Numerical Analysis for Artificial Intelligence, Week 3

UCSD Summer session II 2018

CSE 190

run by:

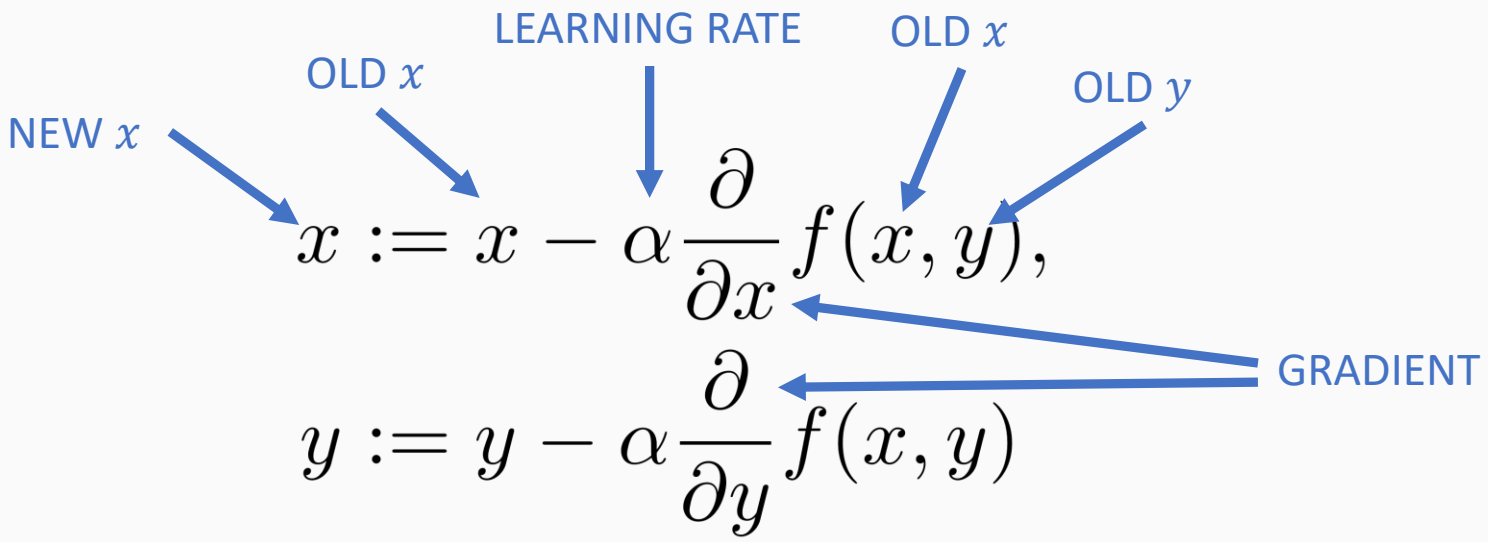
Jacek (*Yatzek)* Cyranka (instructor)

&

Siyang Wang (TA)

Introduction to the gradient descent algorithm

Gradient descent of a 2D function



Update simultaneously x and y.
$\alpha > 0$ – *learning rate* parameter.
Repeat until convergence

Problem when using numerical derivatives , they generate error at each step.
Better to use analytic derivatives of backprop algorithm for computing gradients.

Simultaneous update

$$\texttt{temp0} := x - \alpha \frac{\partial}{\partial x} f(x, y)$$

$$\texttt{temp1} := y - \alpha \frac{\partial}{\partial y} f(x, y)$$

$$x := \texttt{temp0}$$

$$y := \texttt{temp1}$$

INCORRECT !

$$x := x - \alpha \frac{\partial}{\partial x} f(x, y)$$

$$y := y - \alpha \frac{\partial}{\partial y} f(x, y)$$

Another notation for gradient descent

Given a function

$$f : \mathbb{R}^n \to \mathbb{R}.$$

Using vector notation $x \in \mathbb{R}^n$, and x^k for the gradient descent iterations, we have

$$x^{k+1} = x^k - \alpha \nabla f(x^k)$$

Solving simple minimization problems using gradient descent

For a given function f (convex or nonconvex) solve the *minimization problem*, i.e.

$$\min_{x \in \mathbb{R}^n} \{f(x)\},$$

or

$$\arg\min_{x \in \mathbb{R}^n} \{f(x)\}.$$

Gradient descent iterations converge to a critical point, i.e. x, such that

$$\nabla f(x) = 0.$$

Using the math notation

$$x^k \to x^\star, \text{ as } k \to \infty, \text{ and } \nabla f(x^\star) = 0.$$

Hence, if there is a unique critical point (minimum), then g.d. converge to it.

Convex Optimization

Show optimization using gradient descent of a quadratic function [mathematica_demonstrations\ConvergenceOfMinimizationMethods.cdf](mathematica_demonstrations\ConvergenceOfMinimizationMethods.cdf)

1. Given a convex function f
2. Locate the minimizer by following the descent

It follows that there is a unique minimizer

The first convex optimization problem

Minimizing a quadratic function, where Q is a square symmetric positive definite matrix, and b is a vector

$$f(x) = \frac{1}{2}x^T \cdot Q \cdot x + b^T \cdot x$$

Observe that now x denotes a vector $x \in R^n$,
we use interchangeably x as a vector and a number
(x is a number on slide 30)

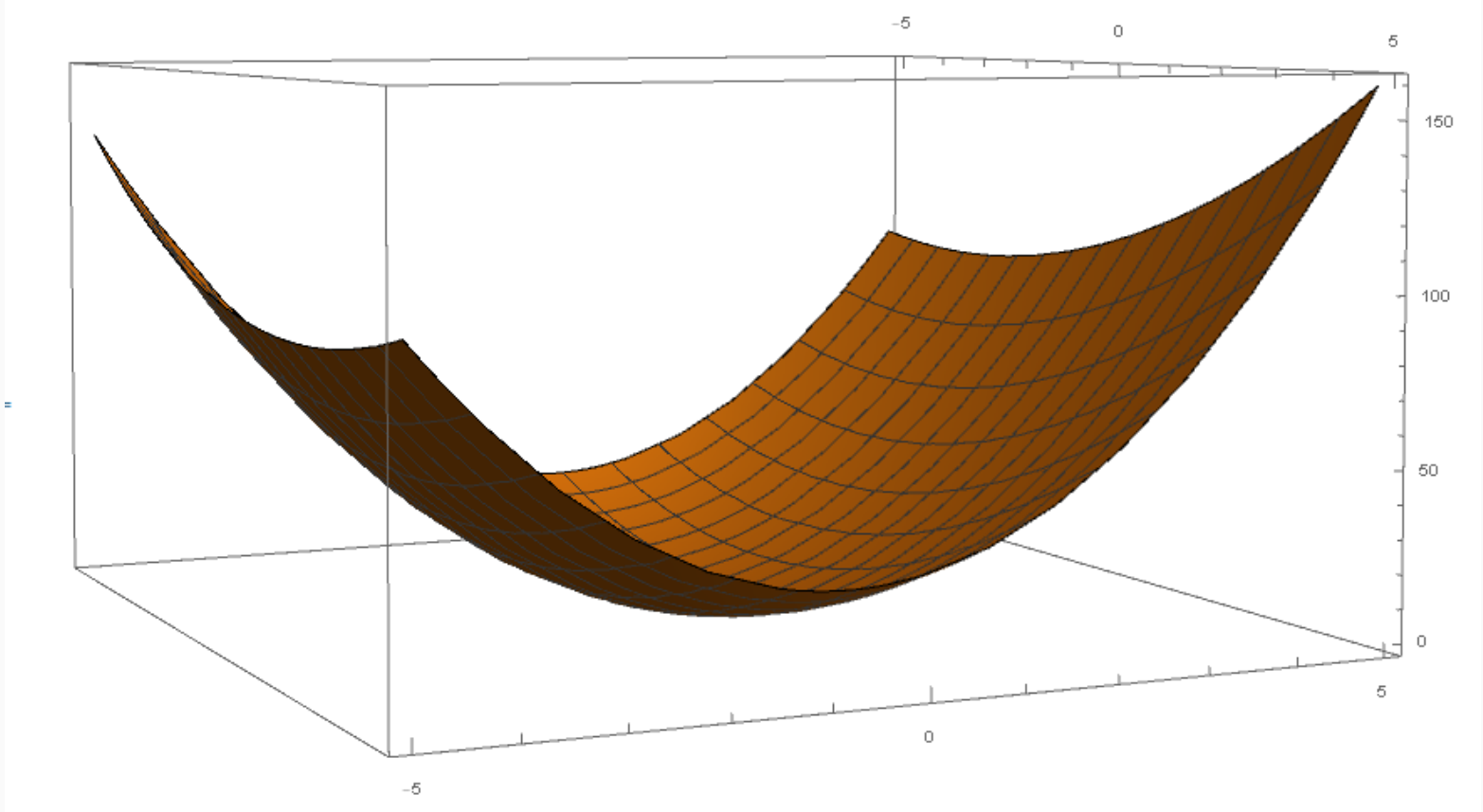
Symmetric matrix satisfies $Q = Q^T$

The minimal point can be computed analytically by the formula $x_{min} = -b^T \cdot Q^{-1}$

The formula for the gradient $\nabla_x f(x)$ is simple $\nabla_x f(x) = x^T Q + b^T$

Example of a quadratic function

$$\text{For example take } Q = \begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & 2 \end{bmatrix}, \text{ and } b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$



Gradient descent for a quadratic function

$$\text{Consider example } Q = \begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & 2 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

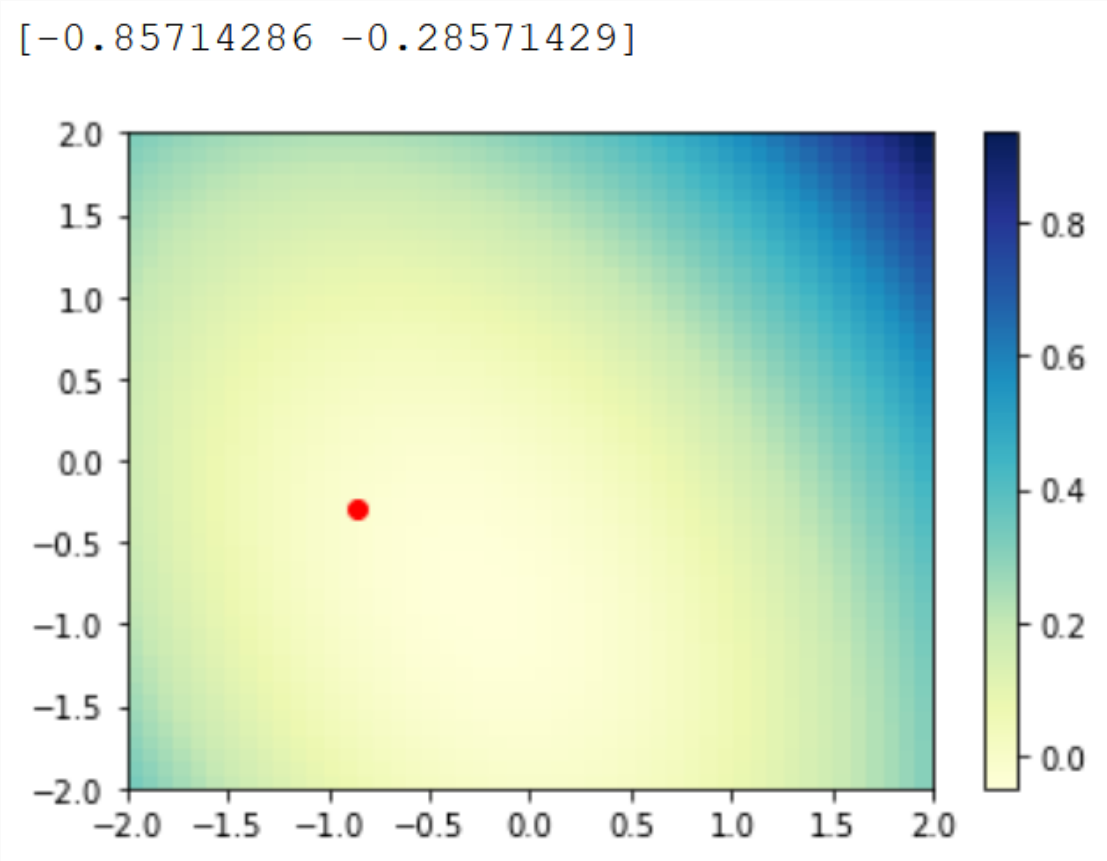
$$x^\star = [-0.85714286, -0.28571429]$$

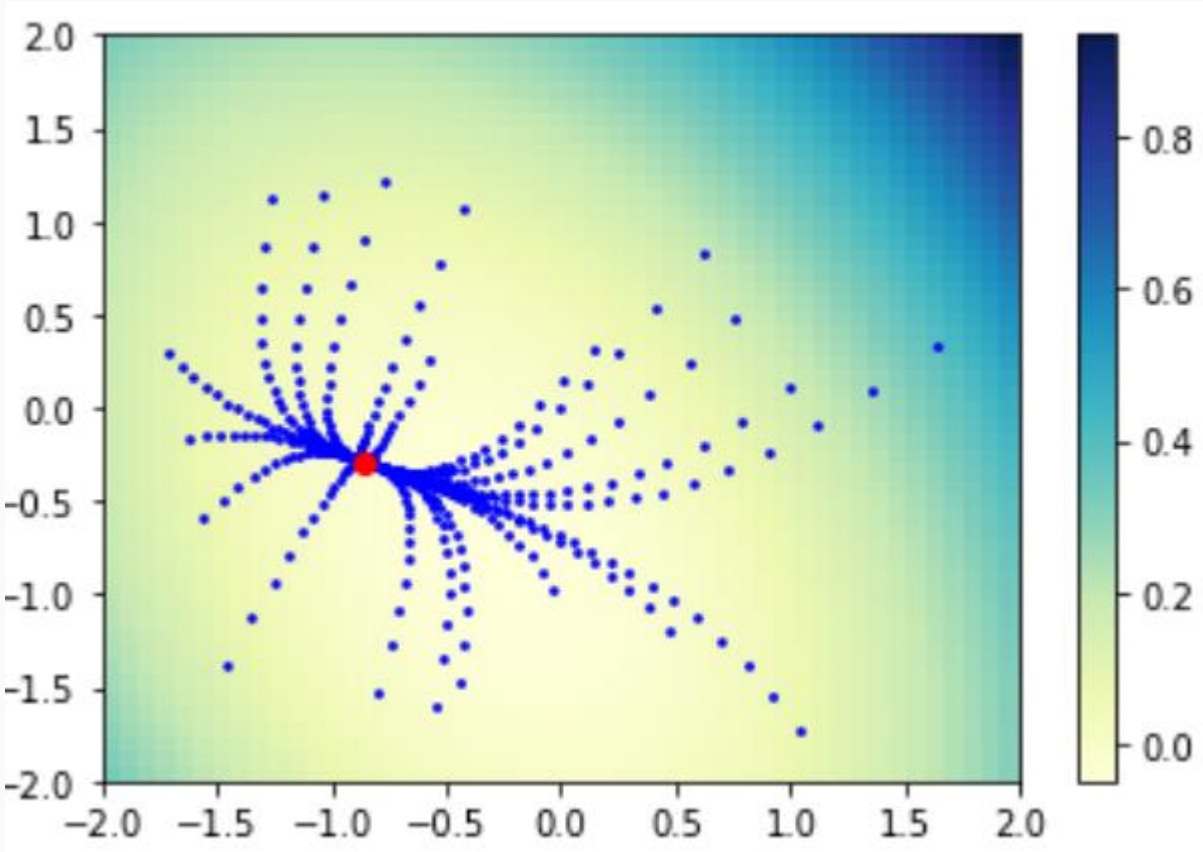
$$f(x) = \frac{1}{2} x^T \cdot Q \cdot x + b^T \cdot x$$

Gradient descent $x := x - \alpha \nabla f(x)$
(compactified notation)
$x \epsilon R^2$ is a two-dimensional vector,
x_t denotes the t-th iteration.

See example computations and plots in jupyter_notebooks\week2_2.ipynb

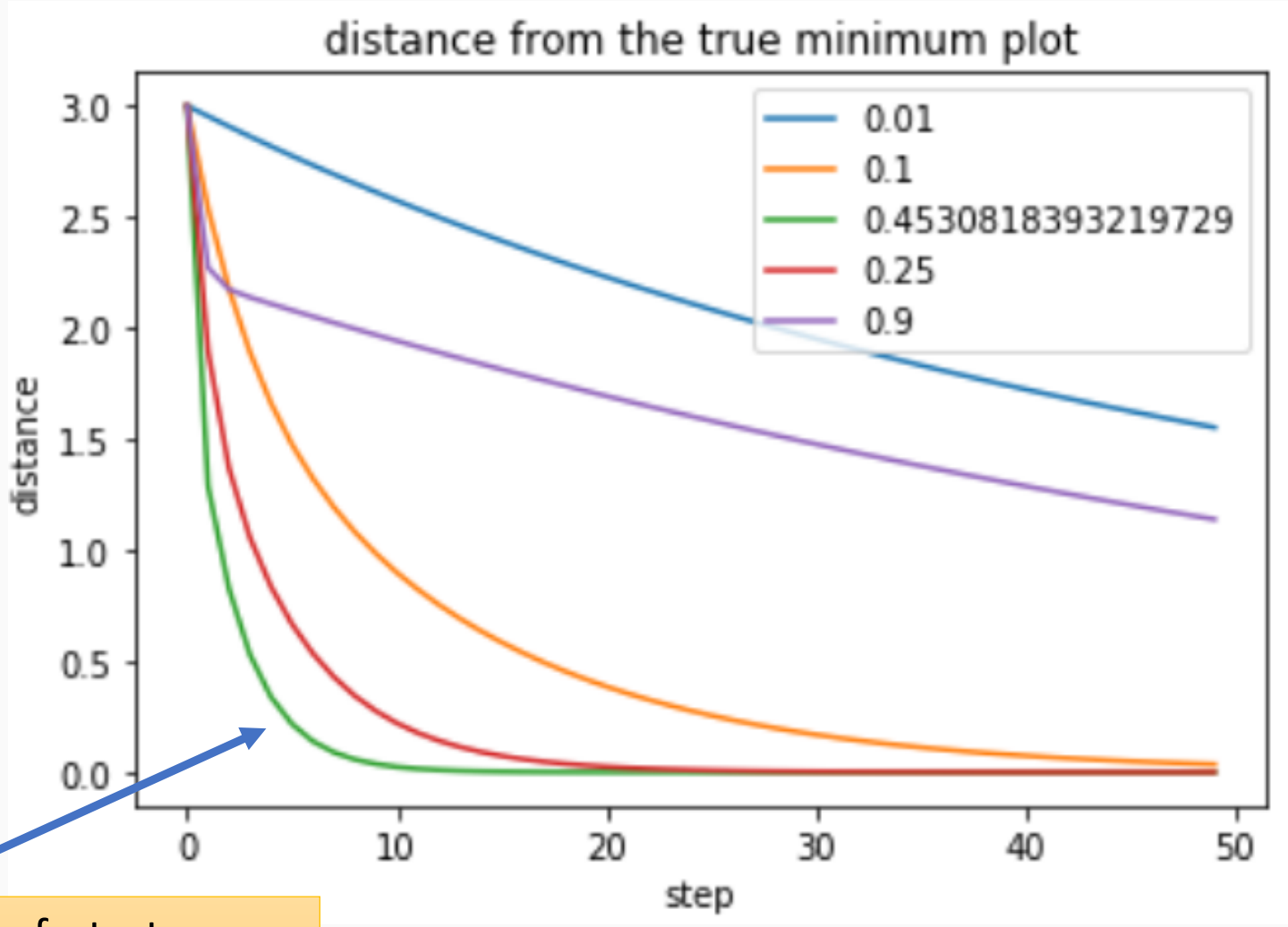
Gradient descent for the quadratic function





Behavior for different learning rates

the plot shows
$dist = ||x^* - x_t||$



We observe the fastest convergence for $\alpha = 0.453$ … **Why ?**

A Glimpse at gradient descent convergence theory

Let $L > 0$ be the *Lipschitz constant* of the gradient, i.e.

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\| \text{ for all } x, y \in \mathbb{R}^d.$$

The following theorem holds (informal statement below)

Theorem 1. *Let $x^\star$ be a global minimum of f. ∇f is* Lipschitz *with a constant $L > 0$ (see above). Choosing*

$$\alpha = \frac{1}{L}.$$

Then, gradient descent *with any x_0 satisfies*

1. *Function values are monotone decreasing*

$$f(x_{t+1}) \leq f(x_t) - \frac{1}{2L}\|\nabla f(x_t)\|^2$$

2.

$$\|f(x_T) - f(x^\star)\| \leq \frac{L}{2T}\|x_0 - x^\star\|^2, \quad T > 0.$$

Remark on the Theorem

The Theorem provides us with a good guess for the learning rate.
Informally, it guarantees that for the given choice of learning rate the function value along the gradient descent path is decreasing at least like $\frac{C}{T}$.

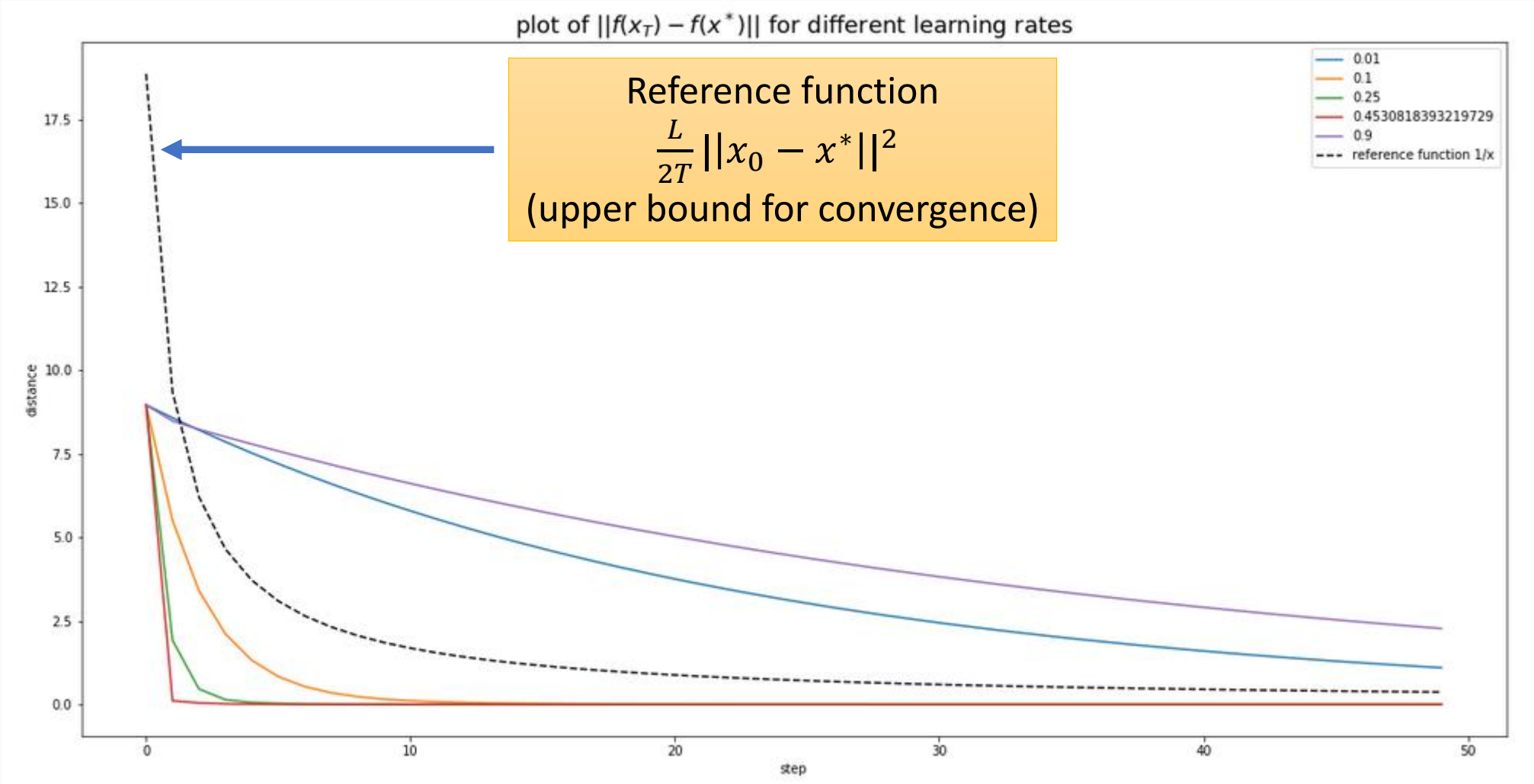
It serves as an example of mathematical statement about gradient descent applied to convex problems.

However, for most practical problems Theorem cannot be applied literally, it serves as a guide exclusively

- There is no global Lipschitz constant, only local, so the learning rate is usually being adapted from step to step,
- Lipschitz constant is computationally expensive to estimate,
- Practical optimization problems are nonconvex.

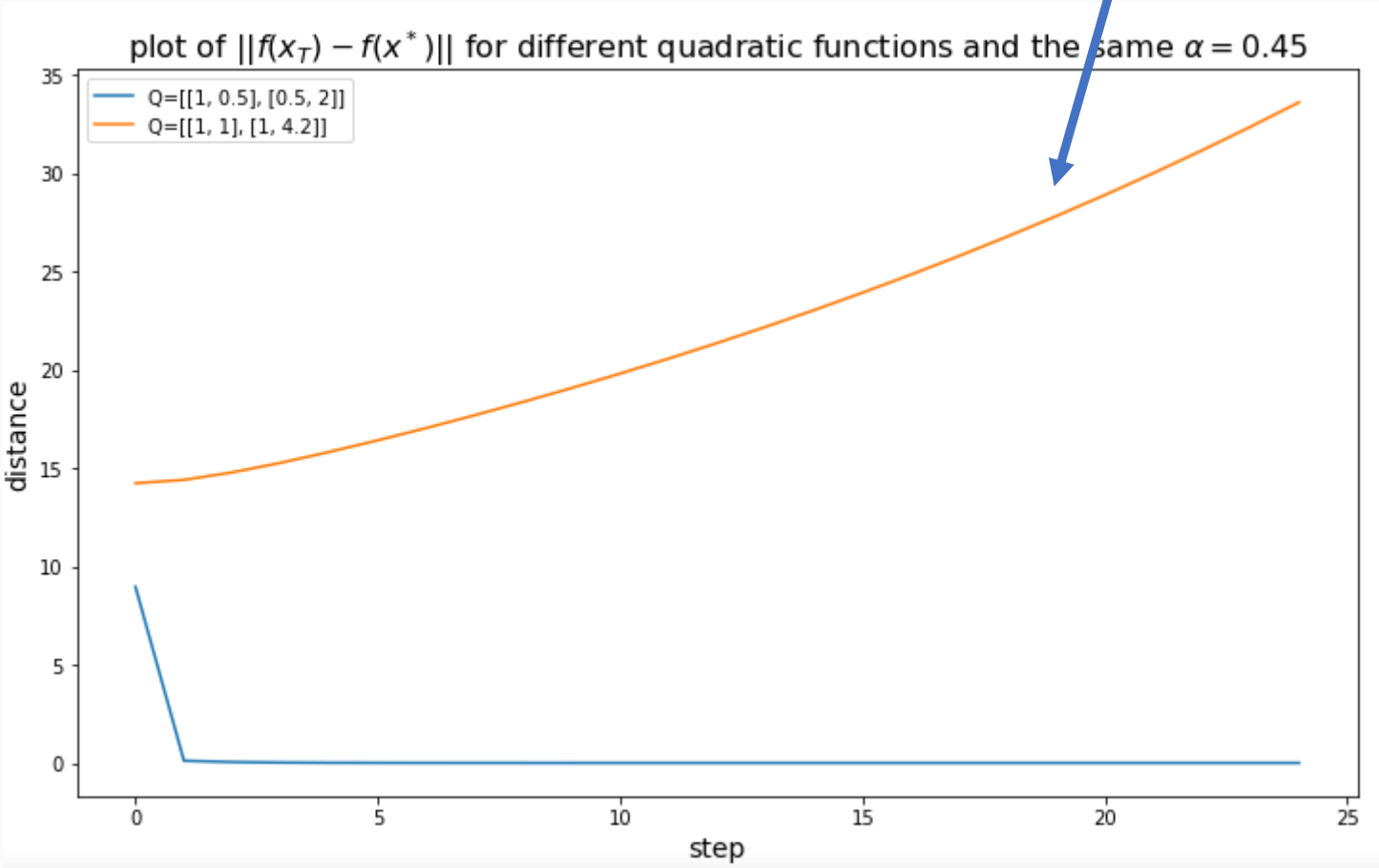
Meaning of Theorem for practice

$$\|f(x_T) - f(x^\star)\| \leq \frac{L}{2T}\|x_0 - x^\star\|^2, \quad T > 0.$$



Meaning of Theorem for practice II

Do not have convergence for a slightly different quadratic eq. with Q=[[1, 1], [1, 4.2]] for the same learning rate. **Why?**



Meaning of Theorem for practice III

```
print("max eigenvalue of [[1, 0.5], [0.5, 2]]=%f" % np.max(np.linalg.eigvals(Q)))
print("max eigenvalue of [[1, 1], [1, 4.2]]=%f" % np.max(np.linalg.eigvals(Q2)))
```

```
max eigenvalue of [[1, 0.5], [0.5, 2]]=2.207107
max eigenvalue of [[1, 1], [1, 4.2]]=4.486796
```

$$\text{The max eigenvalue of } Q' = \begin{bmatrix} 1 & 1 \\ 1 & 4.2 \end{bmatrix}$$

$$\text{is larger than the max eigenvalue of } Q = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 2 \end{bmatrix}$$

$$\downarrow$$

The learning rate for quadratic equation with Q',
should be smaller compared to the learning rate for quadratic with Q.

When do we have to use GD to solve a quadratic problem?

The minimum of a quadratic problem can be computed directly by the formula $x_{min} = -b^T \cdot Q^{-1}$

Question: is there any application when we need to solve a *quadratic equation* using *gradient descent*?

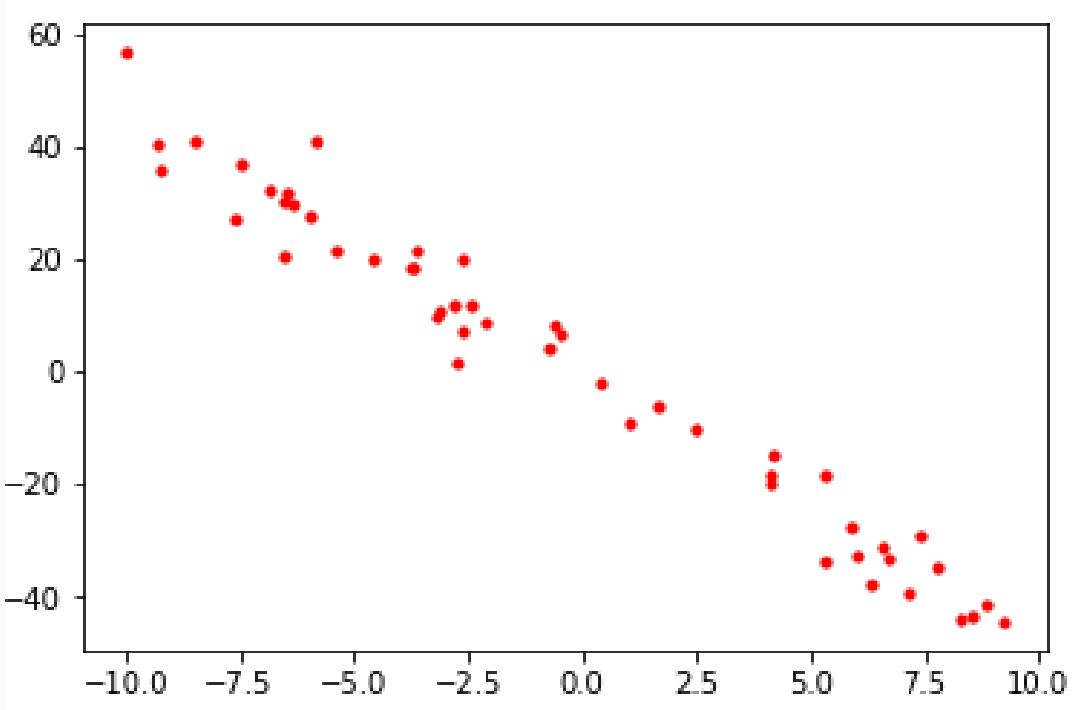
Answer: yes. When dealing with large scales.

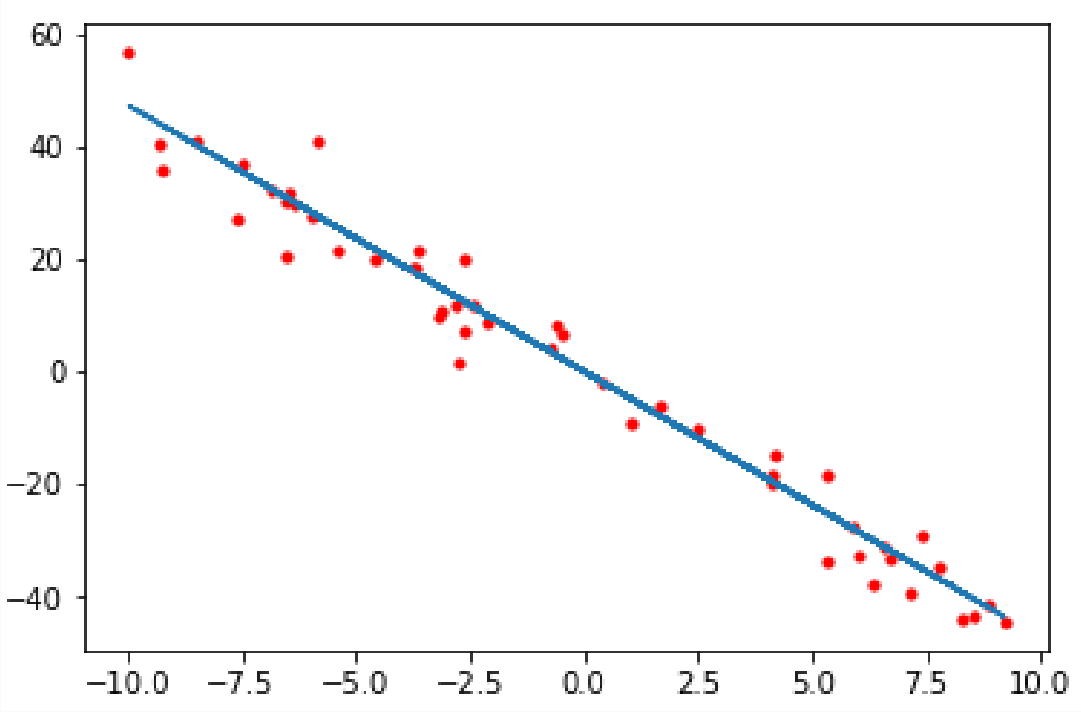
Large scale quadratic function example

Show the large scale quadratic function in week3_1.ipynb

Linear Regression

Statement of the *linear regression problem*:
given data , find a linear function which is the best approximator





Given a set of N data-points $\{(x_i, y_i)\}$

Best approximator = the line $mx + b$ that minimizes $\frac{1}{N}\sum(y_i - (mx_i + b))^2$

Linear regression using gradient descent

The task is

Find $\boldsymbol{m}, \boldsymbol{b}$ that minimize the '*mean square error*' $MSE = \frac{1}{N}\sum(y_i - (mx_i + b))^2$

1. Compute the gradient

$$\frac{\partial MSE}{\partial m} = \frac{2}{N}\sum_{i=1}^{N} -x_i(y_i - (mx_i + b)),$$

$$\frac{\partial MSE}{\partial b} = \frac{2}{N}\sum_{i=1}^{N} -(y_i - (mx_i + b)),$$

It can be written using vectors (NumPy way)

$$\frac{\partial MSE}{\partial m} = \frac{2}{N}\left(-x^T y + mx^T x + x^T \hat{b}\right),$$

$$\frac{\partial MSE}{\partial b} = \frac{2}{N}(-1^T y + m1^T x + 1^T \hat{b}),$$

where $1 = [1, 1, \ldots, 1]$, i.e. is a vector of ones, and $\hat{b} = [b, b, \ldots, b]$ is a vector of b's.

Linear regression using gradient descent the Lipschitz const

$$\frac{\partial MSE}{\partial m} = \frac{2}{N}\left(-x^T y + m x^T x + x^T \hat{b}\right),$$
$$\frac{\partial MSE}{\partial b} = \frac{2}{N}(-1^T y + m 1^T x + 1^T \hat{b}),$$

$$\nabla MSE(m,b) = \frac{2}{N}\left[-x^T y + m x^T x + x^T \hat{b},\ -1^T y + m 1^T x + 1^T \hat{b}\right],$$

$$\|\nabla MSE(m,b) - \nabla MSE(m',b')\| \leq \frac{2}{N}\left\|\left[x^T x(m-m') + x^T 1(b-b'),\ 1^T x(m-m') + 1^T 1(b-b')\right]\right\| \leq$$
$$\frac{2}{N}\left\|\begin{bmatrix} x^T x & x^T 1 \\ 1^T x & 1^T 1 \end{bmatrix}\begin{bmatrix} m-m' \\ b-b' \end{bmatrix}\right\| \leq \frac{2}{N}\left\|\begin{bmatrix} x^T x & x^T 1 \\ 1^T x & 1^T 1 \end{bmatrix}\right\| \left\|\begin{bmatrix} m-m' & b-b' \end{bmatrix}\right\|$$

Remark 1. *The Lipschitz constant of the gradient of the linear regression problem is bounded by*

$$\frac{2}{N}\left\|\begin{bmatrix} x^T x & x^T 1 \\ 1^T x & 1^T 1 \end{bmatrix}\right\| \quad \textit{(the spectral matrix norm)}$$

Intuitively the problem gets harder (the learning rate needs to be decreased) if x's are of larger magnitude.

Nonlinear polynomial regression cont.

Nonlinear regression for 2D datapoints

Using the compact polynomial notation

$m_1 x_i^1 + m_2 x_i^2 + \cdots + m_p x_i^p + b = \sum_{j=1}^{p} m_j x_i^j + b$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \left(\sum_{j=1}^{p} m_j x_i^j + b\right)\right)^2$$

Solving nonlinear regression using gradient descent

We can solve nonlinear regression using gradient descent like we solved linear regression

For the quadratic regression we have the following gradients

$$\frac{\partial MSE}{\partial m_1} = \frac{2}{N}\sum_{i=1}^{N} -x_i(y_i - (m_1 x_i + m_2 x_i^2 + b)),$$

$$\frac{\partial MSE}{\partial m_2} = \frac{2}{N}\sum_{i=1}^{N} -x_i^2(y_i - (m_1 x_i + m_2 x_i^2 + b)),$$

$$\frac{\partial MSE}{\partial b} = \frac{2}{N}\sum_{i=1}^{N} -(y_i - (m_1 x_i + m_2 x_i^2 + b)),$$

And perform simultaneous update (like on slide 41) of the three parameters m_1, m_2, b

Nonlinear regression of a general polynomial

We can solve nonlinear regression for a arbitrary polynomial

For the quadratic regression we have the following gradients

If doing nonlinear regression with a p-th order polynomial, then there are $p+1$ parameters for gradient descent (can store them in a vector).

$$\frac{\partial MSE}{\partial m_1} = \frac{2}{N}\sum_{i=1}^{N} -x_i \left(y_i - \left(\sum_{j=1}^{p} m_j x_i^j + b \right) \right),$$

$$\frac{\partial MSE}{\partial m_2} = \frac{2}{N}\sum_{i=1}^{N} -x_i^2 \left(y_i - \left(\sum_{j=1}^{p} m_j x_i^j + b \right) \right),$$

$$\cdots$$

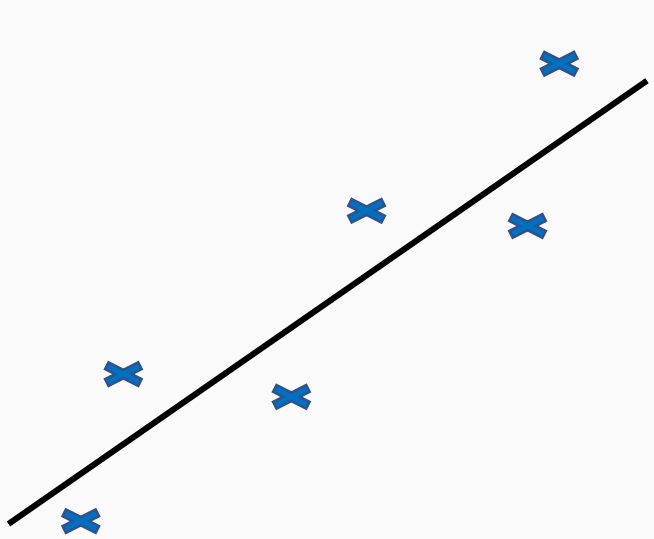
$$\frac{\partial MSE}{\partial m_j} = \frac{2}{N}\sum_{i=1}^{N} -x_i^j \left(y_i - \left(\sum_{j=1}^{p} m_j x_i^j + b \right) \right),$$

$$\cdots$$

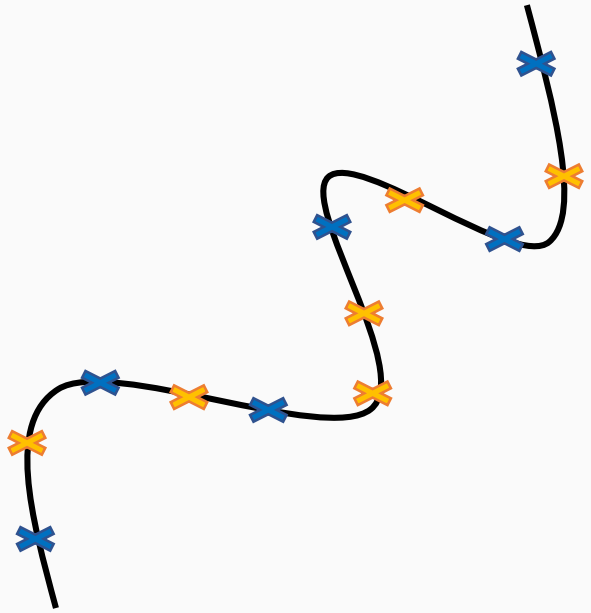
$$\frac{\partial MSE}{\partial b} = \frac{2}{N}\sum_{i=1}^{N} -(y_i - (\sum_{j=1}^{p} m_j x_i^j + b)),$$

Overfitting / Underfitting

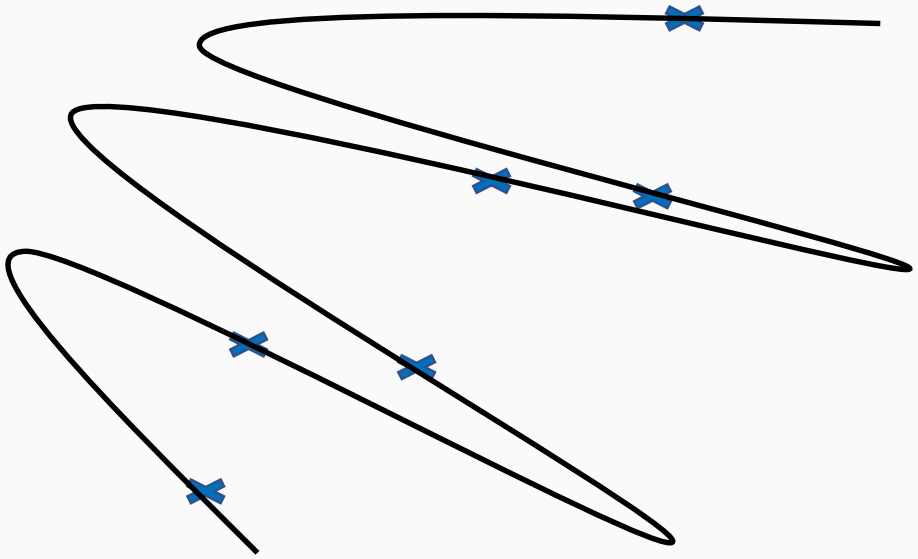
There is no a general recipe of choosing appropriate polynomials for nonlinear regression



Underfitted
(order too low)



Good generalization
order fits data well
Test datapoints



Overfitted
(order too large)

Accelerated gradient descent (Momentum method)

Algorithm 1 Accelerated scheme I

Require: $x^0 \in \mathbf{R}^n$, $y^0 = x^0$, $\theta_0 = 1$ and $q \in [0, 1]$

1: **for** $k = 0, 1, \ldots$ **do**
2: $\quad x^{k+1} = y^k - t_k \nabla f(y^k)$
3: $\quad \theta_{k+1}$ solves $\theta_{k+1}^2 = (1 - \theta_{k+1})\theta_k^2 + q\theta_{k+1}$
4: $\quad \beta_{k+1} = \theta_k(1 - \theta_k)/(\theta_k^2 + \theta_{k+1})$
5: $\quad y^{k+1} = x^{k+1} + \beta_{k+1}(x^{k+1} - x^k)$
6: **end for**

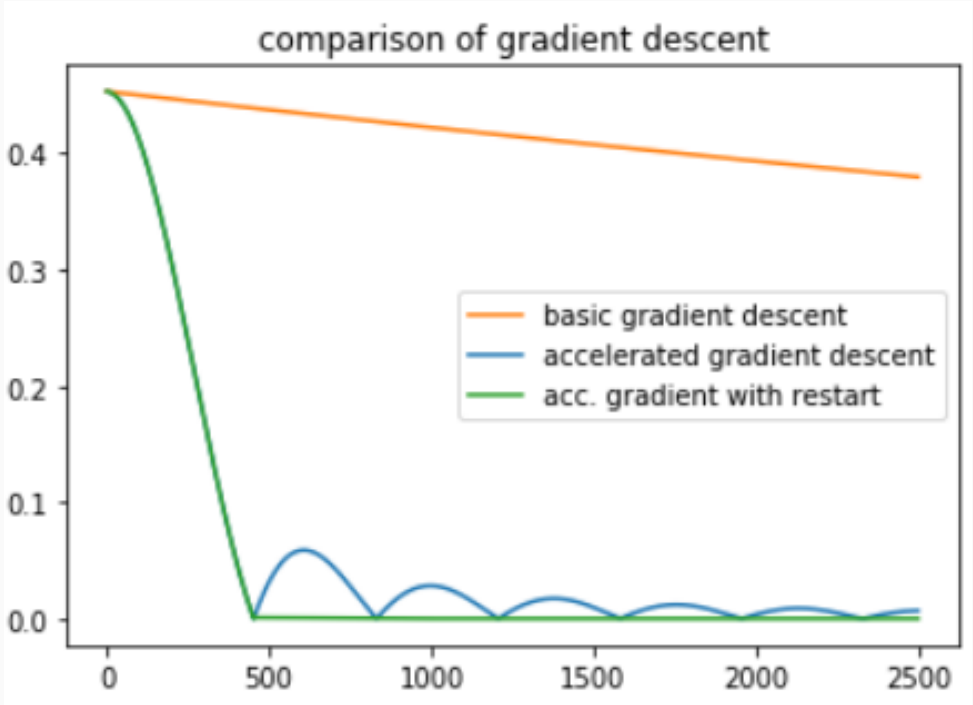
Theoretically proved to converge with twice the speed of the regular gradient descent!!!

Source:

- Yurii Nesterov, *Introductory Lectures on Convex Optimization,* Springer, 2003
- Y. Nesterov. *A method of solving a convex programming problem with convergence rate O(1/k2)*. Soviet Mathematics Doklady, 27(2):372–376, 1983.

Comparison of regular and accelerated gradient descents

Comparison of regular gradient descent with accelerated gradient descent for minimizing a quadratic equation



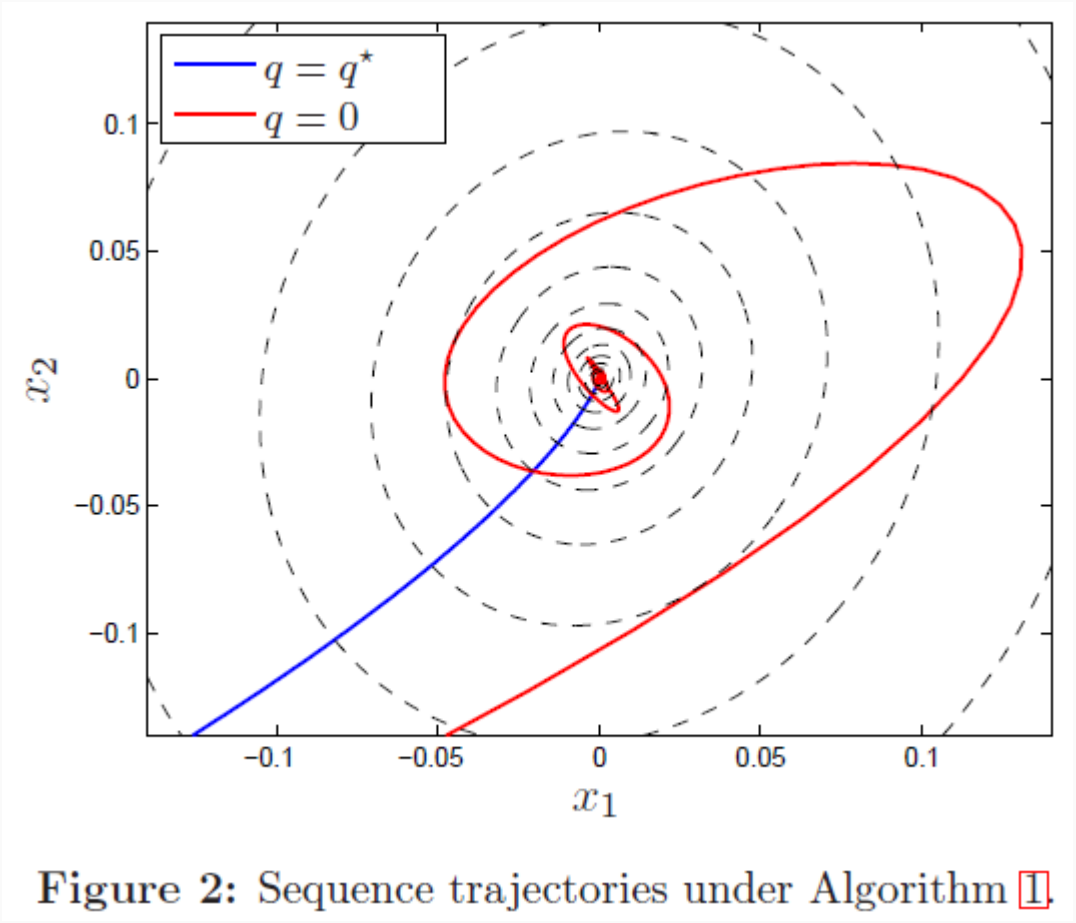


Figure 2: Sequence trajectories under Algorithm 1.

Source: Brendan O'Donoghue, Emmanuel Candes, *Adaptive Restart for Accelerated Gradient Schemes,* arXiv:1204.3982v1

Another way of accelerating – Newton method

Gradient descent

$$x_{k+1} = x_k - \alpha \nabla f(x_k)$$

Newton method, there is no learning rate – instead use inverse of the Hessian matrix

$$x_{k+1} = x_k - (\nabla^2 f(x_k))^{-1} \nabla f(x_k)$$

HESSIAN MATRIX

GRADIENT (COLUMN VEC)

Like a matrix learning rate that changes at each step

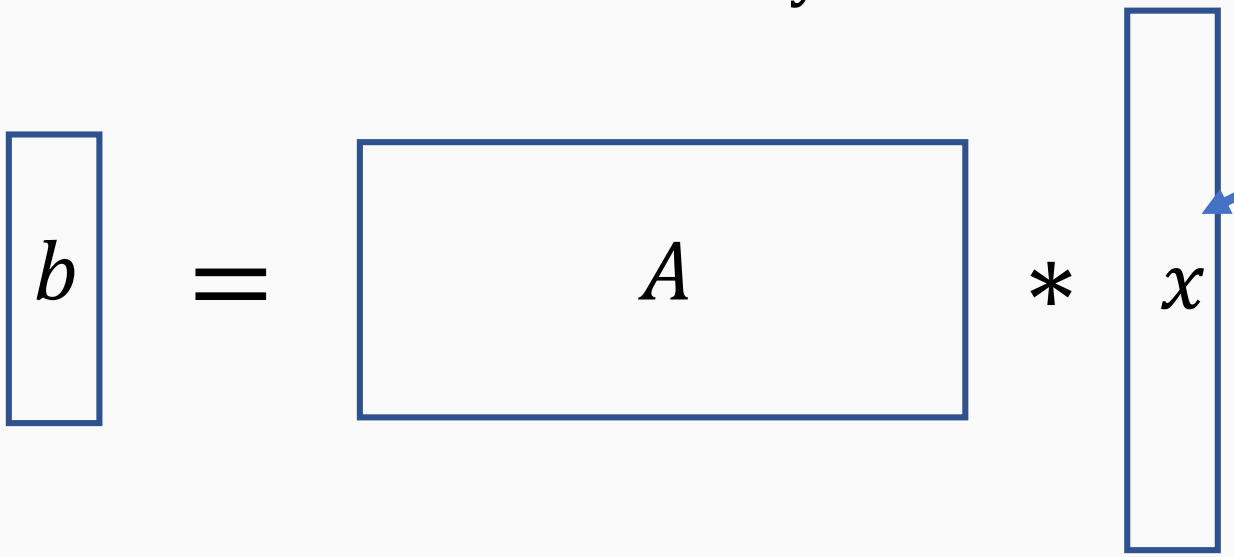
$$\alpha_k = (\nabla^2 f(x_k))^{-1}$$

For minimizing of the quadratic equation

Applying the Newton method = solving the problem analytically
(show it)

Sparse Linear regression using gradient descent

Underdetermined system of linear equations $Dx = y$



FIND x

$x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$,
$A \in \mathbb{R}^{m \times n}$ and $n \gg m$.

FIND A VECTOR x

$$\text{Minimize } \frac{1}{2}\|Ax - b\|_2^2 + \rho\|x\|_1, \text{ where}$$

$$\|Ax - b\|_2^2 = \sum_{i=1}^{m} ((Ax)_i - b_i)^2,$$
$$\|x\|_1 = \sum_{i=1}^{n} |x_i|.$$

SATISFYING THE LINEAR EQ.

DAMPING PARAMETER

NORM OF x IS SMALL

Algorithm for solving sparse linear regression

ISTA 'iterative soft-threshold algorithm (ISTA)'
A slightly modified gradient descent scheme

Algorithm 4 ISTA

Require: $x^{(0)} \in \mathbf{R}^n$

1: **for** $k = 0, 1, \ldots$ **do**
2: $\quad x^{k+1} = \mathcal{T}_{\rho t}(x^k - tA^T(Ax^k - b))$.
3: **end for**

$$t = \alpha$$

Soft thresholding operator

$$\mathcal{T}_{\rho t}(x_i) = \text{sign}(x_i)\max\left(|x_i| - \rho t, 0\right) \text{ for all } i = 1, \ldots, n \text{ (size of } x \text{ is } n),$$

which means iterate the whole vector x and update each of its components using this rule.

Sources: I. Daubechies, M. Defrise, and C. De Mol. *An iterative thresholding algorithm for linear inverse problems with a sparsity constraint*. Communications on Pure and Applied Mathematics, 57(11):1413–1457, November 2004.
D. Donoho. *Compressed sensing*. IEEE Transactions on Information Theory, 52(4):1289–1306, April 2006.
E. Candes and M. Wakin. *An introduction to compressive sampling*. Signal Processing Magazine, IEEE, 25(2):21–30, March 2008.

Gradient of the sparse linear regression operator

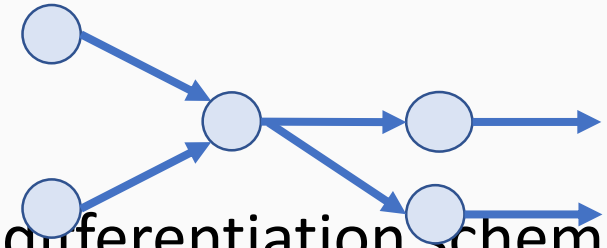
$$\nabla_x \frac{1}{2}\|Ax - b\|_2^2 = A^T(Ax - b),$$

$$\nabla_x \|x\|_1 = \operatorname{sign} x \text{ if } x \neq 0.$$

TOPIC: BACKPROPAGATION

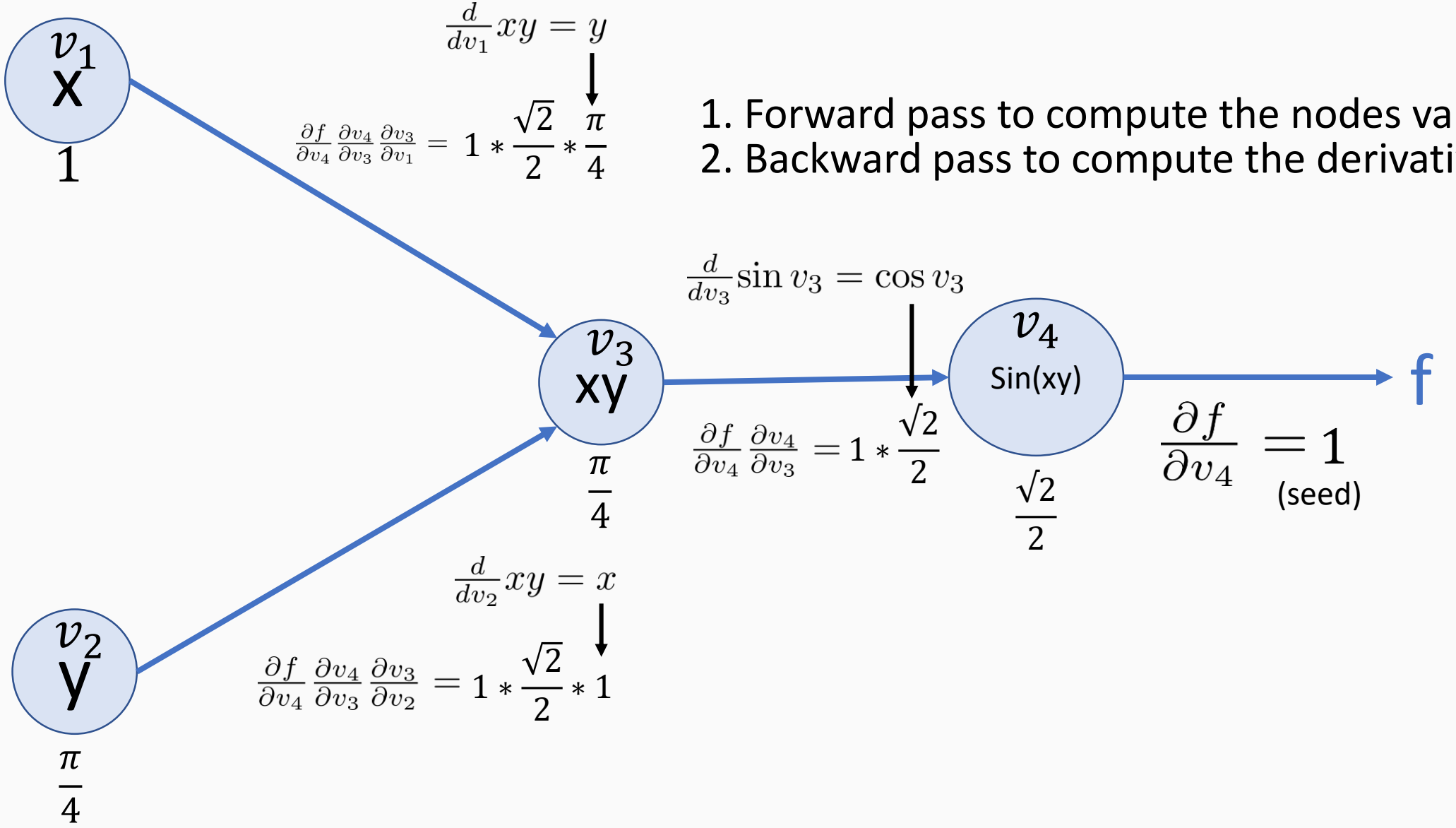
Computing derivatives of any graph using backprop

- Backprop algorithm is used for computing gradients of any *directed acyclic graph*



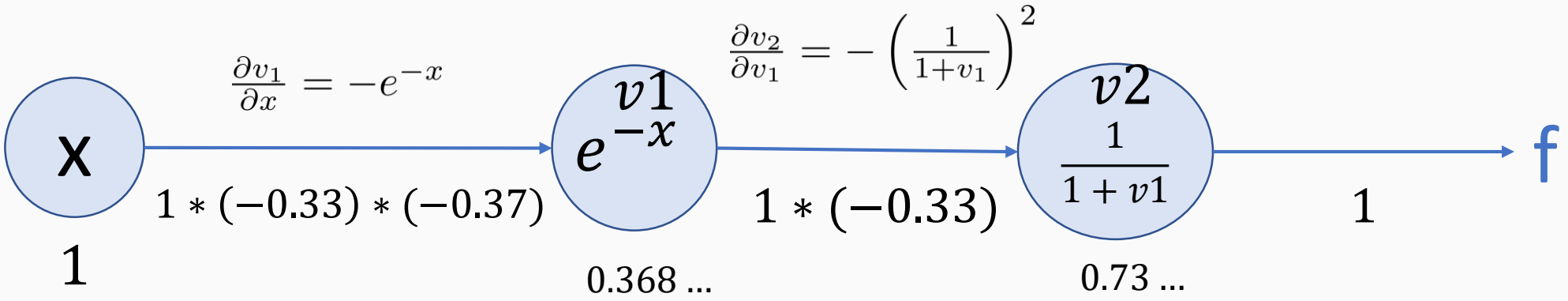
- It is numerical differentiation scheme. The outputs from backprop are the values of partial derivatives. This can used as inputs to gradient descent algorithm,
- *Backprop* itself is not performing learning of a neural net, the learning is done by gradient descent, backprop computes gradients,
- It is very efficient algorithm avoiding any unnecessary repetitions, the complexity is $O(\#\ of\ graph\ nodes)^2$,
- There are many ways of computing gradients of complicated graphs, finding the best way is a hard problem (NP-complete) computationally,

Backprop of f(x,y)=sin$(x * y)$



1. Forward pass to compute the nodes values.
2. Backward pass to compute the derivatives.

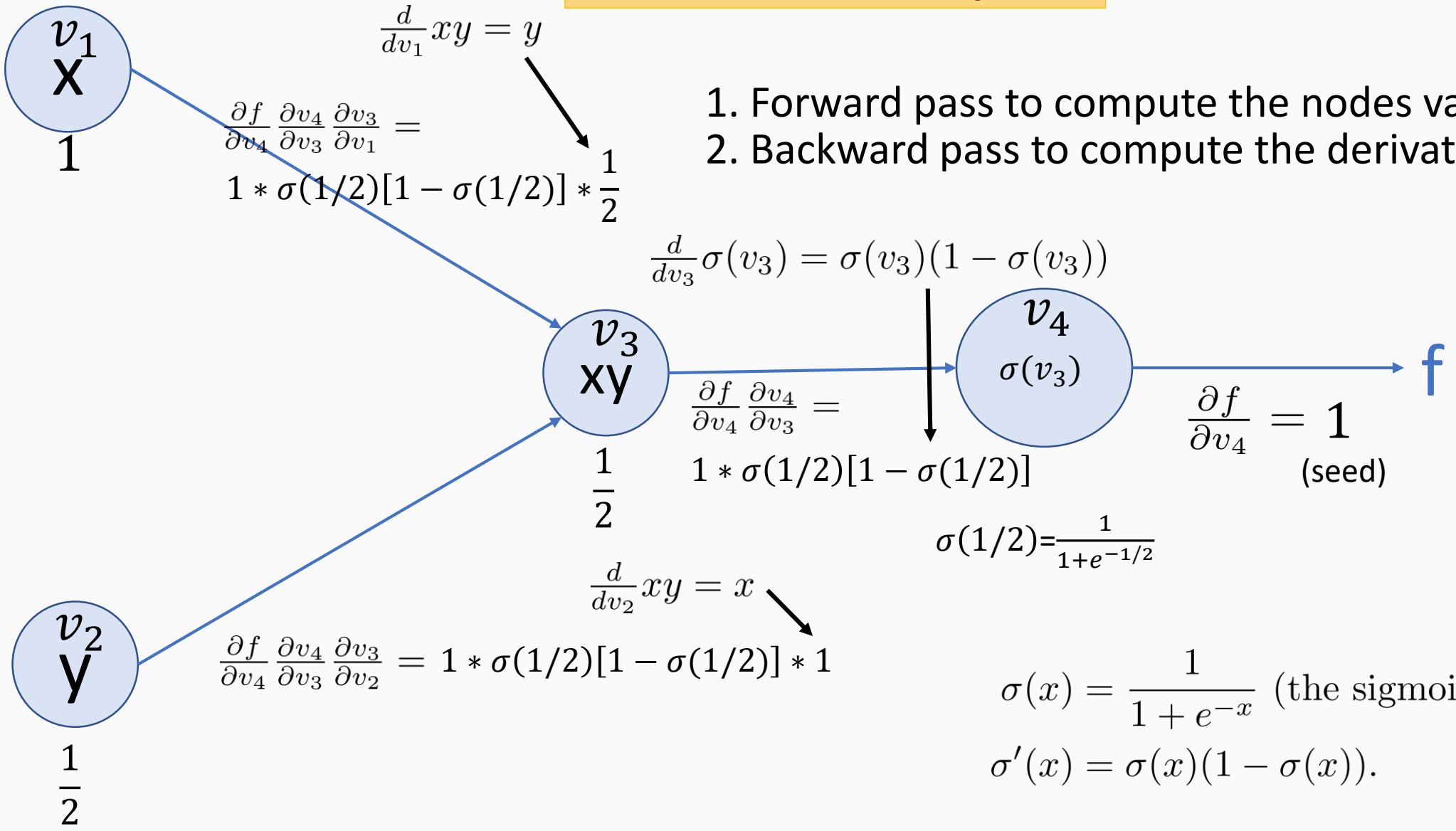
Backprop of f(x,y)=$\frac{1}{1+e^{-x}}$



But in fact can use a single node for the sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}, \quad \sigma'(x) = \sigma(x)\,(1-\sigma(x))$$

Backprop of f(x,y)=$\frac{1}{1+e^{-x*y}}$

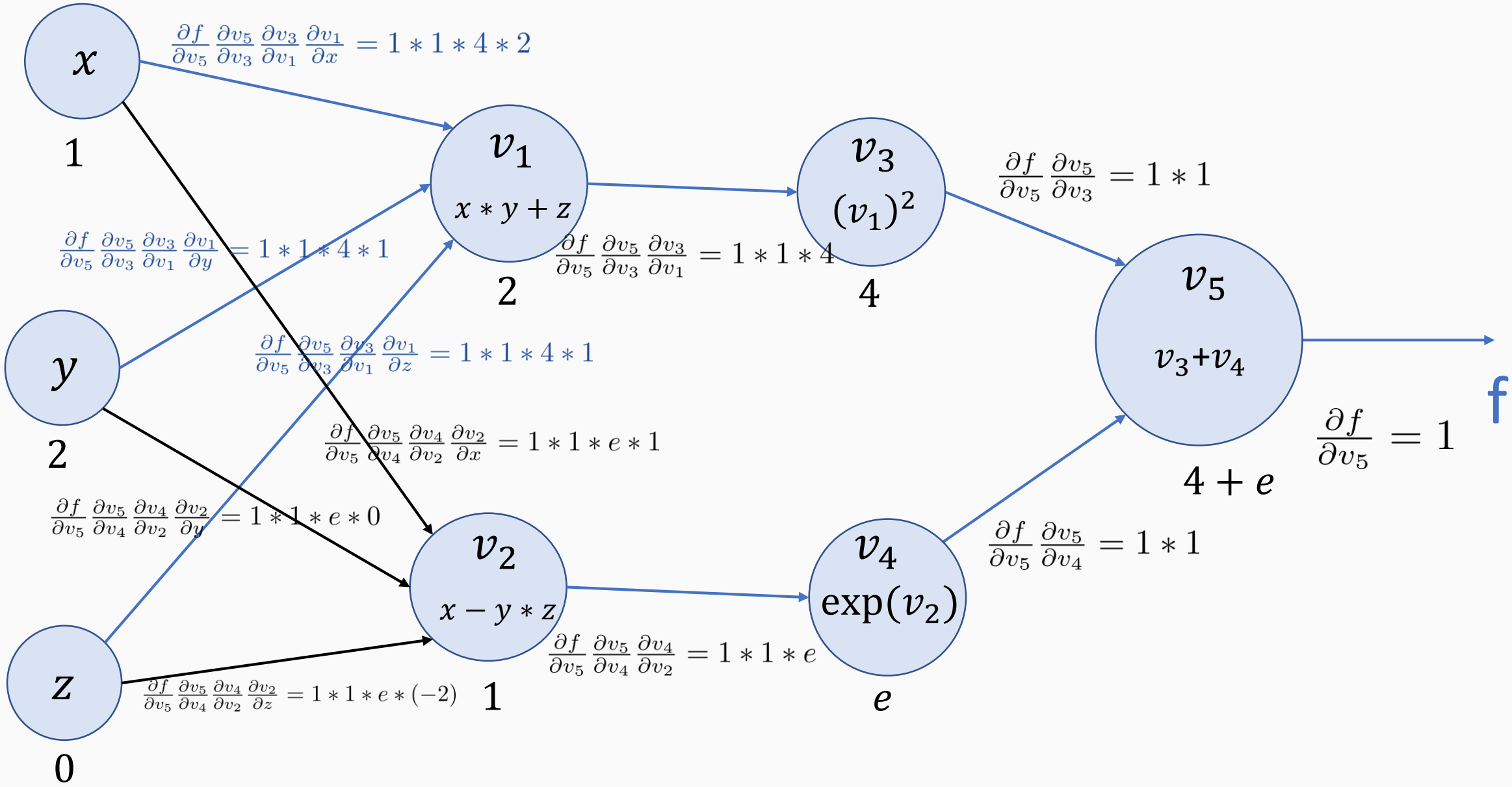


1. Forward pass to compute the nodes values.
2. Backward pass to compute the derivatives.

$$\sigma(x) = \frac{1}{1+e^{-x}} \text{ (the sigmoid)},$$
$$\sigma'(x) = \sigma(x)(1-\sigma(x)).$$

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

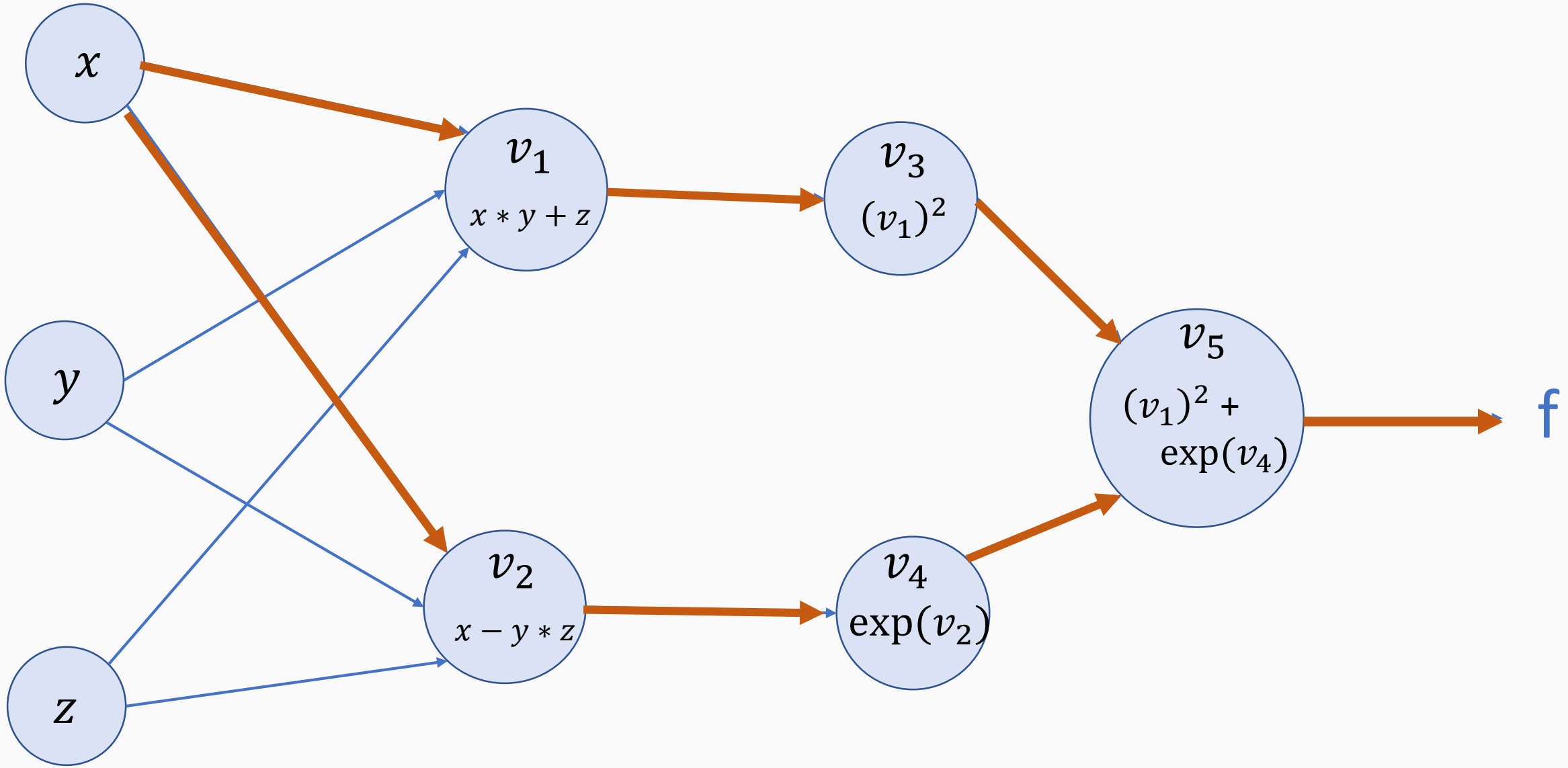
A more complicated graph (function)



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

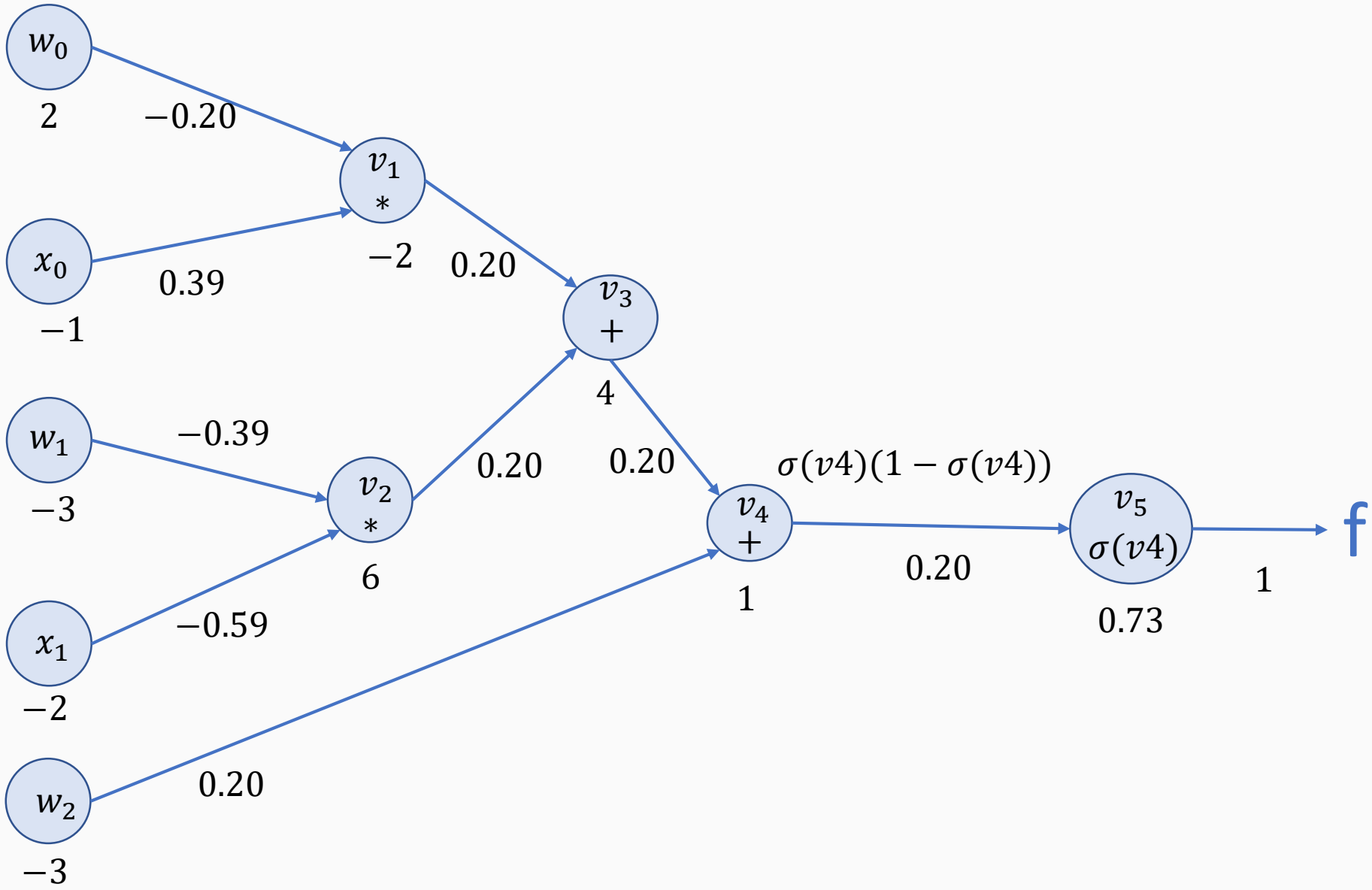
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial v_5}\frac{\partial v_5}{\partial v_3}\frac{\partial v_3}{\partial v_1}\frac{\partial v_1}{\partial x} + \frac{\partial f}{\partial v_5}\frac{\partial v_5}{\partial v_4}\frac{\partial v_4}{\partial v_2}\frac{\partial v_2}{\partial x} = 1*1*4*1+1*1*e*1,$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial v_5}\frac{\partial v_5}{\partial v_3}\frac{\partial v_3}{\partial v_1}\frac{\partial v_1}{\partial y} + \frac{\partial f}{\partial v_5}\frac{\partial v_5}{\partial v_4}\frac{\partial v_4}{\partial v_2}\frac{\partial v_2}{\partial y} = 0+1*1*4*2,$$

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

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.)

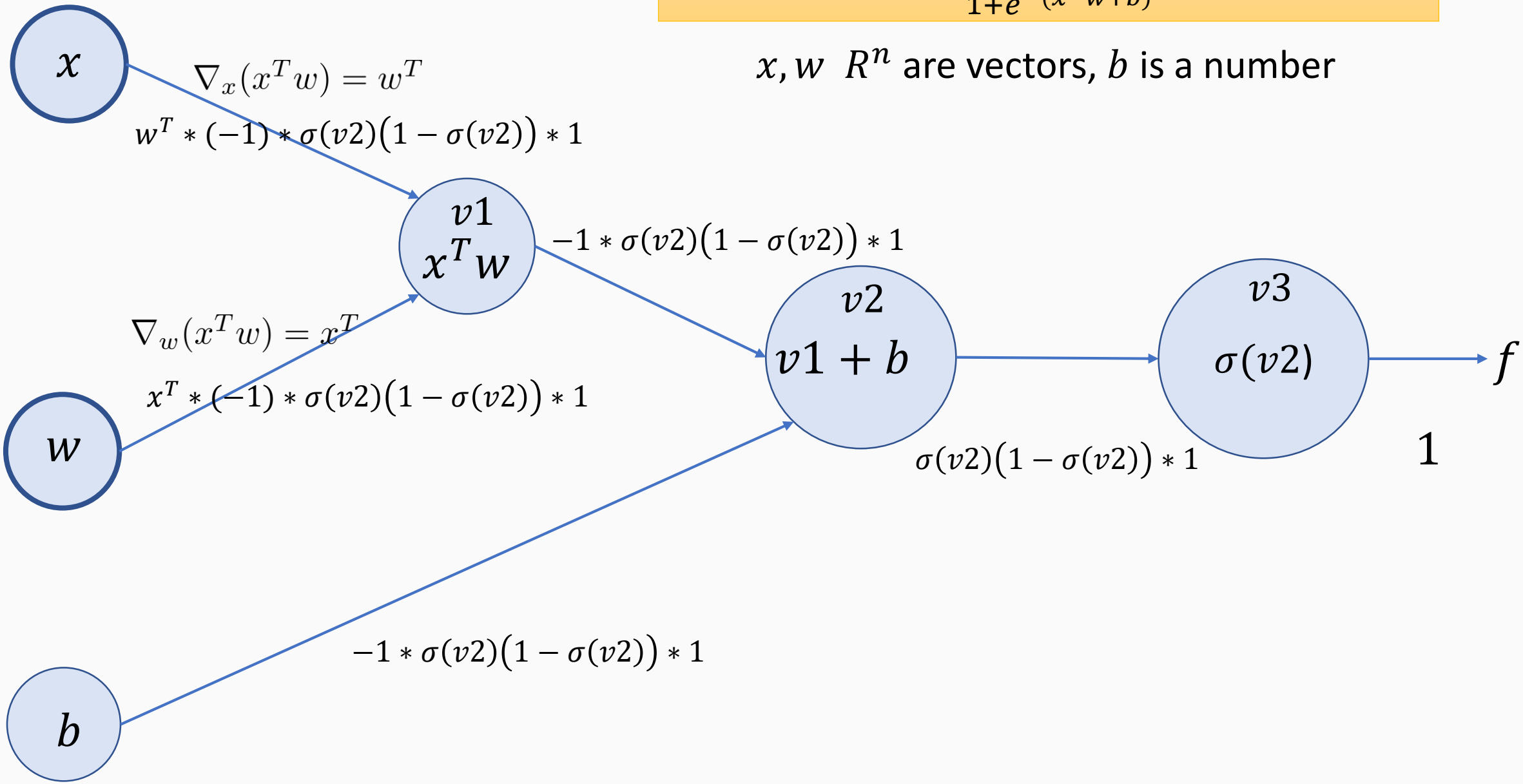
z is a scalar, $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, then

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

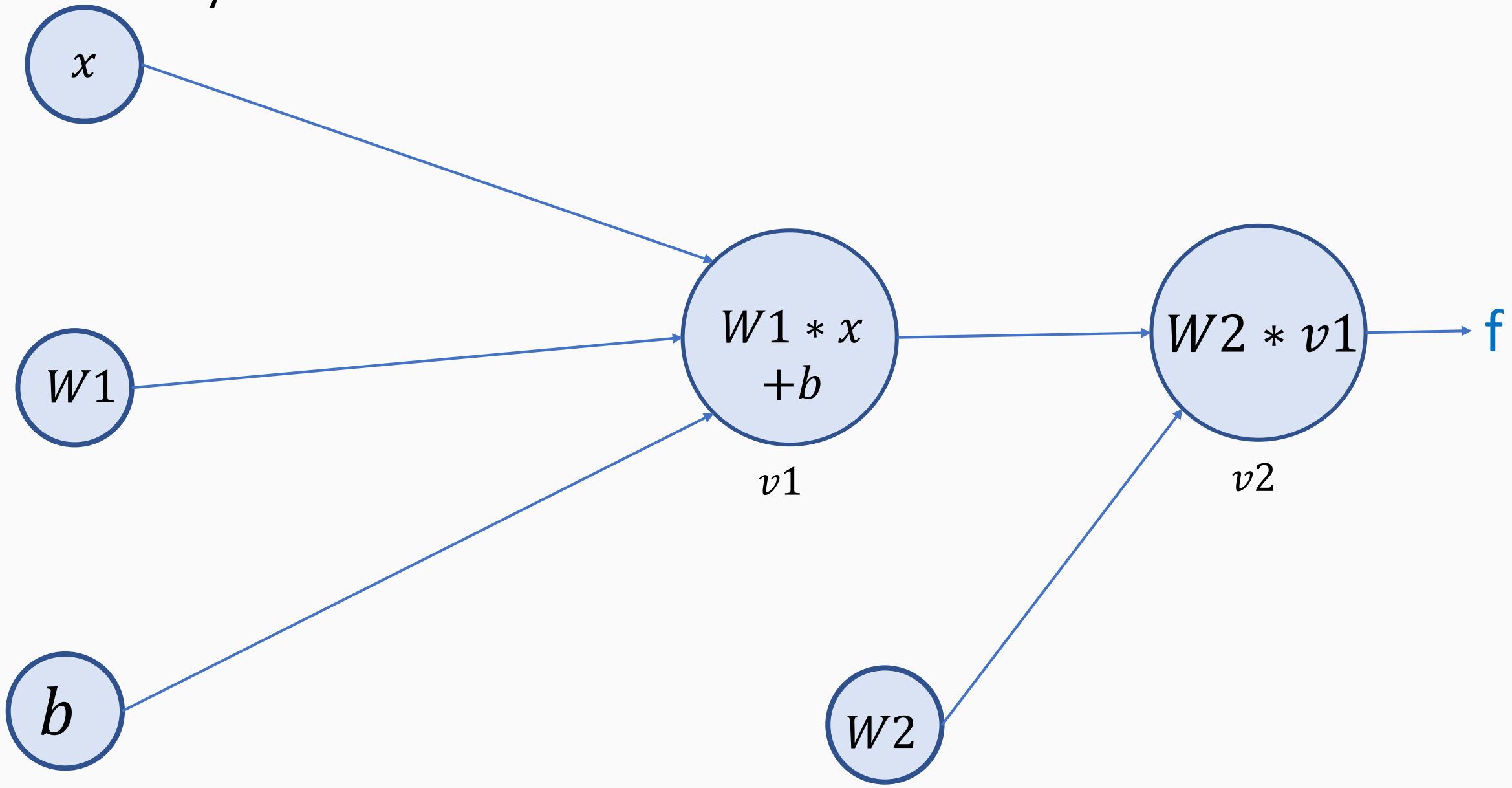
Vectorized backprop case study

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



One layered neural net with linear activation



Vectorization of matrices (tensors) , basic tensor calculus

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

$$\frac{\partial v_2}{\partial W_2}$$

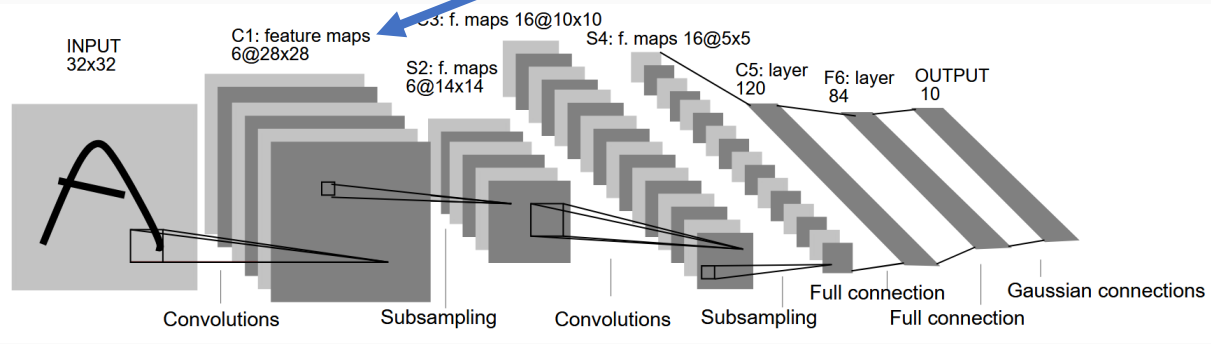
Derivative w.r.t. matrix W2
(it is a tensor)

Definition 1. *For a matrix M, we use the notation*

$$vec(M) = \begin{bmatrix} M_{col(1)} \\ M_{col(2)} \\ \cdots \\ M_{col(n)} \end{bmatrix}.$$

The operation $vec(A)$ transforms a matrix A into a vector, by stacking the columns of A one underneath another.

Convolutional layers
(tensors)



Vectorization of matrices (tensors) cont.

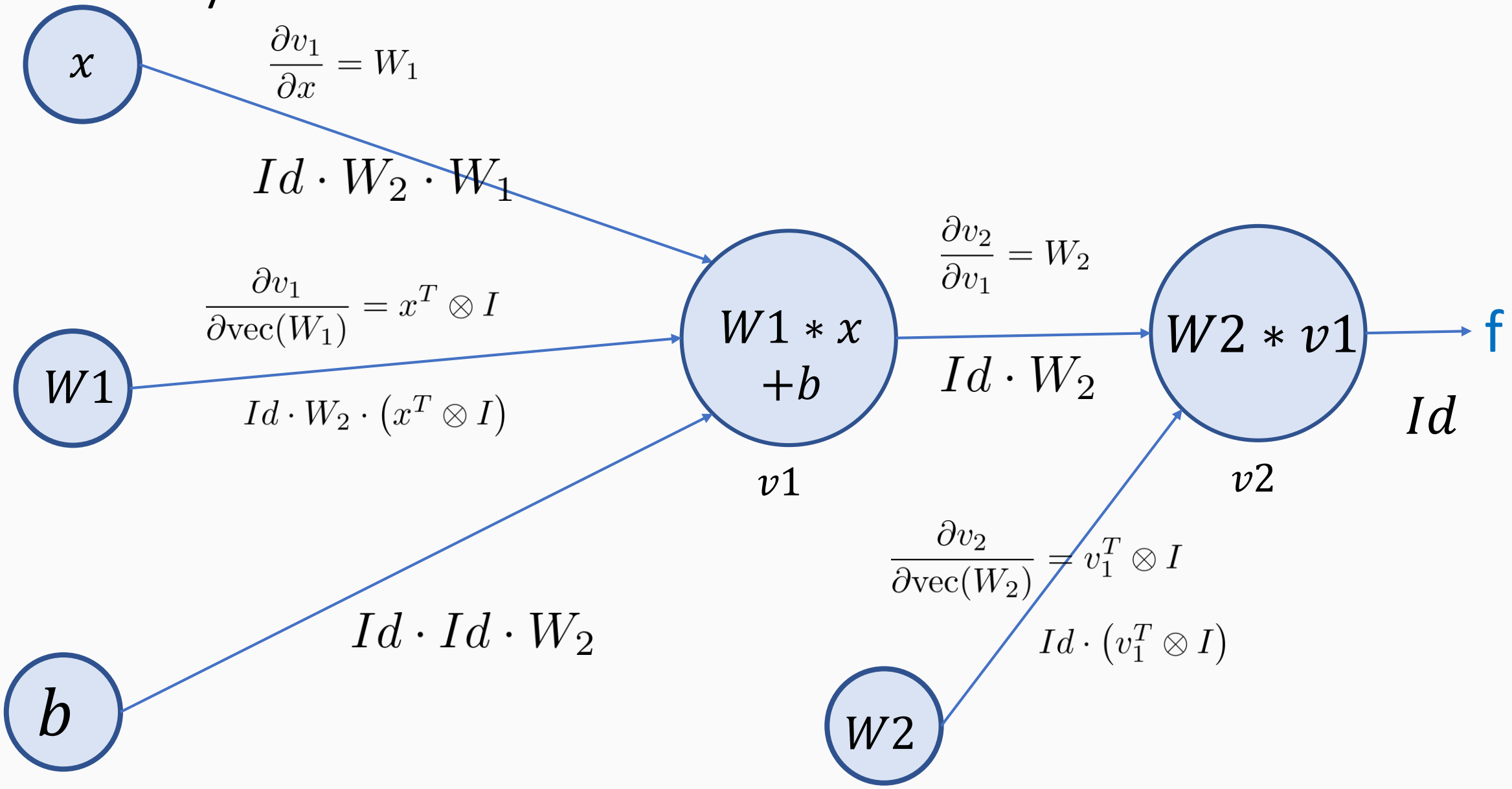
Instead of dealing with tensors, can use vectorization, and compute like that for $v_2 = W_2 v_1$

$$\frac{\partial v_2}{\partial \mathrm{vec}(W_2)} = v_1^T \otimes I,$$

where

$$A \otimes B = \begin{bmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn}B \end{bmatrix}$$

One layered neural net with linear activation



Object oriented code design

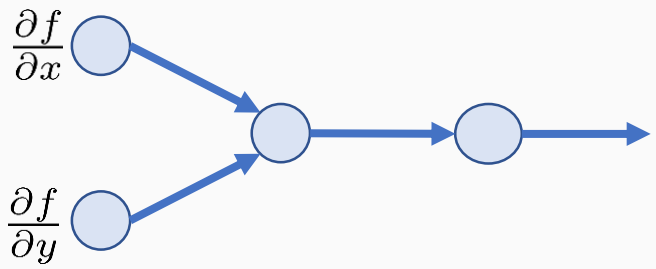
…Show on the blackboard…

Backpropagation gradient checking

How to debug my backprop algorithm implementation ???

Given (x, y)

Backprop to compute the partial derivatives



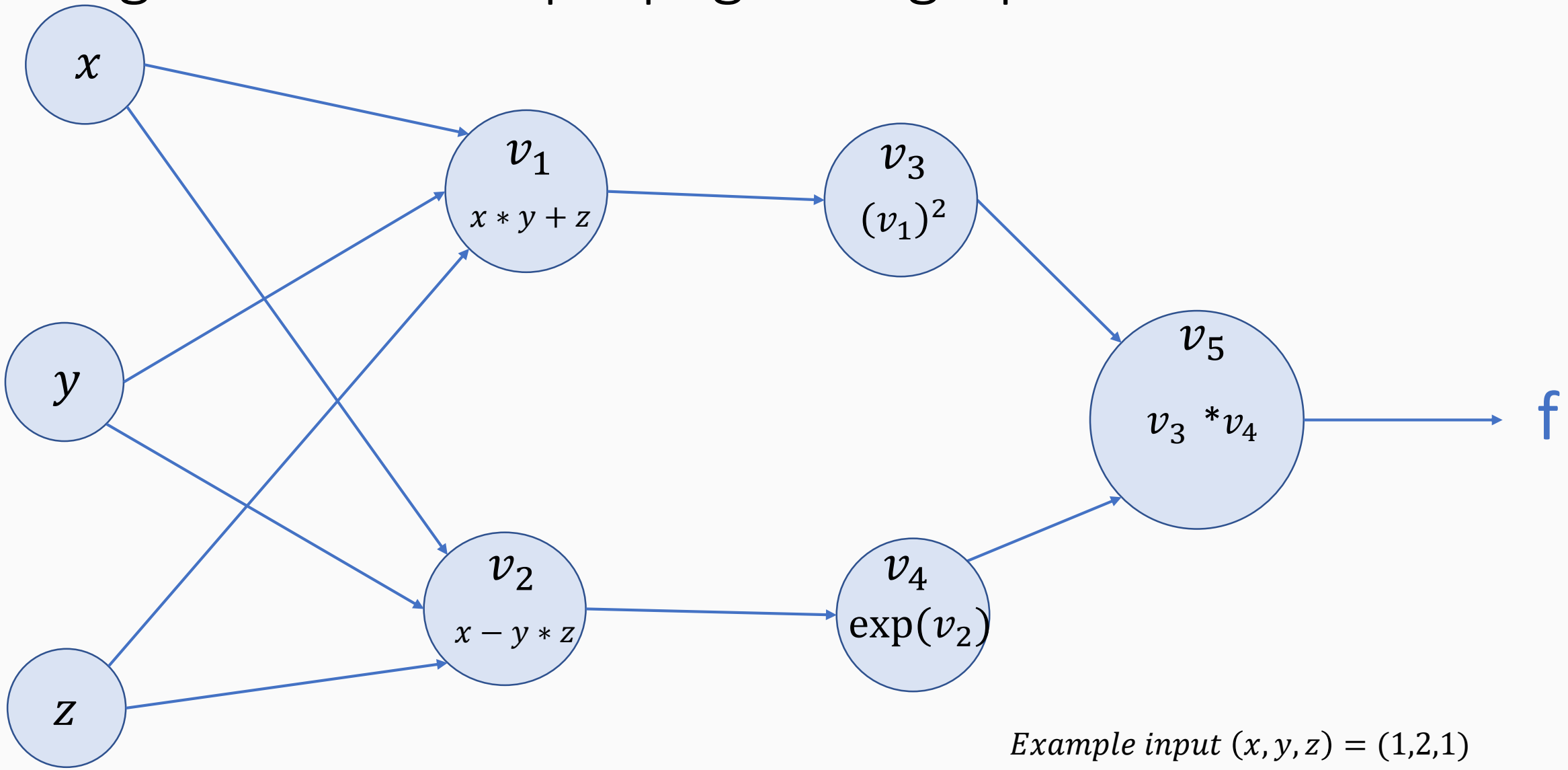
Estimate the rate of change of $f(x, y)$ for fixed (small) h in x and y direction

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

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

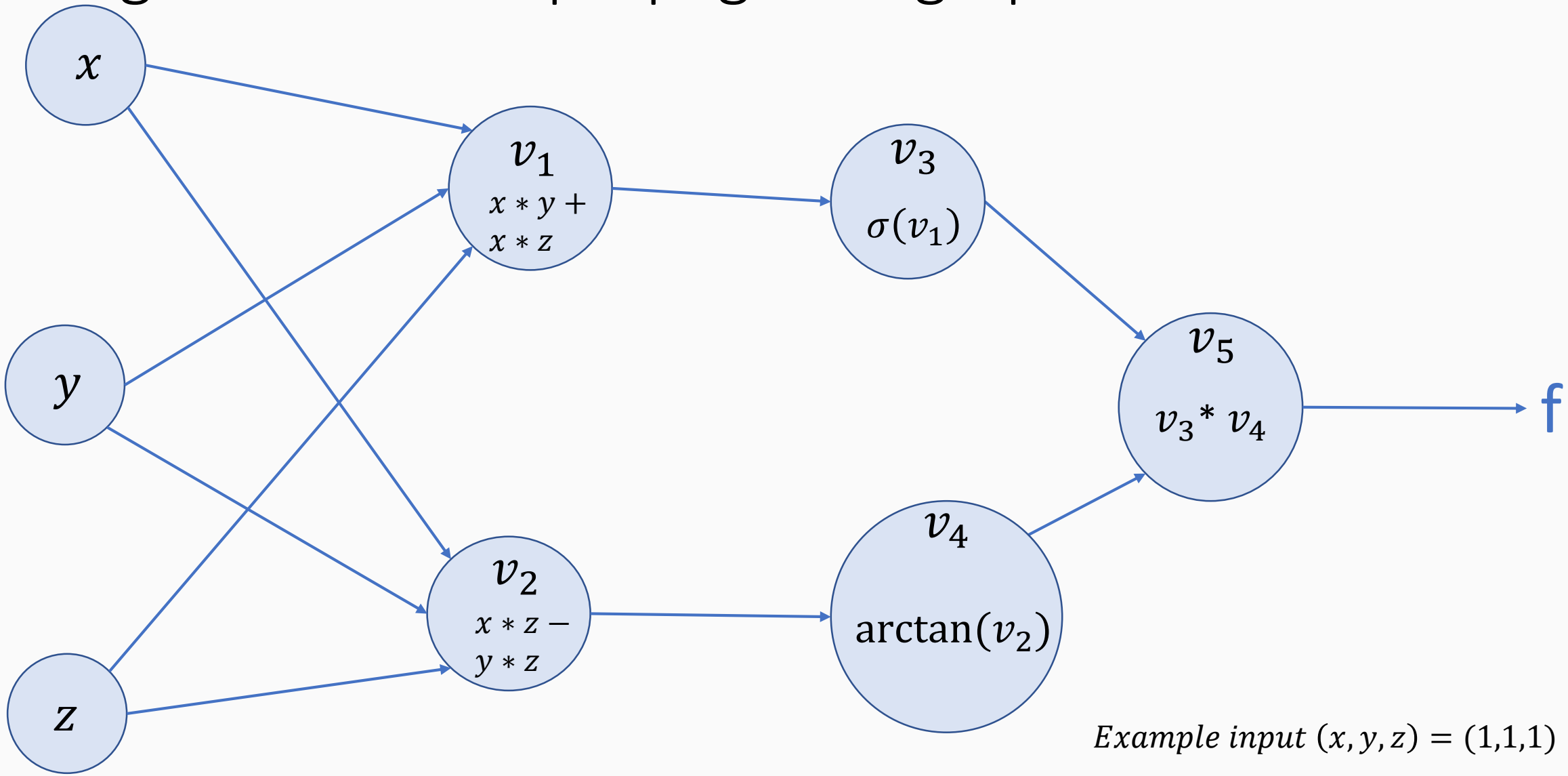
Should be comparable!

Assignment 6 backpropagation graph 1



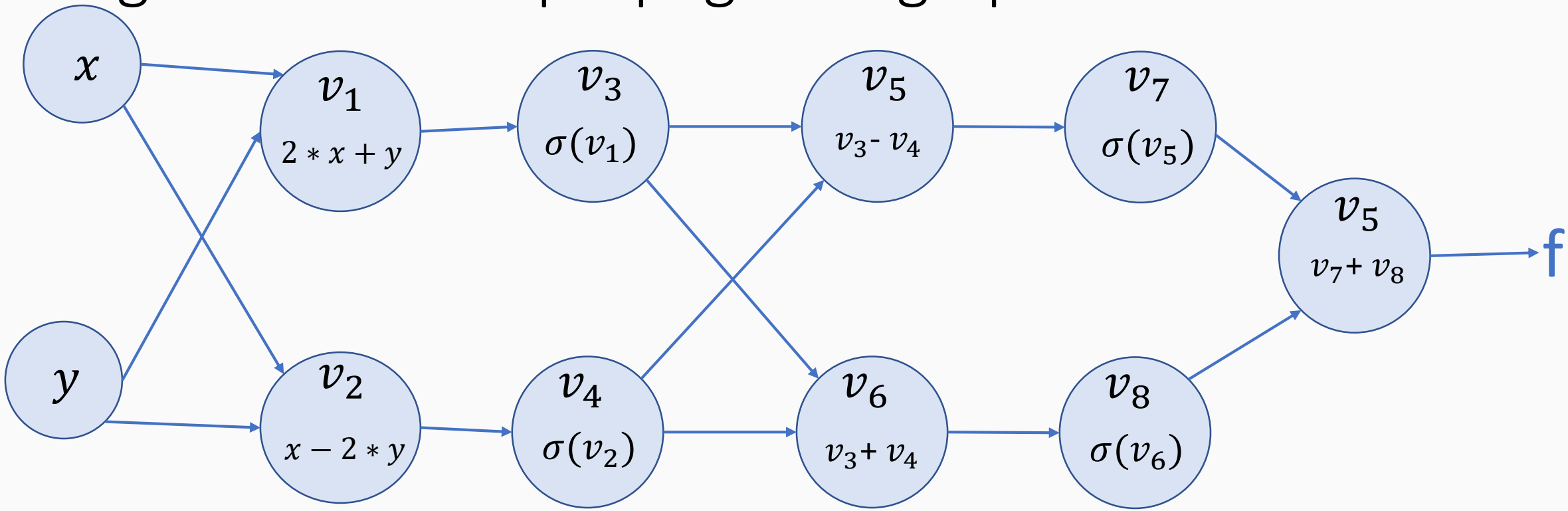
$Example\ input\ (x, y, z) = (1,2,1)$

Assignment 6 backpropagation graph 2



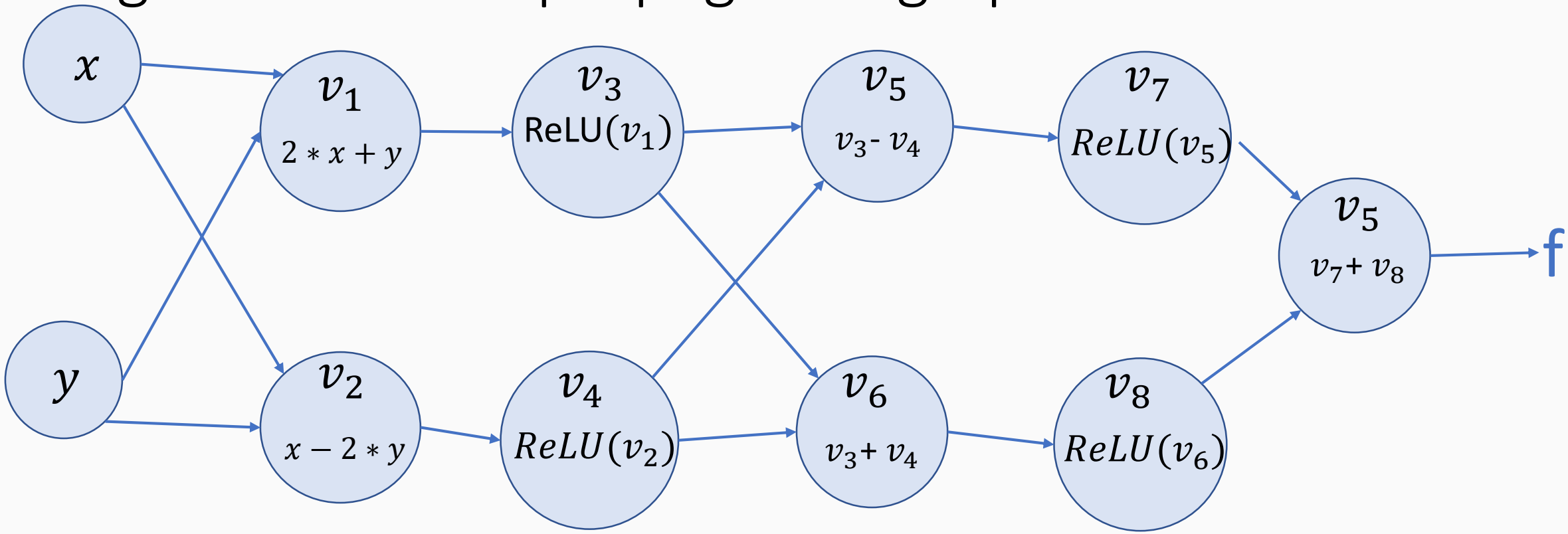
$Example\ input\ (x, y, z) = (1,1,1)$

Assignment 6 backpropagation graph 3



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

Assignment 6 backpropagation graph 4



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

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

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