Numerical Analysis for Artificial Intelligence, Week 4

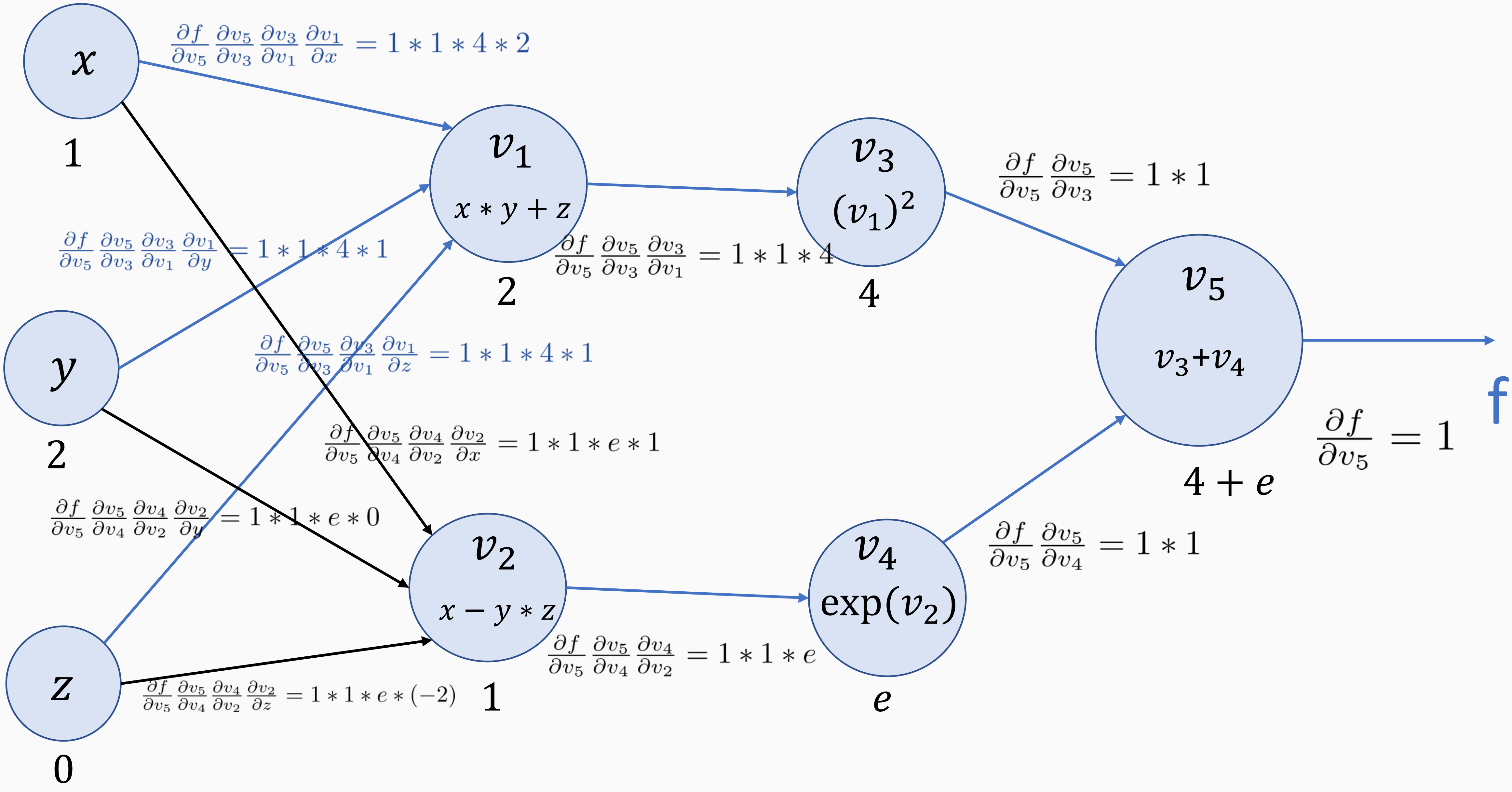
UCSD Summer session II 2018

CSE 190

Jacek (*Yatzek)* Cyranka

Backprop of f(x,y)=$(xy+z)^2$+$\exp(x-yz)$

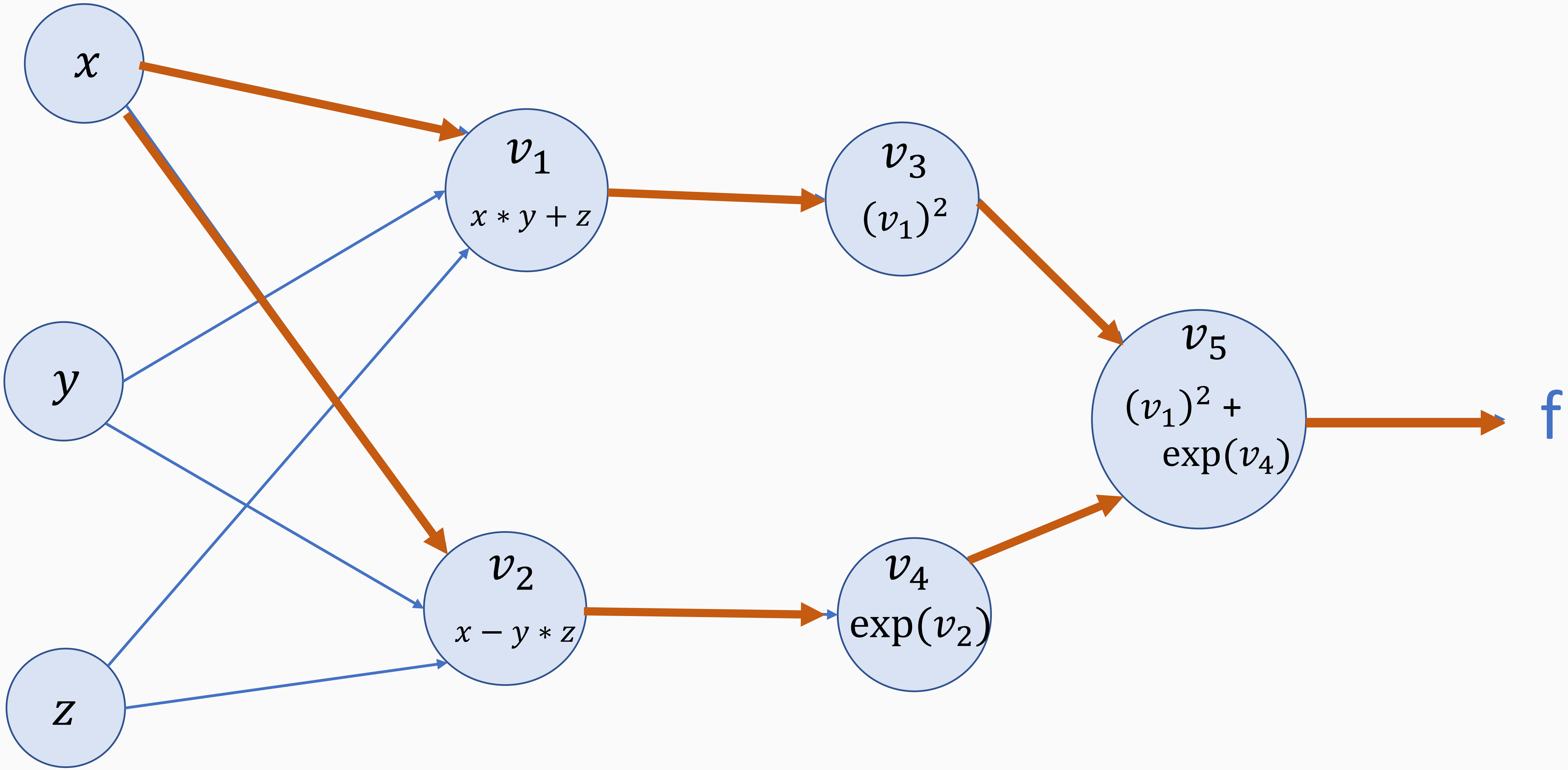
A more complicated graph (function)



Interpretation of backprop as sum of paths

$$\frac{\partial f}{\partial x} = \sum_{\substack{\text{path in the graph} \\ \text{from } x \text{ to } f \\ (v_{i_1}, v_{i_2}, \ldots, v_{i_k})}} \frac{\partial f}{\partial v_{i_k}} \frac{\partial v_{i_k}}{\partial v_{i_{k-1}}} \cdots \frac{\partial v_{i_2}}{\partial v_{i_1}}$$

Partial derivatives – paths interpretation



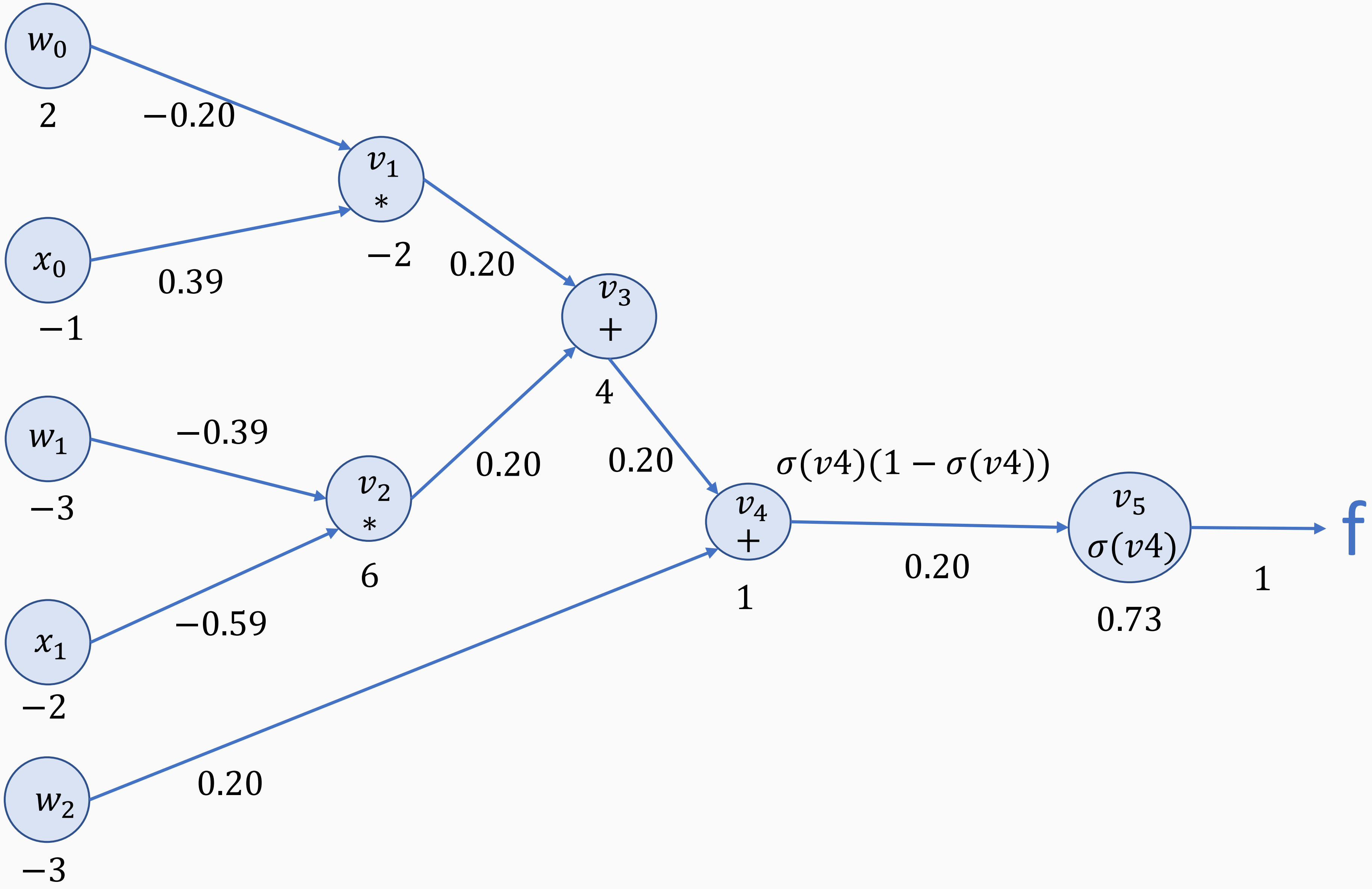
Hence the total partial derivatives are

We combine the results obtained by ‘traversing’ all paths in the graph, and the final result for f(x,y)=$(xy+z)^2$+exp$(x-yz)$ is

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial v_5}\frac{\partial v_5}{\partial v_3}\frac{\partial v_3}{\partial v_1}\frac{\partial v_1}{\partial x} + \frac{\partial f}{\partial v_5}\frac{\partial v_5}{\partial v_4}\frac{\partial v_4}{\partial v_2}\frac{\partial v_2}{\partial x} = 1*1*4*1+1*1*e*1,$$
$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial v_5}\frac{\partial v_5}{\partial v_3}\frac{\partial v_3}{\partial v_1}\frac{\partial v_1}{\partial y} + \frac{\partial f}{\partial v_5}\frac{\partial v_5}{\partial v_4}\frac{\partial v_4}{\partial v_2}\frac{\partial v_2}{\partial y} = 0+1*1*4*2,$$
$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial v_5}\frac{\partial v_5}{\partial v_4}\frac{\partial v_3}{\partial v_1}\frac{\partial v_1}{\partial z} + \frac{\partial f}{\partial v_5}\frac{\partial v_5}{\partial v_4}\frac{\partial v_4}{\partial v_2}\frac{\partial v_2}{\partial z} = 1*1*4*1+1*1*e*(-2).$$

Multivariate sigmoid

Backprop of $f = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



Backpropagation using vectorization

If nodes have vector values , use the generalized chain rule

$$\frac{\partial z}{\partial x_k} = \sum_{j=1}^{n} \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_k},$$

$$\nabla_x z = \left(\frac{\partial y}{\partial x}\right)^T \nabla_y z.$$ ← Gradient (column vec.)

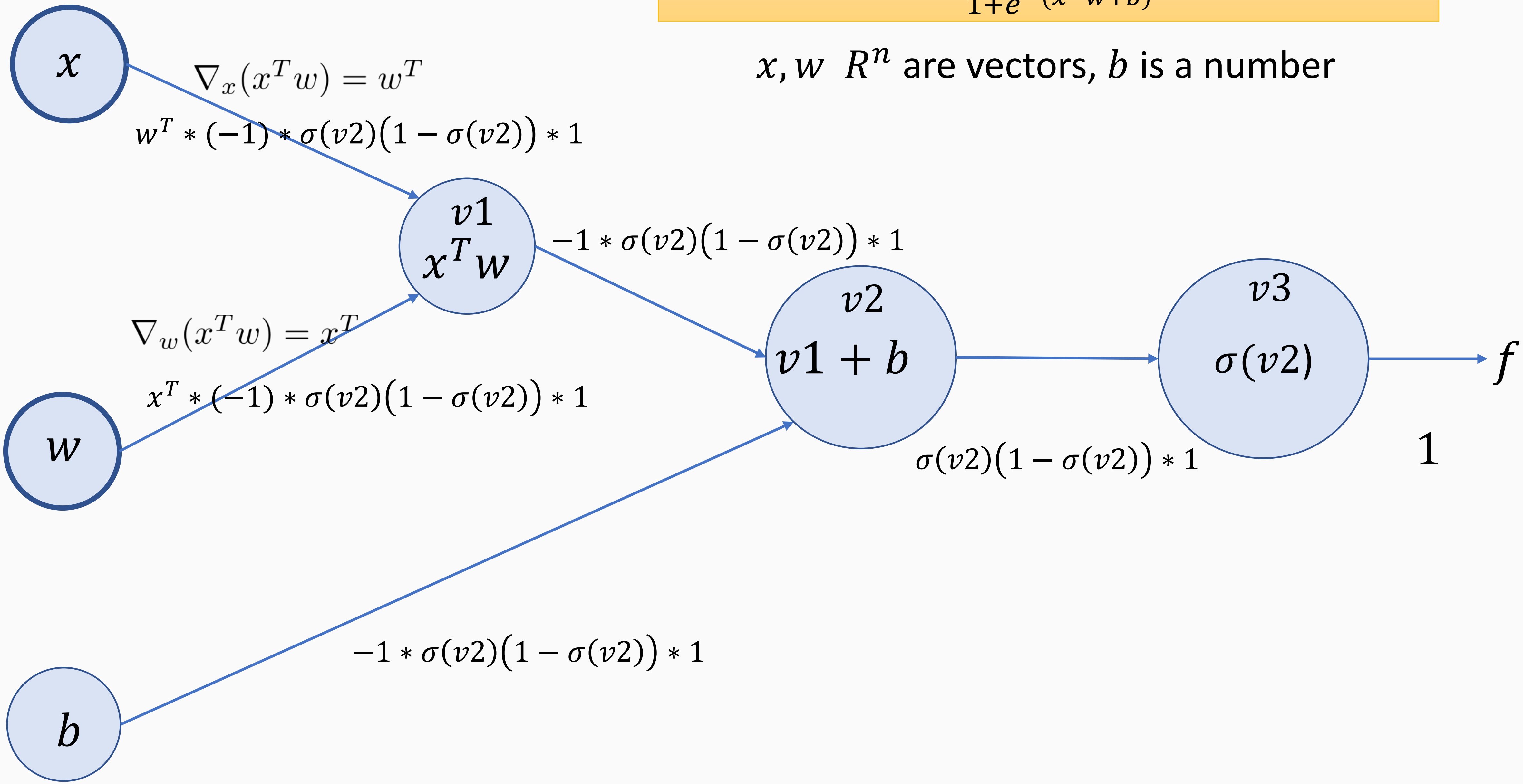
z is a scalar, $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, then

$$\frac{\partial y}{\partial x} \in \mathbb{R}^{n \times m}$$ ← Jacobian matrix

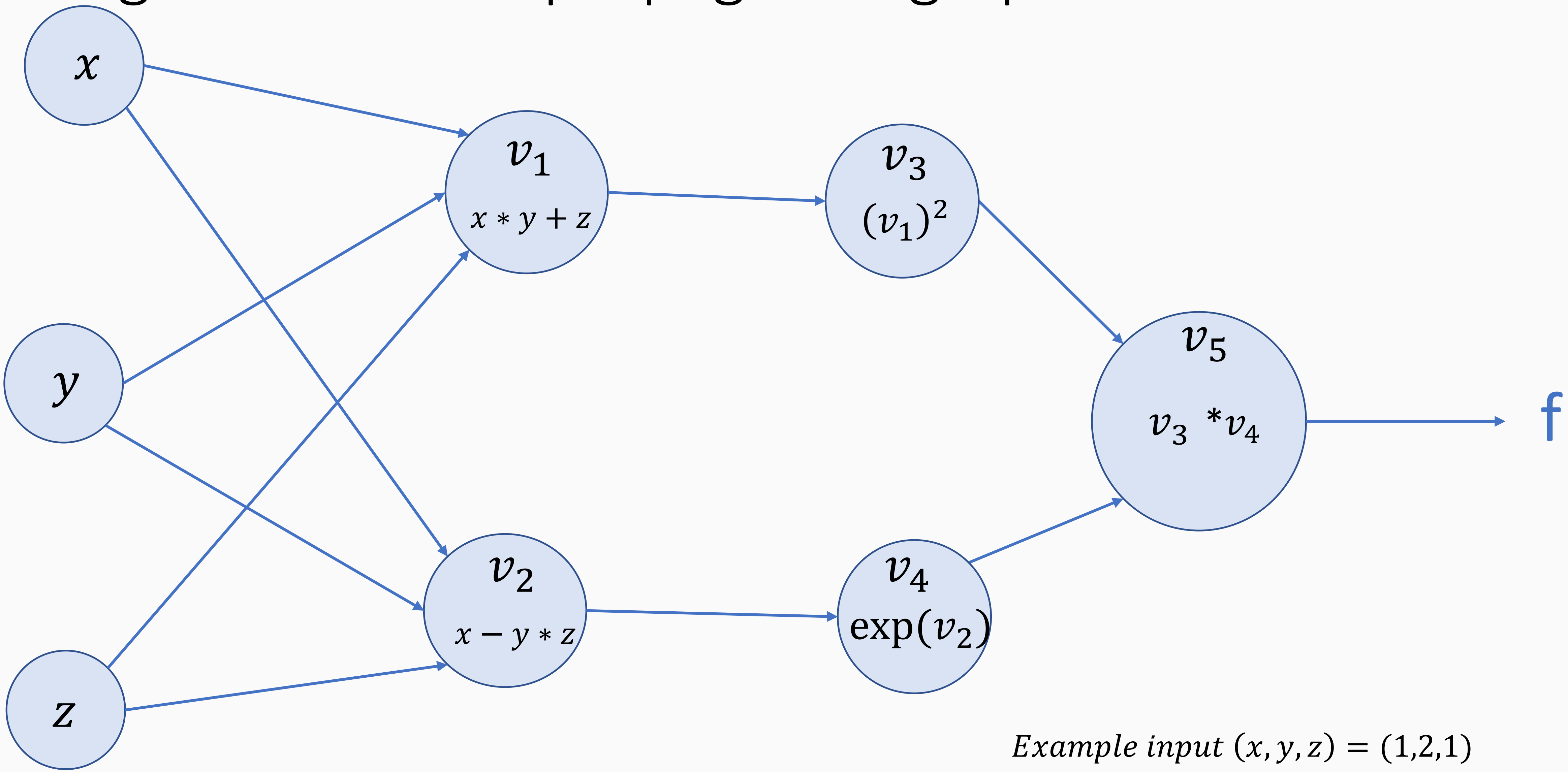
Vectorized backprop case study

Backprop of $f = \frac{1}{1+e^{-(x^T w+b)}} = \sigma(x^T w + b)$

$x, w \; R^n$ are vectors, b is a number

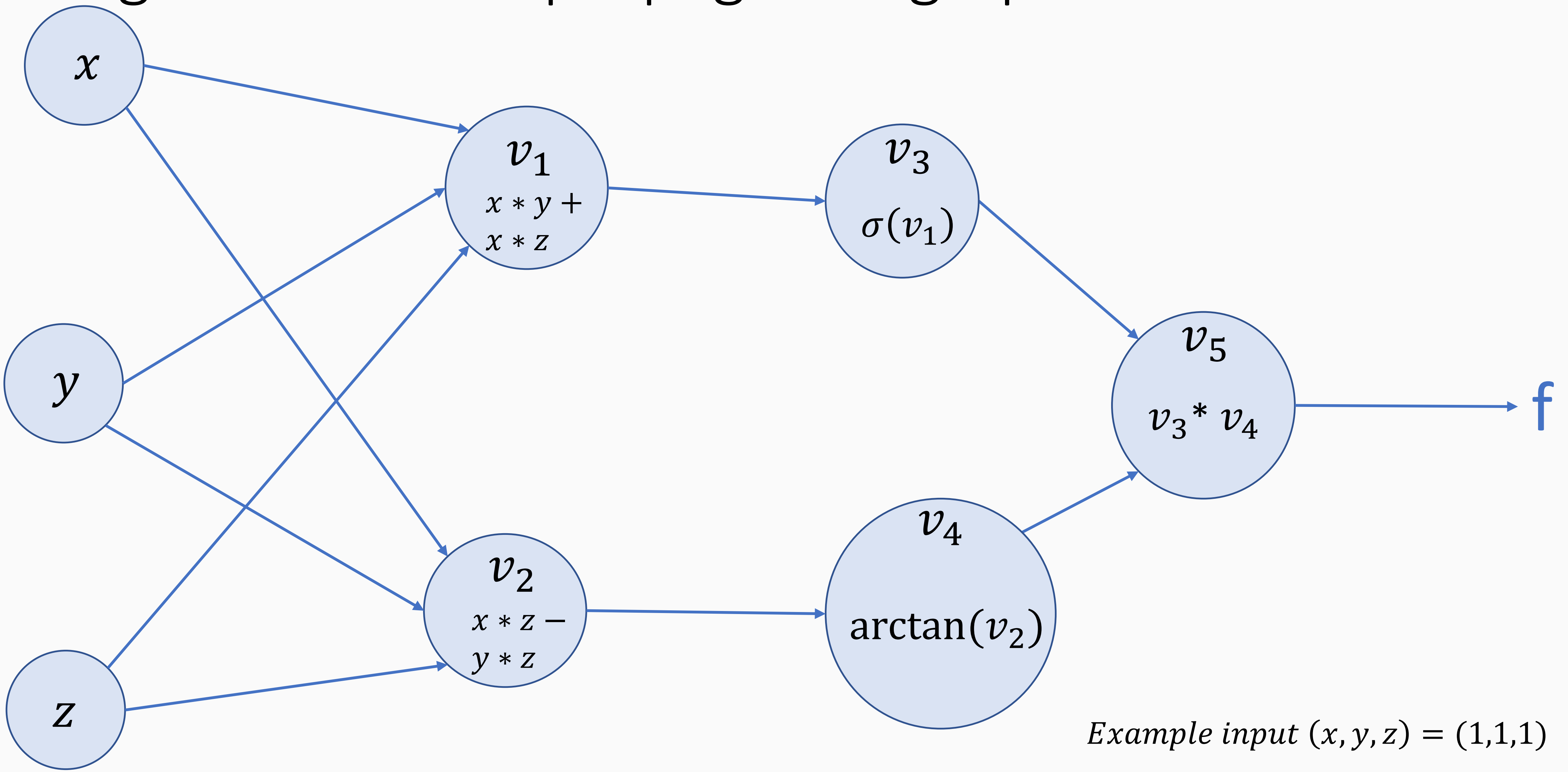


Assignment 6 backpropagation graph 1



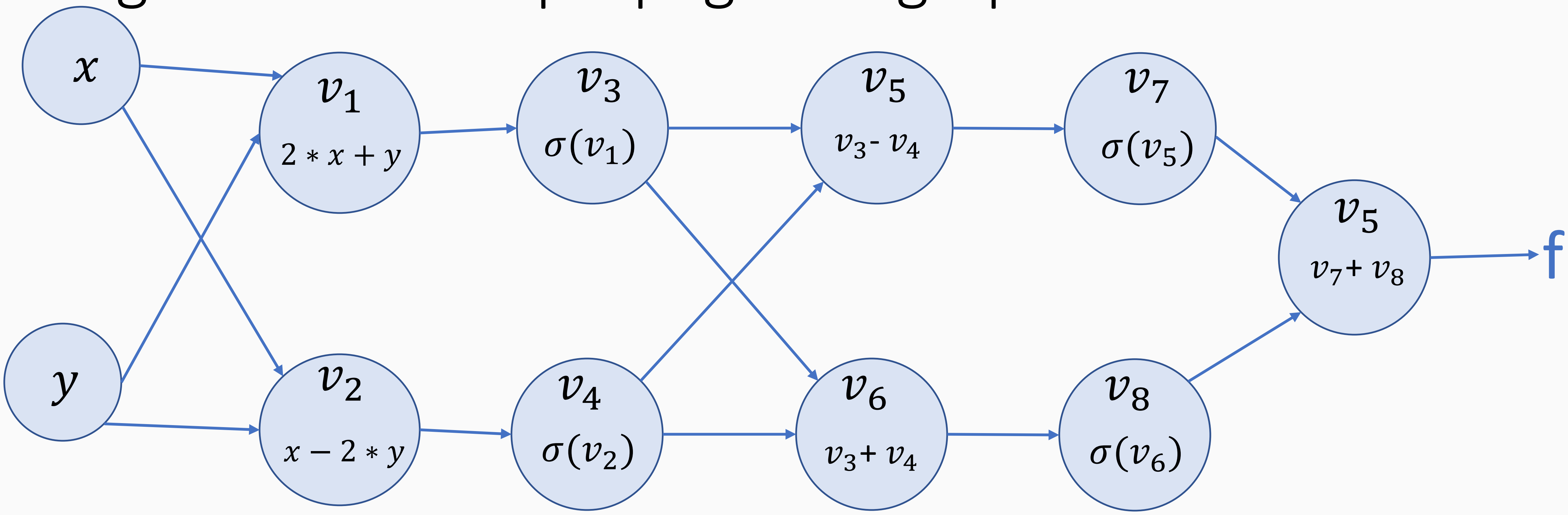
$Example\ input\ (x, y, z) = (1,2,1)$

Assignment 6 backpropagation graph 2



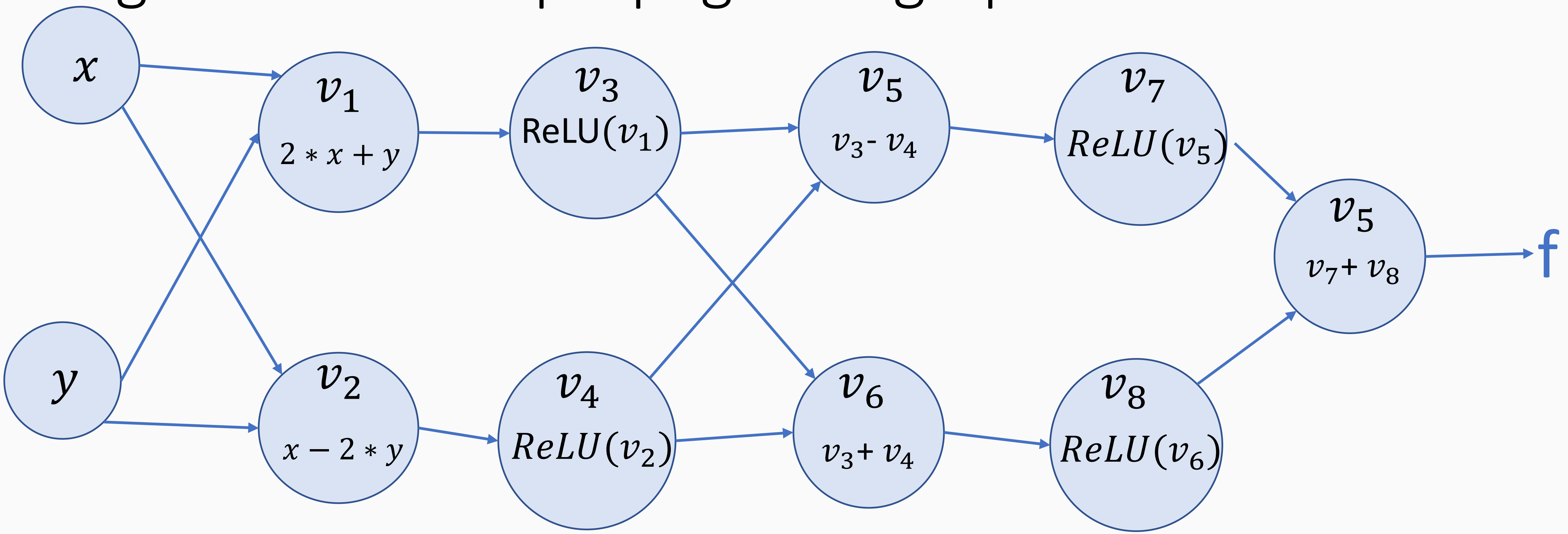
$Example\ input\ (x, y, z) = (1,1,1)$

Assignment 6 backpropagation graph 3



$Example\ input\ (x, y) = (1,1)$

Assignment 6 backpropagation graph 4



$$ReLU(x) = \begin{cases} x \text{ if } x > 0, \\ 0 \text{ if } x \leq 0 \end{cases},$$

$Example\ input\ (x, y) = (1,1)$

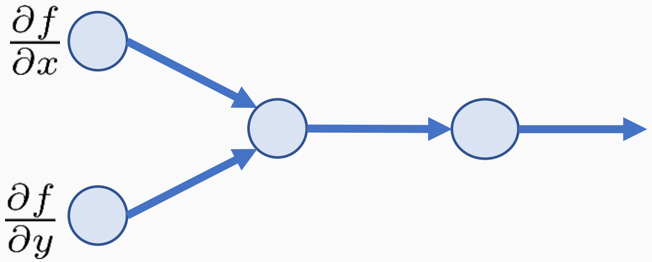
$$ReLU'(x) = \begin{cases} 1 \text{ if } x > 0, \\ 0 \text{ if } x \leq 0 \end{cases}$$

Backpropagation gradient checking

`How to debug my backprop algorithm implementation ???`

Given (x, y)

Backprop to compute the partial derivatives



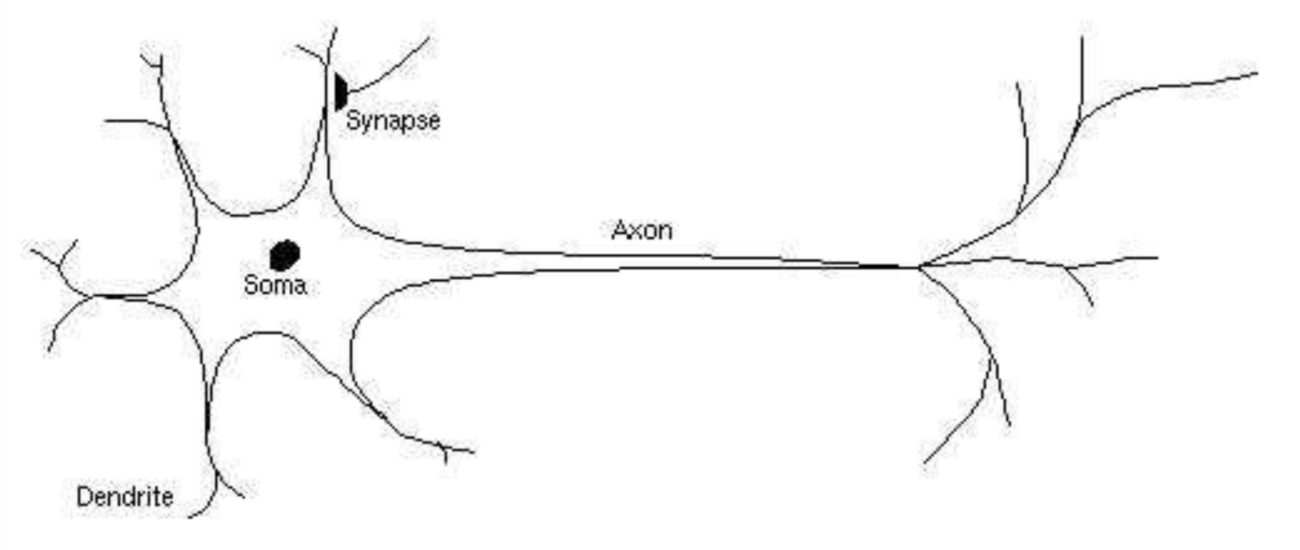
Estimate the rate of change of $f(x, y)$ for fixed (small) h in x and y direction

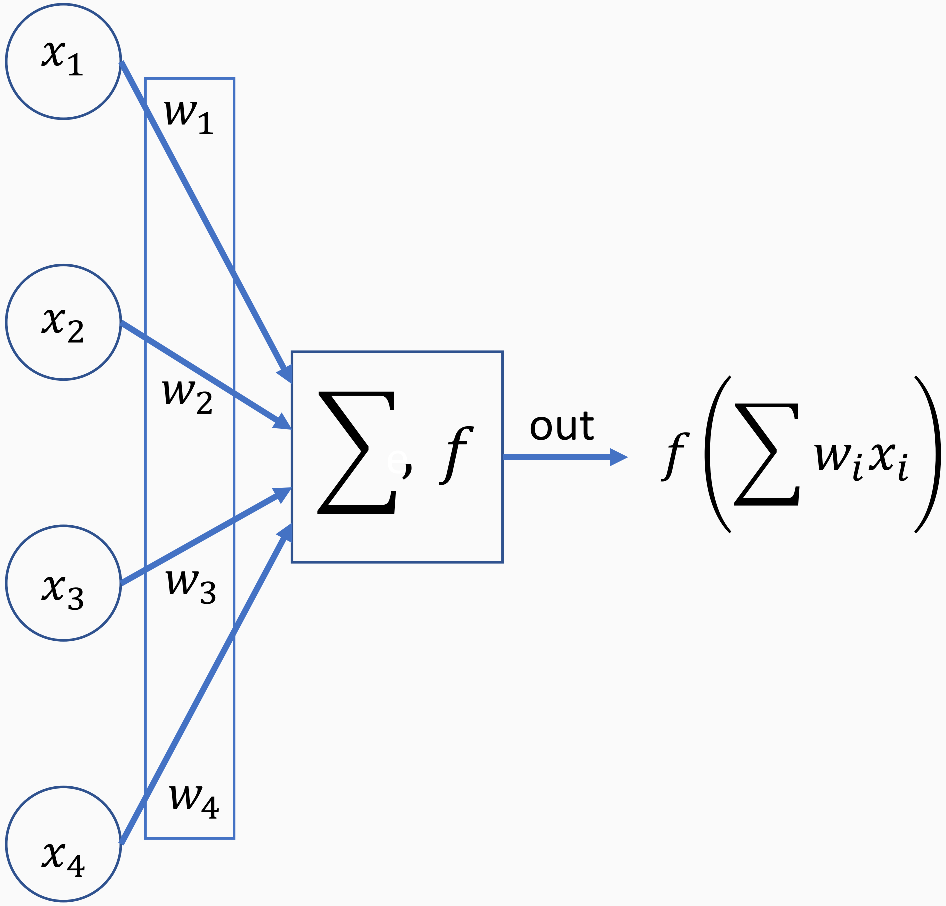
$$\frac{\partial f}{\partial x} \approx \frac{f(x+h, y) - f(x, y)}{h}$$

$$\frac{\partial f}{\partial y} \approx \frac{f(x, y+h) - f(x, y)}{h}$$

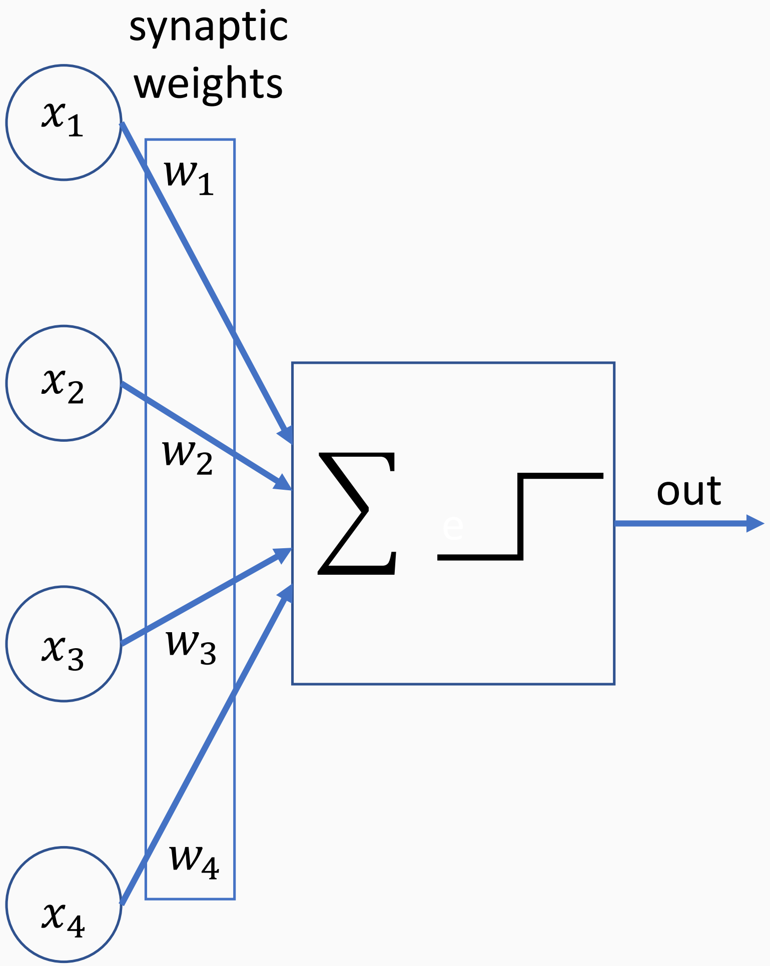
`Should be comparable!`

Biological vs artificial neural nets





Original perceptron model



Original perceptron used the *Heaviside activation function*

$$f(x) = \begin{cases} 1, \text{ if } w \cdot x + b > 0, \\ 0, \text{ otherwise} \end{cases}$$

Rosenblatt, Frank (1957), *The Perceptron--a perceiving and recognizing automaton*. Report 85-460-1, Cornell Aeronautical Laboratory.

Separates two regions of the space using linear (hyper)plane, can perform linear separable tasks.

1	0	1
0	0	0
AND	0	1

1	1	1
0	0	1
OR	0	1

1	1	0
0	1	0
NOT	0	1

However, not all logic operations are linearly separable

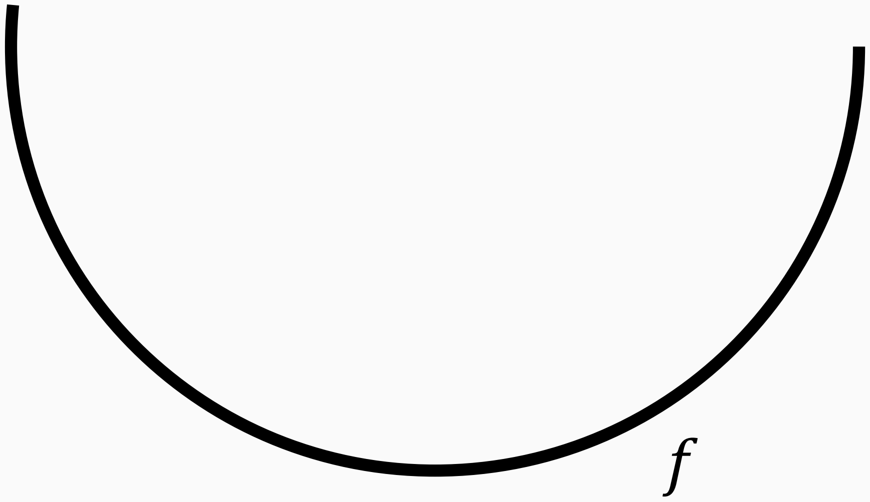
1	1	0
0	0	1
XOR	0	1

<u>Need to use two hidden layers to solve XOR</u>

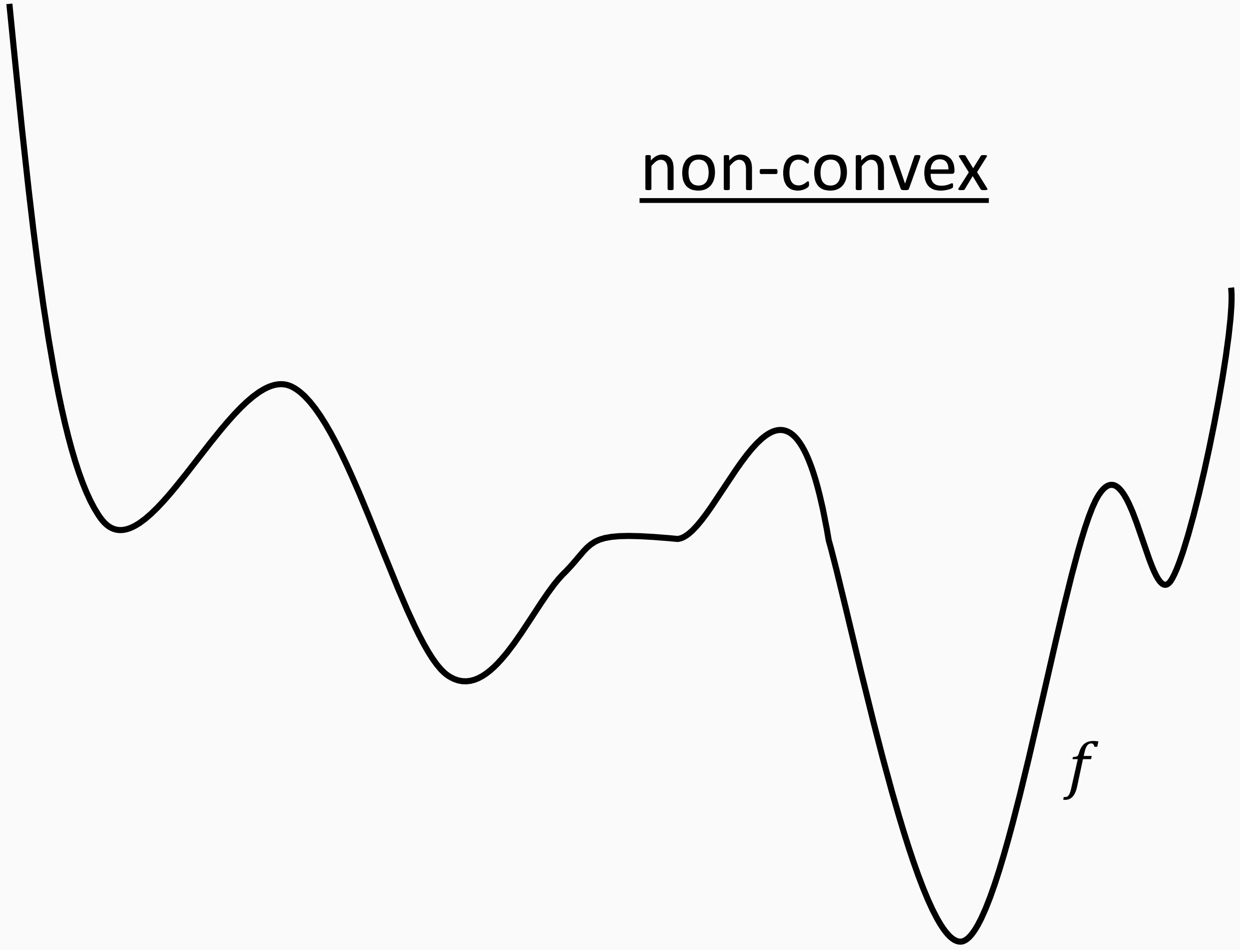
Why training NN is hard ???

The function of NN weights that is being optimized is nonconvex

convex

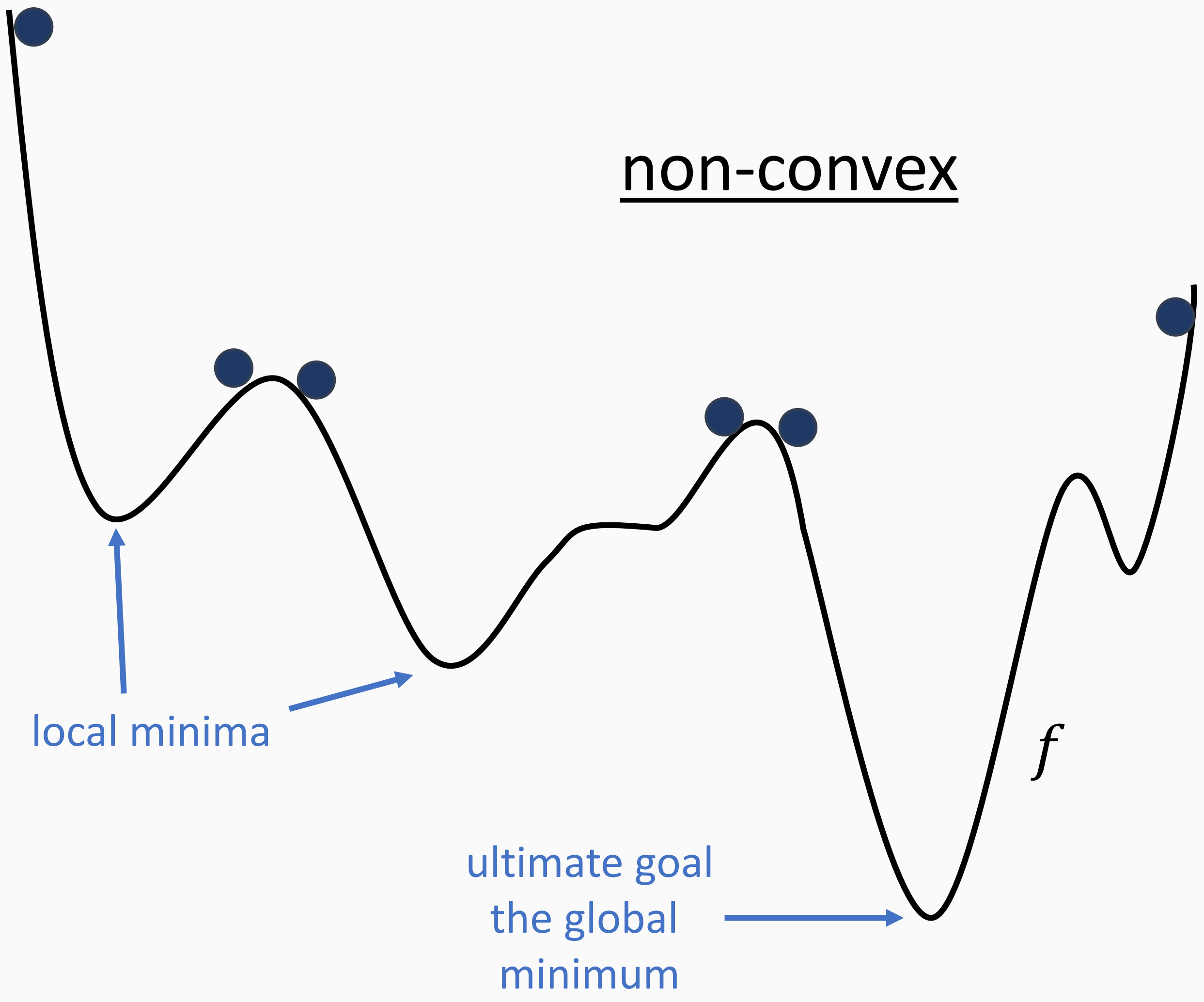


non-convex



Hardness of nonconvex optimization

1. There can be local minima, which are way worse than global,
2. There can be saddle-points,
3. No control where gradient descent will converges to,
4. Gradient descent converge to different critical points depends on initial points.



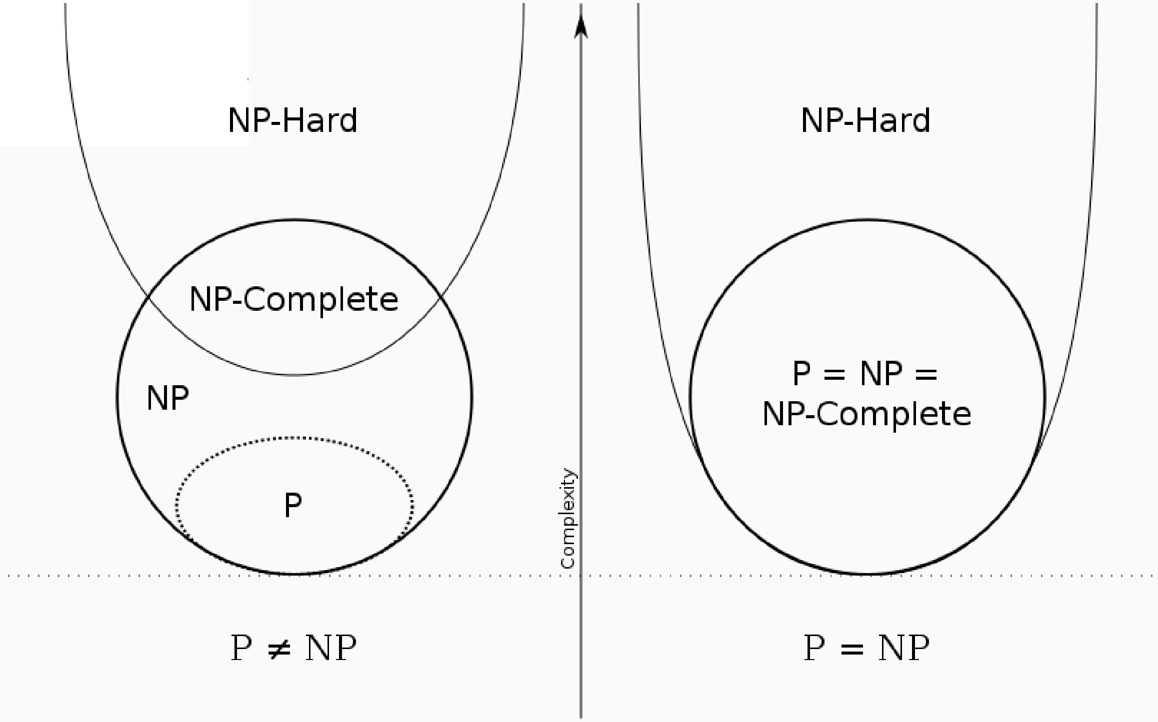
Why training NN is hard ??? (NP-complete)

Training a 3-Node Neural Network is NP-Complete

*Avrim L. Blum and Ronald L. Rivest**

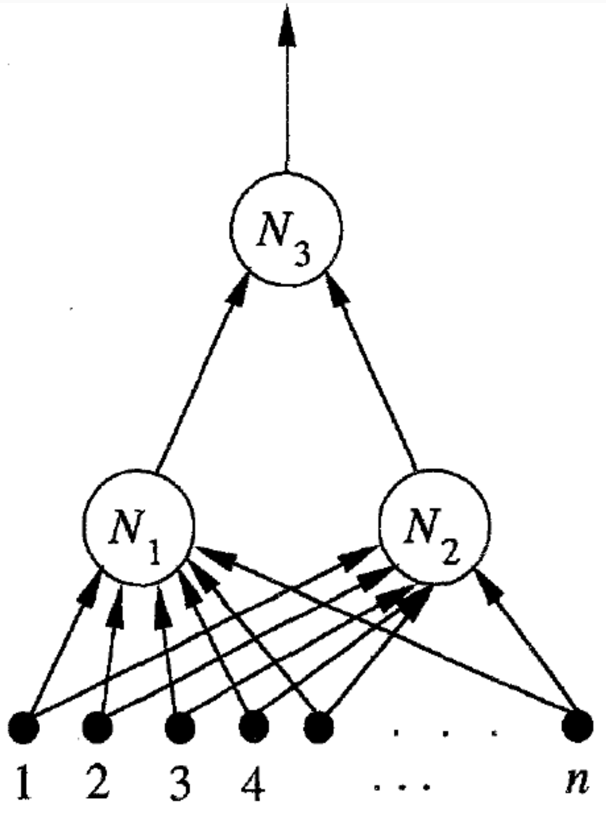
MIT Laboratory for Computer Science
Cambridge, Massachusetts 02139
avrim@theory.lcs.mit.edu
rivest@theory.lcs.mit.edu

1. NP-complete problem means any provided example solution can be verified in polynomial time.
2. For any NP problem the best known algorithm of finding a solution runs in polynomial time.
3. Any problem in NP can be reduced to a NP-complete problem in polynomial time.
4. If there exists a polynomial time algorithm for solving any of NP-complete problems, then P=NP.



NP-completeness cont.

Training of the three node neural network to perform AND on the outputs of N_1, N_2 is NP-complete.



Where each node N_i computes a linear threshold function

$$N_i(x) = \begin{cases} +1 \text{ if } a_1x_1 + a_2x_2 + \cdots + a_mx_m > a_0, \\ -1 \text{ otherwise} \end{cases}$$

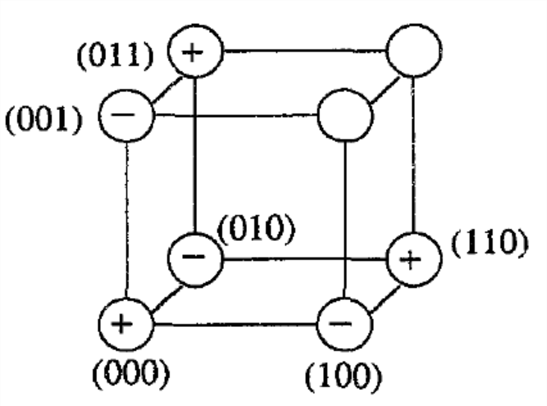
Show that this problem is NP-complete by reduction to the problem of *Set-Splitting* known to be NP-complete

The following problem, *Set-Splitting*, was proven to be NP-complete by Lovász (see Garey and Johnson [6]).

> "Given a finite set S and a collection C of subsets c_i of S, do there exist disjoint sets S_1, S_2 such that $S_1 \cup S_2 = S$ and for each i, $c_i \not\subset S_1$ and $c_i \not\subset S_2$?"

NP-completeness cont.

1. Arrange all training examples on a n-dimensional hypercube



2. Label all nodes (training examples) +/− according to the set-splitting instance

– Let the origin 0^n be labeled '+'.
– For each s_i, put a point labeled '−' at the neighbor to the origin that has a 1 in the ith bit: that is, at $(\overset{1\,2\,\dots}{00}\cdots\overset{i}{0\,1\,0}\cdots\overset{\dots\,n}{0})$. Call this point $\mathbf{p}_i$.
– For each $c_j = \{s_{j1}, \dots, s_{jk_j}\}$, put a point labeled '+' at the location whose bits are 1 at exactly the positions $j_1, j_2, \dots, j_{k_j}$: that is, at $\mathbf{p}_{j1} + \cdots + \mathbf{p}_{jk_j}$.

Set-splitting example corresponding to the cube above is

$S = \{s_1, s_2, s_3\}$, $c_1 = \{s_1, s_2\}$, $c_2 = \{s_2, s_3\}$.
Hence, '−' for nodes 001, 010, 100, and '+' for nodes 000, 011, 110.

3. There is a solution of set-splitting problem ↔ there exists two hyperplanes such that all positive/negative nodes are separated within one of the quadrants.

Sketch the NP-completeness proof

<u>one way ⇒</u>

1. Given $S_{1,} S_2$ from the solution of to the Set-Splitting. Define planes
   P_1: $a_1 x_1 + \cdots + a_n x_n = -\frac{1}{2}$, where $a_i = -1$ if $s_i \in S_1$, and $a_i = n$ if $s_i \notin S_1$
   And similarly
   P_2: $b_1 x_1 + \cdots + b_n x_n = -\frac{1}{2}$, where $b_i = -1$ if $s_i \in S_2$, and $b_i = n$ if $s_i \notin S_2$
2. P_1 separates all $-$ nodes from the origin, because they evaluate to $-1 < -\frac{1}{2}$, and also P_2 separates all $-$ nodes from the origin as they evaluate to $-1 < -\frac{1}{2}$ also.
3. Hence, the quadrant $P_1 > -\frac{1}{2}, P_2 > -\frac{1}{2}$ contains $+$ nodes exclusively and therefore the three-node network can be trained to perform AND($N_{1,} N_2$) operation.

Sketch of the NP-completeness proof II

<u>the other way $\Leftarrow$</u>

1. Given two planes P_1, P_2, let S_1 be the set of points separated from the origin by P_1 ($-$ nodes), and S_2 be the set of points sep. from the origin by P_2 ($-$ nodes).

2. It holds $S = S_1 \cup S_2$, as all $-$ nodes are separated by either P_1 or P_2.

3. Consider $c_j = \{s_{j_1}, \dots, s_{j_k}\}$, if say $c_j \subset S_1$, then P_1 must separate all $p_i = (0 \ldots 0\overset{j_i}{1}0 \ldots 0)$ from the origin.

4. But then P_1 must separate also $p_1 + p_2 + \cdots + p_k$ ($+$ node) which means that $+$ nodes are not confined to a single quadrant, which contradicts the assumption.

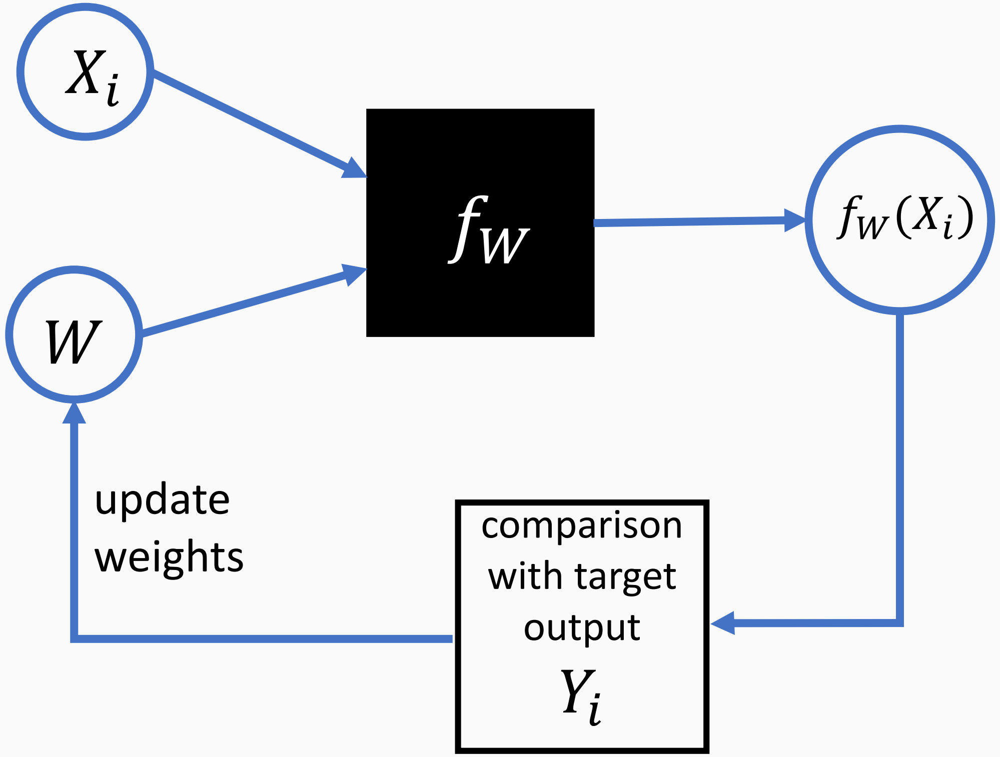
TOPIC:
SINGLE HIDDEN LAYER NEURAL NETWORK, SUPERVISED LEARNING

General algorithm of neural network supervised learning

In supervised learning there need to be a teacher

Teacher in the context of NN ≡ set of N training pairs $\{(X_i, Y_i)\}_{i=1}^{N}$

For each i, and training pair (X_i, Y_i)



Goal is to adjust W, such that the loss value

$L(W) = \frac{1}{N}\sum_{i=1}^{N} ||f_W(X_i) - Y_i||_2^2$ is minimized.

New convention of indexing

As we start dealing with multiple layers and tensors we need to introduce a new convention for indexing

position in sequence (layer number)

indices (row and column #)

multi-index

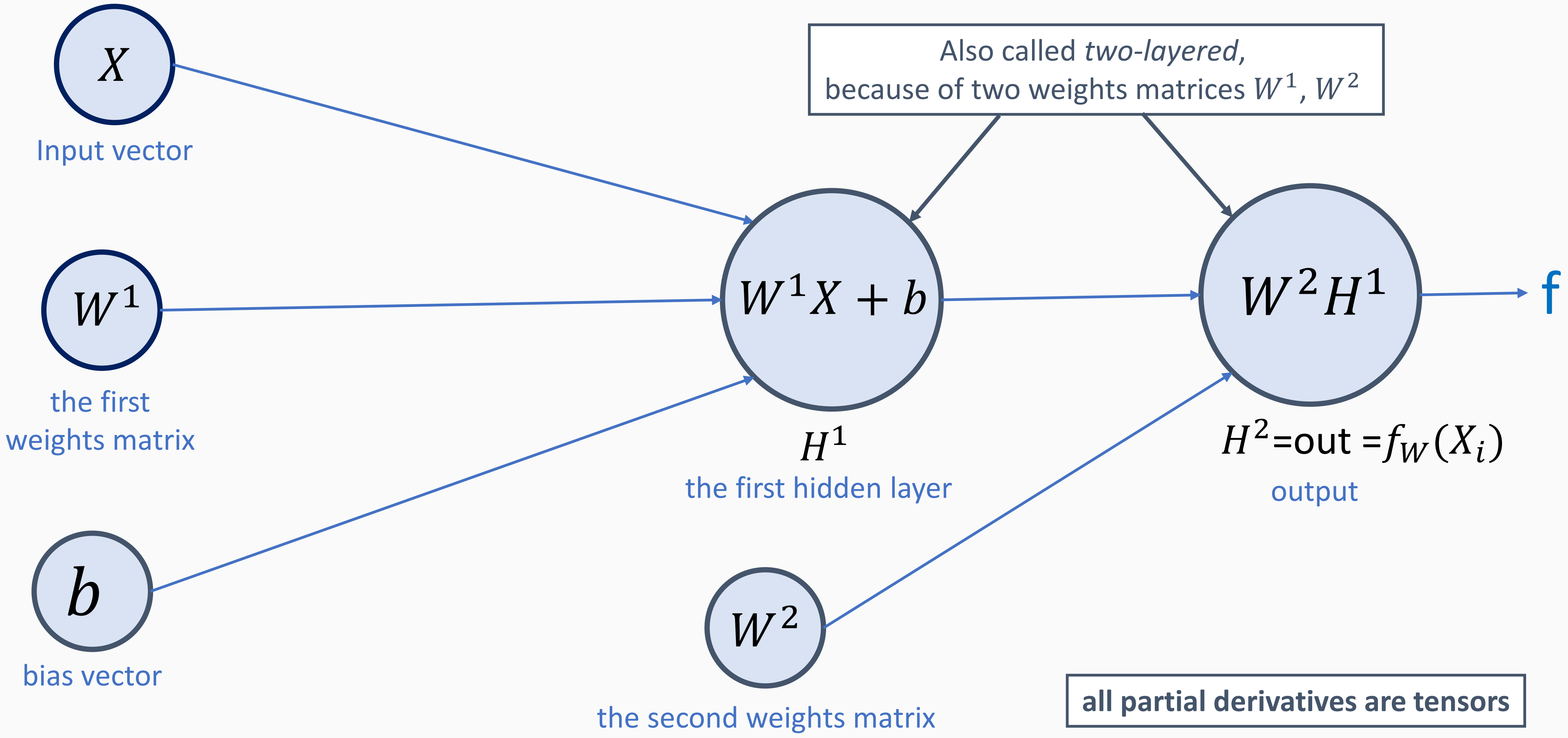
$$W^i \text{ weight matrix,}$$

$$W^i_{jk} \text{ weight matrix components,}$$

$$H^i \text{ matrix of hidden layer values,}$$

$$T_j \text{ a tensor (partial derivatives).}$$

One hidden-layer neural net with linear activation



all partial derivatives are tensors

Simple network dimensions

There are t, n dimensional examples, hence $X \in \mathbb{R}^{n \times t}$, and $X_i \in \mathbb{R}^n$ is a single example

NN has n input, m hidden layer, k output neurons (denoted $NN(n, m, k)$)

Hence,

- $Y \in \mathbb{R}^{k \times t}$, $Y_i \in \mathbb{R}^k$
- $W^1 \in \mathbb{R}^{m \times n}$,
- $W^2 \in \mathbb{R}^{k \times m}$.

By convention, W is the cartesian product of weight matrices

$$W = W^1 \times W^2$$

Goal of supervised learning

Minimize

$$L(W) = \frac{1}{t}\sum_{i=1}^{t} \|f_W(X_i) - Y_i\|_2^2$$

take sum of squares norm, because X_i, Y_i and NN output are vectors.

Apply gradient descent for finding $W^* = argmin\{\, L(W) \,\}$

$$W := W - \alpha \nabla L(W)$$

Compute the gradient of $L(W)$ with respect to weights.

$$\nabla L(W) = \begin{bmatrix} \nabla_{W^1} L(W), \\ \nabla_{W^2} L(W) \end{bmatrix}$$

$$W^1 := W^1 - \alpha \nabla_{W^1} L(W), \quad W^2 := W^2 - \alpha \nabla_{W^2} L(W).$$

Compute the gradient of the loss function

$$L(W) = \frac{1}{t}\sum_{i=1}^{t} \|f_W(X_i) - Y_i\|_2^2$$

Iterate over whole training set $\{X_i, Y_i\}_{i=1}^t$

derivative of NN with respect to weights W1 (tensor)

matrix

$$\nabla_{W^1} L(W) = \frac{2}{t}\sum_{i=1}^{t} (f_W(X_i) - Y_i) \cdot \nabla_{W^1} f_W(X_i),$$

vector

$$\nabla_{W^2} L(W) = \frac{2}{t}\sum_{i=1}^{t} (f_W(X_i) - Y_i) \cdot \nabla_{W^2} f_W(X_i)$$

derivative of NN with respect to weights W2 (tensor

Vector - tensor product

$$(f_W(X_i) - Y_i) \cdot \nabla_{W^1} f_W(X_i) = \sum_{l=1}^{k} (f_W(X_i) - Y_i)_l \, \nabla_{W^1}(f_W(X_i))_l$$

Tensor partial derivatives

`Question: how to compute derivatives with respect to a matrix ???`

H^2=out $=f_W(X_i) = W^2 \cdot H^1$

$$\frac{\partial H^2}{\partial W^2}$$

H^2 is a vector

Derivative w.r.t. matrix W^2
(it is a tensor)

$$H^2 \in \mathbb{R}^k,$$

$$W^2 \in \mathbb{R}^{k \times m},$$

$$\frac{\partial H^2}{\partial W^2} \in \mathbb{R}^{k \times k \times m},$$

a tensor

$$\frac{\partial H^2}{\partial W^1} \in \mathbb{R}^{k \times m \times n}$$

Partial derivatives are tensors

We wanna compute the partial derivatives with respect to weights

$$\underbrace{\frac{\overset{\text{vector}}{\partial H^1}}{\underset{\text{matrix}}{\partial W^1}}}, \quad \frac{\overset{\text{vector}}{\partial H^2}}{\underset{\text{matrix}}{\partial W^2}}, \quad \frac{\overset{\text{vector}}{\partial H^2}}{\underset{\text{matrix}}{\partial W^1}}$$

$$H^1 = W^1 \cdot X_i$$
$$H^2 = W^2 \cdot H^1$$

Hence all of these are 3D tensors, as opposed to 2D matrix

First, we use the explicit formula for

$$\frac{\partial H_j^2}{\partial W^2} = \begin{bmatrix} \overbrace{0\ldots0}^{m \text{ zeroes}} \\ \cdots \\ 0\ldots0 \\ (H^1)^T \\ 0\ldots0 \\ \cdots \\ 0\ldots0 \end{bmatrix} \qquad j\text{-th row} \qquad \begin{bmatrix} \overbrace{0\ldots0}^{n \text{ zeroes}} \\ \cdots \\ 0\ldots0 \\ X_i^T \\ 0\ldots0 \\ \cdots \\ 0\ldots0 \end{bmatrix} = \frac{\partial H_j^1}{\partial W^1}$$

Chain rule for tensors

How to compute the partial derivatives with respect to weights

$j-th$ element of vector

Scalar

matrix

$$\frac{\partial H_j^2}{\partial W^1} = \sum_k \frac{\partial H_j^2}{\partial H_k^1} \frac{\partial H_k^1}{\partial W^1}$$

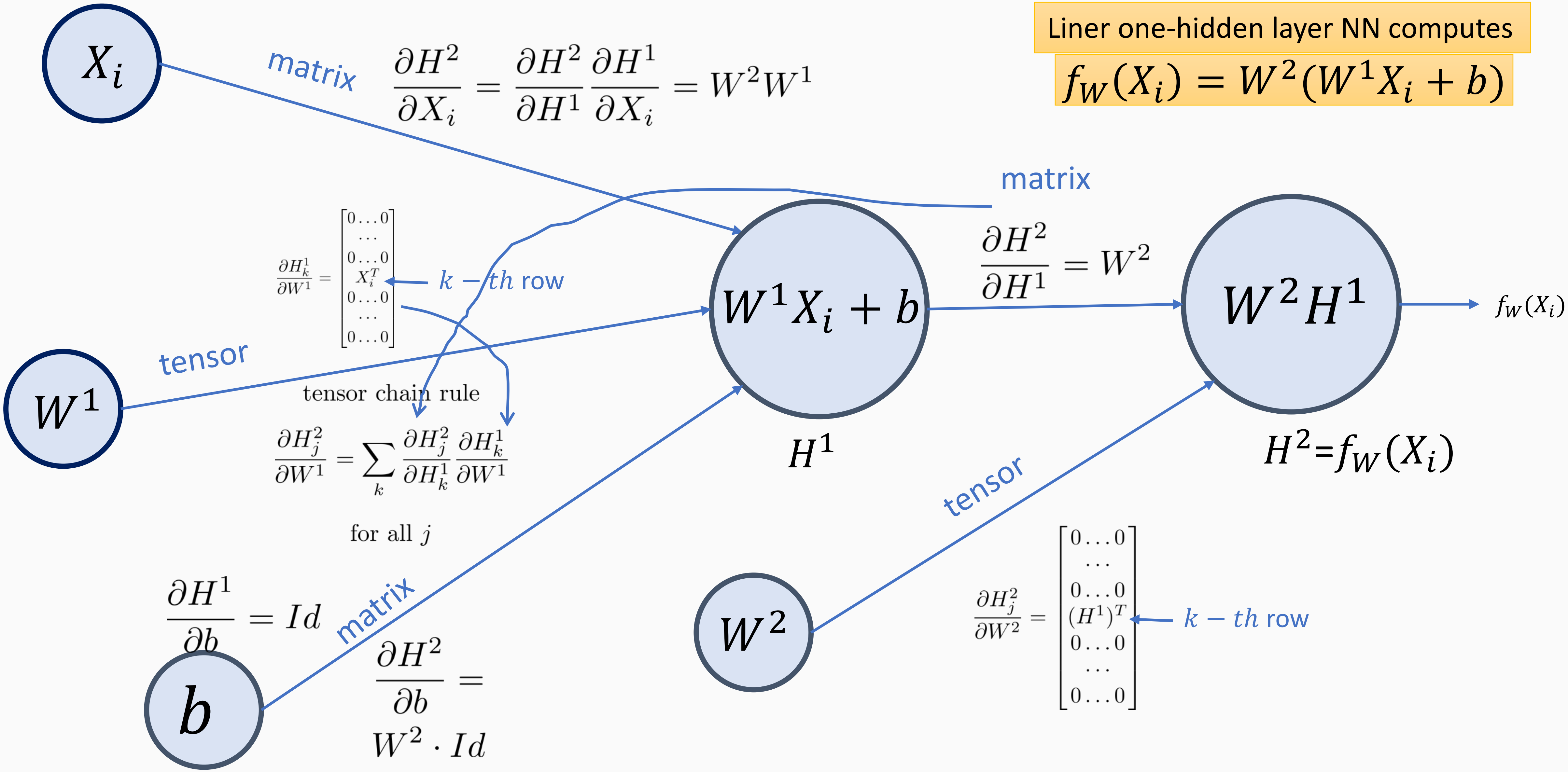
a matrix

Sum over all indices

Scalar

Repeat computation for all indices j of H^2

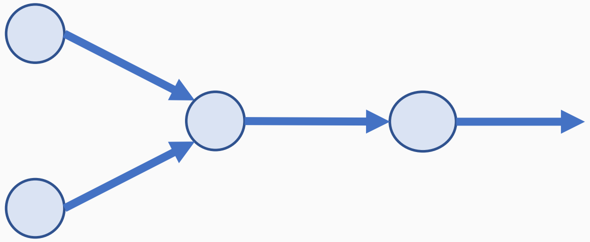
Backprop compute $\nabla_{W^1} f_W(X_i), \nabla_{W^2} f_W(X_i)$



Verify your backprop implementation using gradient checking

Given X_i

Backprop to compute the partial derivatives



$\sim$ comparable

Estimate the rate of change of $f_W(X_i)$ for fixed (small) h in all W^1, W^2 directions

$$\frac{\partial f_W(X_i)}{\partial W} \approx \frac{f_{W+h}(X_i) - f_W(X_i)}{h},$$

where $W + h$ means we add h to a sigle component of W (and repeat it for all i to compute the whole partial derivatives tensor

Obtaining tensor partial *numerical* derivative (for gradient checking)

$\frac{\partial f_W(X_i)}{\partial W^1}$ 3D tensor

For example to compute numerical derivative $\approx \frac{\partial f_W(X_i)}{\partial W^1}$
need to estimate the finite difference in all possible directions,
i.e. for all $1 \le i \le m, 1 \le j \le n$ compute modified weights W^1 by

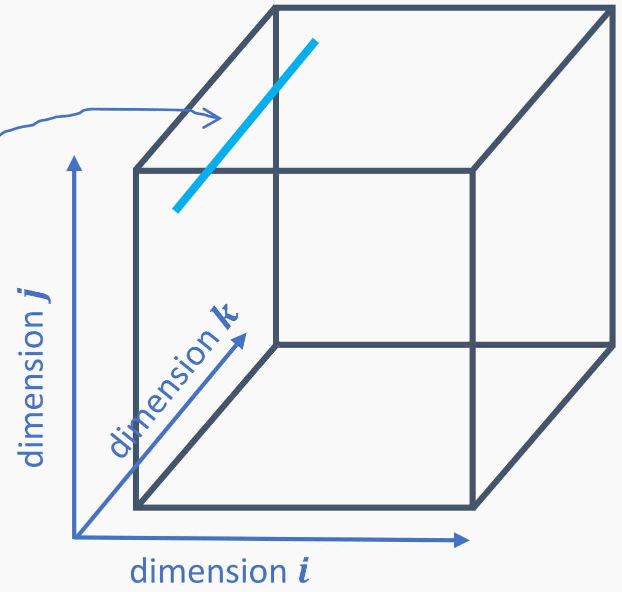
$W^{1*} = W^1$, where for the selected entry we set $W_{ij}^{1*} = W_{ij}^1 + h$, we obtain whole slice of the partial derivatives tensor,
remembering $W = W^1 \times W^2$, we set $W^* = W^{1*} \times W^2$

Weight matrices pair having i, j-th entry of W^1 modified

$$\frac{\partial f_W(X_i)}{\partial W_{ij}^1} \approx \frac{f_{W^*}(X_i) - f_W(X_i)}{h}$$

And repeat this for all i, j

Value of this is a vector [:,i,j] slice of the partial derivative tensor

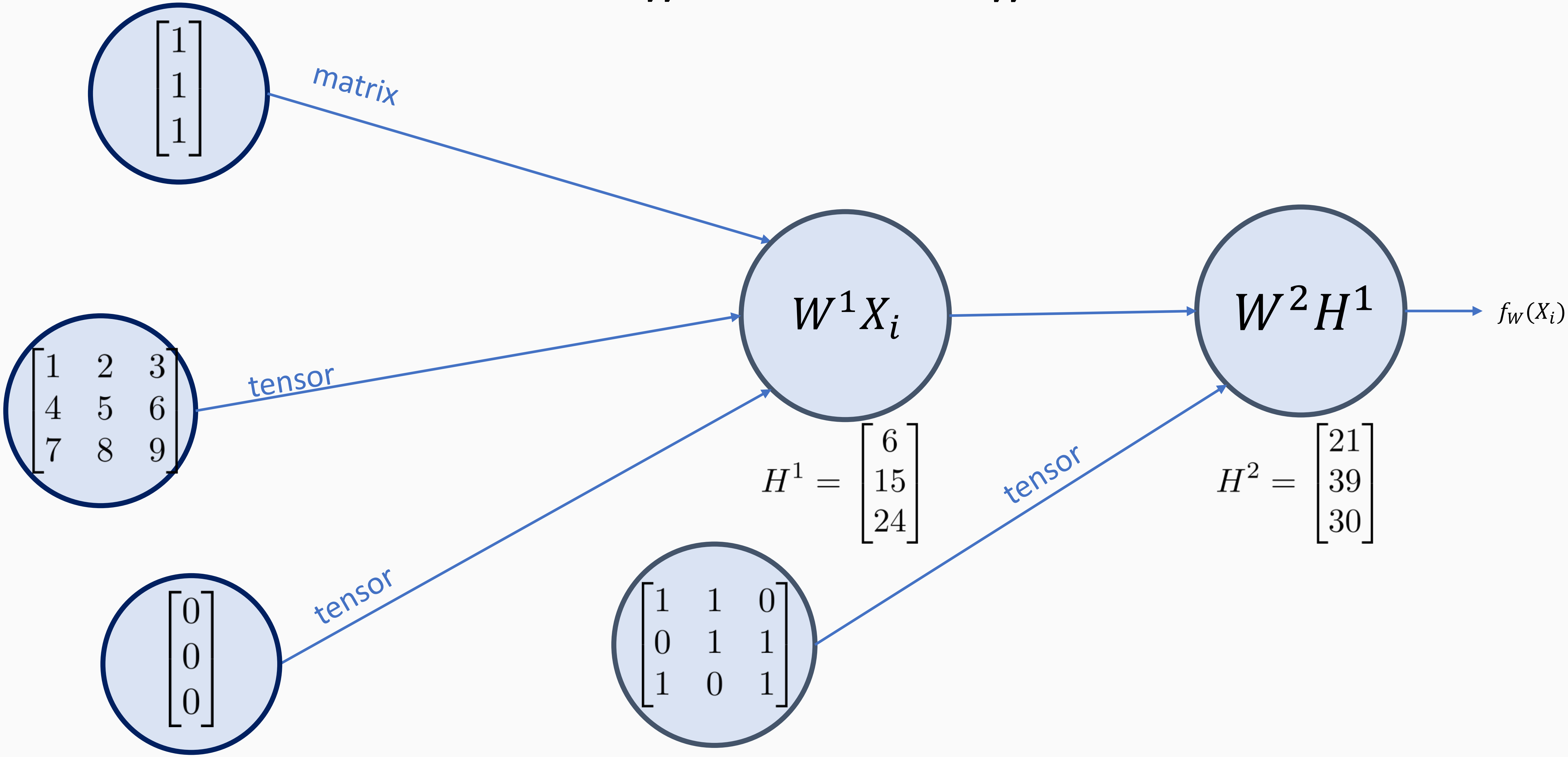


element of the tensor indexed by (k, i, j) is

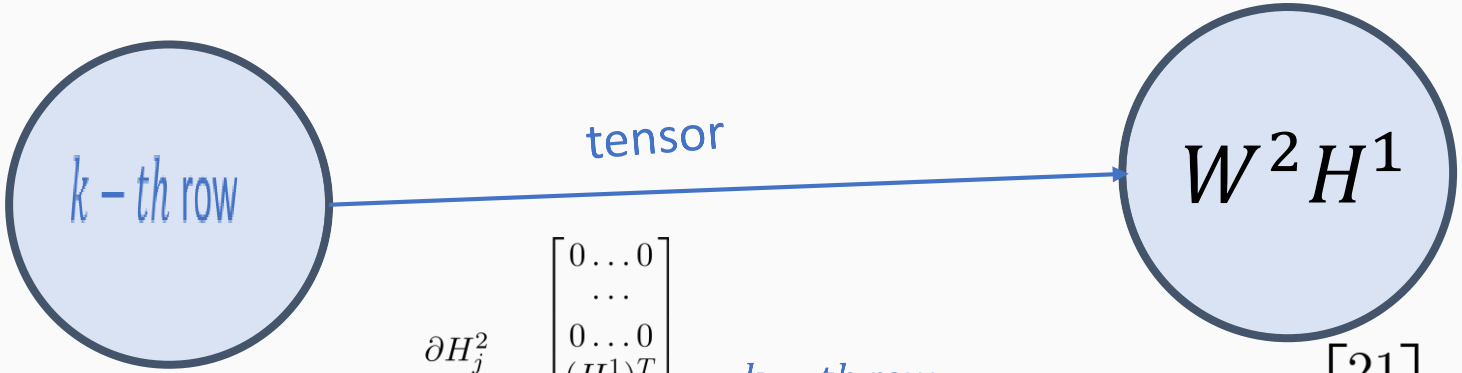
$$\frac{\partial f_W(X_i)_k}{\partial W_{ij}^1}$$

i.e. dimension k is the dimension of vector, i, j are dimensions of weight matrices.

Concrete example $\nabla_{W^1} f_W(X_i), \nabla_{W^2} f_W(X_i), b = 0$



Edge $W^2 - H^2$



$$H^1 = \begin{bmatrix} 6 \\ 15 \\ 24 \end{bmatrix}$$

$$\frac{\partial H_j^2}{\partial W^2} = \begin{bmatrix} 0 \dots 0 \\ \dots \\ 0 \dots 0 \\ (H^1)^T \\ 0 \dots 0 \\ \dots \\ 0 \dots 0 \end{bmatrix} \quad k - th \text{ row}$$

$$H^2 = \begin{bmatrix} 21 \\ 39 \\ 30 \end{bmatrix}$$

$$\text{In this case } \frac{\partial H^2}{\partial W^2} \text{ is a } 3 \times 3 \times 3 \text{ dimensional tensor}$$

$$\frac{\partial H_1^2}{\partial W^2} = \begin{bmatrix} 6 & 15 & 24 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \frac{\partial H_2^2}{\partial W^2} = \begin{bmatrix} 0 & 0 & 0 \\ 6 & 15 & 24 \\ 0 & 0 & 0 \end{bmatrix} \quad \frac{\partial H_3^2}{\partial W^2} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 6 & 15 & 24 \end{bmatrix}$$

$$\frac{\partial H^2}{\partial W^2} = \begin{bmatrix} 6 & 15 & 24 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 6 & 15 & 24 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 6 & 15 & 24 \end{bmatrix}$$

$$H^2 = f_W(X_i)$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \xrightarrow{\text{tensor}} W^1 X_i \longrightarrow W^2 H^1 \longrightarrow f_W(X_i)$$

$$\frac{\partial H^1_k}{\partial W^1} = \begin{bmatrix} 0 \ldots 0 \\ \cdots \\ 0 \ldots 0 \\ X_i^T \\ 0 \ldots 0 \\ \cdots \\ 0 \ldots 0 \end{bmatrix}$$

$$\frac{\partial H^2}{\partial H^1} = W^2$$

$$W^2 = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

tensor chain rule

$$\frac{\partial H^2_j}{\partial W^1} = \sum_k \frac{\partial H^2_j}{\partial H^1_k} \frac{\partial H^1_k}{\partial W^1}$$

for all j

$$X = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \qquad H^1 = \begin{bmatrix} 6 \\ 15 \\ 24 \end{bmatrix}$$

In this case $\frac{\partial H^1}{\partial W^1}$ is a $3 \times 3 \times 3$ dimensional tensor

$$\frac{\partial H^1_1}{\partial W^1} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \frac{\partial H^1_2}{\partial W^1} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \frac{\partial H^1_3}{\partial W^1} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

$$\frac{\partial H^2_1}{\partial W^1} = \sum_k \frac{\partial H^2_1}{\partial H^1_k} \frac{\partial H^1_k}{\partial W^1} = 1 \cdot \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + 1 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} + 0 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Other components of the tensor

$$\frac{\partial H_2^2}{\partial W^1} = \sum_k \frac{\partial H_2^2}{\partial H_k^1} \frac{\partial H_k^1}{\partial W^1} = 0 \cdot \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + 1 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} + 1 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\frac{\partial H_3^2}{\partial W^1} = \sum_k \frac{\partial H_3^2}{\partial H_k^1} \frac{\partial H_k^1}{\partial W^1} = 1 \cdot \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + 0 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} + 1 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\frac{\partial H^2}{\partial W^1} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Compute the gradient of the loss function

$$L(W) = \frac{1}{t}\sum_{i=1}^{t} \|f_W(X_i) - Y_i\|_2^2$$

By chain rule

derivative of NN with respect to weights W1 (tensor) (inner function) backprop

$$\nabla_{W^1} L(W) = \frac{2}{t}\sum_{i=1}^{t} (f_W(X_i) - Y_i) \cdot \nabla_{W^1} f_W(X_i),$$

matrix

vector

$$\nabla_{W^2} L(W) = \frac{2}{t}\sum_{i=1}^{t} (f_W(X_i) - Y_i) \cdot \nabla_{W^2} f_W(X_i)$$

derivative of NN with respect to weights W2 (tensor) (inner function) backprop

Vector - tensor product

$$(f_W(X_i) - Y_i) \cdot \nabla_{W^1} f_W(X_i) = \sum_{l=1}^{k} (f_W(X_i) - Y_i)_l \, \nabla_{W^1}(f_W(X_i))_l$$

Gradient of the loss in practice

$$(f_W(X_i) - Y_i) \cdot \nabla_{W^1} f_W(X_i) = \sum_{l=1}^{k} (f_W(X_i) - Y_i)_l \, \nabla_{W^1}(f_W(X_i))_l$$

Assume training pair

$$(X_i, Y_i) = \left(\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 20 \\ 0 \\ 0 \end{bmatrix} \right) \text{ and } f_W(X_i) = \begin{bmatrix} 21 \\ 39 \\ 30 \end{bmatrix}$$

Recall W^1 gradient of the output

$$\frac{\partial H^2}{\partial W^1} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Then, in this case (for the given training pair (X_i, Y_i))

$$(f_W(X_i) - Y_i) \cdot \nabla_{W^1} f_W(X_i) = 1 \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} + 39 \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} + 30 \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

We know everything to perform NN learning

$$(f_W(X_i) - Y_i) \cdot \nabla_{W^1} f_W(X_i) = \begin{bmatrix} 31 & 31 & 31 \\ 40 & 40 & 40 \\ 69 & 69 & 69 \end{bmatrix}$$

Repeat for all pairs in the training set and sum up, this is the final gradient $\nabla_{W^1} L(W)$.

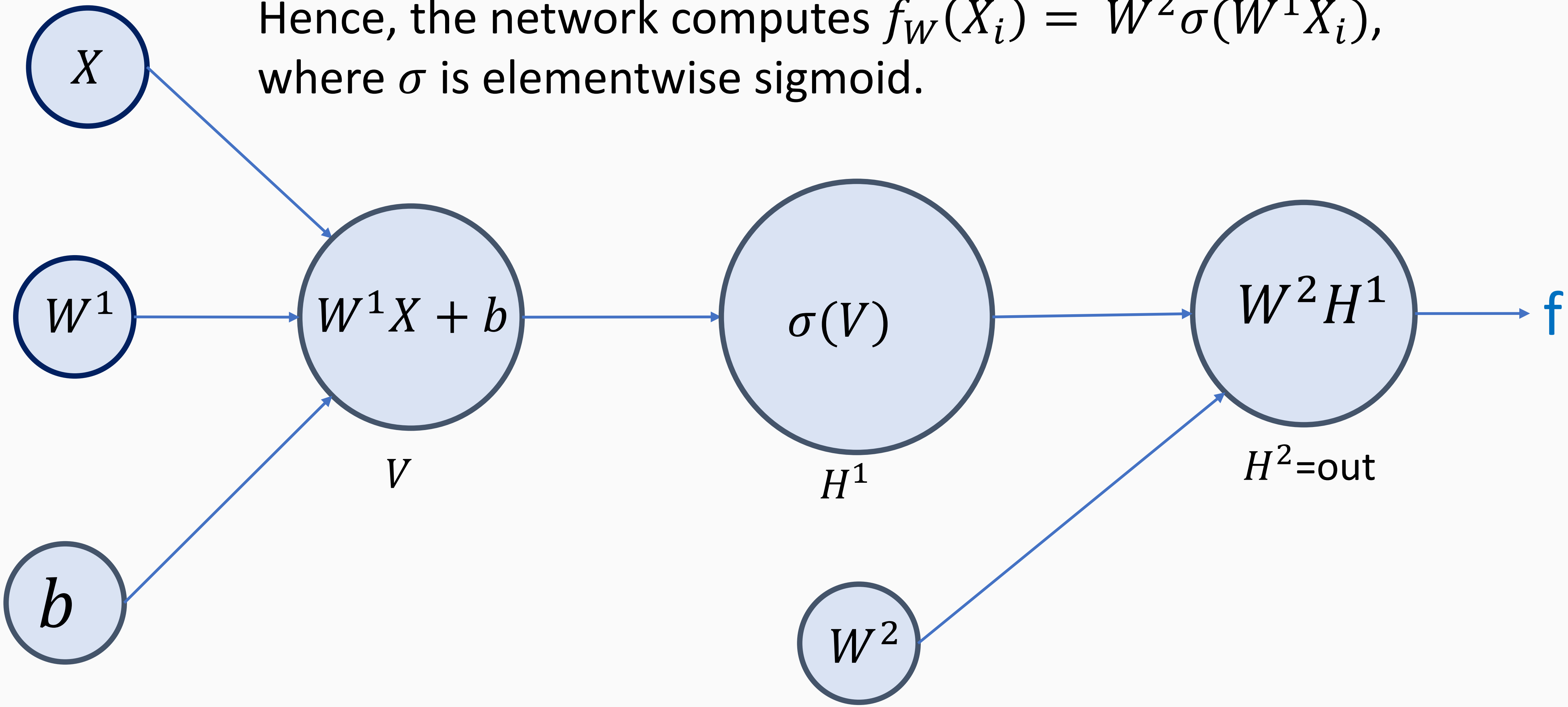
$$\nabla_{W^1} L(W) = \frac{2}{t} \sum_{i=1}^{t} (f_W(X_i) - Y_i) \cdot \nabla_{W^1} f_W(X_i)$$

Compute $\nabla_{W^2} L(W)$ the same way.
And perform the gradient descent update

$$W^1 := W^1 - \alpha \nabla_{W^1} L(W), \quad W^2 := W^2 - \alpha \nabla_{W^2} L(W).$$

Now the hidden layer computes sigmoids

Hence, the network computes $f_W(X_i) = \ W^2\sigma(W^1X_i)$,
where σ is elementwise sigmoid.



The same loss, but sigmoidal network

$$L(W) = \frac{1}{t}\sum_{i=1}^{t} \|f_W(X_i) - Y_i\|_2^2$$

$$\nabla_{W^1} L(W) = \frac{2}{t}\sum_{i=1}^{t} (f_W(X_i) - Y_i) \cdot \nabla_{W^1} f_W(X_i)$$

Gradient of sigmoidal NN

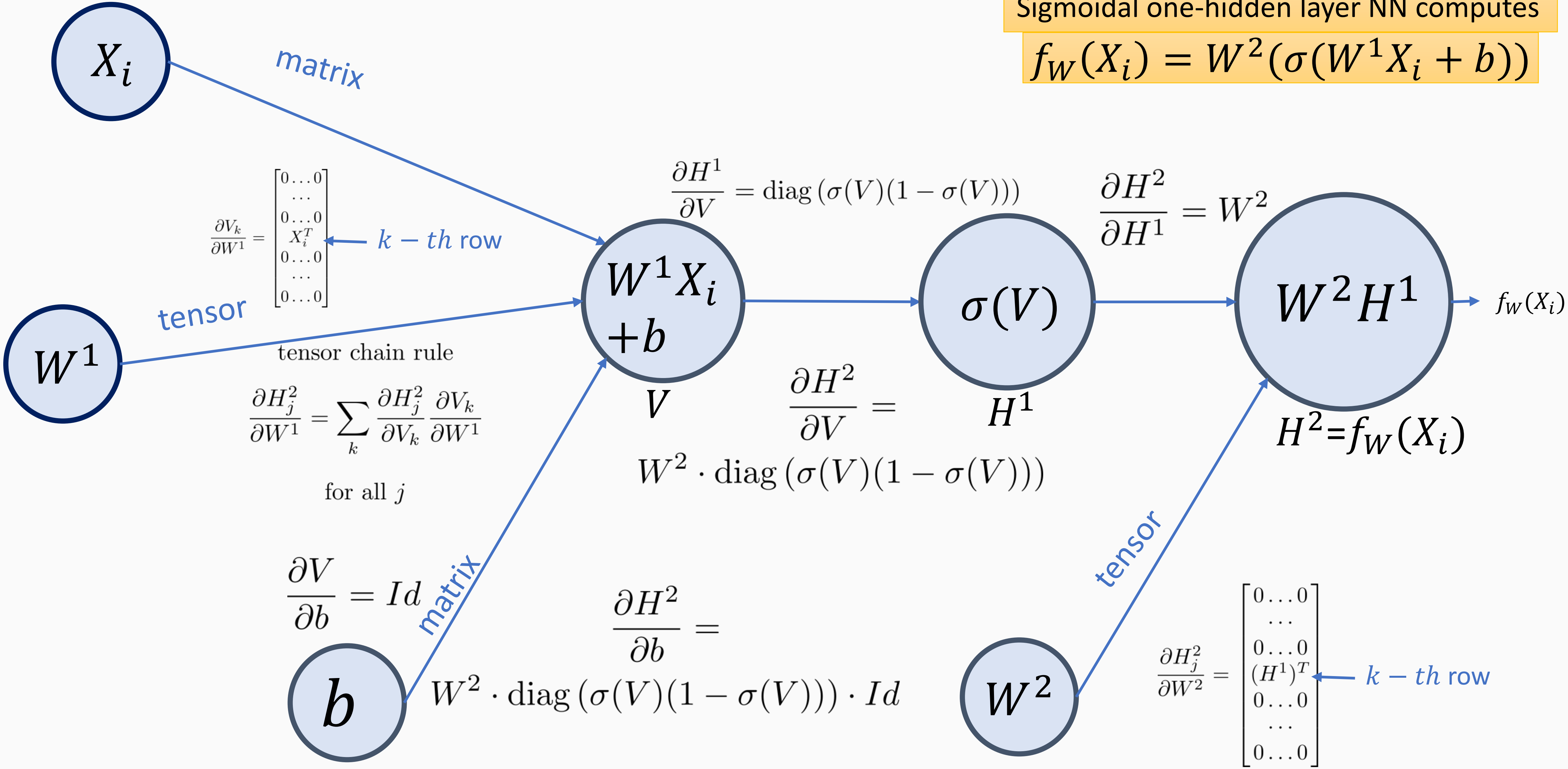
$$\boxed{f_W(X_i) = W^2(\sigma(W^1 X_i + b))}$$

Element-wise sigmoid function

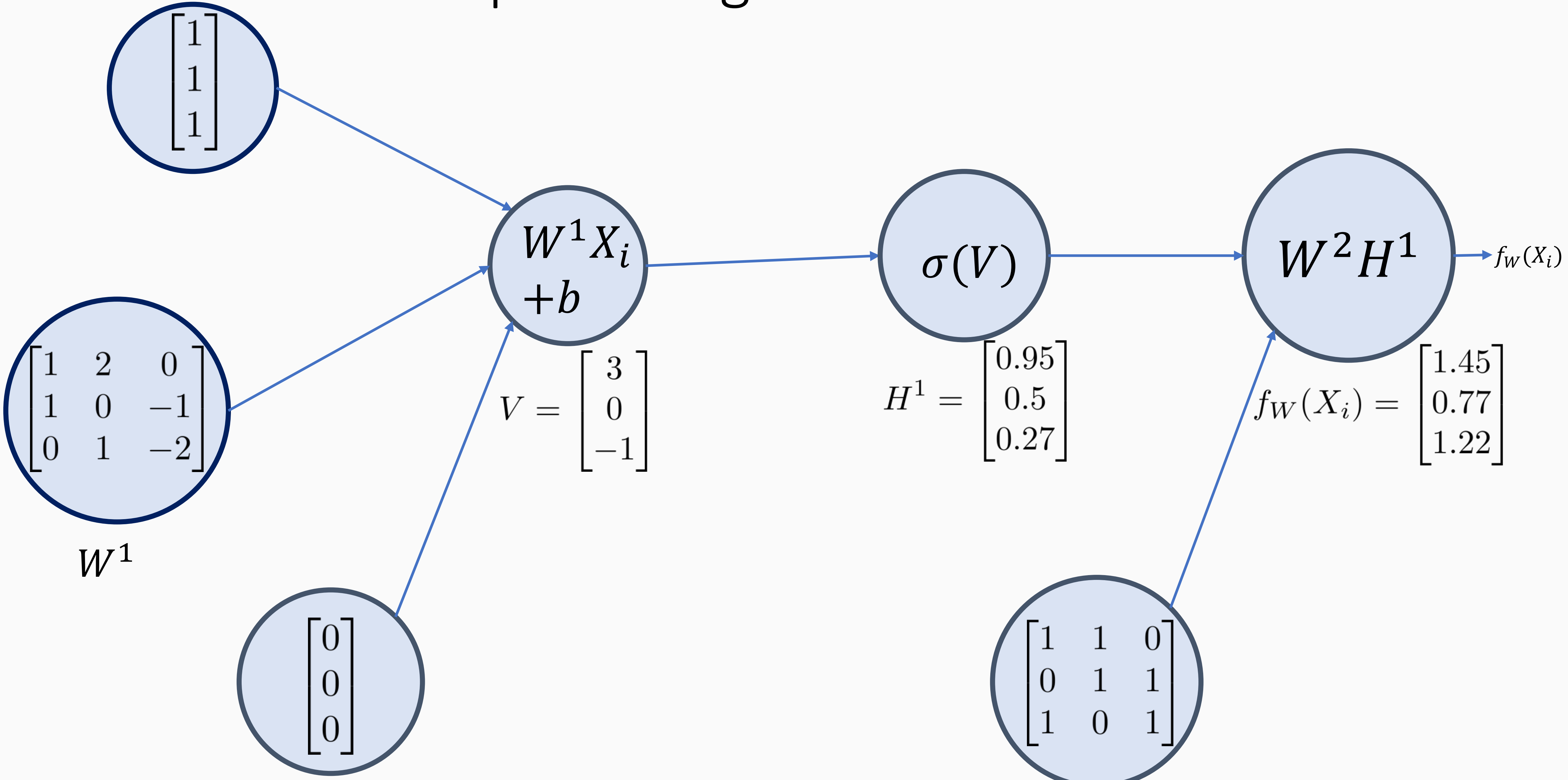
The next slides present how to compute using backprop gradients of sigmoidal net $\nabla_{W^1} f_W(X_i)$, $\nabla_{W^2} f_W(X_i)$

Backprop compute $\nabla_{W^1} f_W(X_i)$, $\nabla_{W^2} f_W(X_i)$ for sigmoidal net

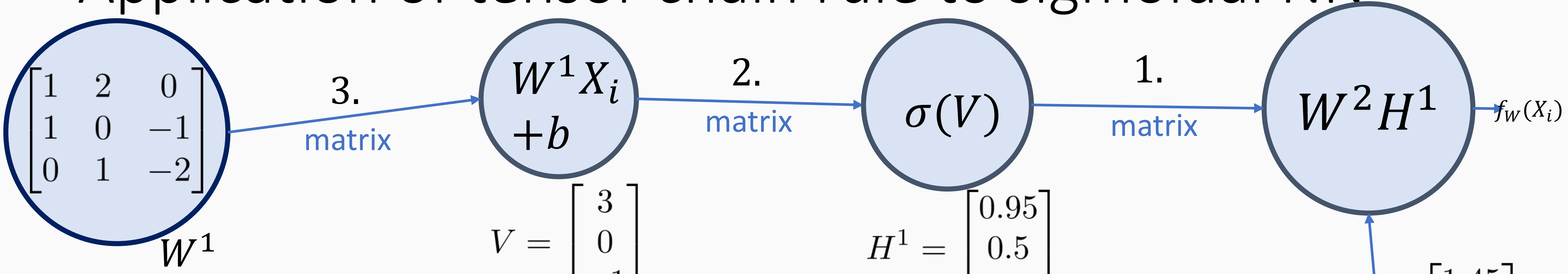
Sigmoidal one-hidden layer NN computes
$f_W(X_i) = W^2(\sigma(W^1 X_i + b))$



Concrete example for sigmoidal network



Application of tensor chain rule to sigmoidal NN



$$V = \begin{bmatrix}3\\ 0\\ -1\end{bmatrix}$$

$$H^1 = \begin{bmatrix}0.95\\ 0.5\\ 0.27\end{bmatrix}$$

$$X = \begin{bmatrix}1\\ 1\\ 1\end{bmatrix}$$

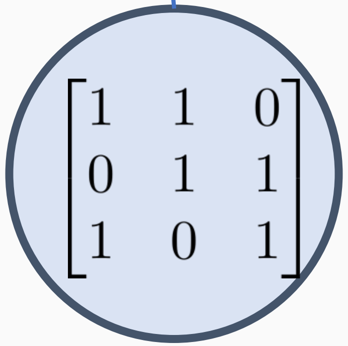
$$H^2 = f_W(X_i) = \begin{bmatrix}1.45\\ 0.77\\ 1.22\end{bmatrix}$$

Using the general technique from previous slides, we compute backprop for the edges marked 1. and 2.

1. $$\frac{\partial H^2}{\partial H^1} = W^2 = \begin{bmatrix}1 & 1 & 0\\ 0 & 1 & 1\\ 1 & 0 & 1\end{bmatrix}$$

2. $$\frac{\partial H^2}{\partial V} = W^2 \cdot \operatorname{diag}\left(\sigma(V)(1-\sigma(V))\right) =$$

$$\begin{bmatrix}1 & 1 & 0\\ 0 & 1 & 1\\ 1 & 0 & 1\end{bmatrix} \cdot \begin{bmatrix}0.95\cdot 0.05 & 0 & 0\\ 0 & 0.5\cdot 0.5 & 0\\ 0 & 0 & 0.27\cdot 0.73\end{bmatrix} = \begin{bmatrix}0.95\cdot 0.05 & 0.5\cdot 0.5 & 0\\ 0 & 0.5\cdot 0.5 & 0.27\cdot 0.73\\ 0.95\cdot 0.05 & 0 & 0.27\cdot 0.73\end{bmatrix}$$



Compute the tensor partial derivative for edge 3.

$$H^2 = f_W(X_i)$$

We apply the general formulas

From step 2. we have

3. $$\frac{\partial V_k}{\partial W^1} = \begin{bmatrix} 0\dots0 \\ \dots \\ 0\dots0 \\ X_i^T \\ 0\dots0 \\ \dots \\ 0\dots0 \end{bmatrix}$$ ← $k-th$ row

tensor chain rule

$$\frac{\partial H_j^2}{\partial W^1} = \sum_k \frac{\partial H_j^2}{\partial V_k} \frac{\partial V_k}{\partial W^1}$$

for all j

$$\frac{\partial H^2}{\partial V} = \begin{bmatrix} 0.95 \cdot 0.05 & 0.5 \cdot 0.5 & 0 \\ 0 & 0.5 \cdot 0.5 & 0.27 \cdot 0.73 \\ 0.95 \cdot 0.05 & 0 & 0.27 \cdot 0.73 \end{bmatrix}$$

$$\frac{\partial H_1^2}{\partial W^1} = 0.95 \cdot 0.05 \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + 0.5 \cdot 0.5 \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.95 \cdot 0.05 & 0.95 \cdot 0.05 & 0.95 \cdot 0.5 \cdot 0.5 \\ 0.5 \cdot 0.5 & 0.5 \cdot 0.5 & 0.5 \cdot 0.5 \\ 0 & 0 & 0 \end{bmatrix},$$

$$\frac{\partial H_2^2}{\partial W^1} = 0.5 \cdot 0.5 \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} + 0.27 \cdot 0.73 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0.5 \cdot 0.5 & 0.5 \cdot 0.5 & 0.5 \cdot 0.5 \\ 0.27 \cdot 0.73 & 0.27 \cdot 0.73 & 0.27 \cdot 0.73 \end{bmatrix},$$

$$\frac{\partial H_3^2}{\partial W^1} = 0.95 \cdot 0.05 \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + 0.27 \cdot 0.73 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0.95 \cdot 0.05 & 0.95 \cdot 0.05 & 0.95 \cdot 0.05 \\ 0 & 0 & 0 \\ 0.27 \cdot 0.73 & 0.27 \cdot 0.73 & 0.27 \cdot 0.73 \end{bmatrix}$$

Combining results

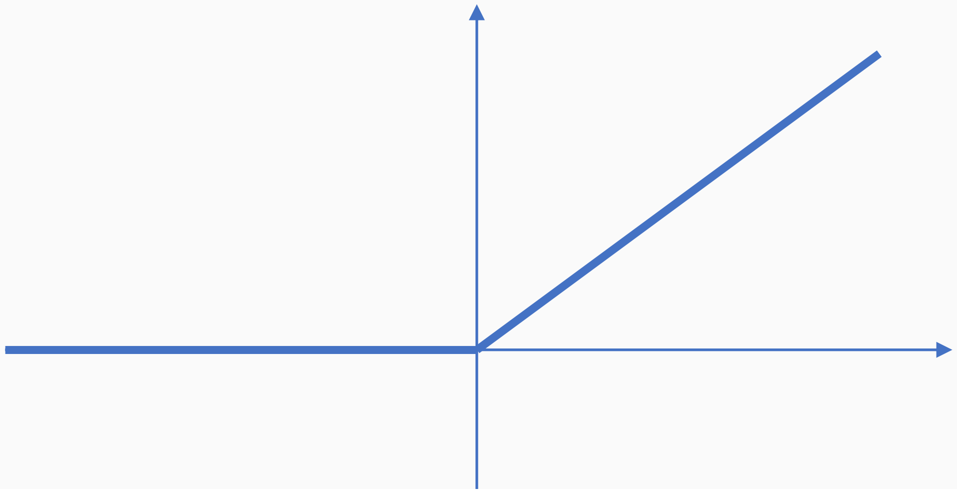
This will give $\nabla_{W^1} f_W(X_i)$, $\nabla_{W^2} f_W(X_i)$ for sigmoidal net.

To compute the gradients of the loss function still need to perform the computation presented on slides **39, 40, 41** (combining derivative of the loss with the gradient of NN).

ReLU net

$$ReLU(x) = \begin{cases} x \text{ if } x > 0, \\ 0 \text{ if } x \leq 0 \end{cases},$$

$$ReLU'(x) = \begin{cases} 1 \text{ if } x > 0, \\ 0 \text{ if } x \leq 0 \end{cases}$$

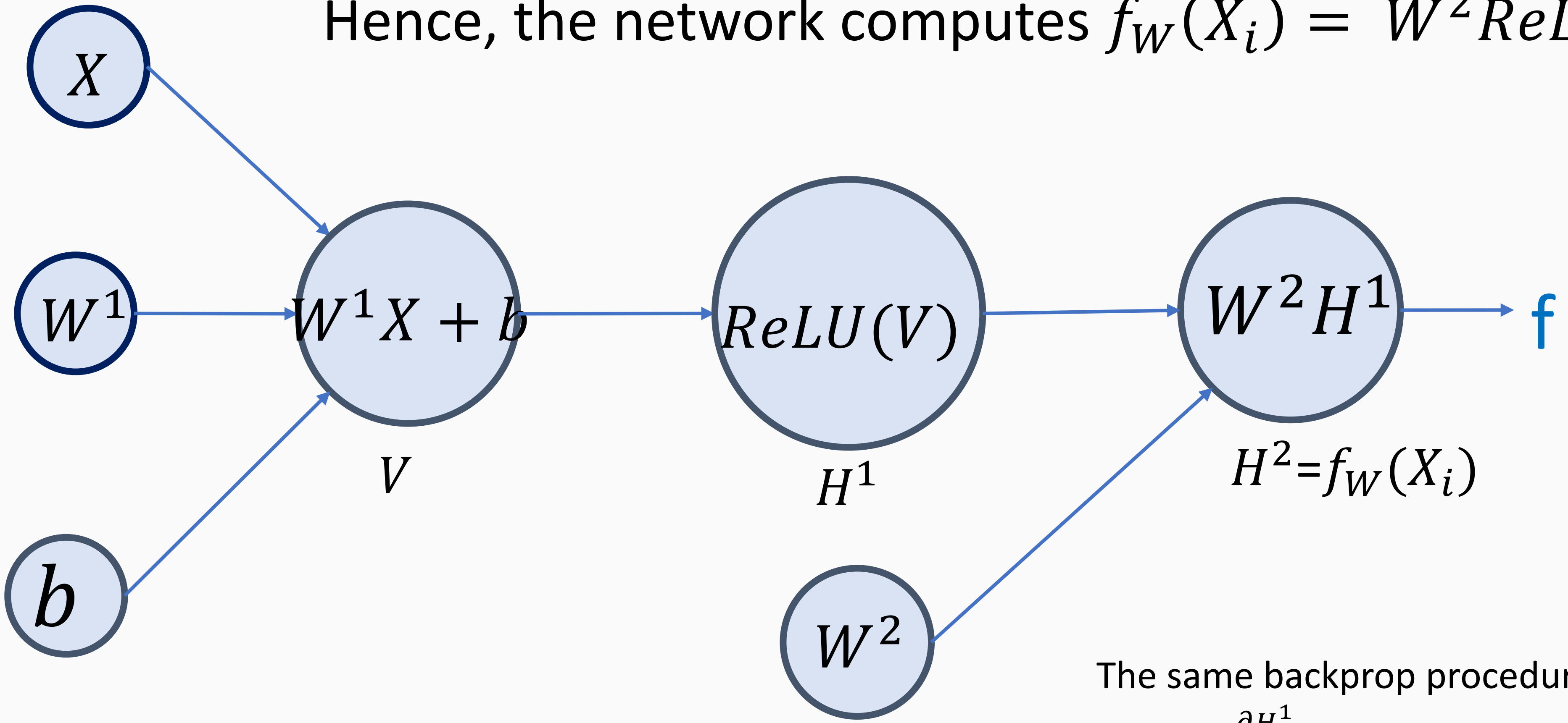


Rectified Linear Units Improve Restricted Boltzmann Machines

Vinod Nair VNAIR@CS.TORONTO.EDU
Geoffrey E. Hinton HINTON@CS.TORONTO.EDU
Department of Computer Science, University of Toronto, Toronto, ON M5S 2G4, Canada

ReLU net

Hence, the network computes $f_W(X_i) = W^2 ReLU(W^1 X_i)$



The same backprop procedure as for sigmoidal net, but $\frac{\partial H_i^1}{\partial V_i} = 1 \; if \; ReLU(V_i) > 0,$

$0 \; otherwise.$

Another loss – Cross entropy loss

$$L(W) = -\sum_{i=1}^{t} Y_i \cdot \log\left(f_W(X_i)\right)$$

Desired output
(usually binary vector
having one element = 1)

Gradient of sigmoidal NN

Observe that the outputs of NN are now interpreted as probabilities ,
in particular it holds that $f_w(X_i) \in [0,1]$ (the loss stays positive),

need to normalize i.e. an additional layer computing $f_w(X_i)$ / $||f_w(X_i)||$,
is required, i.e. normalized output (like *softmax*), but this
<u>requires one additional step in backprop</u>.

Analytic formula for global minima of (linear) NN

Neural Networks and Principal Component Analysis: Learning from Examples Without Local Minima

PIERRE BALDI AND KURT HORNIK*

University of California, San Diego

(*Received* 18 *May* 1988; *revised and accepted* 16 *August* 1988)

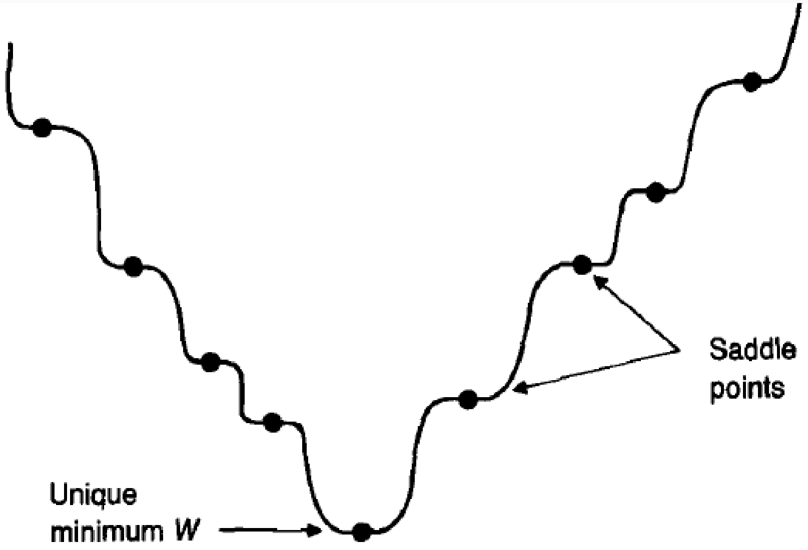


FIGURE 2. The landscape of E.

$$A = U_{\mathcal{J}} C$$

$$B = C^{-1} U'_{\mathcal{J}} \Sigma_{YX} \Sigma_{XX}^{-1}.$$

For such a critical point we have

$$W = P_{U_{\mathcal{J}}} \Sigma_{YX} \Sigma_{XX}^{-1}$$

$$E(A, B) = tr\Sigma_{YY} - \sum_{i \in \mathcal{J}} \lambda_i.$$

$$\Sigma_{YX} = Y^T X,$$
$$\Sigma_{XX} = X^T X,$$

$$U = [u_1, u_2, \ldots, u_p]$$

matrix of p eigenvectors, corresponding to the p largest eigenvalues of

$$\Sigma = \Sigma_{YX} \Sigma_{XX}^{-1} \Sigma_{XY}$$

Sigmoidal vs. ReLU (One layer)

Sigmoidal

Easy to train

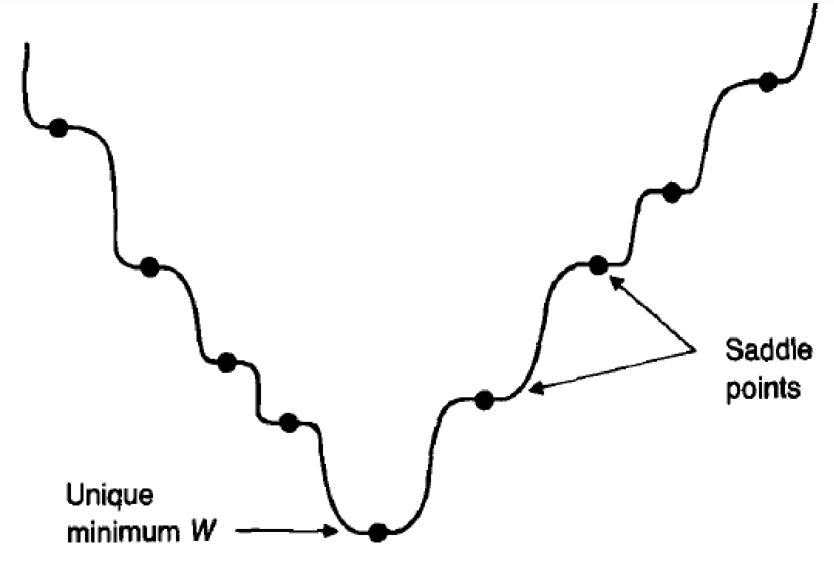
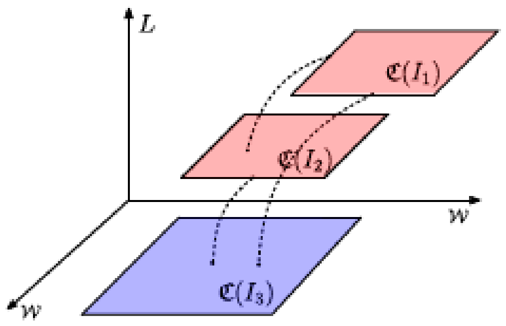
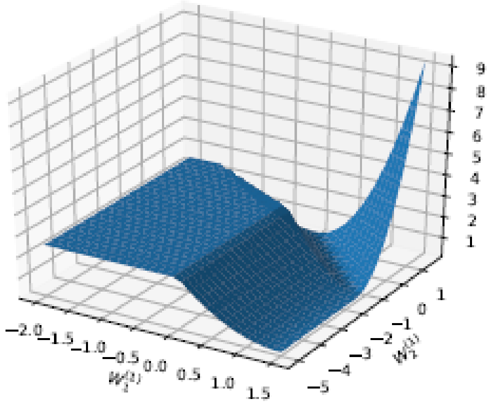


FIGURE 2. The landscape of *E*.

ReLU

Hard to train





Tensors in NumPy

```
import numpy as np
#tensors in NumPy

#how to define a tensor
n = 3
m = 3

A = np.ones( (n, n, n) ) # n,m,k dimensional tensor
B = np.ones( (n, n, n) )

#tensor product
C = np.tensordot(A, B, 0)
print(C.shape)
```

(3,3,3,3,3,3) (its like Cartesian product of two tensors)

```
C = np.tensordot(A, B, 1)
print(C.shape)
```

(3,3,3,3)

$$C_{i,j,l,m} = \sum_{k=1}^{n} A_{i,j,k} B_{k,l,m}$$

```
C = np.tensordot(A, B, 2)
print(C.shape)
```

(3,3)

$$C_{i,m} = \sum_{j=1}^{n} \sum_{k=1}^{n} A_{i,j,k} B_{k,j,m}$$

```
C = np.tensordot(A, B, ([0, 1], [1,2]))
print(C.shape)
```

(3,3)

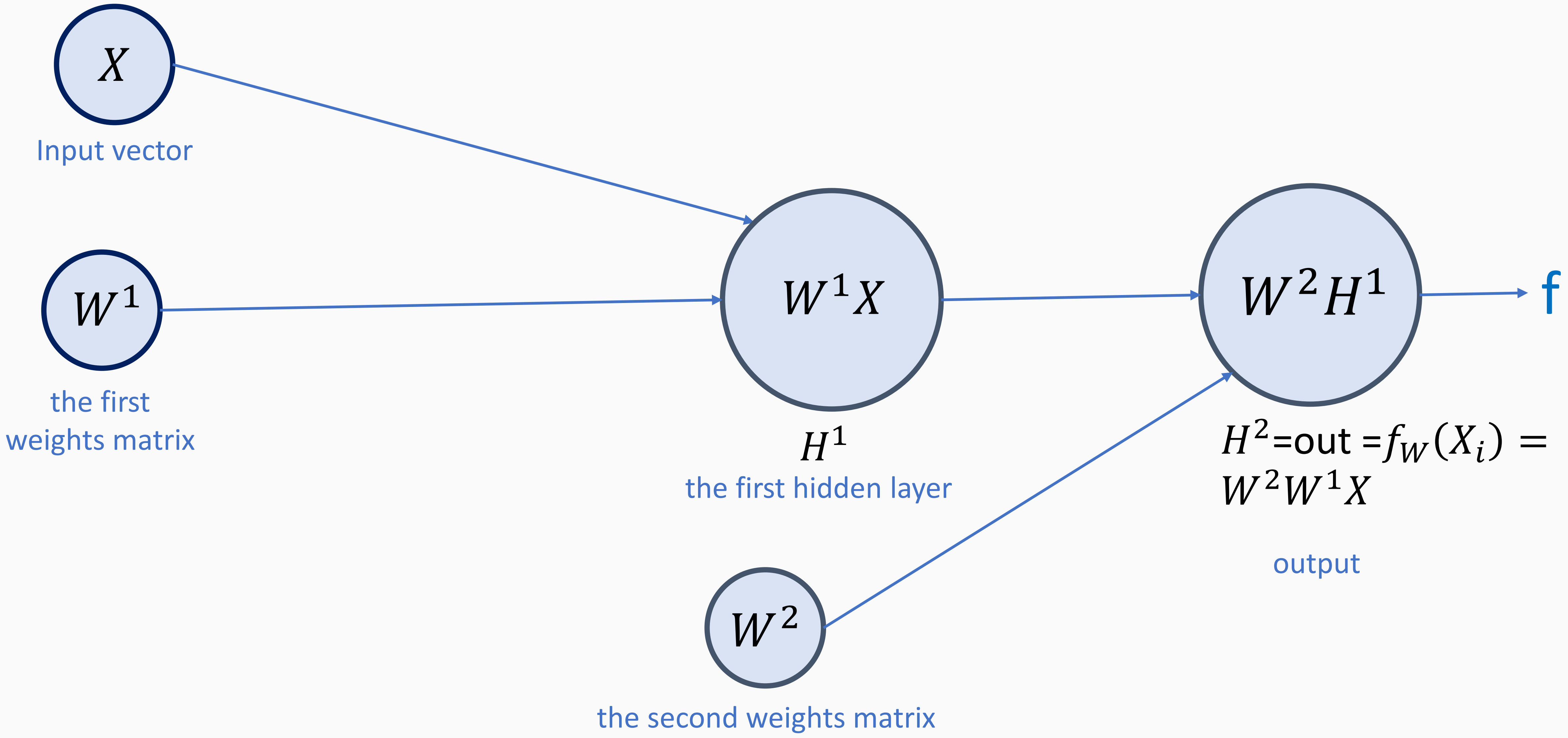
$$C_{i,m} = \sum_{j=1}^{n} \sum_{k=1}^{n} A_{j,k,i} B_{m,j,k}$$

(any combination possible)

Better to use loops to start with instead of tensordot

Remark
For coding tensor operations I suggest using just loops in python instead of tensordot function.

Assignment 8 NN1 architecture



Assignment 8 NN2 architecture