Bitdefender

Training Neural Networks

Florin Gogianu,
Bitdefender ML Team

February 24, 2021

Slides adapted from the original course created by Ștefan Postăvaru.
Inspired by CS231n - Stanford, CS421 - University of Toronto

Outline

Bitdefender

Goals

Bitdefender

1. Optimize large, deep neural networks...

Goals

Bitdefender

1. Optimize large, deep neural networks...
2. ...for learning *useful* representations.

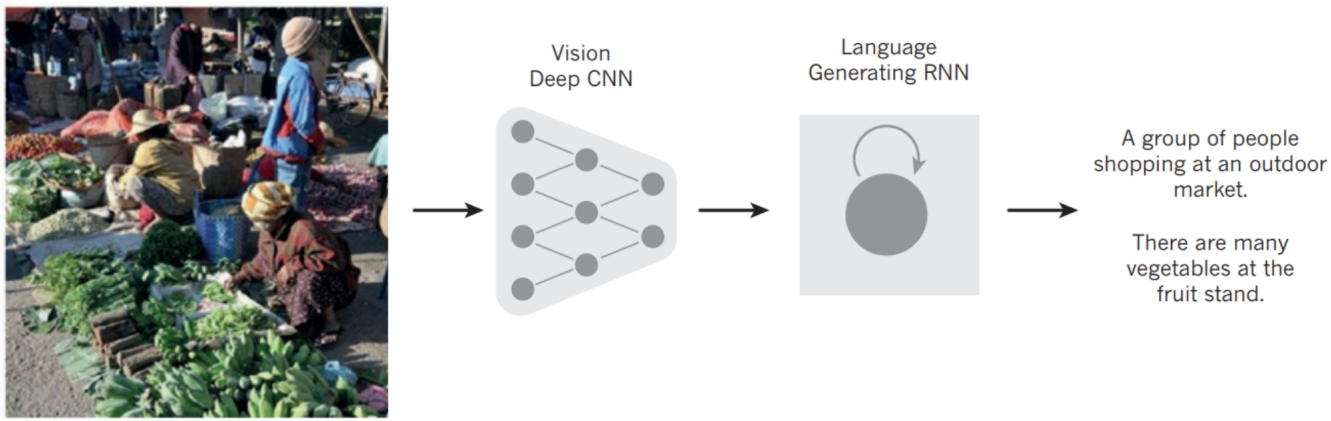
Terminology

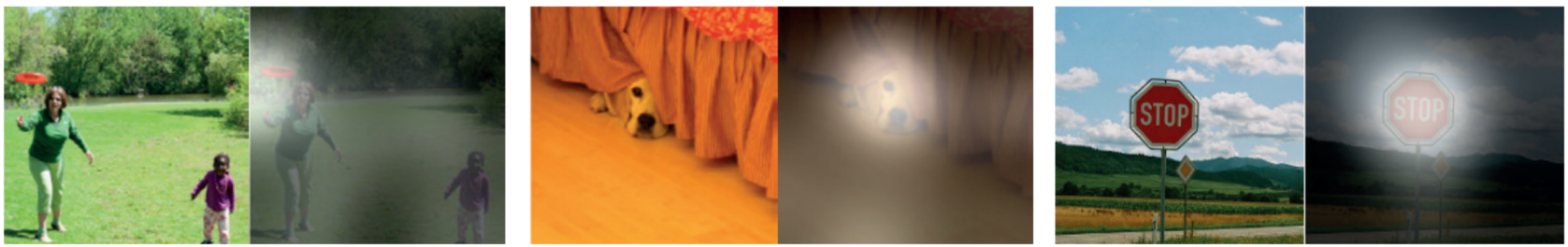
Bitdefender

- **Supervised learning:** labeled data points of the correct behaviour.
- **Reinforcement learning:** receive some reward signal and try to maximize it by improving the model's behaviour
- **Unsupervised learning:** no labels - the aim is discovering interesing patterns in the data and usefull representations

Supervised Learning[1]

Caption generation[2]





A woman is throwing a **frisbee** in a park.

A **dog** is standing on a hardwood floor.

A **stop** sign is on a road with a mountain in the background

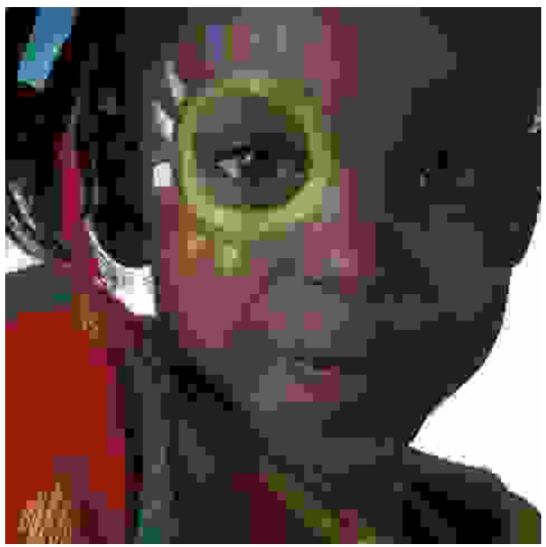
[1] from Grosse, Ba, 2019

[2] Xu et al., 2015

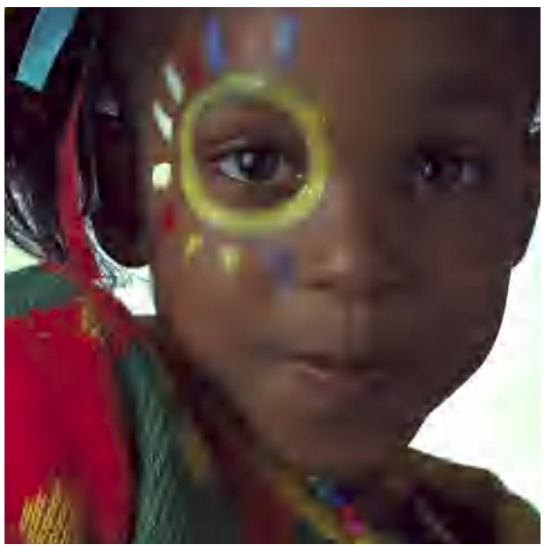
Unsupervised Learning[1]

Bitdefender

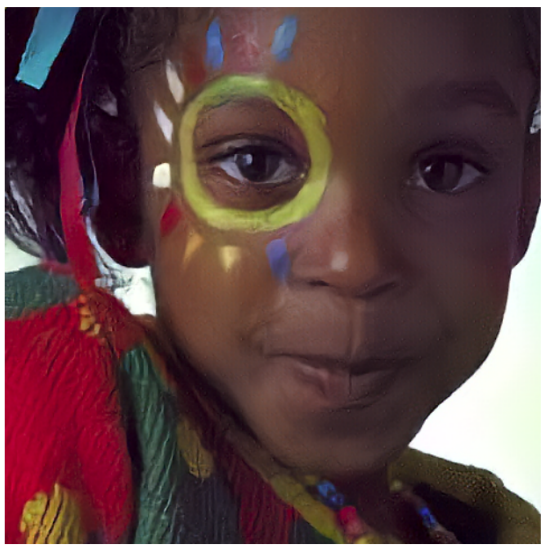
Image compression[2]



JPEG



JPEG 2000



WaveOne

[1] from CS294-158, 2019

[2] Rippel, Bourdev, 2017

Unsupervised Learning

Bitdefender

Images from text[1]

TEXT PROMPT

an armchair in the shape of an avocado [...]

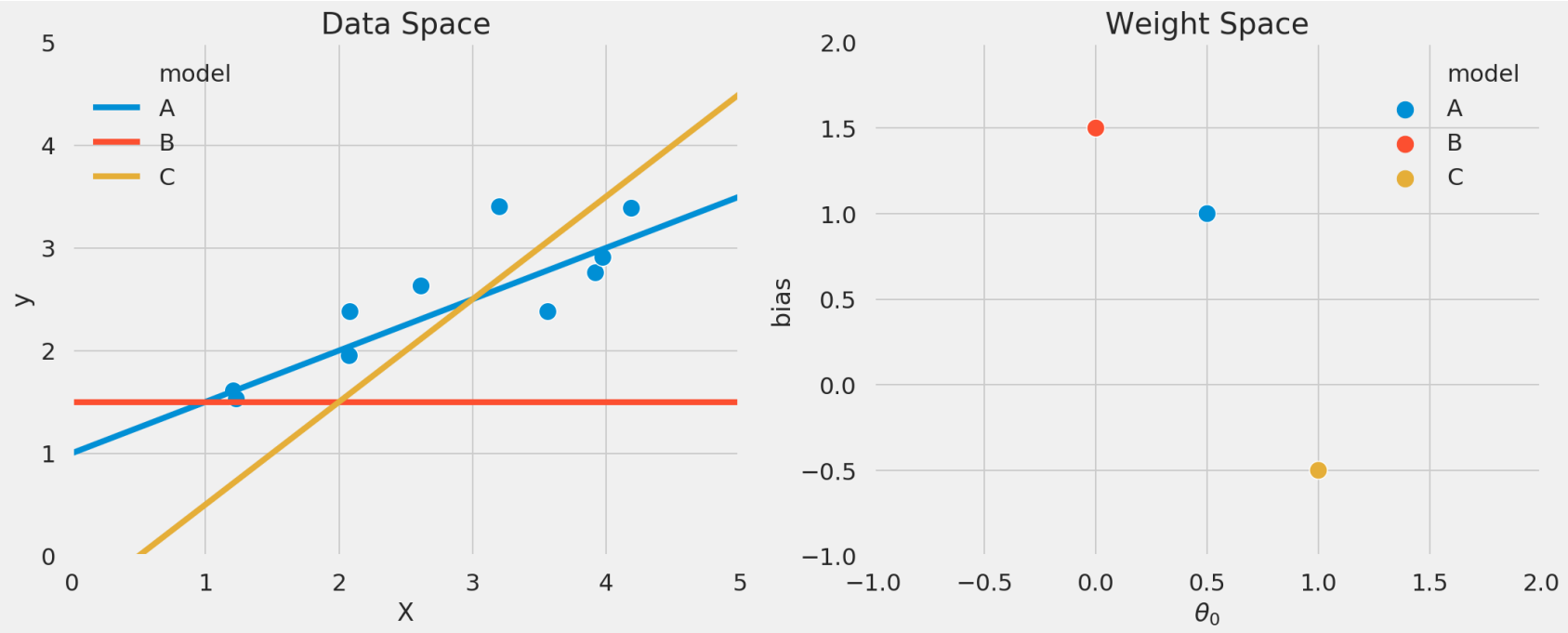
AI-GENERATED IMAGES

an illustration of a baby daikon radish in a tutu walking a dog

AI-GENERATED IMAGES

[1] OpenAI DALL-E, 2021

Linear regression



Three different linear models in the data and weight space [1].

$$y = \boldsymbol{\theta}^{\mathsf{T}} \boldsymbol{x} + b$$

[1] following Grosse, Ba - CS421, 2019

Linear models

Bitdefender

- Linear model $y = \boldsymbol{\theta}^\intercal \, \boldsymbol{\phi}(\boldsymbol{x})$, with `Mean Squared Error` objective function $\mathcal{L}(\boldsymbol{\theta}) = (t - y)^2$

Linear models

Bitdefender

- Linear model $y = \boldsymbol{\theta}^\intercal \, \boldsymbol{\phi}(\boldsymbol{x})$, with `Mean Squared Error` objective function $\mathcal{L}(\boldsymbol{\theta}) = (t - y)^2$
- Has a nice closed form solution: $(\boldsymbol{\Phi}^\intercal \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\intercal \boldsymbol{t}$

Linear models

Bitdefender

- Linear model $y = \boldsymbol{\theta}^\intercal \, \boldsymbol{\phi}(\boldsymbol{x})$, with `Mean Squared Error` objective function $\mathcal{L}(\boldsymbol{\theta}) = (t - y)^2$
- Has a nice closed form solution: $(\boldsymbol{\Phi}^\intercal \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\intercal \boldsymbol{t}$
- But the solution can also be found iteratively:
  - Compute the gradient of $\mathcal{L}(\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = (t - y)\boldsymbol{\phi}(\boldsymbol{x})$$

Linear models

Bitdefender

- Linear model $y = \boldsymbol{\theta}^\intercal \, \phi(\boldsymbol{x})$, with `Mean Squared Error` objective function $\mathcal{L}(\boldsymbol{\theta}) = (t - y)^2$
- Has a nice closed form solution: $(\boldsymbol{\Phi}^\intercal \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\intercal \boldsymbol{t}$
- But the solution can also be found iteratively:
  - Compute the gradient of $\mathcal{L}(\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$:

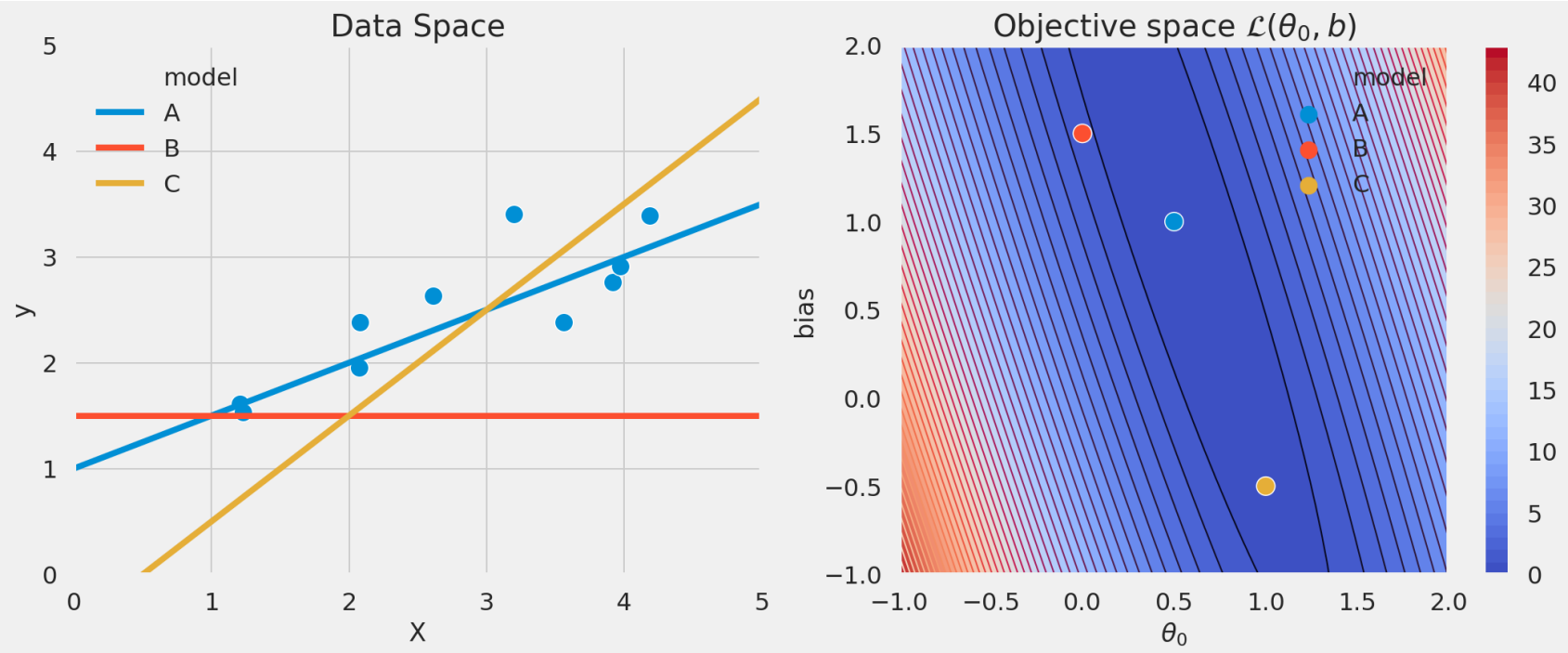
$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = (t - y)\phi(\boldsymbol{x})$$

  - Perform gradient descent:

$$\boldsymbol{\theta}_{j+1} \leftarrow \boldsymbol{\theta}_j - \alpha \nabla_{\boldsymbol{\theta}_j} \mathcal{L}$$

- Why do gradient descent if we can find the minimum analytically?

Linear models



Three different linear models in the data and objective $\mathcal{L}(\theta_0, b) = [t - f(x; \theta_0, b)]^2$ space.

following Grosse, Ba - CS421, 2019

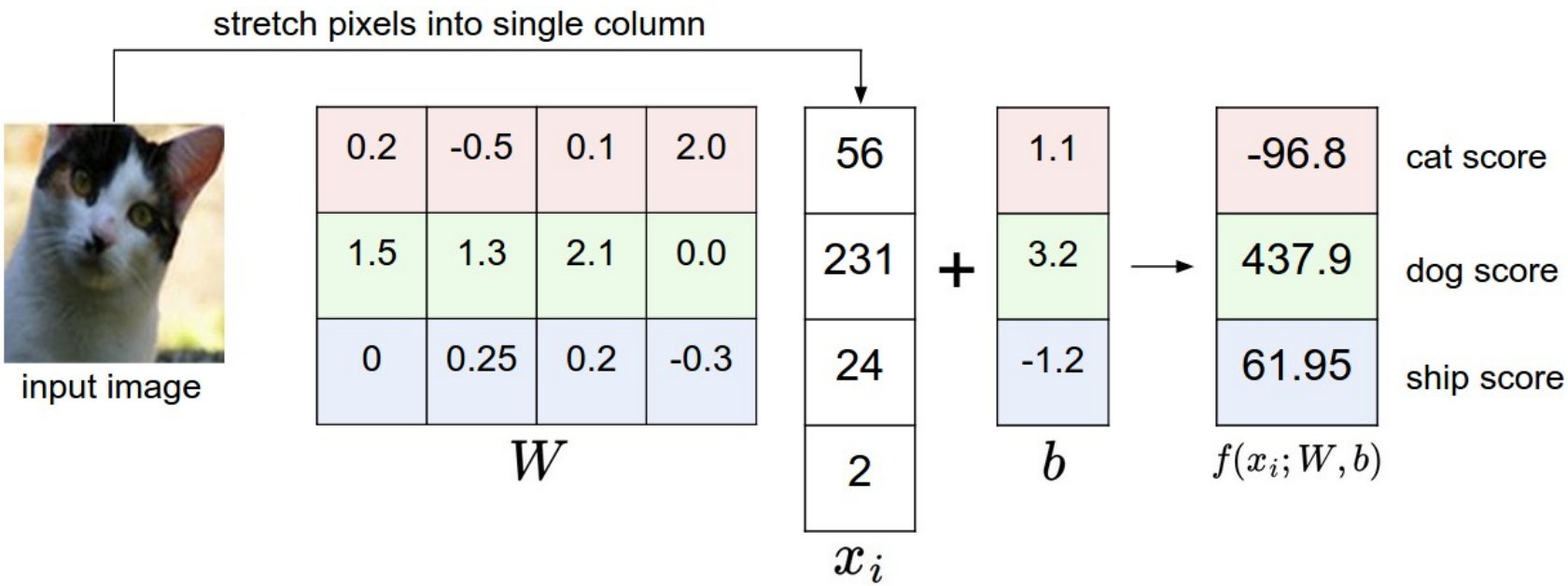
Also discussed

Bitdefender

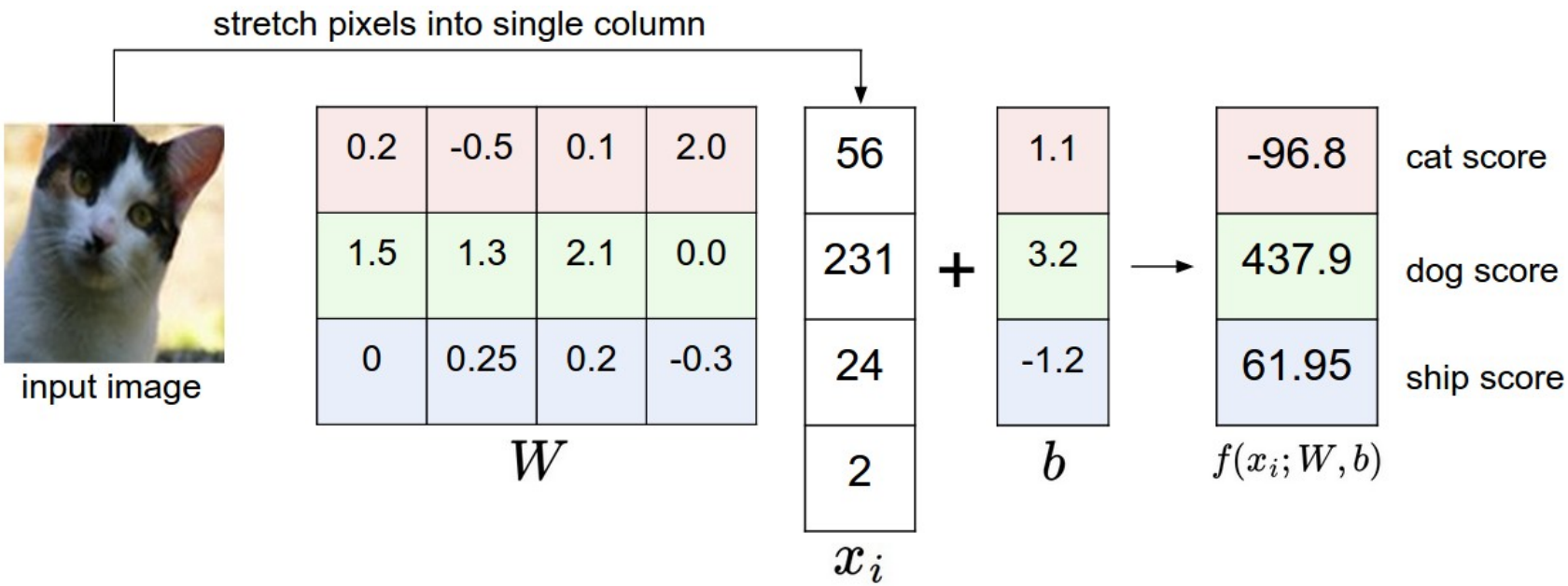
- Model complexity
- Regularization
- Bias / Variance trade-off
- Over-fitting / Under-fitting

Outline

Linear classification

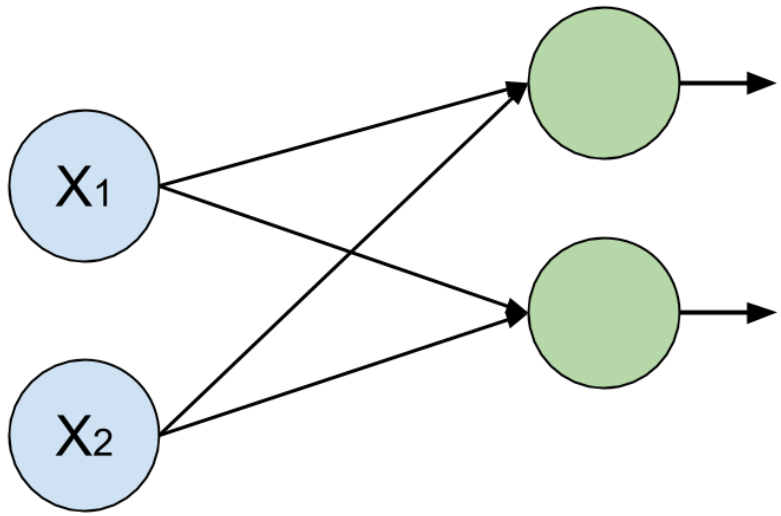


Linear classification



Learned weights:

A simple transformation

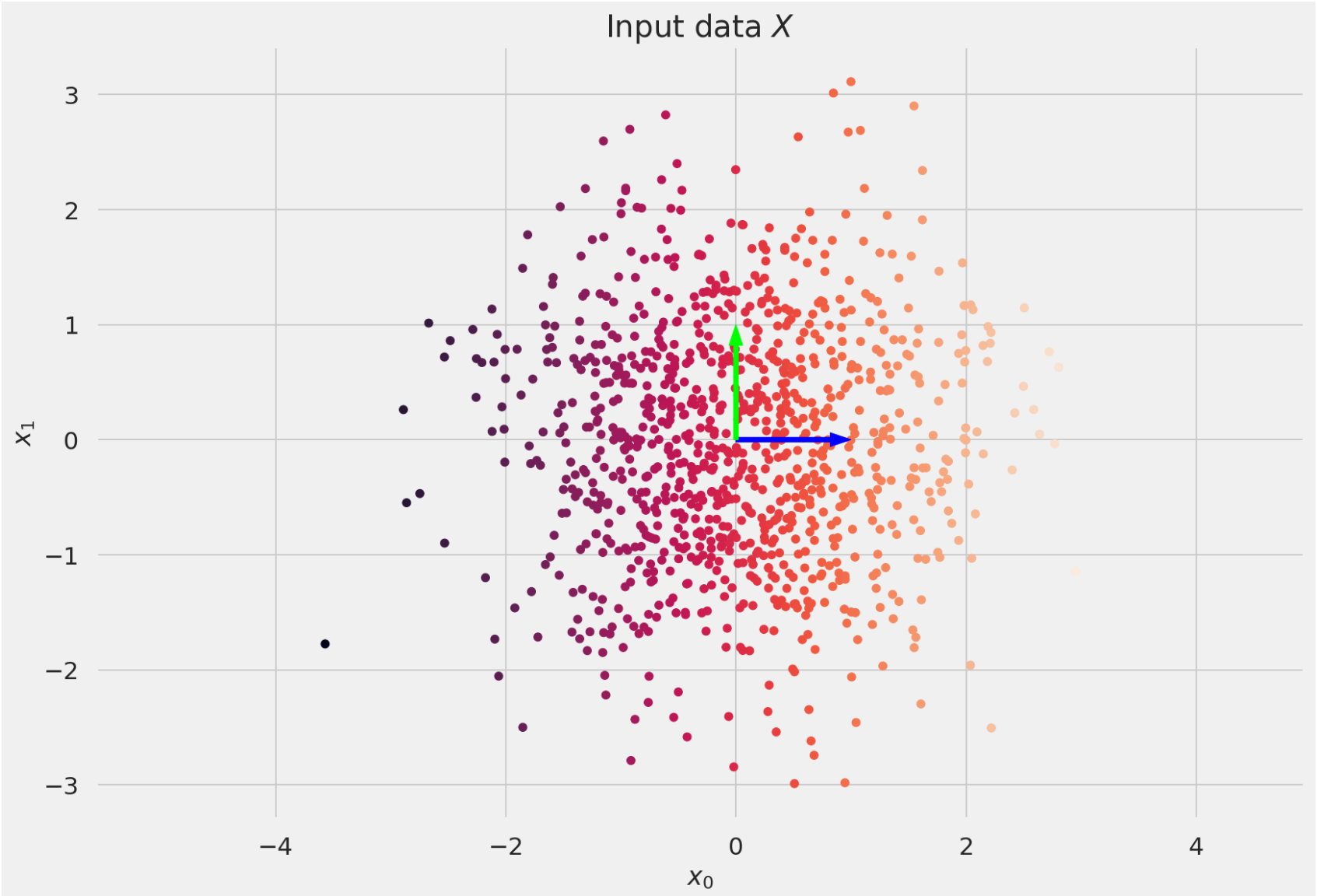


A linear model with four weights. Also can be seen as two stacked "neurons" without activation functions.

Linear case: $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$

A simple transformation

Bitdefender



following LeCun, Canziani - DS-GA, 2000

A simple transformation

Bitdefender

- $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$
- What is $\boldsymbol{W}$ actually doing to $\boldsymbol{x}$?

A simple transformation

Bitdefender

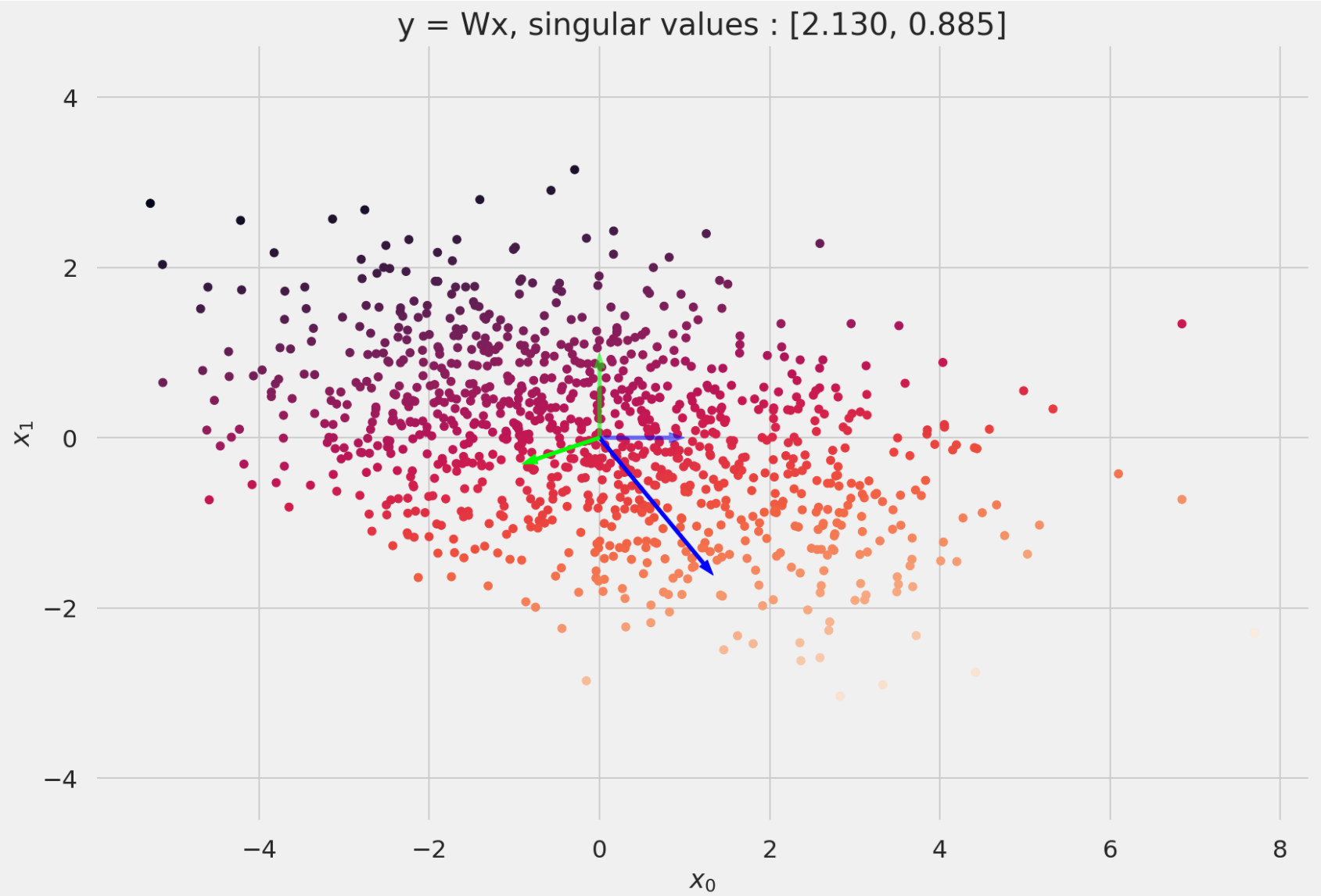
- $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$
- What is $\boldsymbol{W}$ actually doing to $\boldsymbol{x}$?
- Let's perform singular value decomposition for some intuition:

$$\boldsymbol{W} = \boldsymbol{U} \times \boldsymbol{S} \times \boldsymbol{V}^{\mathsf{T}}$$

$$\begin{bmatrix} 0.19 & 1.84 \\ -0.97 & -0.93 \end{bmatrix} = \underbrace{\begin{bmatrix} 0.83 & -0.55 \\ -0.55 & -0.83 \end{bmatrix}}_{\text{rotation}} \times \underbrace{\begin{bmatrix} 2.17 & 0.00 \\ 0.00 & 0.74 \end{bmatrix}}_{\text{scale}} \times \underbrace{\begin{bmatrix} 0.32 & 0.94 \\ 0.94 & -0.32 \end{bmatrix}}_{\text{reflection}}$$

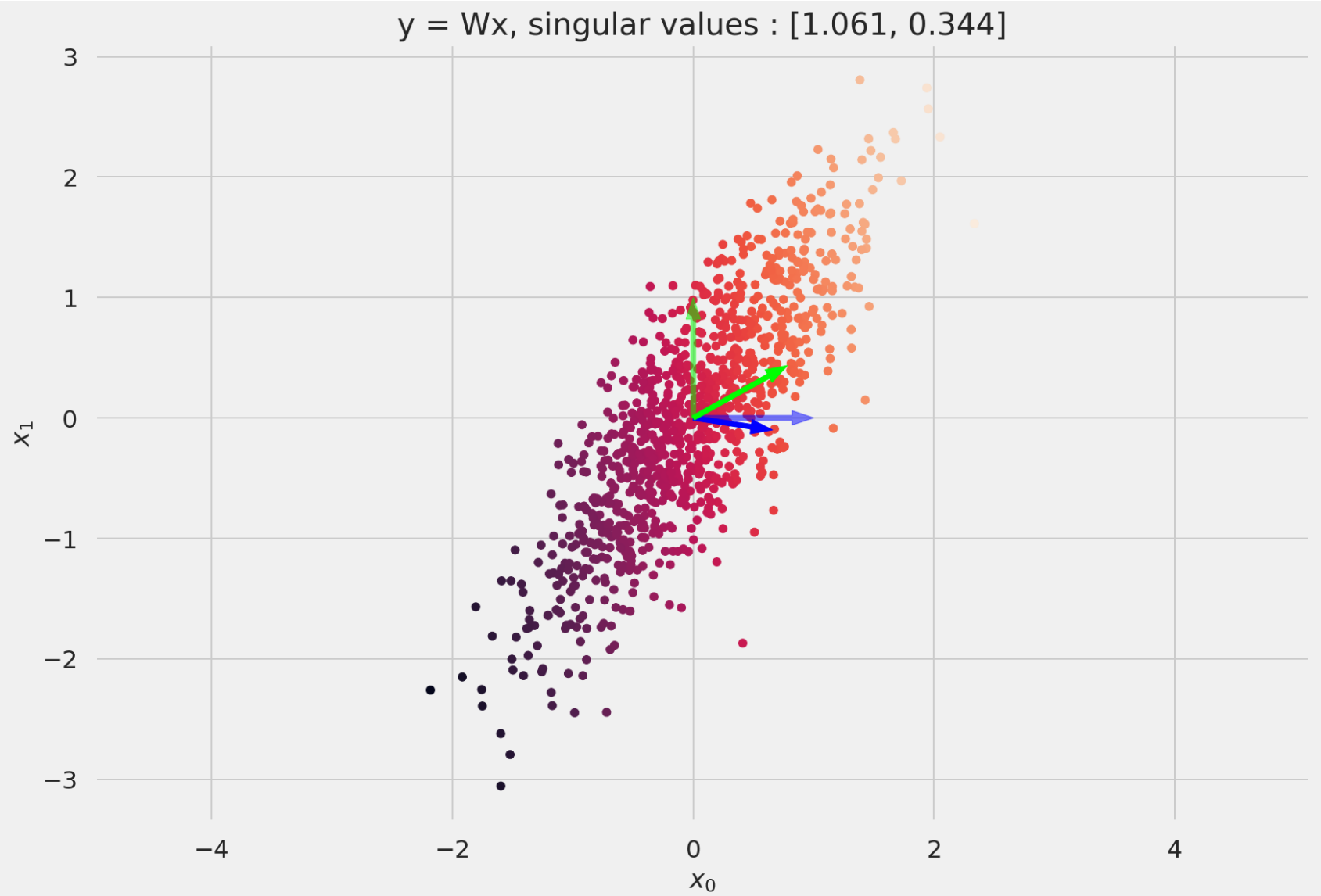
A simple transformation

Bitdefender



A simple transformation

Bitdefender



A simple transformation

Bitdefender

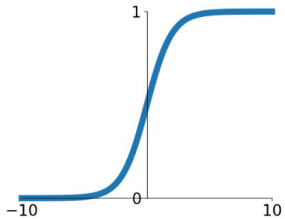
We can **scale**, **rotate** and **reflect** data. Can we do more?

Non-linear functions[1]

Bitdefender

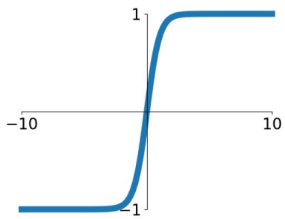
Sigmoid

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



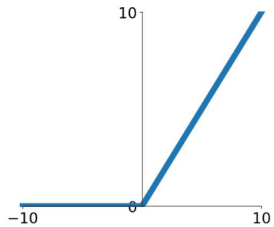
tanh

$$\tanh(x)$$



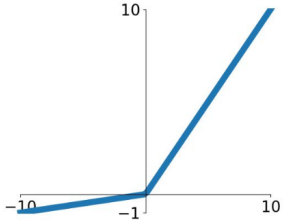
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

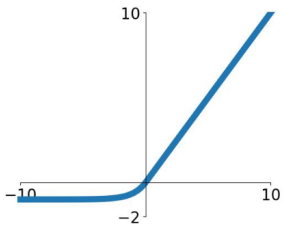


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

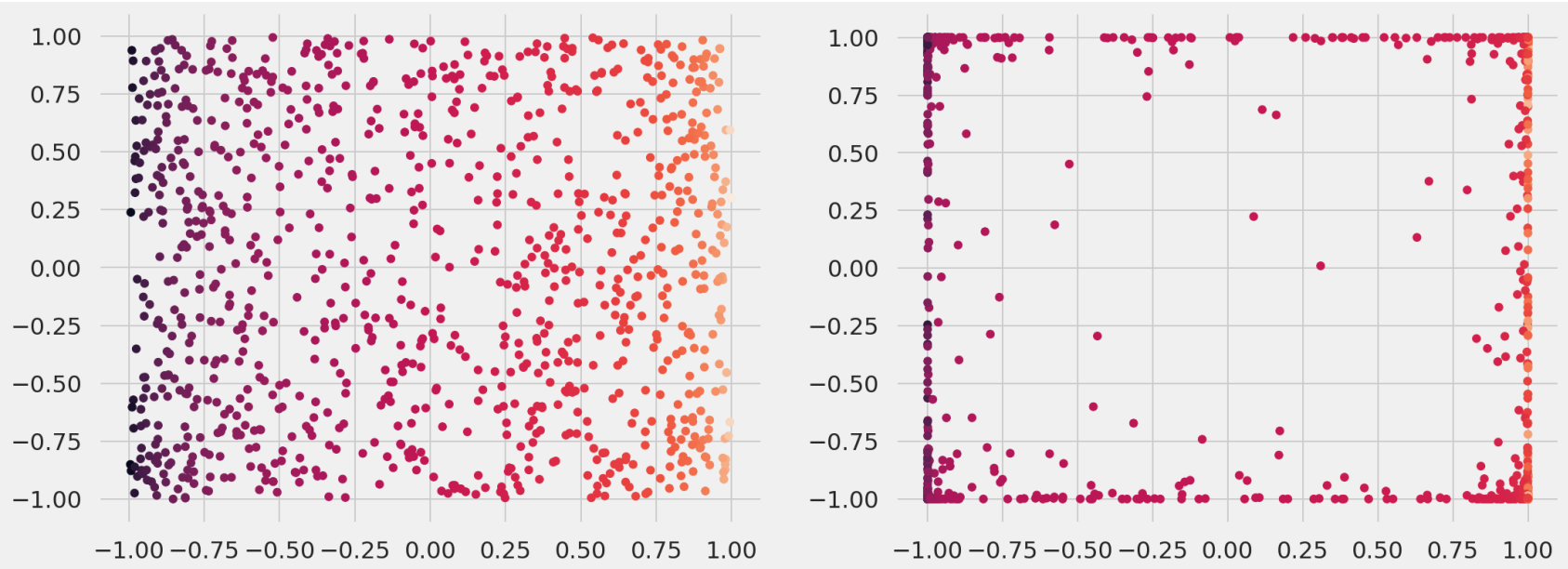
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



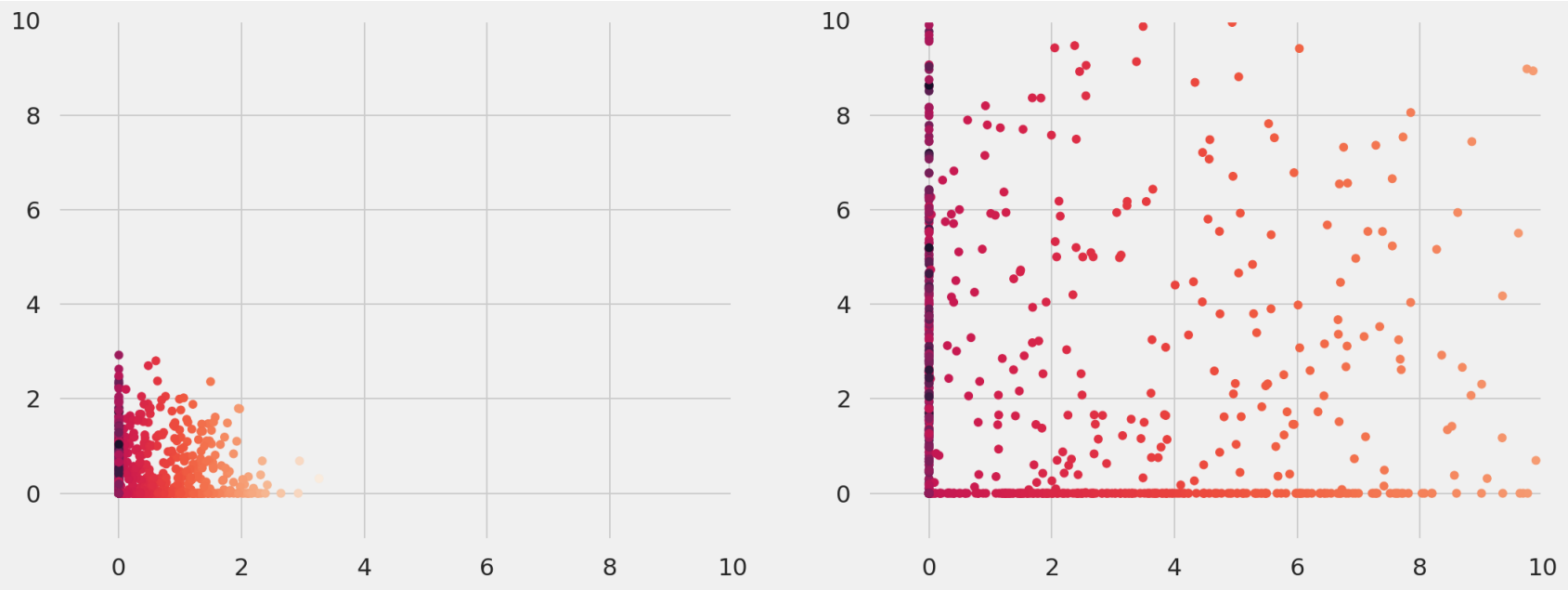
[1] from CS231n, lecture 4, 2019

Non-linear transformation



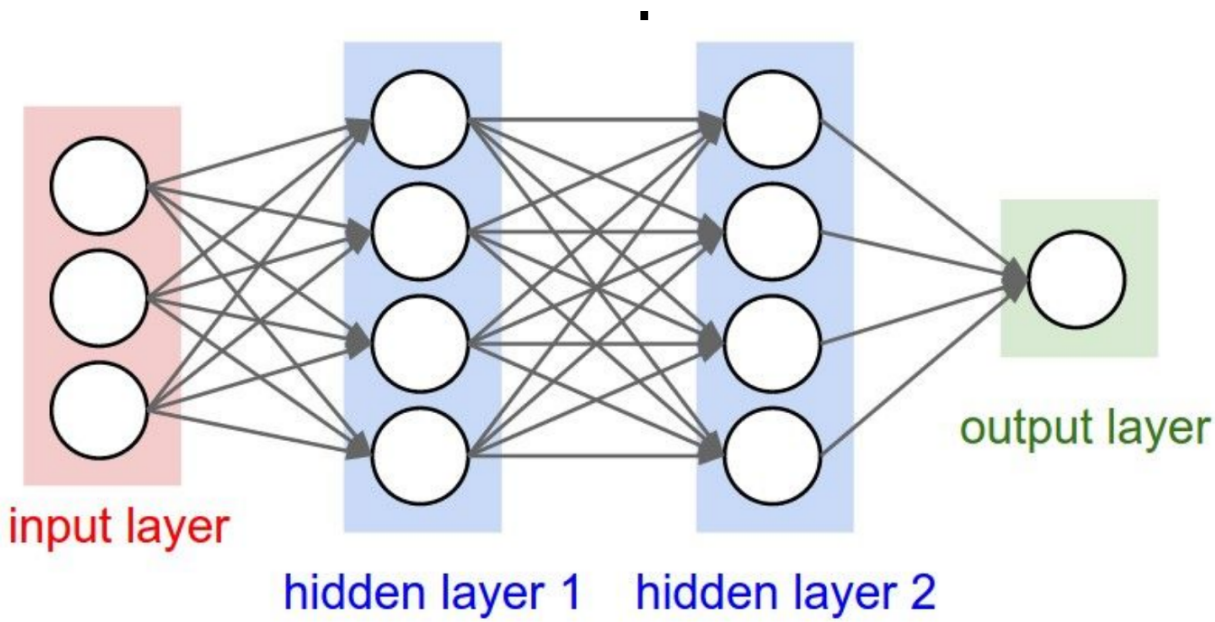
$\tanh(\boldsymbol{S}\boldsymbol{x})$ activation function with two different scale factors (left=1.0, right=5.0).

Non-linear transformation



ReLU($\boldsymbol{Sx}$) activation function with two different scale factors (left=1.0, right=5.0).

Neural Networks, finally

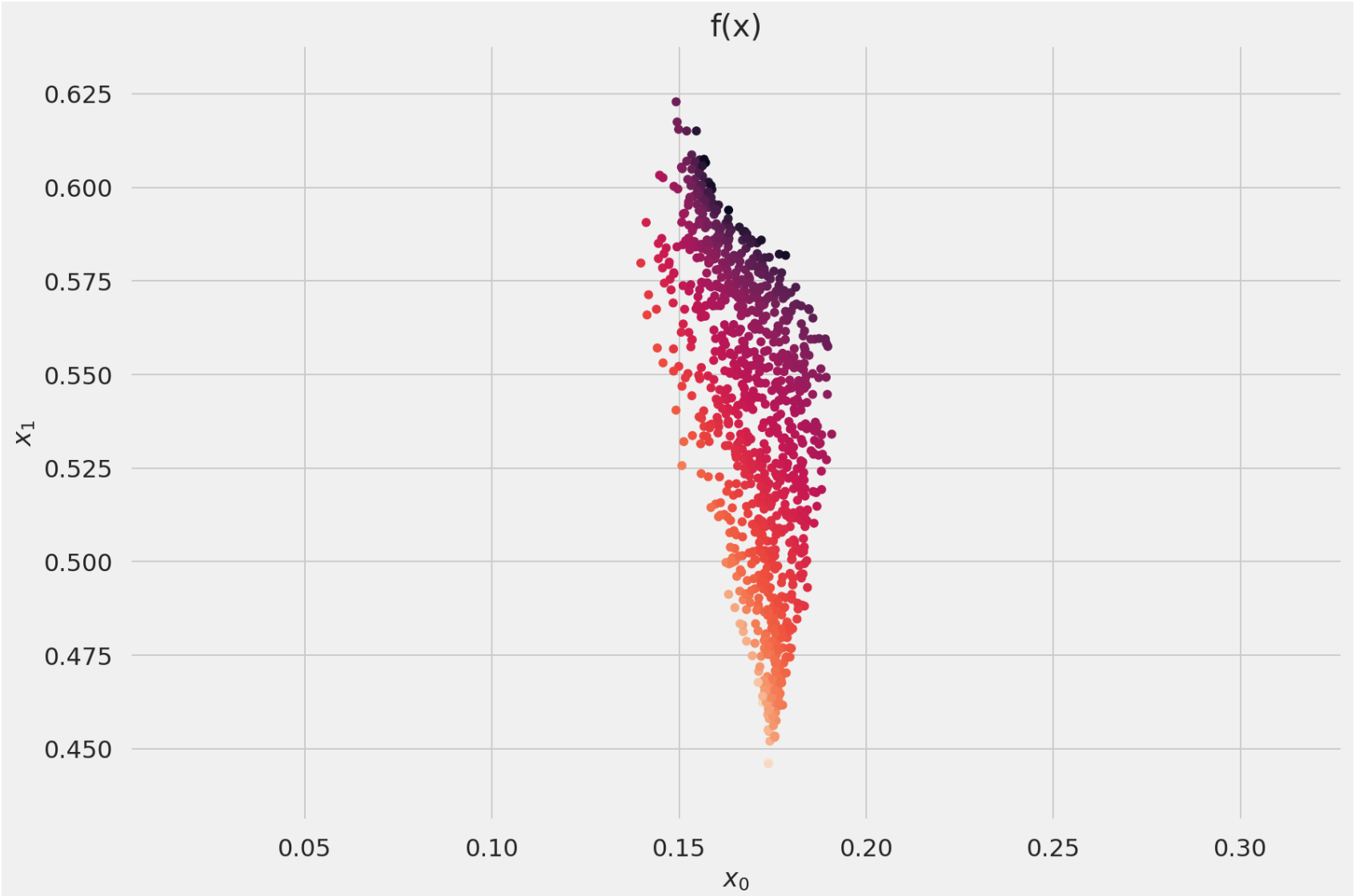


Typical "three-layer" or "two-hidden-layer" neural network

- Linear case: $f(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x}$
- Neural Net: $f(\boldsymbol{x}) = \boldsymbol{W}_3\,\sigma(\boldsymbol{W}_2\,\sigma(\boldsymbol{W}_1\boldsymbol{x}))$

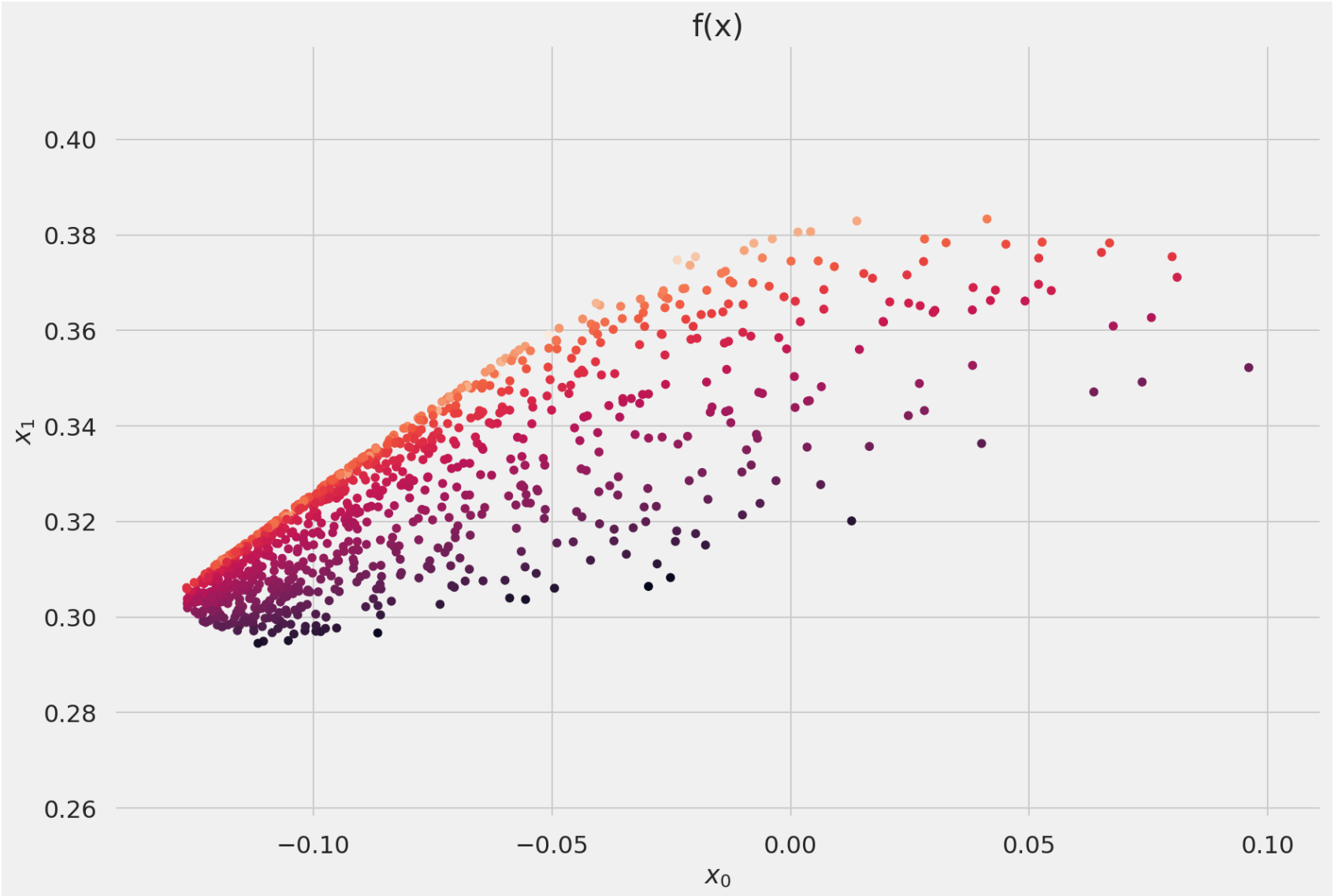
NN transformation

Bitdefender



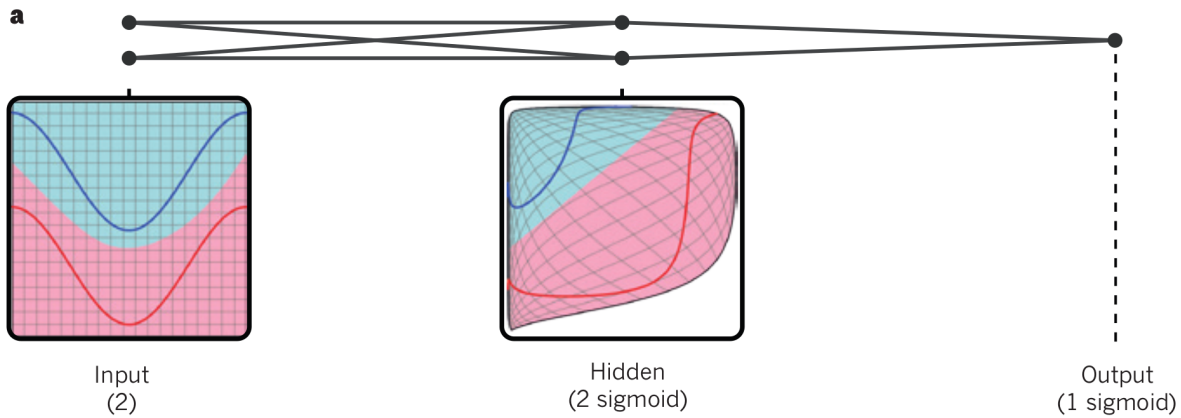
Transformation performed by a 5-layer **random** neural network

NN transformation



Transformation performed by a 5-layer **random** neural network

Warping space. Why is it usefull.

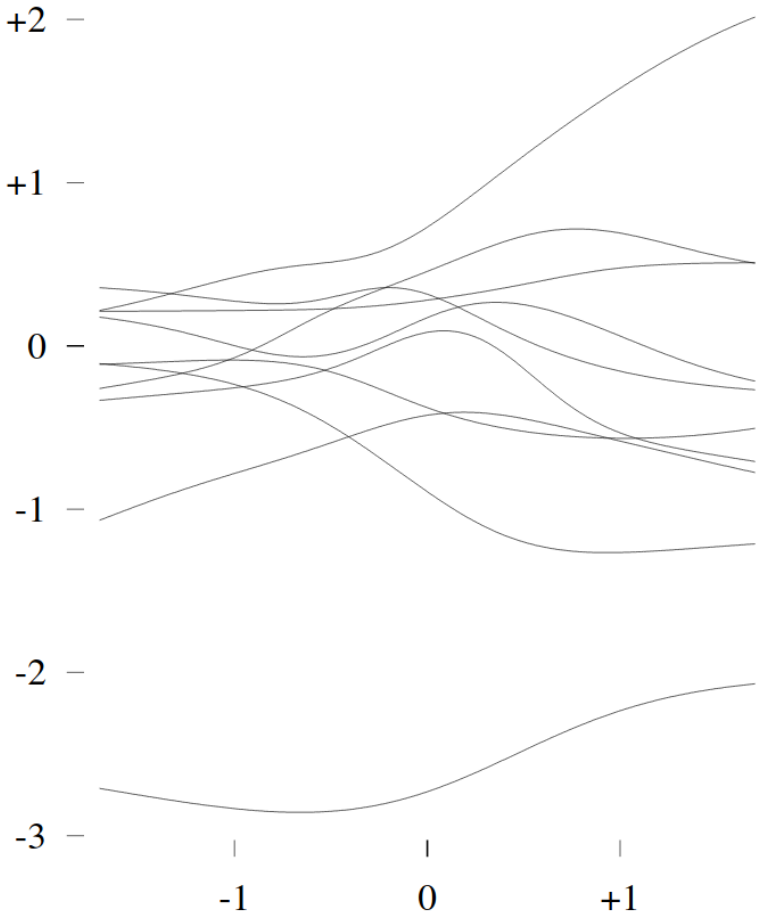


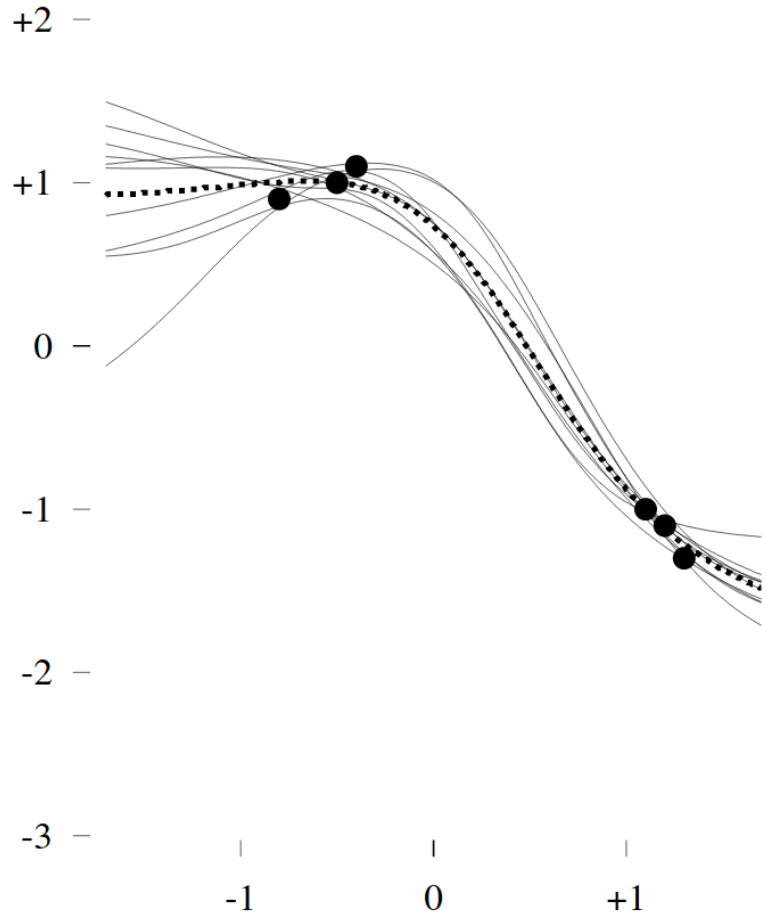
Space warping by a one hidden layer neural network for learning a linearized representation of the decision boundary[1].

[1] from LeCun 2015, Nature.

Outline

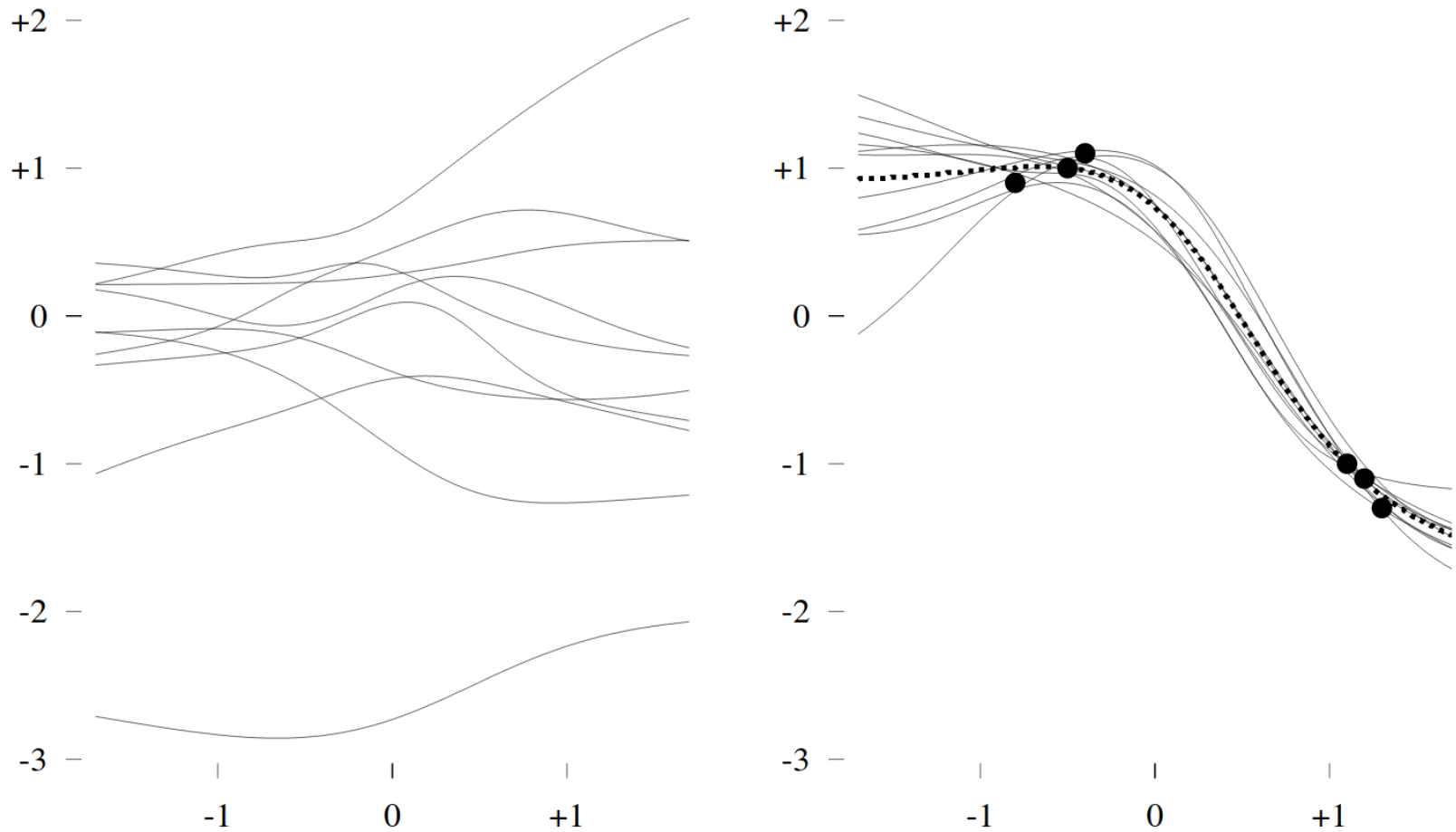
How to learn θ?





[1] Radford M. Neal, Bayesian Learning for Neural Networks

How to learn θ?



Sample 2.6 million networks and you're done!

[1] Radford M. Neal, Bayesian Learning for Neural Networks

How to learn θ?

Bitdefender

- Can we do better?

How to learn θ?

Bitdefender

- Can we do better?
- Use the lesson from the iterative linear regression

How to learn θ?

Bitdefender

- Can we do better?
- Use the lesson from the iterative linear regression
- Compute the gradient of $\mathcal{L}(\mathbf{\Theta})$ w.r.t. $\mathbf{\Theta}$,

How to learn θ?

Bitdefender

- Can we do better?
- Use the lesson from the iterative linear regression
- Compute the gradient of $\mathcal{L}(\mathbf{\Theta})$ w.r.t. $\mathbf{\Theta}$,
- Find the weights using gradient descent.

How to learn θ?

Bitdefender

- Can we do better?
- Use the lesson from the iterative linear regression
- Compute the gradient of $\mathcal{L}(\mathbf{\Theta})$ w.r.t. $\mathbf{\Theta}$,
- Find the weights using gradient descent.
- It can get tedious and inflexible:
  - Additional cost functions, regularization loss, etc...
  - We would want to quickly experiment with different layers and activations.

But it's not that complicated[1]

Bitdefender

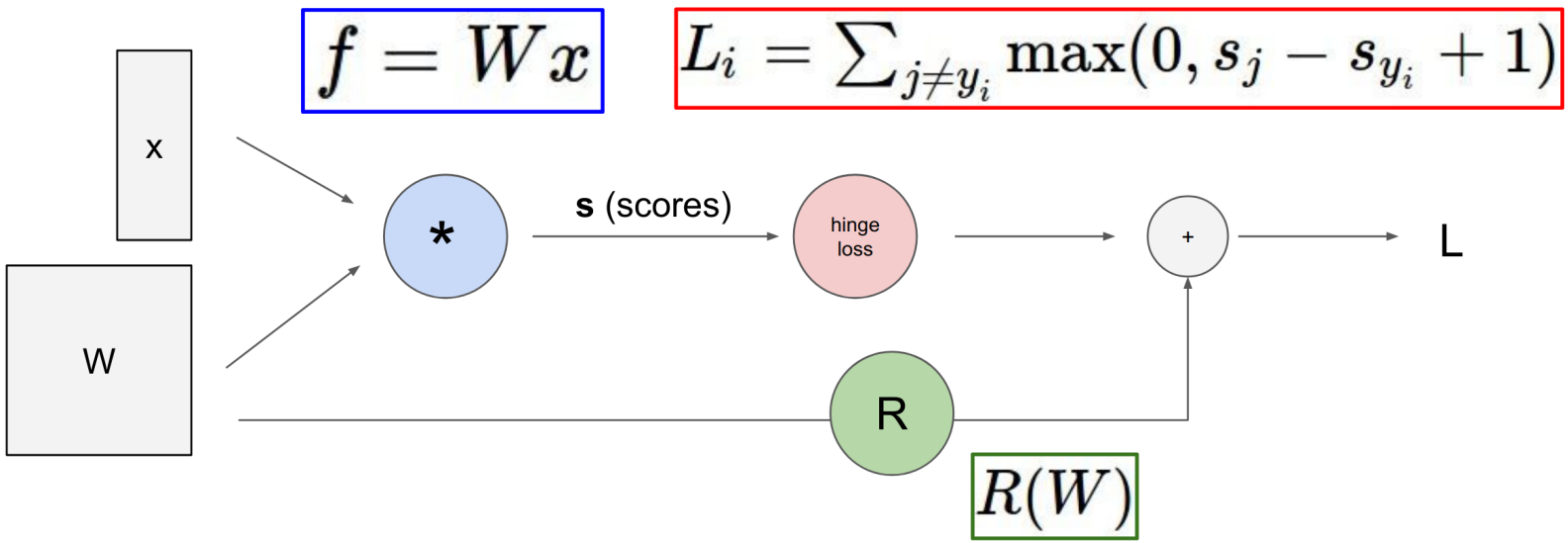
```python
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

[1] from CS231n, lecture 4, 2019

Outline

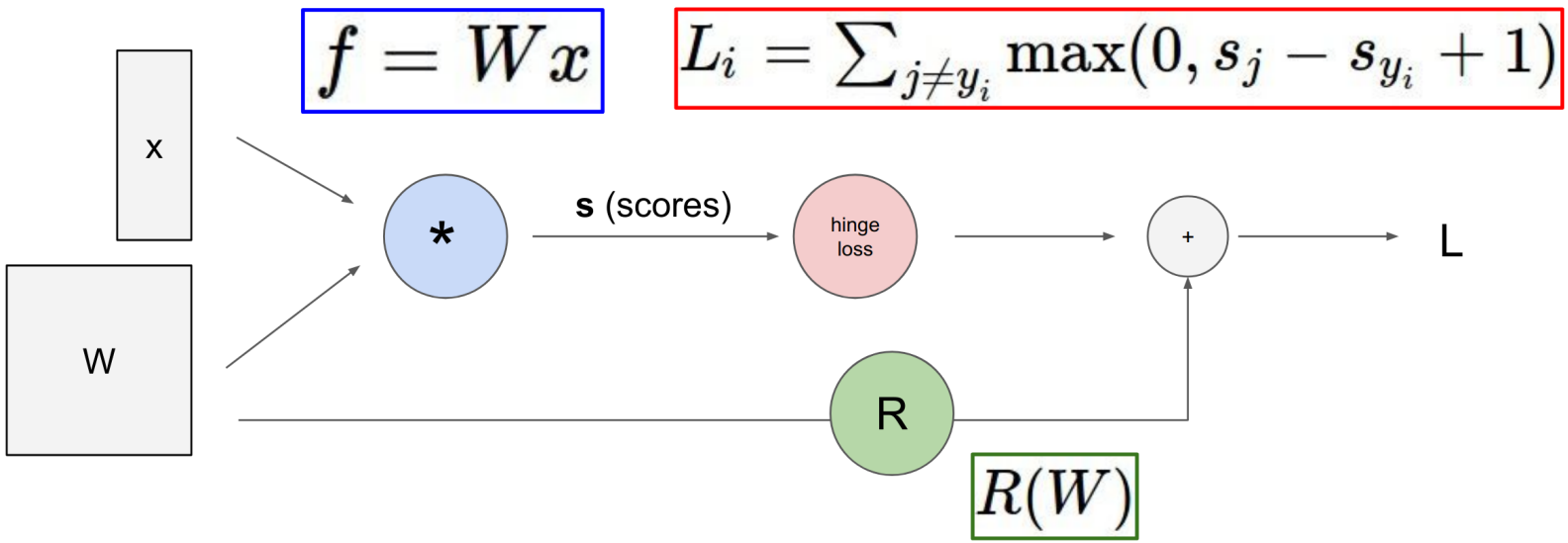
Bitdefender

Computation graphs + automatic differentation



[1] Grosse, Ba, CS421

Computation graphs + automatic differentation



Automatic differentiation: takes a program which computes a value, and automatically constructs a procedure for computing derivatives of that value[1].

[1] Grosse, Ba, CS421

Simple example[1]

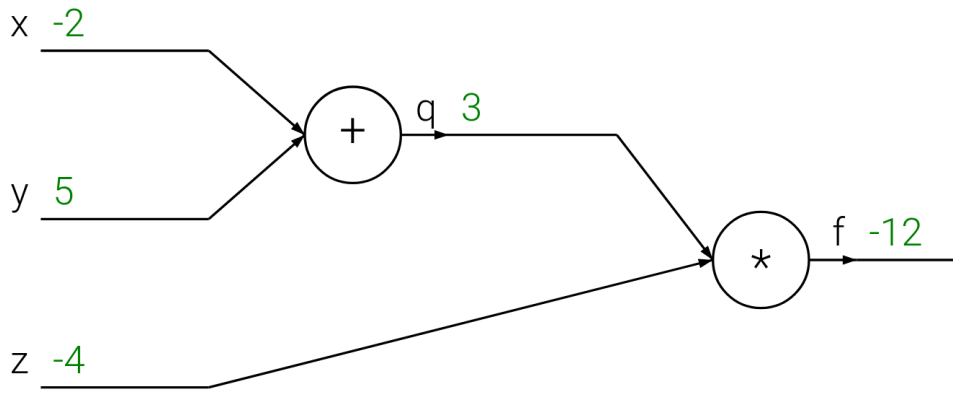
Bitdefender

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

[1] from CS231n, lecture 4, 2019

Simple example[1]

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



[1] from CS231n, lecture 4, 2019

Simple example[1]

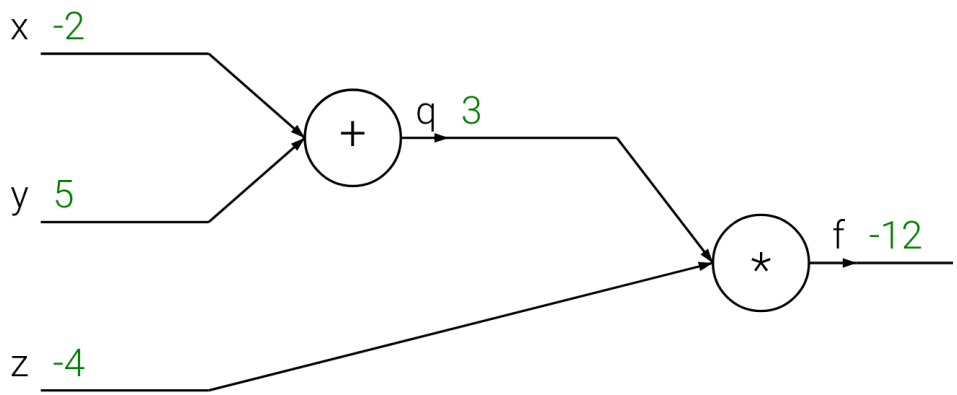
$$f(x, y, z) = (x + y)z$$

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

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

We want:

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



[1] from CS231n, lecture 4, 2019

Simple example[1]

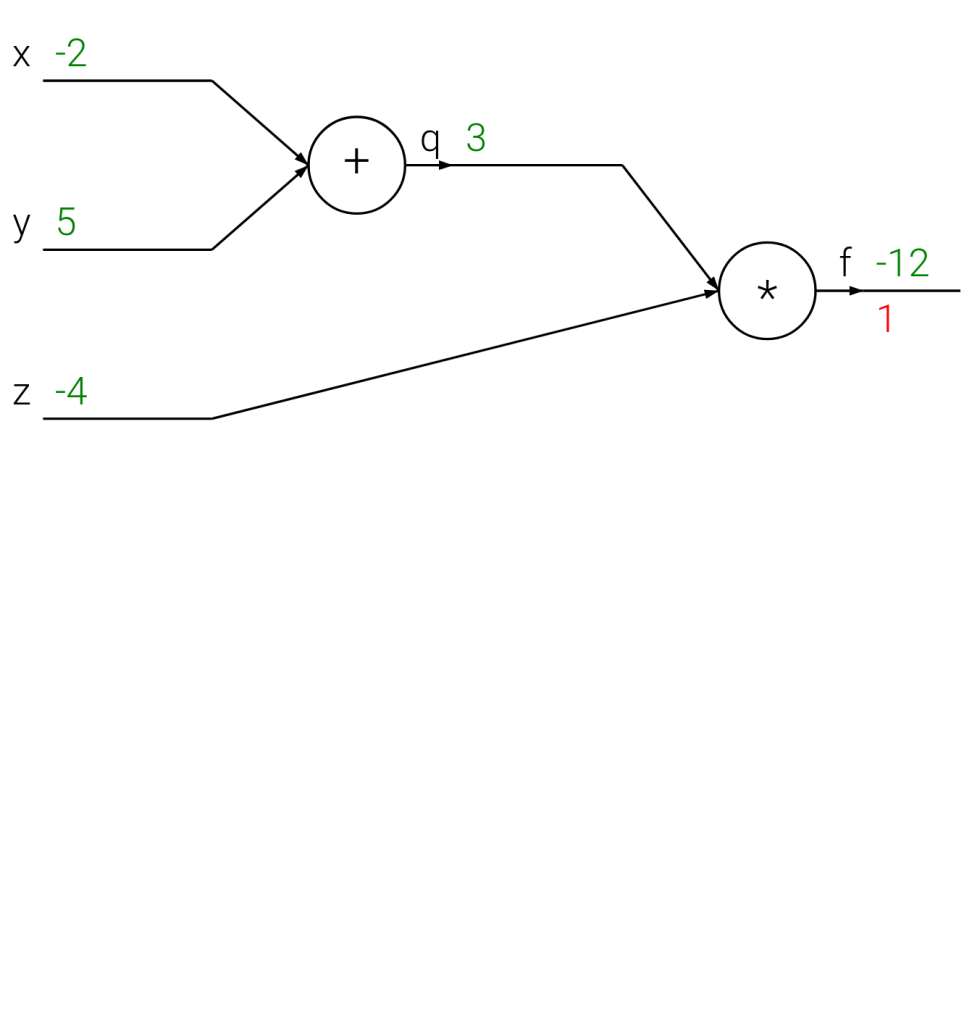
$$f(x, y, z) = (x + y)z$$

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

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

We want:

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



[1] from CS231n, lecture 4, 2019

Simple example[1]

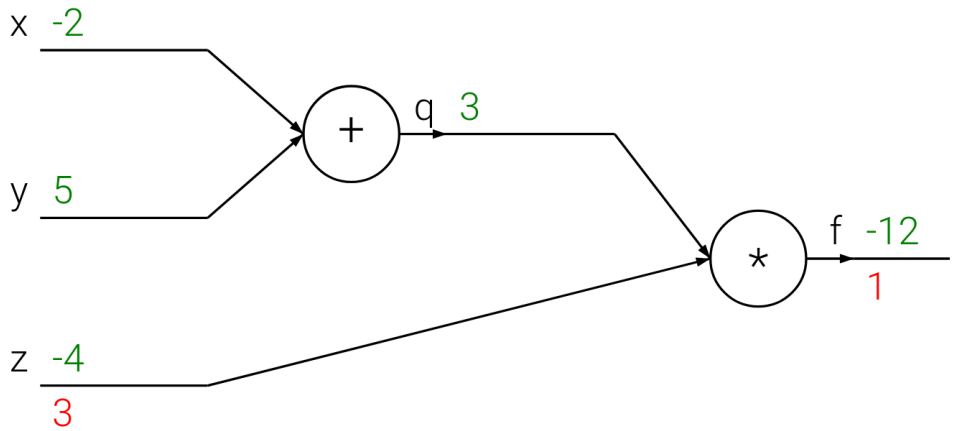
$$f(x, y, z) = (x + y)z$$

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

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

We want:

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



[1] from CS231n, lecture 4, 2019

Simple example[1]

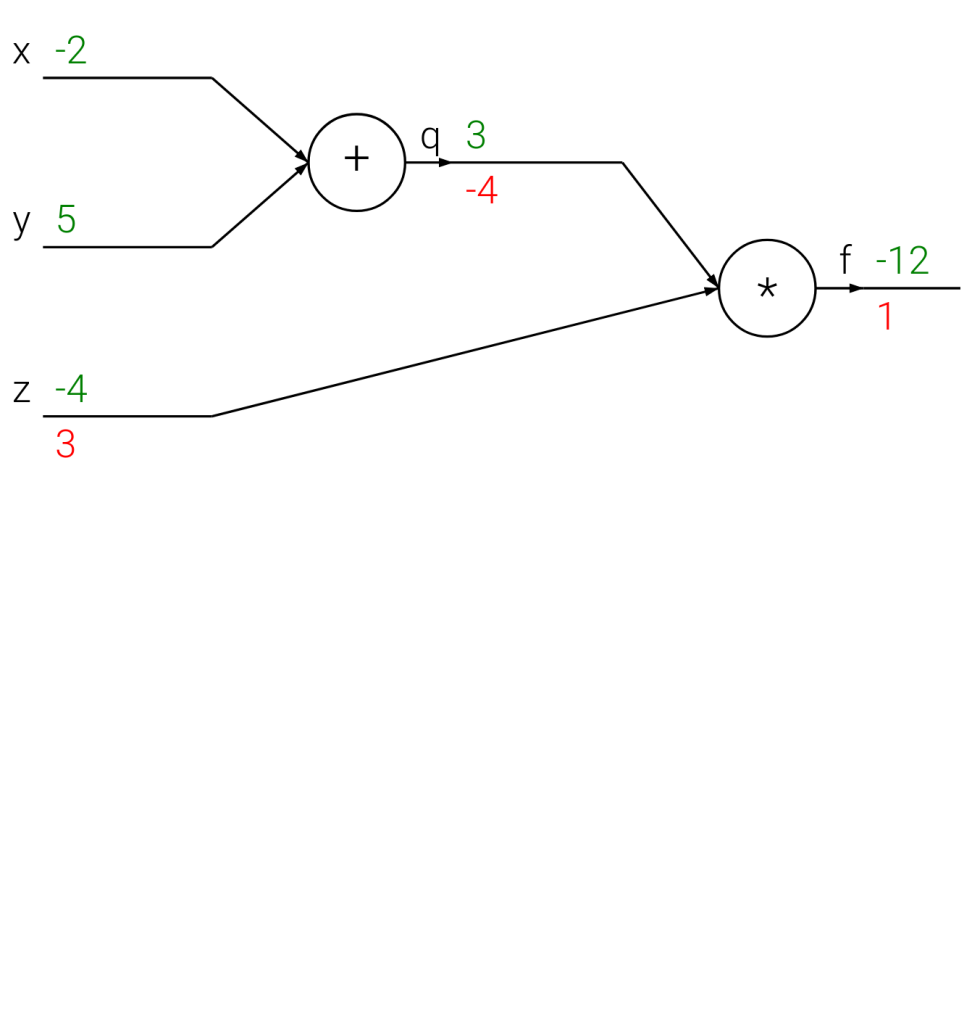
$$f(x, y, z) = (x + y)z$$

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

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

We want:

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



[1] from CS231n, lecture 4, 2019

Simple example[1]

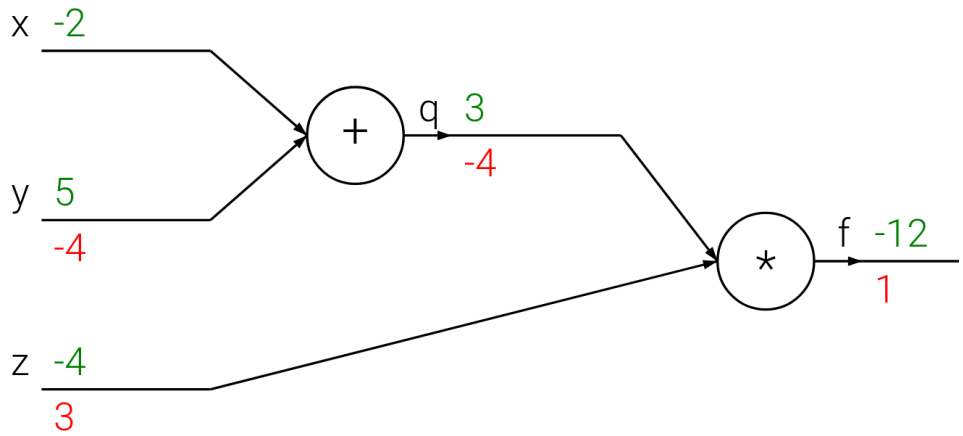
$$f(x, y, z) = (x + y)z$$

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

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

We want:

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



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

[1] from CS231n, lecture 4, 2019

Simple example[1]

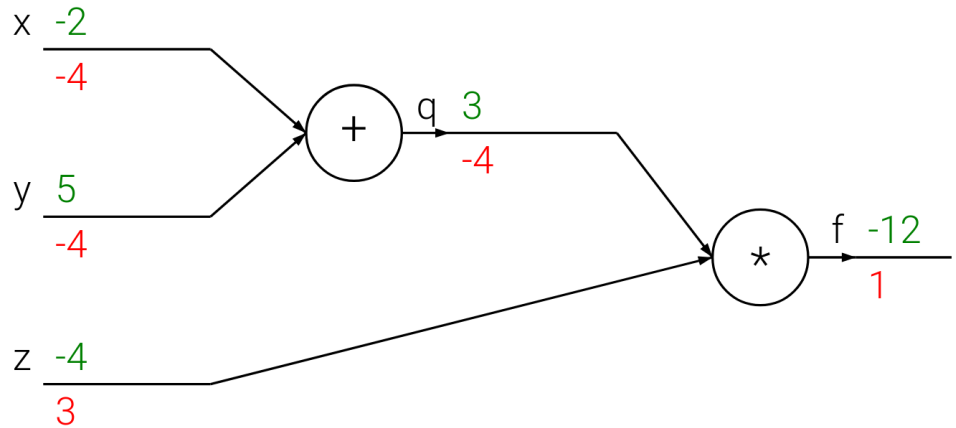
$$f(x, y, z) = (x + y)z$$

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

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

We want:

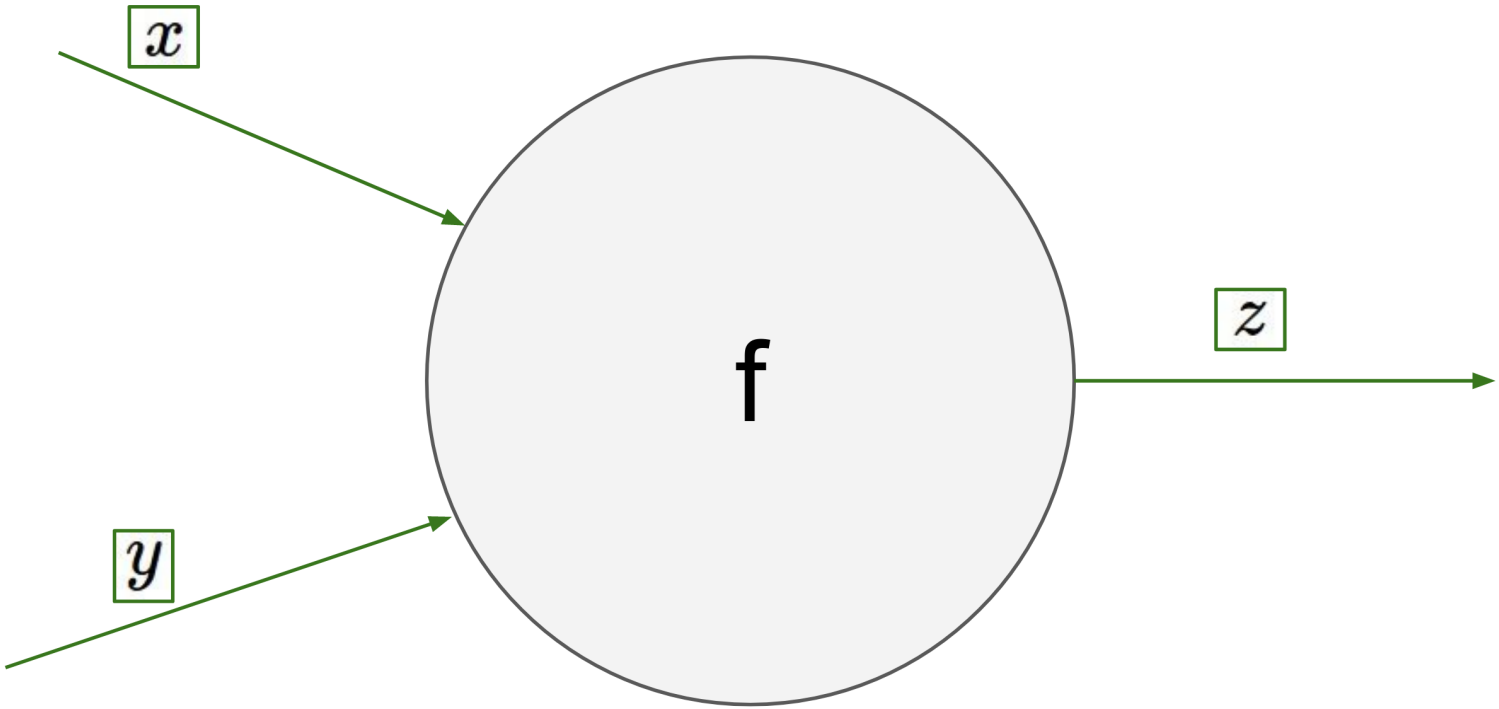
$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



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

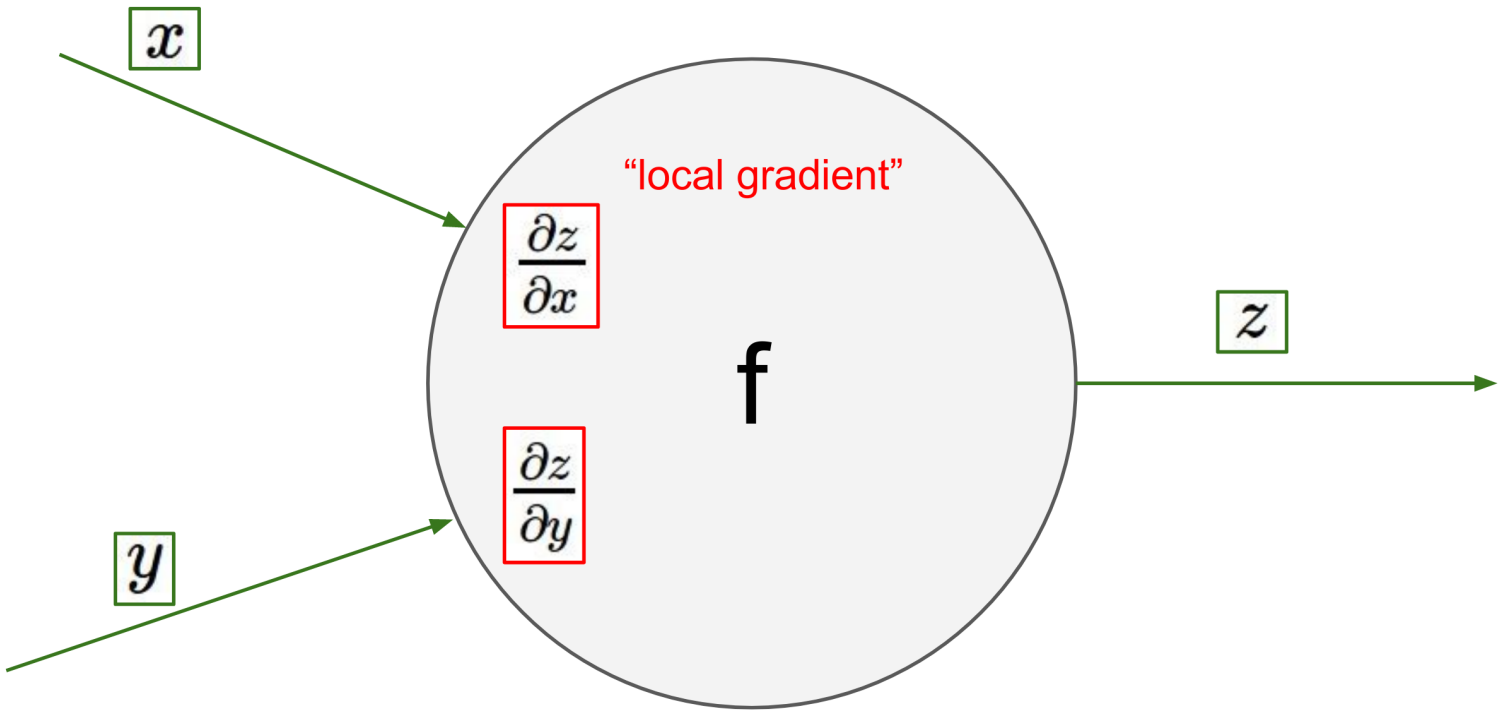
[1] from CS231n, lecture 4, 2019

Backprop[1]



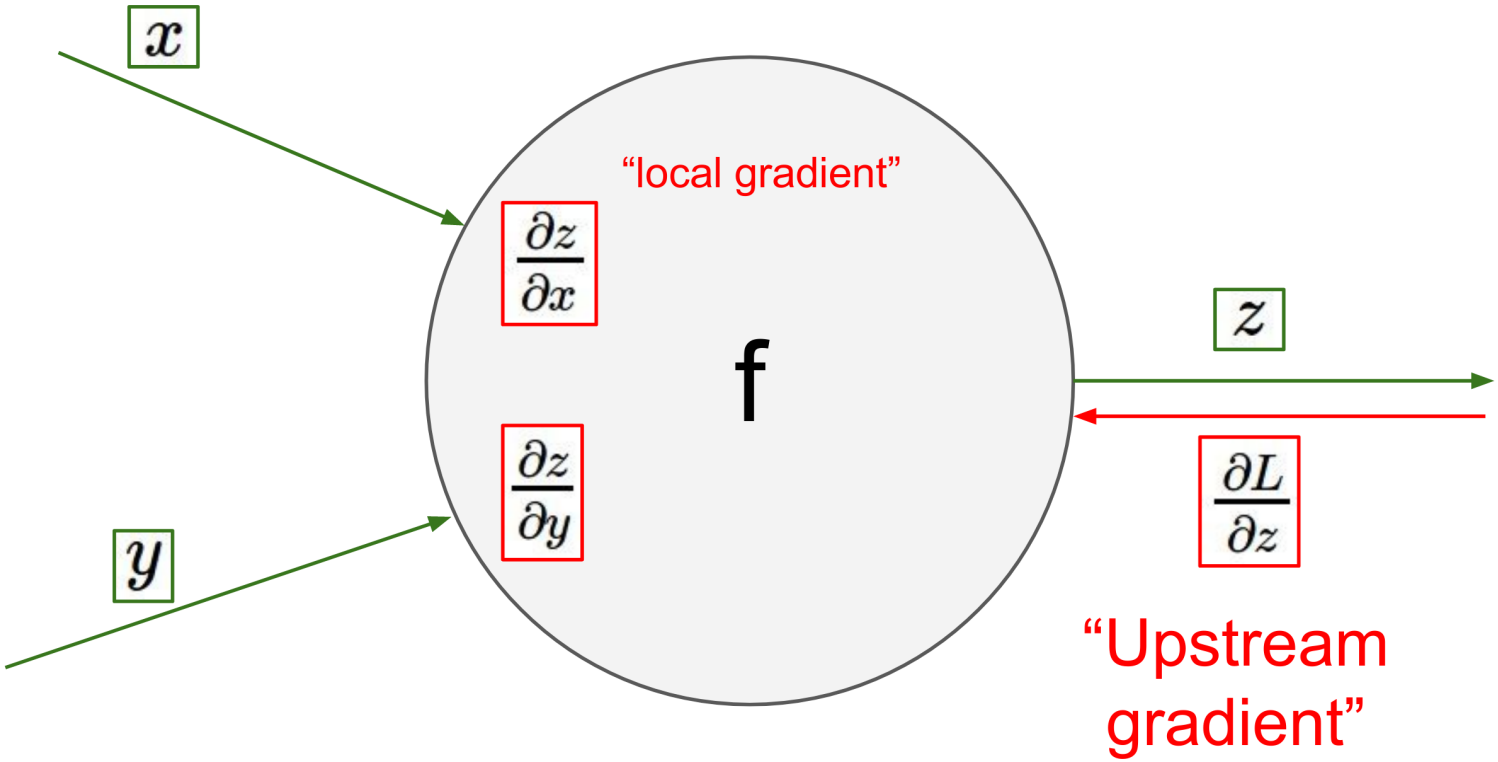
[1] from CS231n, lecture 4, 2019

Backprop[1]



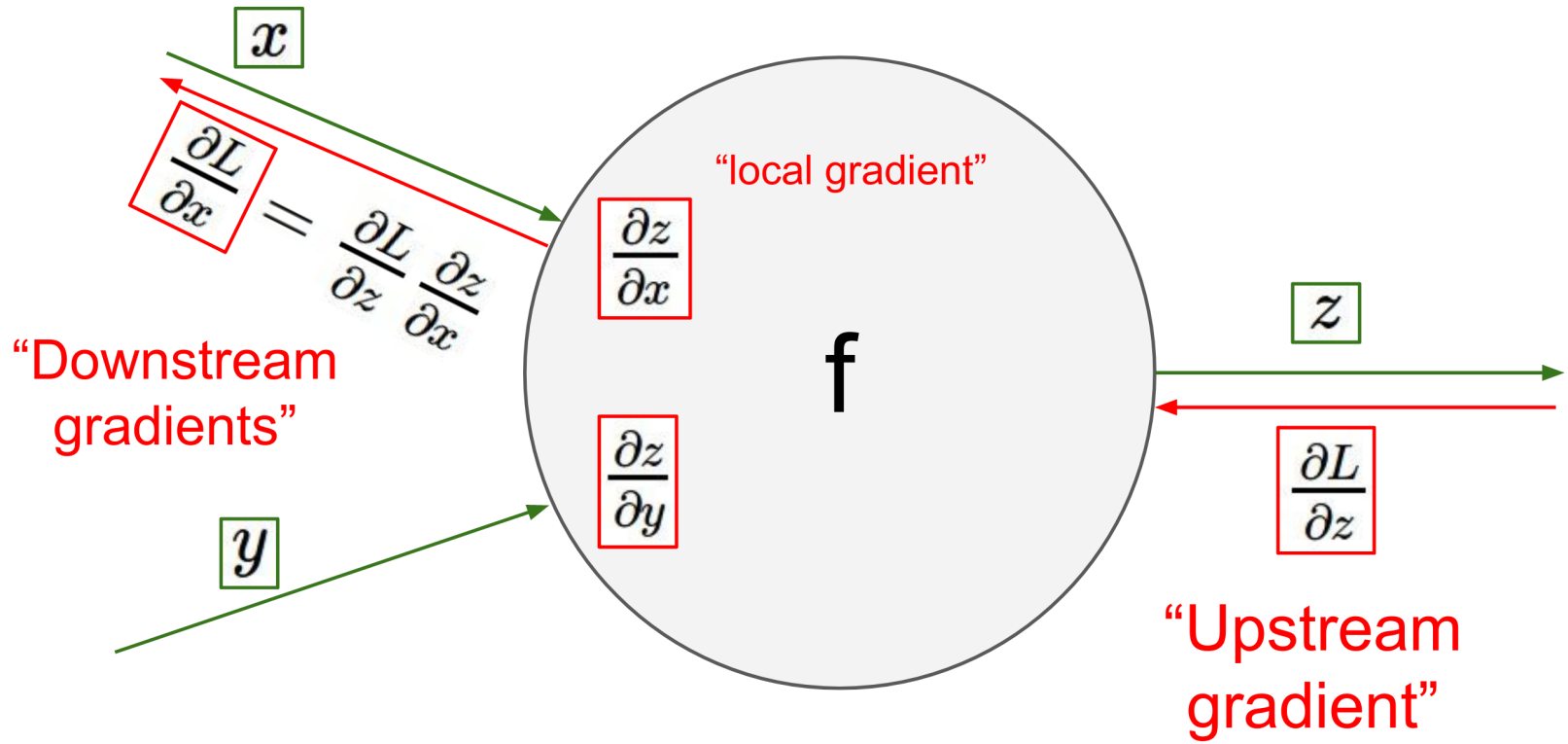
[1] from CS231n, lecture 4, 2019

Backprop[1]



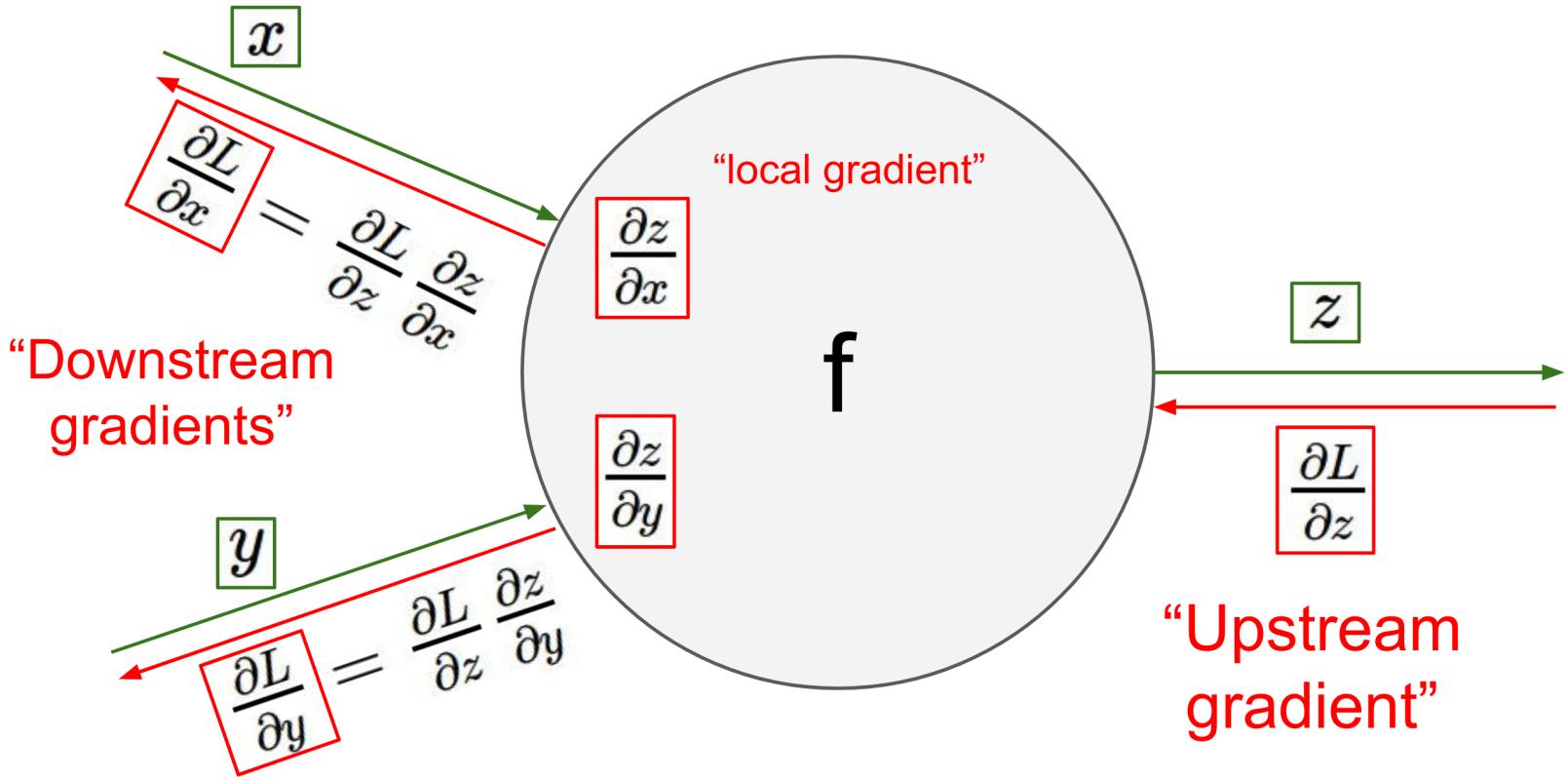
[1] from CS231n, lecture 4, 2019

Backprop[1]



[1] from CS231n, lecture 4, 2019

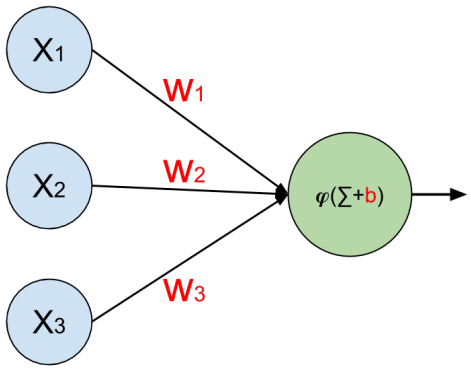
Backprop[1]



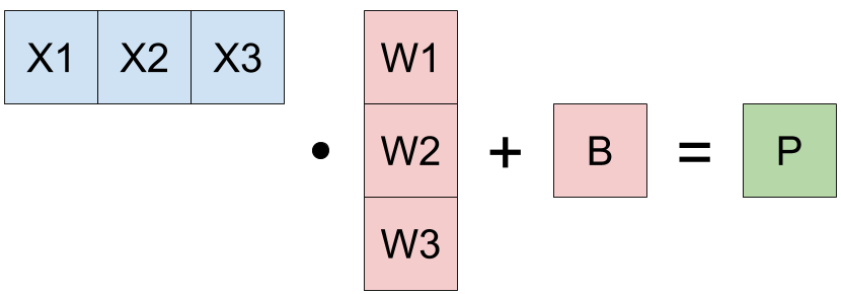
[1] from CS231n, lecture 4, 2019

Outline

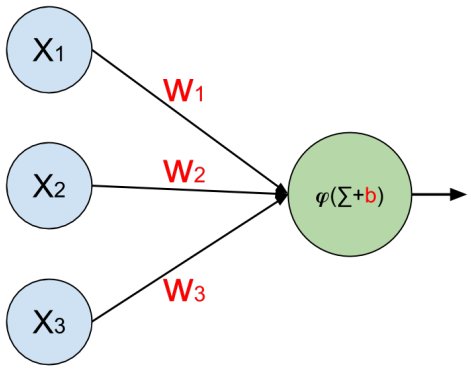
NN = Tensor Operations



Perceptron

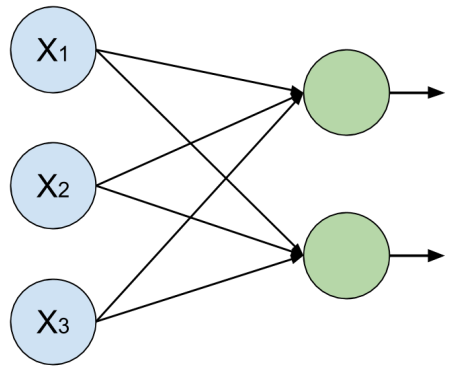


NN = Tensor Operations



Perceptron

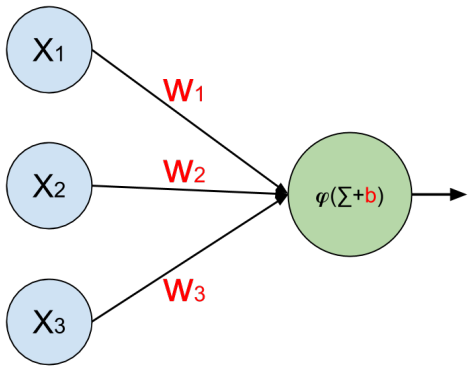
X1	X2	X3	•	W1			+	B	=	P
				W2						
				W3						



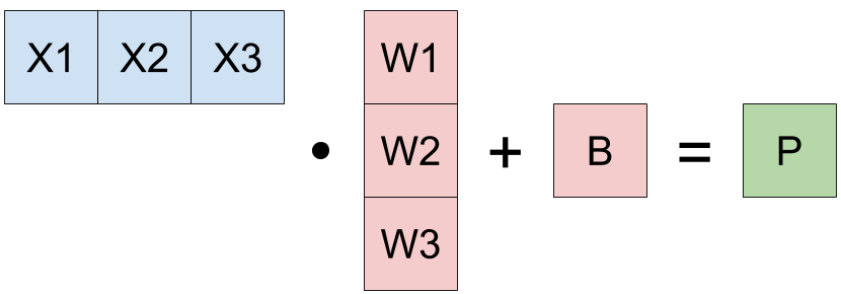
Linear Layer

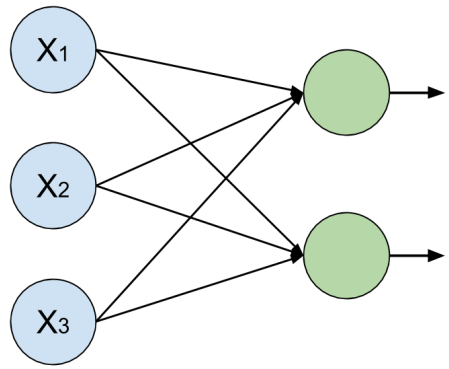
X1	X2	X3	•	W11	W12	+	B1	B2	=	P1	P2
				W21	W22						
				W31	W32						

NN = Tensor Operations

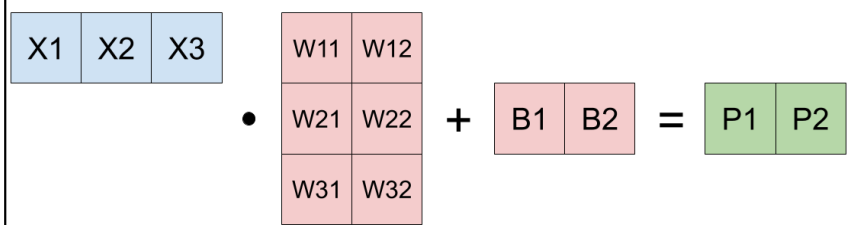


Perceptron





Linear Layer

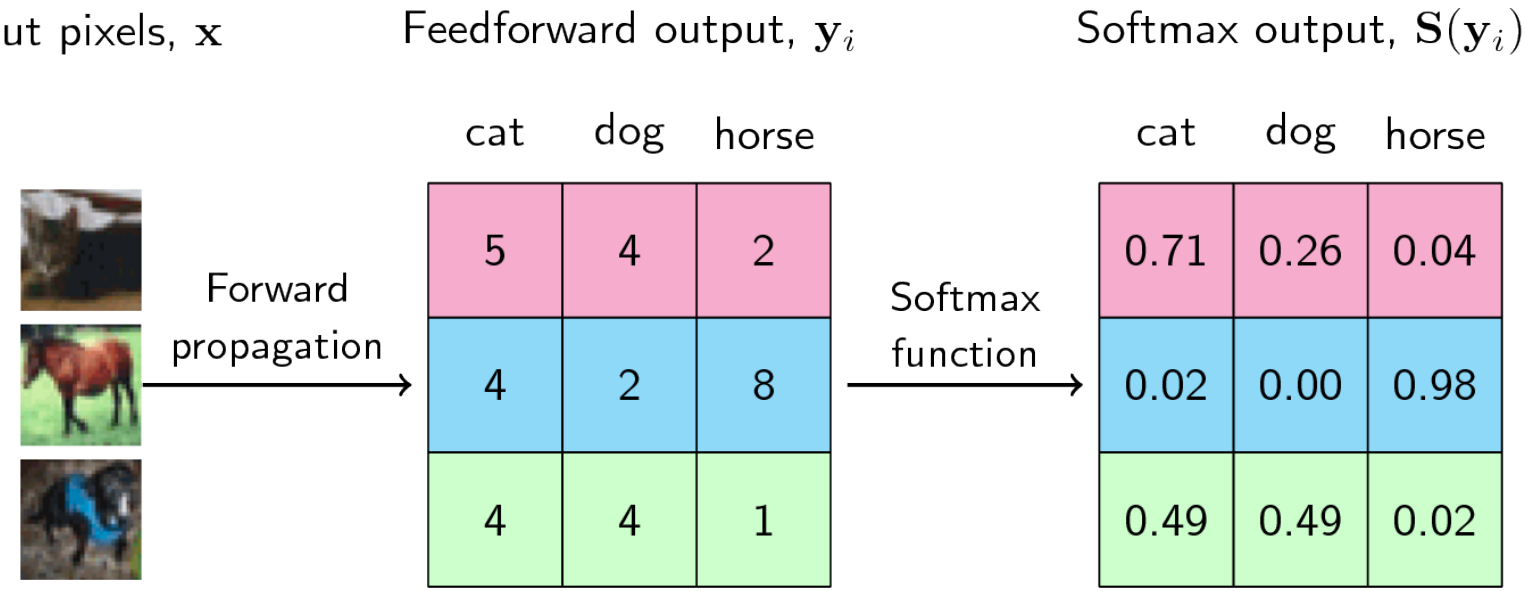


How about the case with multiple input tensors?

Softmax

Bitdefender

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$



Softmax [1]

[1] ljvmiranda921.github.io

Loss Function

Bitdefender

$\frac{1}{n}\sum_i (h_\theta(x_i) - y_i)^2$

$\frac{1}{n}\sum_i -log(h_\theta(x_i)_{y_i})$

Input pixels, $\mathbf{x}$

Softmax output, $\mathbf{S}(\mathbf{y}_i)$

Loss, $\mathbf{L}(a)$

cat	dog	horse
0.71	0.26	0.04
0.02	0.00	0.98
0.49	0.49	0.02

$-\log(a)$ at the correct classes

NLL
0.34
0.02
0.71

The correct class is highlighted in red

Total: **1.07**

NLL [1]

[1] ljvmiranda921.github.io

Outline

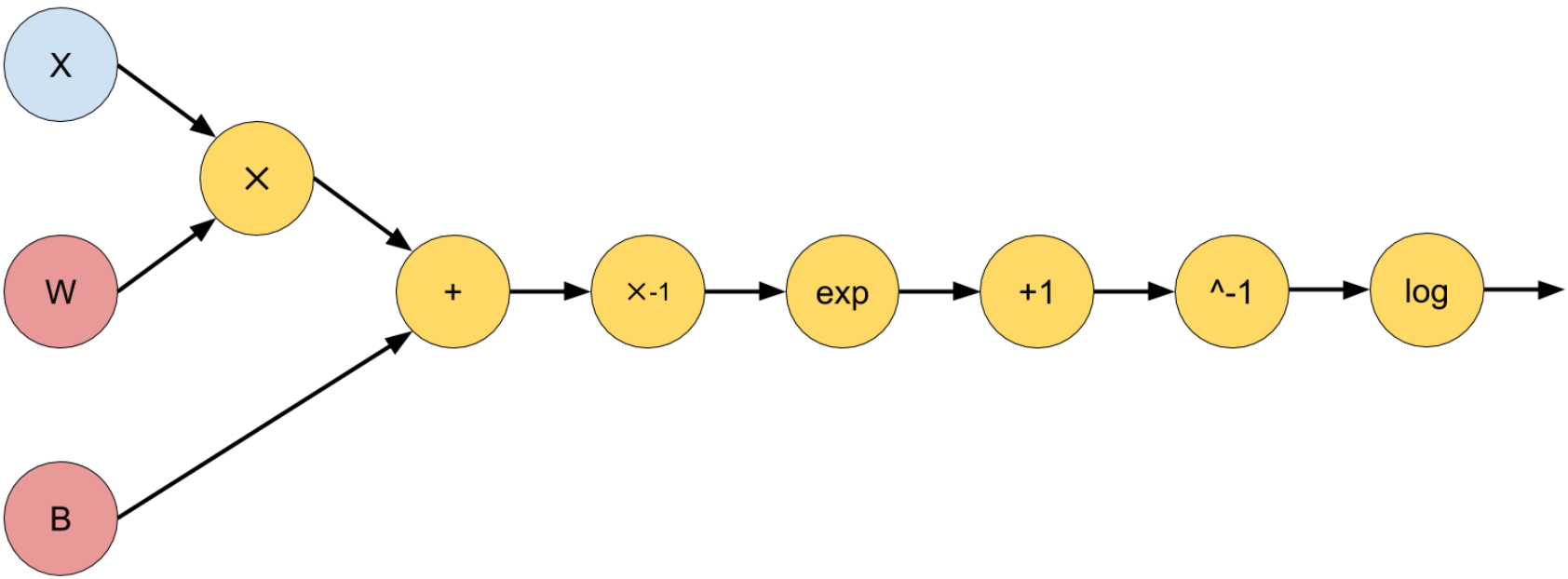
Computation Graph

Bitdefender

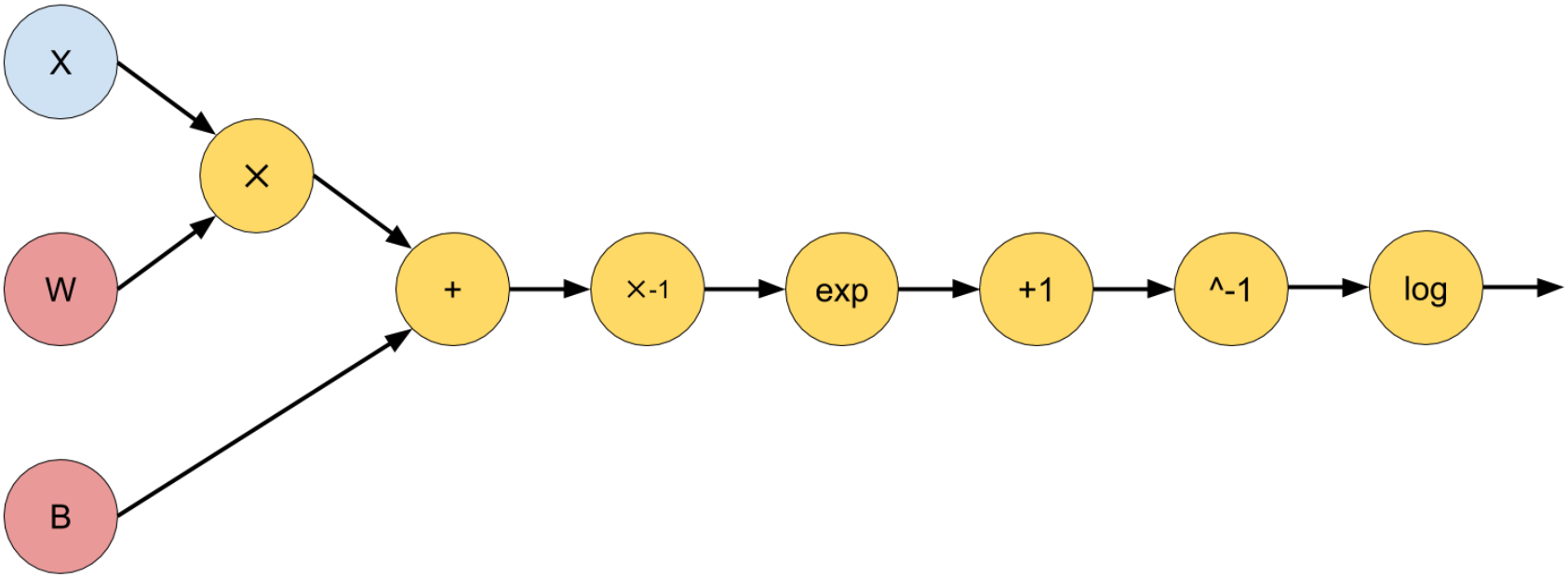
$$L(x, w, b) = log\left(\frac{1}{e^{-(xw+b)} + 1}\right)$$

Computation Graph

$$L(x, w, b) = log\left(\frac{1}{e^{-(xw+b)} + 1}\right)$$

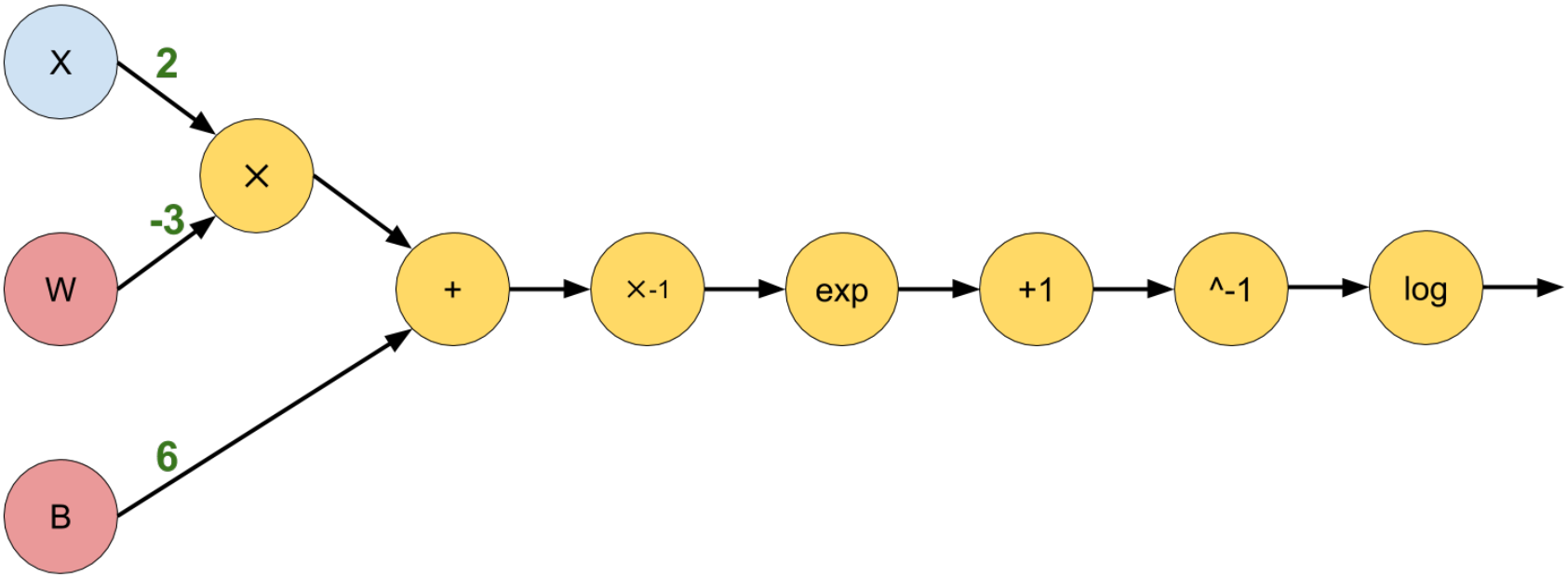


Forward



Task: Compute $L(x, w, b)$

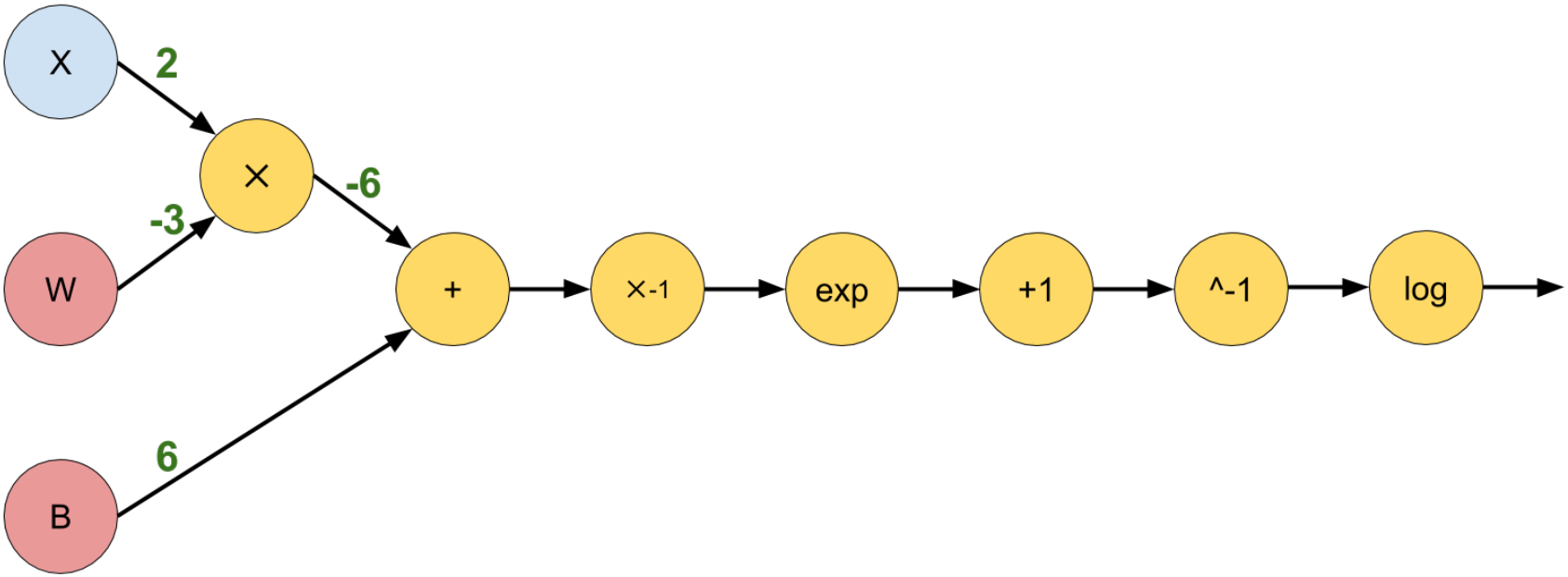
Forward



Task: Compute $L(x, w, b)$

$$x = 2, w = -3, b = 6$$

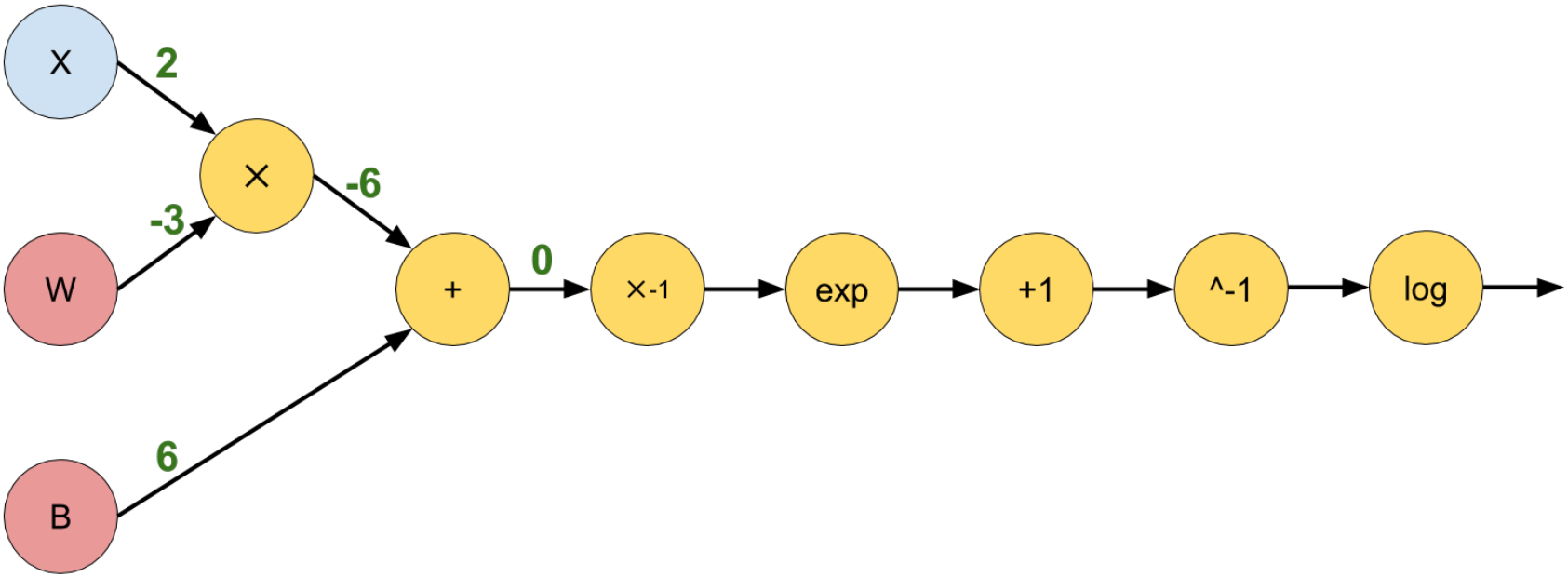
Forward



Task: Compute $L(x, w, b)$

$$x = 2, w = -3, b = 6$$

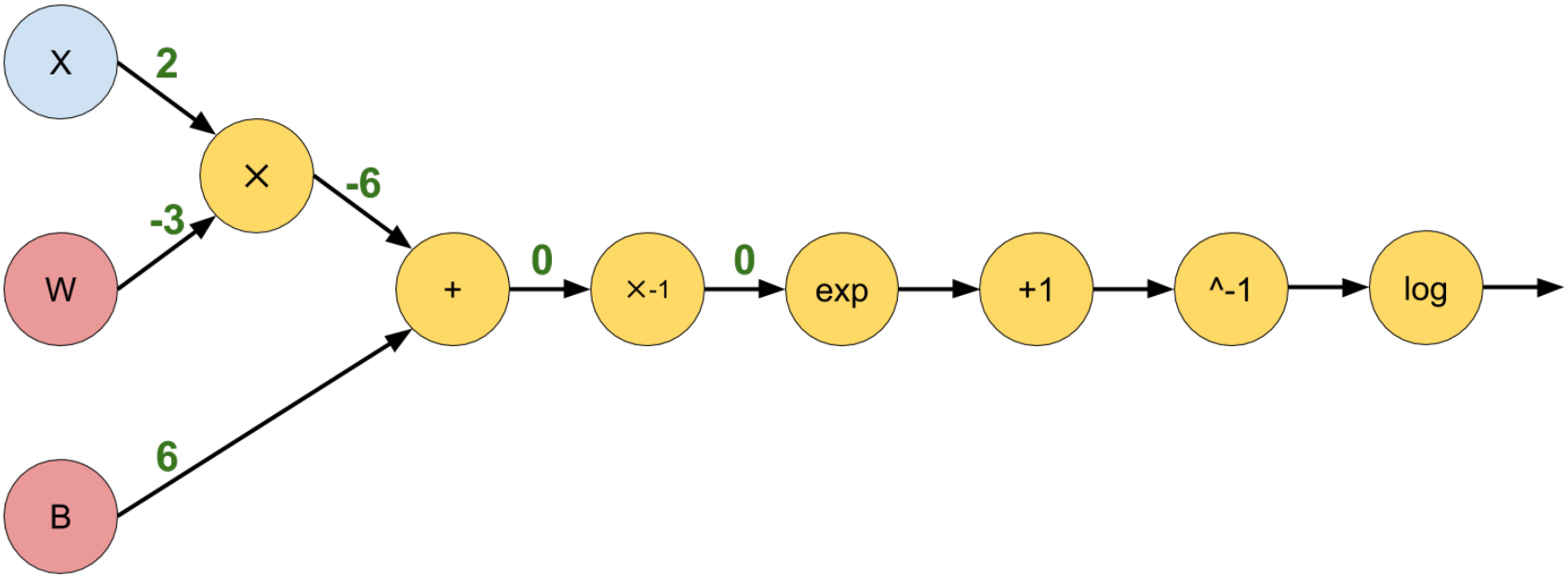
Forward



Task: Compute $L(x, w, b)$

$$x = 2, w = -3, b = 6$$

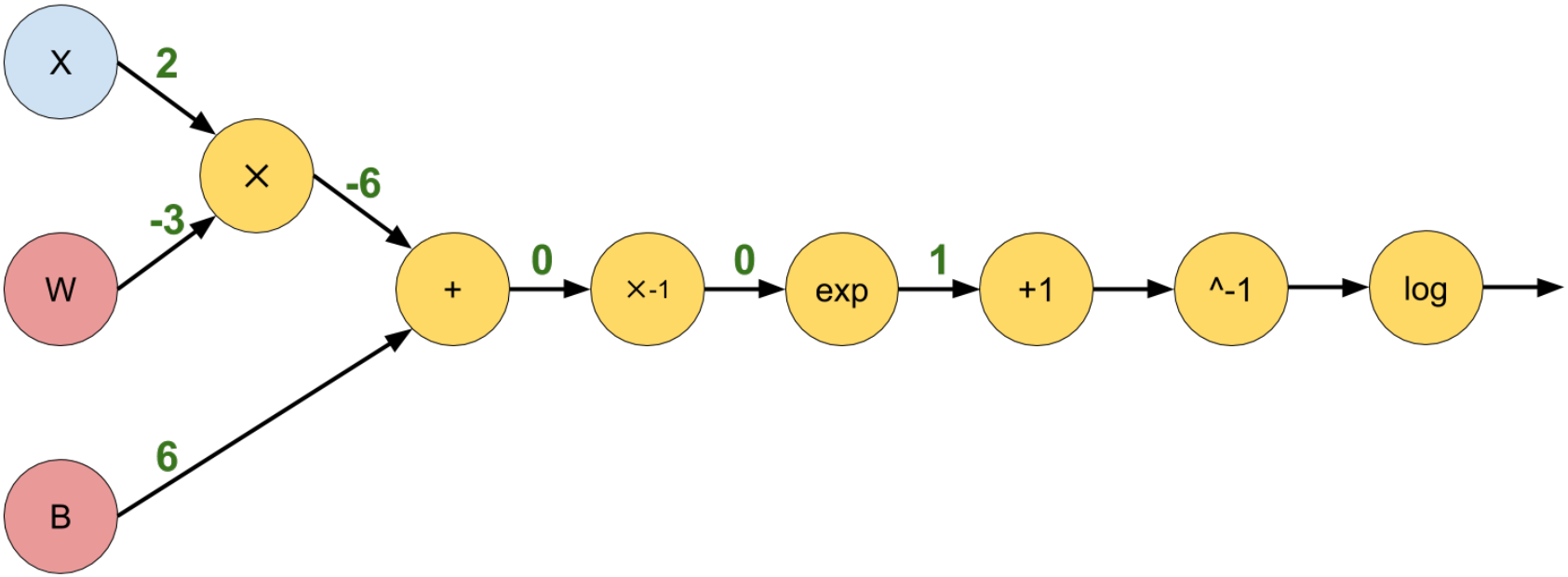
Forward



Task: Compute $L(x, w, b)$

$$x = 2, w = -3, b = 6$$

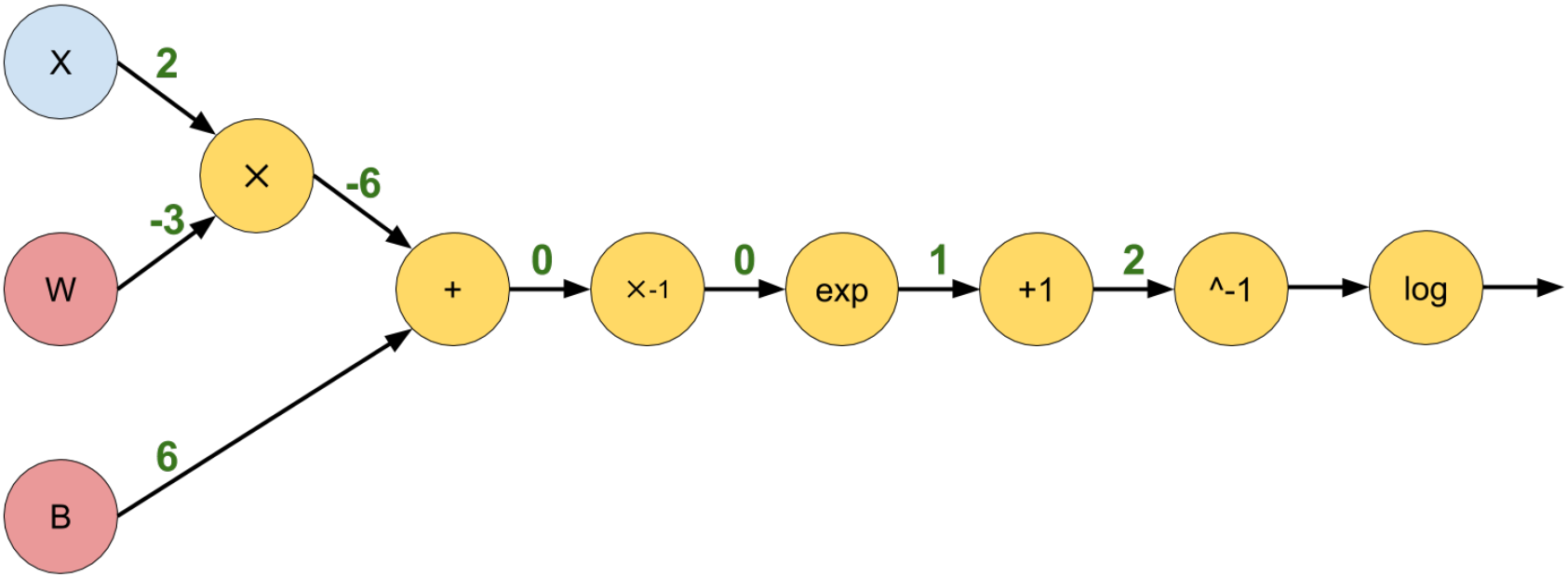
Forward



Task: Compute $L(x, w, b)$

$$x = 2, w = -3, b = 6$$

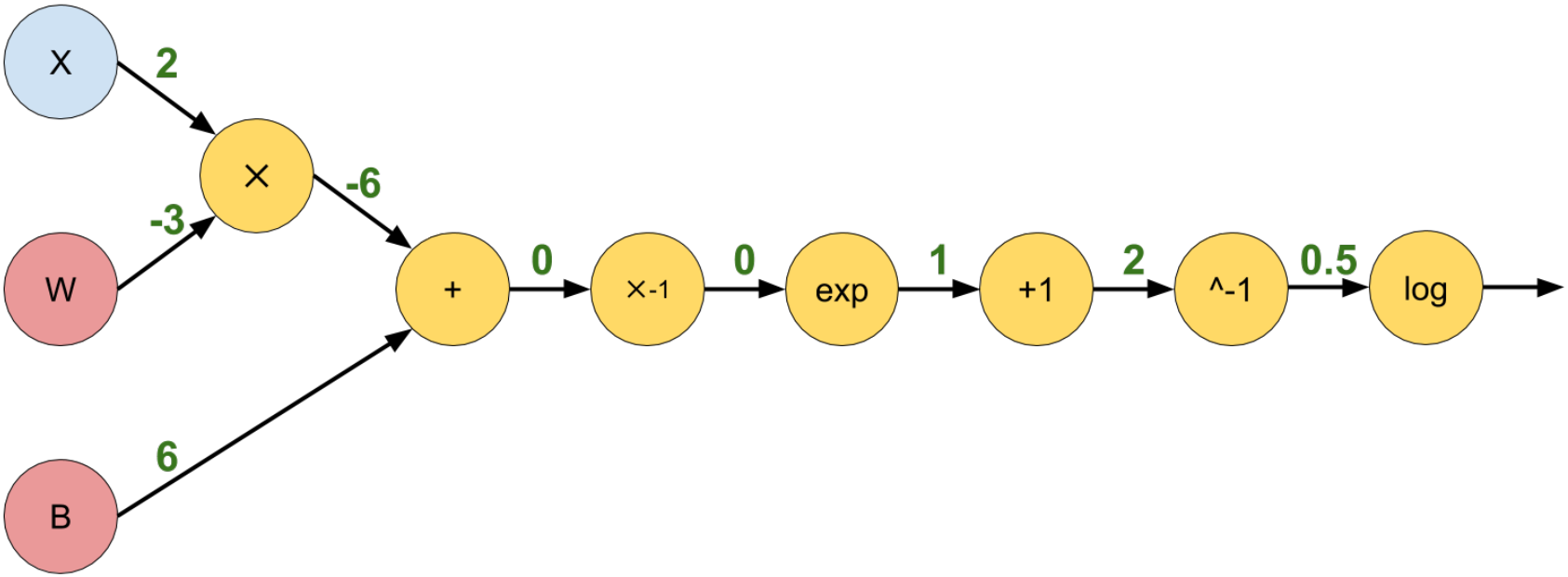
Forward



Task: Compute $L(x, w, b)$

$$x = 2, w = -3, b = 6$$

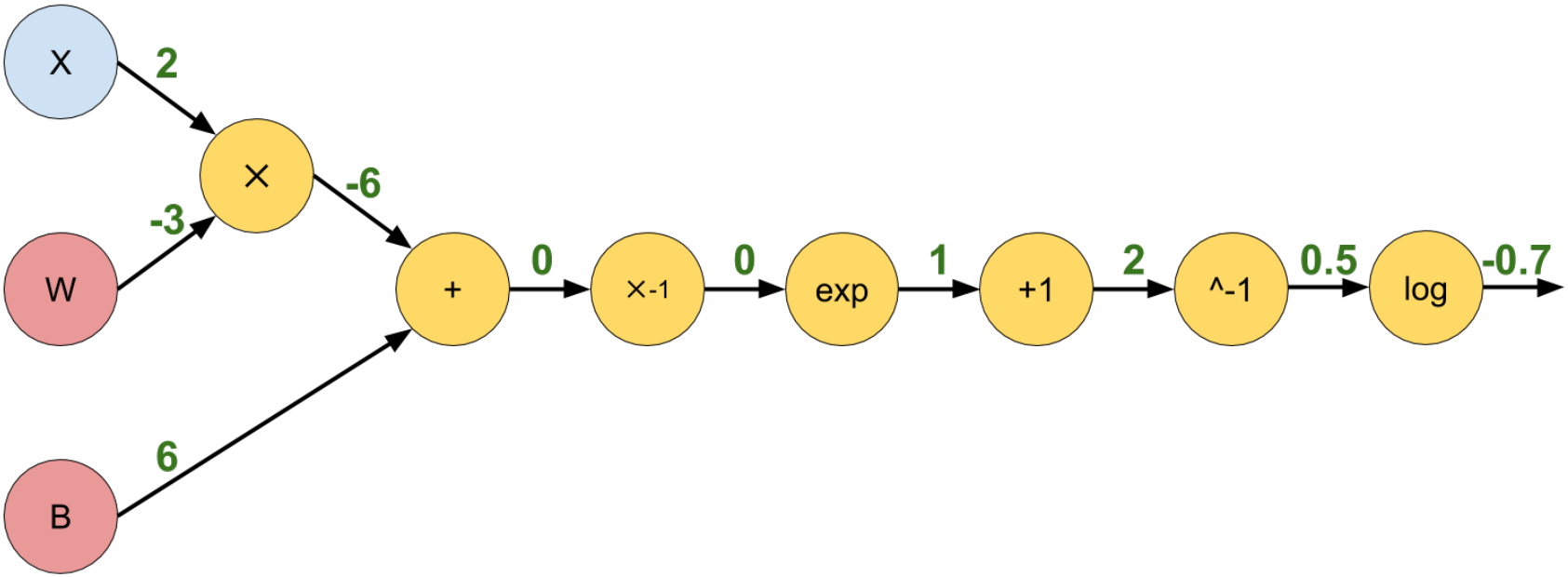
Forward



Task: Compute $L(x, w, b)$

$$x = 2, w = -3, b = 6$$

Forward

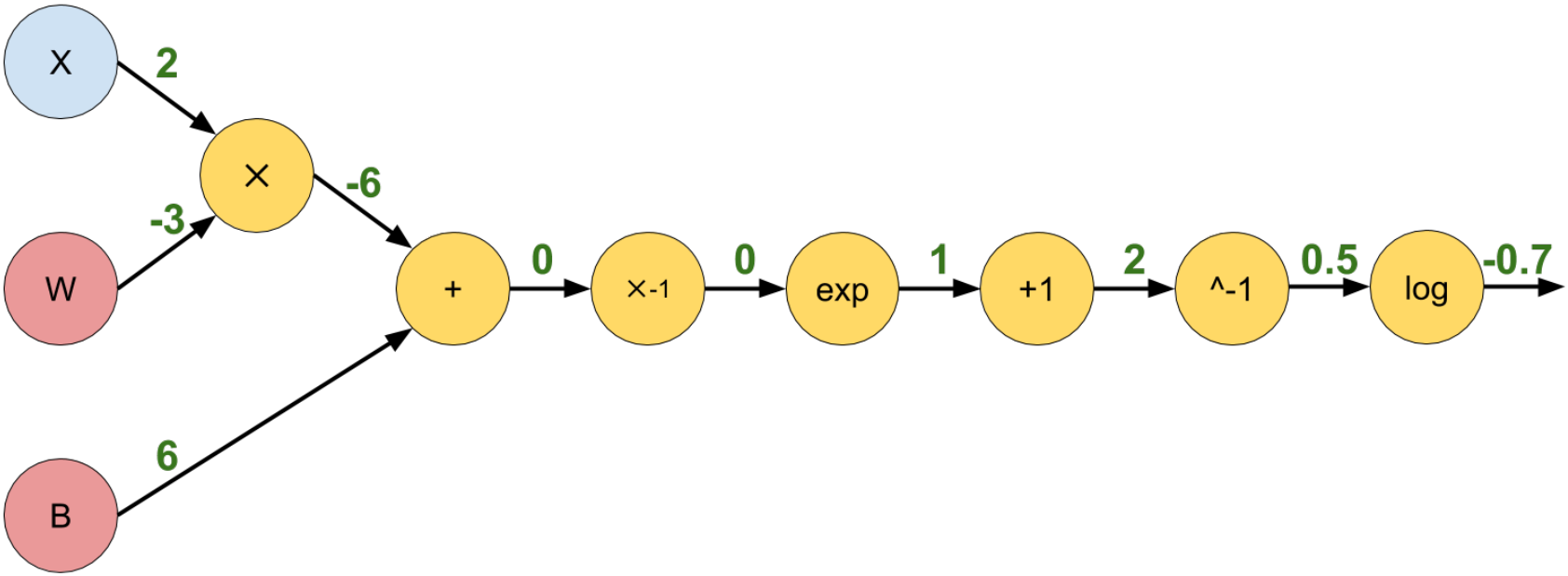


Task: Compute $L(x, w, b)$

$$x = 2, w = -3, b = 6$$

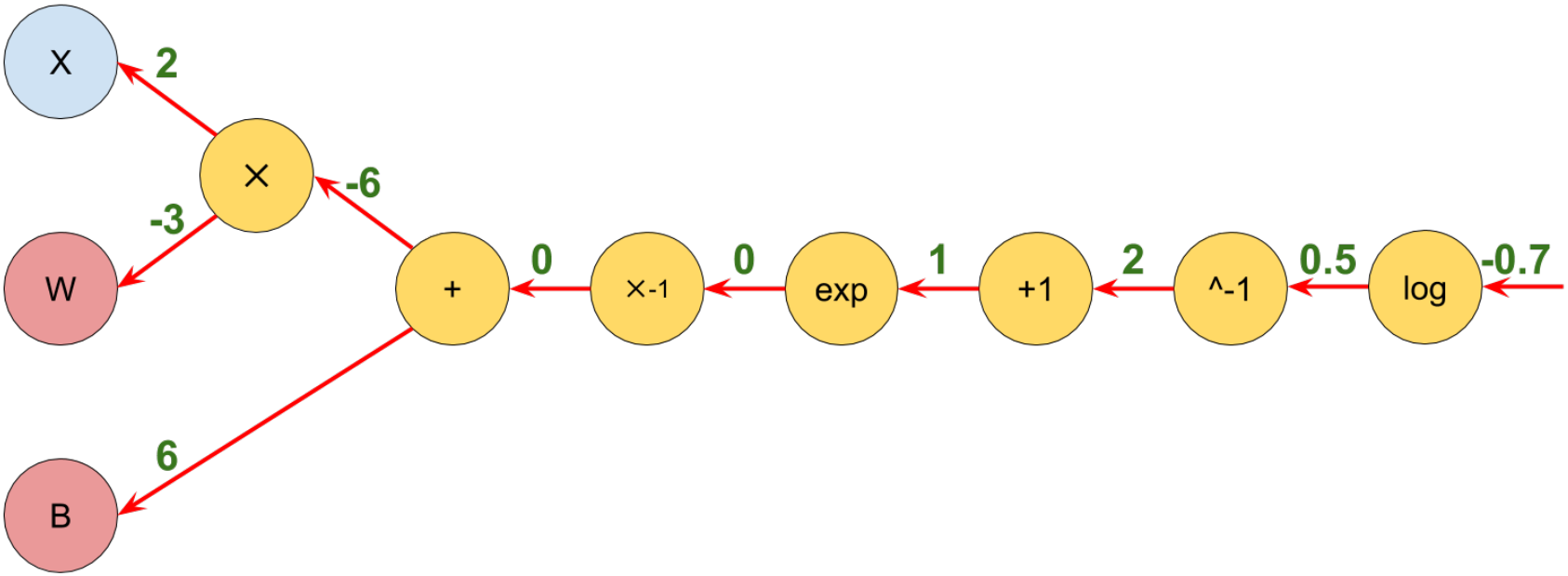
$$L(2, -3, 6) \simeq 0.7$$

Backward



Task: Compute $\nabla L = [\frac{\partial L}{\partial x}, \frac{\partial L}{\partial w}, \frac{\partial L}{\partial b}]$

Backward

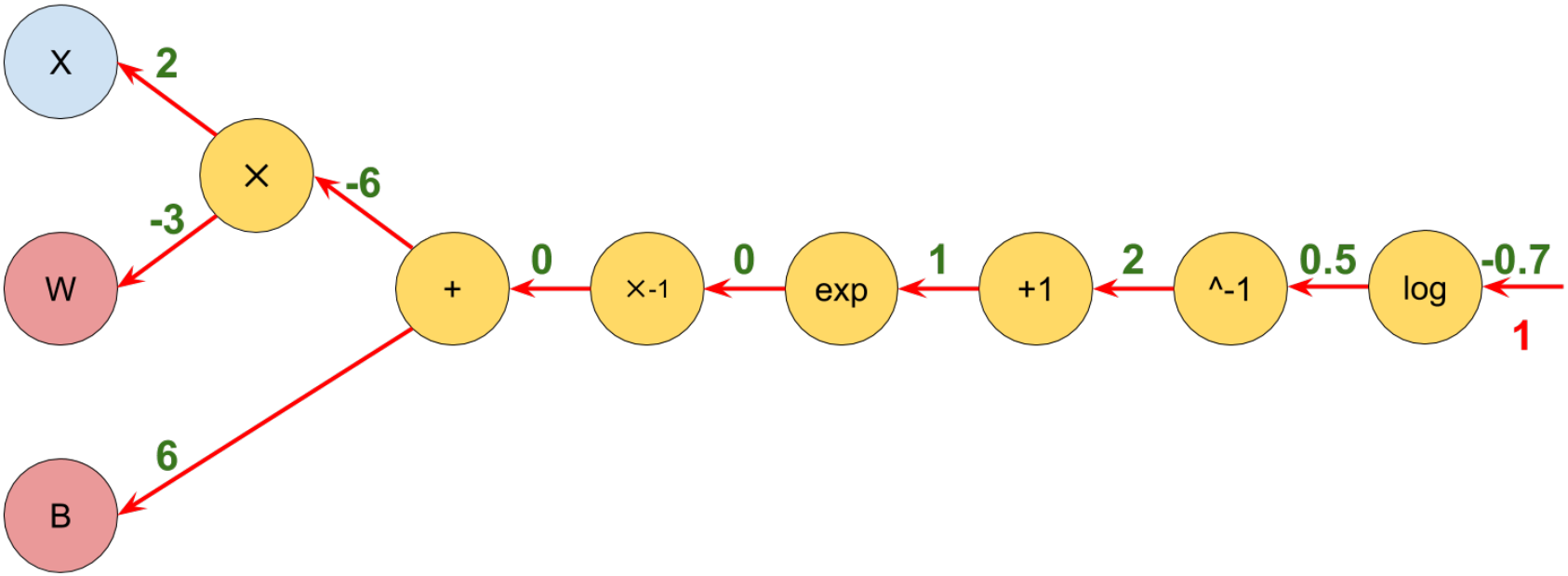


Task: Compute $\nabla L = \left[\frac{\partial L}{\partial x}, \frac{\partial L}{\partial w}, \frac{\partial L}{\partial b}\right]$

Chain-rule $\implies$ each sub-module M:

- receives $\frac{\partial L}{\partial M(in)}$
- computes $\frac{\partial M(in)}{\partial in_k}$
- returns $\frac{\partial L}{\partial in_k} = \frac{\partial L}{\partial M(in)} \frac{\partial M(in)}{\partial in_k}$

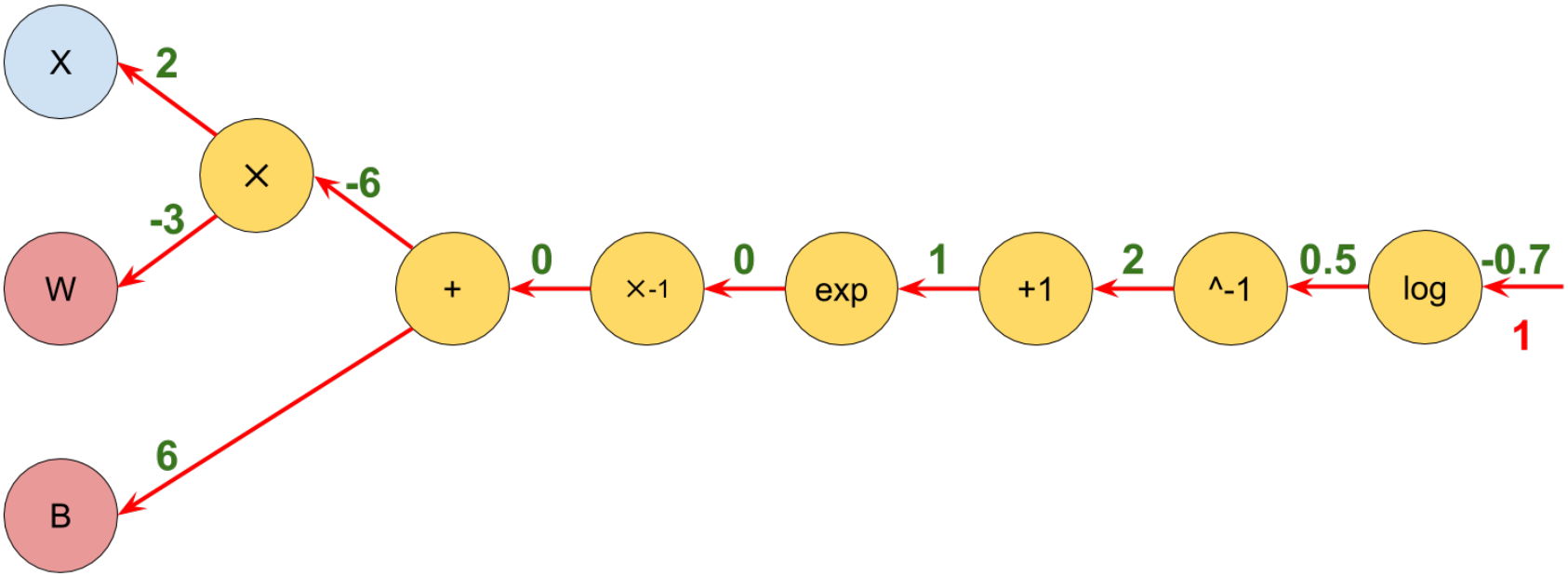
Backward



log:

- receives $\frac{\partial L}{\partial log(in)} = 1$

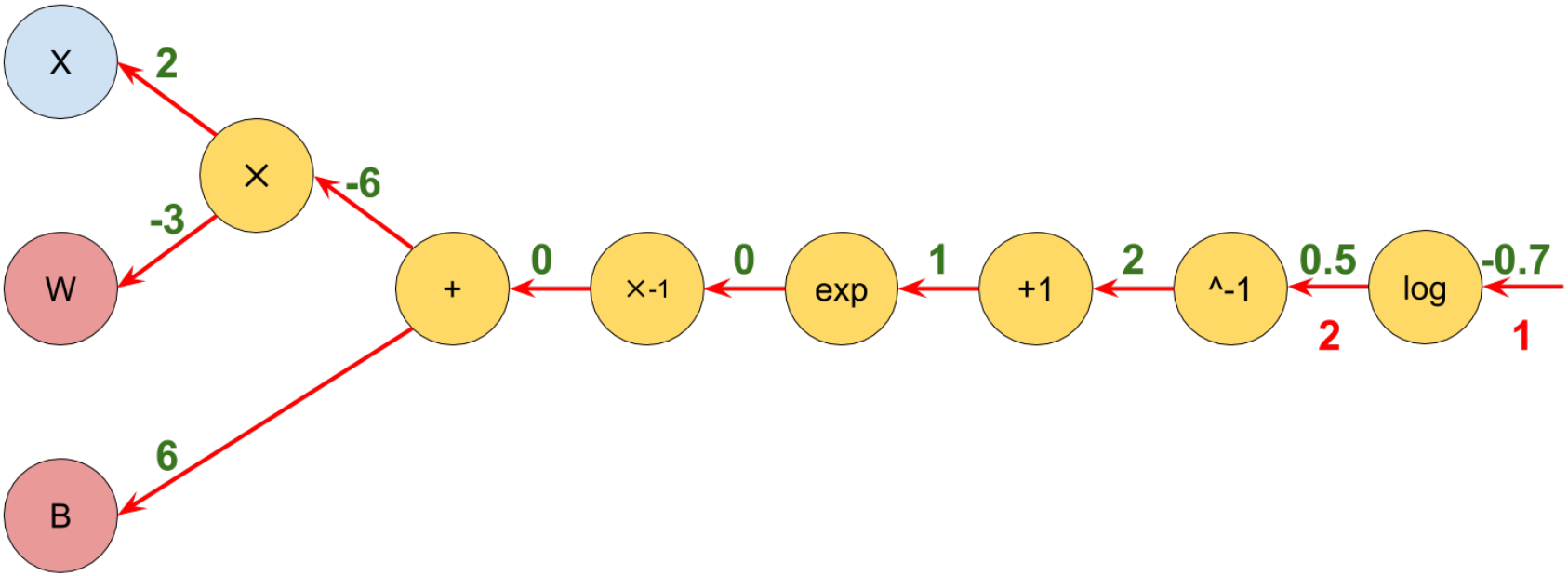
Backward



log:

- receives $\frac{\partial L}{\partial log(in)} = 1$
- computes $\frac{\partial log(in)}{\partial in} = \frac{1}{in} = 2$

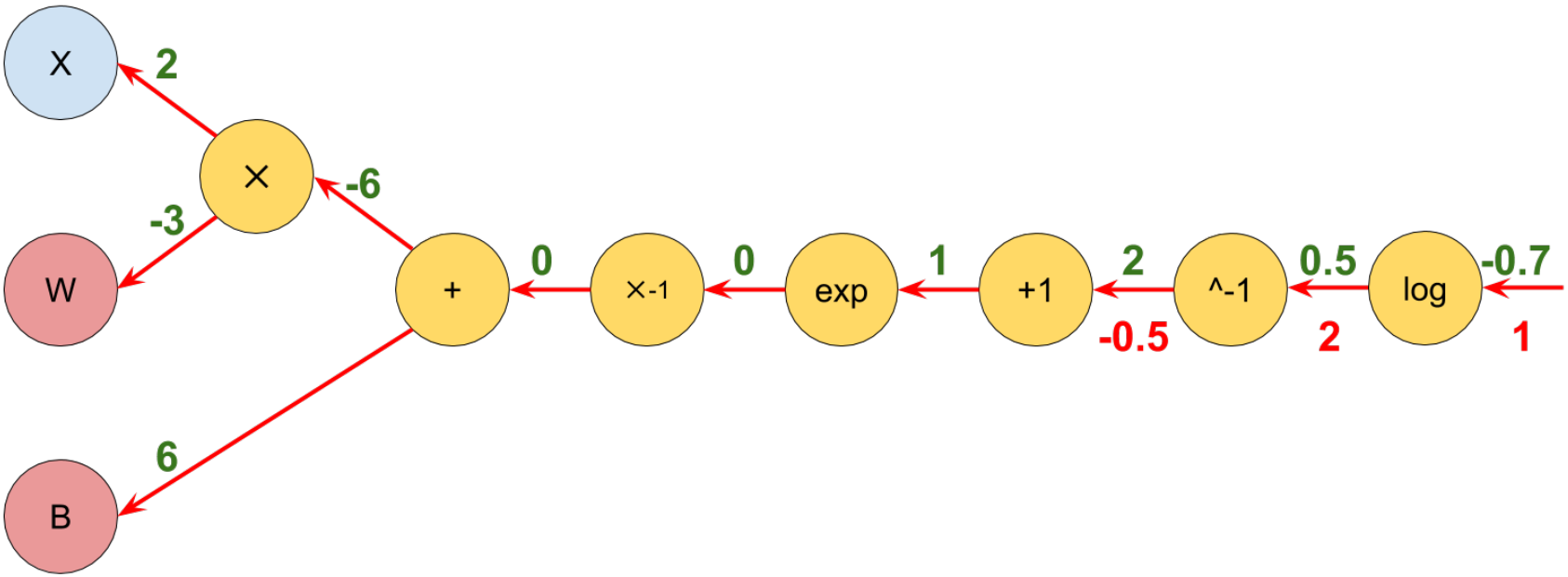
Backward



log:

- receives $\frac{\partial L}{\partial log(in)} = 1$
- computes $\frac{\partial log(in)}{\partial in} = \frac{1}{in} = 2$
- returns $\frac{\partial L}{\partial in} = 1 * 2$

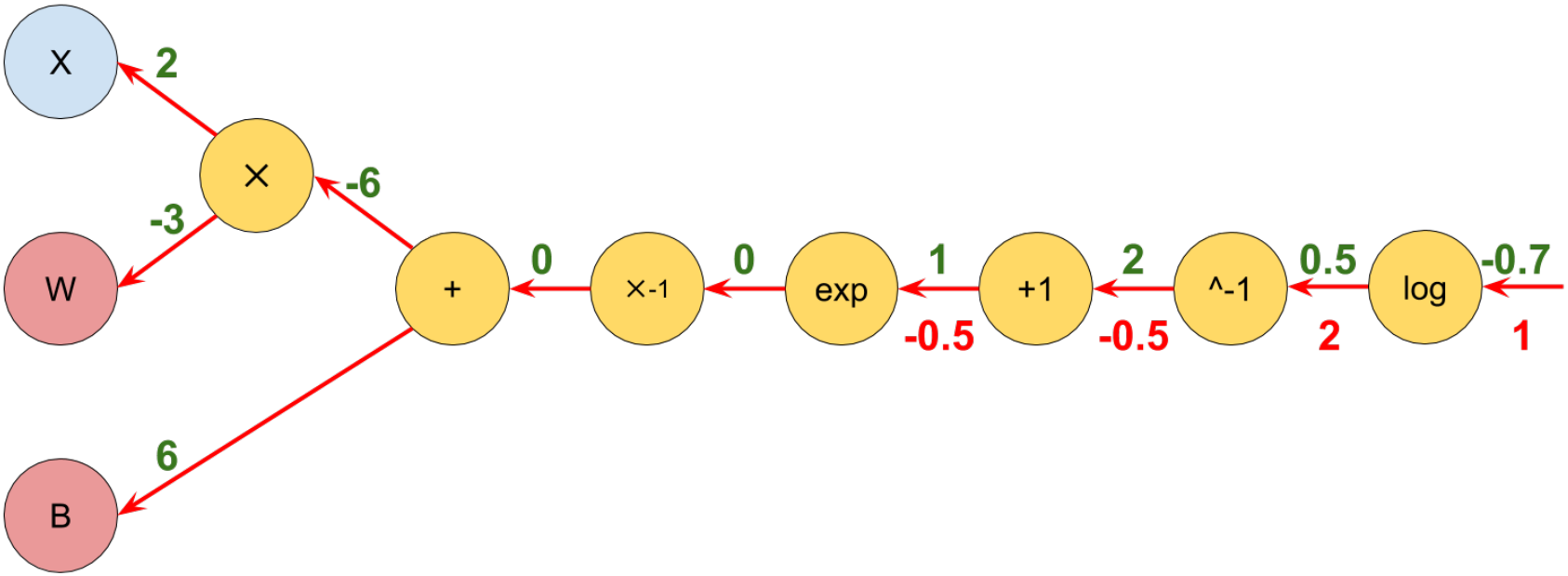
Backward



$\hat{}-1$:

- receives $\frac{\partial L}{\partial (in)^{-1}} = 2$
- computes $\frac{\partial (in)^{-1}}{\partial in} = -(in)^{-2} = -0.25$
- returns $\frac{\partial L}{\partial in} = 2 * -0.25 = -0.5$

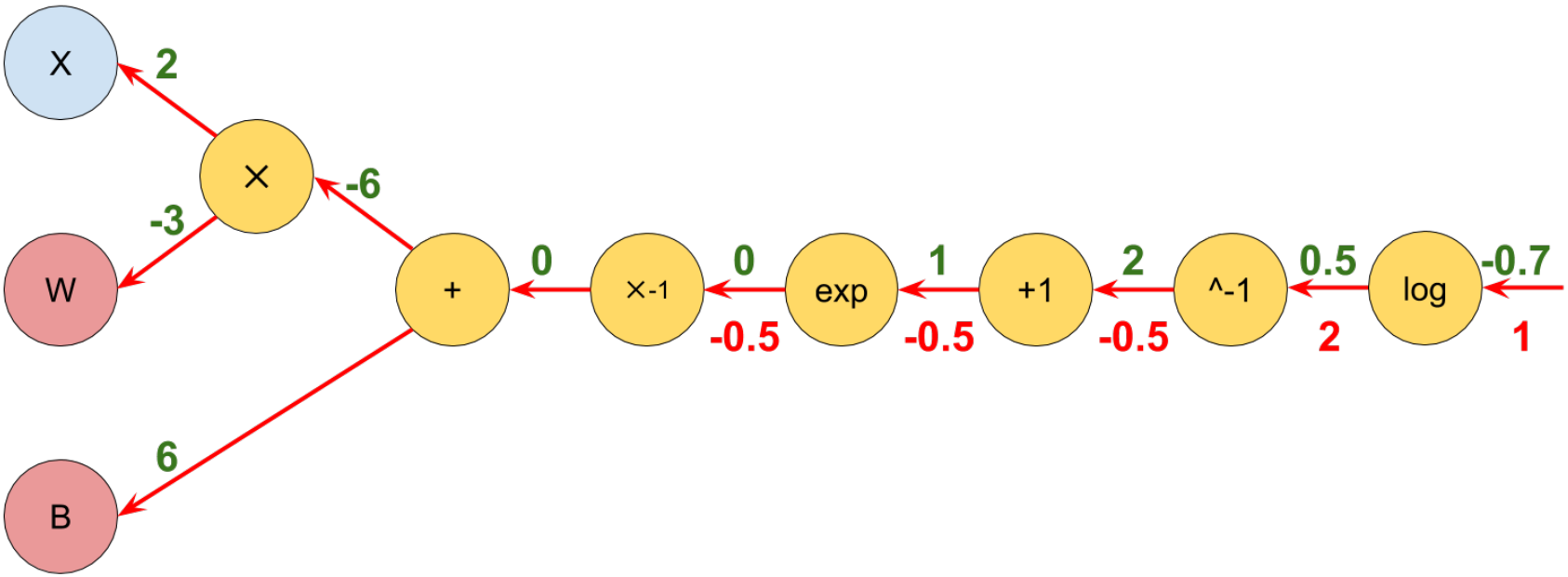
Backward



$+1$:

- receives $\frac{\partial L}{\partial (in+1)} = -0.5$
- computes $\frac{\partial (in+1)}{\partial in} = 1$
- returns $\frac{\partial L}{\partial in} = -0.5 * 1 = -0.5$

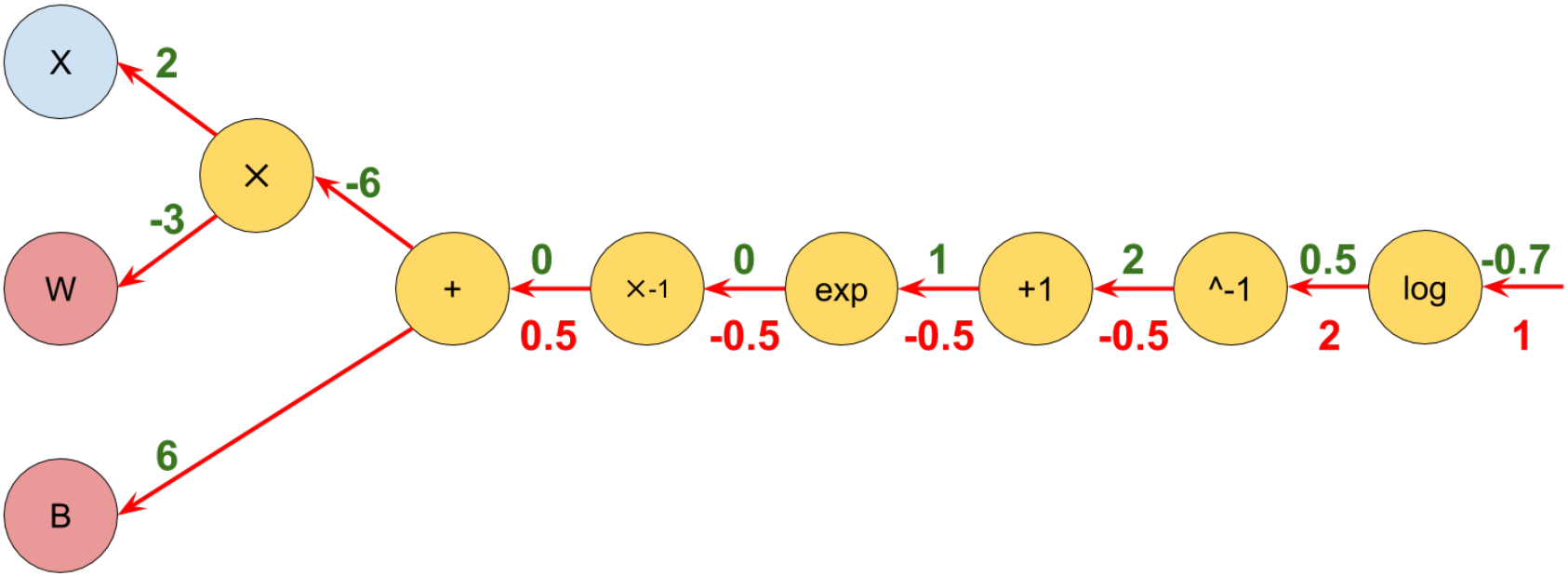
Backward



exp:

- receives $\frac{\partial L}{\partial e^{in}} = -0.5$
- computes $\frac{\partial e^{in}}{\partial in} = e^{in} = 1$
- returns $\frac{\partial L}{\partial in} = -0.5 * 1 = -0.5$

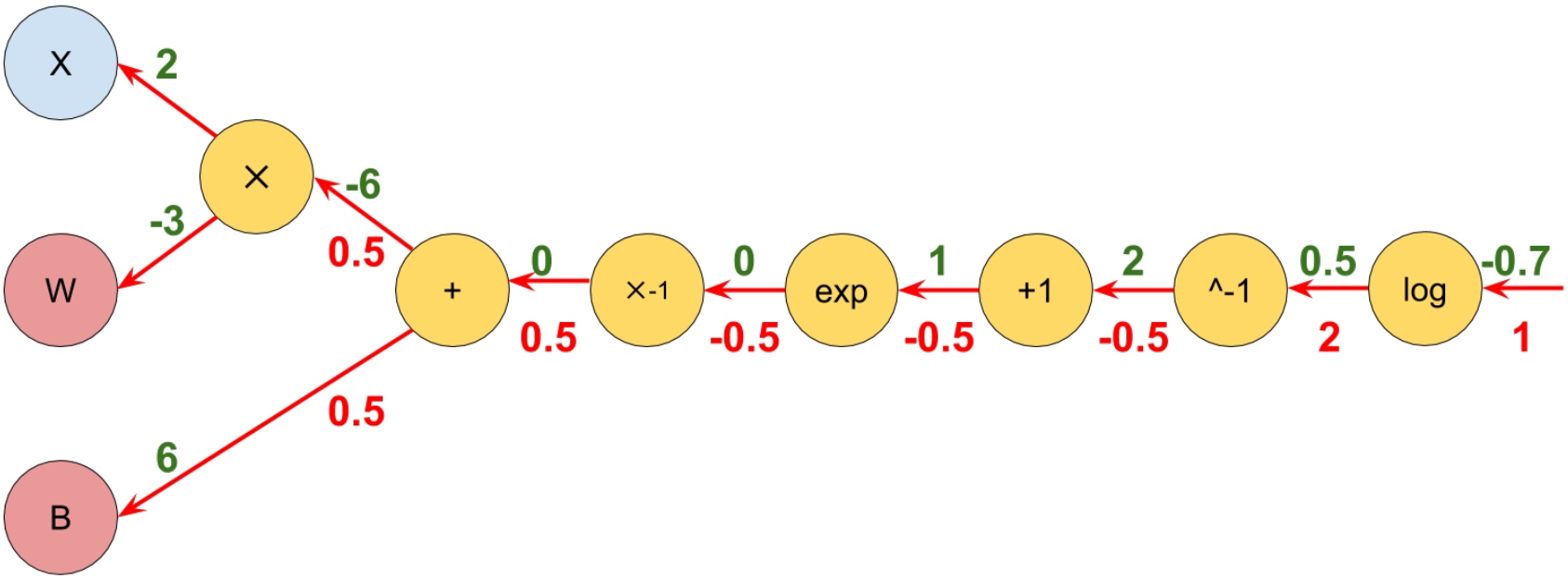
Backward



$\times -1$:

- receives $\frac{\partial L}{\partial(-in)} = -0.5$
- computes $\frac{\partial(-in)}{\partial in} = -1$
- returns $\frac{\partial L}{\partial in} = -0.5 * -1 = 0.5$

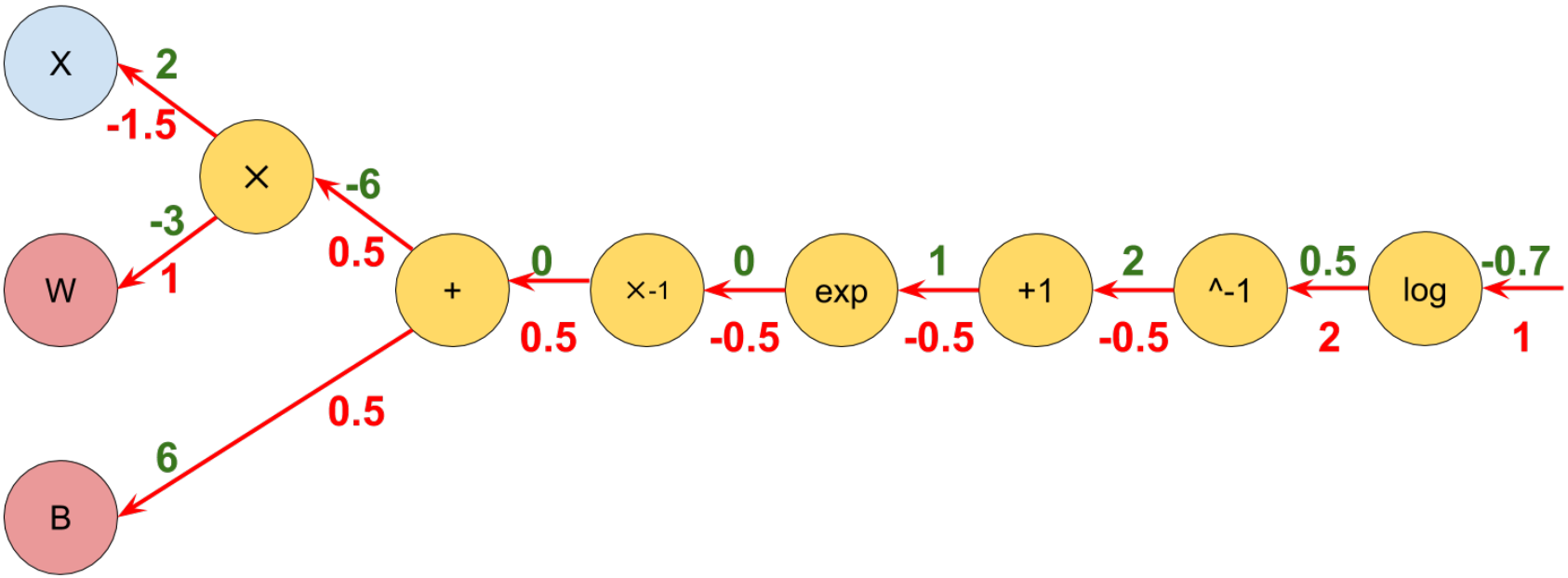
Backward



+:

- receives $\frac{\partial L}{\partial(in_1+in_2)} = 0.5$
- computes $(\frac{\partial(in_1+in_2)}{\partial in_1}, \frac{\partial(in_1+in_2)}{\partial in_2} = (1, 1)$
- returns $(\frac{\partial L}{\partial in_1}, \frac{\partial L}{\partial in_2}) = (0.5, 0.5)$

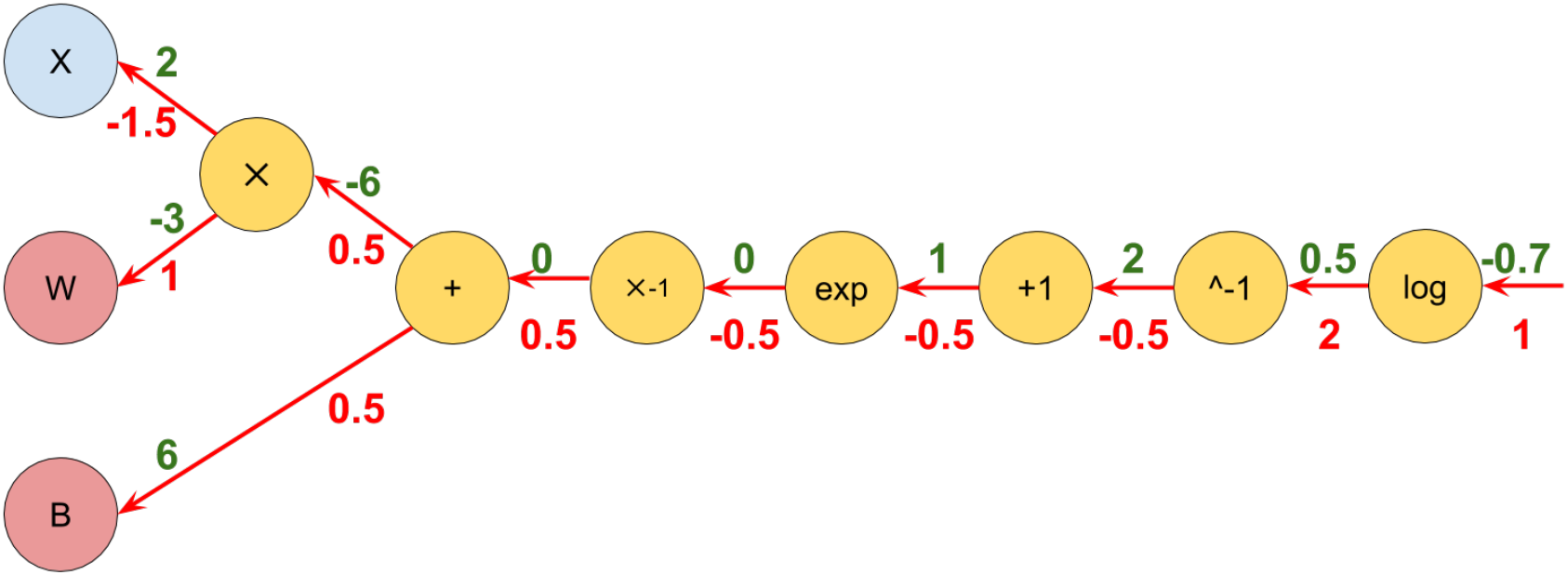
Backward



×:

- receives $\frac{\partial L}{\partial (in_1 \times in_2)} = 0.5$
- computes $(\frac{\partial (in_1 \times in_2)}{\partial in_1}, \frac{\partial (in_1 \times in_2)}{\partial in_2} = (in_2, in_1) = (-3, 2)$
- returns $(\frac{\partial L}{\partial in_1}, \frac{\partial L}{\partial in_2}) = 0.5 * (-3, 2) = (-1.5, 1)$

Backward



$$\nabla L = \left[\frac{\partial L}{\partial x}, \frac{\partial L}{\partial w}, \frac{\partial L}{\partial b}\right] = [-1.5, 1, 0.5]$$

Parameter updating

Bitdefender

- Measure data fitting performance: loss function $L(D, \theta)$,

[1] dsdeepdive.blogspot.com

Parameter updating

Bitdefender

- Measure data fitting performance: loss function $L(D, \theta)$,
- Compute the gradient of the loss function with respect to weights,

[1] dsdeepdive.blogspot.com

Parameter updating

- Measure data fitting performance: loss function $L(D, \boldsymbol{\theta})$,
- Compute the gradient of the loss function with respect to weights,
- Find incremental solutions that better explain the data.

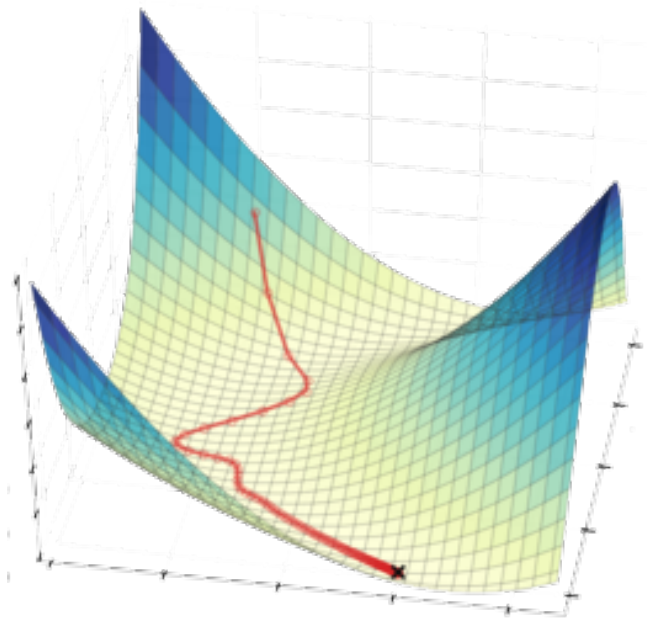
using:

$$\boldsymbol{\theta}_{j+1} \leftarrow \boldsymbol{\theta}_j - \alpha \nabla_{\boldsymbol{\theta}_j} \mathcal{L}$$

sequentially find:

$$\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_k$$

such that $L(D, \boldsymbol{\theta})$decreases.



Sequential minimizing $L(D, \theta)$ [1]

[1] dsdeepdive.blogspot.com

Outline

Universal Approximation Theorem

Bitdefender

A fully connected network with one hidden layer followed by a non-linear function can approximate any function from one finite-dimensional space to another in the limit of the **width** of the hidden layer (Hornik et al., 1990).

Universal Approximation Theorem

Bitdefender

A fully connected network with one hidden layer followed by a non-linear function can approximate any function from one finite-dimensional space to another in the limit of the **width** of the hidden layer (Hornik et al., 1990).

Potentially the hidden layer can get exponentially large. Eg.: no of possible binary functions on vectors $\boldsymbol{v} \in \{0,1\}^n$ is 2^{2^n}, requiring $O(2^n)$ units, each responding to a single input.

UAT and depth

Bitdefender

- How does depth relate to UAT?

UAT and depth

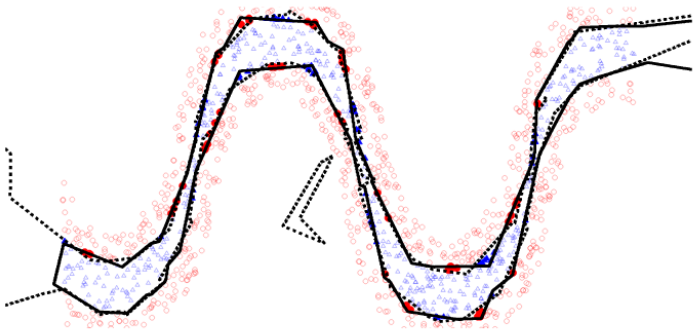
Bitdefender

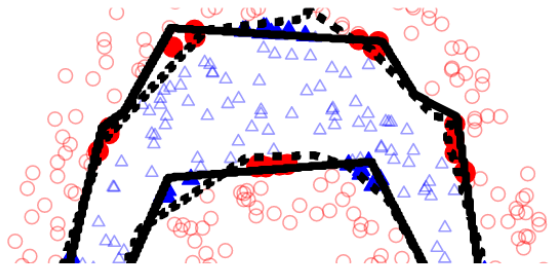
- How does depth relate to UAT?
- There exist families of functions which can be approximated efficiently (in numbers of parameters) by **deep** networks.

UAT and depth

Bitdefender

- How does depth relate to UAT?
- There exist families of functions which can be approximated efficiently (in numbers of parameters) by **deep** networks.
- Montufar et al. 2014:



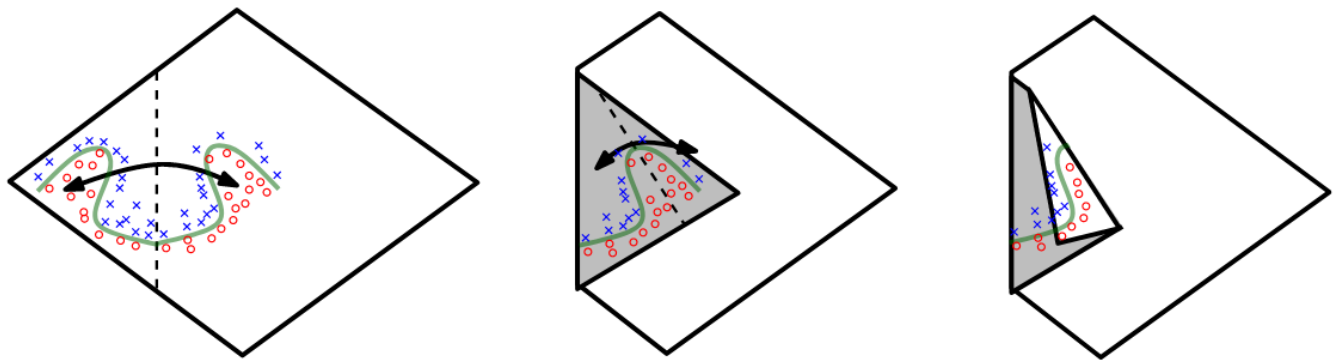


A network with ReLU units learns piece-wise linear functions at each layer and deep networks reuse these computations exponentially more often than shallow networks. Solid line is a 20 units, one layer NN and dotted line is a 2 layers, 10 units each NN.

UAT and depth

Bitdefender

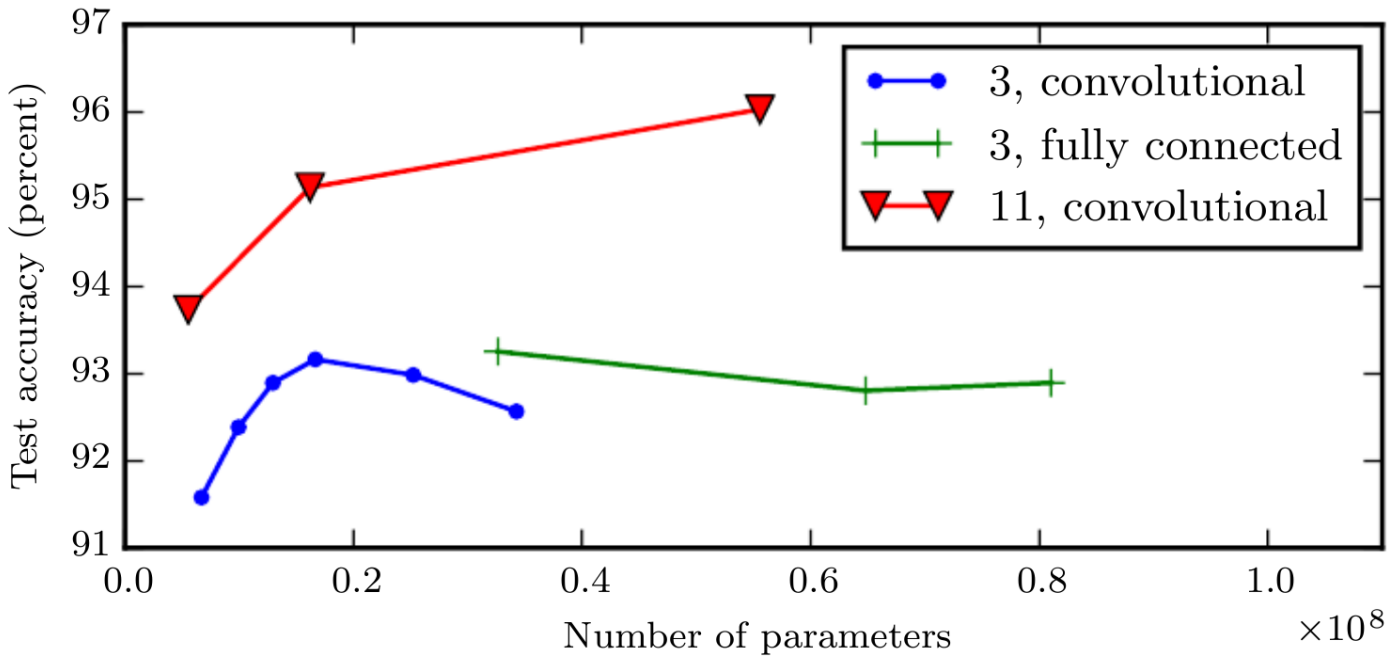
- How does depth relate to UAT?
- There exist families of functions which can be approximated efficiently (in numbers of parameters) by **deep** networks.
- Montufar et al. 2014:



A network with ReLU units creates mirror images of the function computed at the input of some hidden unit resulting in increasingly expressive networks

UAT and depth

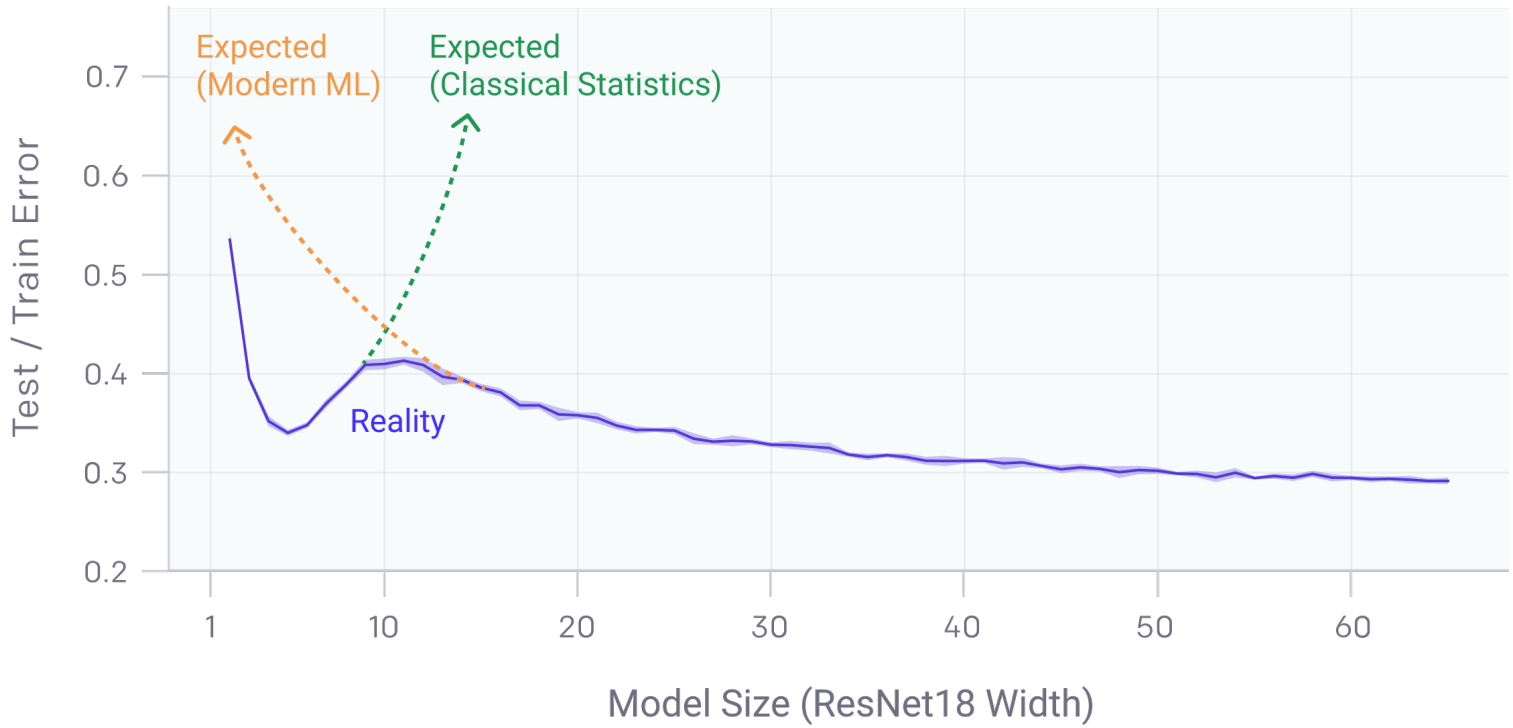
- How does depth relate to UAT?
- There exist families of functions which can be approximated efficiently (in numbers of parameters) by **deep** networks.
- Goodfellow et al. 2014:



Generalization correlates with depth, not number of parameters.

The new generalization theory

Bitdefender



Is the classic generalization theory still valid?