Advanced Deep Learning

Lec 3: Optimization

STAT4744/5744

Xin (Shayne) Xing • Virginia Tech

Today’s Outlines

1. Backpropagation
2. Regulations

Backpropagation

- a simple example

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

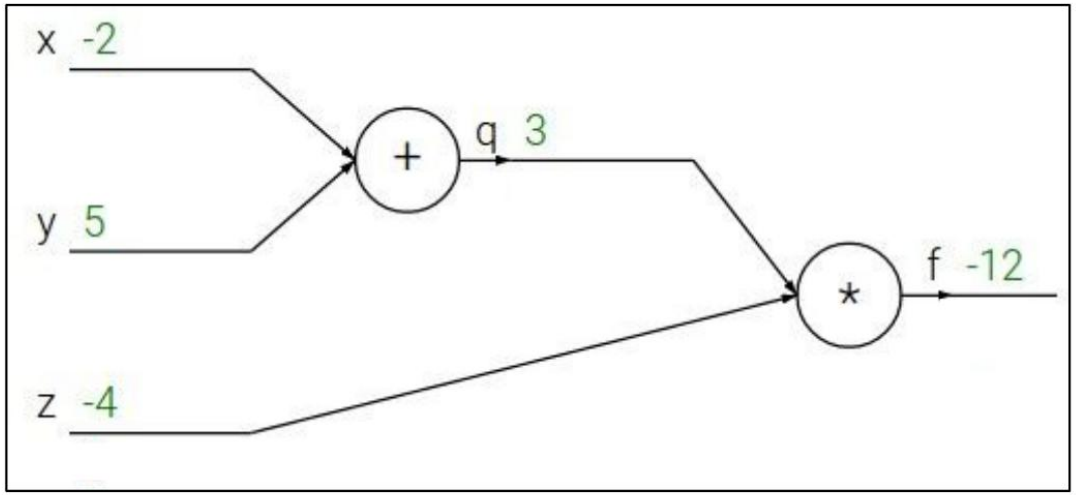
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Graph Representation

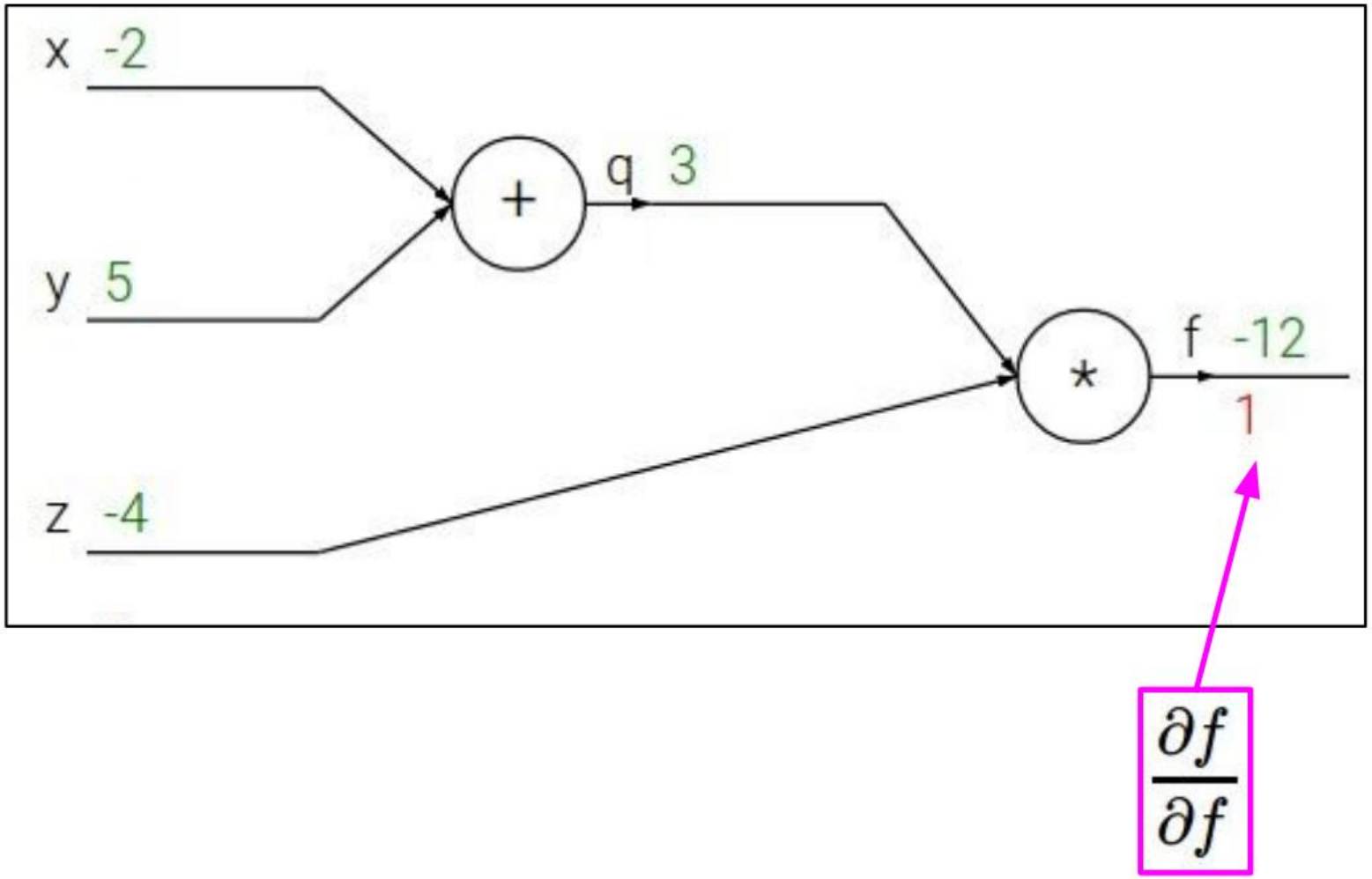
$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

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

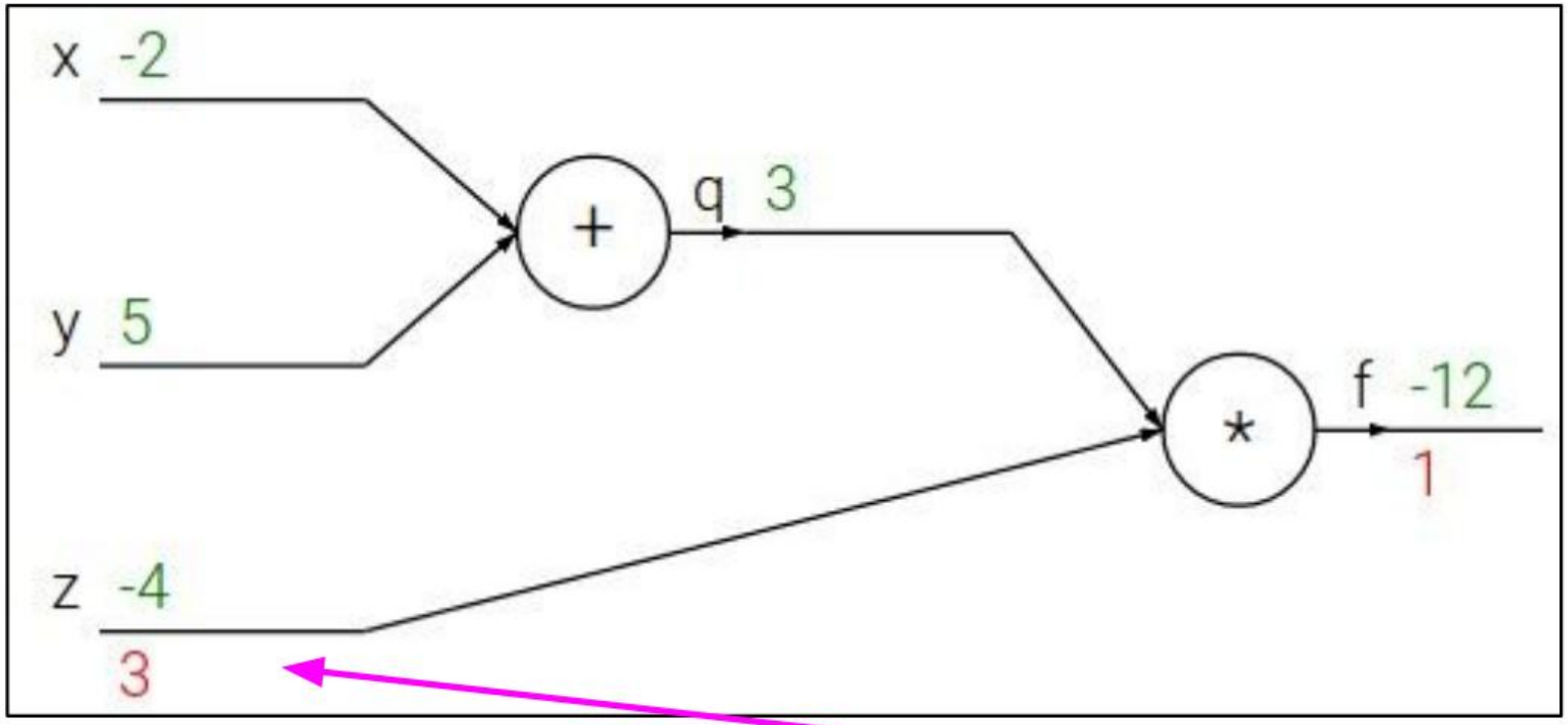
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Top-Down Calculation 1

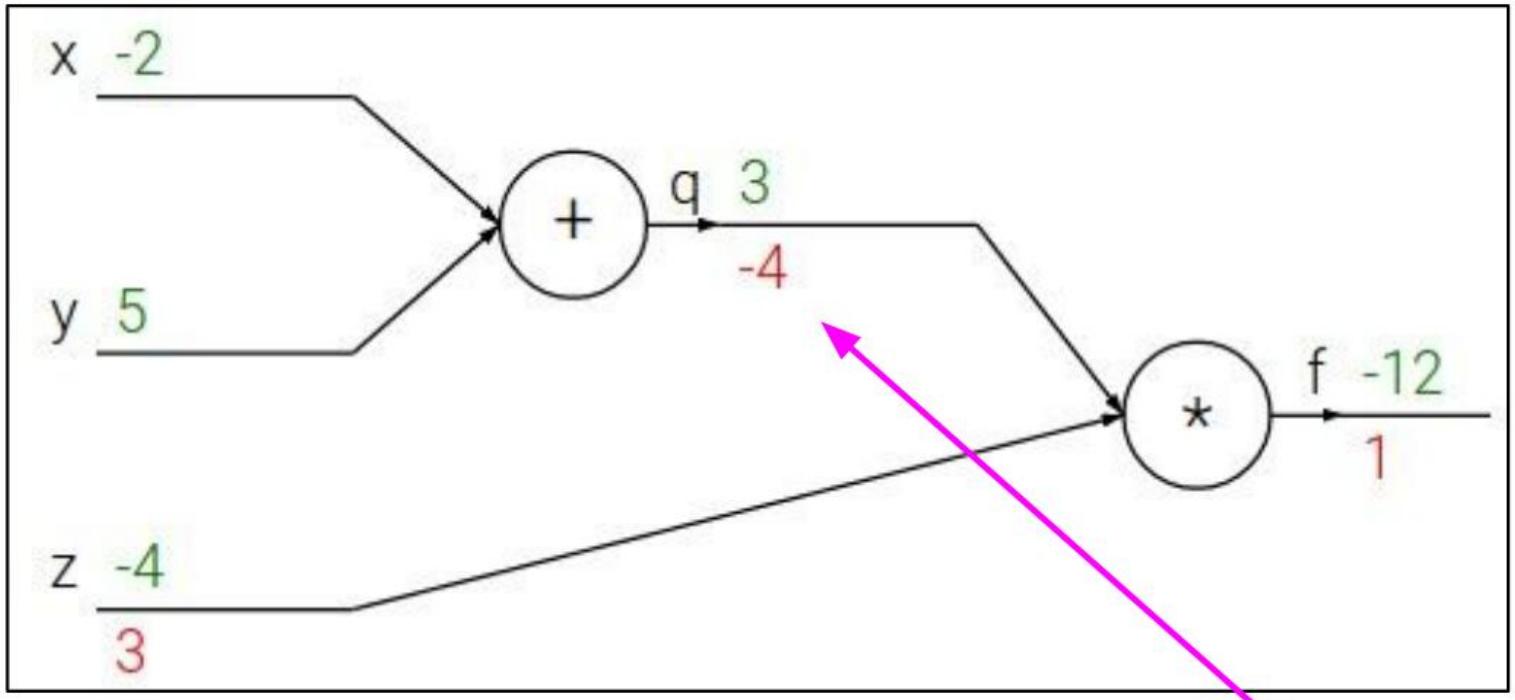


Top-Down Calculation 2



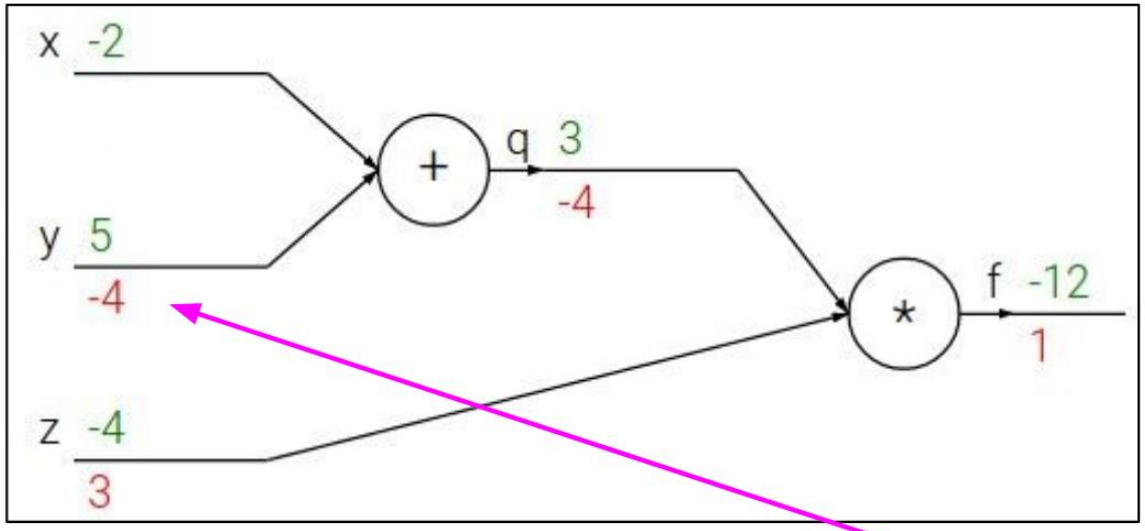
$$\frac{\partial f}{\partial z} = q$$

Top-Down Calculation 3



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

Top-Down Calculation 4



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

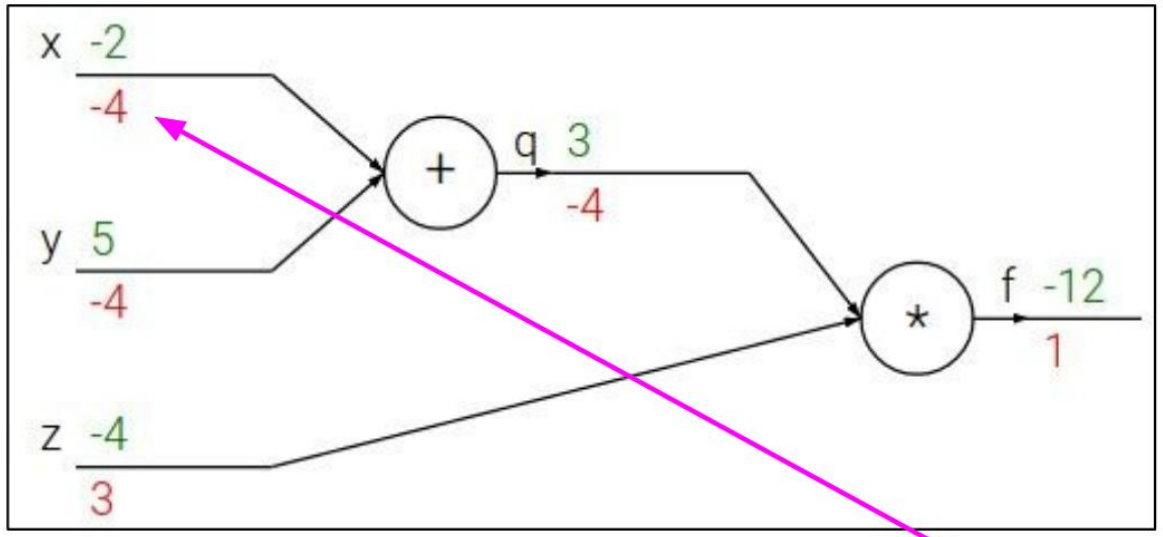
Chain rule:

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

Upstream gradient

Local gradient

Top-Down Calculation 5



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

Chain rule:

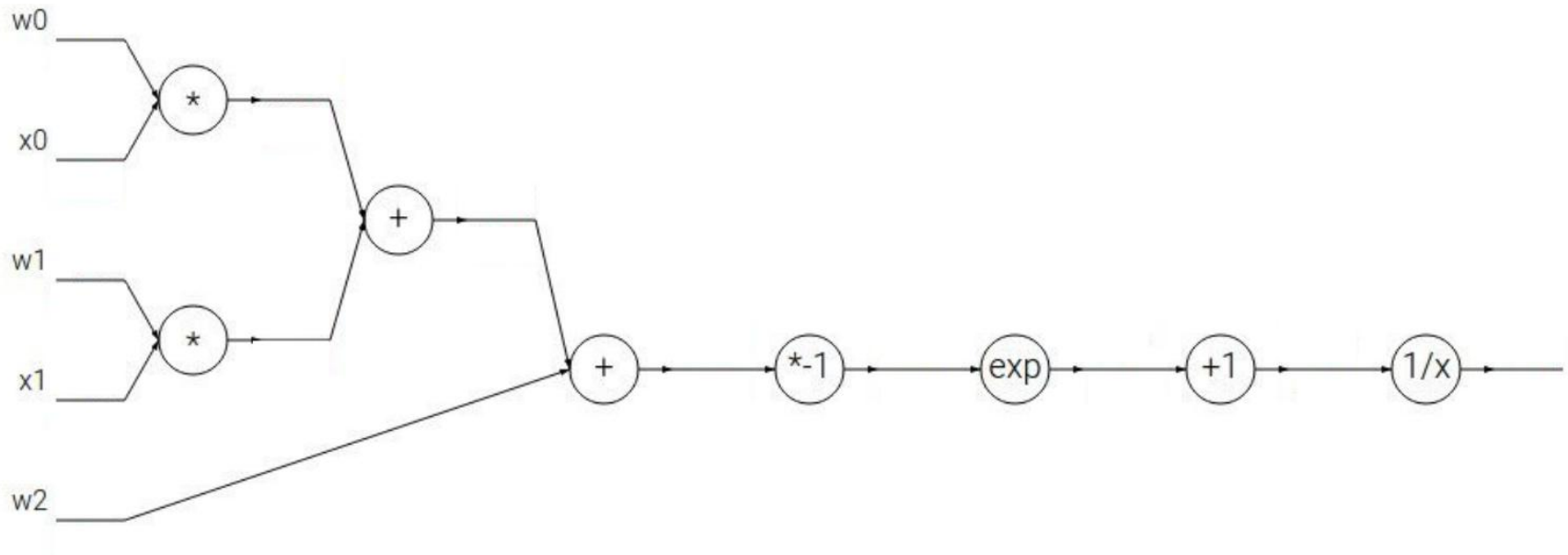
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

Upstream gradient

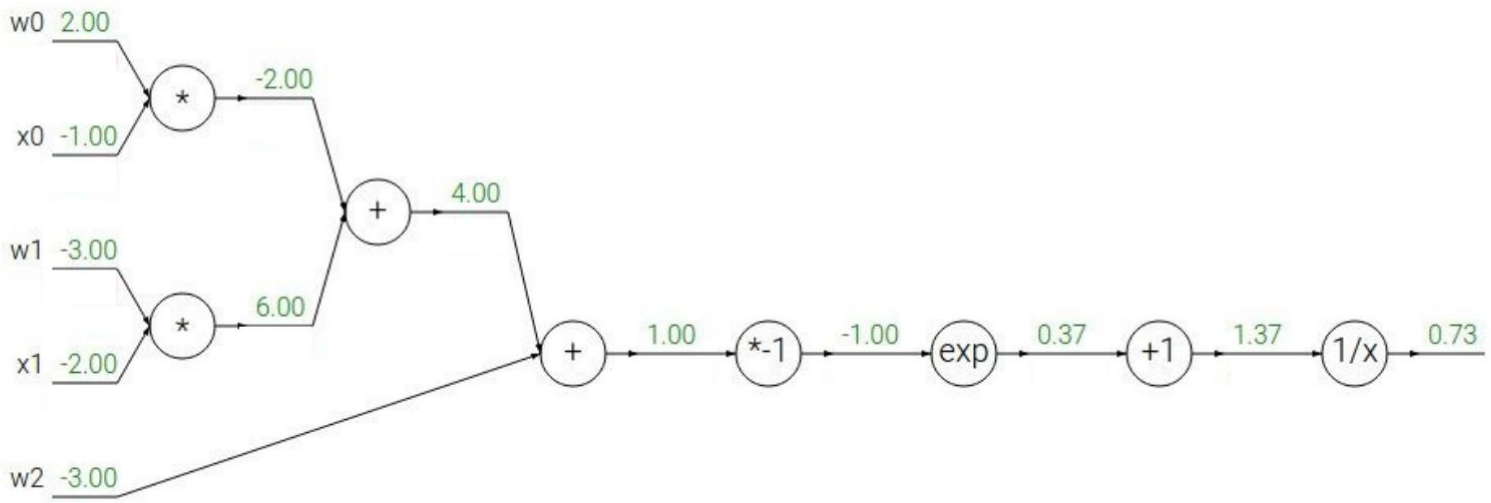
Local gradient

Another Example

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



Graph representation



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

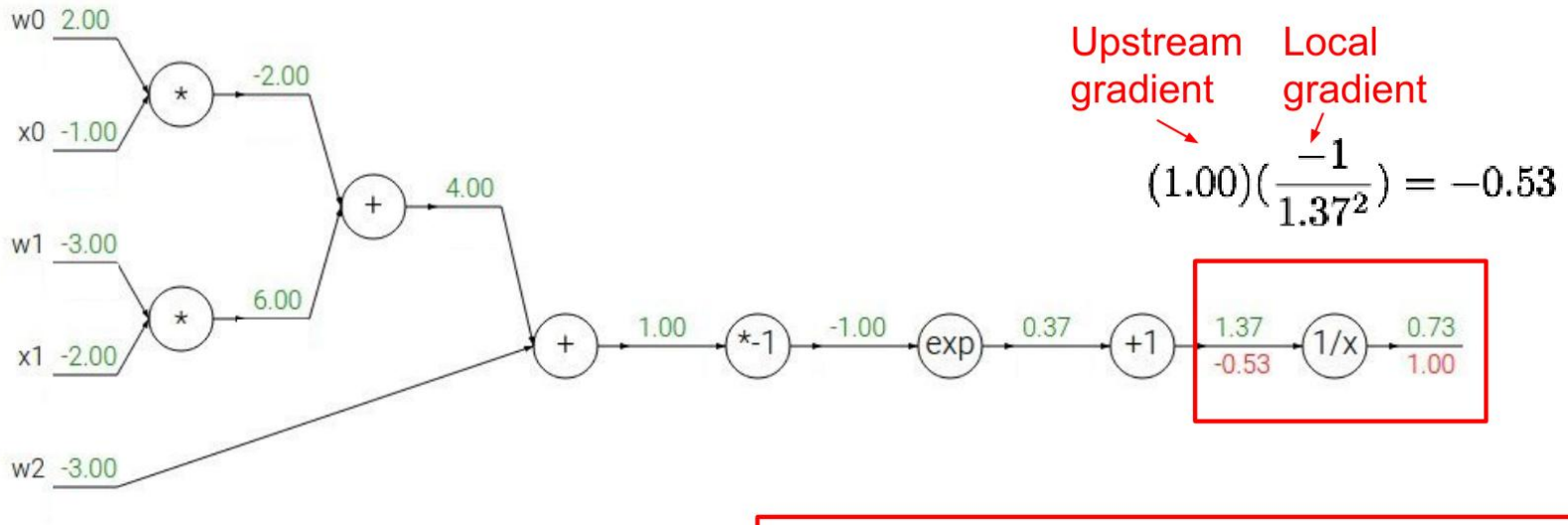
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Top-Down Calculation 1

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



$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

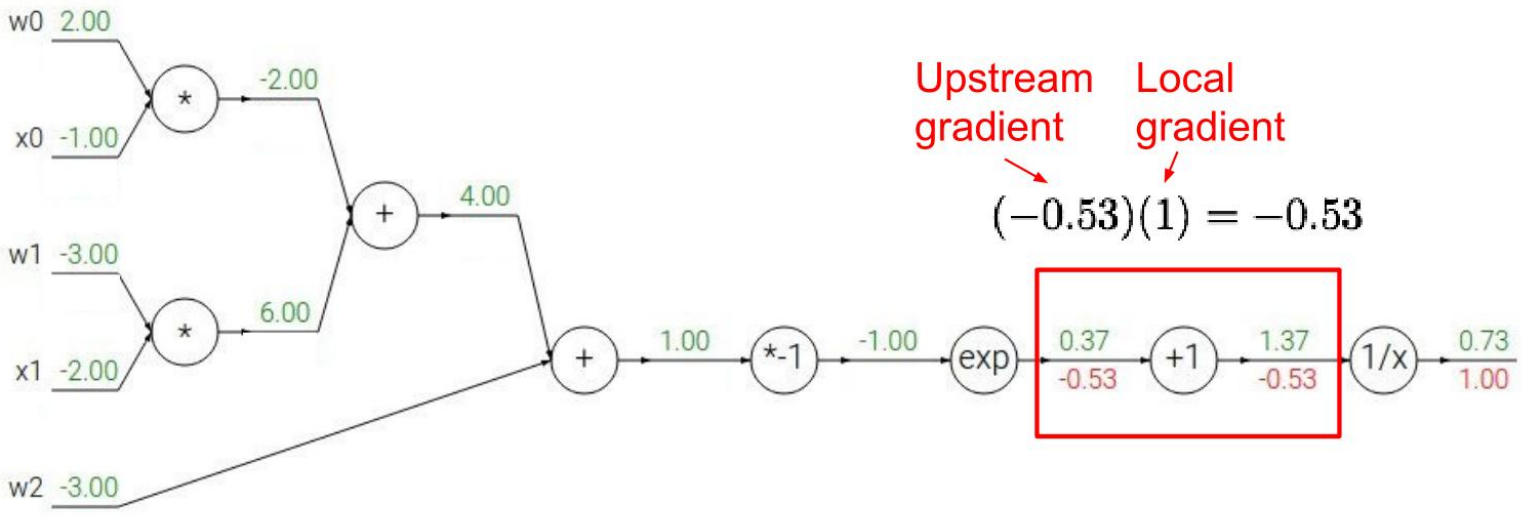
$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

Top-Down Calculation 2

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



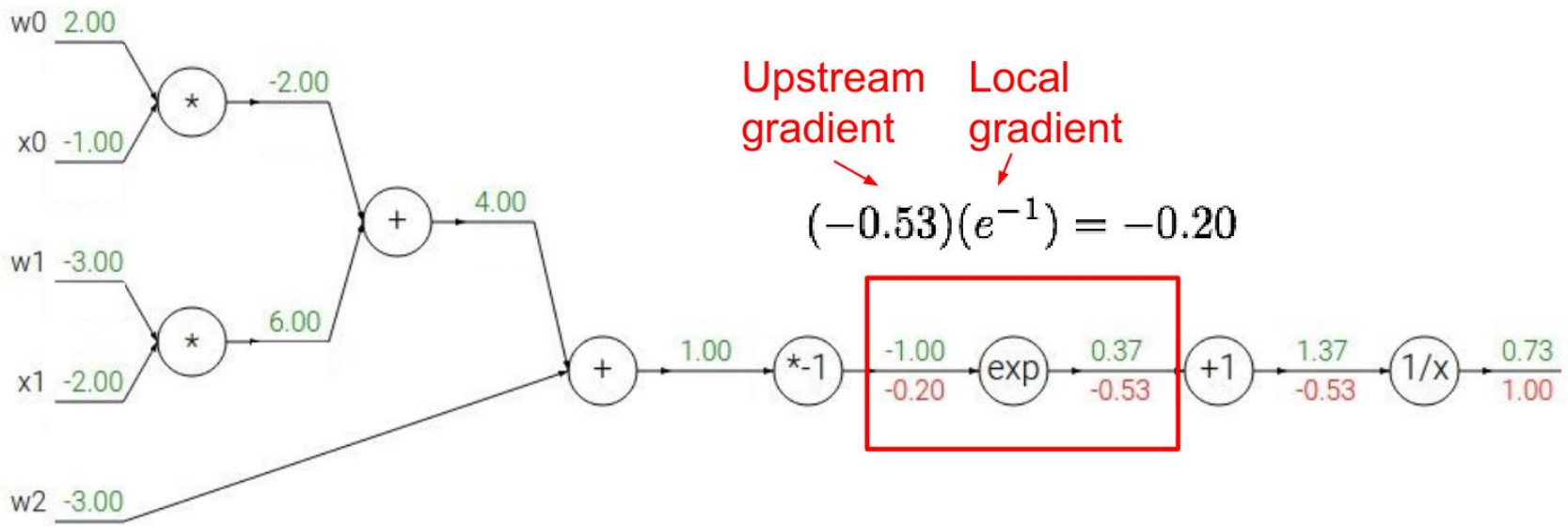
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Top-Down Calculation 3



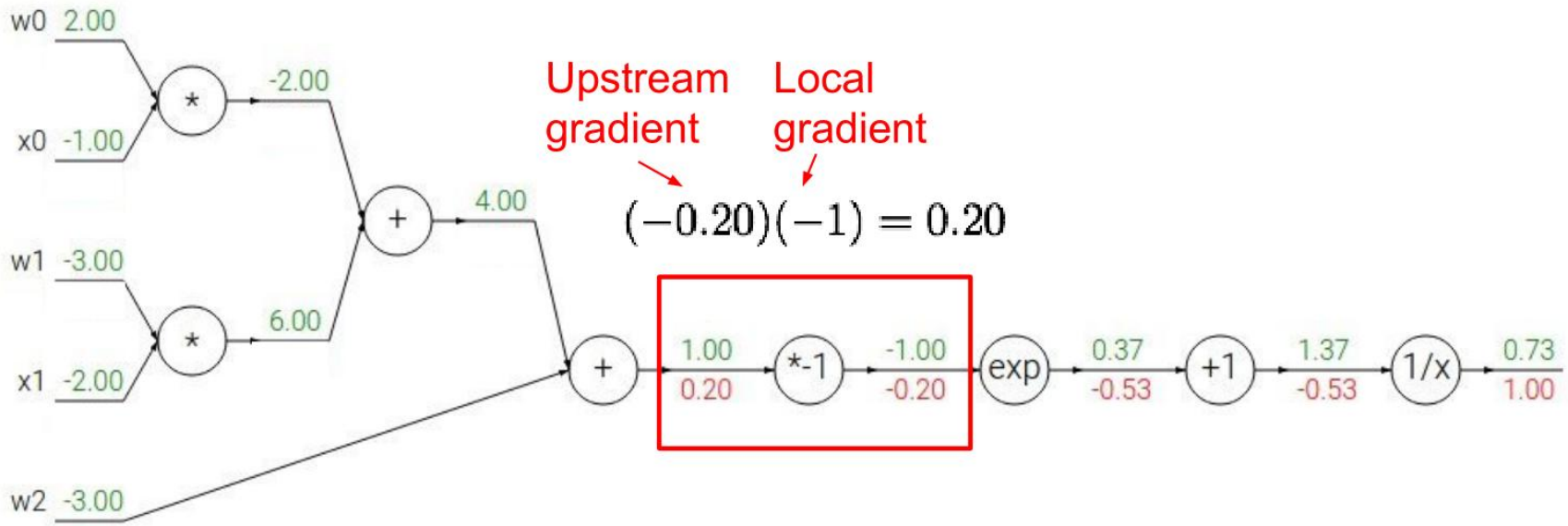
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Top-Down Calculation 4



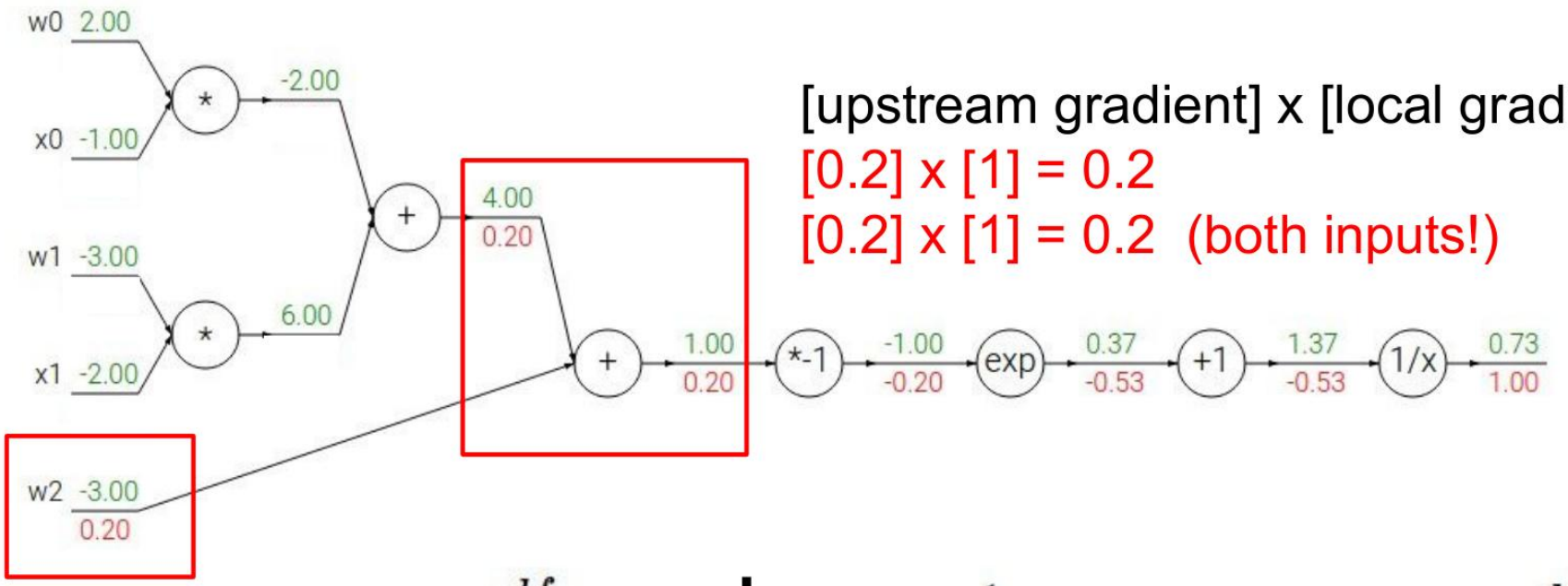
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Top-Down Calculation 5



[upstream gradient] x [local gradient]

[0.2] x [1] = 0.2

[0.2] x [1] = 0.2 (both inputs!)

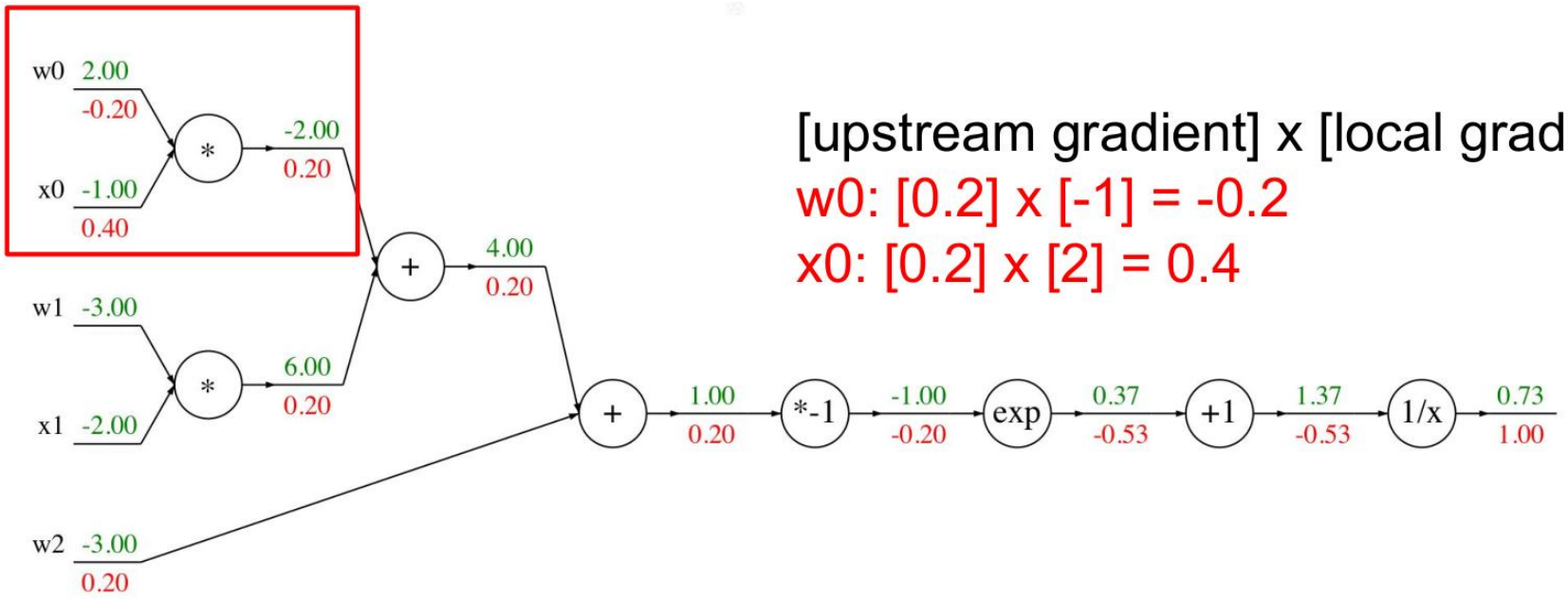
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Top-Down Calculation 6



[upstream gradient] x [local gradient]

w0: [0.2] x [-1] = -0.2

x0: [0.2] x [2] = 0.4

$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Loss Function

- A loss function tells how good our current classifier is

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where x_i is image and
y_i is (integer) label

Loss over the dataset is a
average of loss over examples:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

Computing the Gradient

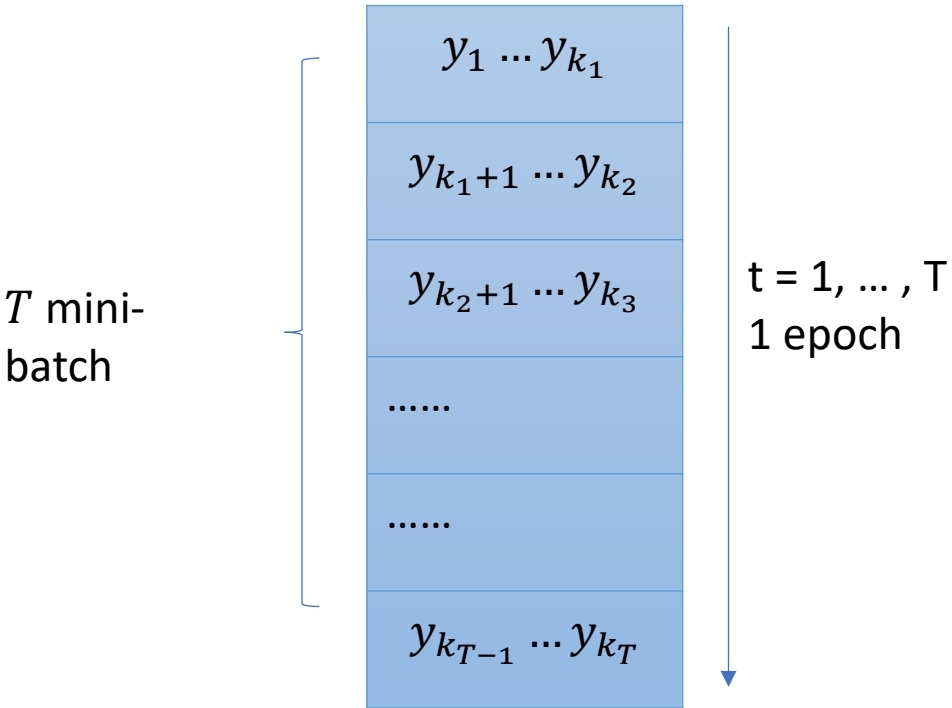
$$L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$

Full sum expensive when N is large!

Stochastic Gradient Decent with Mini-batch

- In large-scale applications, the training data can have on order of millions of examples. Hence, it seems wasteful to compute the full loss function over the entire training set in order to perform only a single parameter update



$$L_t(W) = \frac{1}{k_t - k_{t-1}} \sum_{i=k_{t-1}+1}^{k_t} L_i(x_i, y_i, W)$$

$$\nabla L_t(W) = \frac{1}{k_t - k_{t-1}} \sum_{i=k_{t-1}+1}^{k_t} \nabla_W L_i(x_i, y_i, W)$$

$$W_{t+1} = W_t + \eta \nabla L_t(W_t)$$

Step Size

- **Effect of step size**. The gradient tells us the direction in which the function has the steepest rate of increase, but it does not tell us how far along this direction we should step.

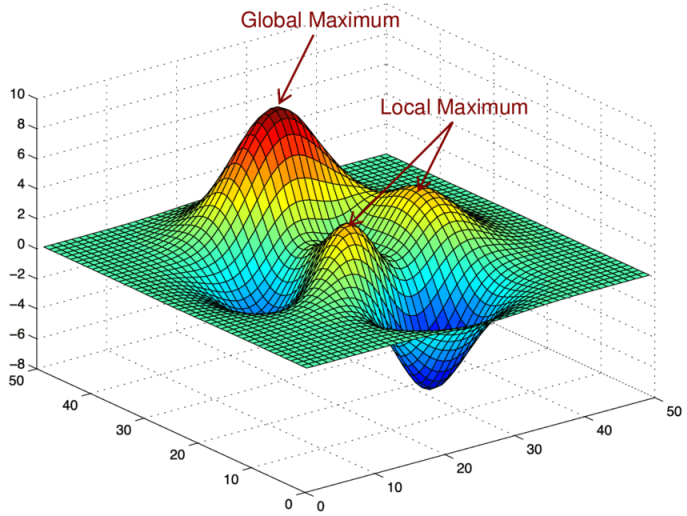
- Choosing the step size (also called the *learning rate*) will become one of the most important hyperparameter settings in training a neural network.

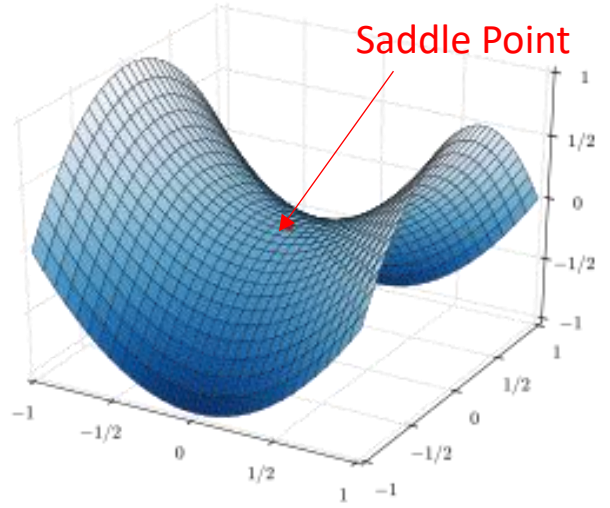
$$W_{t+1} = W_t - \eta \nabla L(W_t)$$

where $\nabla L(W) = \left(\frac{\partial L}{\partial w_1}, \cdots, \frac{\partial L}{\partial w_p} \right)$ and p is the total number of parameters.

Globe vs. local optimal

- Global Optimum: is the optimal solution among all possible solutions.
- Local Optimum: is a solution that is optimal (either maximal or minimal) within a neighboring set of candidate solutions.
- Saddle Point: a point on the surface of the graph of a function where the slopes (derivatives) in orthogonal directions are all zero but is not a local optimum.

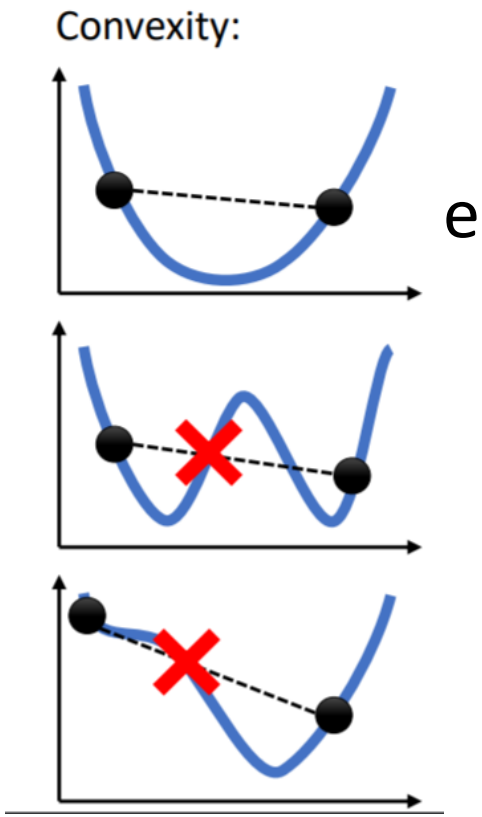




Convexity

- A function is convex if a line segment between any two points lies entirely “above” the graph.

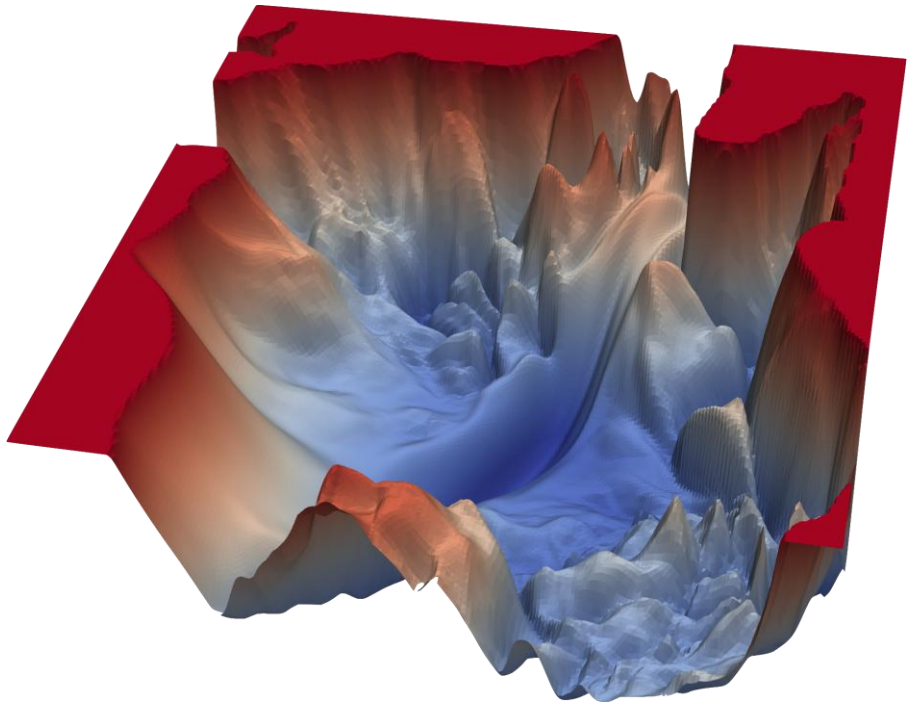
- If loss function is convex, simple algorithms like gradie e strong guarantee to converge to the globe optimum.



Visualizing loss surface of NNs

- The most obvious issue with non-convex loss landscapes.

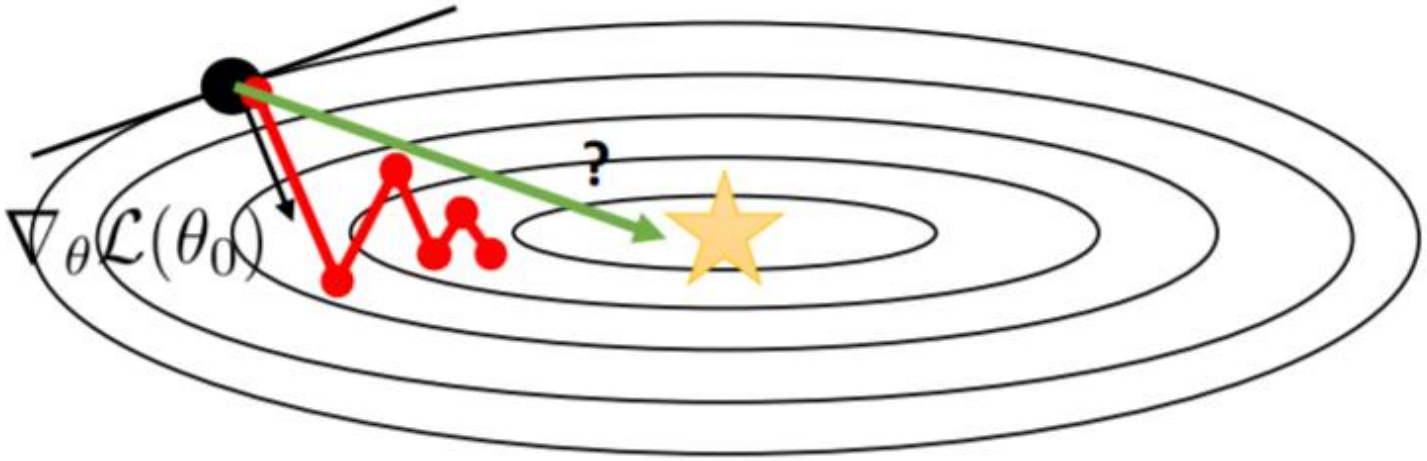
- This becomes less of an issue as the number of parameters increases. For big networks, local optimum exist, but tend to be not much worse than global optimum.



SGD

- Gradient estimates can be very noisy (Good and Bad)
- Use larger mini-batches to reduce stochastics
- Computation per update is efficient and does not depend on number of training samples.
  - Allows convergence on extremely large datasets

Convergence path



Gradient directions do not always move toward the optimum. The steepest direction is not always best.

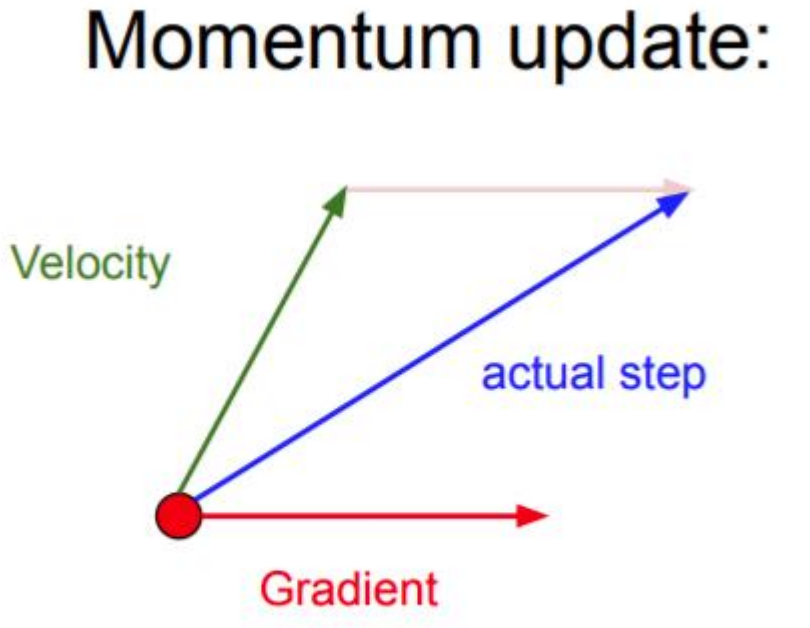
Optimal direction (can’t afford for neural networks)

SGD + Momentum

- Momentum is a method that helps accelerate SGD in the relevant direction and dampens oscillations

$$W_{t+1} = W_t - \eta\,(\nabla L(W) + \gamma \Delta W)$$

ΔW is the update for the previous step (known as velocity) The momentum term γ is usually set to 0.9 or a similar value.



Comparison

- In these scenarios, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum.

Image 2: SGD without momentum

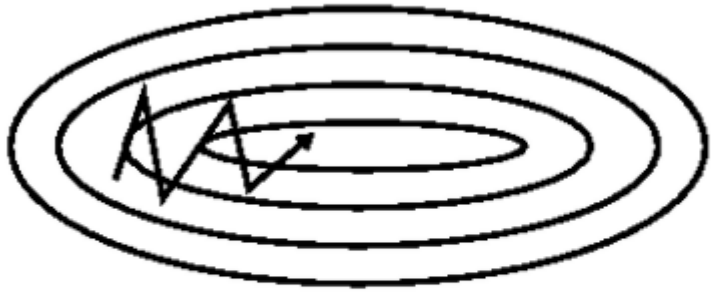


Image 3: SGD with momentum

The ball accumulates momentum as it rolls downhill, becoming faster and faster on the way (until it reaches its terminal velocity if there is air resistance, i.e. $\gamma<1$).

AdaGrad(*Adaptive Gradient Algorithm)*

Adapts the learning rate per parameter based on historical gradients. Frequent parameters get smaller updates; rare ones get larger.

1. ACCUMULATE SQUARED GRADIENTS

$$G(t) = G(t-1) + \nabla L(\theta)^2$$

2. SCALE LEARNING RATE PER PARAM

$$\eta_i = \eta \,/\, \sqrt{(G_i(t) + \varepsilon)}$$

3. UPDATE PARAMETERS

$$\theta \leftarrow \theta - \eta_i \cdot \nabla L(\theta)$$

WHAT IS G(t) ?

G(t)	Diagonal matrix (or per-param vector). Each element G_i accumulates ALL squared gradients for parameter i since training began.
G_i(t)	Scalar for param i: G_i(t) = Σ_{k=1}^{t} (∂L/∂θ_i)² — a running sum that only ever increases, never resets.
∇L(θ)²	Element-wise square of the gradient vector. Captures the magnitude of each partial derivative independently.

RMSprop

We now simply replace the diagonal matrix G_t with the decaying average over past squared gradients $\mathrm{E}\left(\left[\frac{\partial L}{\partial w_j}\right]^2\right)$.

$$w_{t+1,j} = w_{t,j} - \frac{\gamma}{\sqrt{\mathrm{E}\left(\left[\frac{\partial L}{\partial w_j}\right]^2\right) + \epsilon}} \frac{\partial L}{\partial w_j}$$

Adam (A METHOD FOR STOCHASTIC OPTIMIZATION)

- Adam can be looked at as a combination of RMSprop and Adaptive Gradient Descent with momentum. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum. Let $g_t = \frac{\partial L}{\partial W}$

$$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \text{ (Update biased first moment estimate)}$$
$$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \text{ (Update biased second raw moment estimate)}$$
$$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t) \text{ (Compute bias-corrected first moment estimate)}$$
$$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t) \text{ (Compute bias-corrected second raw moment estimate)}$$
$$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon) \text{ (Update parameters)}$$

ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION

Diederik P. Kingma*
University of Amsterdam, OpenAI
dpkingma@openai.com

Jimmy Lei Ba*
University of Toronto
jimmy@psi.utoronto.ca

Regularization

- **L1 regularization** we add a component that will penalize the sparsity of weights.

$$L_t(W) = \frac{1}{k_t - k_{t-1}} \sum_{i=k_{t-1}+1}^{k_t} L_i(x_i, y_i, W) + \frac{\lambda}{k_t - k_{t-1}} \left|\left|W\right|\right|^2_{L1}$$

where λ is the **regularization parameter** and L1 norm is denoted by the subscript *L1*.

Regularization

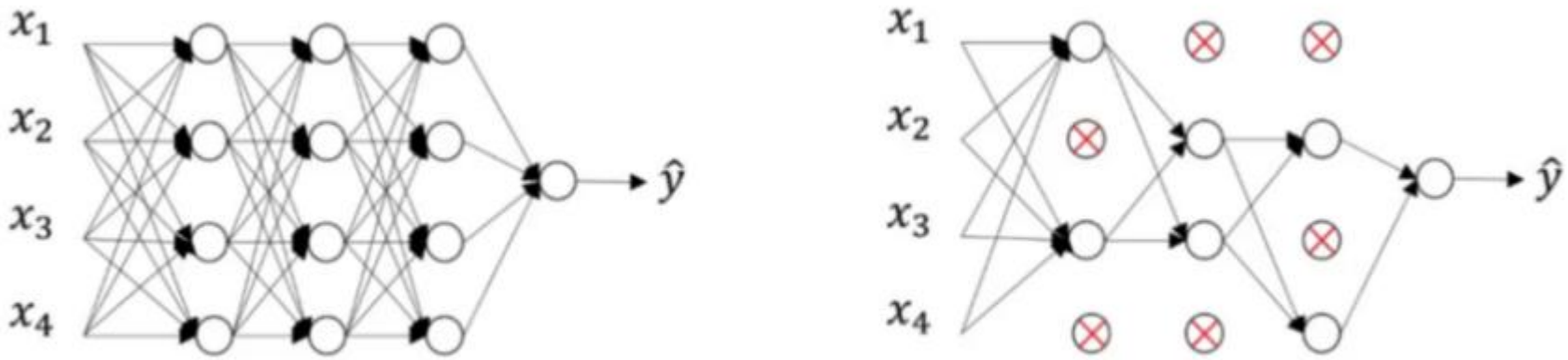
- **L2 regularization** we add a component that will penalize large weights.

$$L_t(W) = \frac{1}{k_t - k_{t-1}} \sum_{i=k_{t-1}+1}^{k_t} L_i(x_i, y_i, W) + \frac{\lambda}{k_t - k_{t-1}} \left|\left|W\right|\right|^2_F$$

where λ is the **regularization parameter** and Frobenius norm is denoted by the subscript *F*.

Dropout regularization

- The probability of keeping each node is set at **random**. You only decide of the **threshold**: a value that will determine if the node is kept or not.
- For example, if you set the threshold to 0.7, then there is a probability of 30% that a node will be removed from the network.



Left: neural network before dropout. Right: neural network after dropout.

Why Dropout works

- Dropout means that the neural network cannot rely on any input node, since each have a random probability of being removed. Therefore, the neural network will be reluctant to give high weights to certain features, because they might disappear.

- Consequently, the weights are spread across all features, making them smaller. This effectively shrinks the model and regularizes it.