Computer Vision and Deep Learning

Lecture 3

Last time – linear classification

Score function: to map inputs (images) to class scores

Loss function: to evaluate a classifier

- Hinge loss
- Softmax loss

Optimization: gradient descent

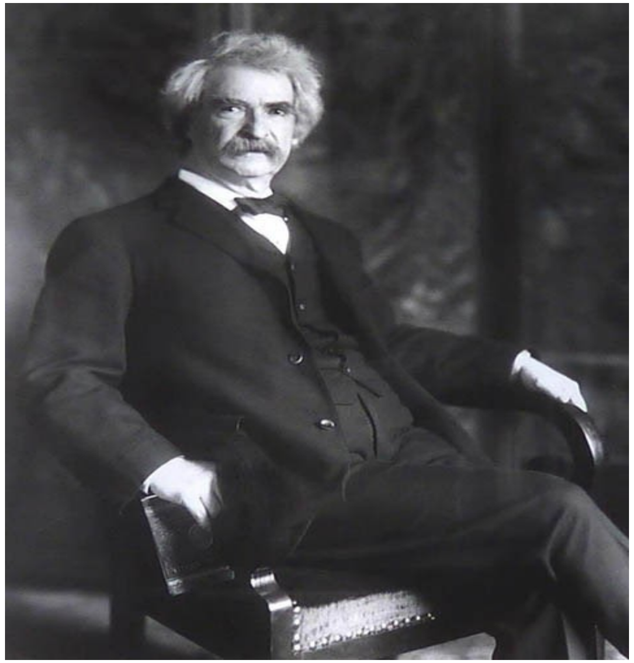
- Numerical gradient
- Analytical gradient

Today’s agenda

- Evaluating a model’s performance
- Computational graphs
- Backpropagation
- Artificial neural networks
  - Activation functions
- Image convolutions (one step forward towards convolutional neural networks)

Evaluating a model's performance

"*Facts are stubborn things, but statistics are more pliable.* " – Mark Twain
"*There are three kinds of lies: lies, damned lies, and statistics.* " – popularized by Mark Twain



Mark Twain

Evaluation metrics

- True positives (TP)
- False positives (FP)
- True negatives (TN)
- False negatives (FN)
- Accuracy: The percent of predictions that the model got right

Evaluation metrics

- True positives (TP)
- False positives (FP)
- True negatives (TN)
- False negatives (FN)
- Accuracy: The percent of predictions that the model got right
  - **Accuracy will yield misleading results if the data set is unbalanced!!!**

Tumour classification

True Positive (TP): • Reality: Malignant • ML model predicted: Malignant • **Number of TP results: 1**	**False Positive (FP):** • Reality: Benign • ML model predicted: Malignant • **Number of FP results: 1**
False Negative (FN): • Reality: Malignant • ML model predicted: Benign • **Number of FN results: 8**	**True Negative (TN):** • Reality: Benign • ML model predicted: Benign • **Number of TN results: 90**

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} = \frac{1+90}{1+90+1+8} = 0.91$$

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{1}{1+1} = 0.5$$

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{1}{1+8} = 0.11$$

Confusion matrix

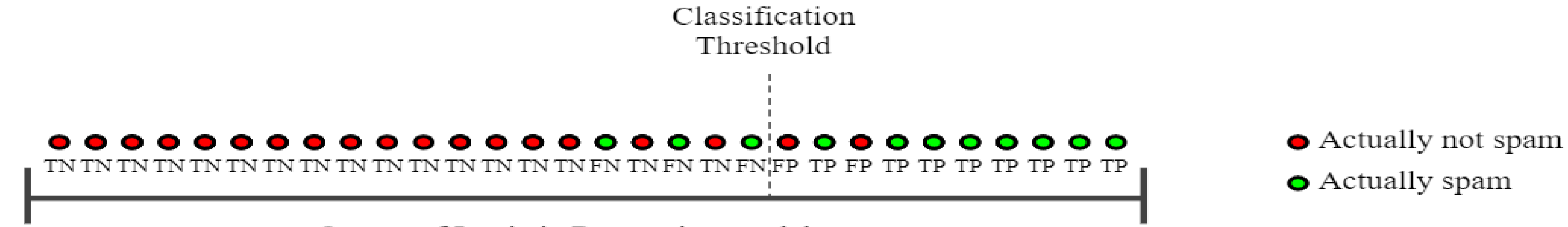
		True condition				
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$	
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection, Power = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$	F_1 score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$		

Image source: https://en.wikipedia.org/wiki/Confusion_matrix

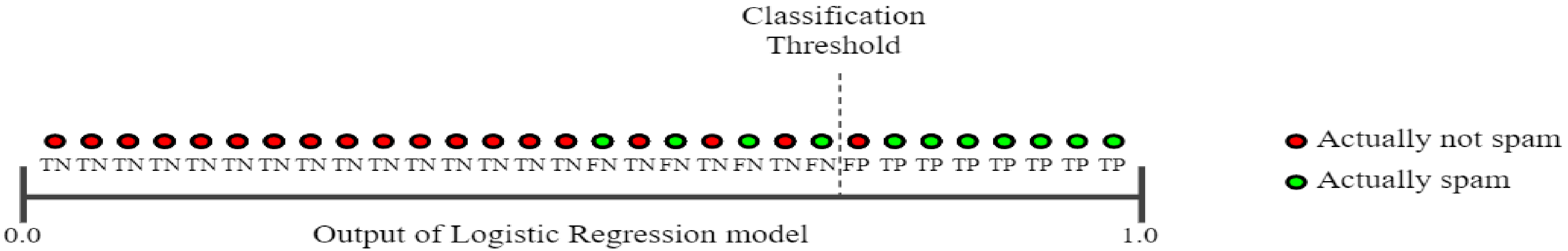
Precision, Recall, F1 score

- Accuracy
  - The percent of prediction that the model got right
- Precisi**on**
  - The ability of a model to identity **only** relevant instances
- Rec**all**
  - The ability of a model to identify **all** relevant instances
- F1 score
  - Harmonic mean between precision and recall

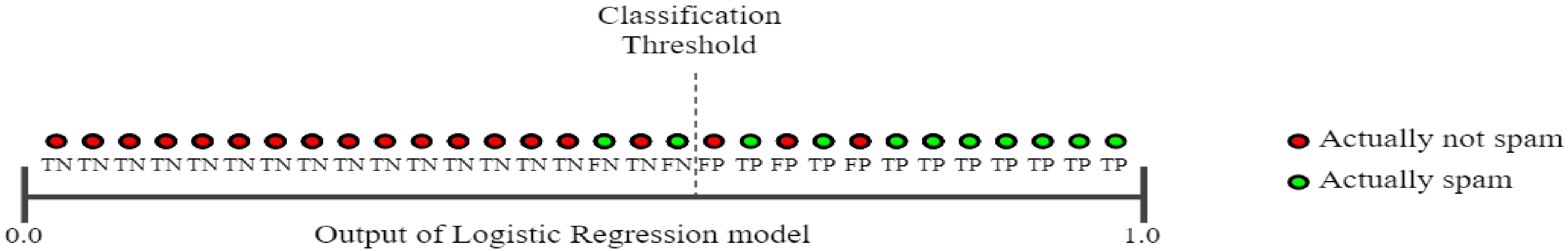
Precision vs. recall



Precision: 0.8
Recall: 0.73



Precision: 0.88
Recall: 0.64



Precision: 0.75
Recall: 0.82

https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall

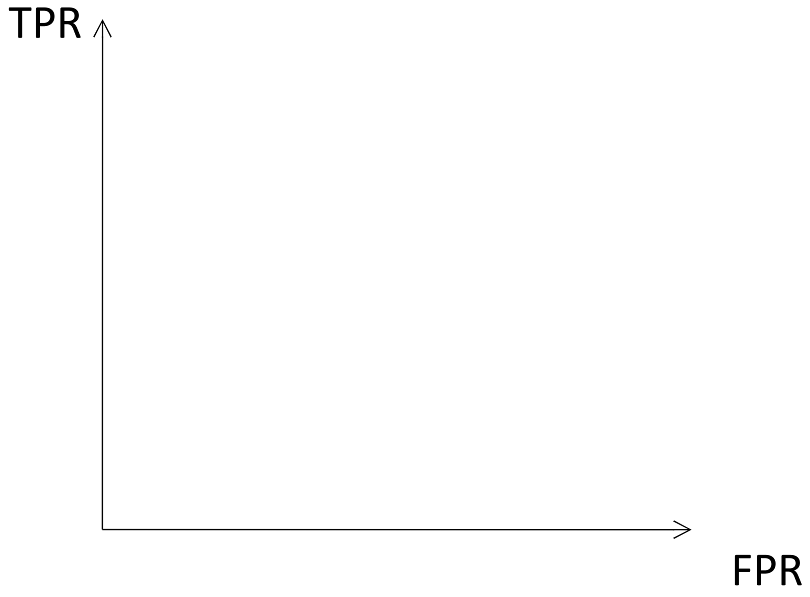
Precision vs. recall

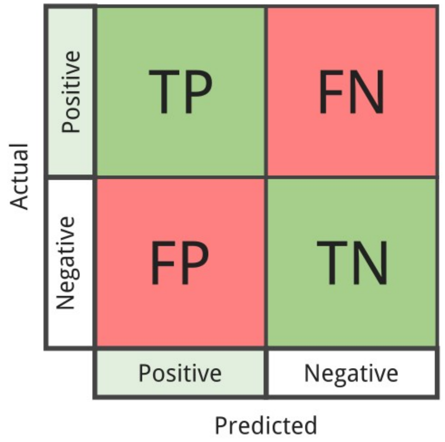
- *Precision*: spam detection, predict when to launch a satellite, pregnancy tests etc.
- *Recall*: airport security (make sure that every suspicions event is investigated), cancer prediction, detecting credit card frauds etc.

Receiver Operating characteristic curve
ROC curve

- Shows the performance of a classifier at different classification thresholds

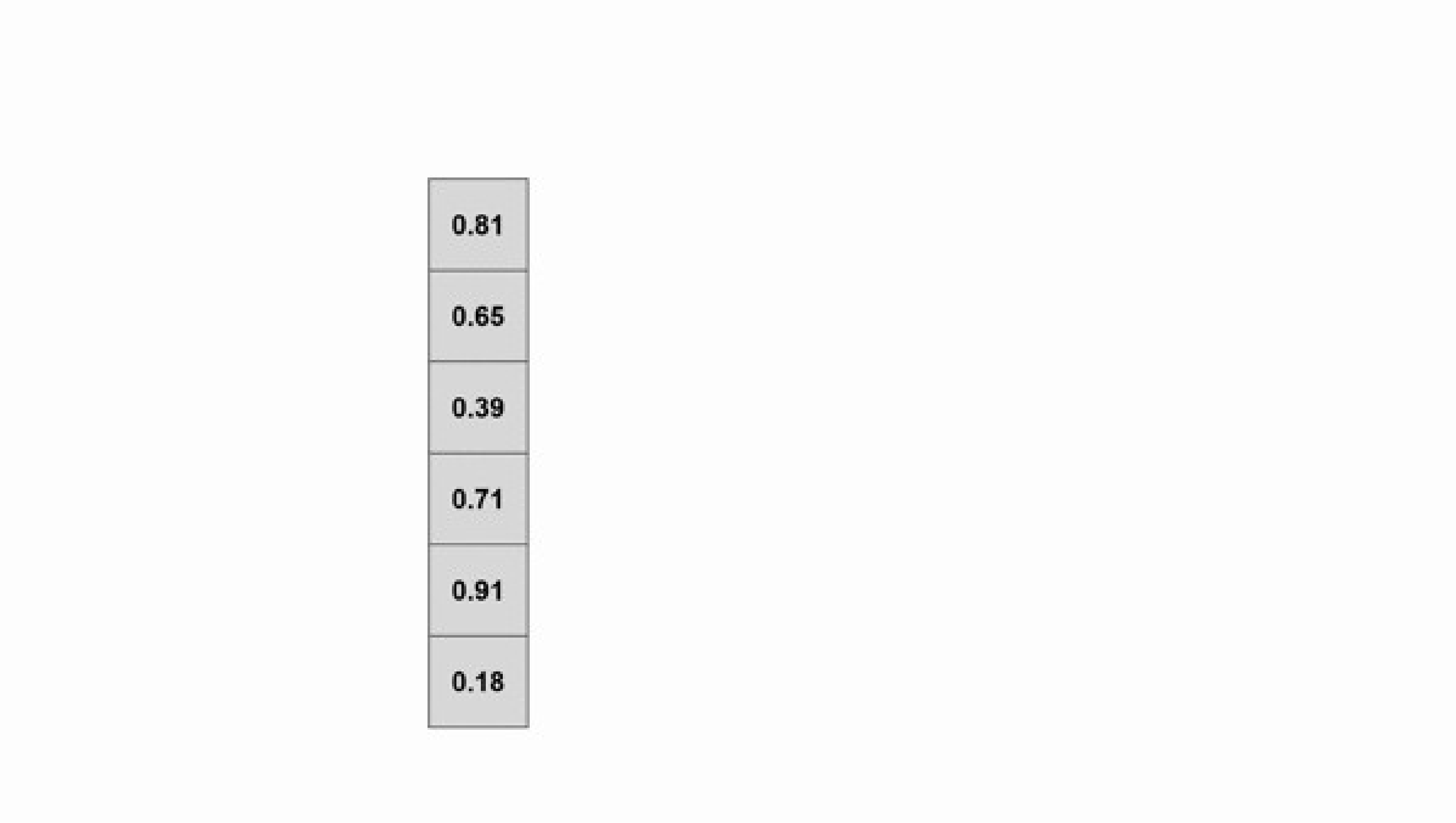
False Positive Rate (1- specificity) – x axis
True Positive Rate (sensitivity) – y axis





Receiver Operating characteristic curve

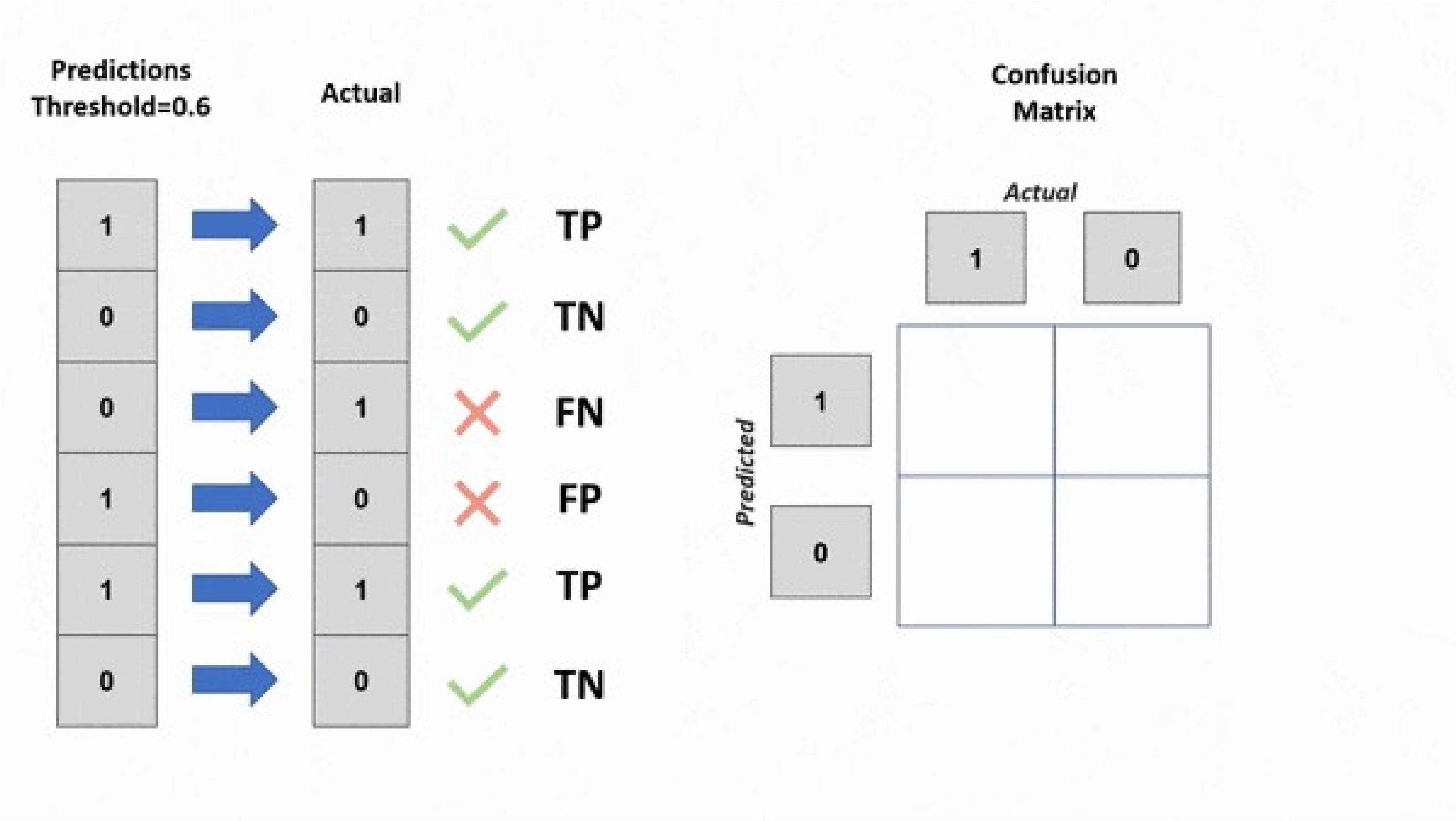
ROC curve



https://towardsdatascience.com/roc-curve-and-auc-from-scratch-in-numpy-visualized-2612bb9459ab

Receiver Operating characteristic curve

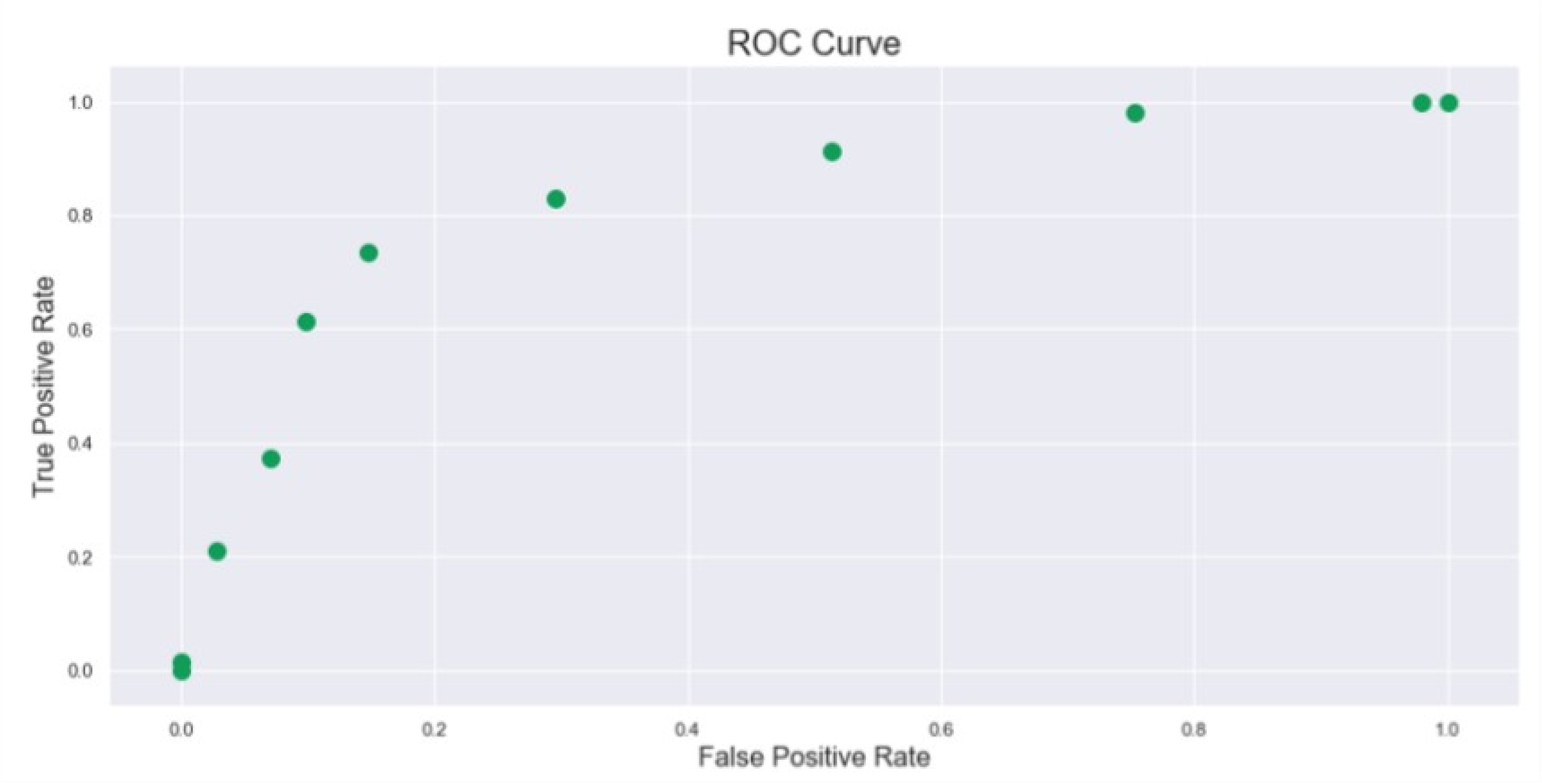
ROC curve



https://towardsdatascience.com/roc-curve-and-auc-from-scratch-in-numpy-visualized-2612bb9459ab

Receiver Operating characteristic curve

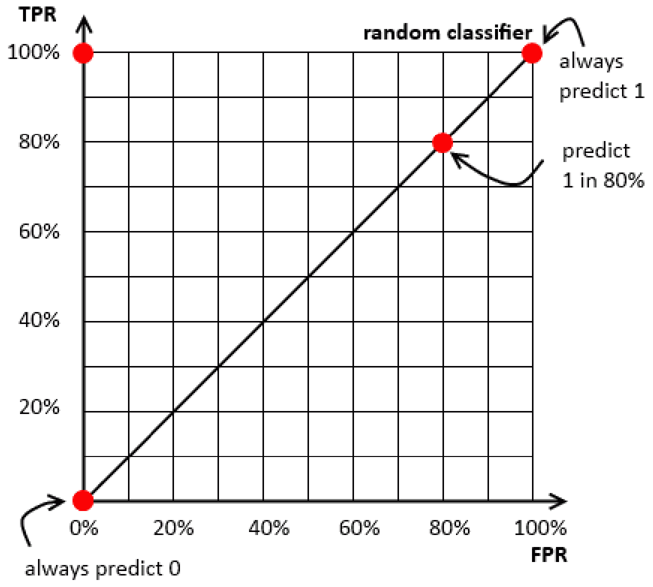
ROC curve



https://towardsdatascience.com/roc-curve-and-auc-from-scratch-in-numpy-visualized-2612bb9459ab

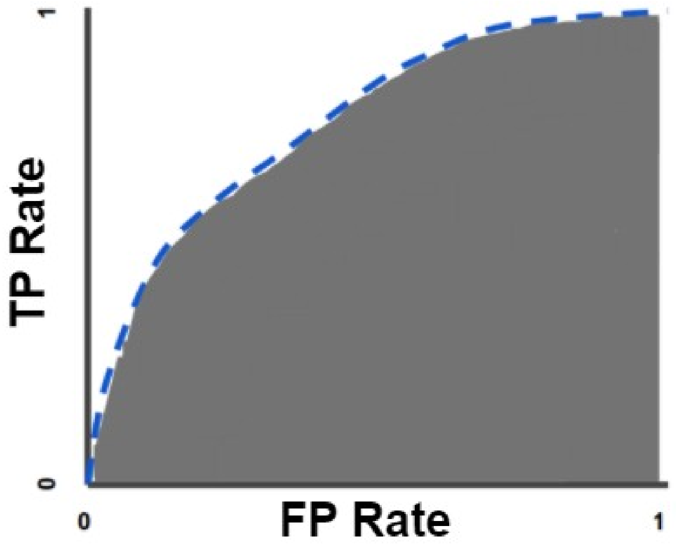
Receiver Operating characteristic curve
ROC curve

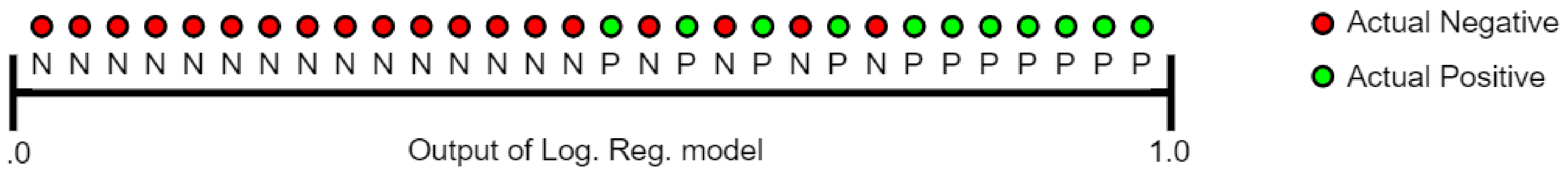
- Shows the performance of a classifier at different classification thresholds



Area under the curve
AUC

- Measures the entire two-dimensional area underneath the entire ROC curve
- Makes it easier to compare one ROC curve to another one



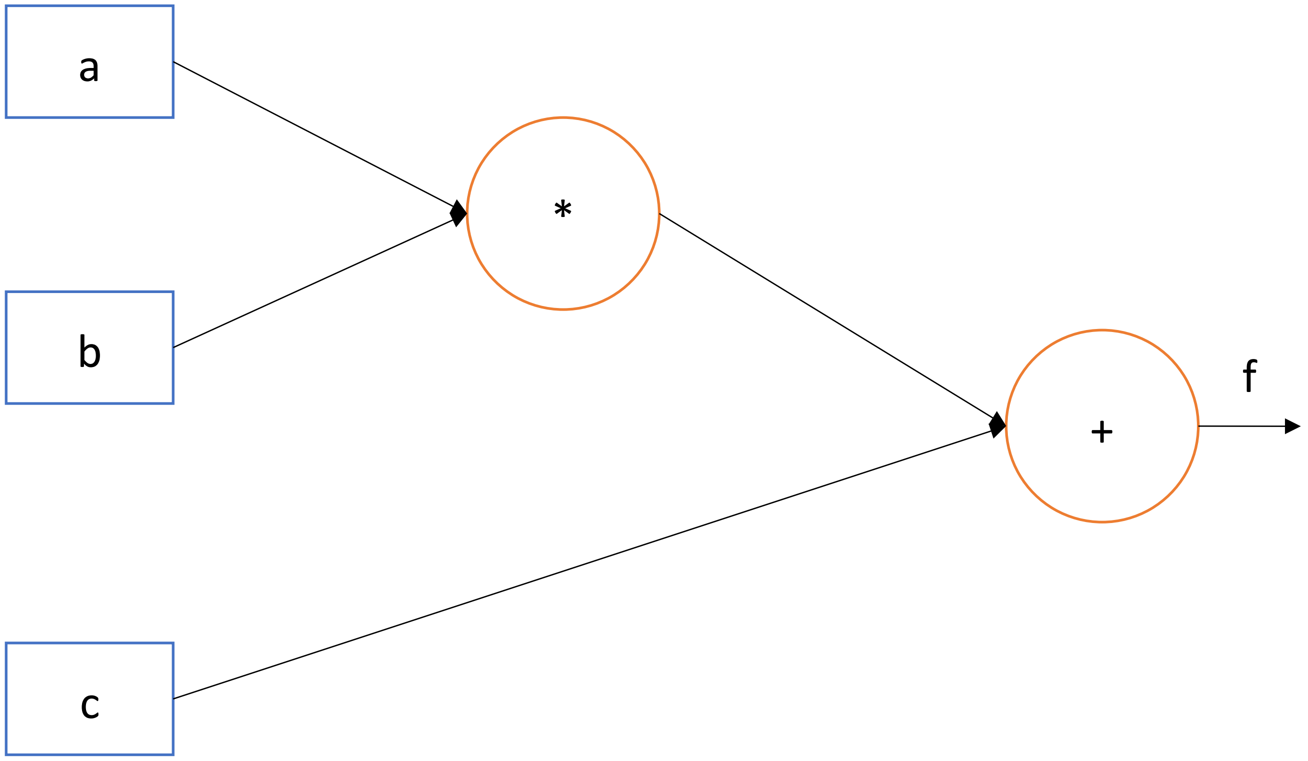


How do we compute the gradients?

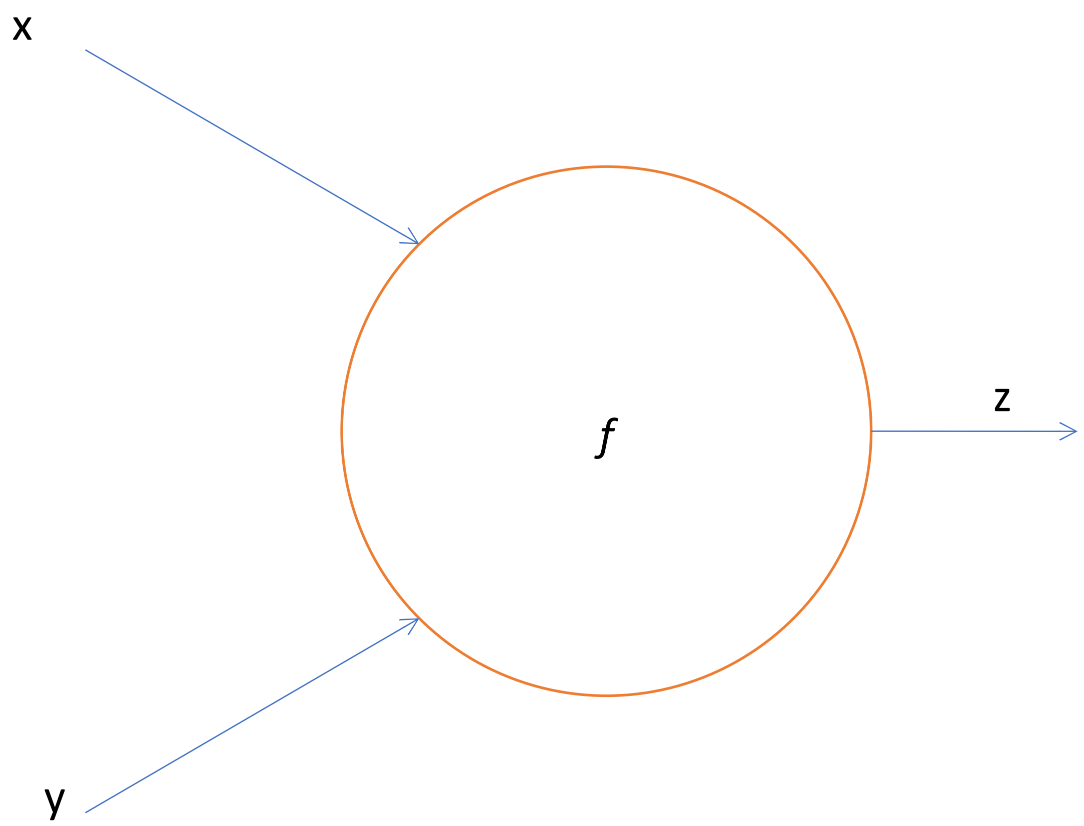
- Numerical gradient
- Analytical gradient
- Better idea: use computational graphs and back-propagation

Computational graphs

$f(a, b, c) = a*b + c$



Back propagation

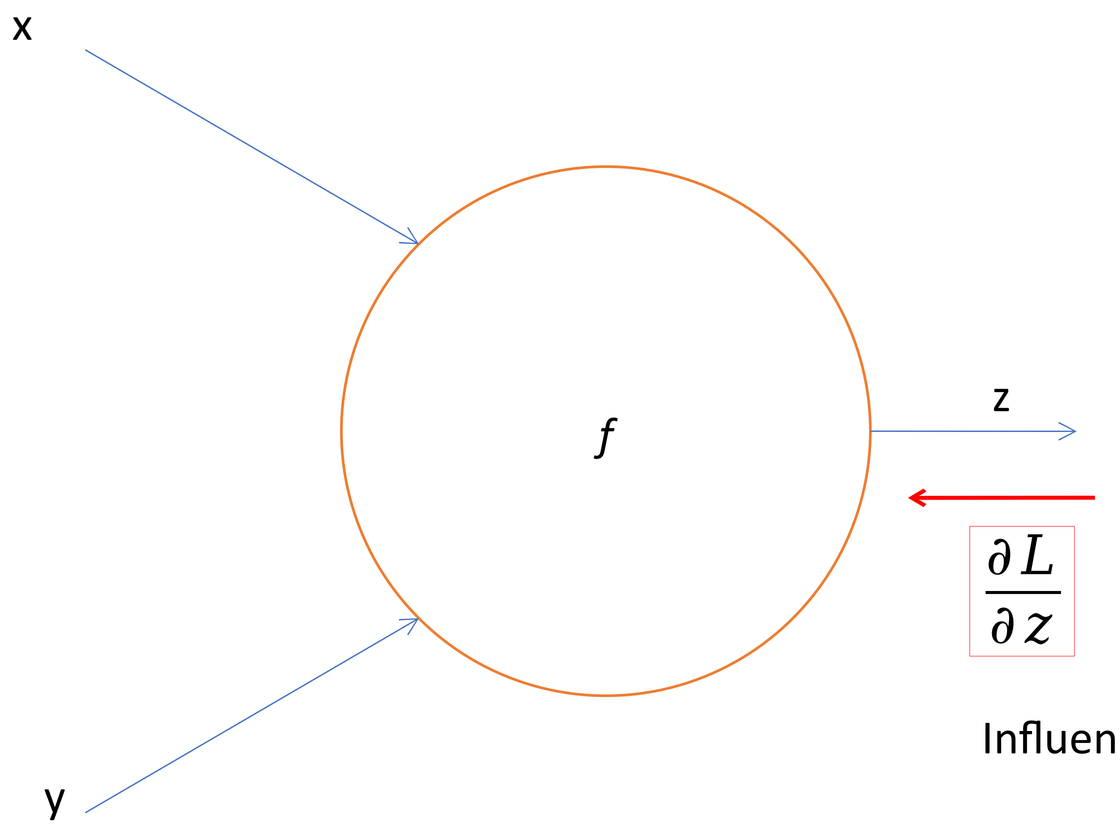


Local gradients

- Influence of x and y on the nodes output z

Back propagation

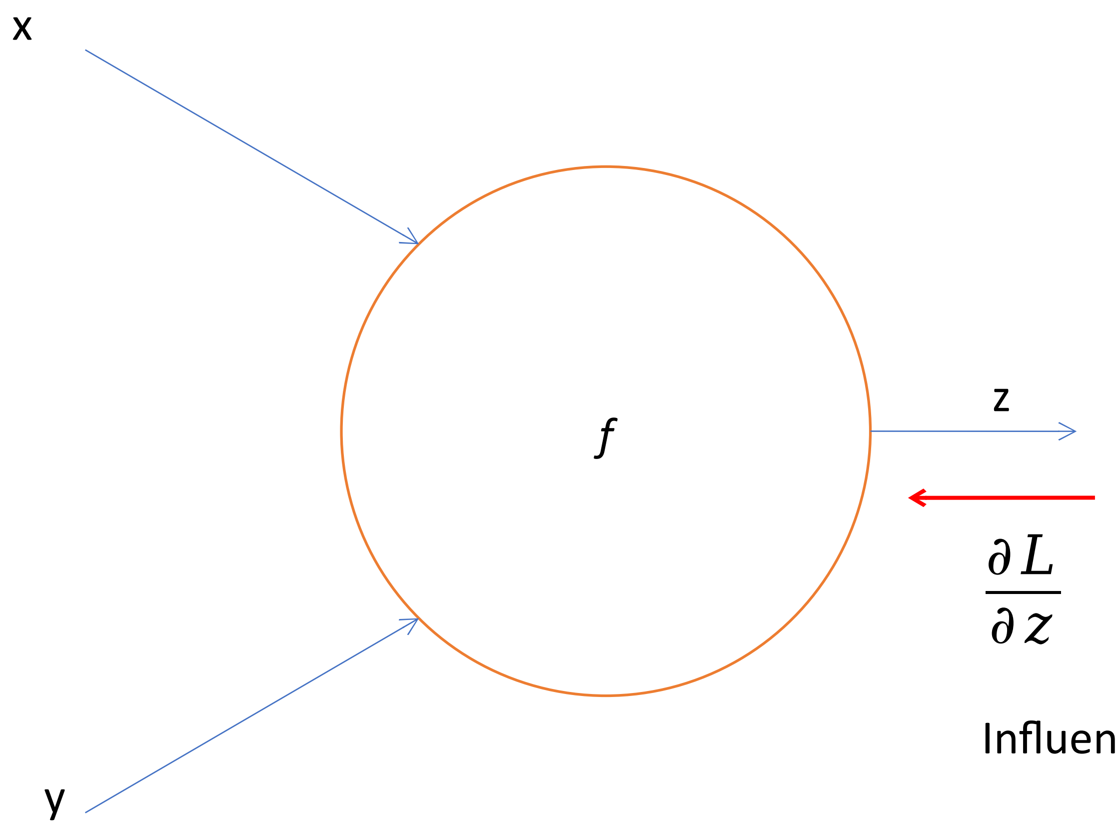
Chain rule



$$\boxed{\frac{\partial L}{\partial z}}$$

Influence of z to the final output of the graph (loss)

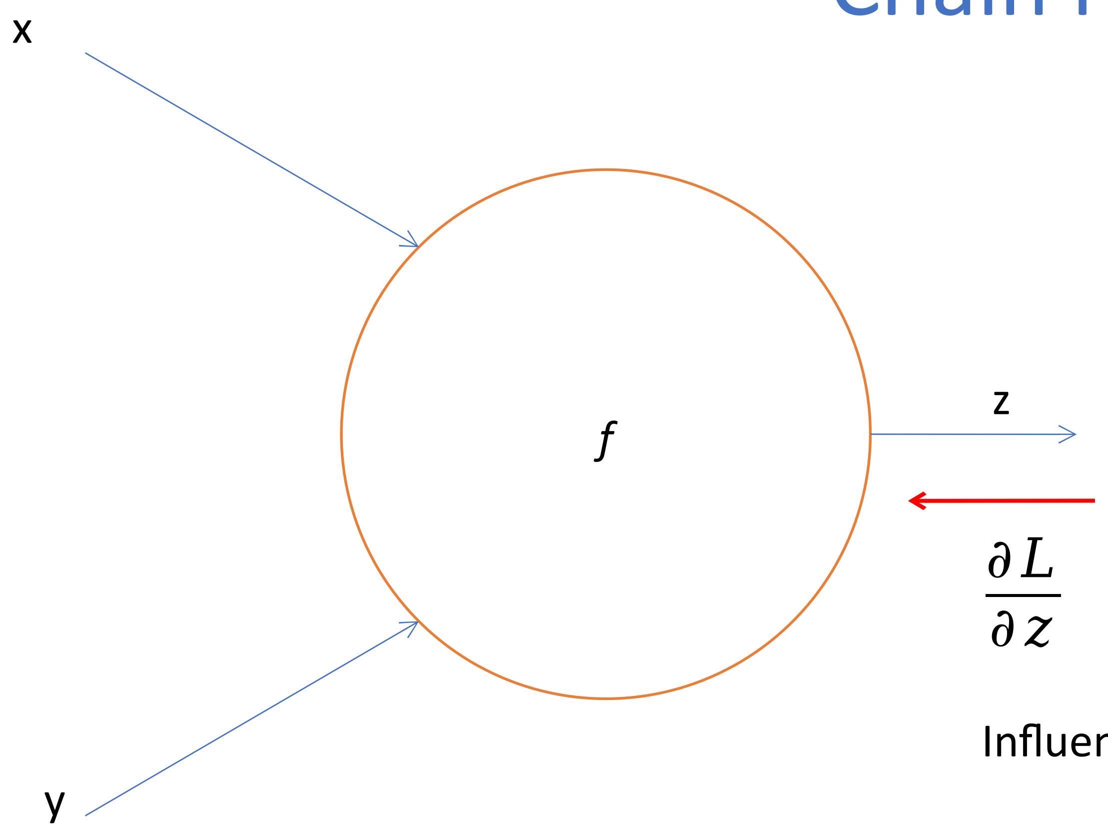
Back propagation



Influence of z to the final output of the graph (loss)

Back propagation

Chain rule

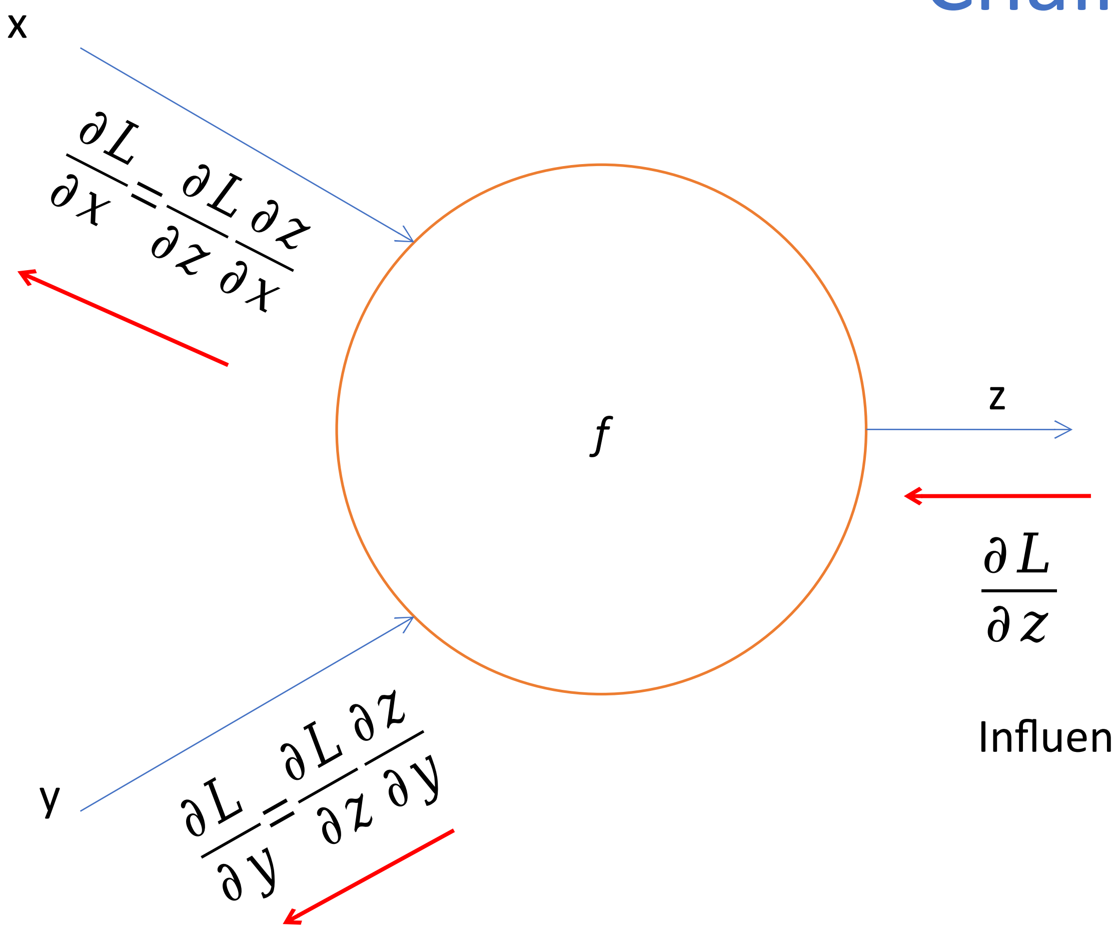


$$\frac{\partial L}{\partial z}$$

Influence of z to the final output of the graph (loss)

Back propagation

Chain rule



Influence of z to the final output of the graph (loss)

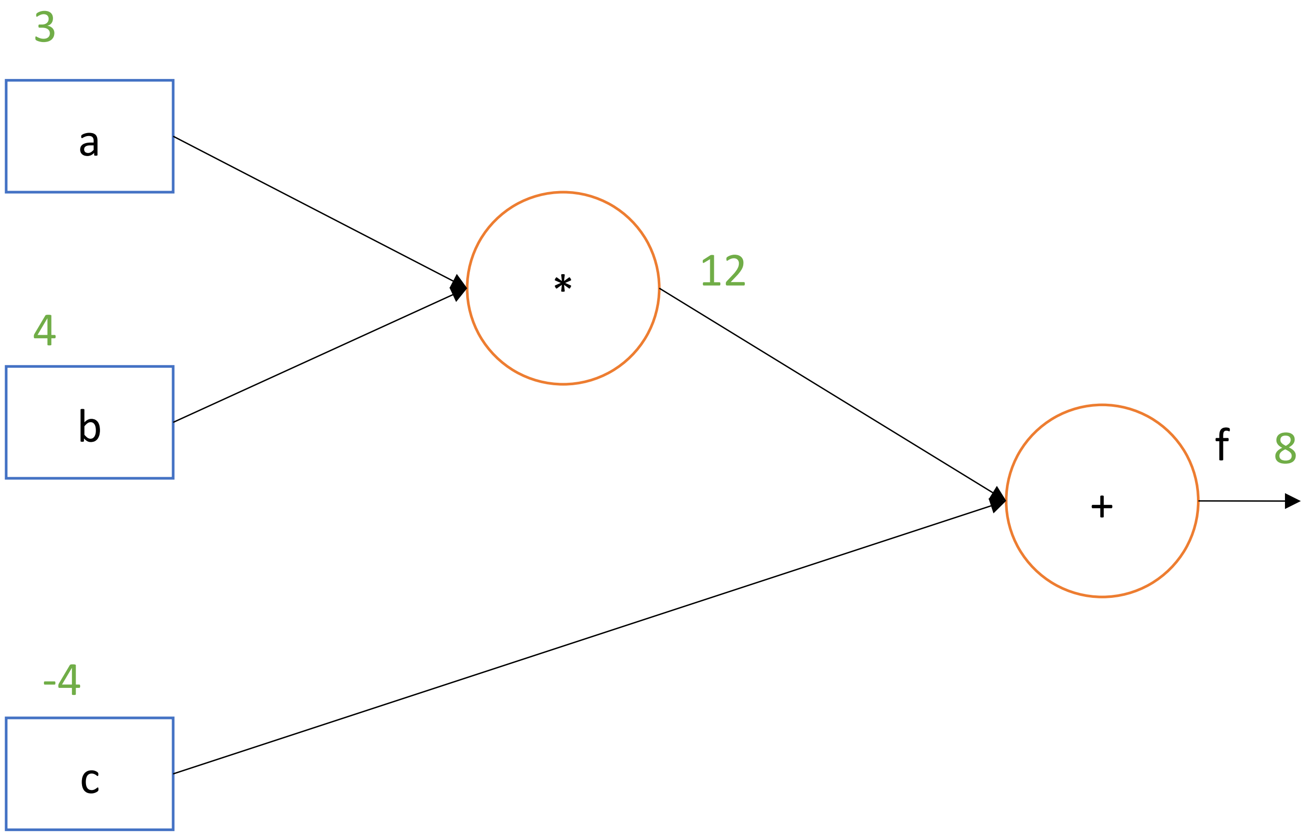
Back propagation

Use a recursive application of the chain rule on every node in the graph to computer influence of all the intermediate nodes on the output of the graph

Computational graphs – forward pass

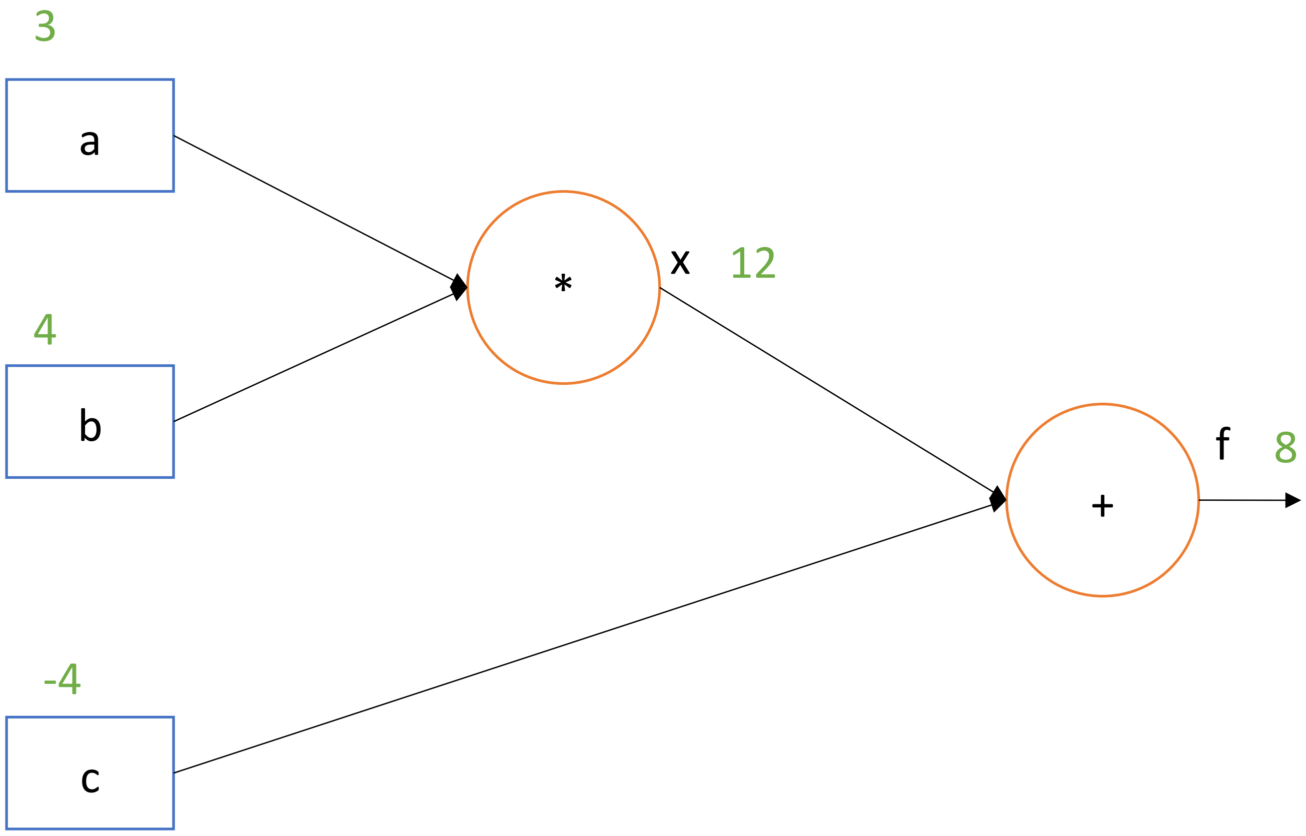
f(a, b, c) = a*b + c

a = 3
b = 2
c = -4



Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$f(a, b, c) = a*b + c$

x = a*b

= b

= a

f = x + c

= 1

= 1

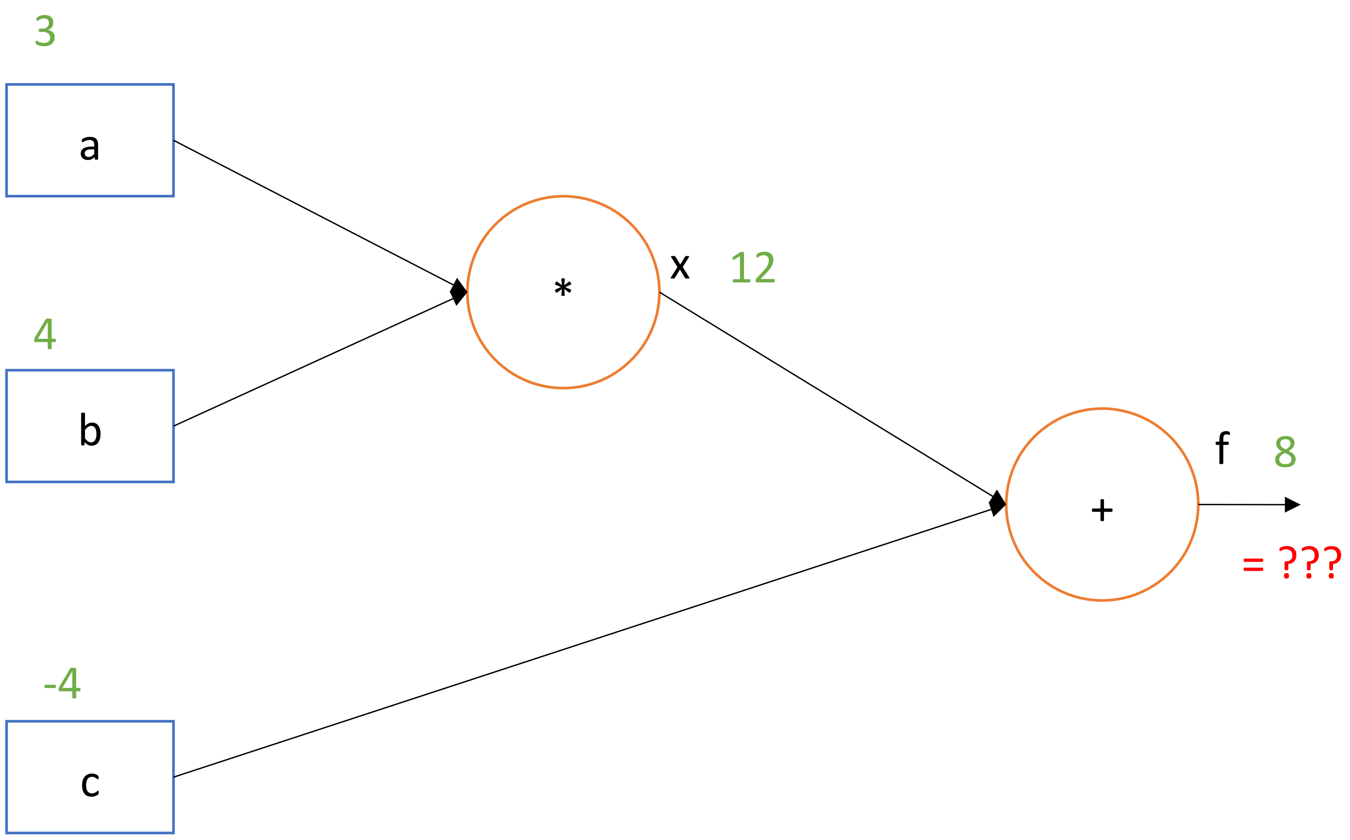
= ???

= ???

= ???

Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$f(a, b, c) = a*b + c$

= b

x = a*b

= a

= 1

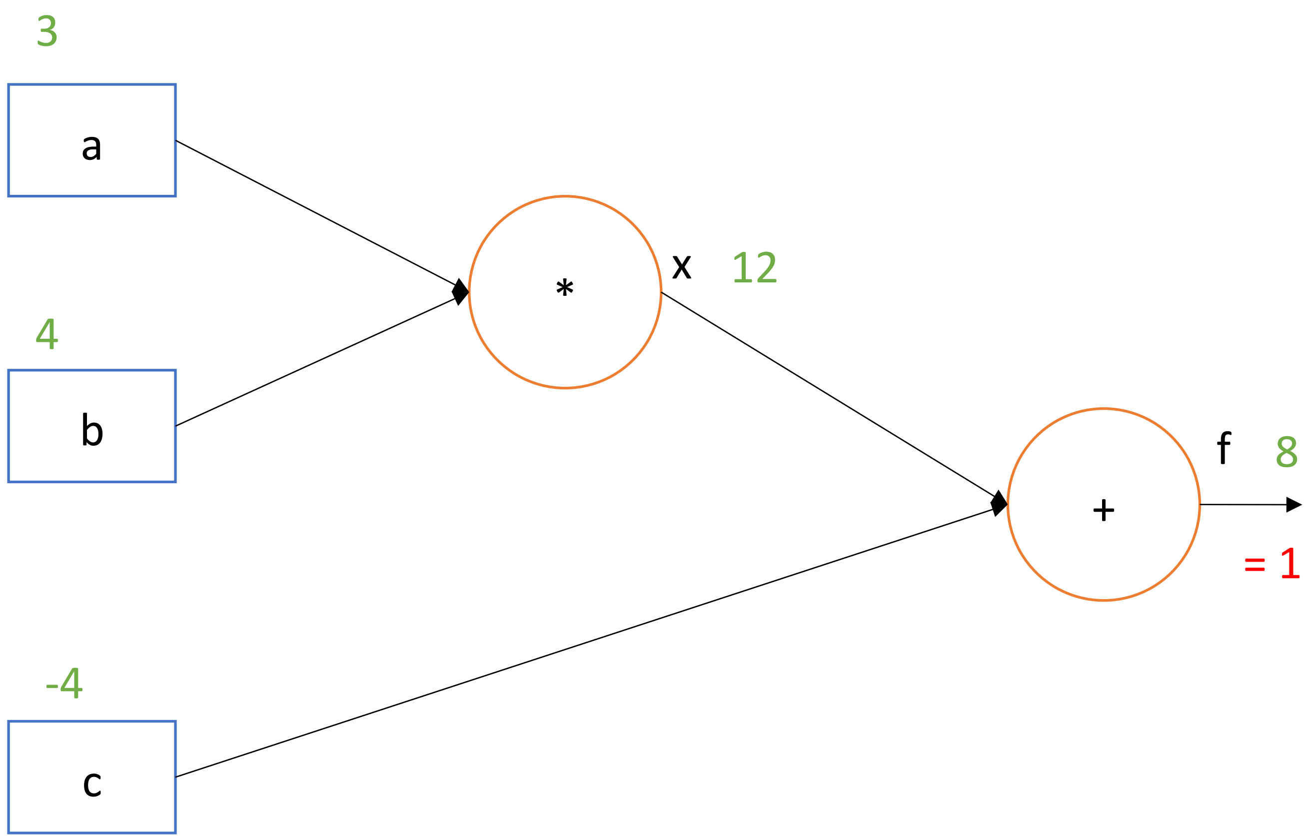
f = x + c

= 1

= ??? = ??? = ???

Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$f(a, b, c) = a*b + c$

x = a*b

= b

= a

f = x + c

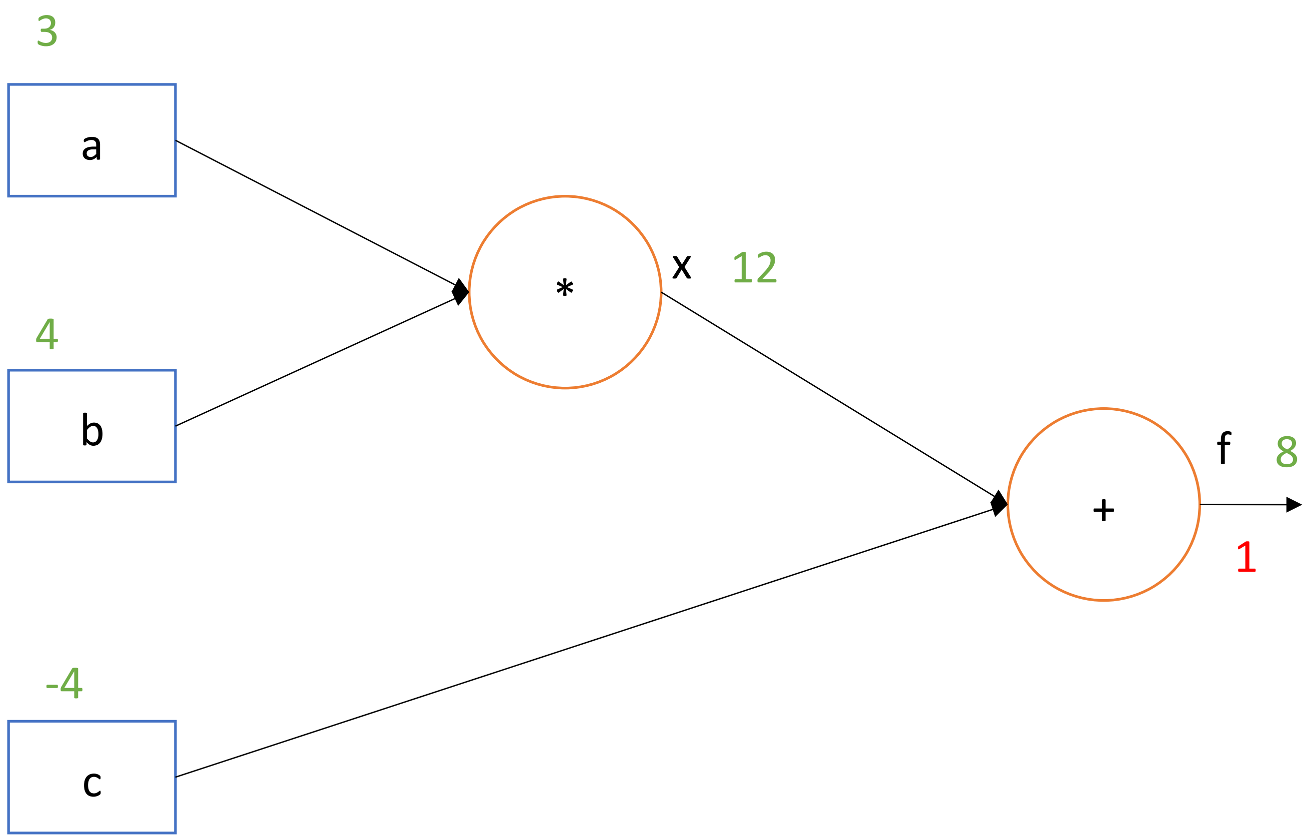
= 1

= 1

= ???   = ???   = ???

Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$$f(a, b, c) = a*b + c$$

$x = a*b$

= b

= a

$f = x + c$

= 1

= 1

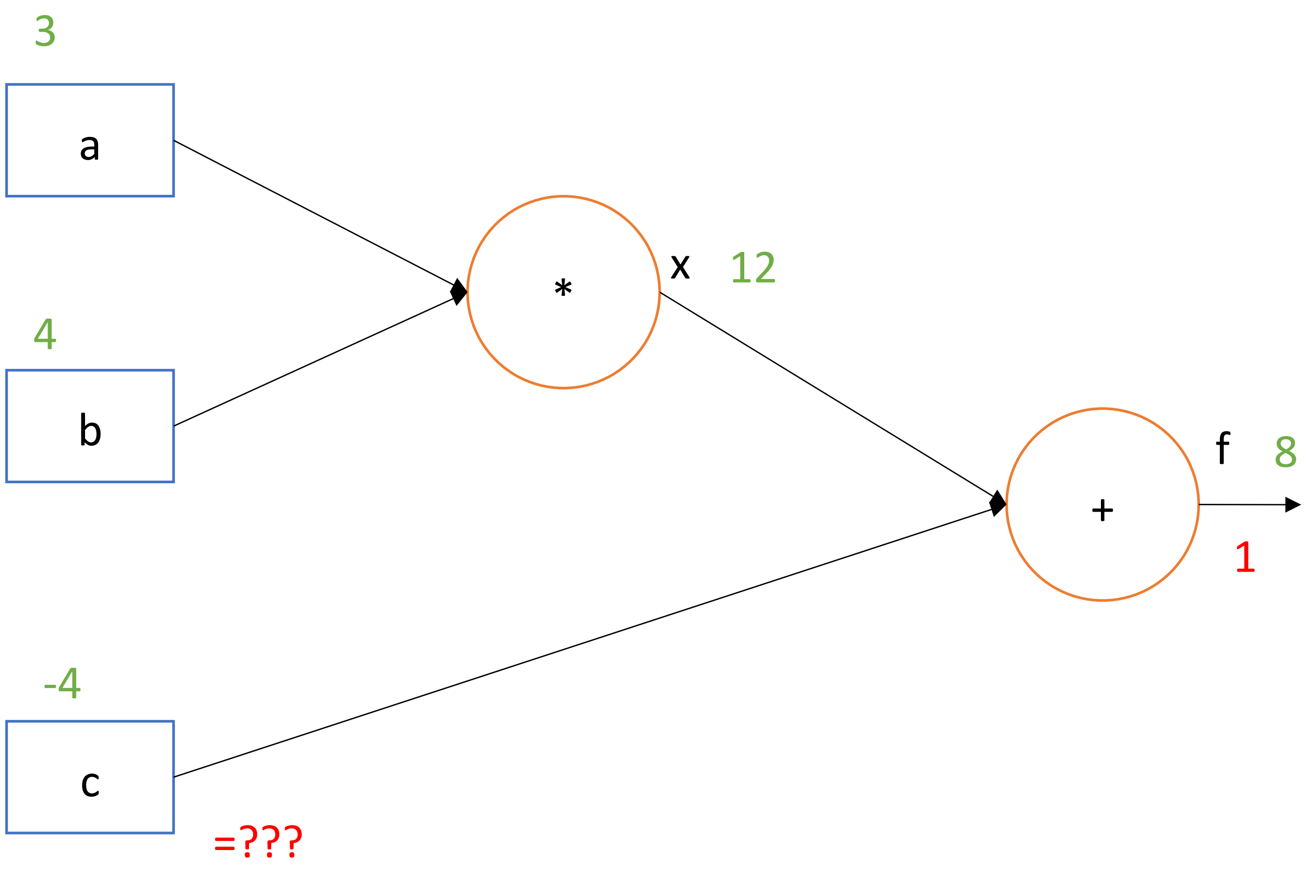
= ???

= ???

= ???

Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$$f(a, b, c) = a*b + c$$

x = a*b

= b

= a

f = x + c

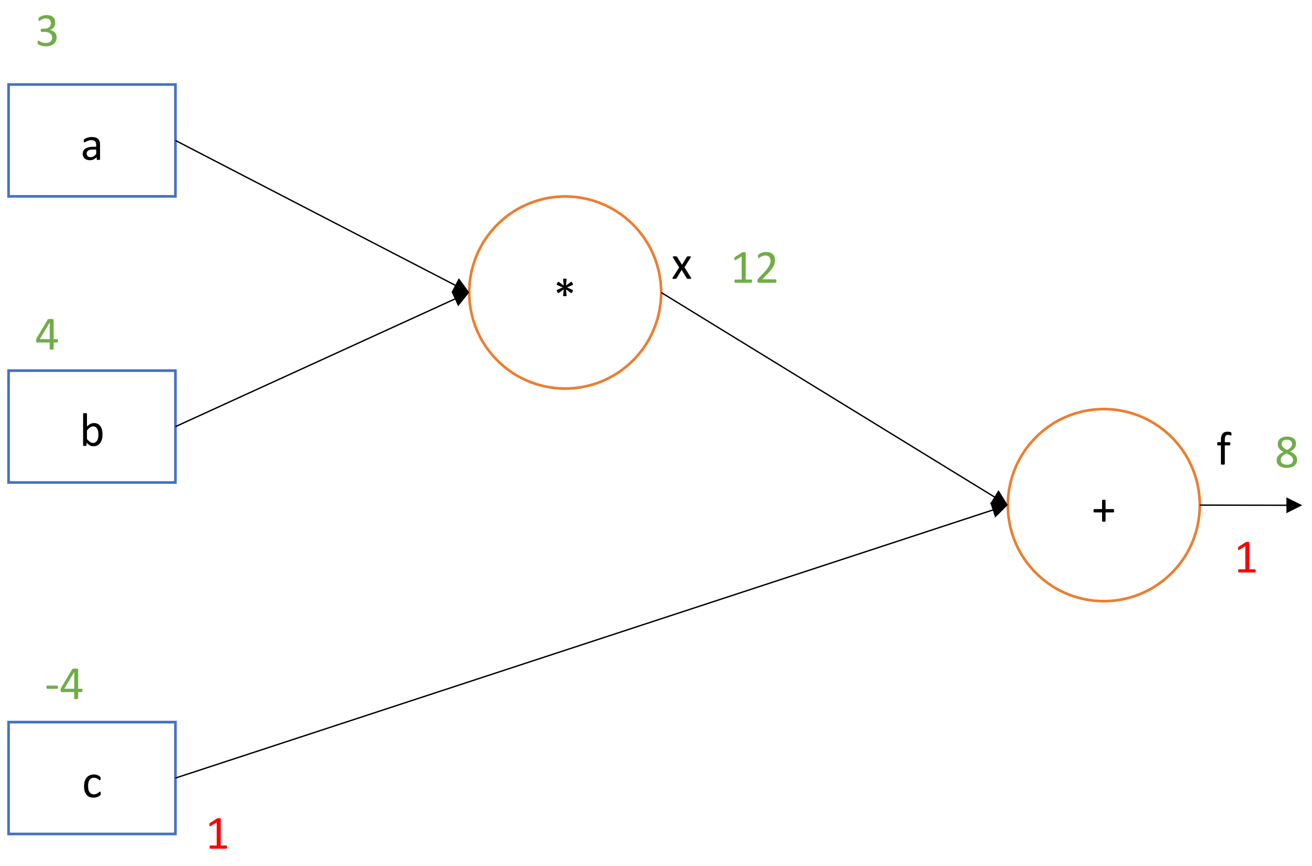
= 1

= 1

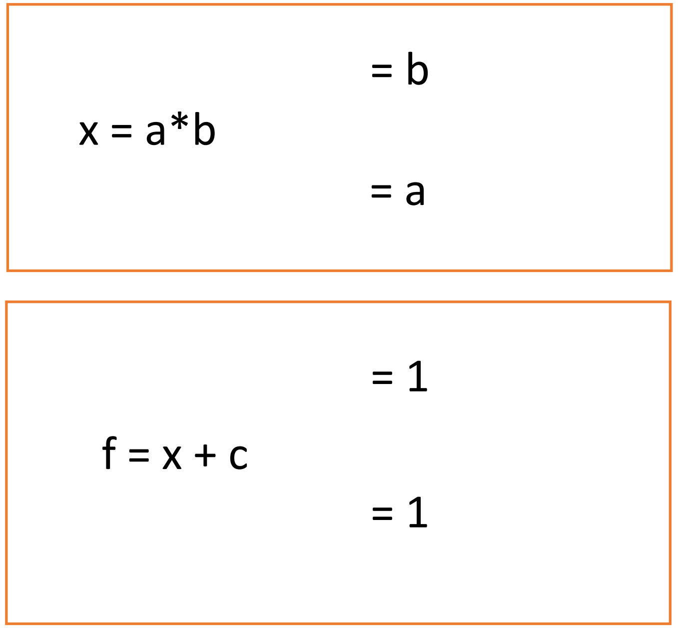
= ??? = ??? = ???

Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$$f(a, b, c) = a*b + c$$



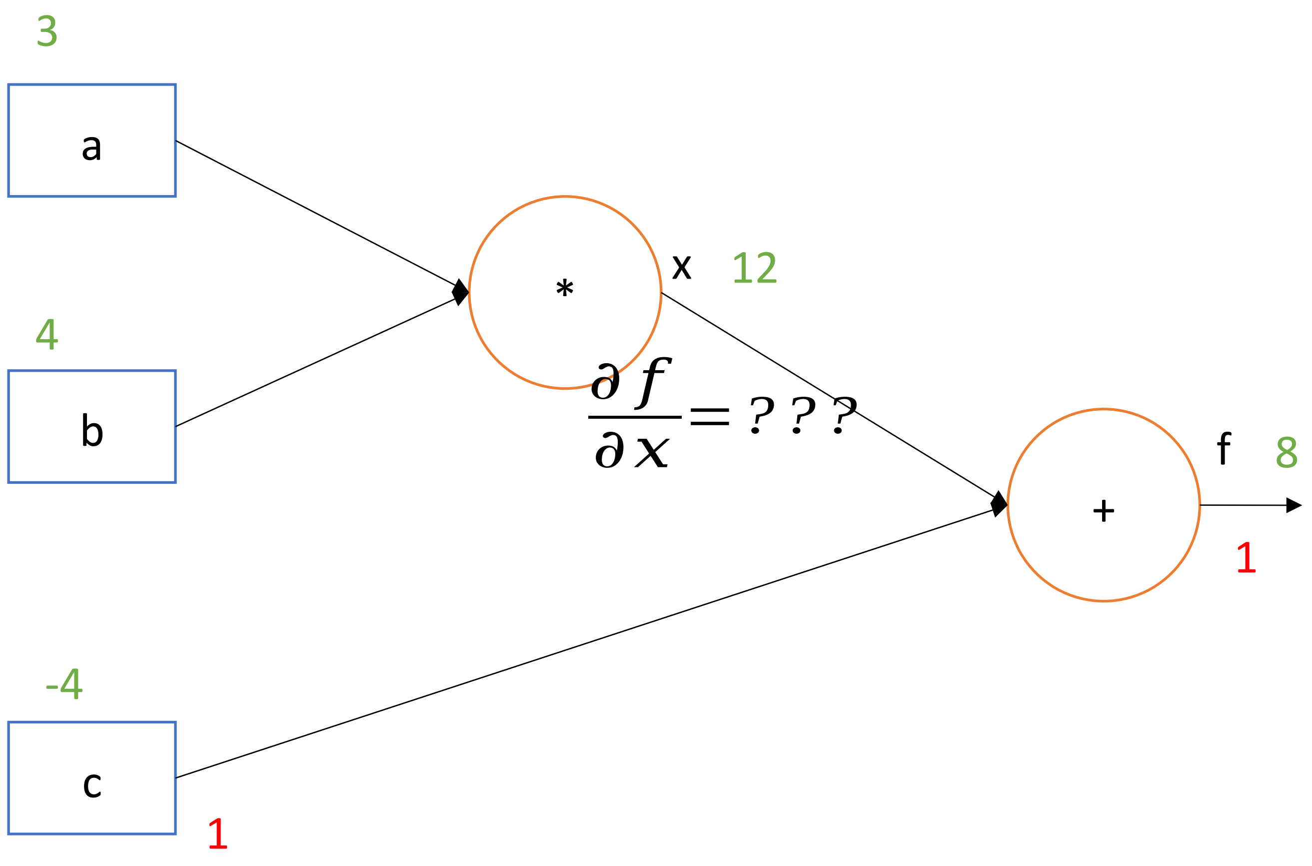
= ???

= ???

= ???

Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$f(a, b, c) = a*b + c$

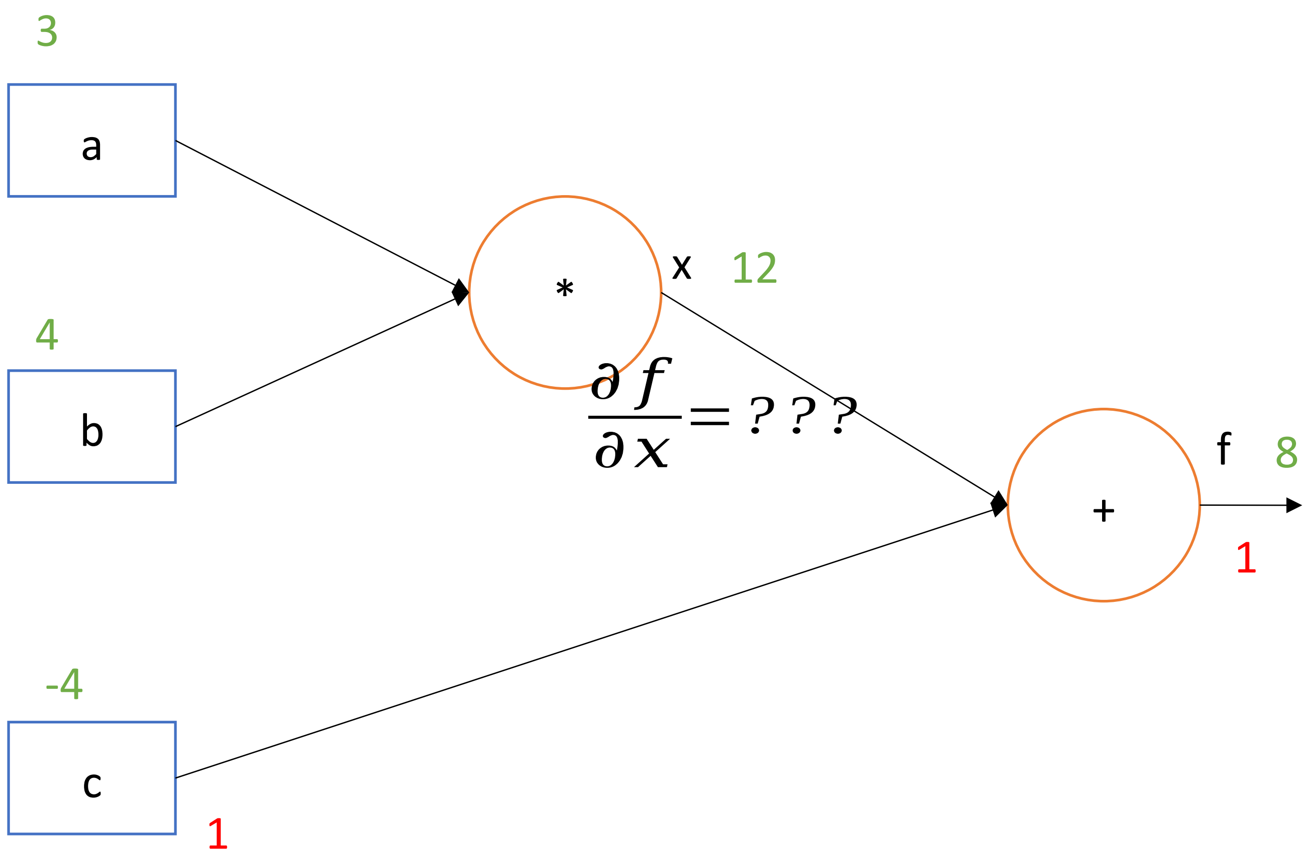
x = a*b
= b
= a

f = x + c
= 1
= 1

= ??? = ??? = ???

Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$f(a, b, c) = a*b + c$

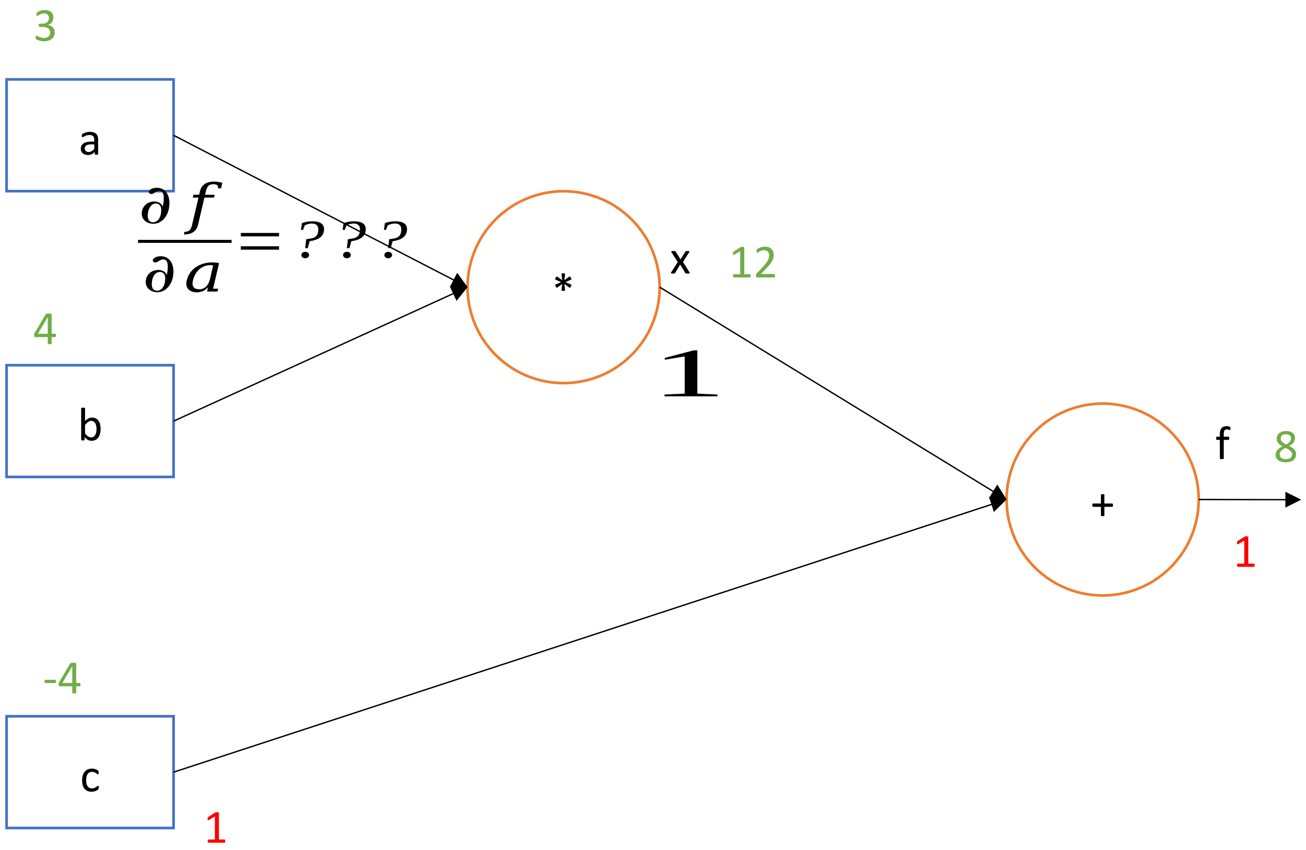
x = a*b
= b
= a

f = x + c
= 1
= 1

= ??? = ??? = ???

Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$f(a, b, c) = a*b + c$

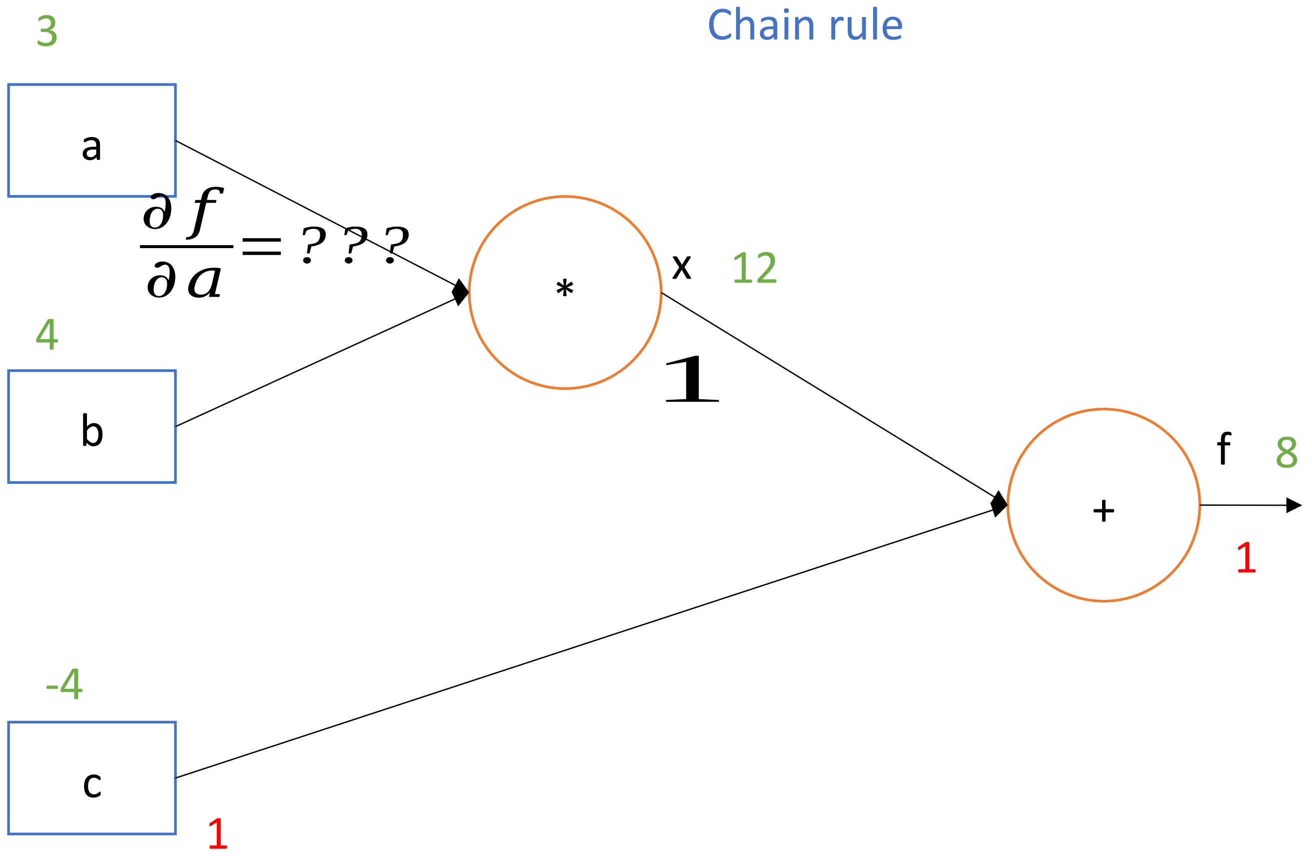
x = a*b
= b
= a

f = x + c
= 1
= 1

= ??? = ??? = ???

Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$$f(a, b, c) = a*b + c$$

x = a*b

= b

= a

f = x + c

= 1

= 1

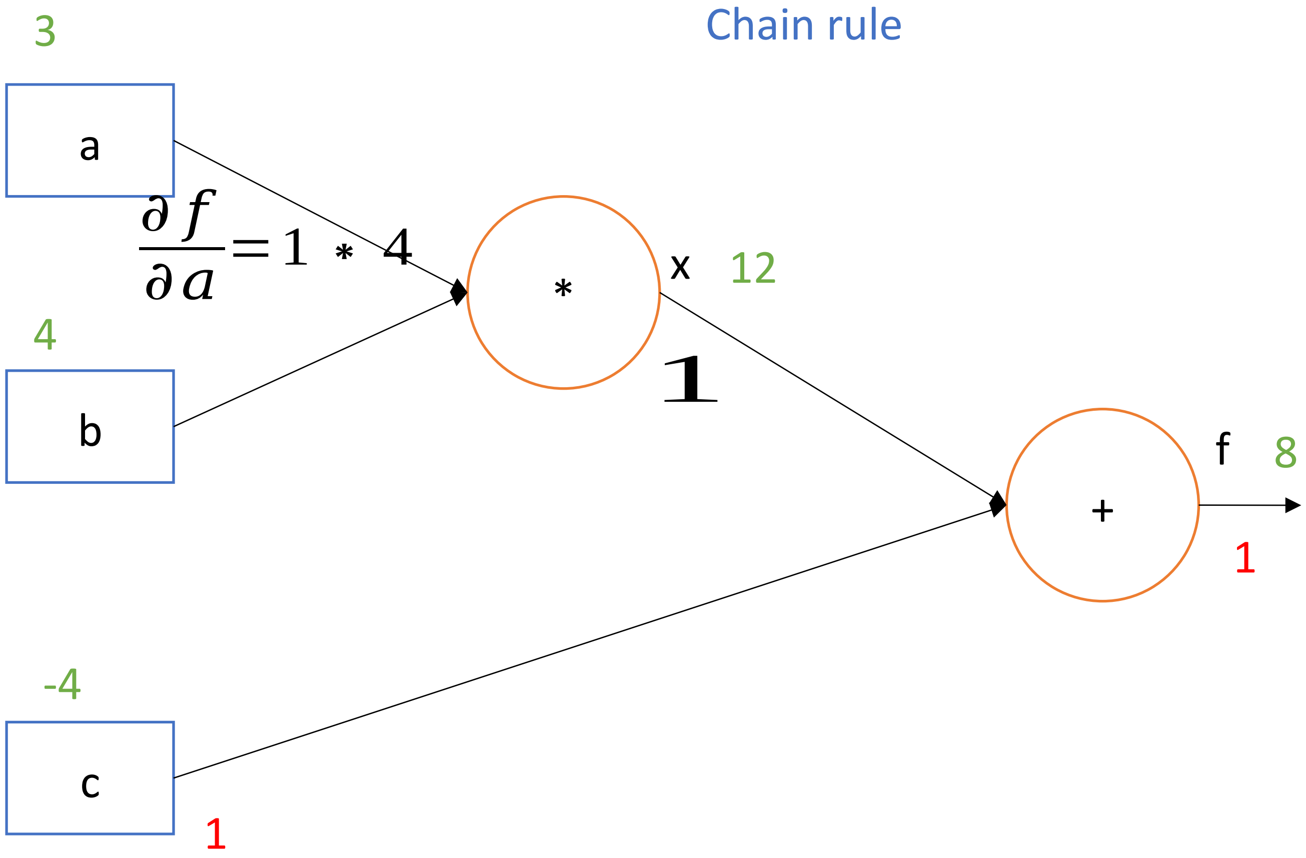
= ???

= ???

= ???

Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$f(a, b, c) = a*b + c$

x = a*b

= b

= a

f = x + c

= 1

= 1

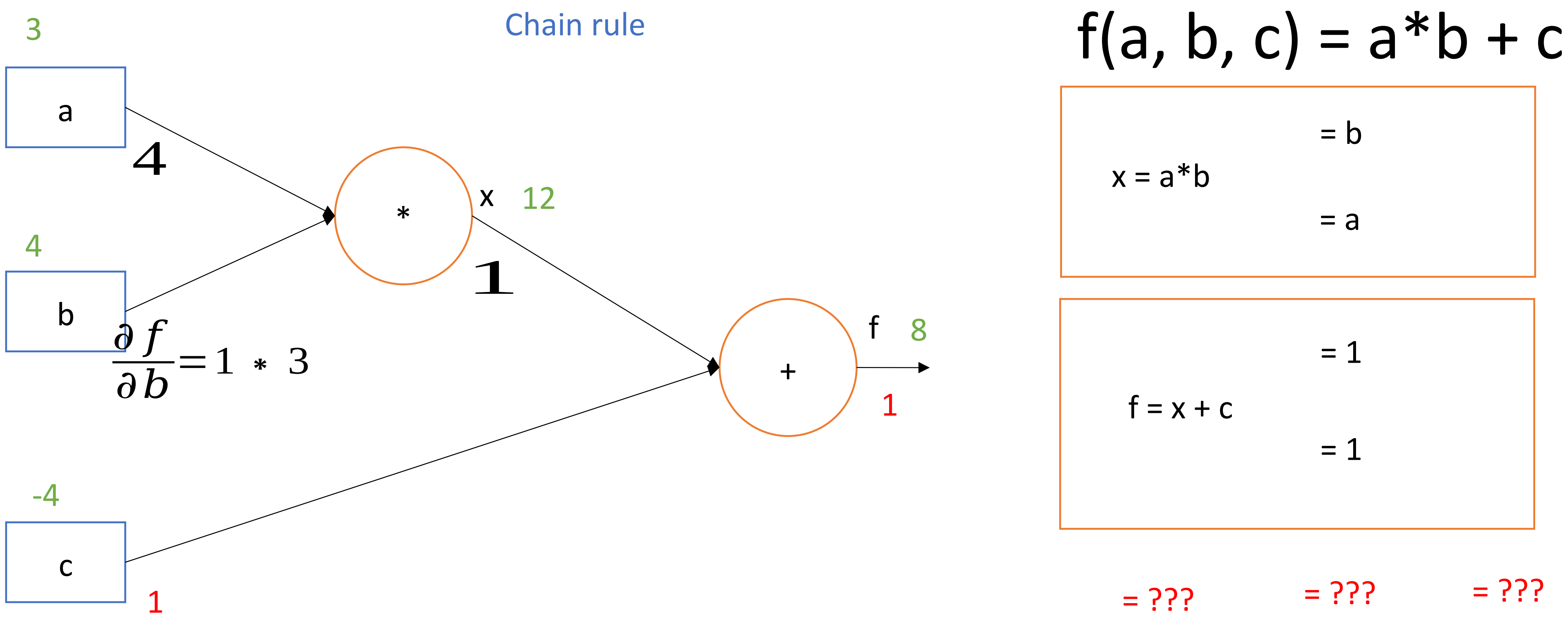
= ???

= ???

= ???

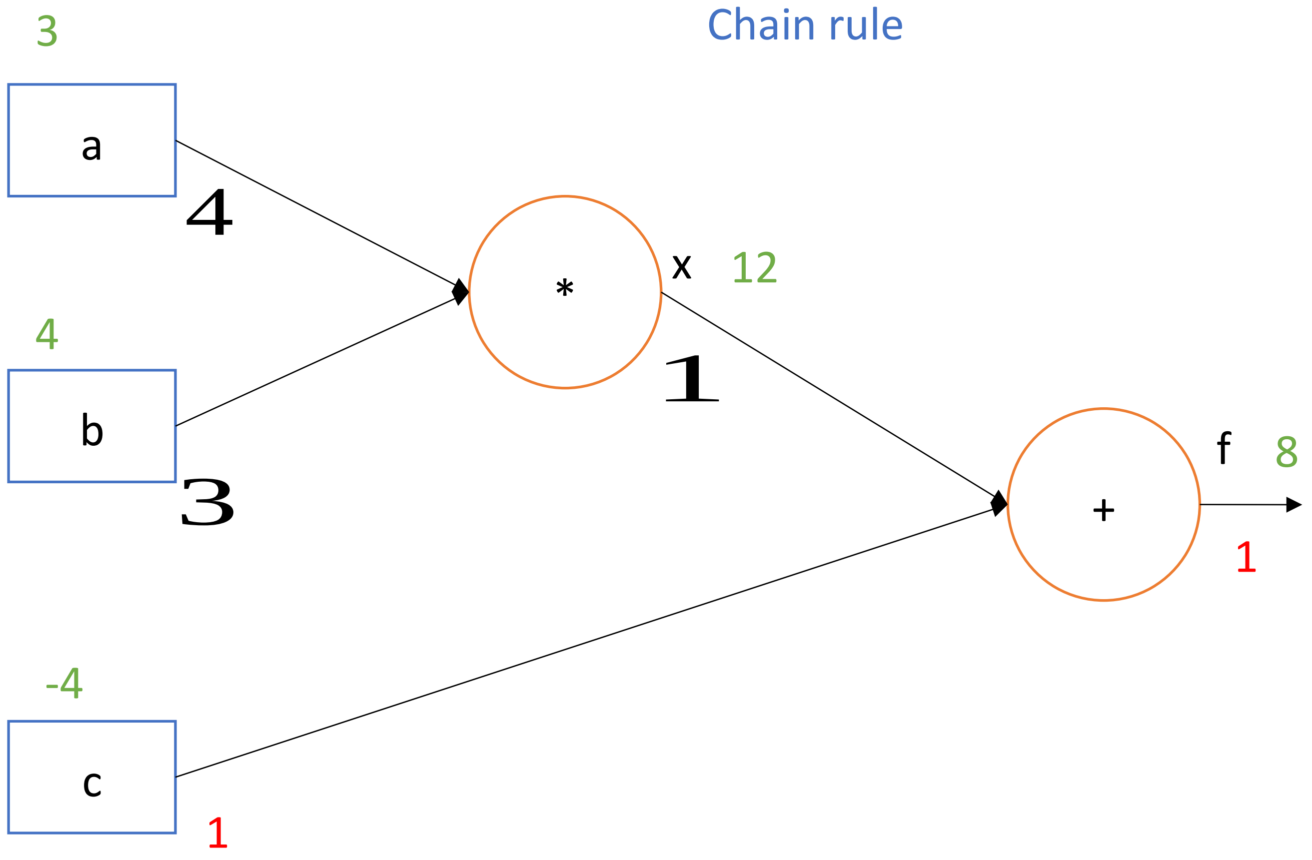
Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$$f(a, b, c) = a*b + c$$

$x = a*b$

= b

= a

$f = x + c$

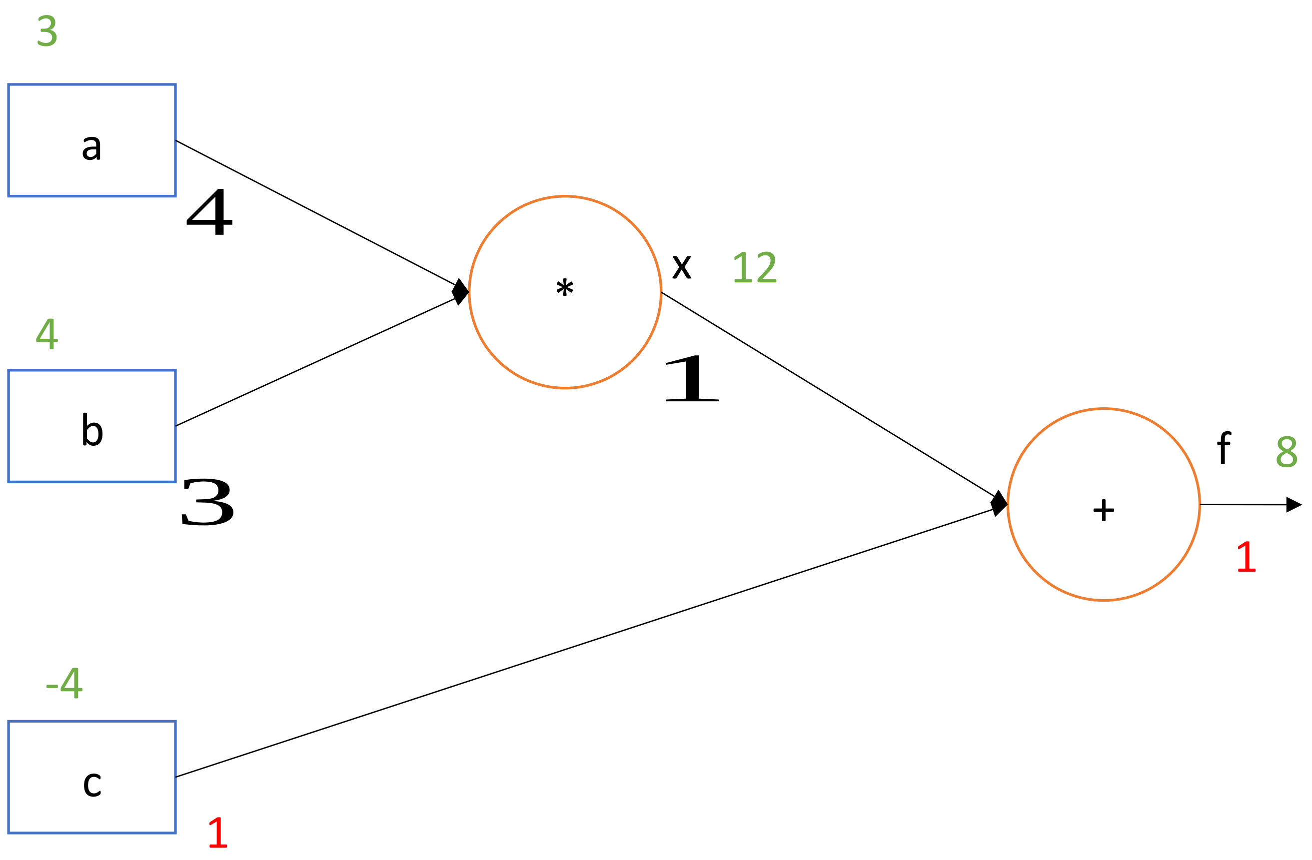
= 1

= 1

= ??? = ??? = ???

Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph

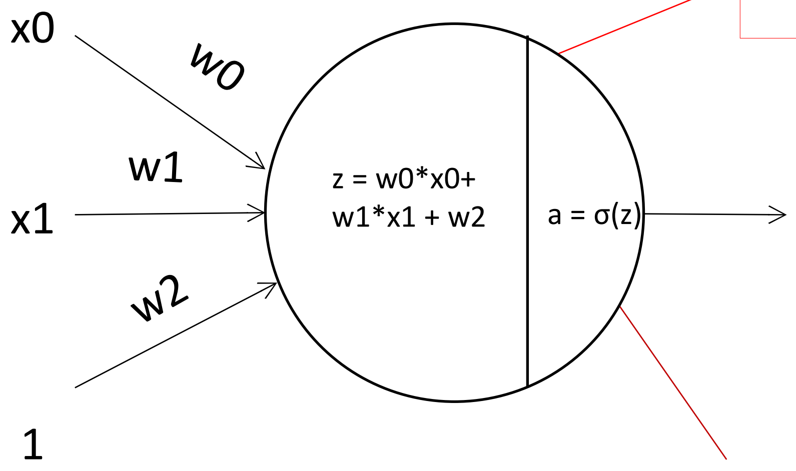


$$f(a, b, c) = a*b + c$$

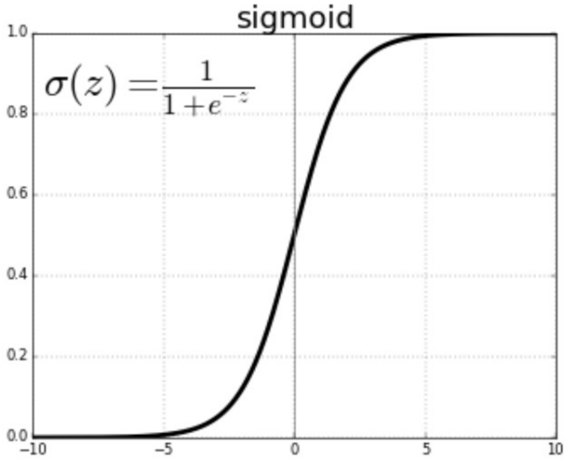
Backprop example
with a neural network in mind

In a neural network a neuron computes a linear function, followed by a non-linearity (activation function).

$$z_i = \sum_i w_i x_i + b$$



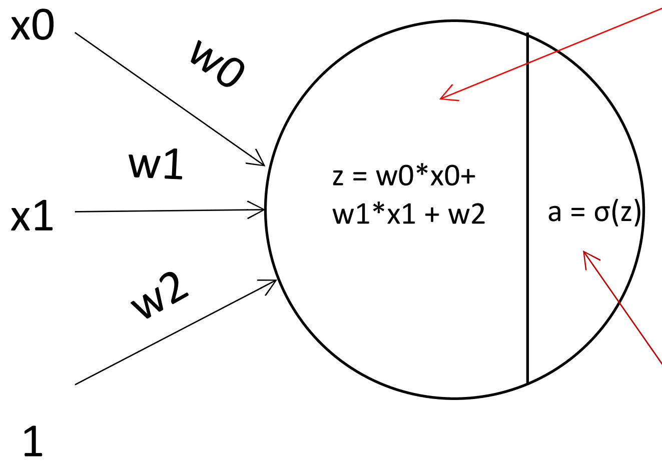
$$a_i = \sigma\left(z_i\right)$$



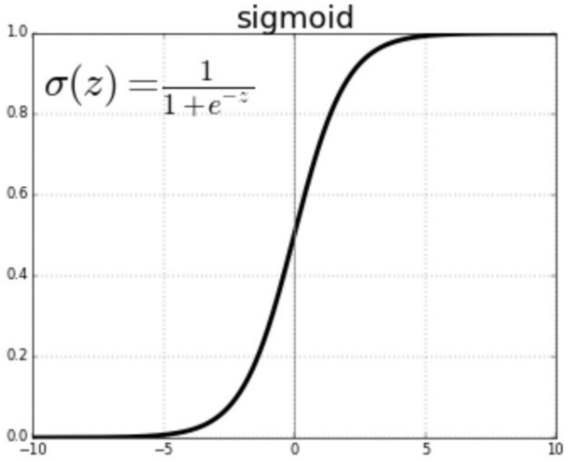
Backprop example
With a neural network in mind

In a neural network a neuron computes a linear function, followed by a non-linearity (activation function).

$$z_i = \sum_i w_i x_i + b$$

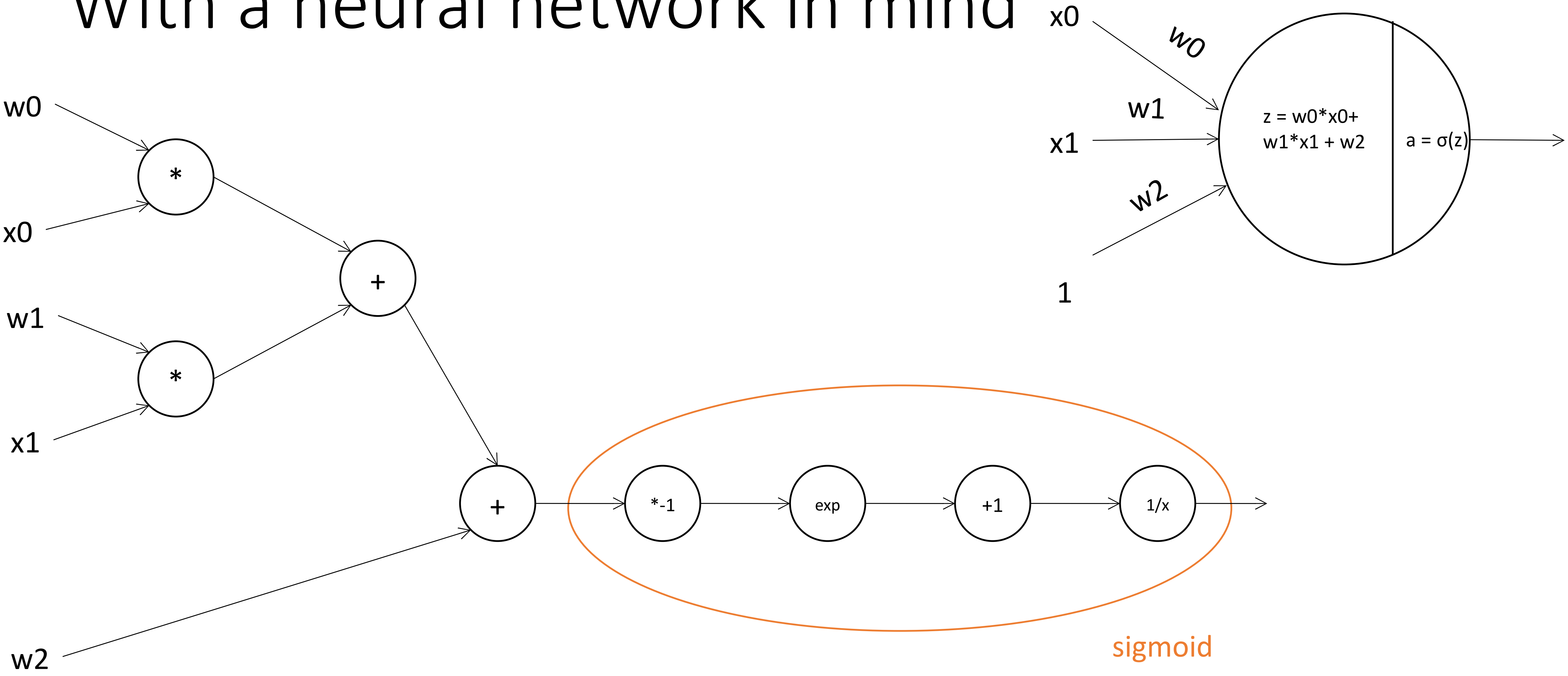


$$a_i = \sigma\left(z_i\right)$$



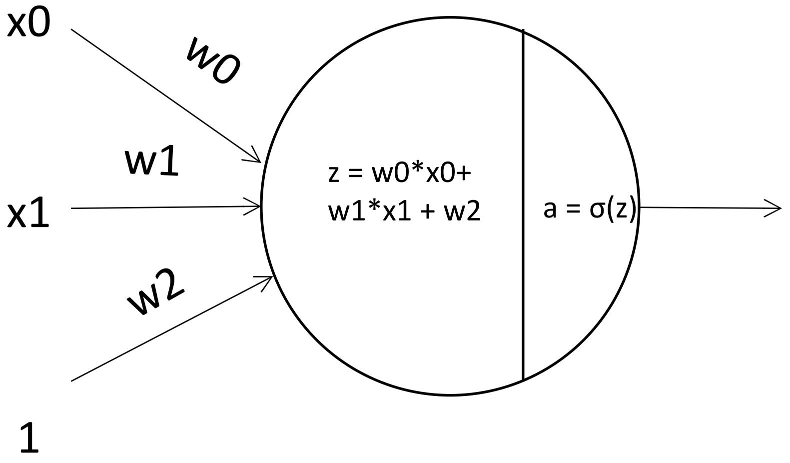
Let's write this basic neuron as a computational graph!

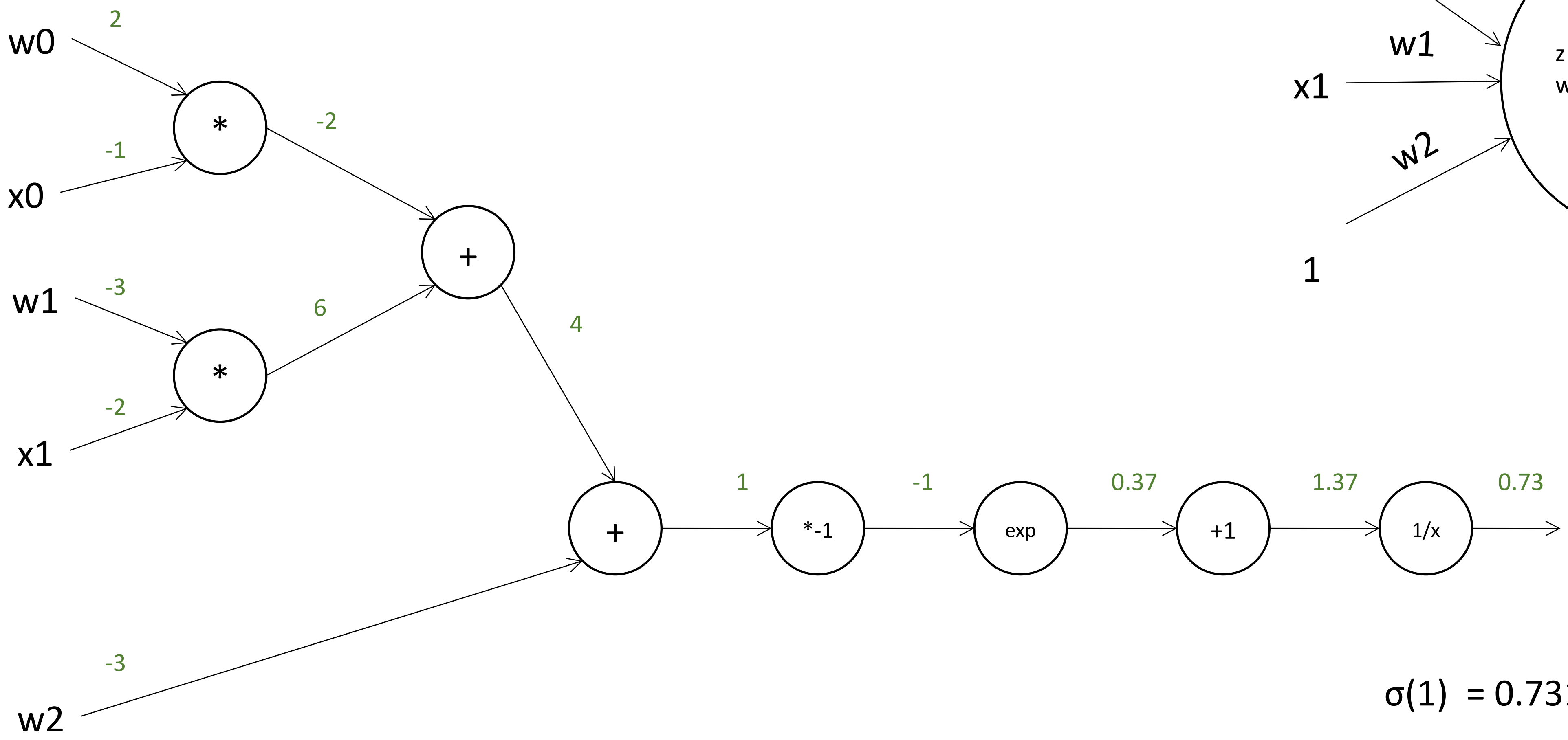
Backprop example
With a neural network in mind



Example credit: https://cs231n.github.io/optimization-2/ , figure 2

Backprop example
With a neural network in mind

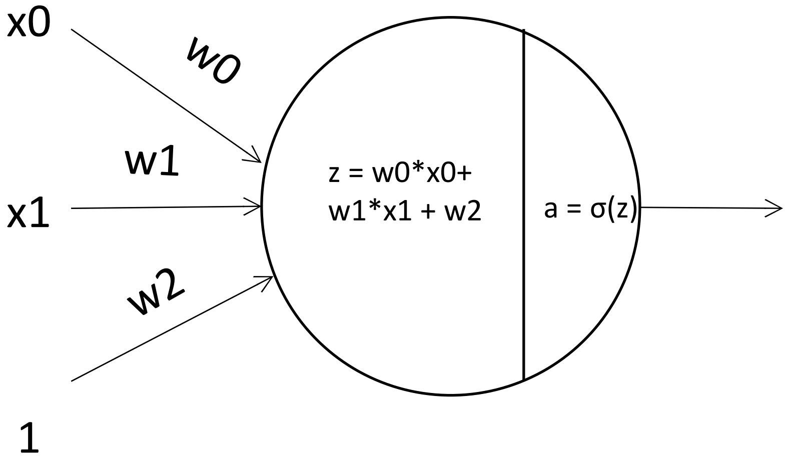


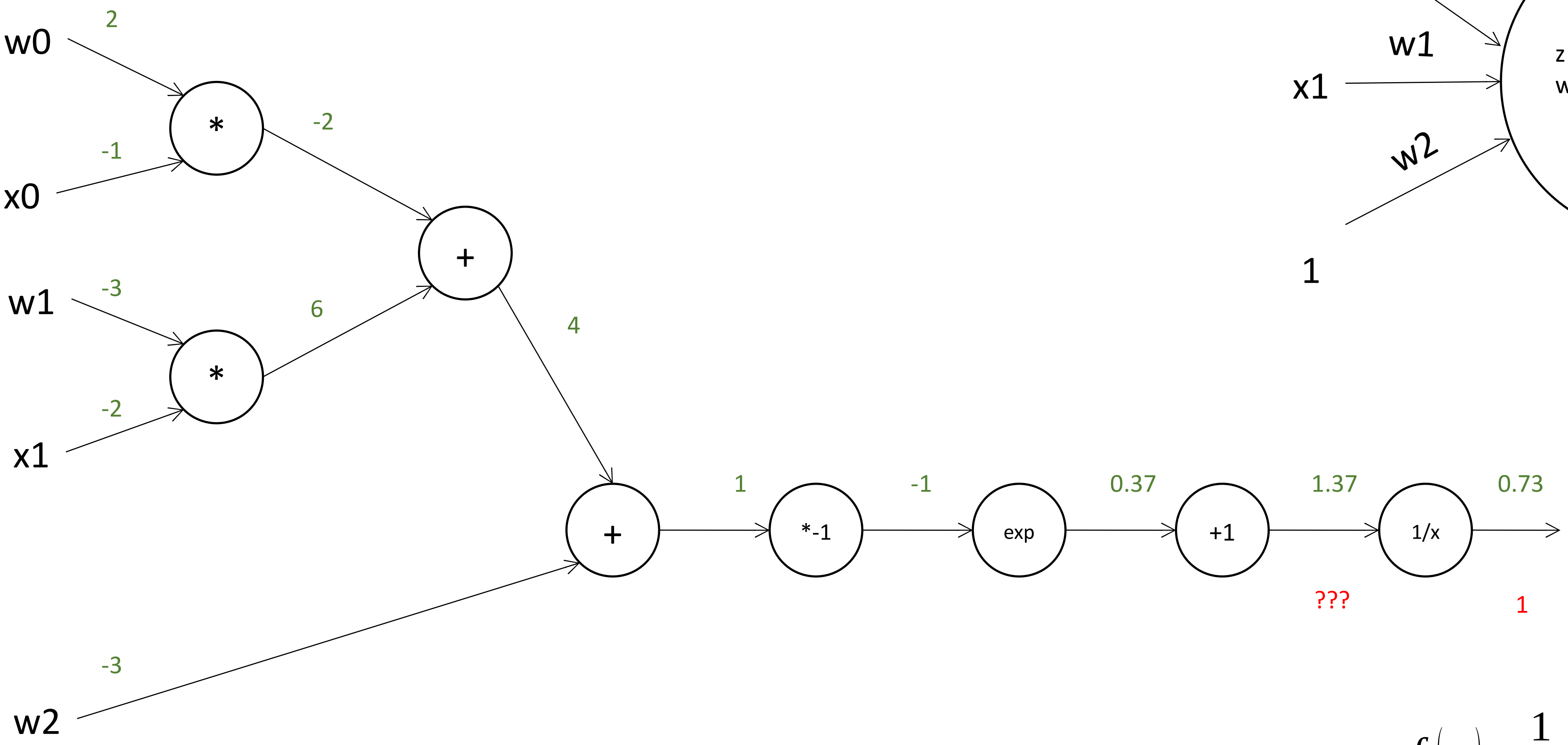


$\sigma(1) = 0.7310585786300048792512$

Example credit: https://cs231n.github.io/optimization-2/ , figure 2

Backprop example
With a neural network in mind

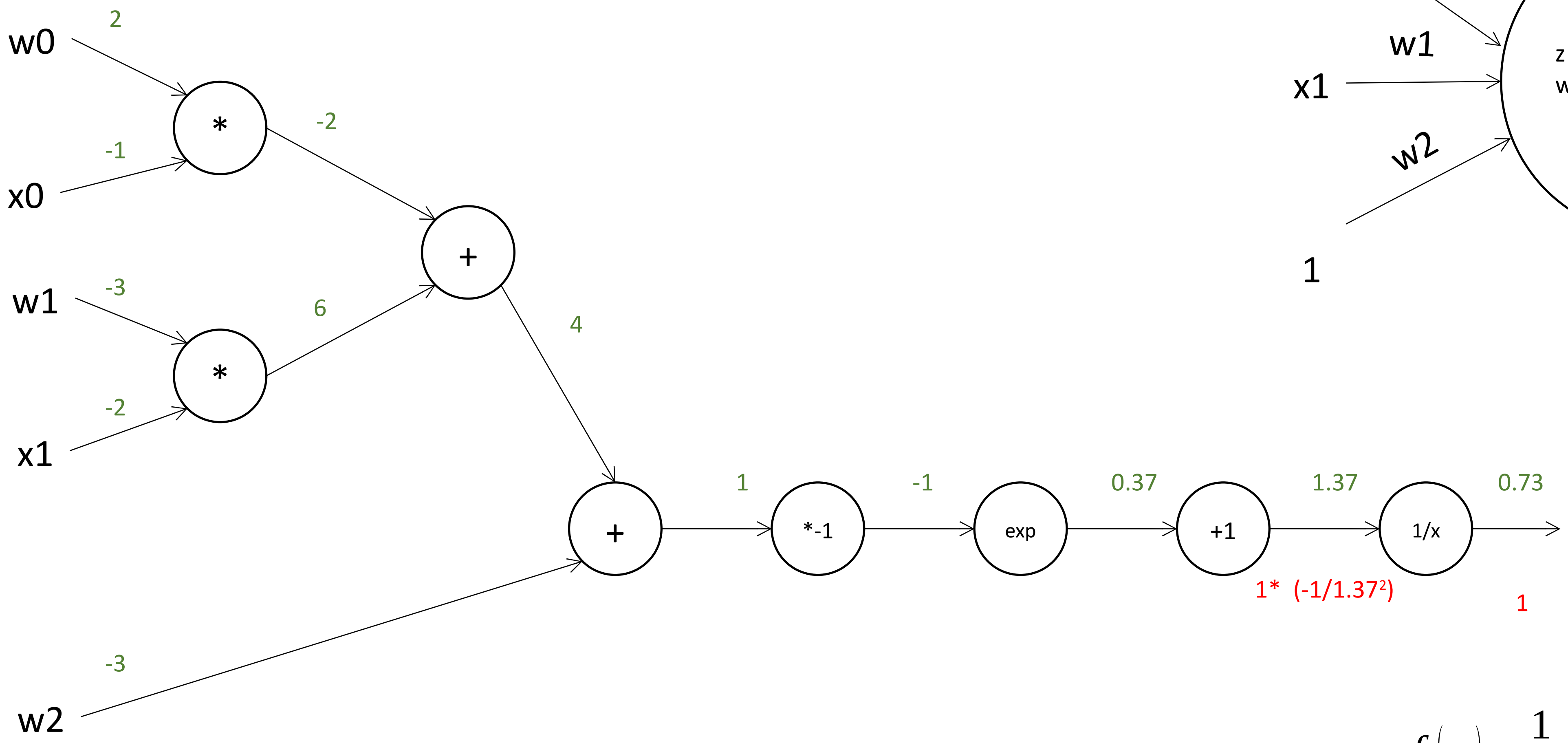


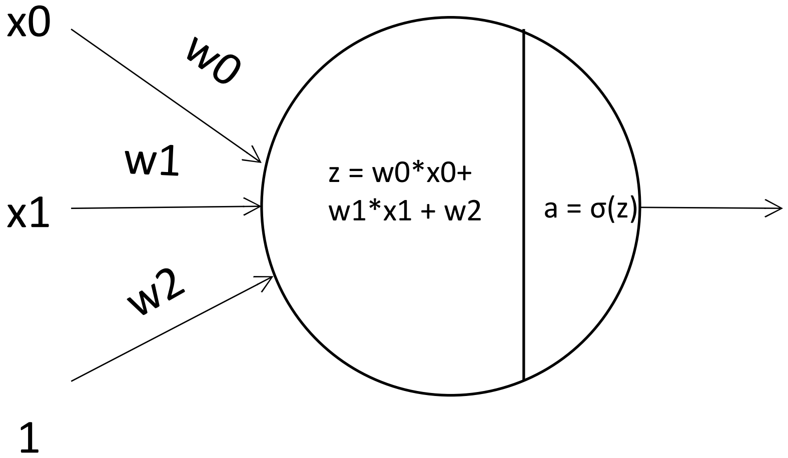


$$f(x)=\frac{1}{x} \rightarrow \frac{df}{dx}=-\frac{1}{x^2}$$

Example credit: https://cs231n.github.io/optimization-2/ , figure 2

Backprop example
With a neural network in mind

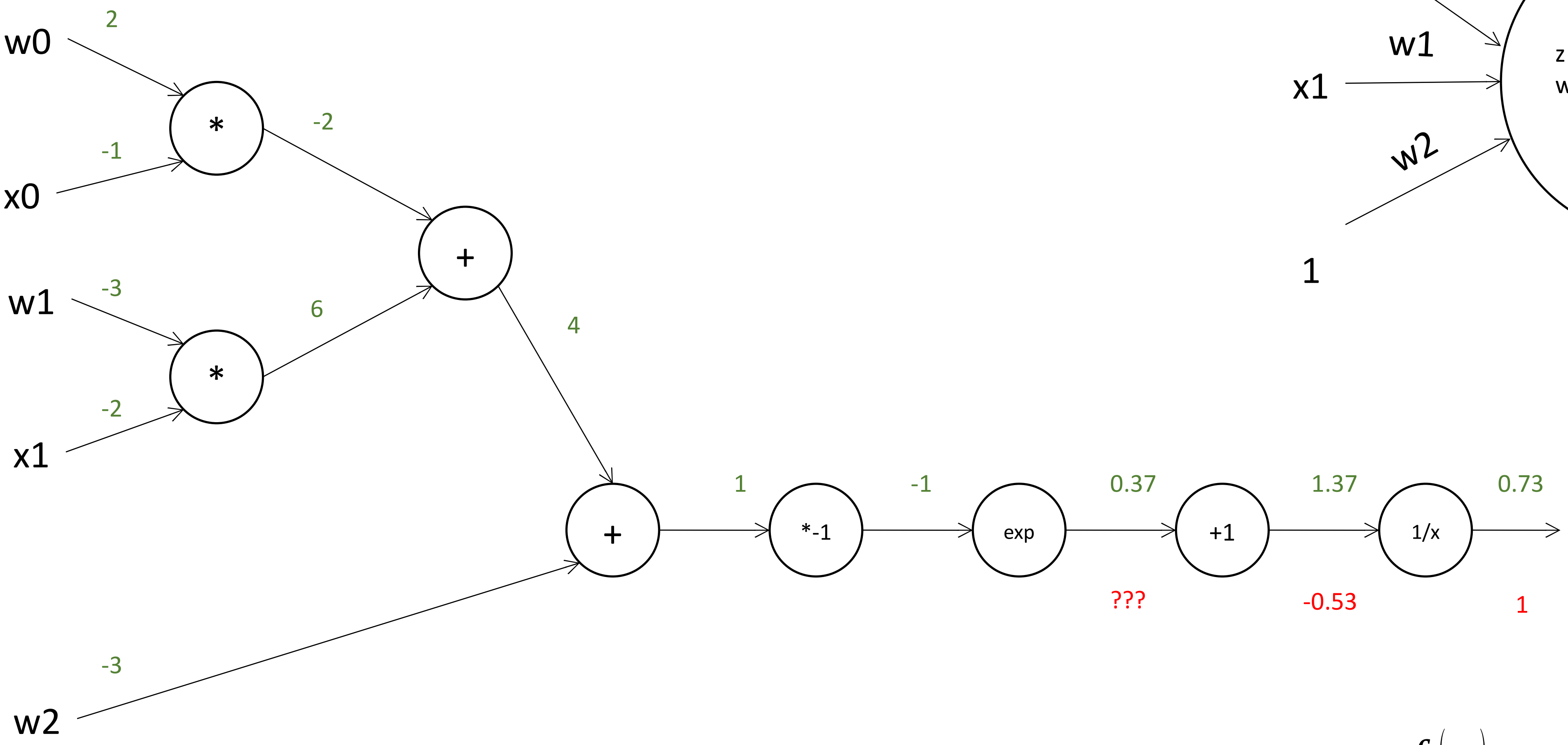


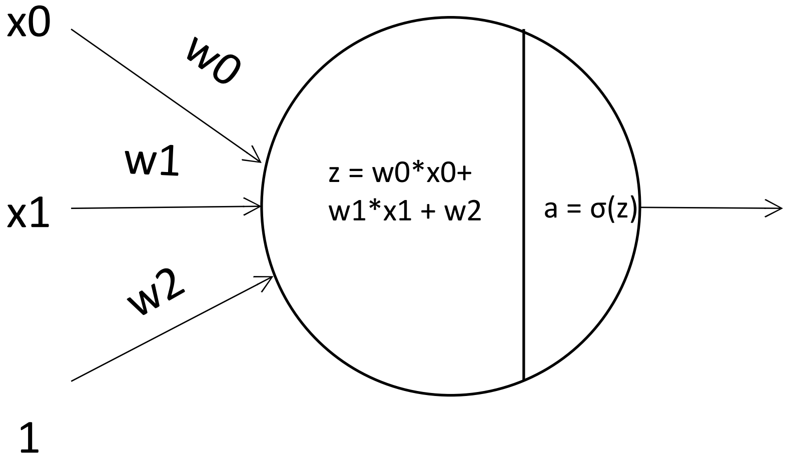


$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -\frac{1}{x^2}$$

Example credit: https://cs231n.github.io/optimization-2/ , figure 2

Backprop example
With a neural network in mind

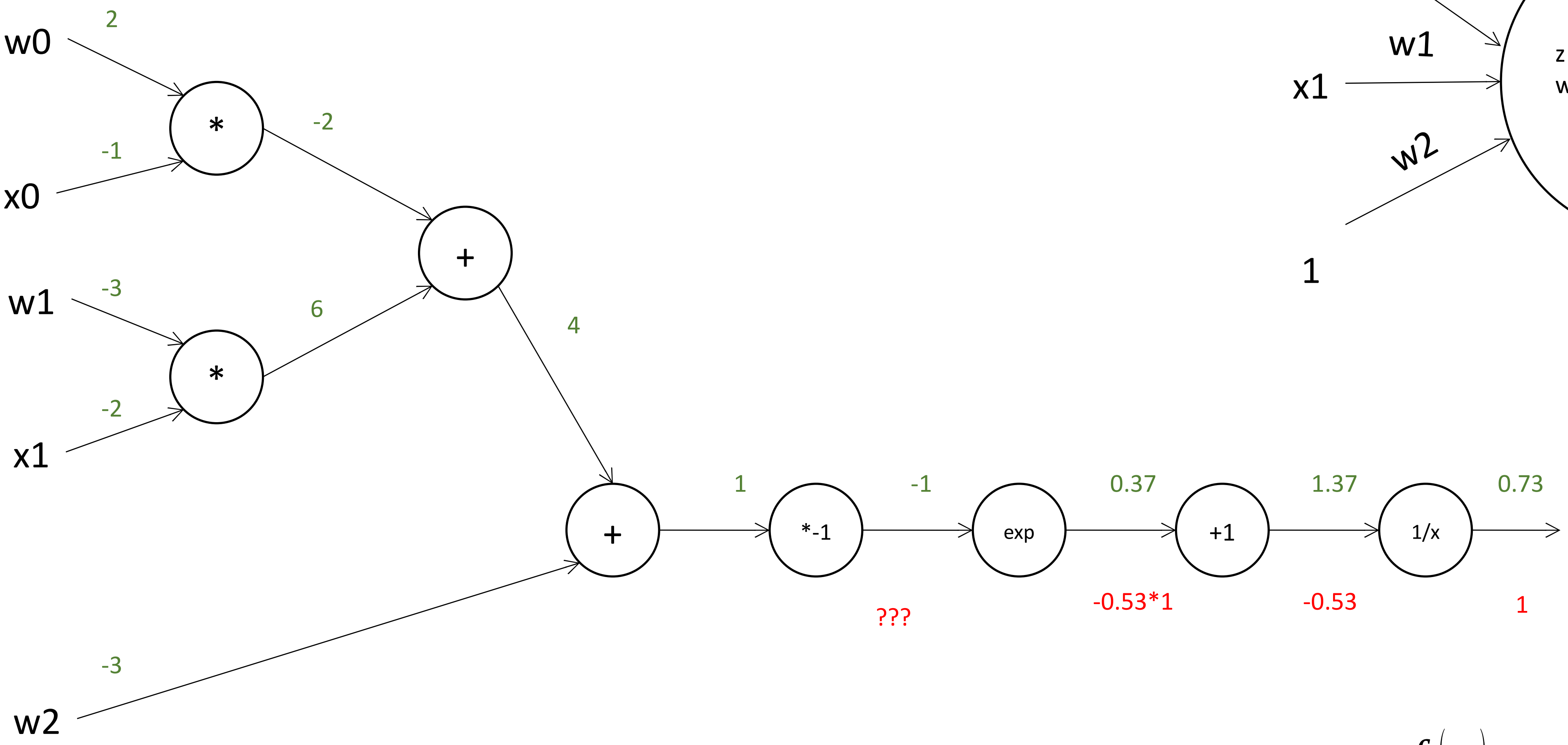


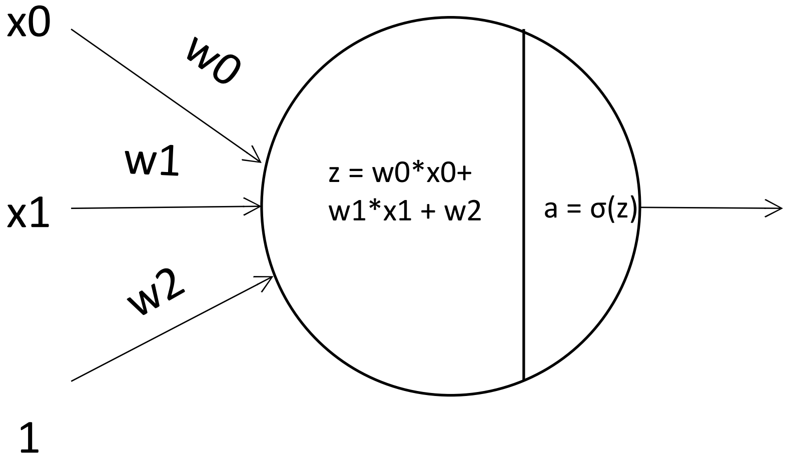


$$f(x) = x + c \rightarrow \frac{df}{dx} = 1$$

Example credit: https://cs231n.github.io/optimization-2/ , figure 2

Backprop example
With a neural network in mind

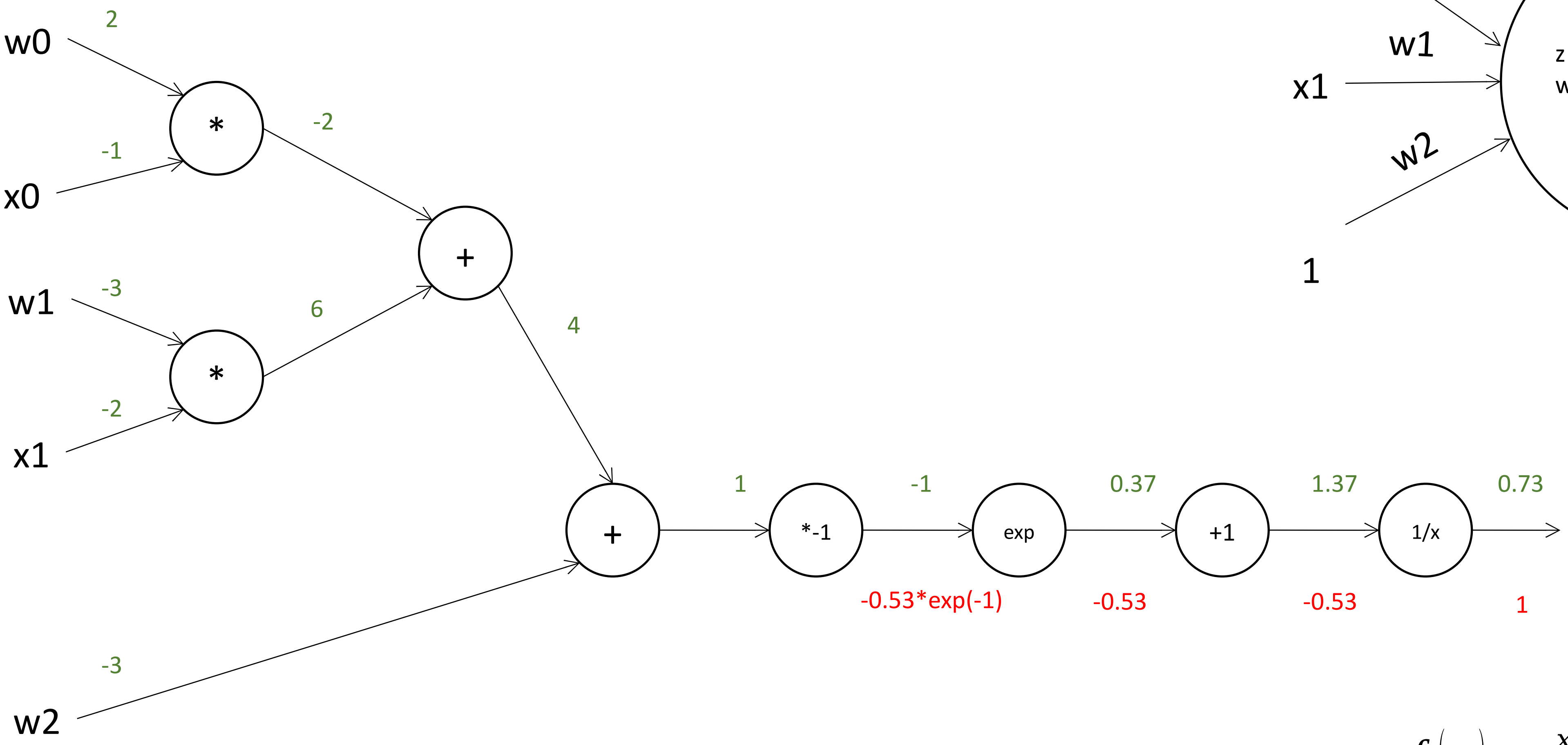


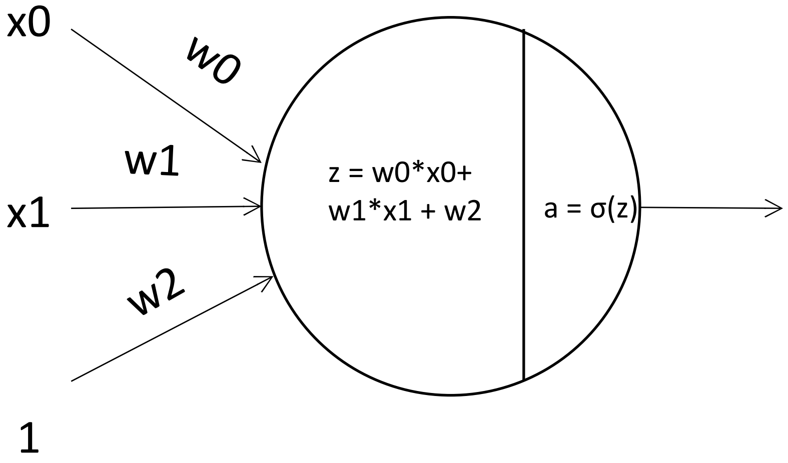


$$f(x) = x + c \rightarrow \frac{df}{dx} = 1$$

Example credit: https://cs231n.github.io/optimization-2/ , figure 2

Backprop example
With a neural network in mind

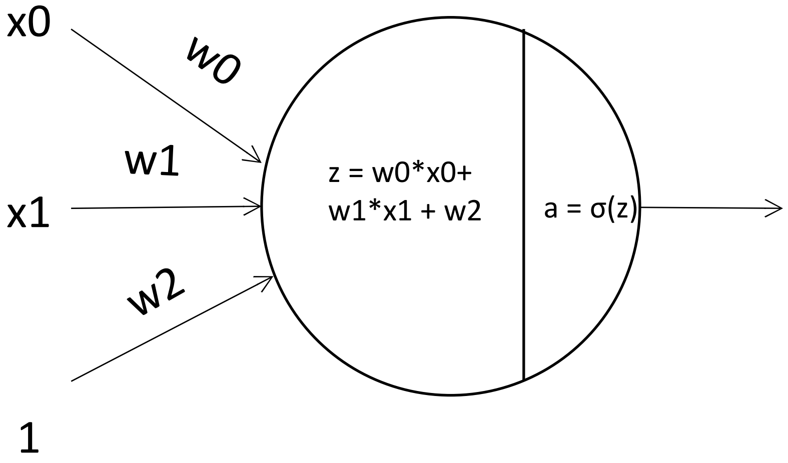


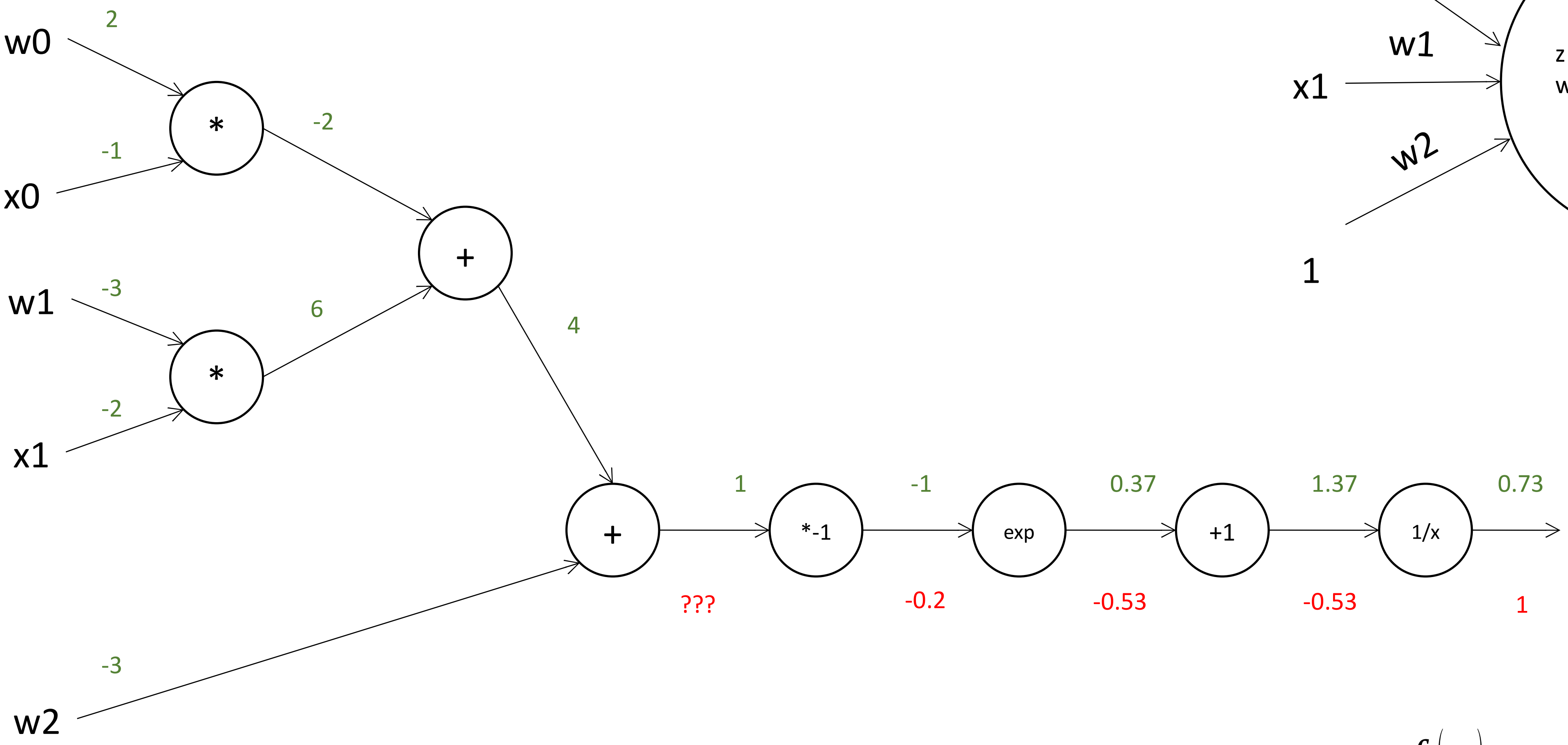


$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

Example credit: https://cs231n.github.io/optimization-2/ , figure 2

Backprop example
With a neural network in mind

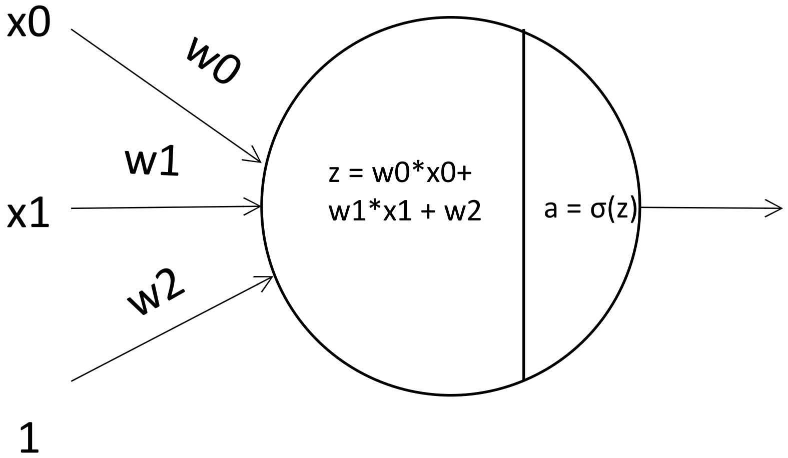


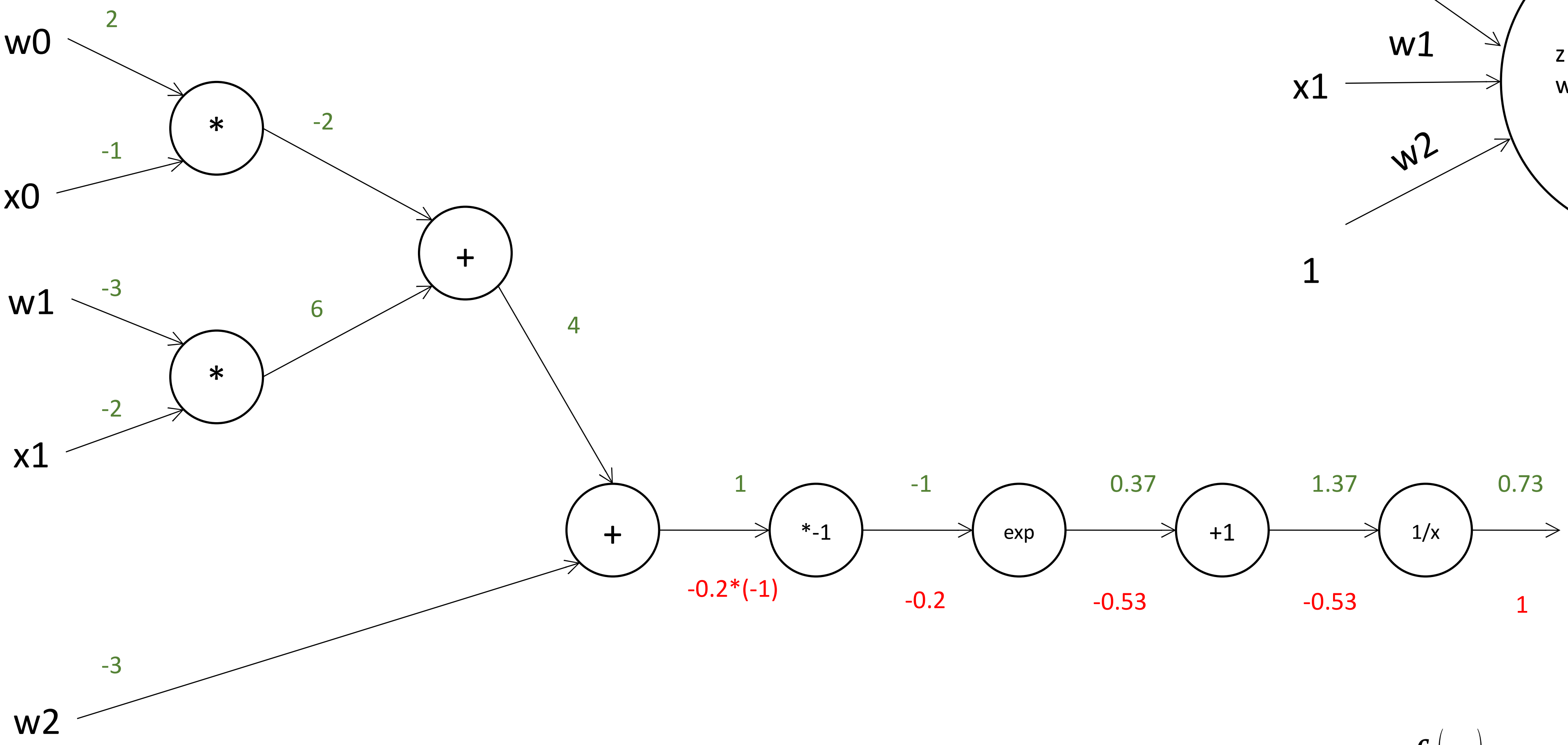


$$f(x) = ax \rightarrow \frac{df}{dx} = a$$

Example credit: https://cs231n.github.io/optimization-2/ , figure 2

Backprop example
With a neural network in mind

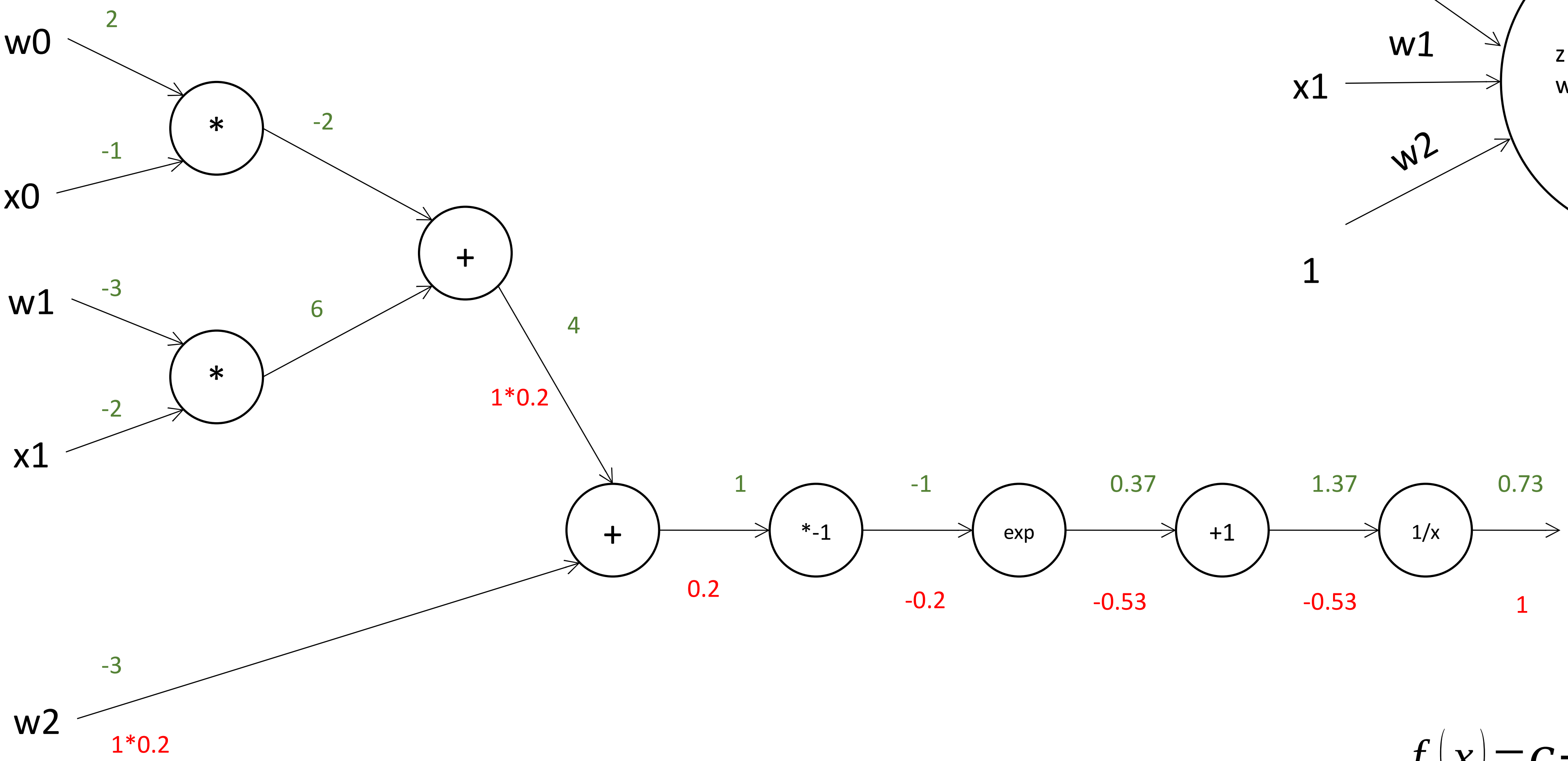


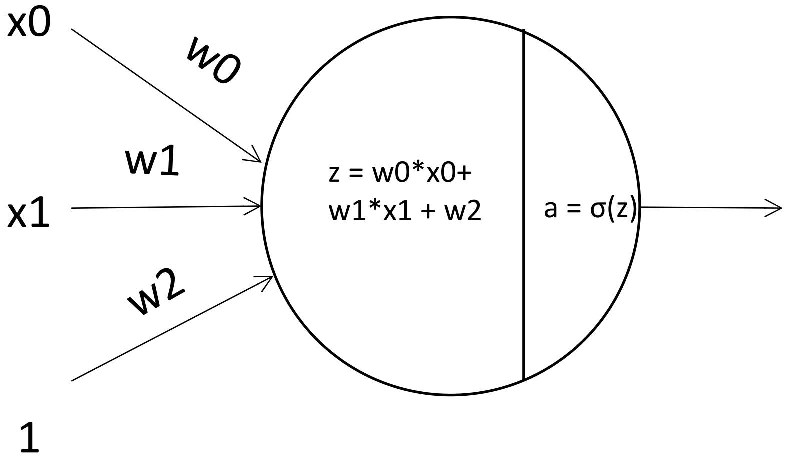


$$f(x) = ax \rightarrow \frac{df}{dx} = a$$

Example credit: https://cs231n.github.io/optimization-2/ , figure 2

Backprop example
With a neural network in mind

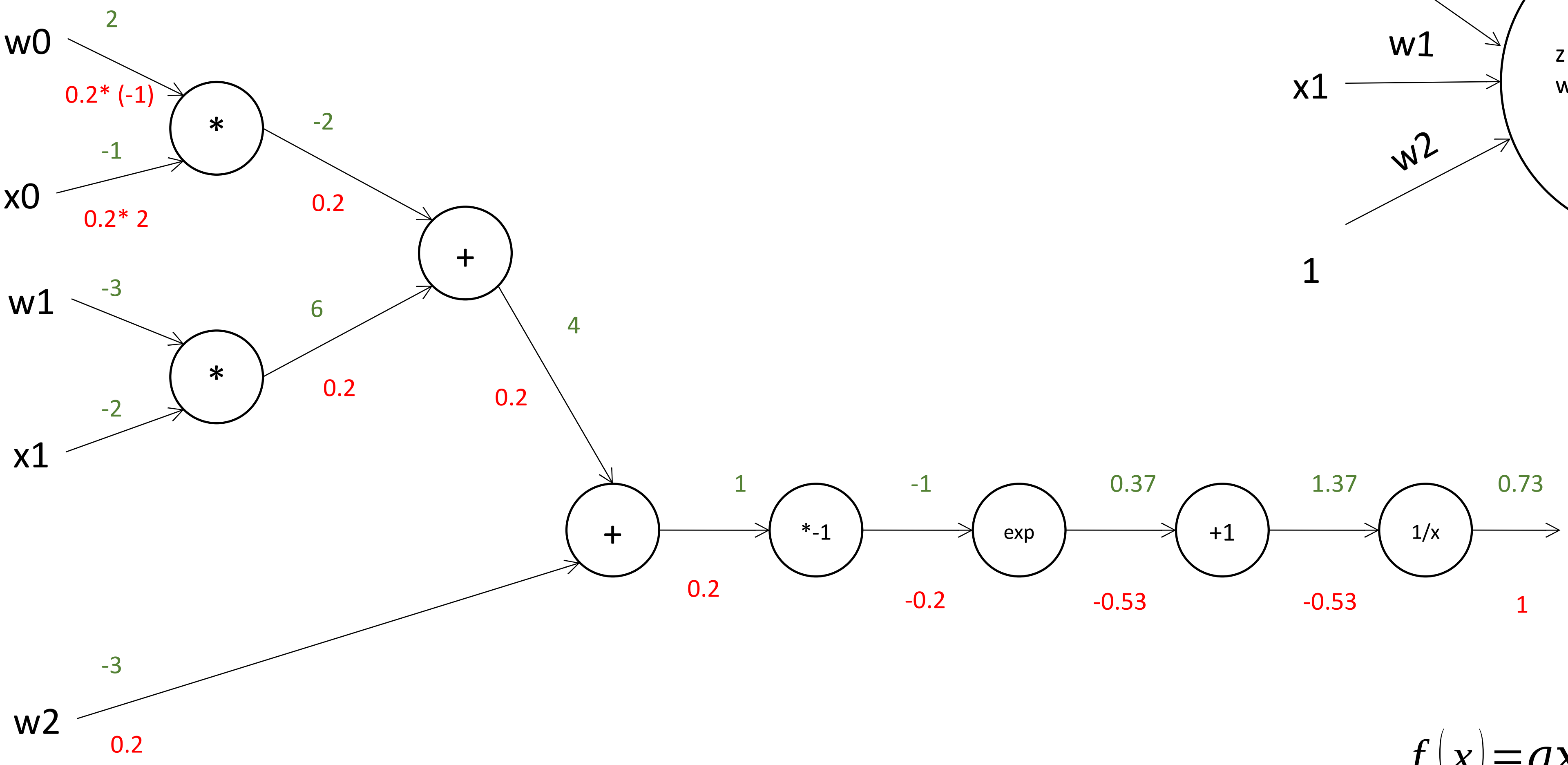


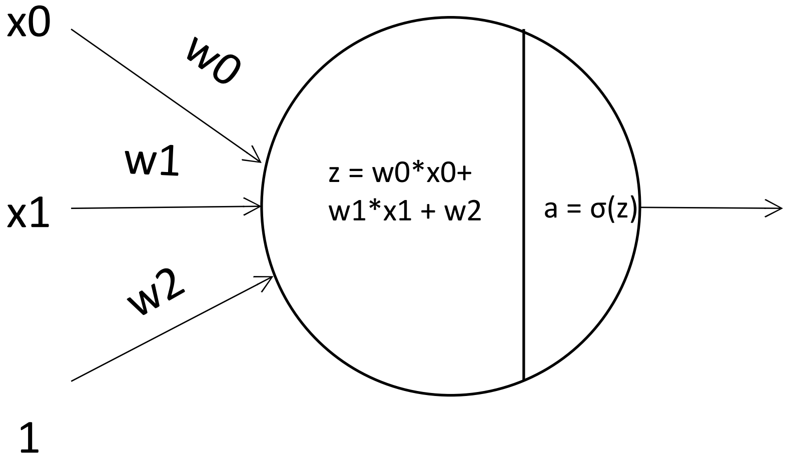


$$f(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Example credit: https://cs231n.github.io/optimization-2/ , figure 2

Backprop example
With a neural network in mind

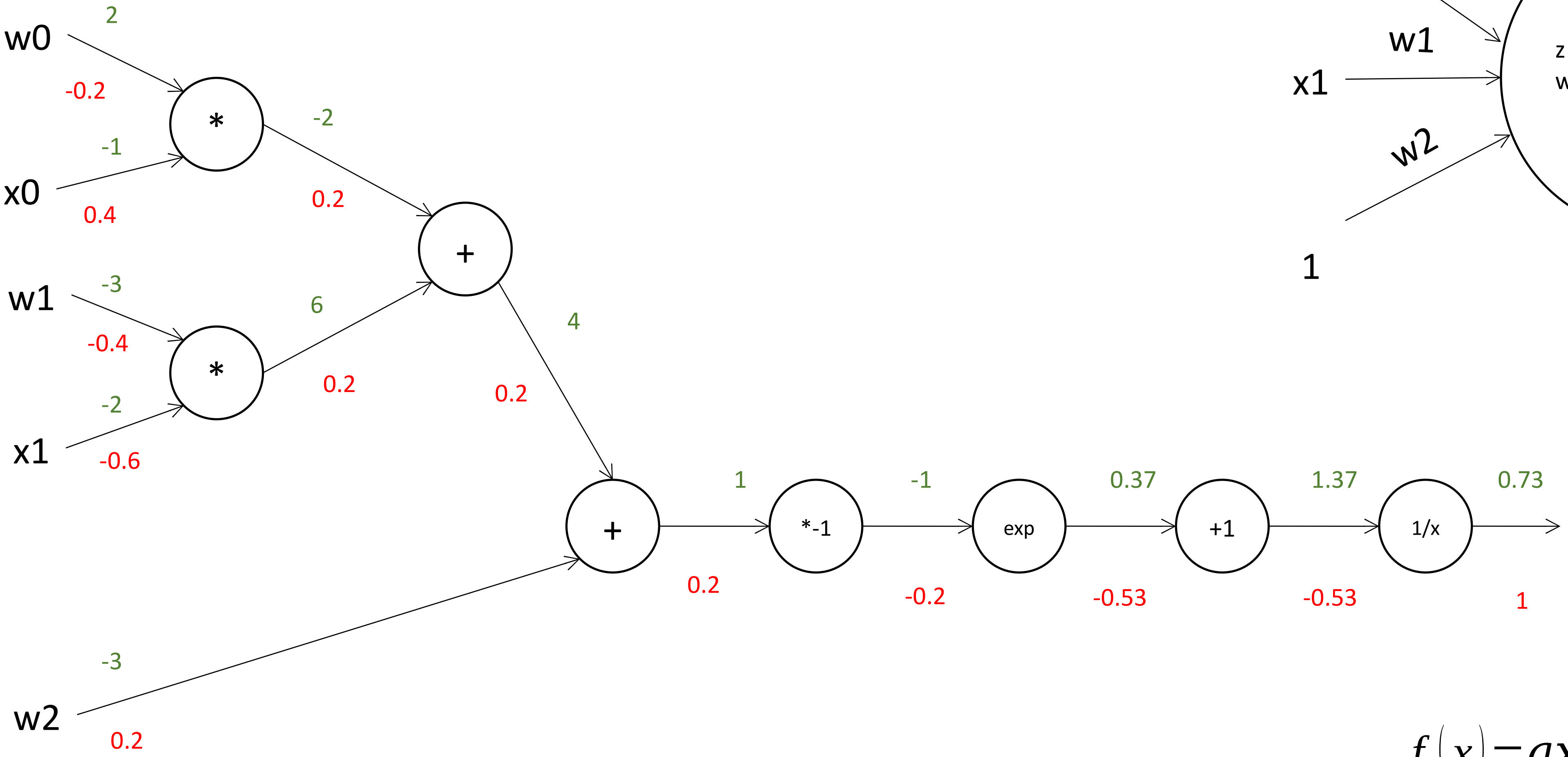


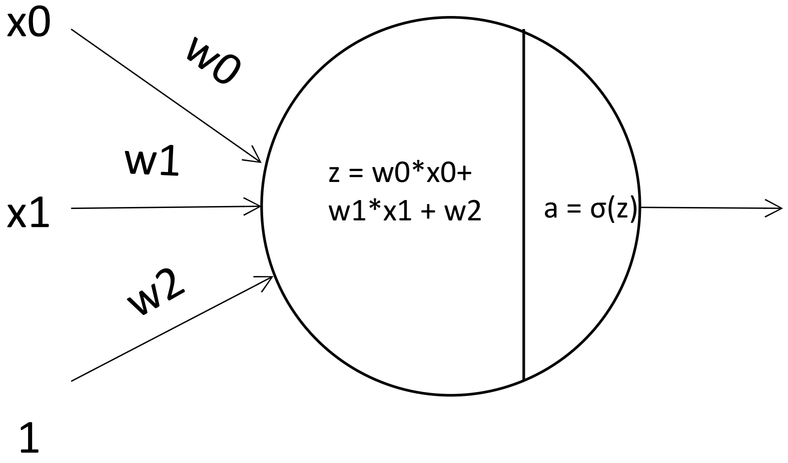


$$f(x) = ax \rightarrow \frac{df}{dx} = a$$

Example credit: https://cs231n.github.io/optimization-2/ , figure 2

Backprop example
With a neural network in mind



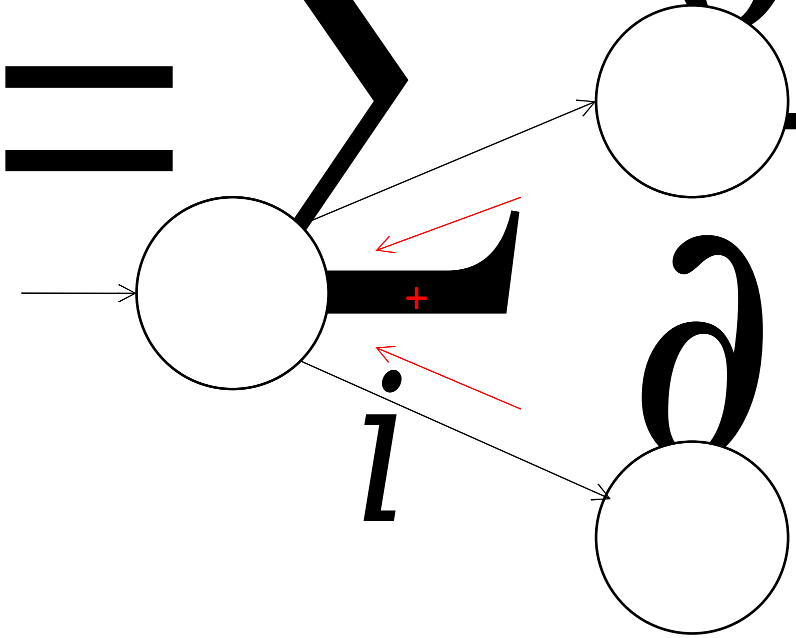


$$f(x) = ax \rightarrow \frac{df}{dx} = a$$

Example credit: https://cs231n.github.io/optimization-2/ , figure 2

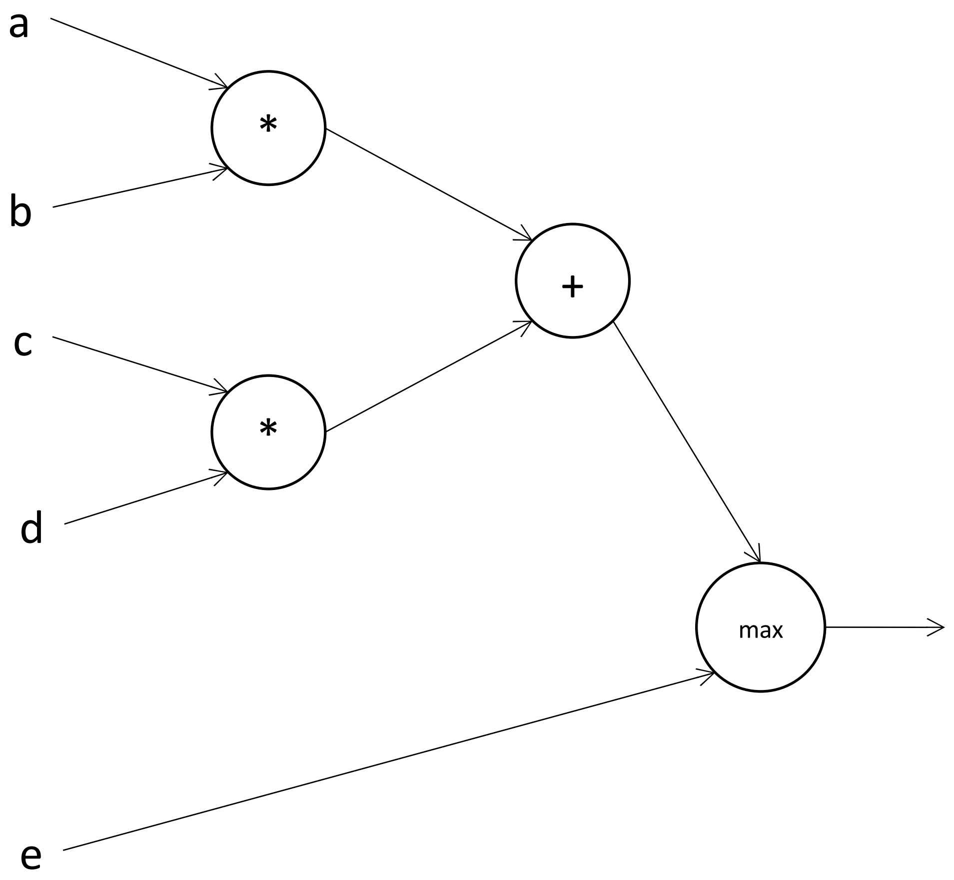
Multivariate chain rule

$$\frac{\partial f}{\partial x} = \sum_i \frac{\partial f}{\partial q_i} \frac{\partial q_i}{\partial x}$$



Patterns in backpropagation

- Addition gate: gradient distributor
- Multiplication gate: gradient switcher
- Max gate: gradient router
- Copy gate: gradient adder



Back-propagation for vectors

Scalar to Scalar

$x \in \mathbb{R}, y \in \mathbb{R}$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

Vector to Scalar

$x \in \mathbb{R}^N, y \in \mathbb{R}$

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N \quad \left(\frac{\partial y}{\partial x}\right)_n = \frac{\partial y}{\partial x_n}$$

For each element of x, if it changes by a small amount then how much will y change?

Vector to Vector

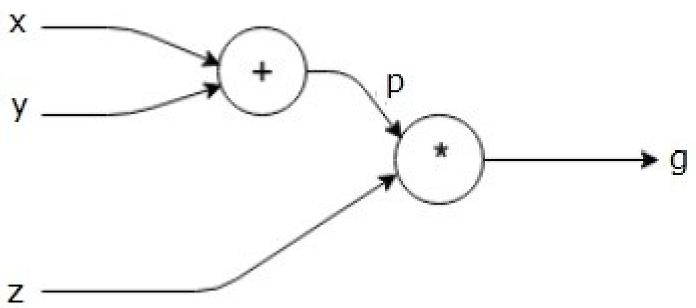
$x \in \mathbb{R}^N, y \in \mathbb{R}^M$

Derivative is **Jacobian**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left(\frac{\partial y}{\partial x}\right)_{n,m} = \frac{\partial y_m}{\partial x_n}$$

For each element of x, if it changes by a small amount then how much will each element of y change?

Forward/backward API




```
class ComputationalGraph():
    …
    def forward(inputs):
        for node in topological_sort(self.nodes):
            # forward inputs through each node
            # (also cache these inputs)
            node.forward()
        return predictions
    …
    def backward(loss):
        for node in reversed(topological_sort(self.nodes)):
            # backpropagation (apply chain rule)
            node.backward()
        return input_gradients
```

Mark I Perceptron machine: first implementation of the perceptron algorithm (~1957)

- It was connected to a camera with 20×20 cadmium sulfide photocells to make a 400-pixel image
- Weights were encoded in [potentiometers](), and weight updates during learning were performed by electric motors

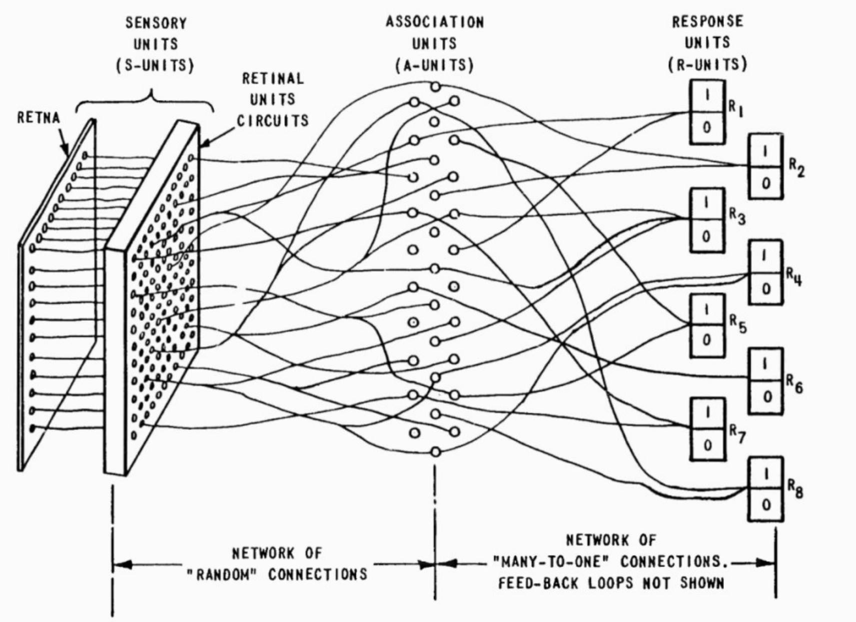


Figure I ORGANIZATION OF THE MARK I PERCEPTRON

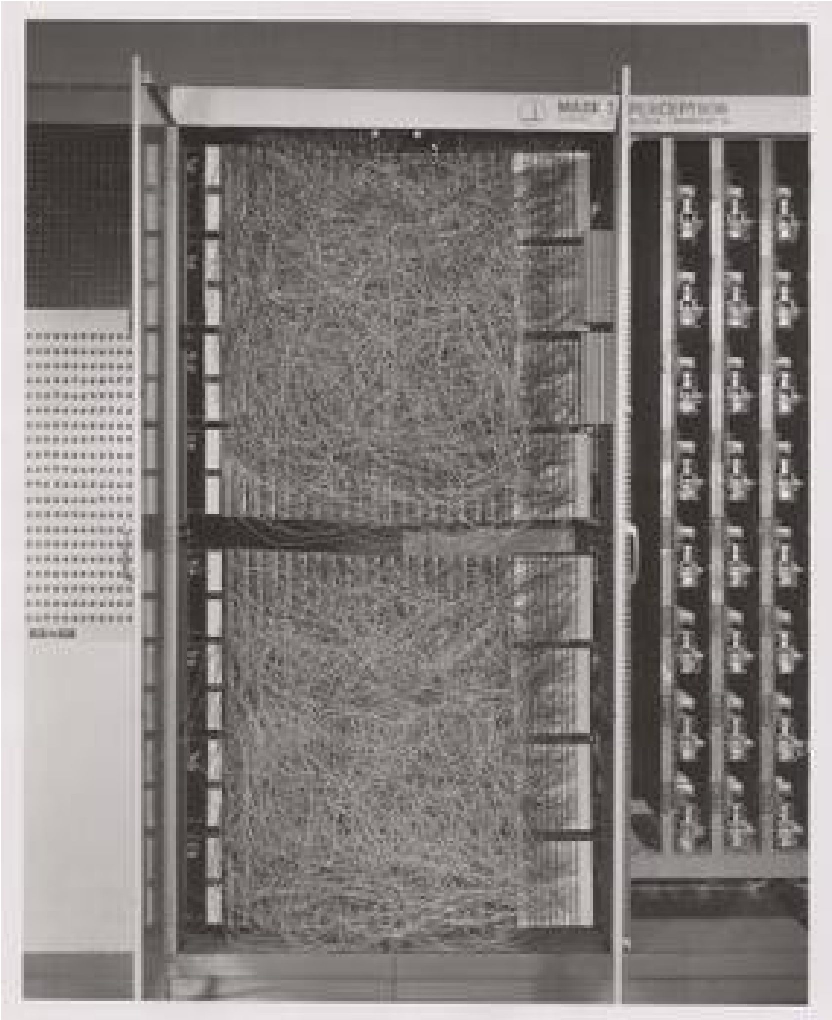
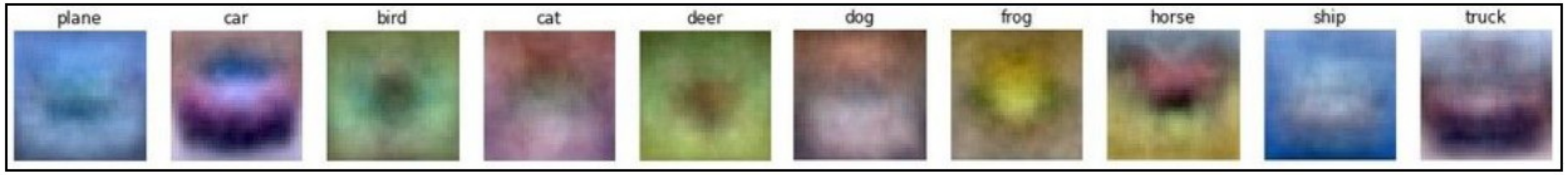


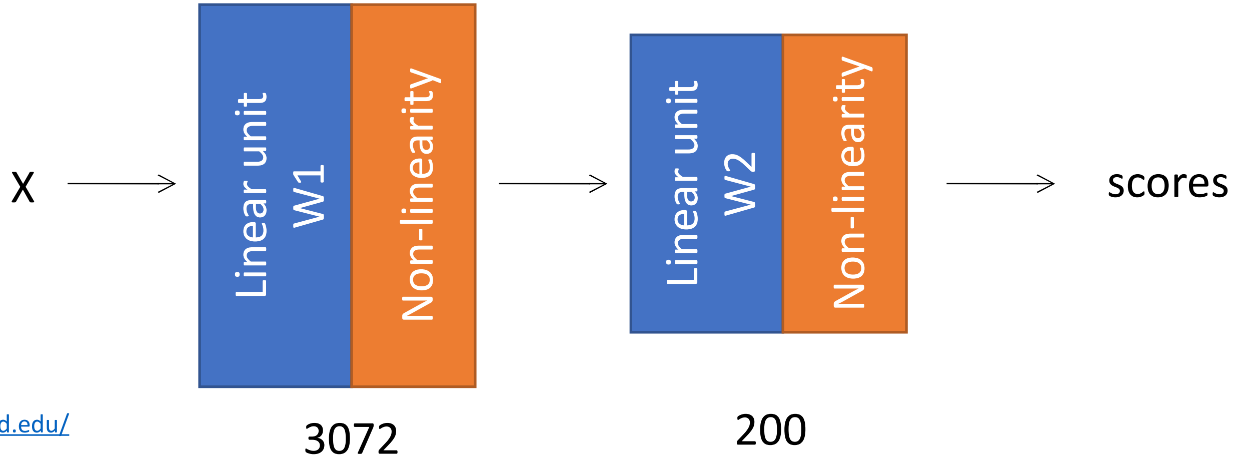
Image source: https://en.wikipedia.org/wiki/Perceptron

https://www.youtube.com/watch?v=cNxadbrN_aI

Neural networks

- Neural network (multi layer perceptron, fully connected networks) :
  - Stack two (or more) linear classifiers on top of each other with a non-linear <u>activation</u> function in between





Templates image source: http://cs231n.stanford.edu/

Why do we need activation functions?

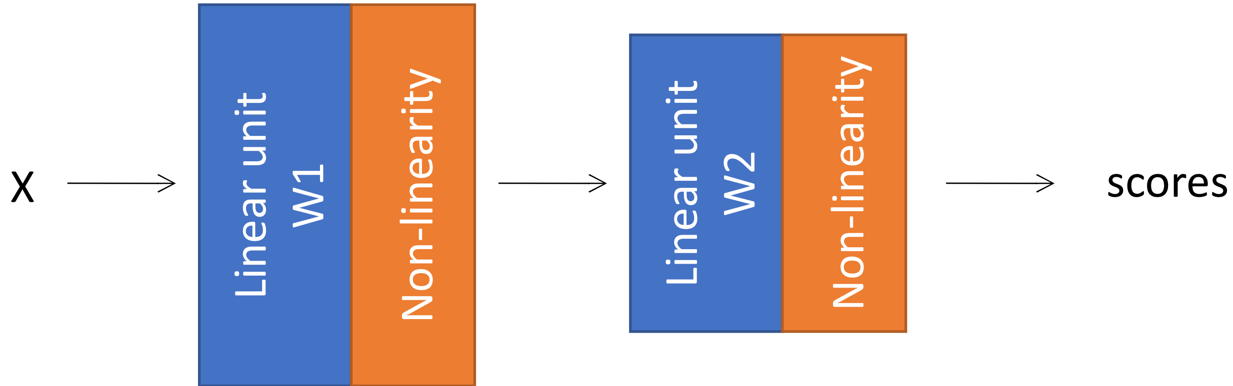
Layer 1:

$Z1 = W1 \cdot X$

Layer 2:

$Z2 = W2 \cdot g(W1 \cdot X)$, where g is the activation function

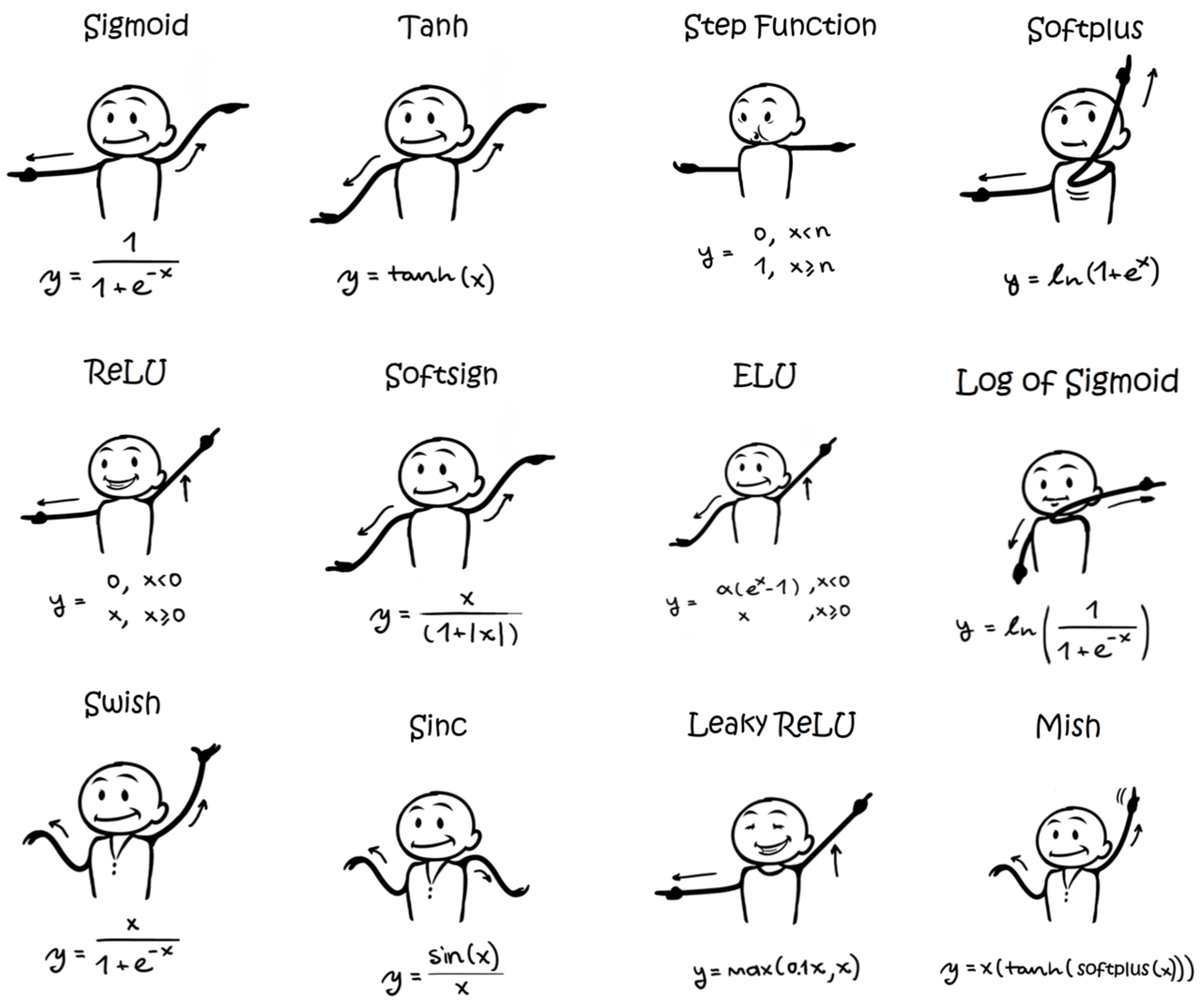
What happens if we didn't use a non-linearity?



Activation functions

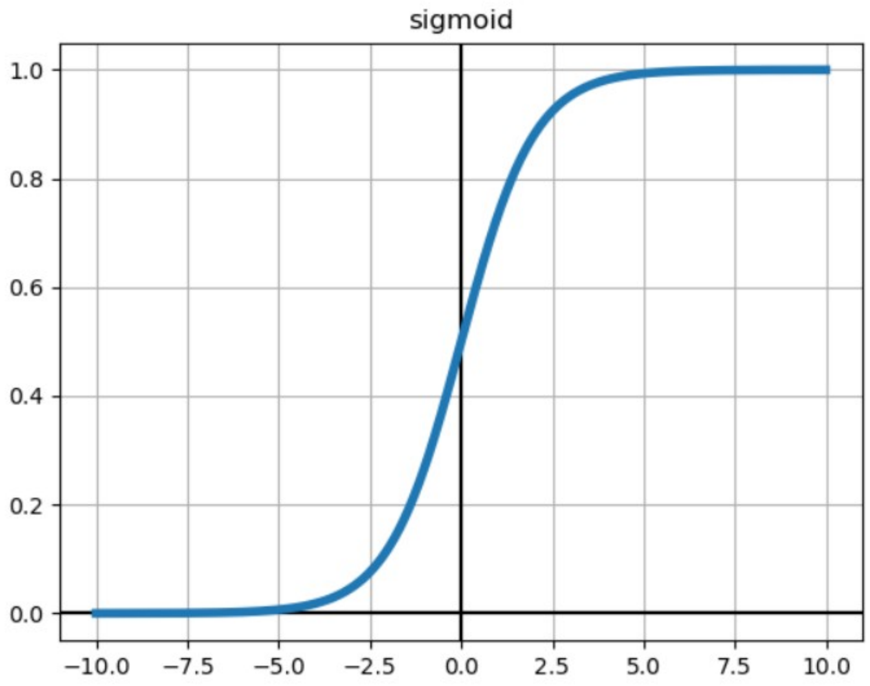
Two properties:

- Differentiable
- Non linear

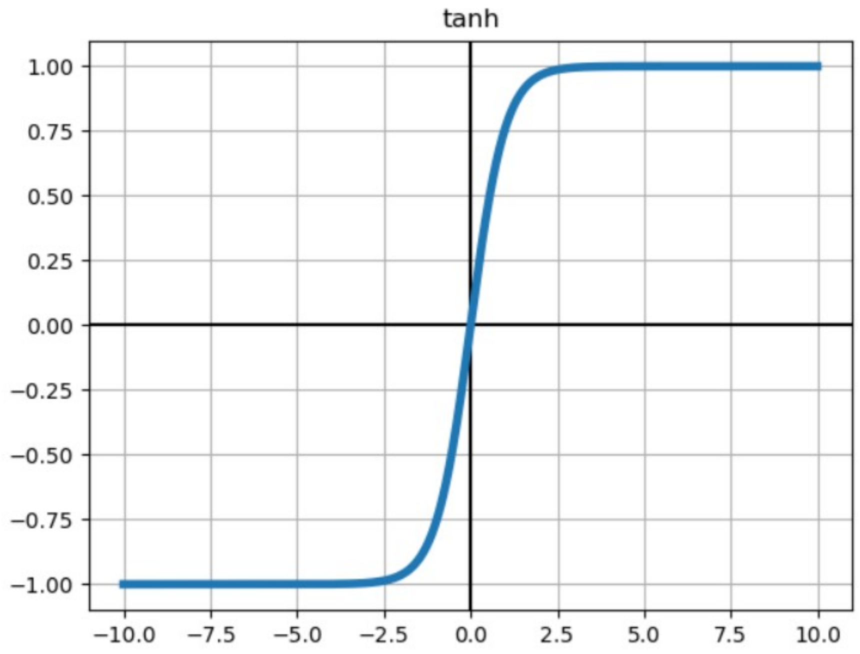


Commonly used activation functions and their derivative
https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html

Activation functions

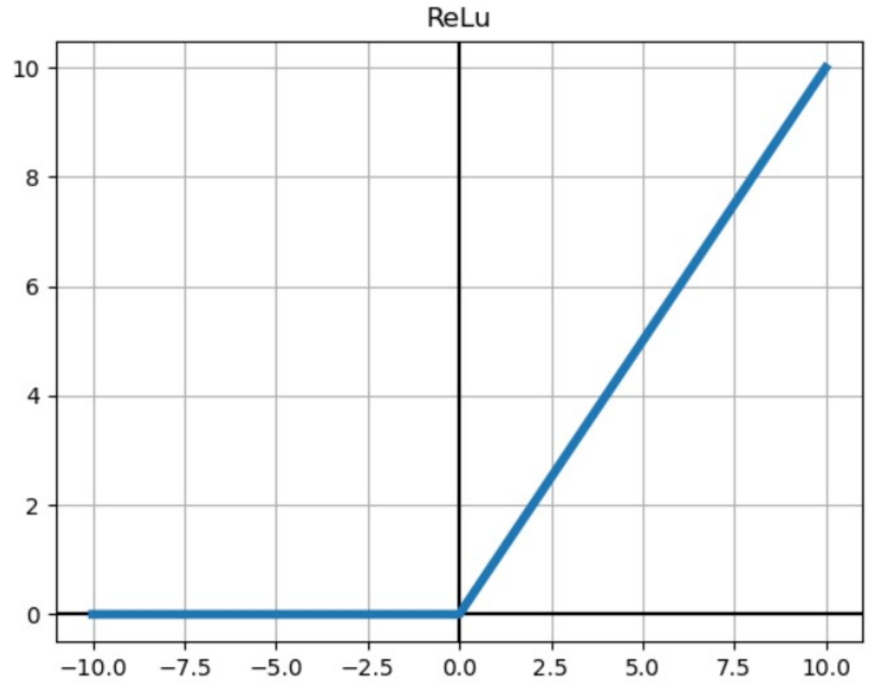


$$\sigma(x) = \frac{1}{1+e^{-x}}$$

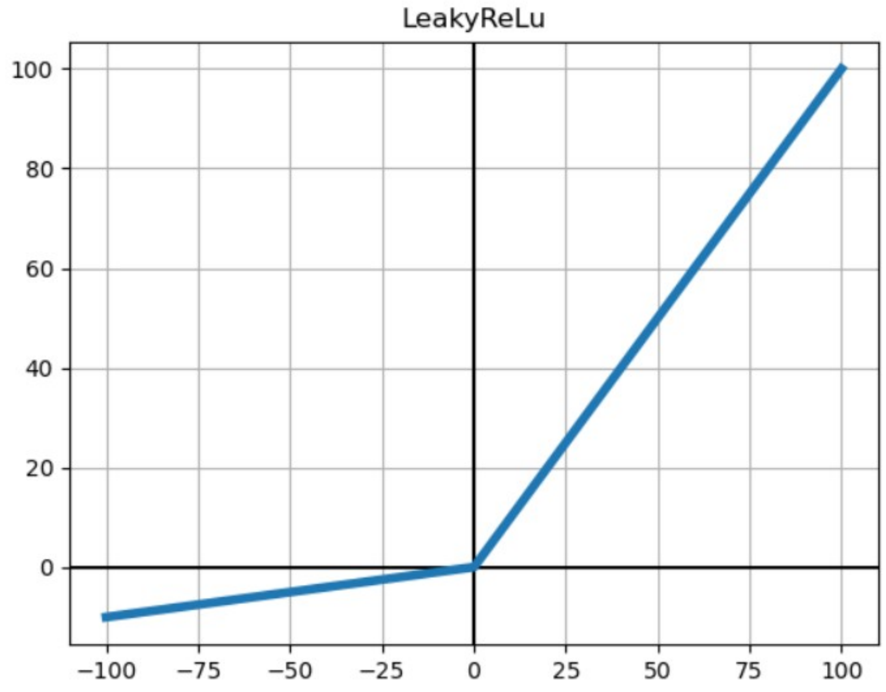


$$\tanh(x)$$

Activation functions



$$\max(0, x)$$



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

Activation functions

- GELU

$$0.5x(1 + \tanh[\sqrt{2/\pi}(x + 0.044715x^3)])$$

- ELU

$$R(z) = \begin{Bmatrix} z & z > 0 \\ \alpha.(e^z - 1) & z <= 0 \end{Bmatrix}$$

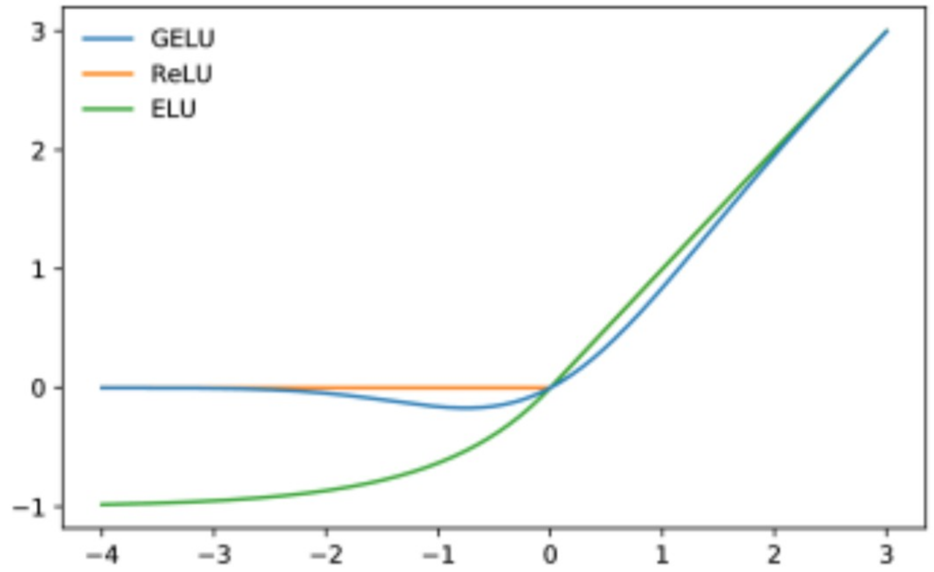
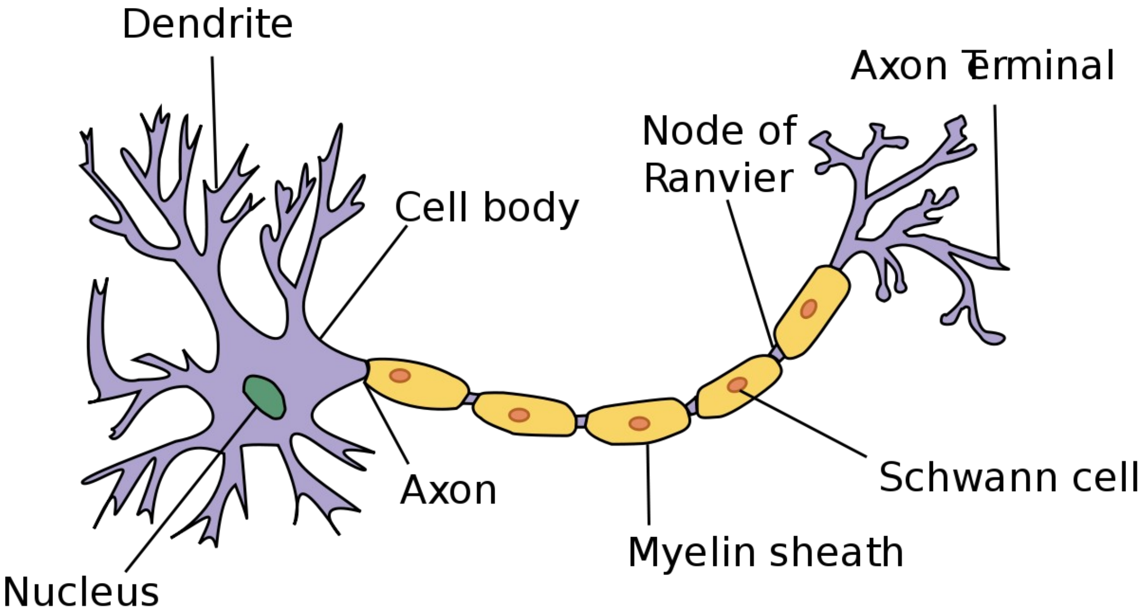
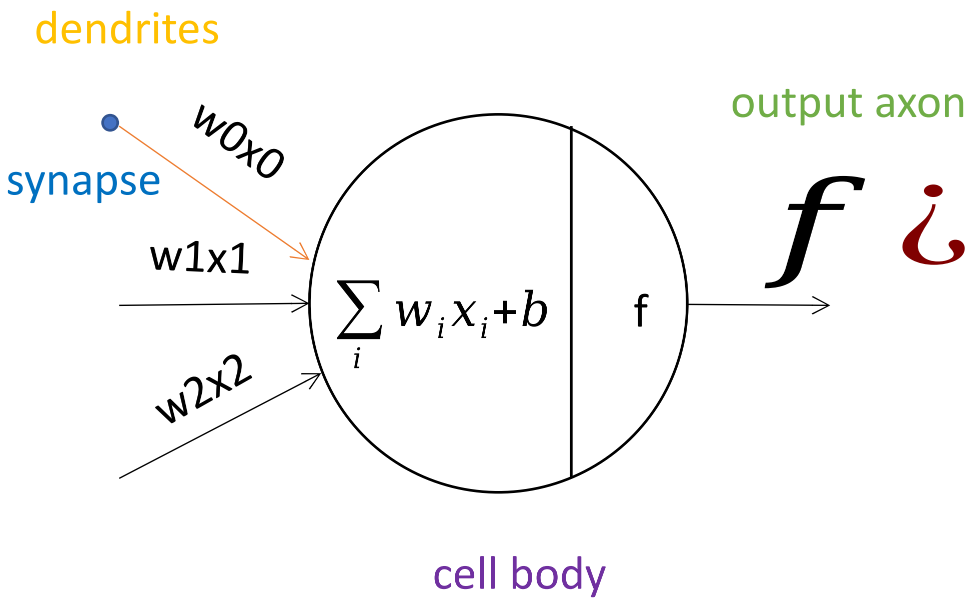


Figure 1: The Gaussian Error Linear Unit ($\mu = 0, \sigma = 1$), the Rectified Linear Unit, and the Exponential Linear Unit ($\alpha = 1$).

Neural networks



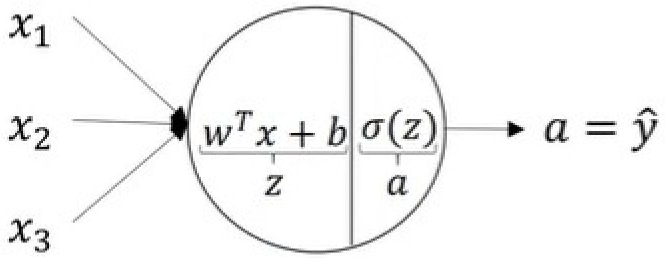


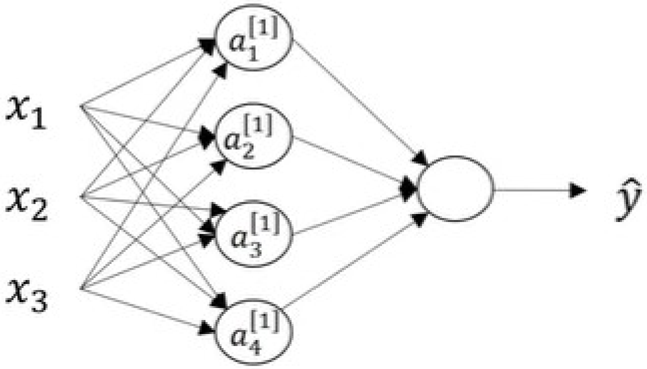
Neural network and the brain

- deeplearning.ai Course 1, Week 4: *What does this have to do with the brain?*
  - *https://www.coursera.org/lecture/neural-networks-deep-learning/what-does-this-have-to-do-with-the-brain-obJnR*

- We should be more reserved about the brain analogies
  - There are actually many different types of biological neurons
  - Dendrites can perform complex non-linear operations
  - Complex connectivity patterns

Neural networks- notation





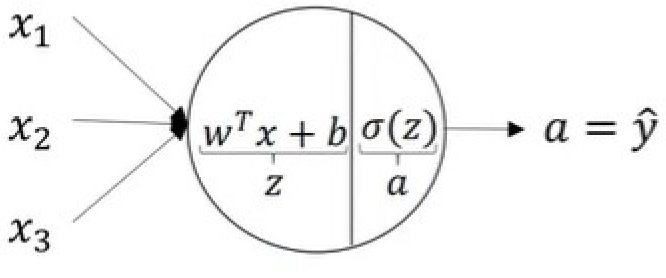
$a^{[l]}$ - the activations of the l^{th} layer
$a^{[l]}_i$ - the activations of i^{th} neuron in the l^{th} layer

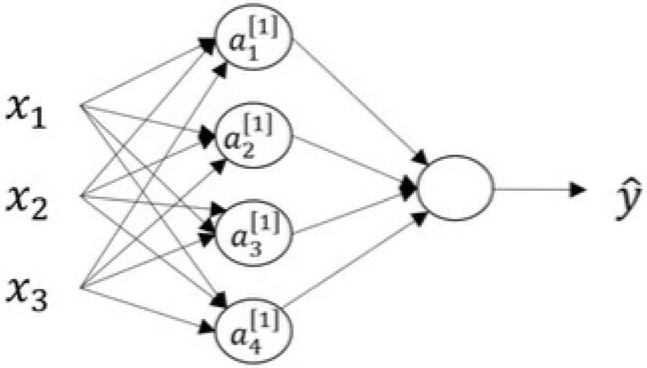
2 layer neural network
We don't count the input layer!

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \; a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \; a_2^{[1]} = \sigma(z_2^{[1]})$$

Neural networks- notation





$a^{[l]}$ - the activations of the l^{th} layer
$a^{[l]}{}_i$ - the activations of i^{th} neuron in the l^{th} layer

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]},\ a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]},\ a_2^{[1]} = \sigma(z_2^{[1]})$$

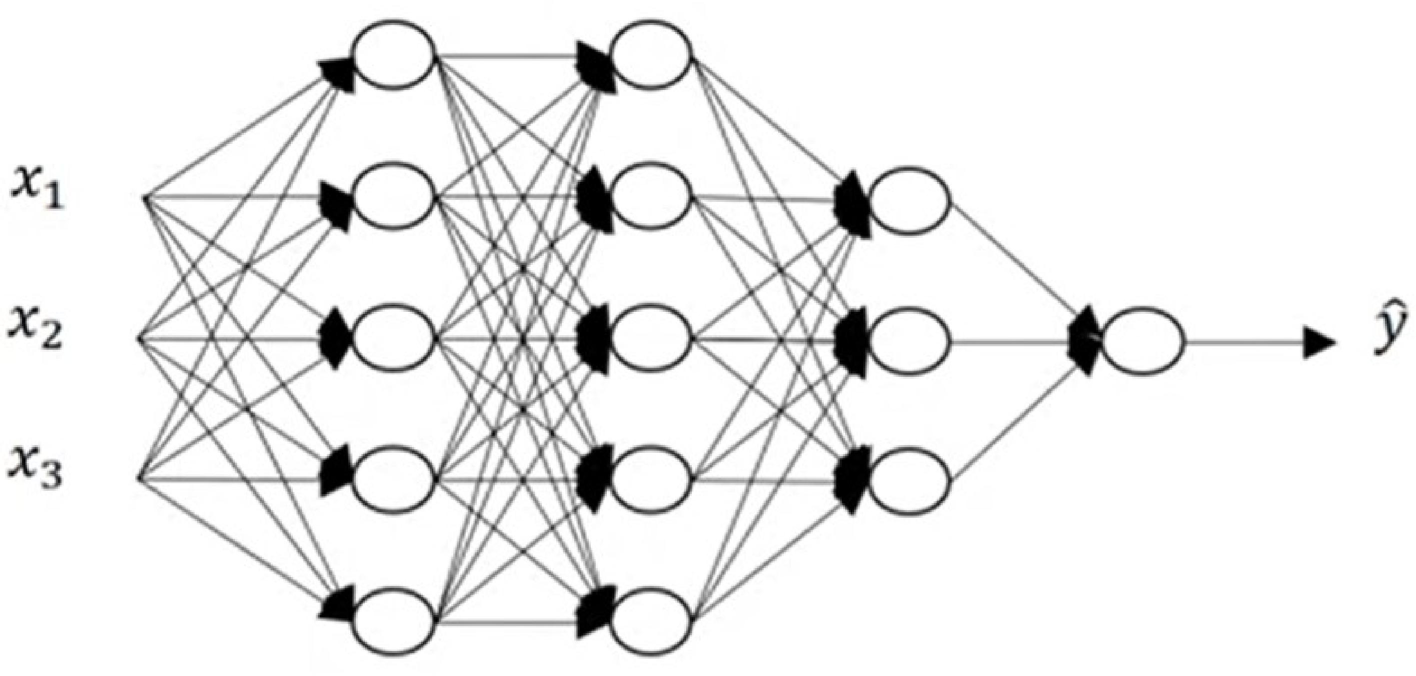
$Z^{[l]} = W^{[l]} \cdot X + b^{[l]}$

2 layer neural network
We don't count the input layer!

X: (3, 1)
$W^{[1]}$: (4, 3)
$b^{[1]}$: (4, 1)
$z^{[1]}$: (4, 1)
$a^{[1]}$: (4, 1)

$a^{[2]}$: (4, 1)
$W^{[2]}$: (1, 4)
$b^{[2]}$: (1, 1)
$z^{[2]}$: (1, 1)
$a^{[2]}$: (1, 1)

Multiple layer hidden network

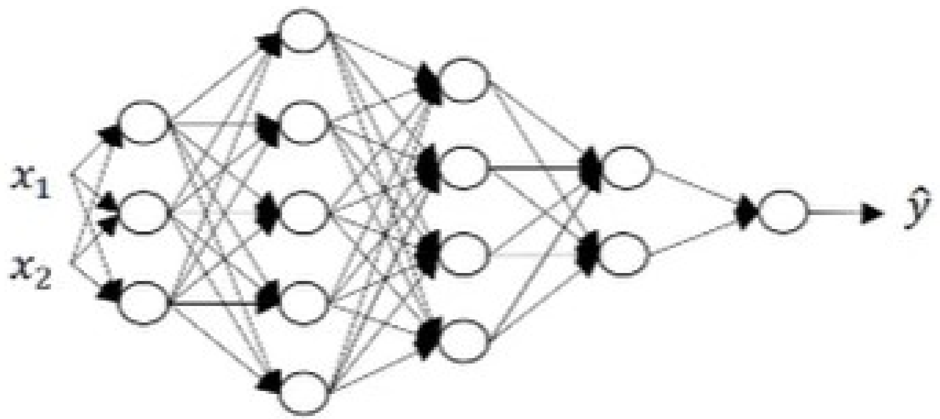


L – number of layers
$n^{[l]}$ – number of hidden layers in each unit
$n^{[0]} = n_x$
$W^{[l]}$ – weight matrix of the l^{th} layer
$b^{[l]}$ – bias vector for the l^{th} layer
$a^{[l]}$ – activations computed by the l^{th} layer

$Z^{[1]} = W^{[1]} \cdot X + b^{[1]}$

$Z^{[2]} = W^{[2]} \cdot a^{[1]} + b^{[2]}$

Multiple layer hidden network
Notation and matrix dimensions



L – number of layers
$n^{[l]}$ – number of hidden layers in each unit
$n^{[0]} = n_x$
$W^{[l]}$ – weight matrix of the l^{th} layer
$b^{[l]}$ – bias vector for the l^{th} layer
$a^{[l]}$ – activations computed by the l^{th} layer

$Z^{[1]} = W^{[1]} \cdot X + b^{[1]}$

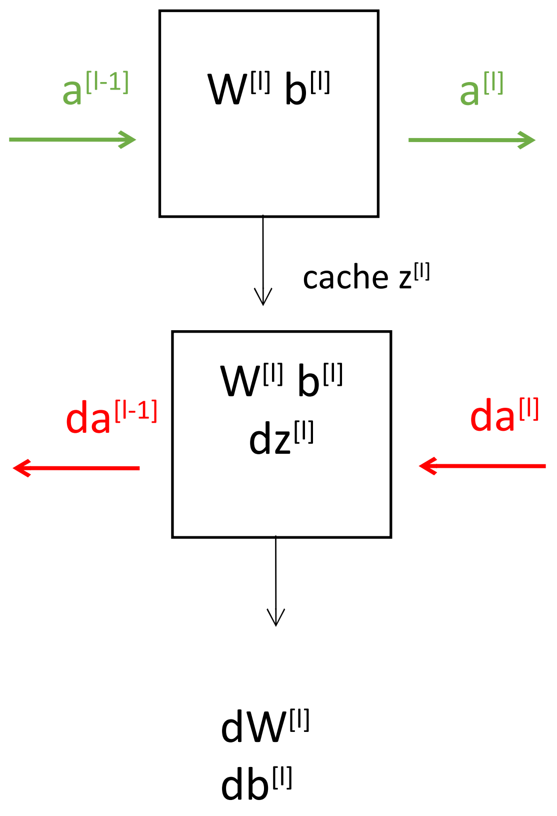
$Z^{[2]} = W^{[2]} \cdot a^{[1]} + b^{[2]}$

Matrix dimensions for layer L:

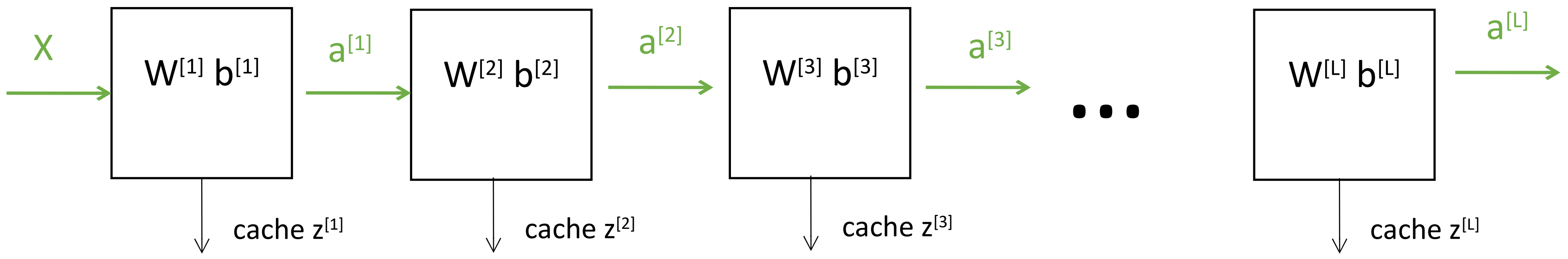
$z^{[l]}$: ($n^{[l]}$, 1)
$a^{[l]}$: ($n^{[l]}$, 1)
$W^{[l]}$: ($n^{[l]}$, $n^{[l-1]}$)
$b^{[l]}$: ($n^{[l]}$,1)
$dW^{[l]}$: ($n^{[l]}$, $n^{[l-1]}$)
$db^{[l]}$: ($n^{[l]}$,1)

Putting it all together

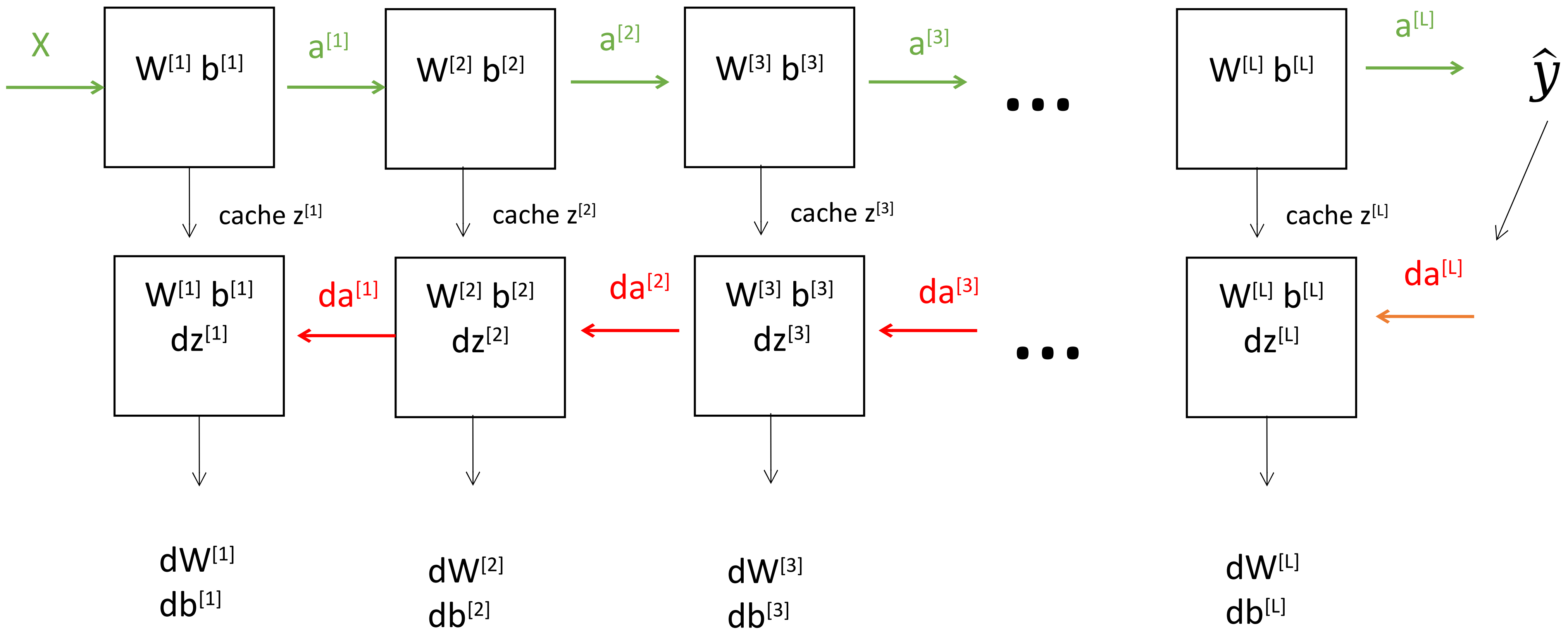
Forward and backward computations at a given layer *l*



Forward and backward functions in NN



Forward and backward functions in NN



Forward propagation for a layer *l*

- Input
  - $a^{[l-1]}$
- Output:
  - $a^{[l]}$
  - Cache: $z^{[l]}$, $W^{[l]}$,$b^{[l]}$

$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$

$a^{[l]} = g^{[l]} (z^{[l]})$

Backward propagation for a layer l

- Input
  - $da^{[l]}$
- Output
  - $da^{[l-1]}$,$dW^{[l]}$, $db^{[l]}$,

$$dz^{[l]} = da^{[l]} * g^{[l]\prime}(z^{[l]})$$

$$dW^{[l]} = dz^{[l]} \cdot a^{[l-1]T}$$

$$db^{[l]} = dz^{[l]}$$

$$da^{[l-1]} = W^{[l]T} \cdot dz^{[l]}$$

https://www.youtube.com/watch?v=yXcQ4B-YSjQ&list=PLkDaE6sCZn6Ec-XTbcX1uRg2_u4xOEky0&index=34

Why do we need deep representation?

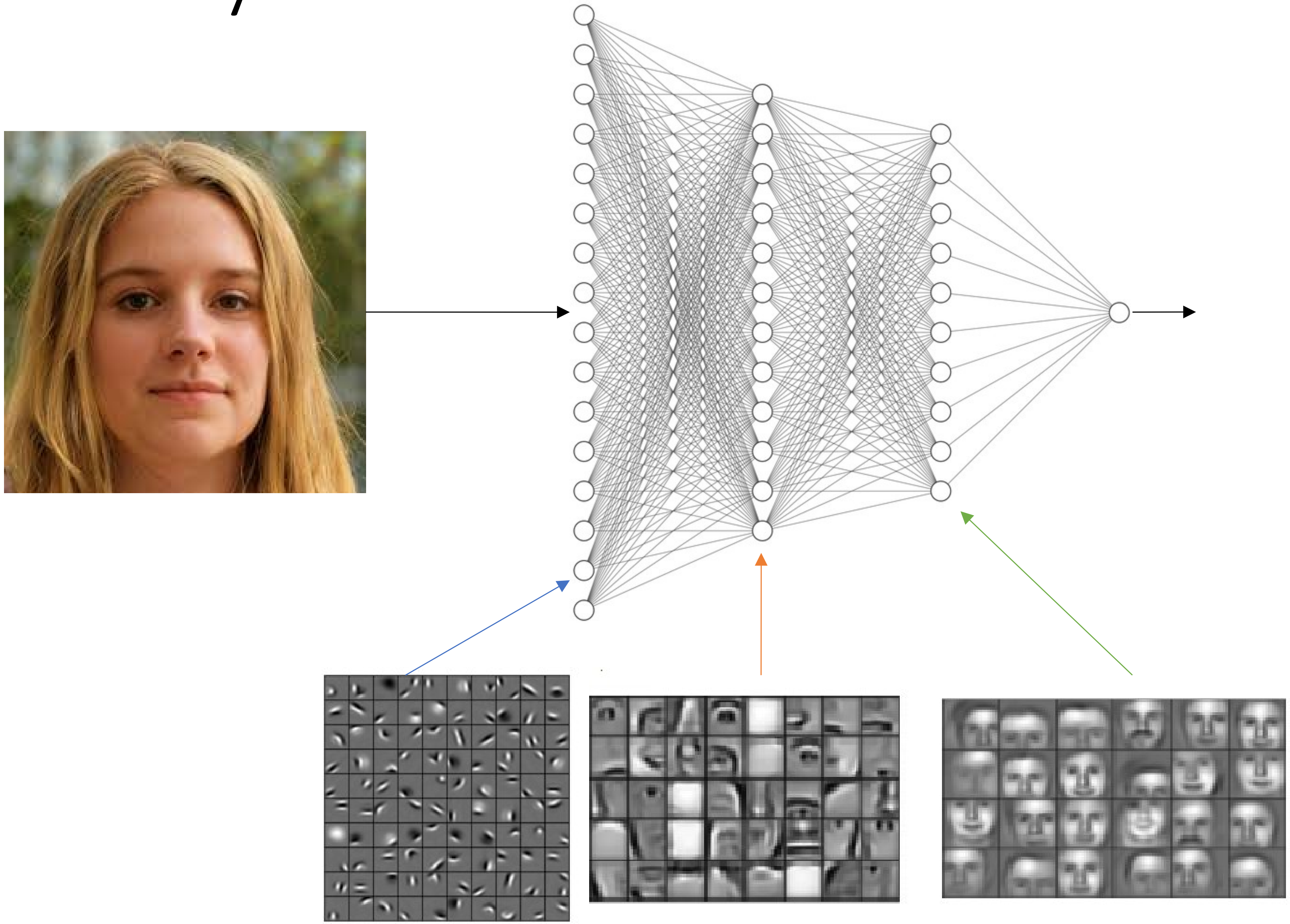
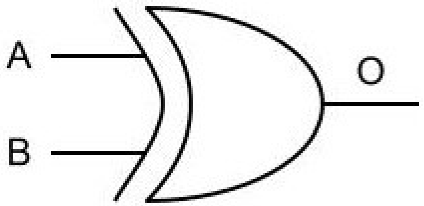


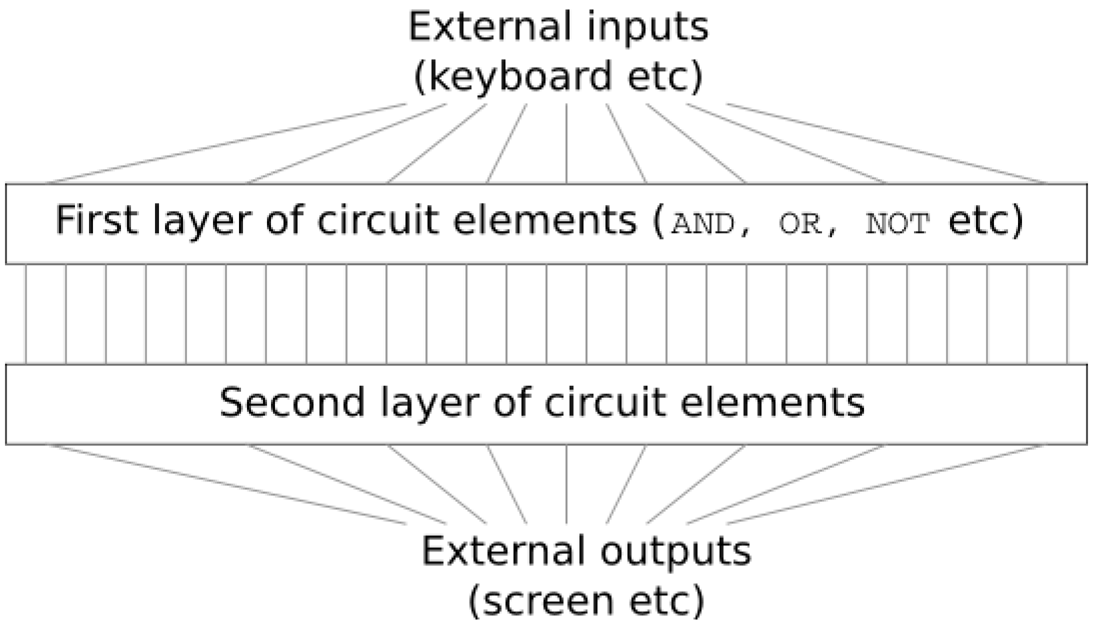
Image source: https://developer.nvidia.com/blog/deep-learning-nutshell-core-concepts/#activation-function

Why do we need deep representation?

- Compute logical functions :
  - AND, OR, NOT etc.
  - XOR
    - requires log(n) layers (where n is the number of inputs)
    - If you were to use a single layer, it would require exponentially neurons in that layer 2^{n-1}



A	B	O
0	0	0
0	1	1
1	0	1
1	1	0



Weights initialization

- Option 1
  - Initialize all weights to 0

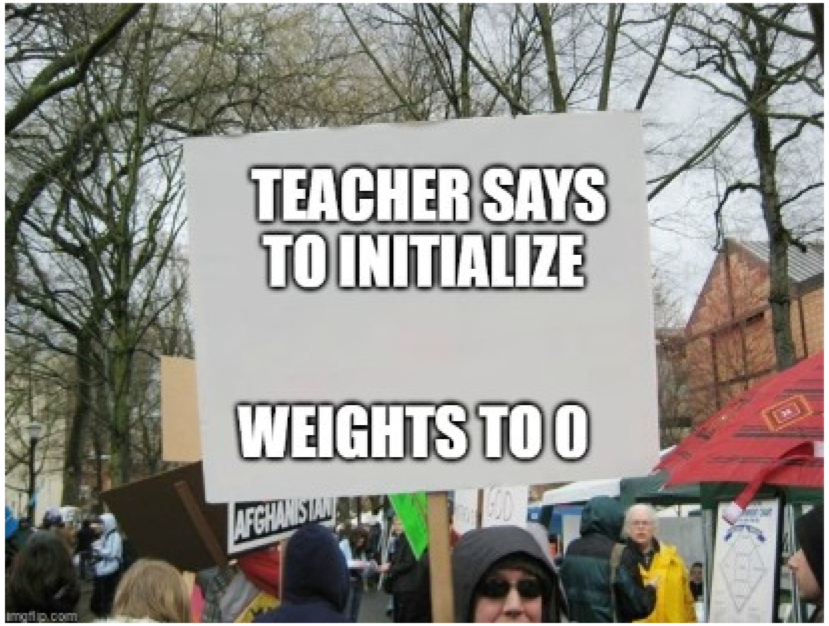


Image credit: A.S., UBB student 2020

Weights initialization

- Option 1
  - ~~Initialize all weights to 0~~
  - All neurons will compute the same output (and implicitly they will have the same gradients during backprop → same parameter output)
  - No source of asymmetry
  - Bad idea!

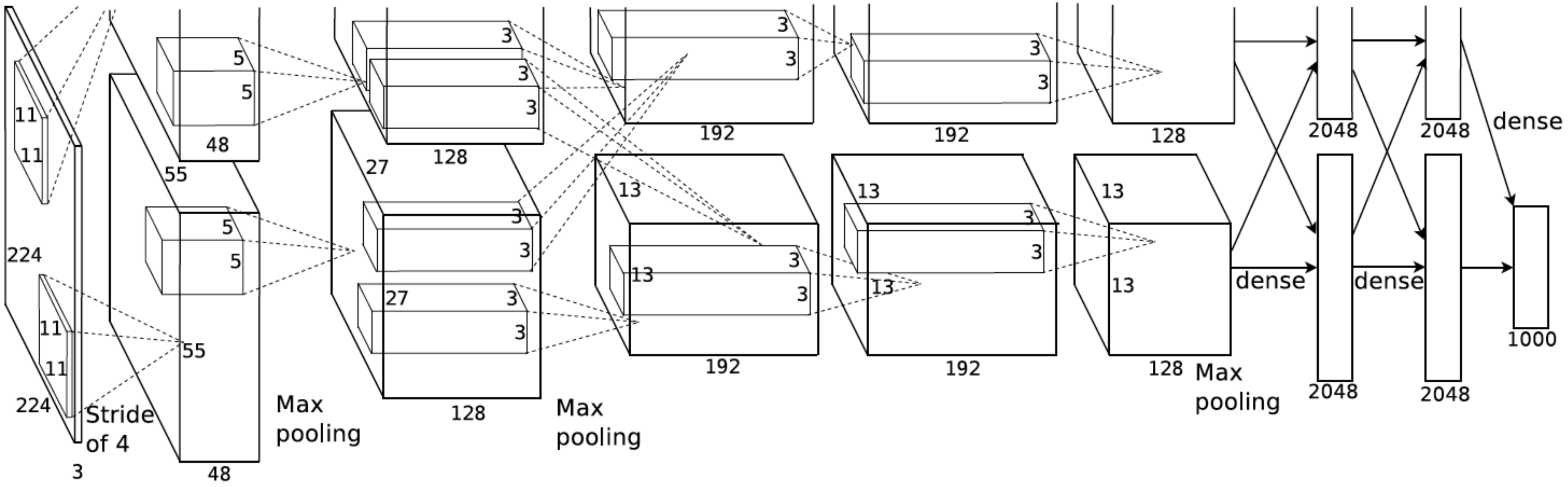
Weights initialization

- ~~Option 1: Initialize all weights to 0~~
- Option 2: Small random numbers
  - *Symmetry breaking*
  - Sample from a uniform distribution: 0.01*np.random.rand(D, H)
  - Sample from a normal distribution: 0.01*np.random.randn(D, H)

Weights initialization

- ~~Option 1: Initialize all weights to 0~~
- Option 2: Small random numbers
  - S*ymmetry breaking*
  - Sample from a uniform distribution: 0.01*np.random.rand(D, H)
  - Sample from a normal distribution: 0.01*np.random.randn(D, H)
- Option 3: Sparse initialization
  - initialize weights to 0, but break symmetry: every neuron is randomly connected to a fixed number of neurons below it (with weights randomly sampled)

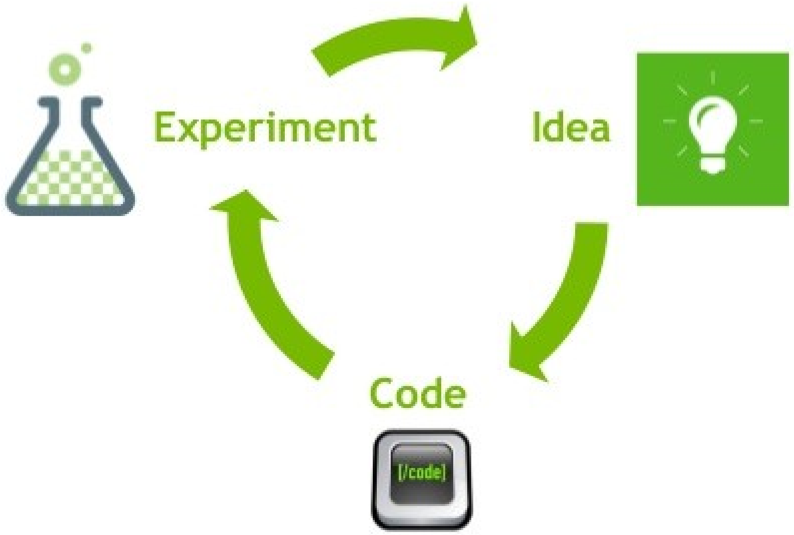
image → Pre-processing → Feature extraction → Classification →



Parameters vs Hyperparameters

- Parameters: W, b
- Hyper-parameters: control/ determine the values of the parameters we compute
  - Learning rate (for gradient descent)
  - Number of iterations for gradient descent
  - Number of hidden layers L
  - Number of hidden units: $n^{[1]}$, $n^{[2]}$
  - Activation functions: ReLu, Leaky ReLu, tanh, ~~sigmoid~~
- Deep learning is an empirical and iterative process
  - Idea
  - Code
  - Experiment

  REPEAT



Data Preprocessing

- Mean subtraction
  - subtracting the mean across every individual *feature* in the data, and has the geometric interpretation of centering the cloud of data around the origin along every dimension
- Normalization
  - normalizing the data so that they are of approximately the same scale
  - divide each dimension by its standard deviation, once it has been zero-centered
  - normalize each dimension so that the min and max along the dimension is -1 and 1 respectively

!!Any pre-processing statistics must only be computed on the training data, and then applied to the validation / test data !!

Playground

https://playground.tensorflow.org/

Convolutions

Image convolutions

$$g(i,j)=\sum_{m=1}^{M}\sum_{n=1}^{N} f(m,n)\,h(i-m,j-n)$$

10	20	30	15	12
11	200	34	23	45
32	35	255	255	10
10	33	6	7	59
13	43	13	45	3

-1	-1	-1
0	0	0
1	1	1

Image convolutions

$$g(i,j)=\sum_{m=1}^{M}\sum_{n=1}^{N} f(m,n)\,h(i-m,j-n)$$

10	20	30	15	12
11	200	34	23	45
32	35	255	255	10
10	33	6	7	59
13	43	13	45	3

-1	-1	-1
0	0	0
1	1	1

262		

-10 – 20 – 30 + 32 + 35 + 255 = 262

Image convolutions

$$g(i,j)=\sum_{m=1}^{M}\sum_{n=1}^{N} f(m,n)h(i-m,j-n)$$

10	20	30	15	12
11	200	34	23	45
32	35	255	255	10
10	33	6	7	59
13	43	13	45	3

-1	-1	-1
0	0	0
1	1	1

262	480	

-20-30-15+35+255+255 = 480

Convolutional filters

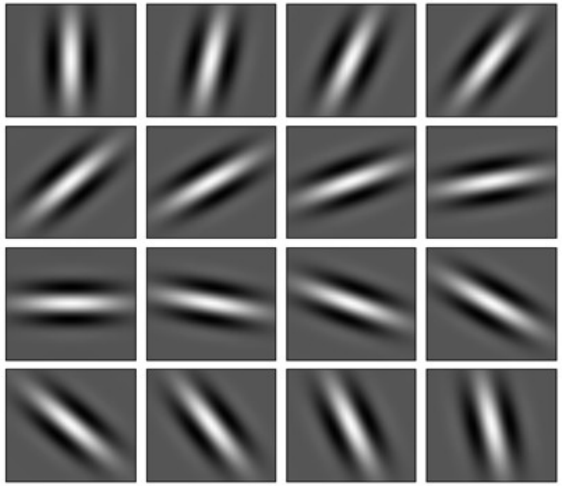
What do you think is the effect of the following filters?

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad \frac{1}{273} \begin{array}{|c|c|c|c|c|} \hline 1 & 4 & 7 & 4 & 1 \\ \hline 4 & 16 & 26 & 16 & 4 \\ \hline 7 & 26 & 41 & 26 & 7 \\ \hline 4 & 16 & 26 & 16 & 4 \\ \hline 1 & 4 & 7 & 4 & 1 \\ \hline \end{array}$$

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \qquad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Kernels learned from AlexNet first convolutional layers



Gabor filters

What did we learn today?

- Backpropagation
  - Forward pass, backward pass
- Artificial neural networks
  - Input, output, hidden layers
  - Activation functions
- Image convolution (final step towards CNN)

Recommended resources

- Course 1: deeplearning.ai Neural networks fundamentals
  - week 1, 2, 3, 4
- http://cs231n.stanford.edu/
- https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b
- https://developer.nvidia.com/blog/deep-learning-nutshell-core-concepts/#activation-function