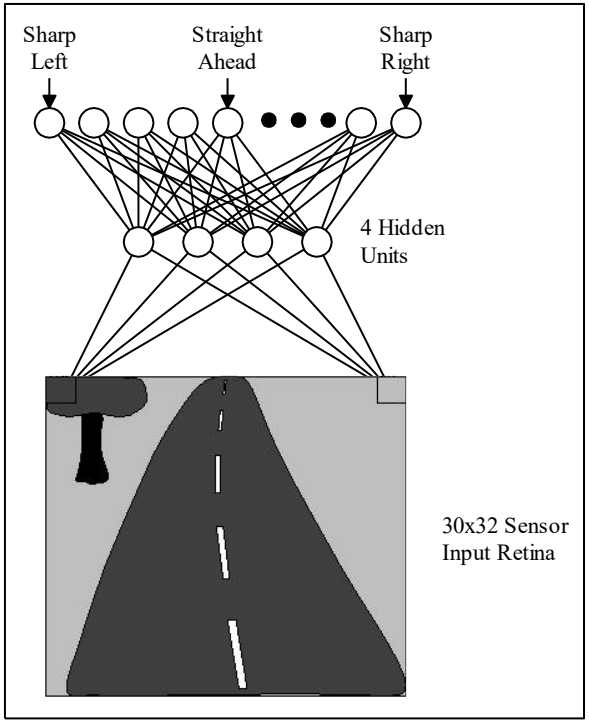
SGD: Iterative Model Training for Machine Learning

UCSB CS140, T. Yang

Table of Content

- Background: Machine learning
- Stochastic gradient descent (SGD) for iterative learning of parameters
  - Parallel SGD
- Gradient computation

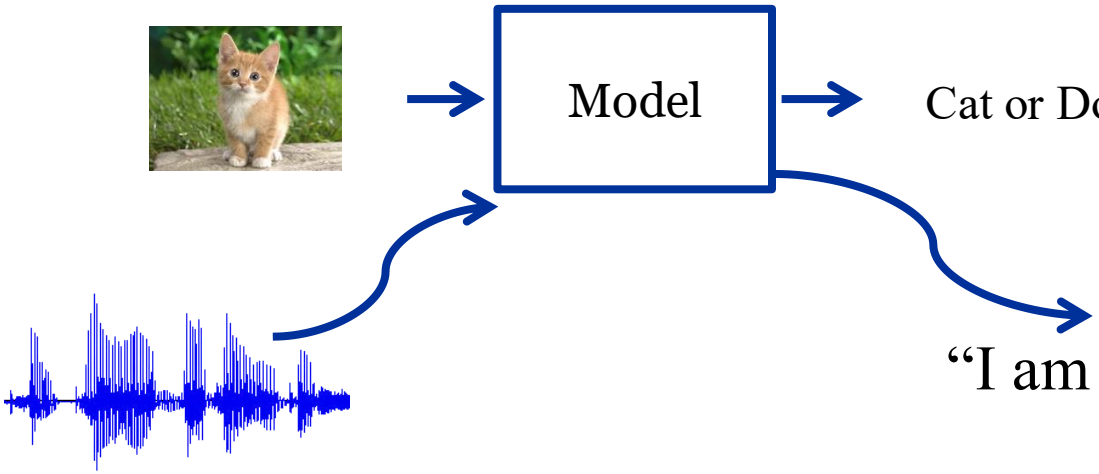


Why?

Wildly used in model training for vision, audio, and text processing with machine learning

What is Machine Learning (ML)?

- The goal of ML is to learn from data
  - Technically, combines statistics and computational tools (optimization)
- Example (supervised learning) tasks



What is a model in ML applications?

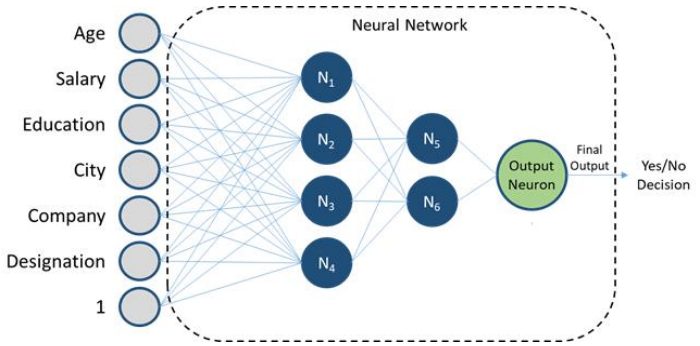
- A model is a function specified by a set of *parameters* θ

$f_\theta(x)$ → 0.99

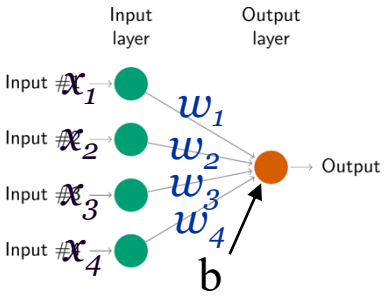
- Example: linear predictor with a parameter vector $\theta = (w, b)$
- $f_\theta(x) = w^T \cdot x + b$

$= w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$

More complex model: multi-layer neural network



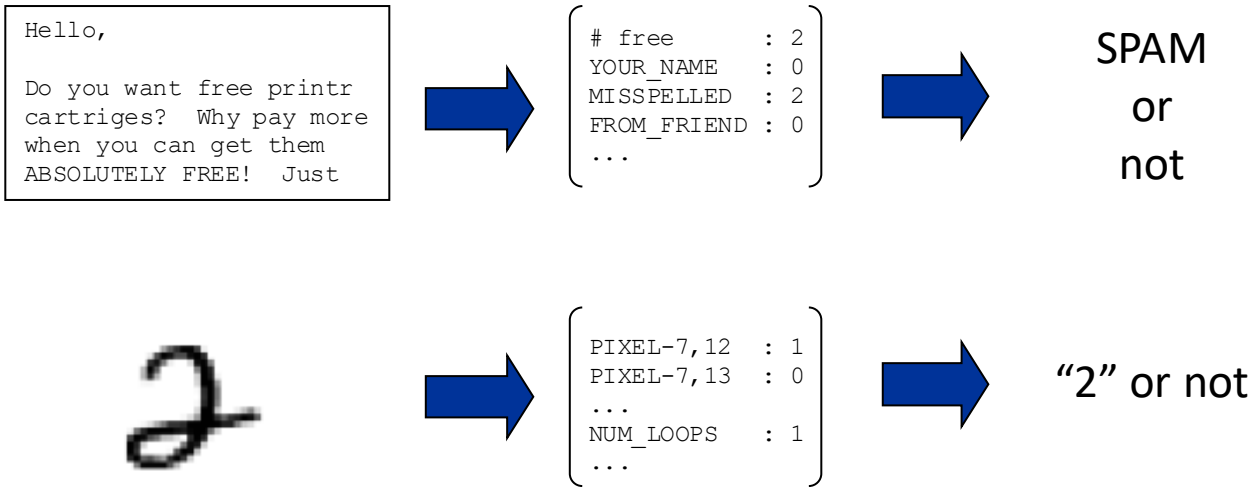
Neural network view:

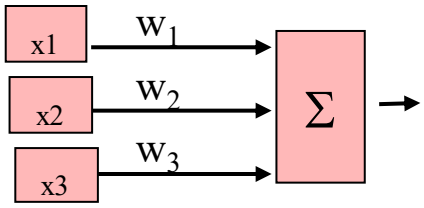


Model example: Binary classifier with linear feature combinations

x $f(x)$ $\hat{y}$

Also called perceptron





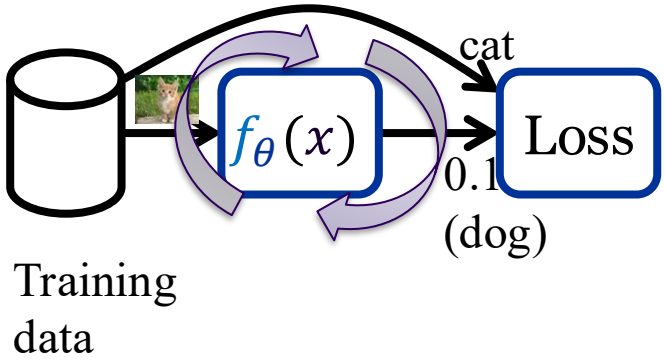
Neural net representation

How to find weight parameters w_1, w_2, w_3?

$$\hat{y} = w_1 x_1 + w_2 x_2 + w_3 x_3$$

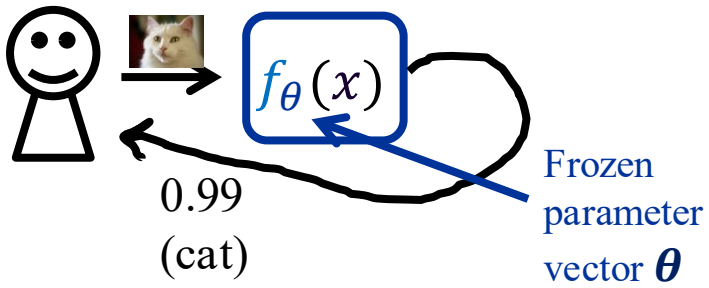
Training and online inference

Training



- The loss tells what the output of the model *should have been*
- Training objective is to find model parameters $\boldsymbol{\theta}$ that minimize the loss

Inference (validation)



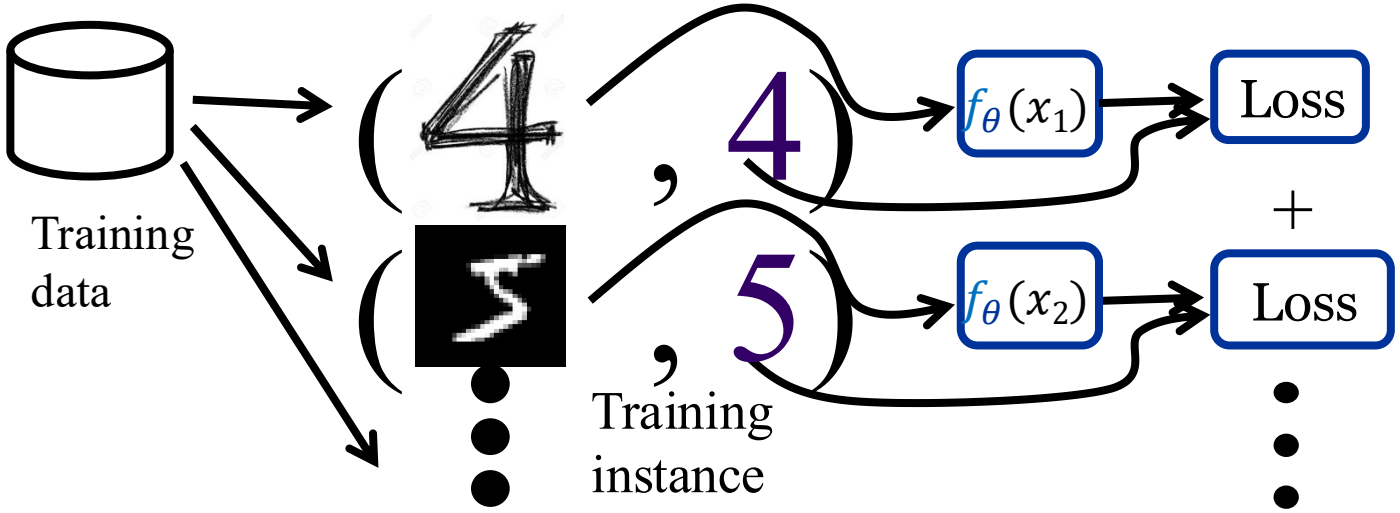
- Given a real data input (an element or a vector), predict a value based on the model trained.

Training a model with *n* instances

1. Compute the prediction for each instance and
2. Compute the loss by comparing against the expected ground truth.
3. Compute the total loss (average or just sum)

Find parameters θ minimizing total loss: $L(\theta) = \frac{1}{n}\sum_{i=1}^{n} Loss(f_\theta(x_i), y_i)$

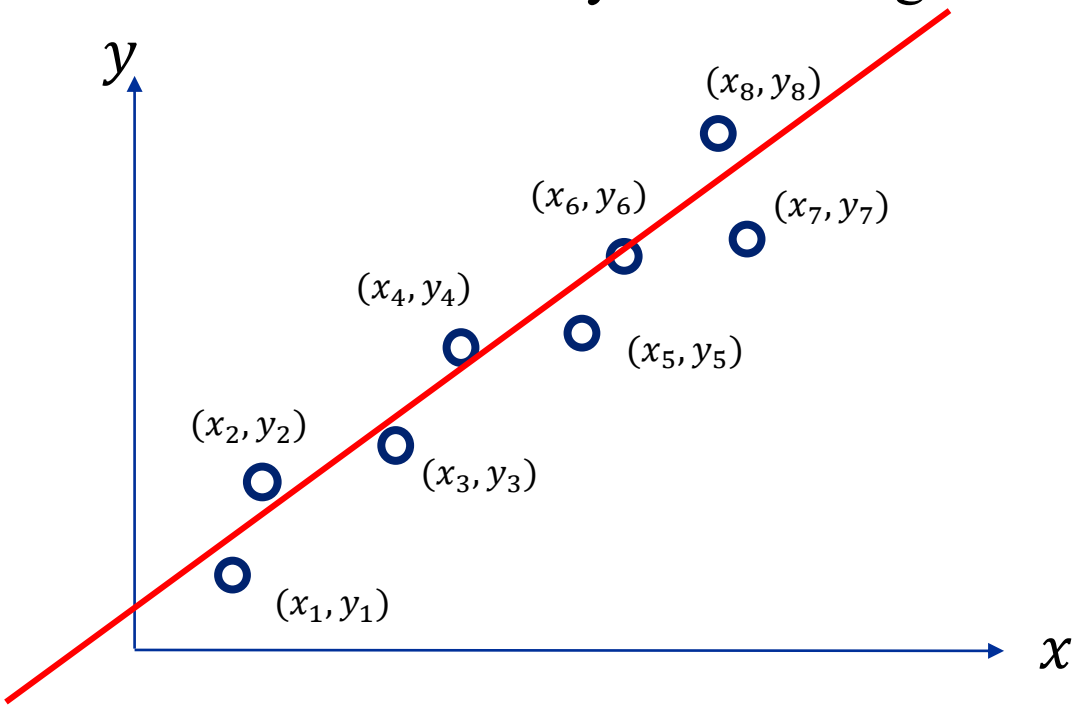
Letter recognition:



ChatGPT: What is the most likely word after "European Central" → Bank

A model with 1-parameter using linear regression

Given 8 training data instances: $(x_1, y_1), \ldots, (x_8, y_8)$
find a function that predicts value $\hat{y} = f(x)$, given input x.
Parameters of f are found by minimizing a loss expression



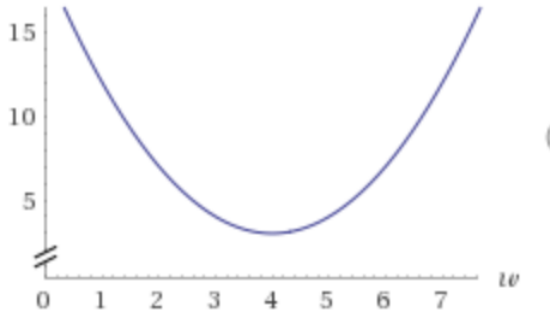
Linear model with two parameters w, b : $\hat{y} = wx + b$

Loss: $L(w, b) = \sum_{i=1}^{i=8} (\hat{y}_i - y_i)^2$

Minimize loss function L(w) with one parameter

Loss function example: with 1 parameter

$$L(w) = 3 + (w - 4)^2$$



Easy way to find minimum (and max): Find where $\frac{\partial}{\partial w} L(w) = 0$

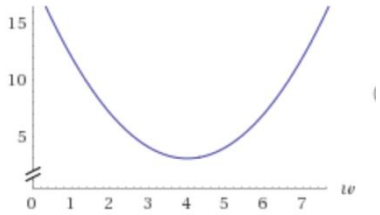
$$\frac{\partial}{\partial w} L(w) = 2(w - 4)$$

This is zero when: $w = 4$

How to find w that minimizes loss function L(w)

Easier for simpler functions by computing the root of derivatives:

$$L(w) = 3 + (w - 4)^2$$



Harder for deriving the root for more complex functions

$$L(w) = e^{-w} + w^2$$

$$\frac{\partial L(w)}{\partial w} = -e^{-w} + 2w$$

$$0 = -e^{-w} + 2w$$

How do you find w?

How to find parameters minimizing loss function L(w)

Easy for loss functions with one parameter:

$$L(w) = 3 + (w - 4)^2$$

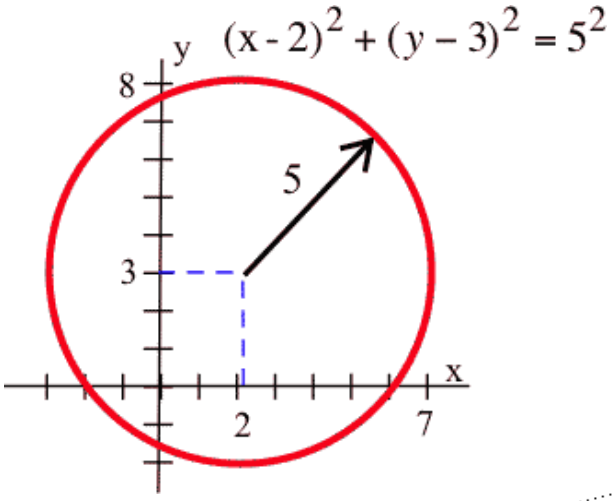
Not easy for loss functions with multiple variables by finding the zero of partial derivatives:

$$L(w_1, w_2, \ldots, w_{12}) = w_1 + w_2 + \ldots + w_{12}$$

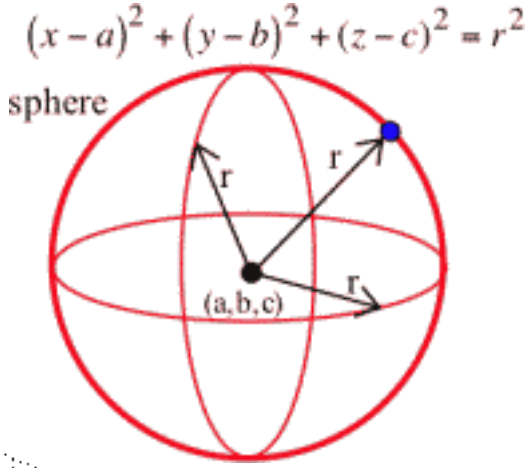
Parameter vector w= $(w_1, w_2, \ldots, w_{12})$

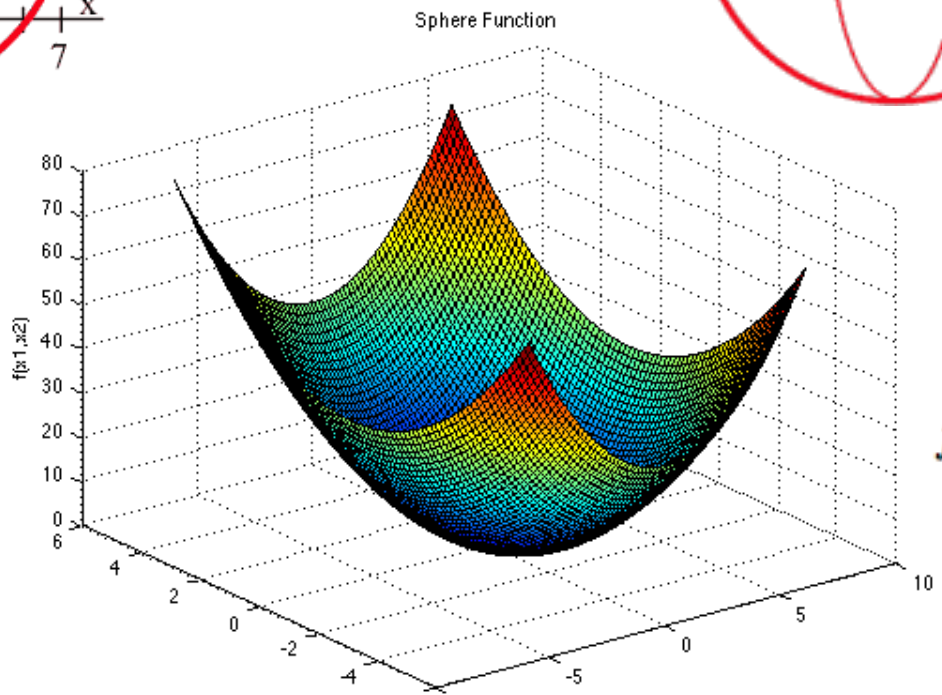
Examples of multi-variable functions

Two variables



3 variables





$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i^2$$

Background Knowledge: Partial Derivatives and Gradient

Single-variable functions

Notation for the Derivative

$$f'(x) \quad y' \quad \frac{dy}{dx} \Bigg\} \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Multi-variable functions

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Gradient

Scalar-valued multivariable function

$$\nabla f(x_0, y_0, \dots) = \begin{bmatrix} \frac{\partial f}{\partial x}(x_0, y_0, \dots) \\ \frac{\partial f}{\partial y}(x_0, y_0, \dots) \\ \vdots \end{bmatrix}$$

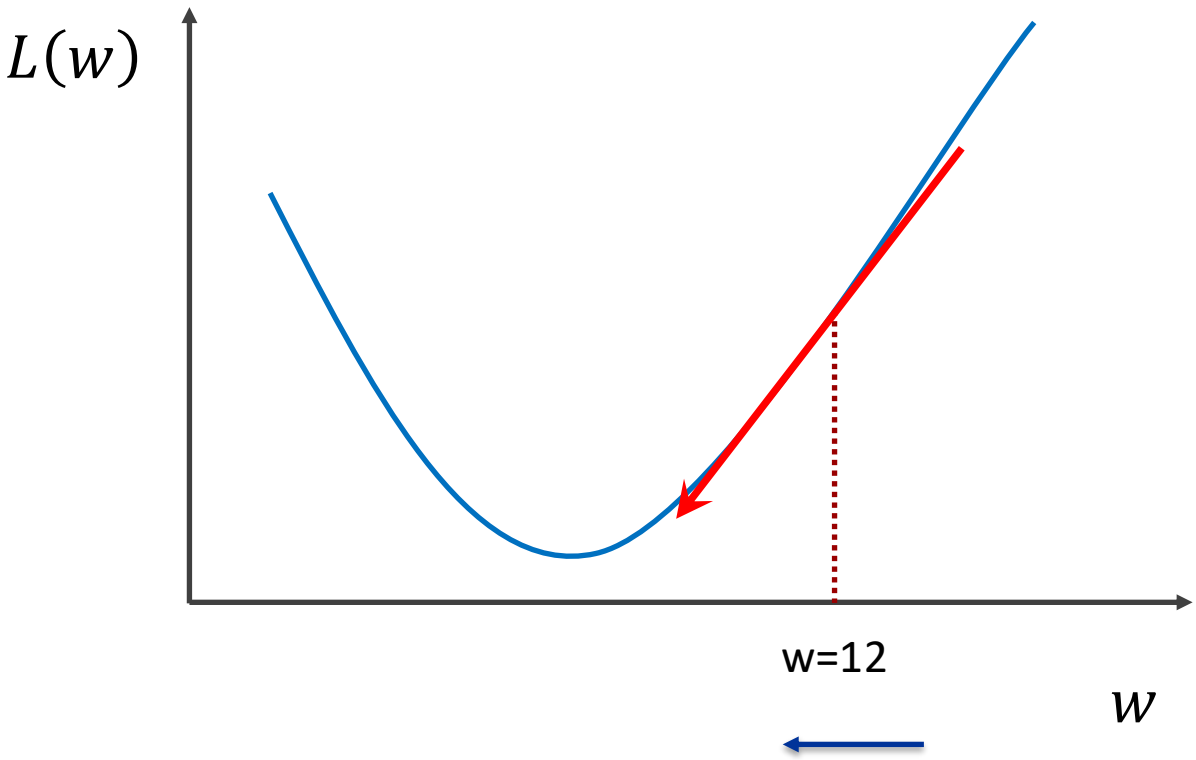
∇f takes the same type of inputs as f

Notation for gradient, called "nabla".

∇f outputs a vector with all possible partial derivatives of f.

Idea of **Gradient Descent (GD)** algorithm to find w that minimizes loss function L(w)

Iterative method to search a local minimum by following the direction of the steepest descent based on gradient



1. Start with a random value of w (e.g. w = 12)
2. Compute the gradient (derivative) of L(w) at point w = 12. (e.g. $\frac{\partial L(w)}{\partial w}$ = 6)
3. Recompute w as:

$$w = w - \lambda \frac{\partial L(w)}{\partial w}$$

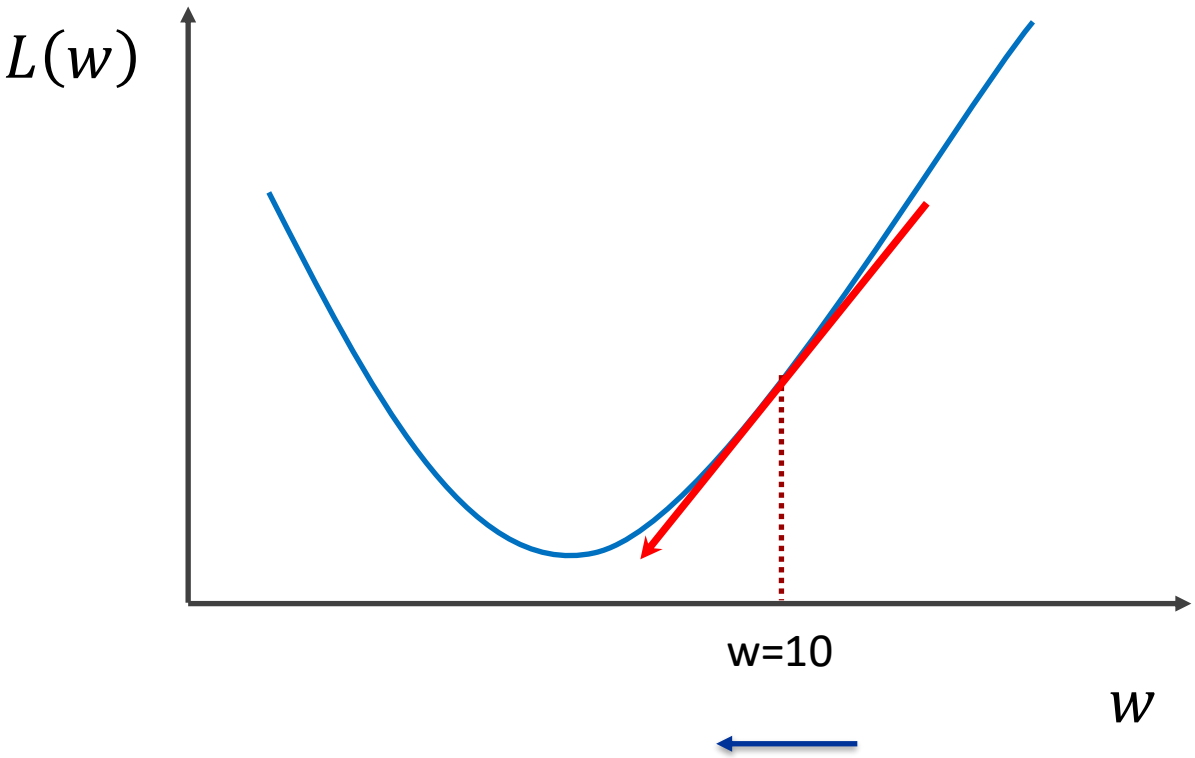
Constant learning rate
λ=1/3

w^{new}=12-1/3*6=10

Gradient Descent (GD)

Find w that minimizes loss function L(w)

Iterative method to search a local minimum by following the direction of the steepest descent based on gradient



2. Compute the gradient (derivative) of L(w) at point w = 10. (e.g. $\frac{\partial L(w)}{\partial w} = 3$)
3. Recompute w as:

$$w = w - \lambda \frac{\partial L(w)}{\partial w}$$

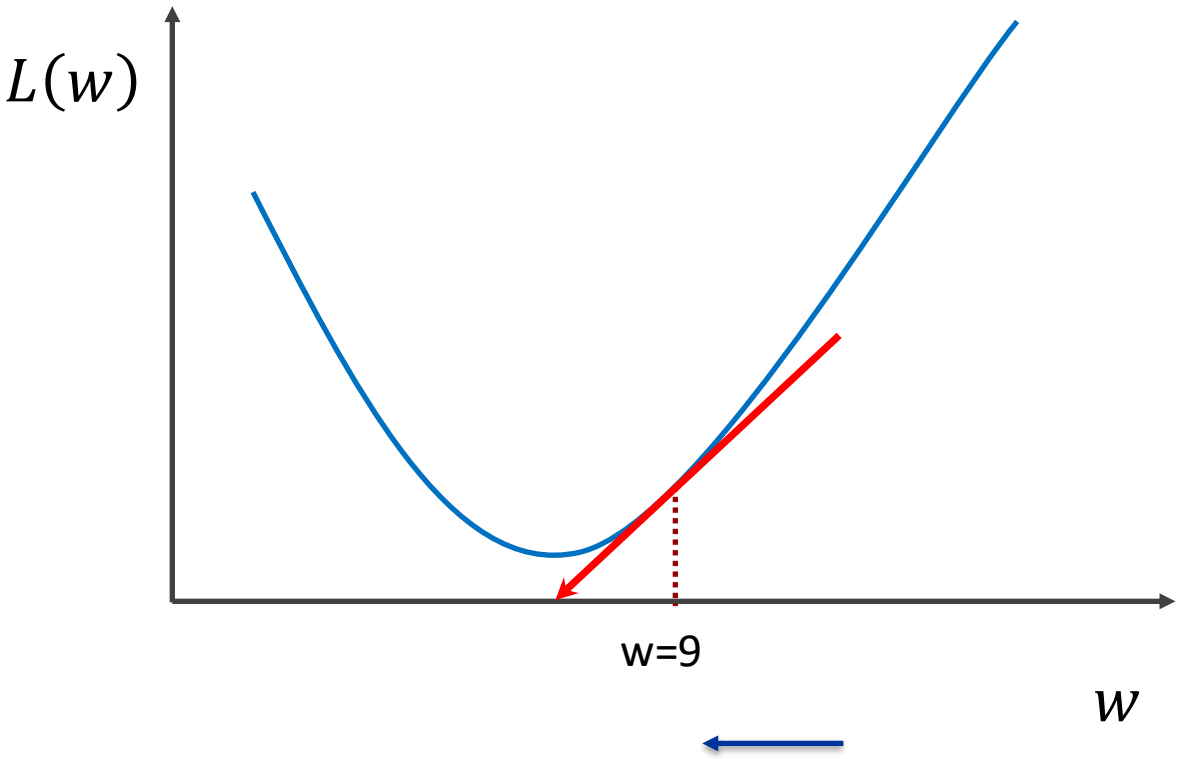
Constant learning rate

λ=1/3

w^{new}=10—1/3*3=9

Gradient Descent (GD)
Find w that minimizes loss function L(w)

Iterative method to search a local minimum by following the direction of the steepest descent based on gradient



2. Compute the gradient (derivative) of L(w) at point w = 9. (e.g. $\frac{\partial L(w)}{\partial w} = 2$)

3. Recompute w as:

$$w = w - \lambda \frac{\partial L(w)}{\partial w}$$

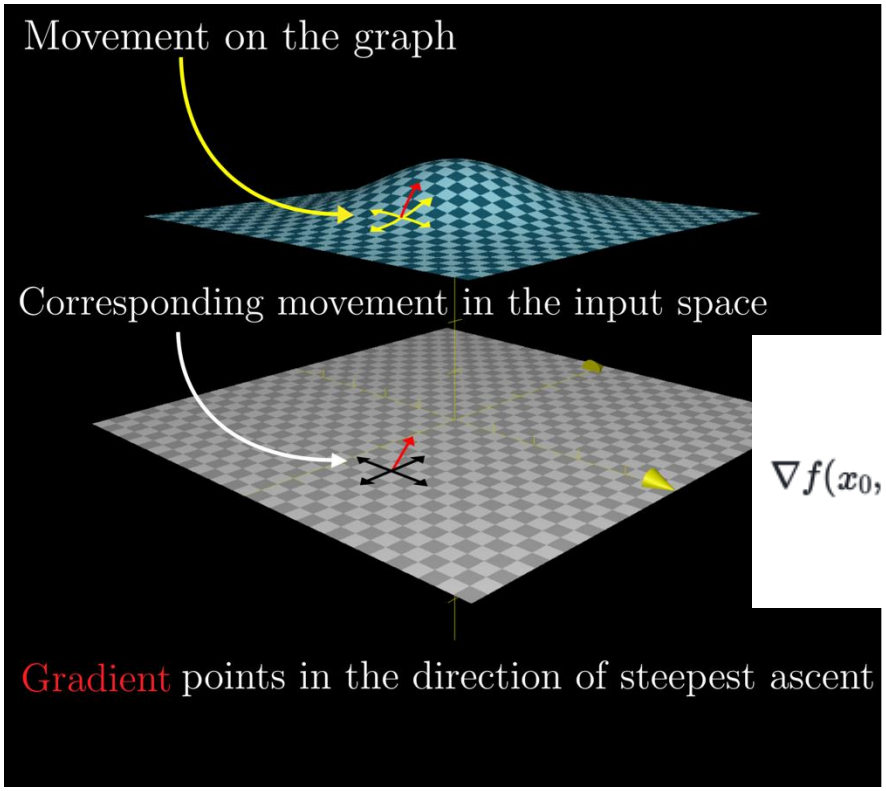
Constant learning rate λ=1/3

w^{new}=9-2/3=8.33

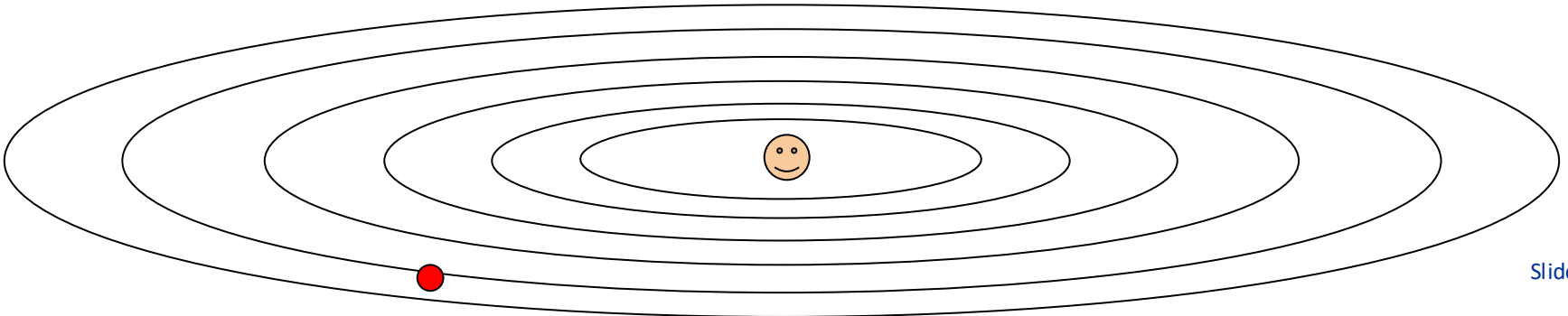
Gradient Descent (GD) with multiple parameters

https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/partial-derivative-and-gradient-articles/a/the-gradient

Iterative method to search a local minimum by following the direction of the steepest descent based on gradient

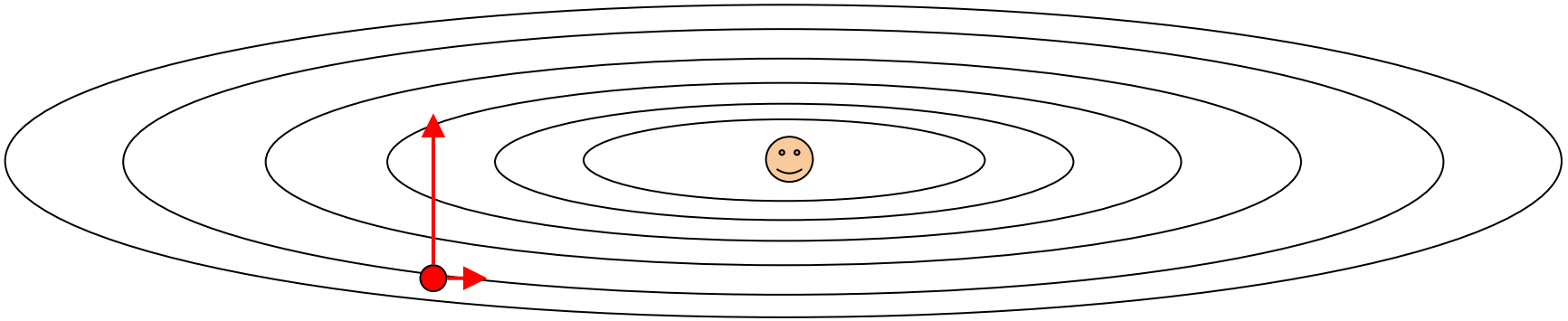


$$\nabla f(x_0, y_0) = \begin{bmatrix} \frac{\partial f}{\partial x}(x_0, y_0) \\ \\ \frac{\partial f}{\partial y}(x_0, y_0) \end{bmatrix}$$



Gradient Descent (GD) with multiple parameters

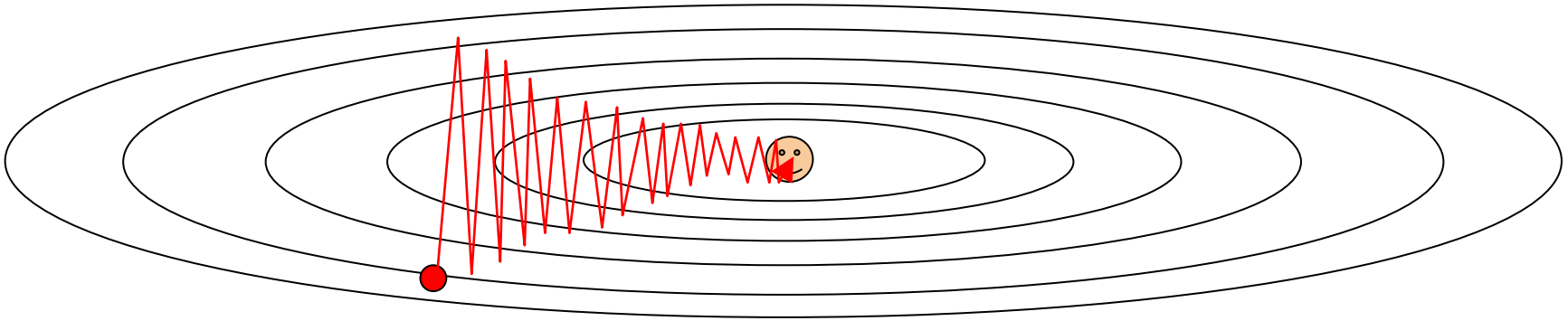
Iterative method to search a local minimum by following the direction of the steepest descent based on gradient



Q: What is the trajectory along which we converge towards the minimum with SGD?

Gradient Descent (GD) with multiple parameters

Iterative method to search a local minimum by following the direction of the steepest descent based on gradient



Q: What is the trajectory along which we converge towards the minimum with Gradient Descent? very slow progress along flat direction, jitter along steep one

Stochastic Gradient Descent (SGD)
for Model Learning with Two Parameters

Update parameters with one instance at a time

Assume training instance (x_i, y_i) has two-feature input vector $x_i = (x_{i,1}, x_{i,2})$ with ground truth y_i

Learning rate: $\lambda = 0.01$

Initialize weight vector w randomly

Repeat

Select a training instance (x_i, y_i)

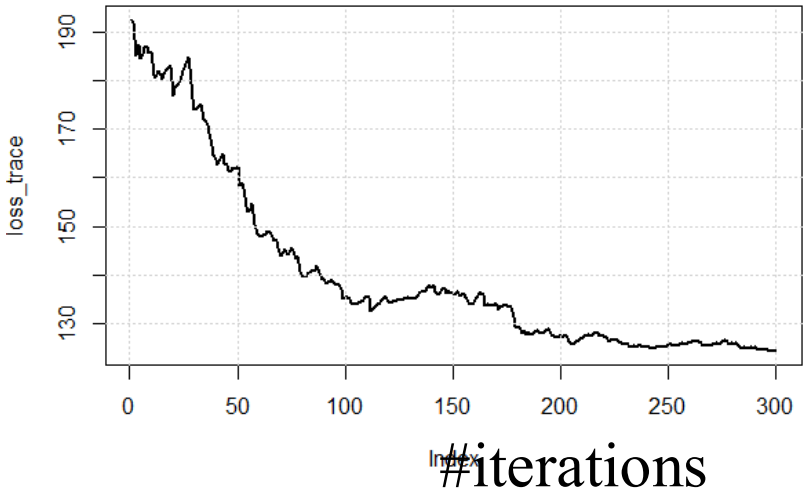
Compute: $\frac{\partial l(w,x_i)}{\partial w_1}$ and $\frac{\partial l(w,x_i)}{\partial w_2}$

Update w as:

$$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} - \lambda \begin{pmatrix} \frac{\partial L(w)}{\partial w_1} \\ \frac{\partial L(w)}{\partial w_2} \end{pmatrix}$$

Stop iteration if meeting some criteria

end



Assume loss is

$$l(w, x_i) = (\hat{y}_i - y_i)^2$$

$$\hat{y}_i = w * x_i = w_1 x_{i,1} + w_2 x_{i,2}$$

Mini-batch SGD with two parameters

Update parameters with a batch of *m* instances at a time

$\lambda = 0.01$

Initialize w and b randomly

For I= 1, #batches **do**

Get *m* instances from a batch

Compute: $\frac{\partial L(w,x)}{\partial w_1}$ and $\frac{\partial L(w,x)}{\partial w_2}$ for *m* instances

Update *w* as $\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} - \lambda \frac{1}{m} \sum_{i=1}^{m} \begin{pmatrix} \frac{\partial l(w, x_i)}{\partial w_1} \\ \frac{\partial l(w, x_i)}{\partial w_2} \end{pmatrix}$

Stop iteration if meeting some criteria

end

Average loss is $\frac{1}{m} \sum_{i=1}^{m} l(w, x_i)$

$l(w, x_i) = (\hat{y}_i - y_i)^2$

$\hat{y}_i = w * x_i = w_1 x_{i,1} + w_2 x_{i,2}$

- Stop after a fixed number of iterations.
- Or when loss is close to a lower bound or has not improved much in a long time.

Example of SGD Learning from training data

- *Classifier*: Predicted value is Size *w_1+color *w_2

Instance	Size $x_{i,1}$	Color $x_{i,2}$	Category y_i
x_1	Small 0	Red 1	Positive 1
x_2	Large 1	Red 1	Positive 1
x_3	Small 0	Blue 0	Negative -1
x_4	Large 1	Blue 0	Negative -1

$$\hat{y}_i = w * x_i = w_1 x_{i,1} + w_2 x_{i,2}$$

Loss function:

$$l(w, x_i) = (\hat{y}_i - y_i)^2$$
$$= (w_1 x_{i,1} + w_2 x_{i,2} - y_i)^2$$

$$\frac{\partial L(w,x)}{\partial w_1} = 2(\hat{y}_i - y_i)\frac{\partial(\hat{y}_i - y_i)}{\partial w_1} = 2(\hat{y}_i - y_i)x_{i,1} = 2(w_1 x_{i,1} + w_2 x_{i,2} - y_i)\, x_{i,1}$$

$$\frac{\partial L(w,x)}{\partial w_2} = 2(\hat{y}_i - y_i)\frac{\partial(\hat{y}_i - y_i)}{\partial w_2} = 2(\hat{y}_i - y_i)x_{i,2} = 2(w_1 x_{i,1} + w_2 x_{i,2} - y_i)\, x_{i,2}$$

SGD update: $$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} - \lambda \begin{pmatrix} \frac{\partial L(w)}{\partial w_1} \\ \frac{\partial L(w)}{\partial w_2} \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} - \lambda \begin{pmatrix} 2(\hat{y}_i - y_i)x_{i,1} \\ 2(\hat{y}_i - y_i)x_{i,2} \end{pmatrix}$$

Example of SGD Learning from training data with 2-feature input vectors

- *Classifier*: Predicted value is Size *w_1+color *w_2

Instance	Size $x_{i,1}$	Color $x_{i,2}$	Category y_i
x_1	Small 0	Red 1	Positive 1
x_2	Large 1	Red 1	Positive 1
x_3	Small 0	Blue 0	Negative -1
x_4	Large 1	Blue 0	Negative -1

$$\hat{y}_i = w * x_i = w_1 x_{i,1} + w_2 x_{i,2}$$

Loss function:

$$l(w, x_i) = (\hat{y}_i - y_i)^2$$
$$= (w_1 x_{i,1} + w_2 x_{i,2} - y_i)^2$$

SGD updates w as $\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} - \lambda \begin{pmatrix} \frac{\partial L(w)}{\partial w_1} \\ \frac{\partial L(w)}{\partial w_2} \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} - \lambda \begin{pmatrix} 2(\hat{y}_i - yi)x_{i,1} \\ 2(\hat{y}_i - yi)x_{i,2} \end{pmatrix}$

Let λ =0.5. Initially w_1=w_2 =0

With Instance 1: $\hat{y}_1$=0+0=0. $\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} - 0.5 \begin{pmatrix} 2(0-1)0 \\ 2(0-1)1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

With Instance 2: $\hat{y}_2$=0+1x1=1. $\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} - 0.5 \begin{pmatrix} 2(1-1)1 \\ 2(1-1)1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

Example of mini-batch SGD with batch size 2

- *Classifier model*: Predicted value is Size *w_1+color *w_2

Instance	Size $x_{i,1}$	Color $x_{i,2}$	Category y_i
x_1	Small 0	Red 1	Positive 1
x_2	Large 1	Red 1	Positive 1
x_3	Small 0	Blue 0	Negative -1
x_4	Large 1	Blue 0	Negative -1

$$\hat{y}_i = w * x_i = w_1 x_{i,1} + w_2 x_{i,2}$$

Avg. loss $L(w,x)$ is $\frac{1}{2}\sum_{i=1}^{2} l(w, x_i)$

$l(w, x_i) = (\hat{y}_i - y_i)^2$
$= (w_1 x_{i,1} + w_2 x_{i,2} - y_i)^2$

Miniatch update: $\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} - 0.5\lambda \begin{pmatrix} \frac{\partial l(w,x_1)}{\partial w_1} \\ \frac{\partial l(w,x_1)}{\partial w_2} \end{pmatrix} - 0.5\lambda \begin{pmatrix} \frac{\partial l(w,x_2)}{\partial w_1} \\ \frac{\partial l(w,x_2)}{\partial w_2} \end{pmatrix}$ Let λ =0.5. Initially w_1=w_2=0

With Instances 1 and 2 as a batch : $\hat{y}_1$=0+0=0. $\hat{y}_2$=0+0=0

$$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} - 0.25 \begin{pmatrix} 2(0-1)0 \\ 2(0-1)1 \end{pmatrix} - 0.25 \begin{pmatrix} 2(0-1)1 \\ 2(0-1)1 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 1 \end{pmatrix}.$$

With Instances 3 and 4 as a batch : $\hat{y}_3$=0. $\hat{y}_4$=1*0.5+0=0.5

$$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 1 \end{pmatrix} - 0.25 \begin{pmatrix} 2(0+1)0 \\ 2(0+1)0 \end{pmatrix} - 0.25 \begin{pmatrix} 2(0.5+1)1 \\ 2(0.5+1)0 \end{pmatrix} = \begin{pmatrix} -0.25 \\ 1 \end{pmatrix}.$$

Re-formulate with matrix-vector multiplication for mini-batch SGD with batch size 2 and 2-feature input vectors

Instance	Size $x_{i,1}$	Color $x_{i,2}$	Category y_i
x_1	Small 0	Red 1	Positive 1
x_2	Large 1	Red 1	Positive 1

$$\hat{y}_i = w * x_i = w_1 x_{i,1} + w_2 x_{i,2}$$

Average loss $L(w,x)$ is $\frac{1}{2}\sum_{i=1}^{2} l(w, x_i)$

$$l(w, x_i) = (\hat{y}_i - y_i)^2$$
$$= (w_1 x_{i,1} + w_2 x_{i,2} - y_i)^2$$

Batch SGD Update: $$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} - 0.5\lambda \begin{pmatrix} \frac{\partial l(w,x_1)}{\partial w_1} \\ \frac{\partial l(w,x_1)}{\partial w_2} \end{pmatrix} - 0.5\lambda \begin{pmatrix} \frac{\partial l(w,x_2)}{\partial w_1} \\ \frac{\partial l(w,x_2)}{\partial w_2} \end{pmatrix}$$

$$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} - 0.5\lambda \begin{pmatrix} 2(\hat{y}_1 - y_1)\, x_{1,1} \\ 2(\hat{y}_1 - y_1)\, x_{1,2} \end{pmatrix} - 0.5\lambda \begin{pmatrix} 2(\hat{y}_2 - y_2)\, x_{2,1} \\ 2(\hat{y}_2 - y_2)\, x_{2,2} \end{pmatrix}$$

$$= \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} - \lambda \begin{pmatrix} (\hat{y}_1 - y_1)\, x_{1,1} + (\hat{y}_2 - y_2)\, x_{2,1} \\ (\hat{y}_1 - y_1)\, x_{1,2} + (\hat{y}_2 - y_2)\, x_{2,2} \end{pmatrix}$$

$$= \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} - \lambda \begin{bmatrix} x_{1,1} & x_{2,1} \\ x_{1,2} & x_{2,2} \end{bmatrix} \begin{pmatrix} \hat{y}_1 - y_1 \\ \hat{y}_2 - y_2 \end{pmatrix}$$

Each iterative update involves matrix-vector multiplication

Mini-batch SGD with batch size m and 2-feature input vectors

Instance	Size $x_{i,1}$	Color $x_{i,2}$	Category y_i
x_1	Small 0	Red 1	Positive 1
x_2	Large 1	Red 1	Positive 1
…			
x_m	…	…	…

$$\hat{y}_i = w * x_i = w_1 x_{i,1} + w_2 x_{i,2}$$

Average loss $L(w,x)$ is $\frac{1}{m}\sum_{i=1}^{m} l(w, x_i)$

$$l(w, x_i) = (\hat{y}_i - y_i)^2$$
$$= (w_1 x_{i,1} + w_2 x_{i,2} - y_i)^2$$

SGD Update: $\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} - \frac{\lambda}{m}\sum_{i=1}^{m} \begin{pmatrix} \frac{\partial l(w, x_i)}{\partial w_1} \\ \frac{\partial l(w, x_i)}{\partial w_2} \end{pmatrix}$

Each iterative update involves addition of more instance vectors

$$= \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} - \frac{2\lambda}{m} \begin{bmatrix} x_{1,1} & x_{2,1} & \cdots & x_{m,1} \\ x_{1,2} & x_{2,2} & & x_{m,2} \end{bmatrix} \begin{bmatrix} \hat{y}_1 - y_1 \\ \cdots \\ \hat{y}_m - y_m \end{bmatrix}$$

Parallel Mini-batch SGD on GPU or a multi-core machine

- Loss function L for m instances $x=(x_1, \ldots, x_m)$ and learning rate λ. Parameter vector $w=(w_1, w_2, \ldots, w_n)$

Average loss is $L(w, x) = \frac{1}{m}\sum_{i=1}^{m} l(w, x_i)$

- Initialize all parameters
- Repeat for many epochs
  - For i = 1, 2, 3, …
    - Update weight vector

$$w = w - \lambda \nabla g(w)$$

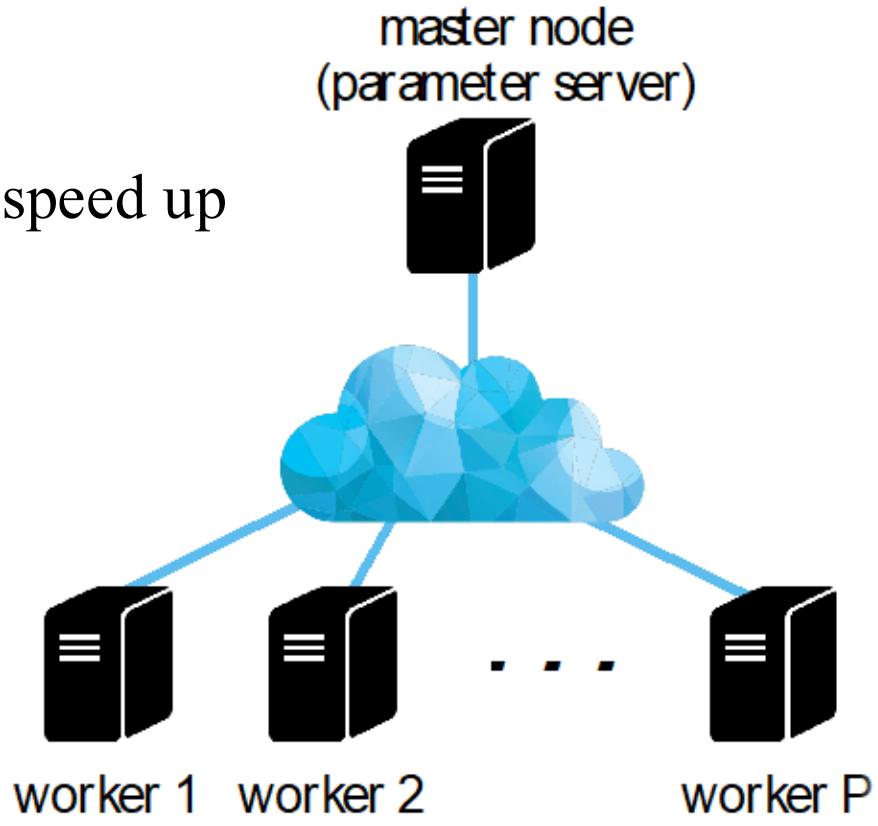
by using a batch of m instances with parallel matrix/vector multiplication on GPU or a multi-core machine

- Model training often runs many epochs to reuse instances
- Each epochs uses all instances in a training set

Parallel minibatch SGD on a cluster of machines

Large-scale machine learning involves hundreds of millions of parameters, requiring days/weeks of training computation.
How to use a cluster of machines to speed up SGD training?

- Store all parameters in a centralized server.
- Partition a minibatch into P subsets and let P workers run in parallel to compute subgradients
- Update parameters in the master server after receiving gradients from all partitions

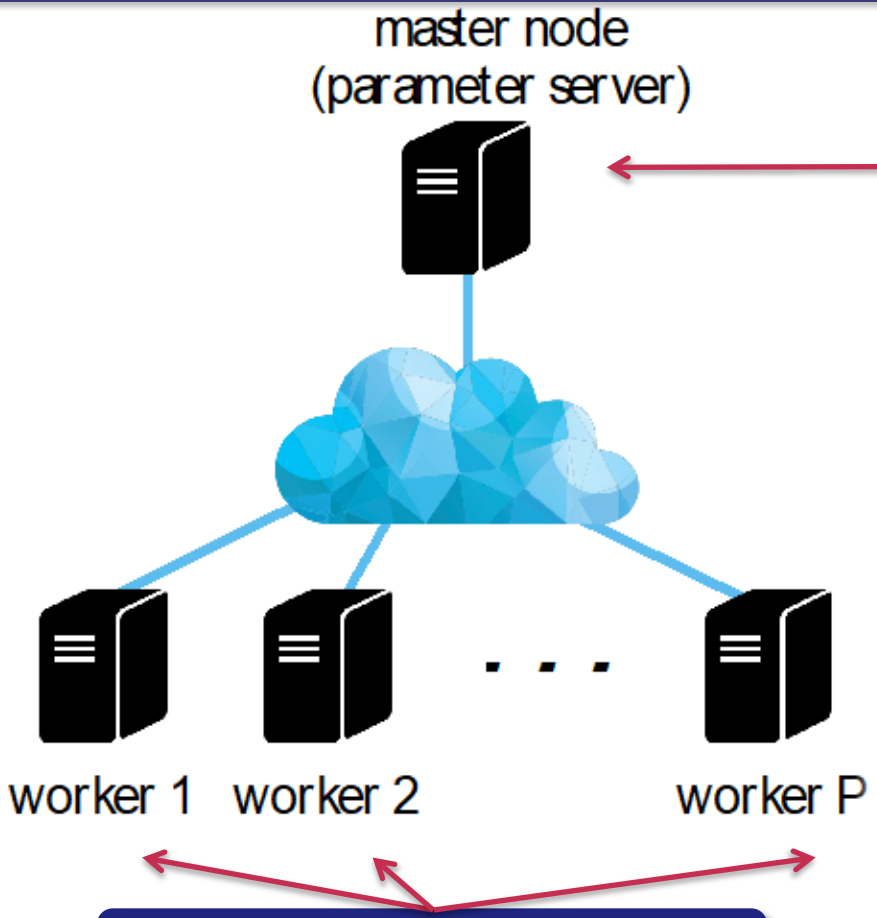


Distributed Mini-batch SGD

#Iterations= DatasetSize/BatchSize
Each epoch runs these iterations to pass through the entire training dataset

- Initialize all parameters
- Repeat many epochs
  - For i = 1, 2, 3, …
    - Master broadcasts parameters
    - P workers computes the gradients for assigned instance subsets in a batch
    - Master collects sub-set gradients from all workers
    - Master updates parameters

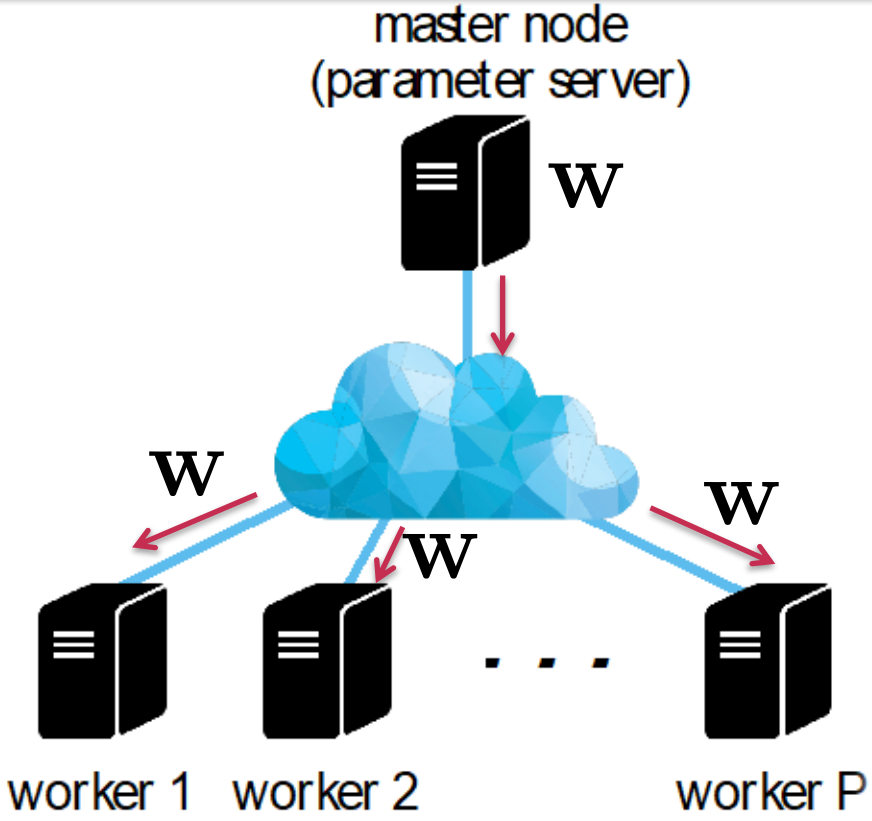
Parallel Minibatch SGD on Distributed Machines



Stores the model

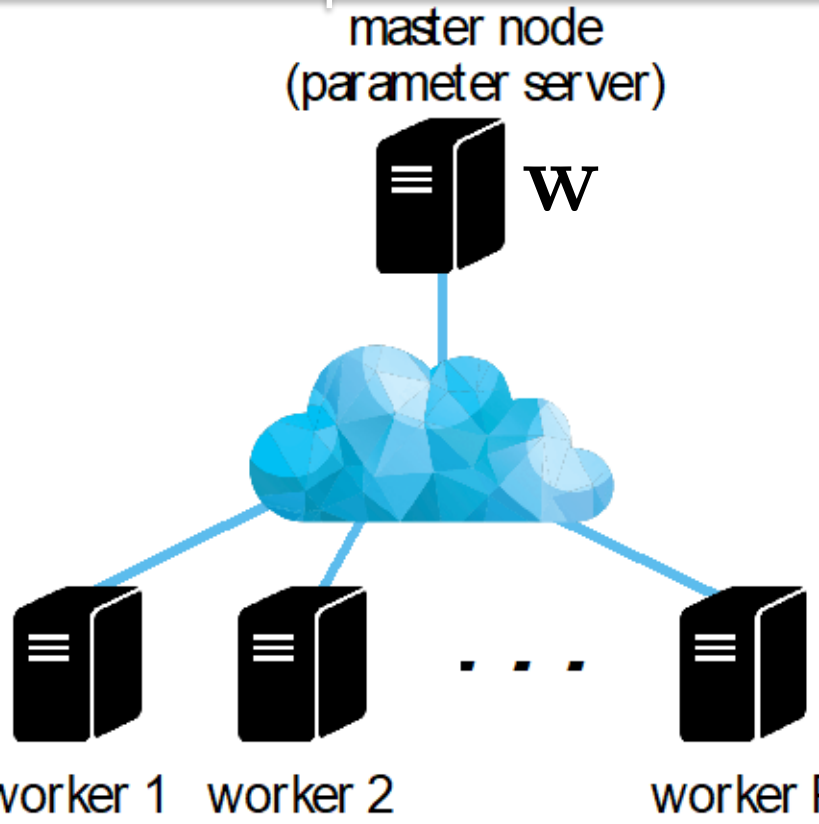
Gradient Computations

Step 1: Distribute workload to P workers



Every worker handles a subset of instances and receives the latest parameter values

Step 2: Compute gradients for all subsets in parallel

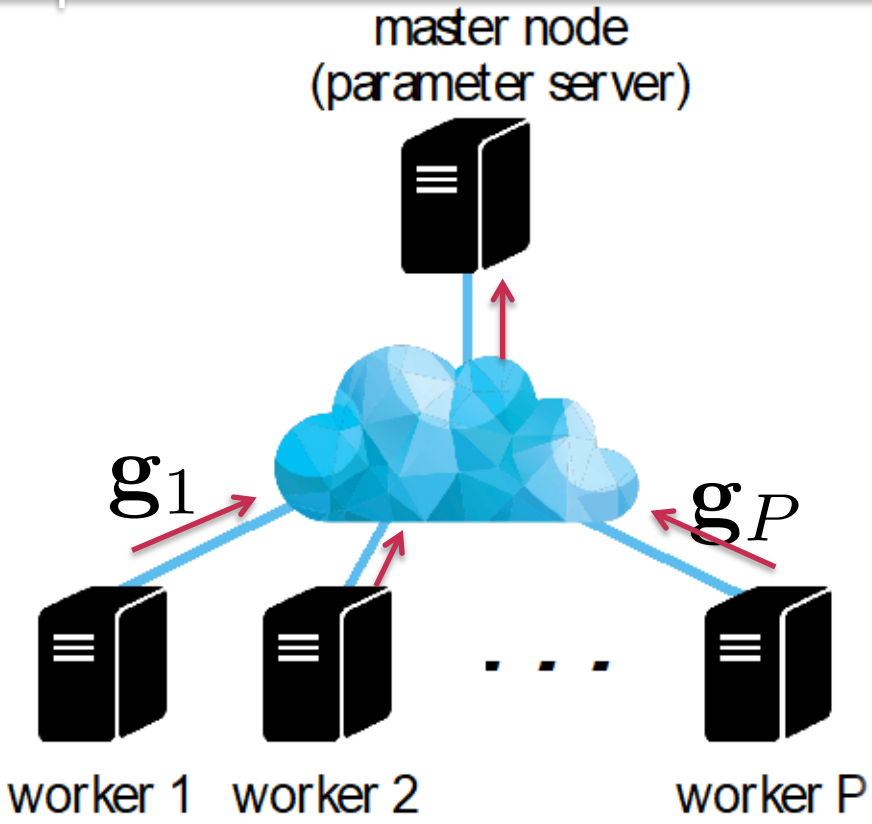


$$\mathbf{g}_1 = \sum_{i \in \mathcal{S}_1} \nabla \ell((\mathbf{x}_i, y_i); \mathbf{w})$$

$$\mathbf{g}_P = \sum_{i \in \mathcal{S}_P} \nabla \ell((\mathbf{x}_i, y_i); \mathbf{w})$$

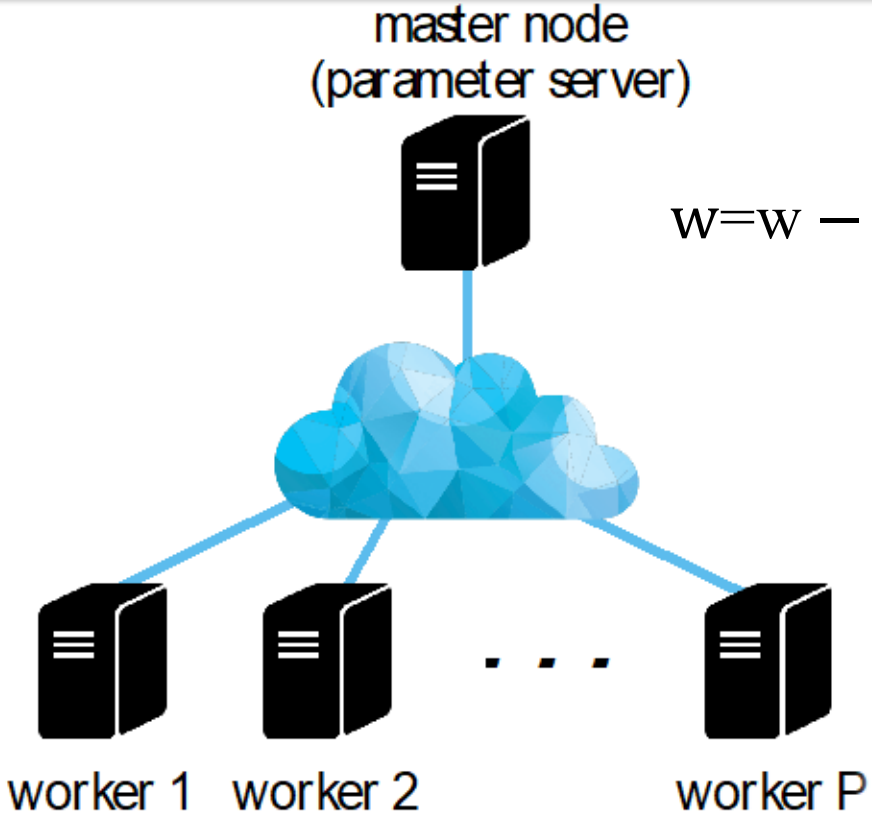
Every worker computes part of gradients

Step 3: Submit gradients to the parameter server



Every worker sends part of gradients to the parameter server

Step 4: Update weight parameters



$$w = w - \frac{\lambda}{m} \sum_{i=1}^{P} g_i$$

λ: learning rate
m: batch size

The master parameter server completes the minibatch SGD update. Restart the next minibatch after that *synchronously*.

Questions

- Does minibatch-SGD yield the same result as SGD?
- Can minibatch-SGD converge as fast as SGD?
- How can we measure speedups for the gain of parallel processing?
- Communication between a master to all workers is expensive. How to reduce? Should we increase/decrease batch size?
- What happens when some machines are slow and delay synchronized minibatch update? What can be done?

Mini-batch SGD vs SGD

Mini-batch SGD:

Update parameters with a batch of m instances at a time

SGD: m=1

Loop

Get m instances from a batch

$$\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} - \lambda \frac{1}{m} \sum_{i=1}^{m} \begin{pmatrix} \frac{\partial l(w, x_i)}{\partial w_1} \\ \frac{\partial l(w, x_i)}{\partial w_2} \end{pmatrix}$$

Use latest results	Allow use of old solutions
Gauss-Seidel	Jacobi
SGD	Mini-batch SGD

end

How to measure the benefit of parallel processing?

- Options to consider the speedup formula
  - $$\frac{\text{Time of serial minibatch SGD}}{\text{Time of parallel minibatch SGD}}$$

Measure parallel implementation efficiency if same accuracy is reached. Does not measure the accuracy tradeoff due to use of minibatch

  - $$\frac{\text{Time of serial SGD}}{\text{Time of parallel minibatch SGD}}$$

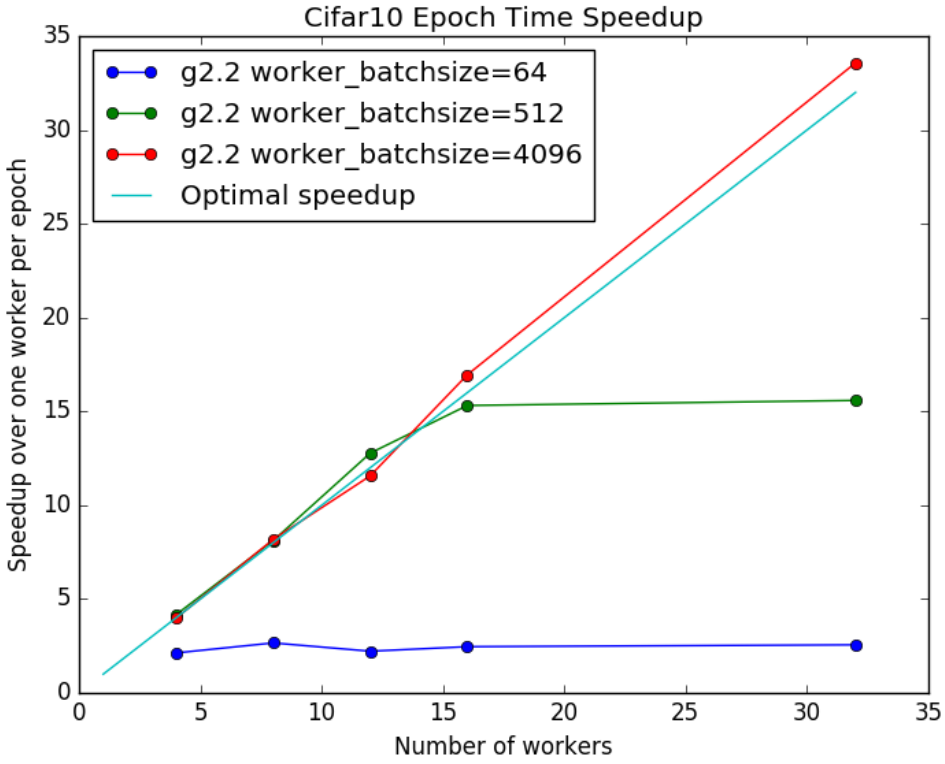
Does not consider accuracy level

  - $$\frac{\text{Time of serial SGD to reach desired accuracy}}{\text{Time of parallel minibatch SGD to reach the same accuracy}}$$

Consider both parallel implementation efficiency and algorithmic accuracy tradeoff

Impact of Batch Size in Minibatch SGD on Time per Epoch

- Speedup under different batch size for an object recognition application (Cifar-10)



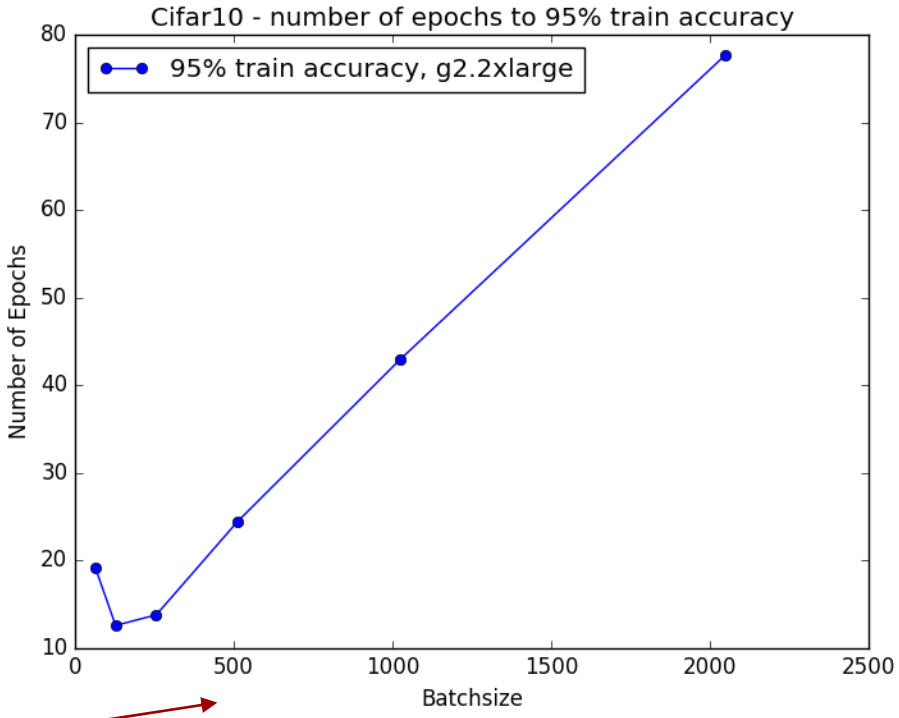
#Iterations= DatasetSize/BatchSize

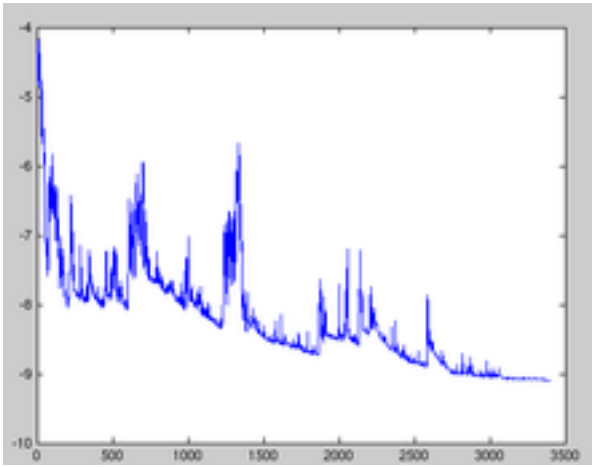
Time per Epoch=
Time for these iterations

Bigger batch size
→ Less master-worker communication overhead
→ Take an advantage of highly optimized code at each worker, (e.g. use GPU, cache).
→ Smaller time per epoch

Impact of batch size in minibatch SGD on time to reach desired accuracy

- # of Epochs needed to reach desired 95% accuracy under different batch size for an object recognition dataset





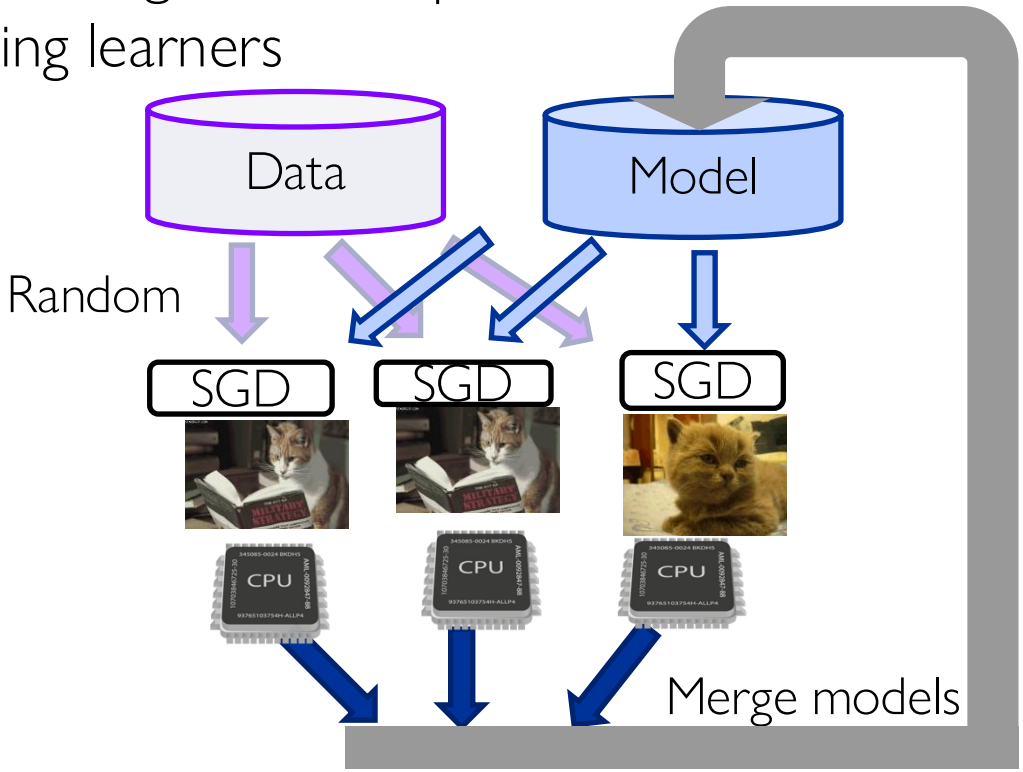
Fluctuations in the total loss objective function as gradient steps with respect to mini-batches are taken.

Bigger batch size
→ Worse training error
→ More epochs to reach desired accuracy

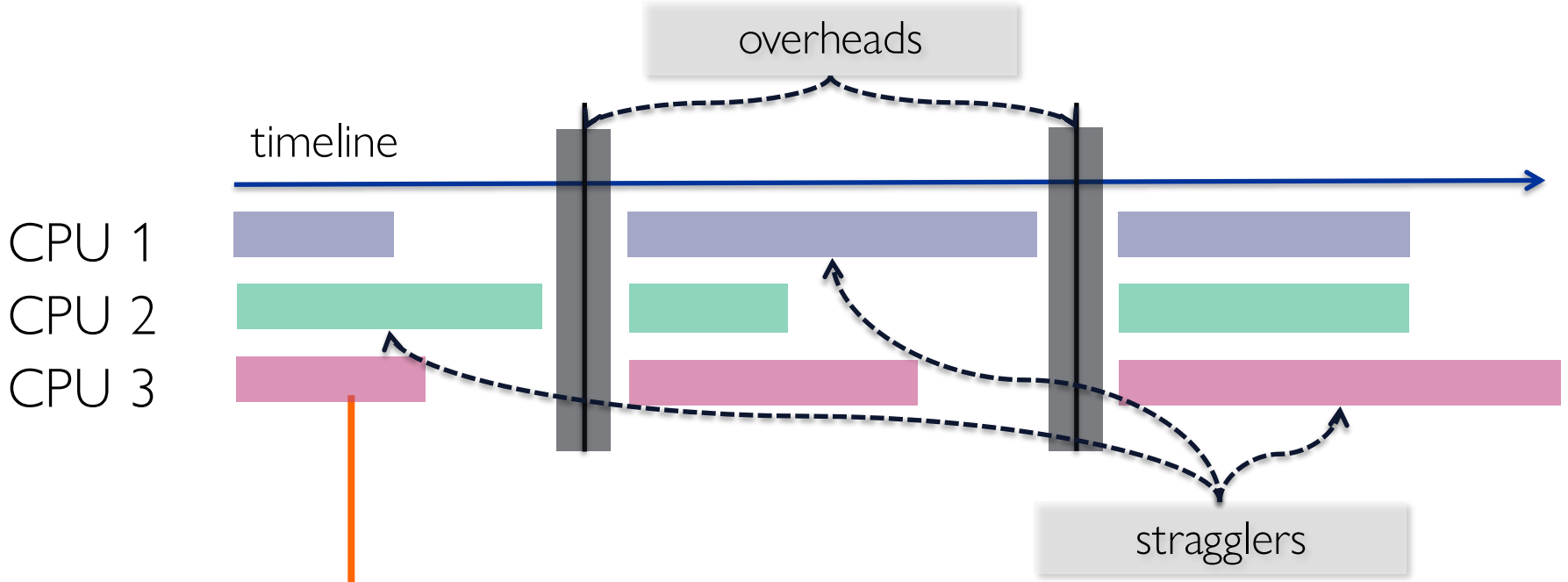
Data Dependence and Synchronization in Parallel SGD

Two bottlenecks slow down parallel SGD

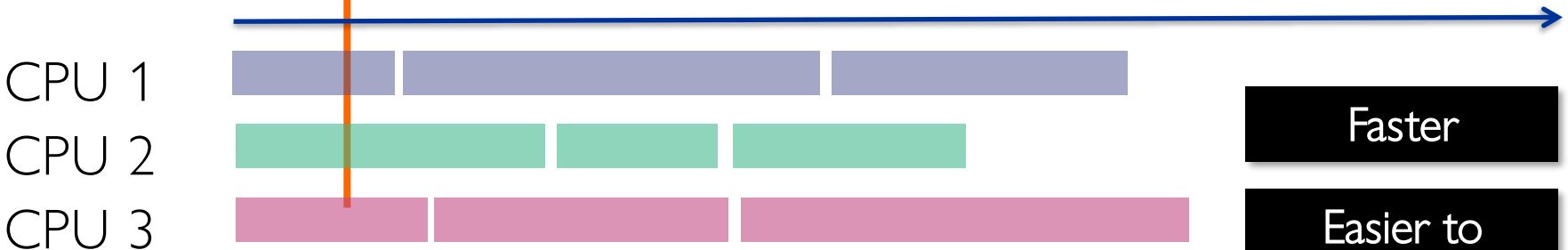
1. Data shuffling due to dependence
2. Straggling learners



A case against synchronization



Asynchronous World

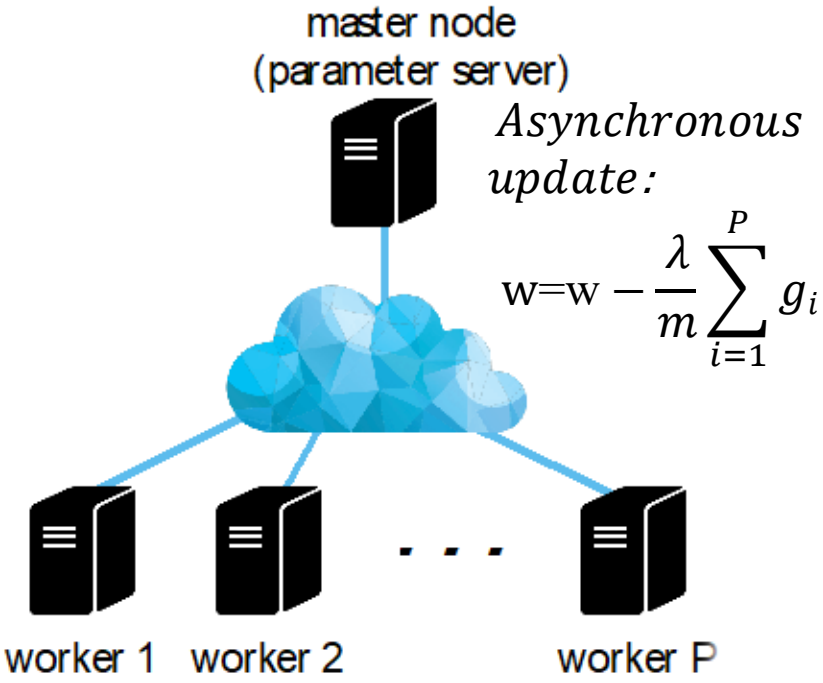


Faster

Easier to Implement

Distributed Mini-batch SGD with Asynchronous Update

- Initialize all parameters
- Repeat many epochs if desired
  - For i = 1, 2, …
- P workers computes the gradients for assigned instance subsets asynchronously using recently received parameters
- Master updates parameters as soon as receiving new gradient from a worker, and broadcasts new parameters
  - Master periodically checks the model accuracy and if it can stop training

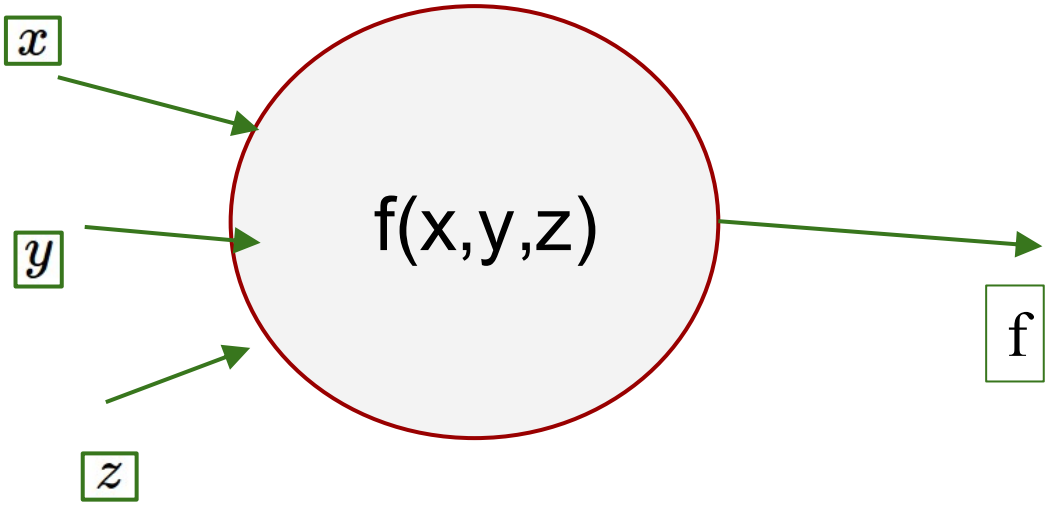


Tradeoff: less synchronization. More epochs for desired accuracy

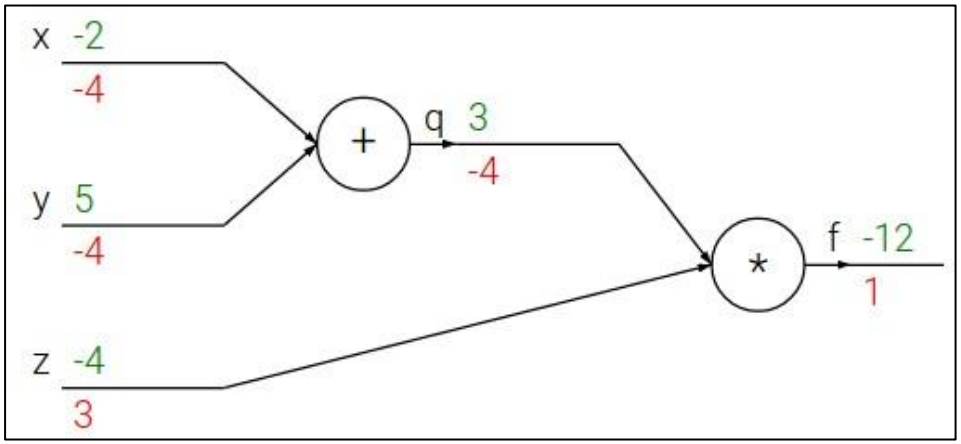
How to Compute Partial Derivatives Algorithmically of a Loss Function for SGD

Given f(x,y,z)= (x+y)z, want

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



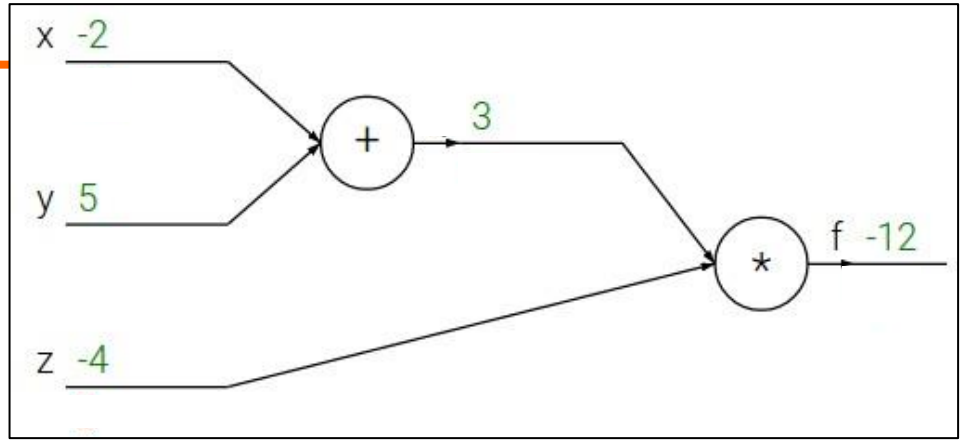
- **Use graph representation** of a loss function
- **Forward computation** to get function value
- **Backward propagation** to get partial derivates with respect to all input parameters



Example of Algorithmic Derivative Computation

- $f(x, y, z) = (x + y)z$

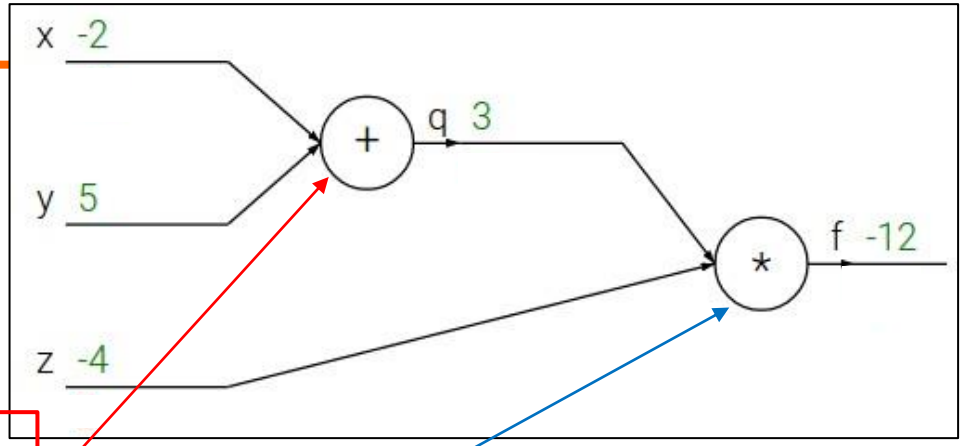
Knowing x = -2, y = 5, z = -4, q=x+y



Stage 1: Get final value f via forward computation

- $f(x, y, z) = (x + y)z$

x = -2, y = 5, z = -4, q=x+y,
f(x,y,z)=-12



Stage 2: Get local derivates for each node

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Stage 3: Conduct a backward propagation in this graph to compute

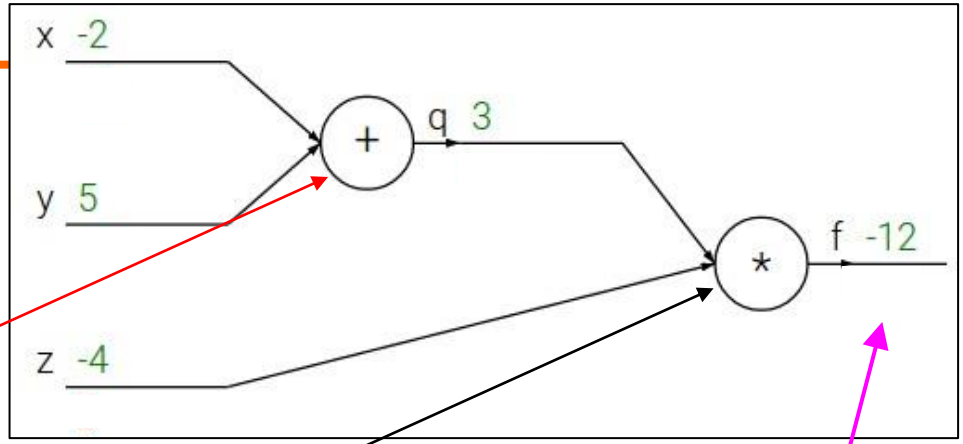
$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

Stage 3: Backward : Start from derivative of last node

$\frac{\partial f}{\partial f}$

- $f(x, y, z) = (x + y)z$

x = -2, y = 5, z = -4, q=x+y,
f(x,y,z)=-12



Local to q

$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$

Local to f

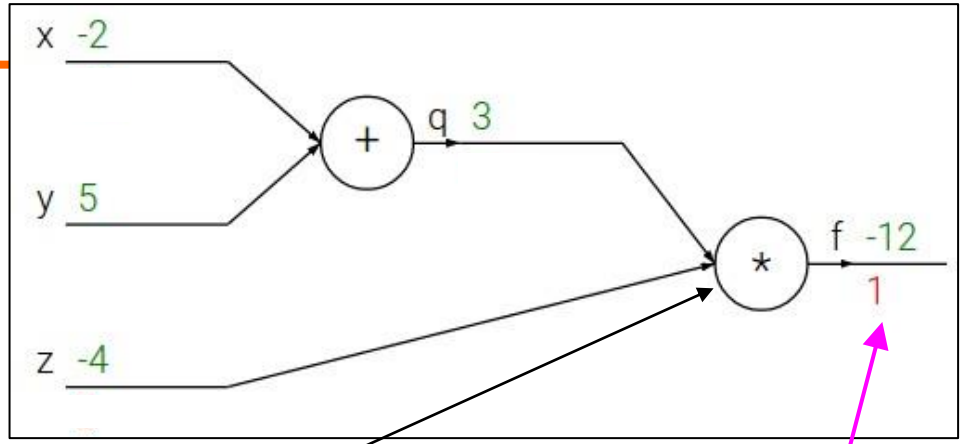
$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$

$\frac{\partial f}{\partial f}$

$\frac{\partial f}{\partial f}$ =1 as local derivative. It is trivial

- $f(x, y, z) = (x + y)z$

x = -2, y = 5, z = -4, q=x+y,
f(x,y,z)=-12



Local to f

$f = qz$ $\quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$

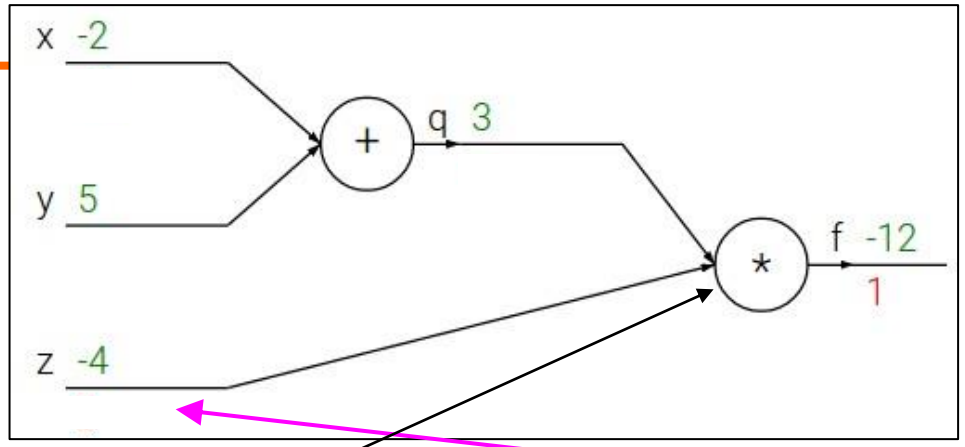
$\frac{\partial f}{\partial f}$

Need to get derivative

$\frac{\partial f}{\partial z}$

- $f(x, y, z) = (x + y)z$

x = -2, y = 5, z = -4, q=x+y,
f(x,y,z)=-12



Local to f

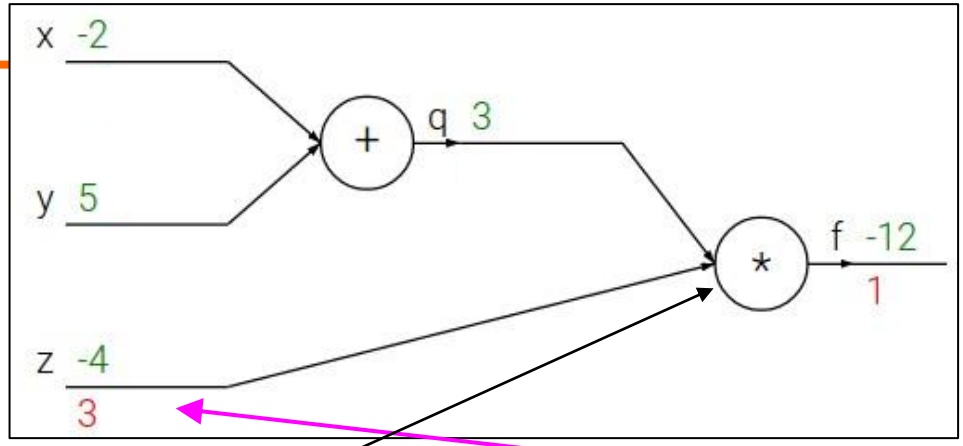
$f = qz$ $\quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$

Want: $\frac{\partial f}{\partial z}$

$\frac{\partial f}{\partial z}$ **=3** **because ∂f/ ∂z= q=3**

- $f(x, y, z) = (x + y)z$

x = -2, y = 5, z = -4, q=x+y,
f(x,y,z)=-12



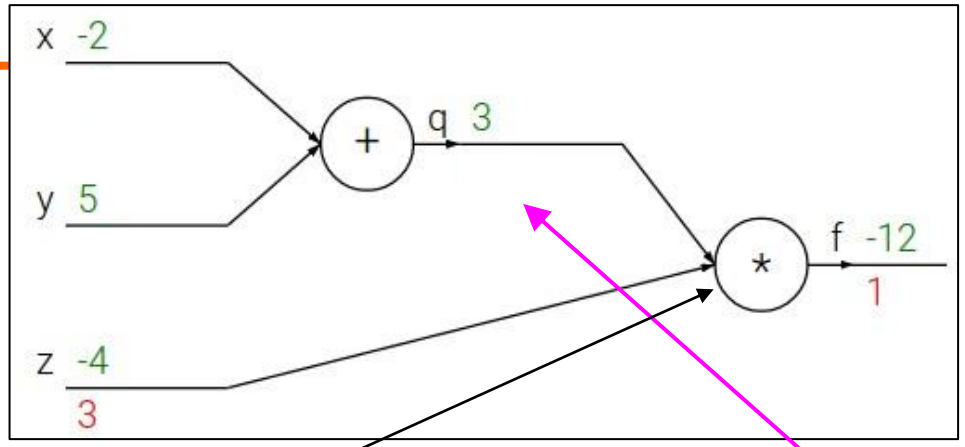
Local to f
$f = qz$ $\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$

$\frac{\partial f}{\partial z}$

Need to get derivative $\frac{\partial f}{\partial q}$

- $f(x, y, z) = (x + y)z$

x = -2, y = 5, z = -4,q=x+y,
f(x,y,z)=-12



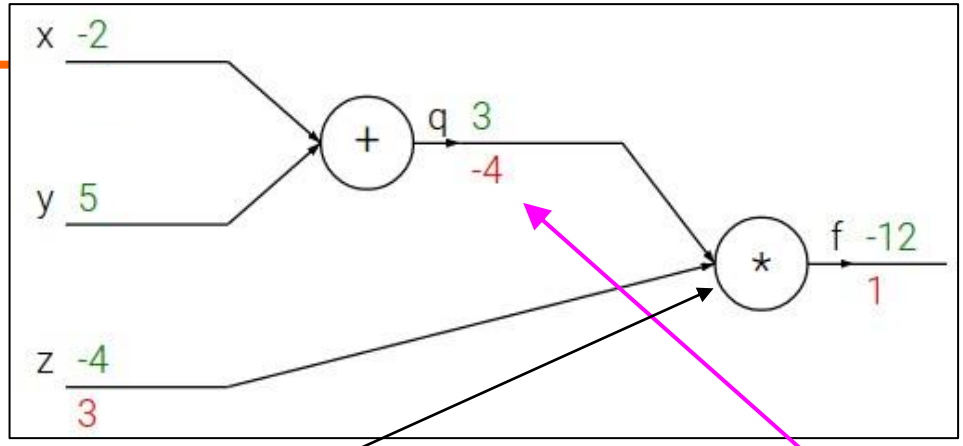
Local to f
$f = qz$ $\quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$

Want: $\frac{\partial f}{\partial q}$

$\frac{\partial f}{\partial q}$ = -4 because ∂f/ ∂q= z=-4

- $f(x, y, z) = (x + y)z$

x = -2, y = 5, z = -4, q=x+y
f(x,y,z)=-12



Local to f
$f = qz$ $\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$

$\frac{\partial f}{\partial q}$

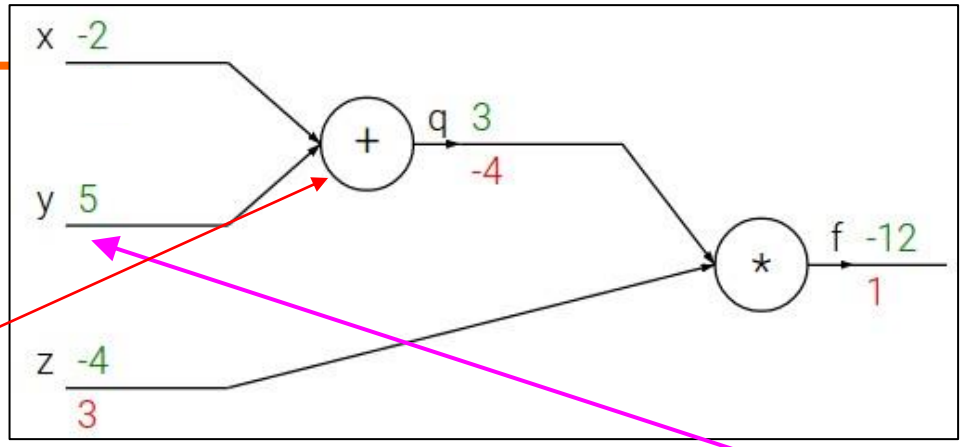
How to compute $\frac{\partial f}{\partial y}$

- $f(x, y, z) = (x + y)z$

x = -2, y = 5, z = -4, f(x,y,z)=-12

Local to q

$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$

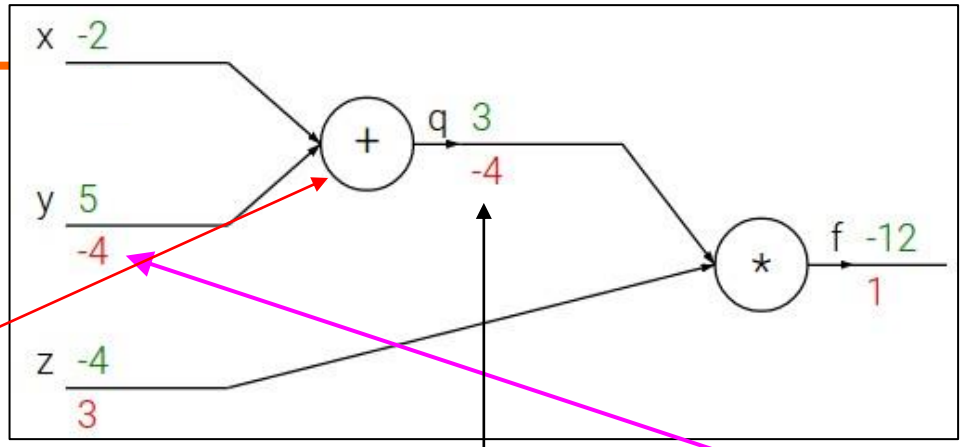


Want: $\frac{\partial f}{\partial y}$

Use the chain rule locally to compute $\frac{\partial f}{\partial y} = (-4) \cdot 1 = -4$

- $f(x, y, z) = (x + y)z$

x = -2, y = 5, z = -4, f(x,y,z)=-12



Local to q

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$\frac{\partial f}{\partial y}$

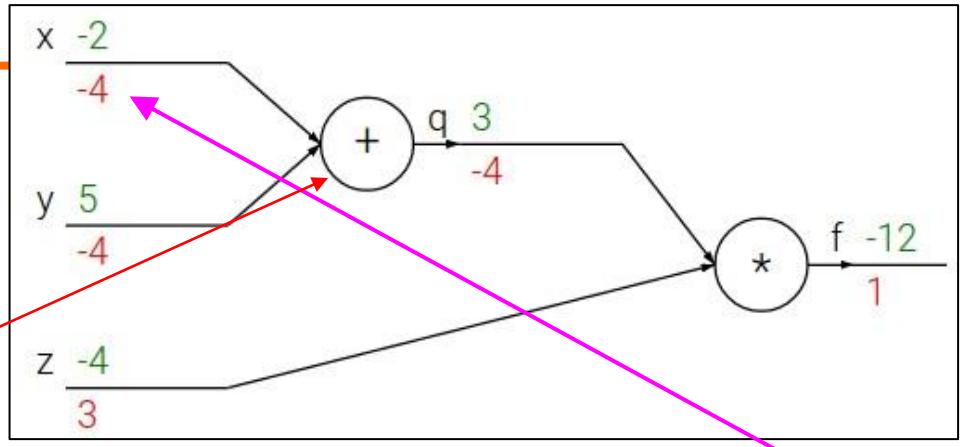
Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

Use the chain rule locally to compute $\frac{\partial f}{\partial x}$ = (-4)· 1 =-4

- $f(x, y, z) = (x + y)z$

x = -2, y = 5, z = -4, f(x,y,z)=-12



Local to q

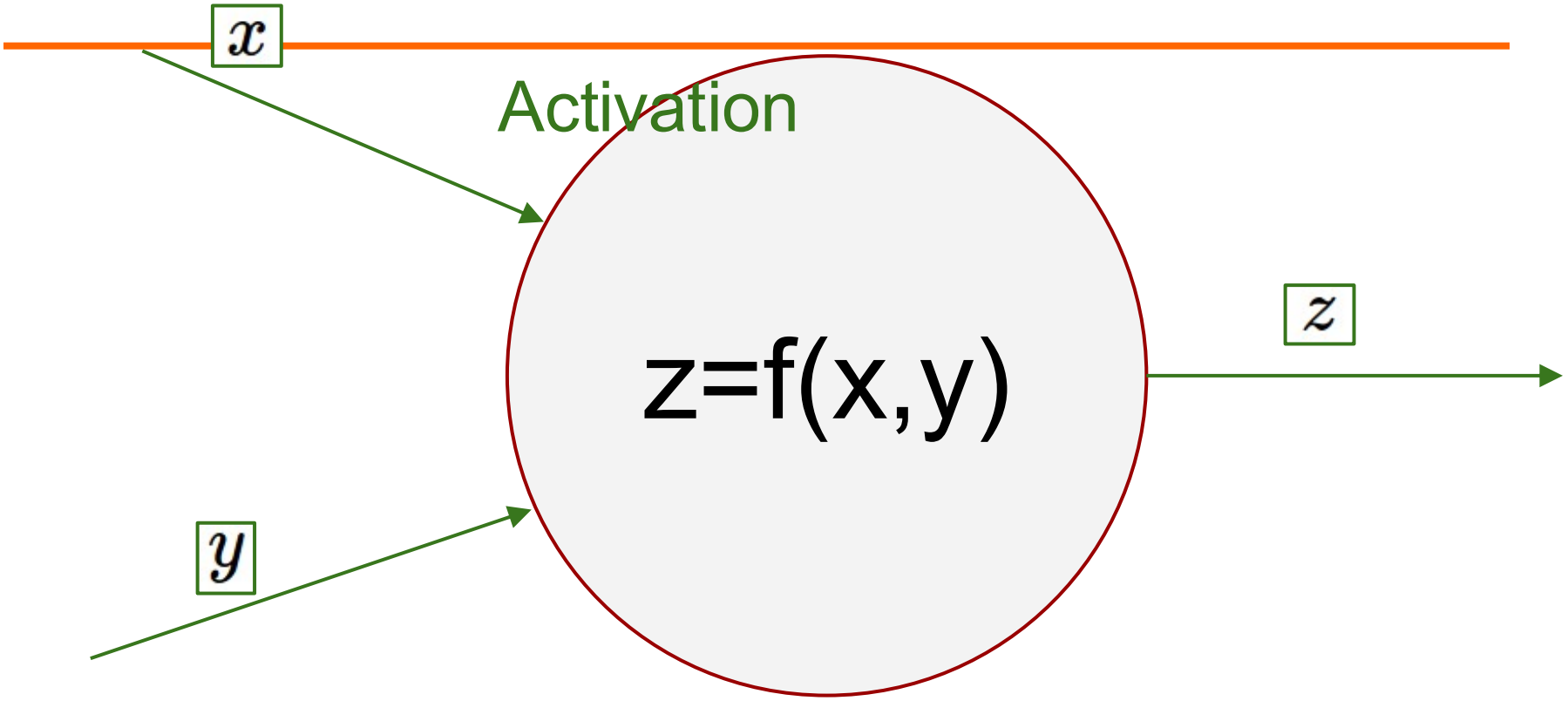
$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$

$\frac{\partial f}{\partial x}$

Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

Flow of backward propagation with chain rule



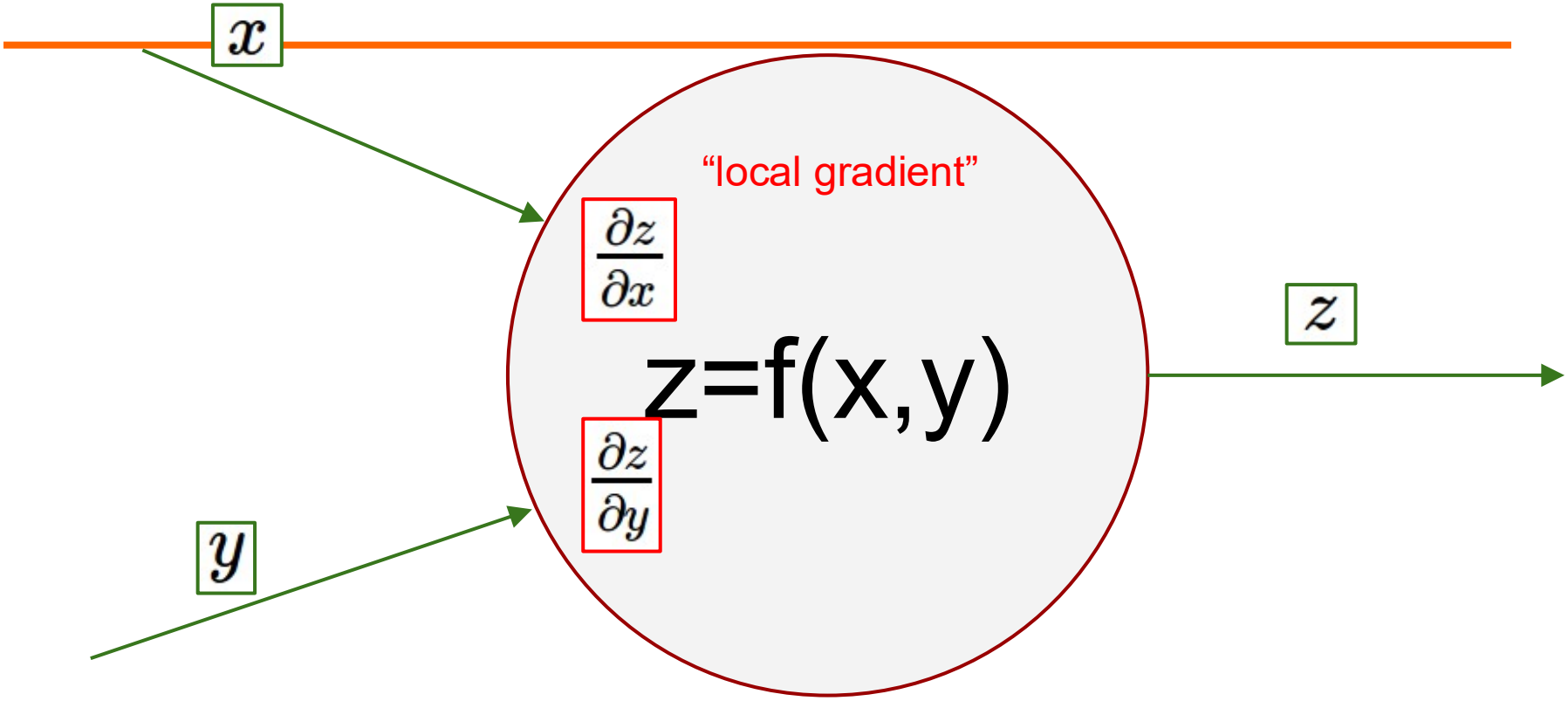
Nodes in a neural network may contain an activation function.
For example, ReLU, sigmoid gating function.

ReLu(x)=max(0,x)

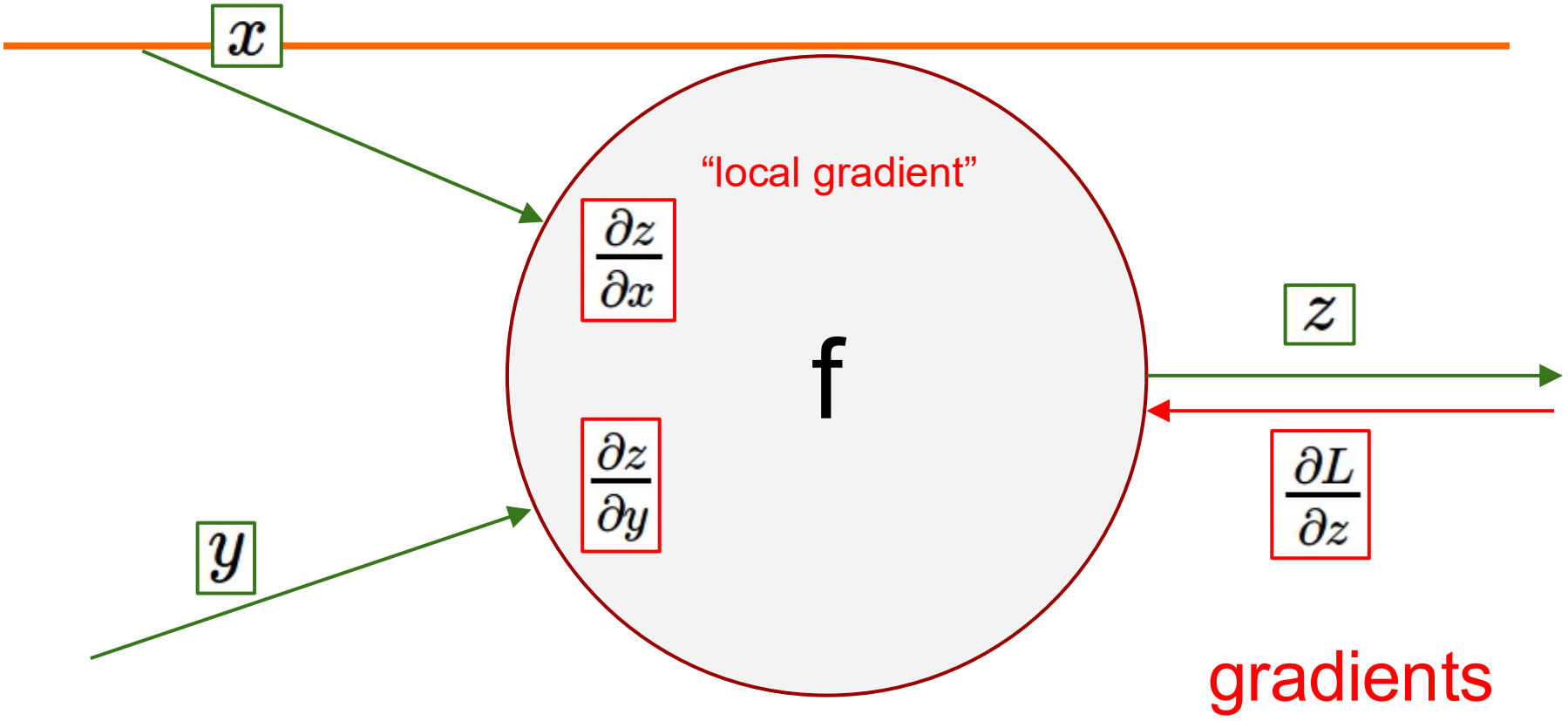
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Transform input into probability range (0,1)

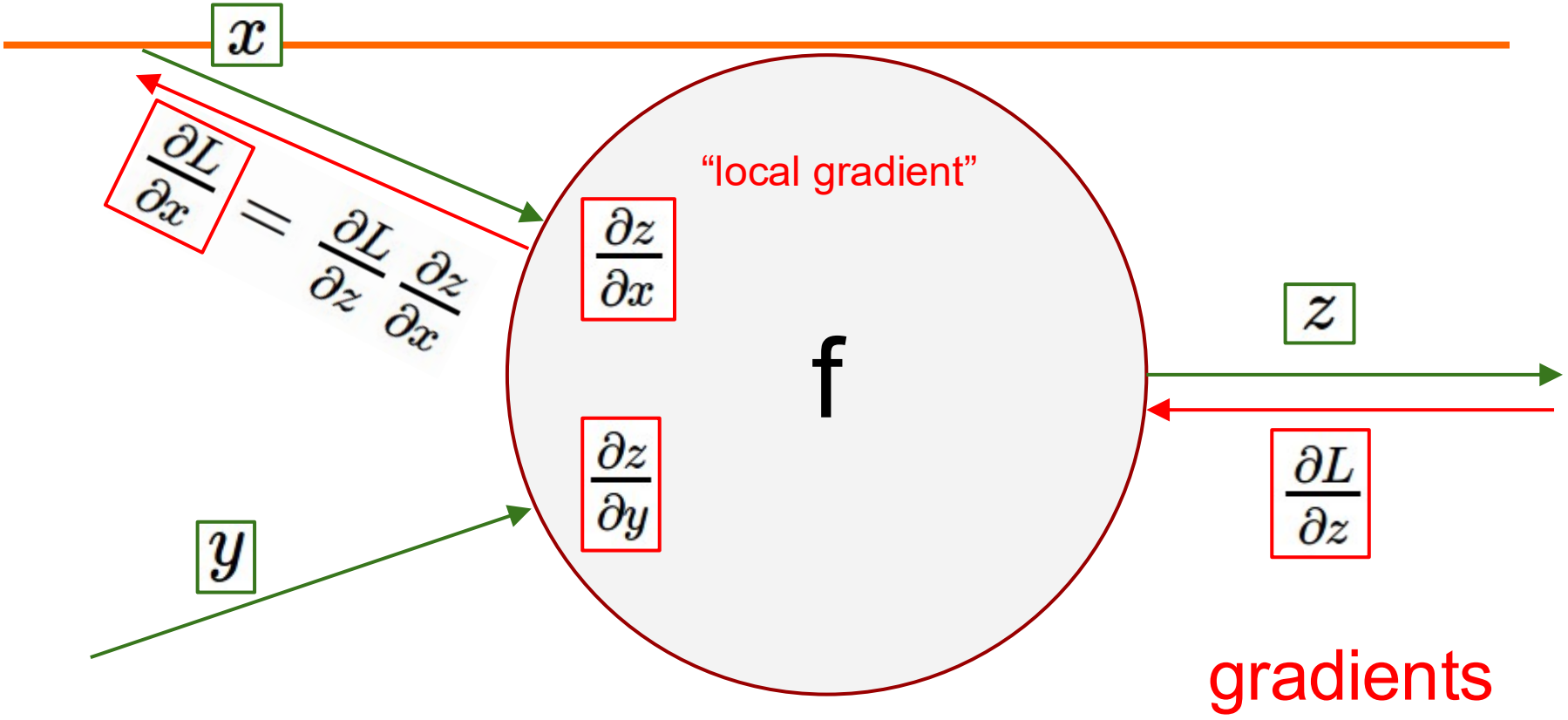
Step 1: Compute the local gradients first



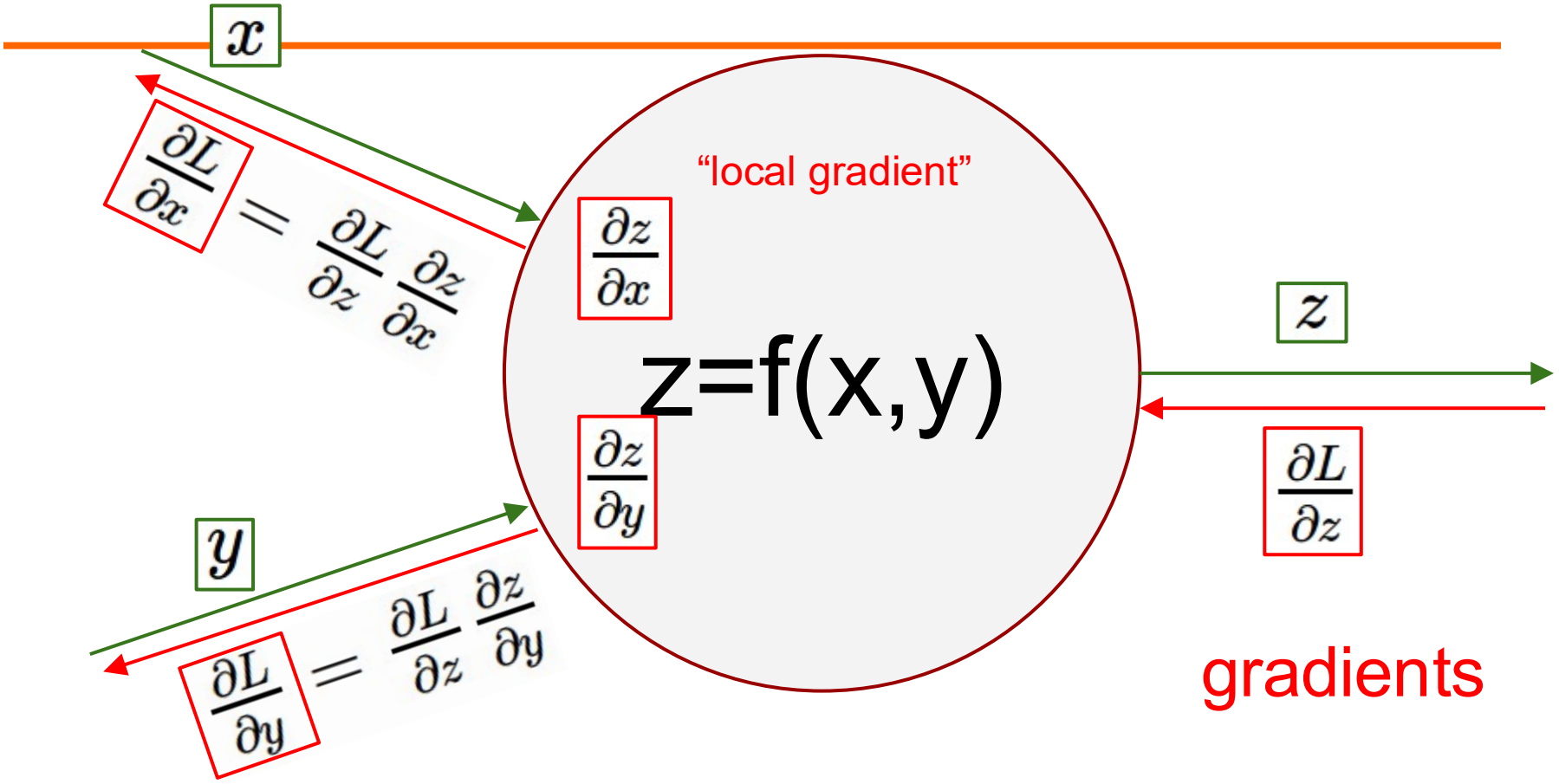
Step 2: Get the incoming gradient



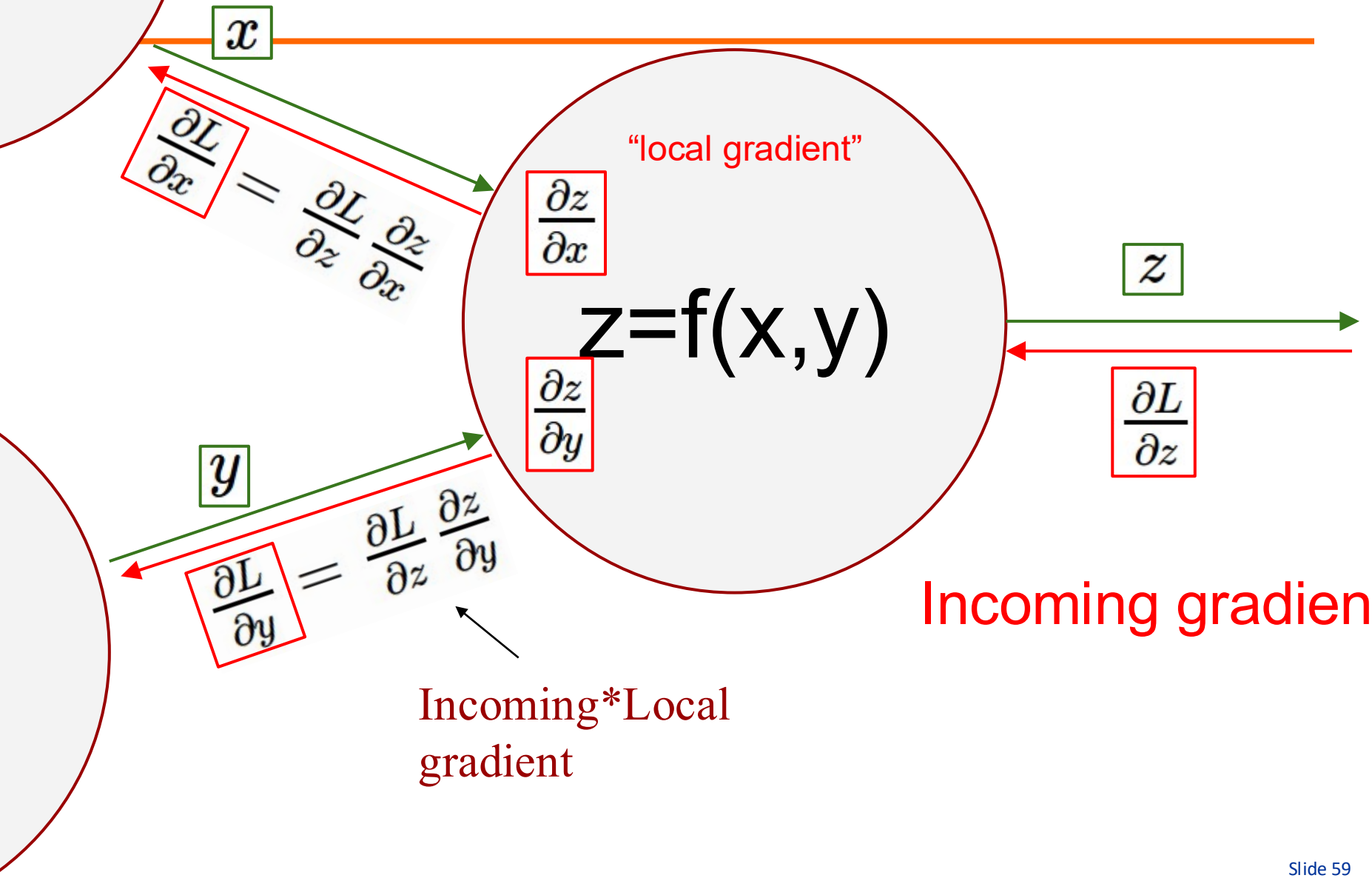
Step 3: Apply the chain rule to compute the gradient
Then propagate backward to one direction



Step 3: Apply the chain rule to compute the gradient
Propagate backward to another direction



Summary of backward flow



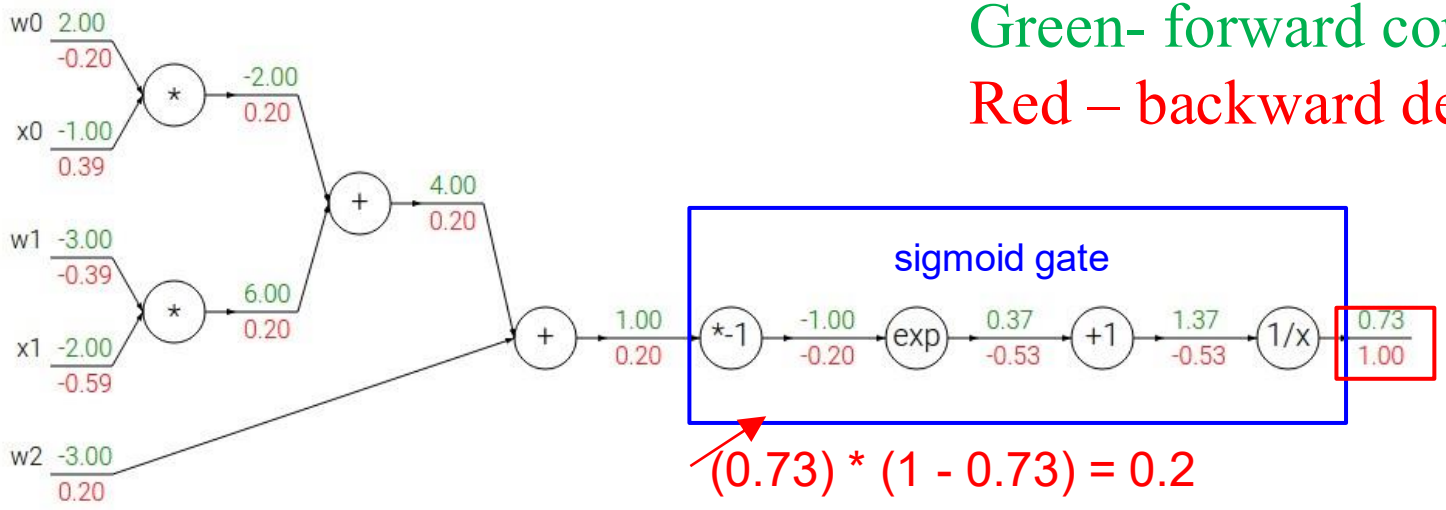
Another example: Linear combination with Sigmod

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$

Transform input into probability range (0,1)

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$



Green- forward computation

Red – backward derivatives

Parallelization Revisited for Mini-batch SGD with Many Parameters (e.g. Billions)

Primary source of parallelism in mini-batch SGD:

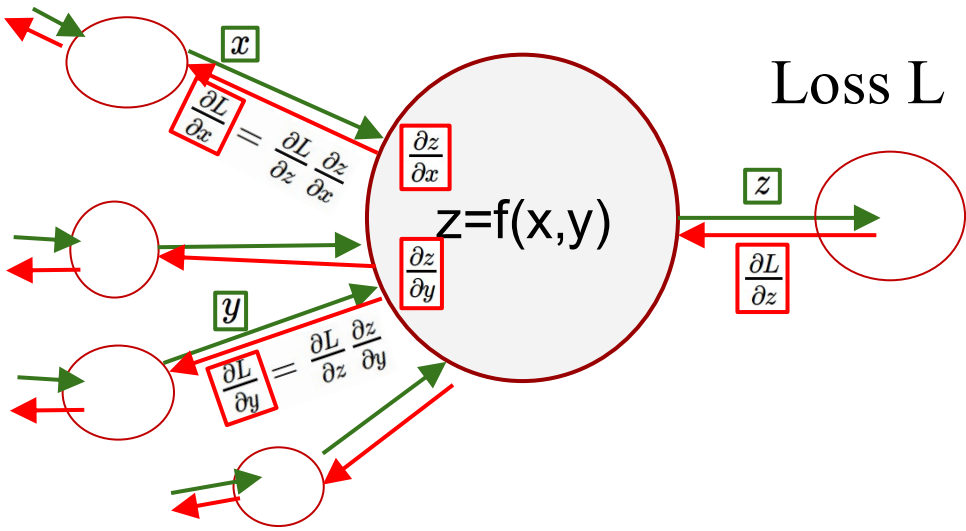
- Data independence of gradient computation among training instances
- Data parallelism in forward/backward computation among parameters

Loop

Get m instances from a batch

$$\begin{pmatrix} w_1 \\ w_2 \\ \dots \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \\ \dots \end{pmatrix} - \lambda \frac{1}{m} \sum_{i=1}^{m} \begin{pmatrix} \frac{\partial l(w, x_i)}{\partial w_1} \\ \frac{\partial l(w, x_i)}{\partial w_2} \\ \dots \end{pmatrix}$$

end



- Parallel loss/gradient computation among m instances
  - Parallel matrix-based forward computation for loss calculation (green arrows)
  - Parallel matrix-based backward propagation for gradient calculation (red arrows)

Summary on SGD

- Model training in machine learning
- Stochastic Gradient Descent (SGD) for model learning
- Mini-batch SGD with parallelizable computation
  - Iterative computation for model convergence
  - Matrix multiplication dominates each gradient update iteration
- Large-scale training with many parameters
  - Parallel/distributed mini-batch SGD
  - Distributed mini-batch SGD with asynchronous update
- Gradient computation for a loss function
  - Forward and backward propagation
  - Parallelizable