \sim p_{data}} l(x) \approx \nabla \frac{1}{n} \sum_{i=1}^{n} l(x_i)$$

n is the size of a random minibatch (batch size)

batch size

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:

$$L = \mathbb{E}_{x \sim p_{data}} l(x) \approx \frac{1}{N} \sum_{i=1}^{N} l(x_i)$$

N is the size of the entire training dataset

This is slow on the entire training dataset, thus we use MCMC to approximate:

$$\nabla L = \nabla \mathbb{E}_{x \sim p_{data}} l(x) \approx \nabla \frac{1}{n} \sum_{i=1}^{n} l(x_i)$$

n is the size of a random minibatch (batch size)

n can be as small as one

data-centric approach

GANs

min max

fixed

Background

A Recipe for Machine Learning

collect

synthetic

data

1. Given training data:

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

2. Choose each of these:

– Decision function

LSTM

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

transfer

CNN

– Loss function

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

model-centric

research

3. Define goal:

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

4. Train with SGD:

(take small steps opposite the gradient)

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

loss function

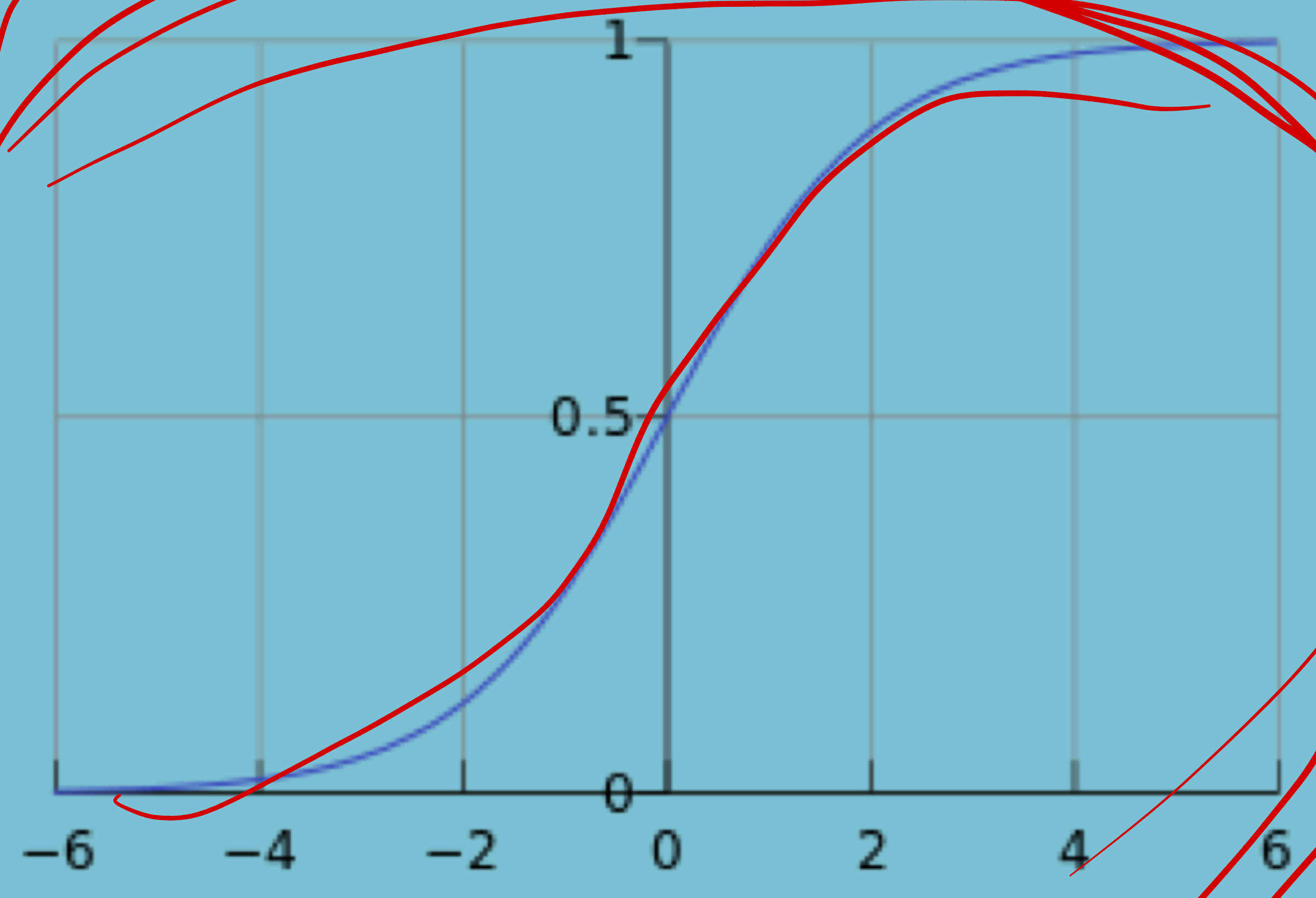
adam

adagrad

Activation Functions

Sigmoid / Logistic Function

$$\text{logistic}(u) \equiv \frac{1}{1+e^{-u}}$$

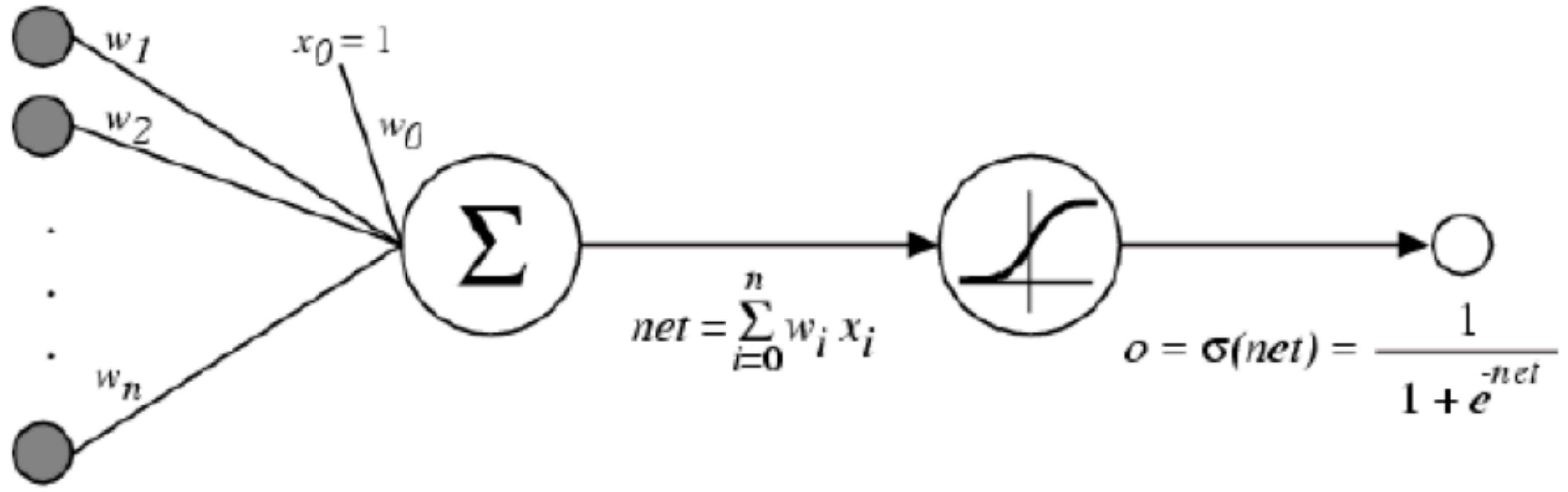


[0, 1]

So far, we've assumed that the activation function (nonlinearity) is always the sigmoid function…

$\frac{1}{1+e^{-u}}$

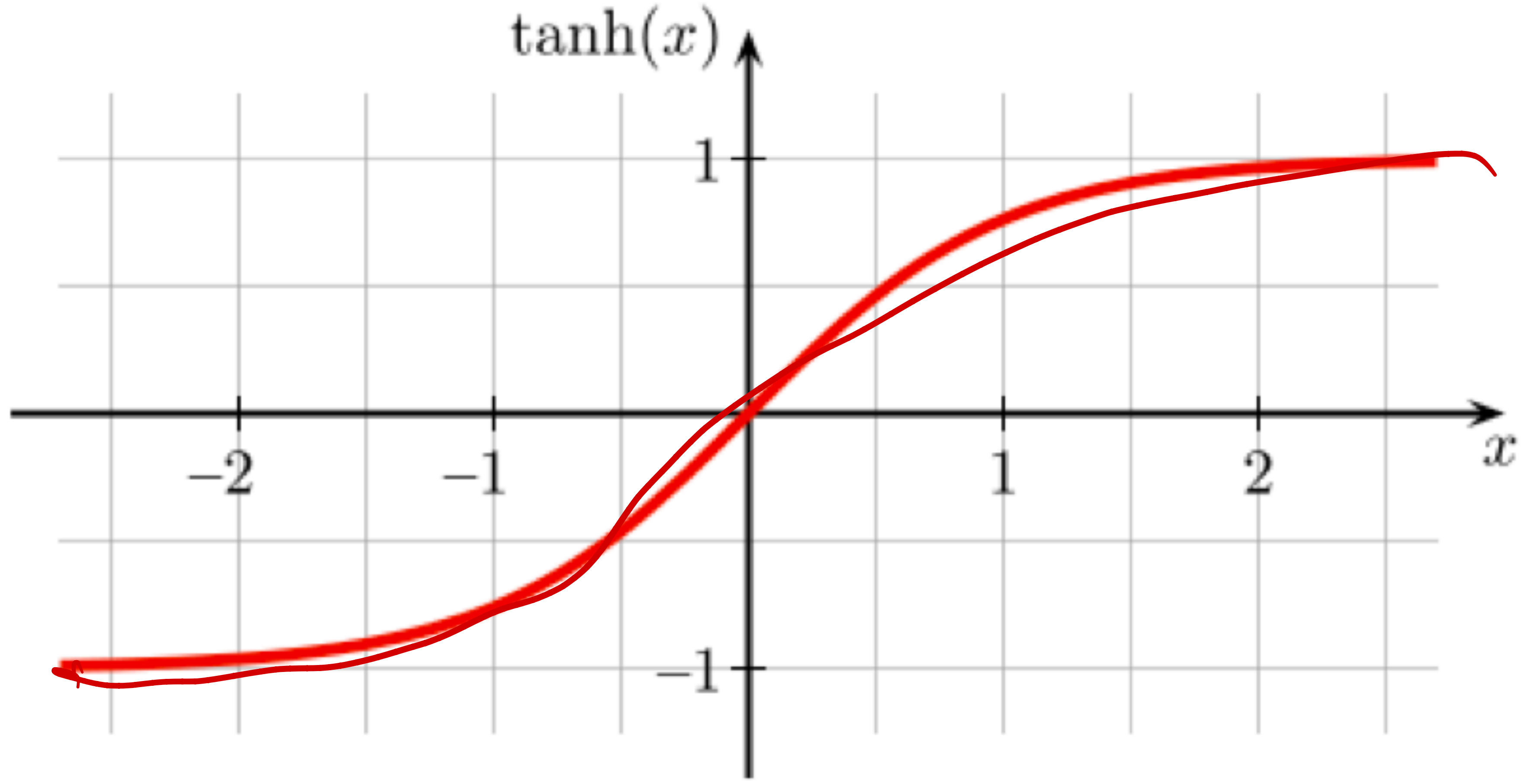
logistic function



Tanh

- A new change: modifying the nonlinearity
  - The logistic is not widely used in modern ANNs

tanh



Alternate 1:
tanh

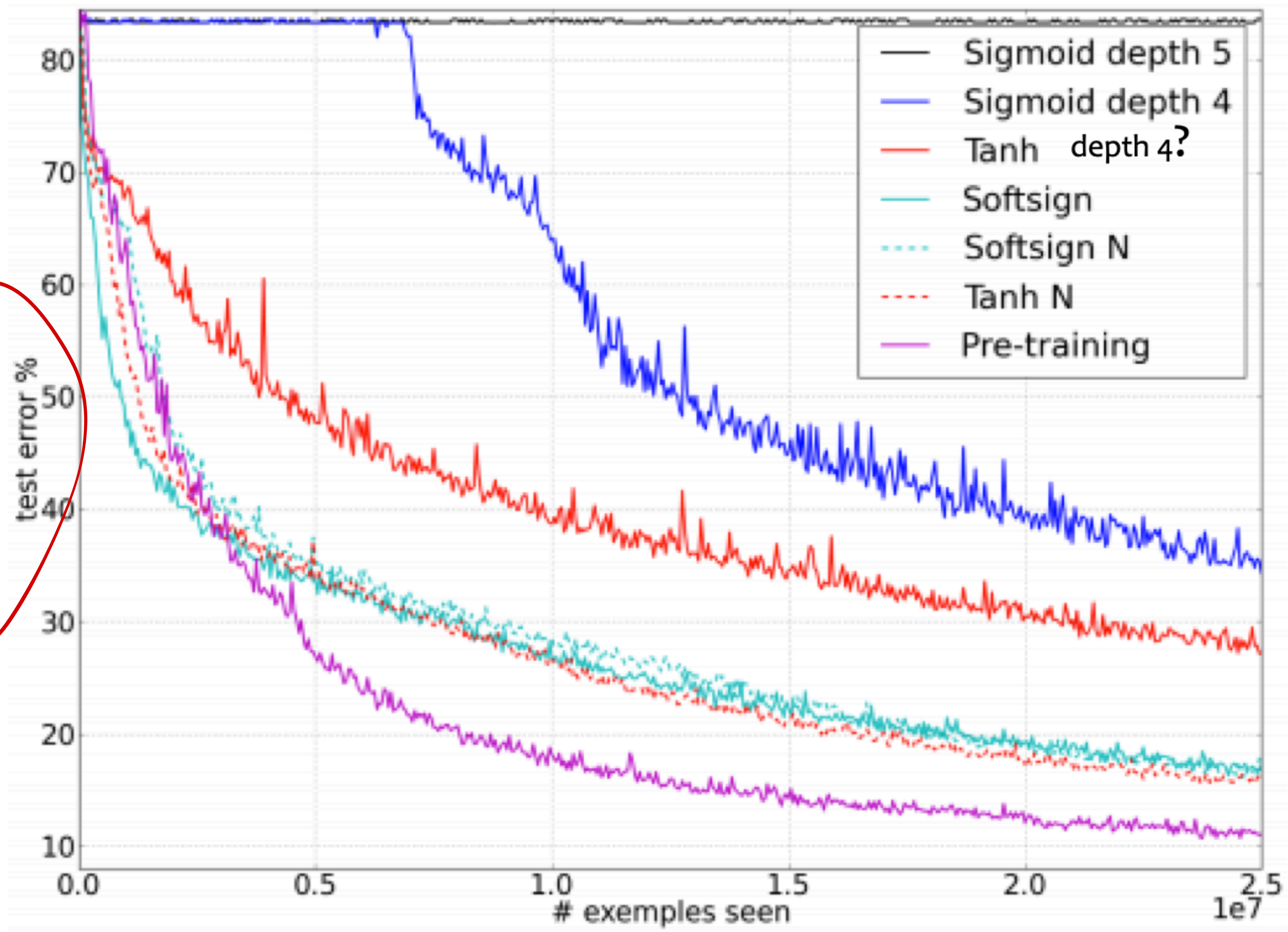
Like logistic function but shifted to range [-1, +1]

tanh

Activation Function

Understanding the difficulty of training deep feedforward neural networks

AI Stats 2010

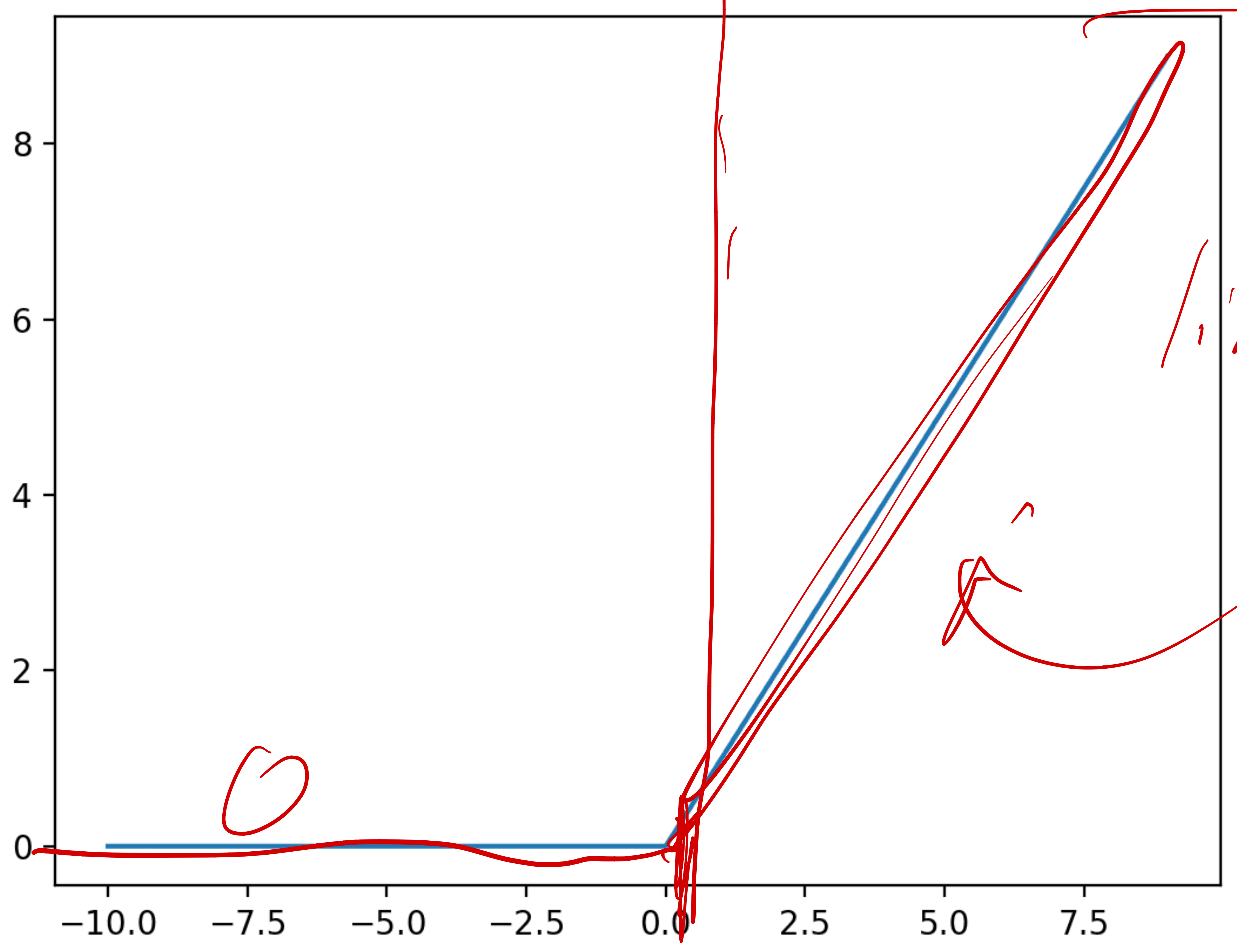


sigmoid
vs.
tanh

Figure from Glorot & Bentio (2010)

ReLU

$f(x) = max(0, x)$



Other Activation Functions

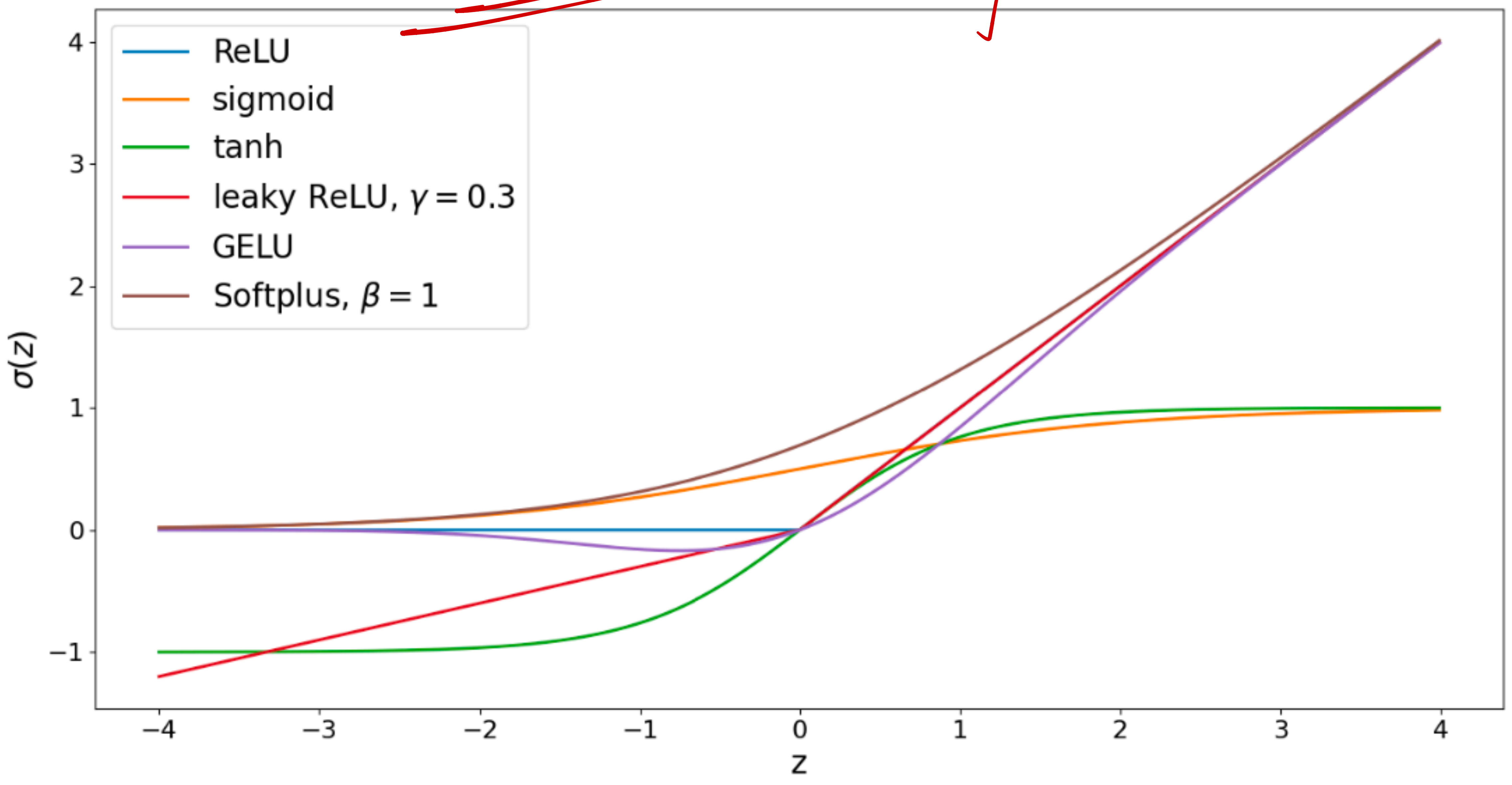
$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad \text{(sigmoid)}$$

$$\sigma(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}} \qquad \text{(tanh)}$$

$$\sigma(z) = \max\{z, \gamma z\}, \gamma \in (0,1) \qquad \text{(leaky ReLU)}$$

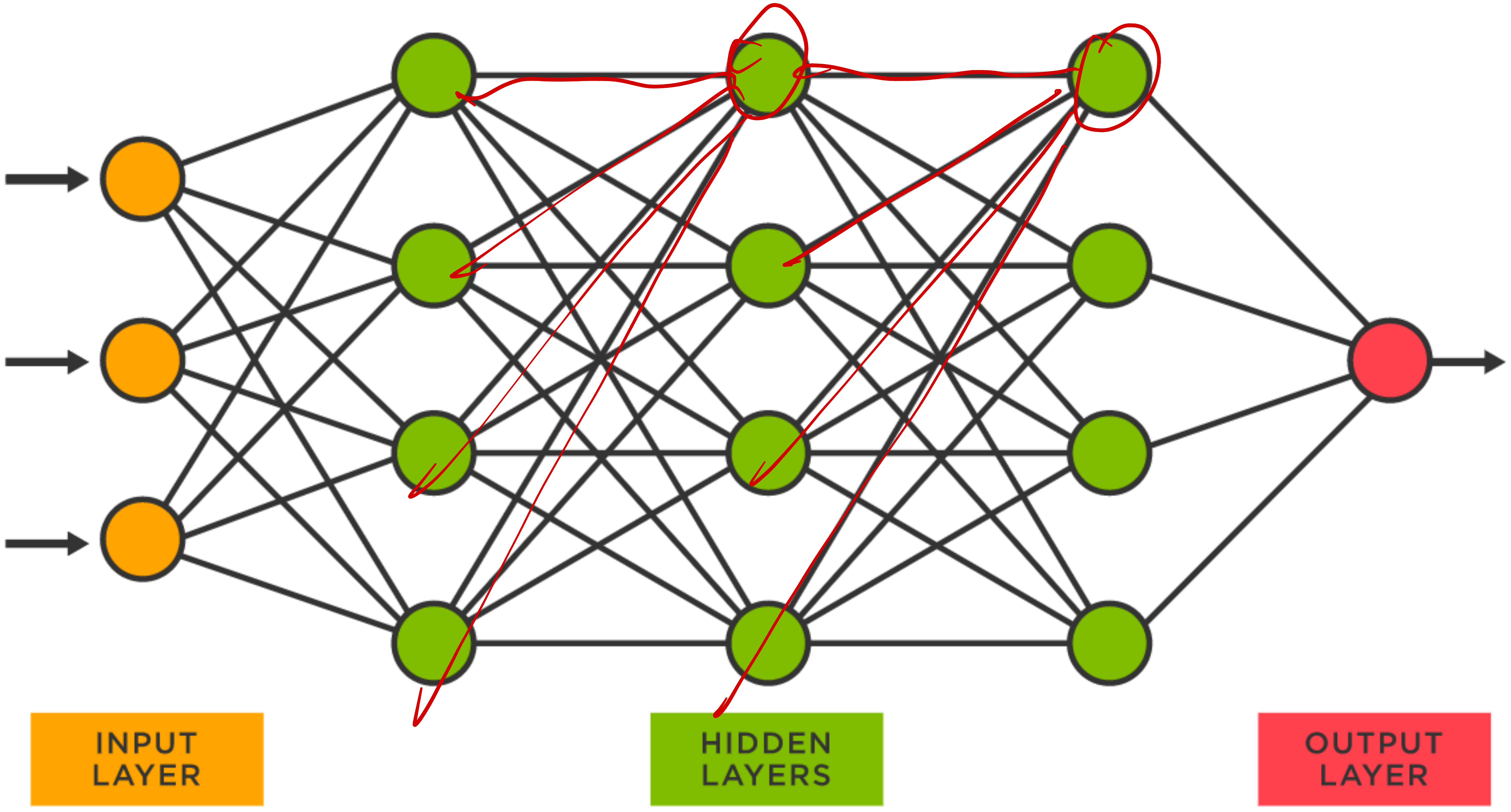
$$\sigma(z) = \frac{z}{2}\left[1+\operatorname{erf}\left(\frac{z}{\sqrt{2}}\right)\right] \qquad \text{(GELU)}$$

$$\sigma(z) = \frac{1}{\beta}\log(1+\exp(\beta z)), \beta > 0 \qquad \text{(Softplus)}$$



nonlinear function

Multilayer Perceptron Neural Networks (MLP)



Thank You!