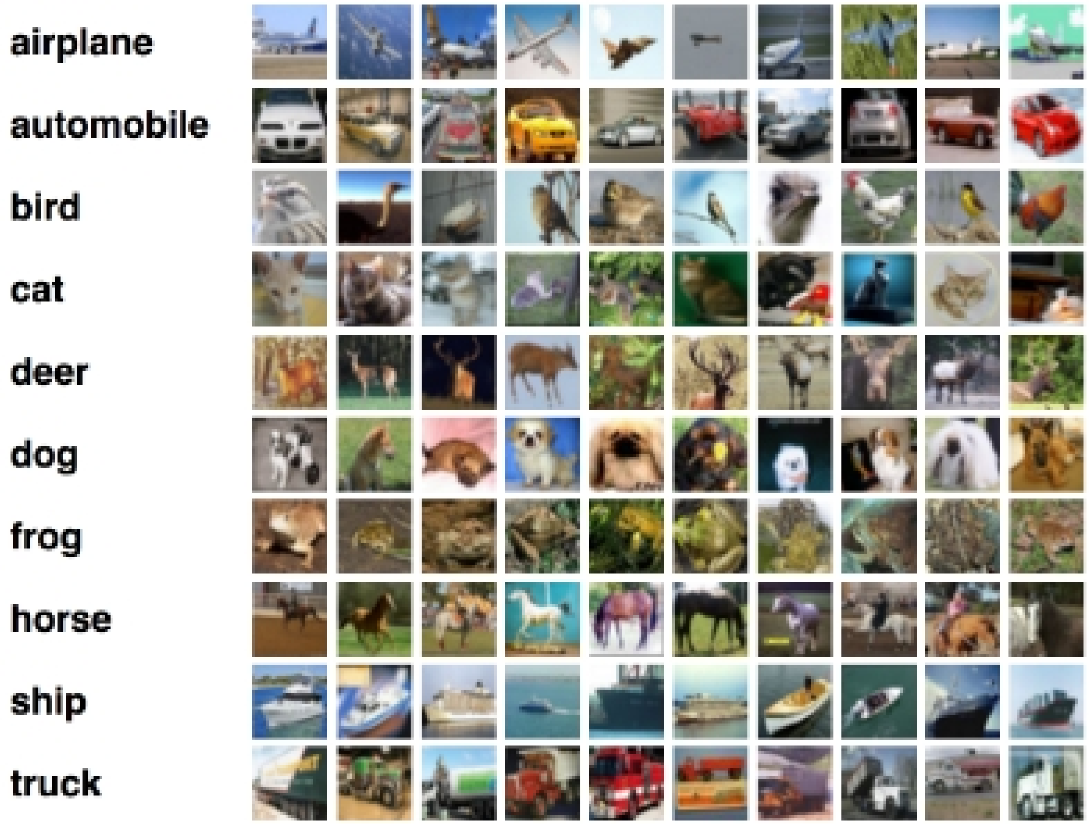
Introduction to PyTorch

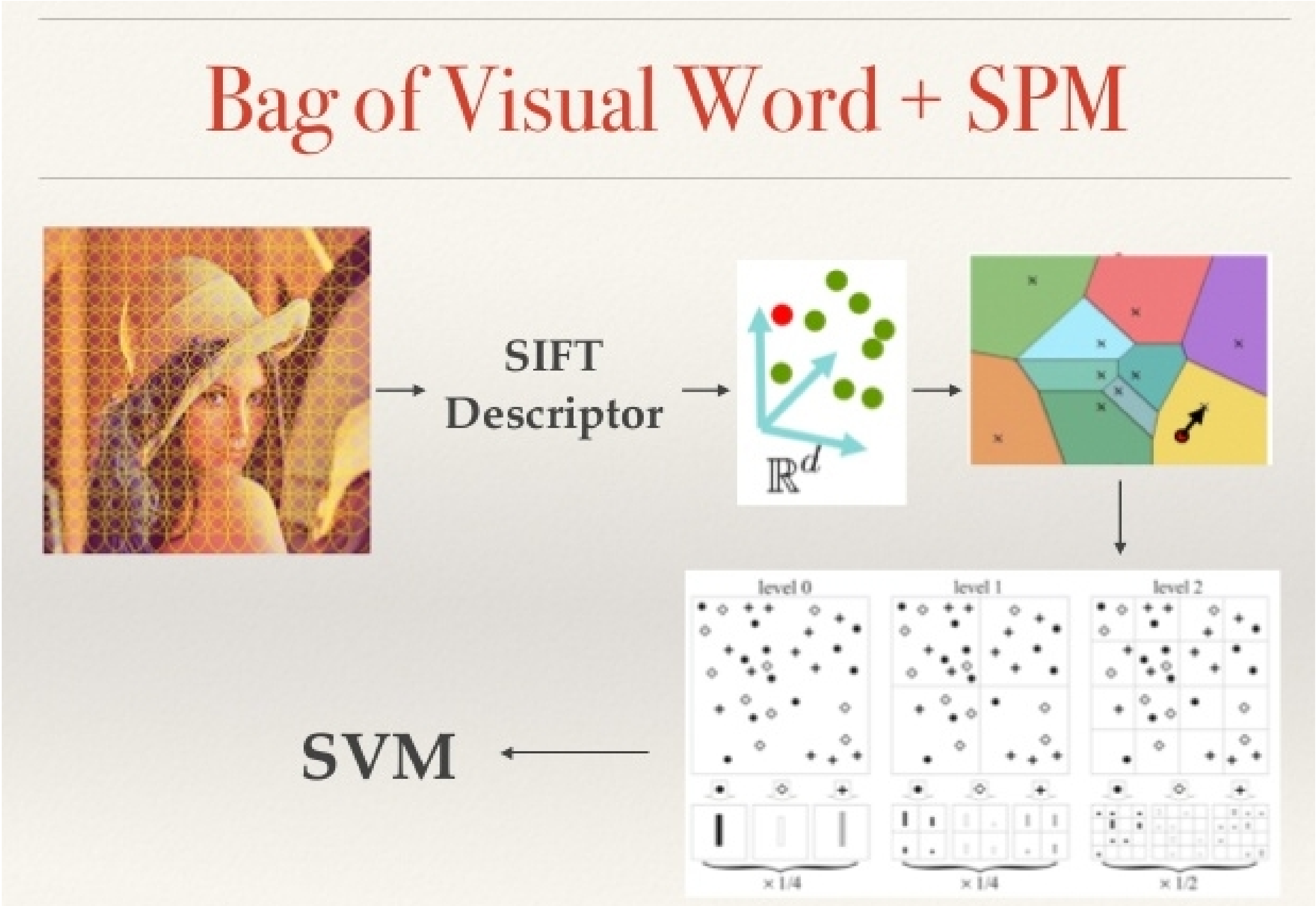
INTRODUCTION TO DEEP LEARNING WITH PYTORCH

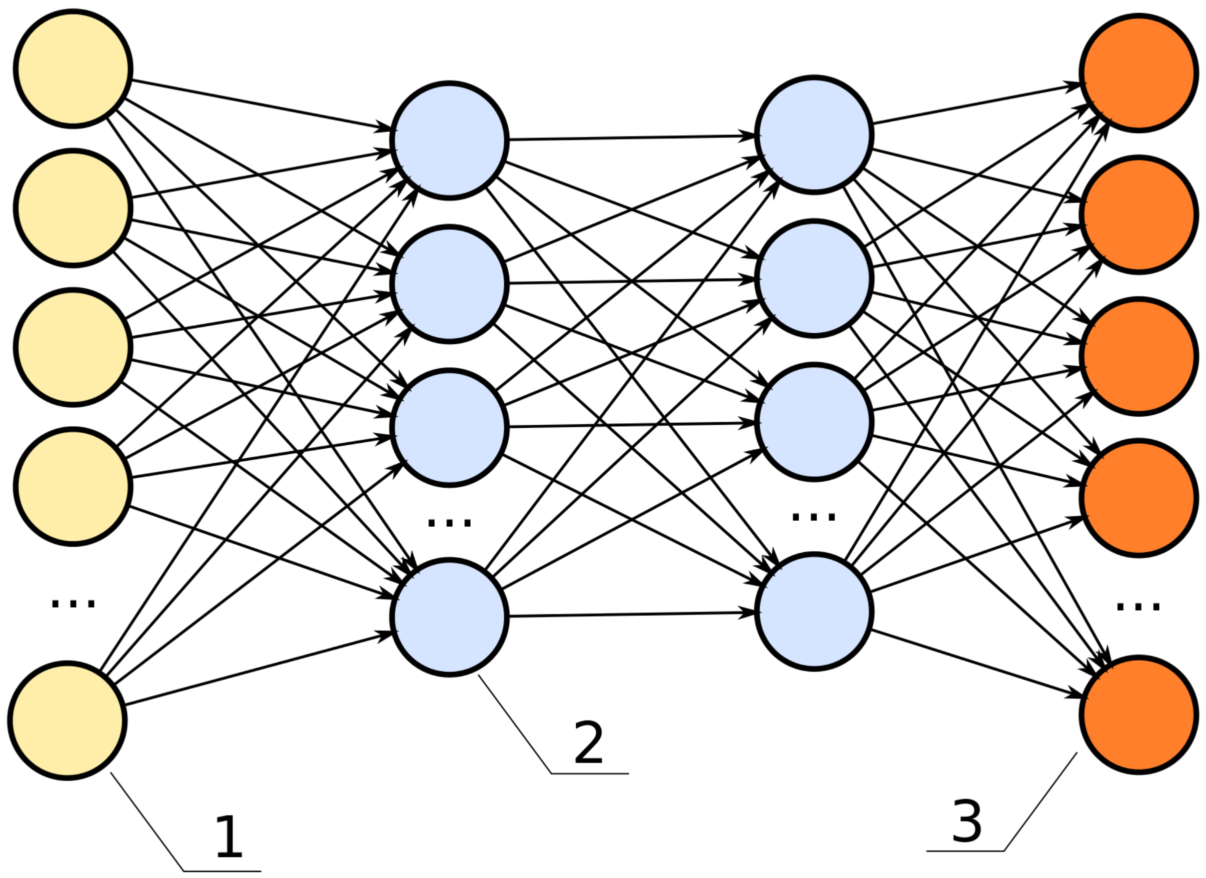
Ismail Elezi
Ph.D. Student of Deep Learning



STARCRAFT II
LEGACY OF THE VOID

Neural networks

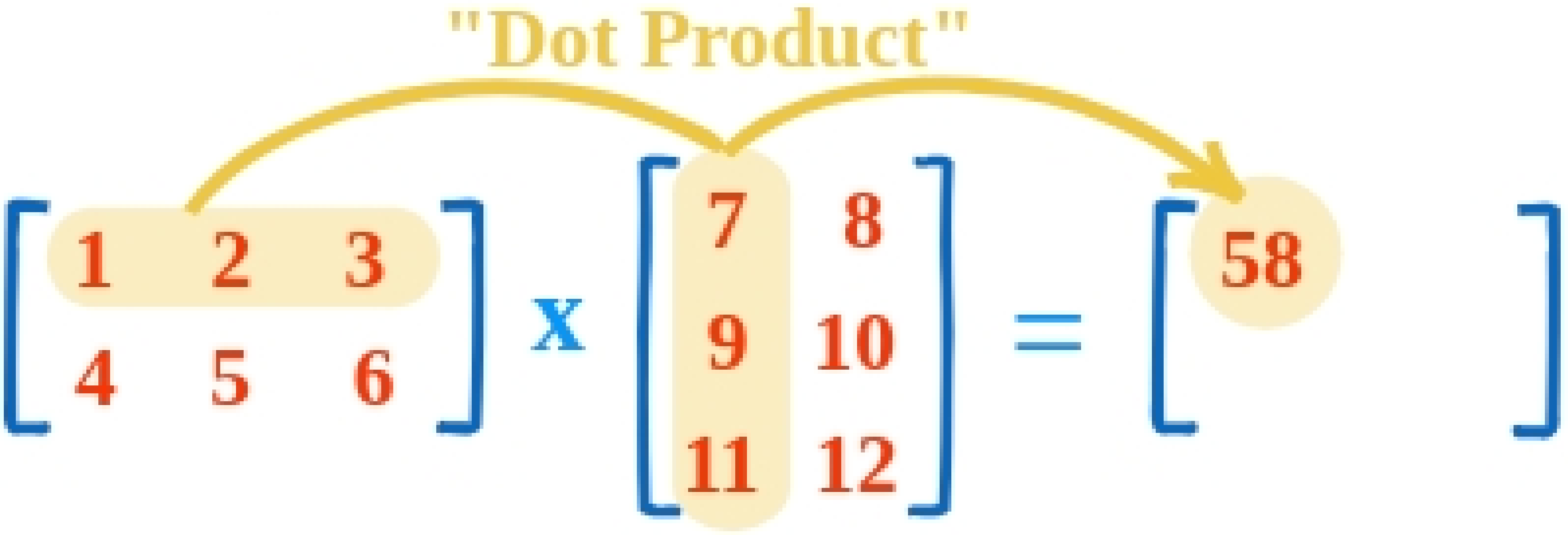




Why PyTorch?

- "PyThonic" - easy to use
- Strong GPU support - models run fast
- Many algorithms are already implemented
- Automatic differentiation - more in next lesson
- Similar to NumPy

Matrix Multiplication



PyTorch compared to NumPy

```python
import torch
torch.tensor([[2, 3, 5], [1, 2, 9]])
```

```
tensor([[ 2,  3,  5],
        [ 1,  2,  9]])
```

```python
torch.rand(2, 2)
```

```
tensor([[ 0.0374, -0.0936],
        [ 0.3135, -0.6961]])
```

```python
a = torch.rand((3, 5))
a.shape
```

```
torch.Size([3, 5])
```

```python
import numpy as np
np.array([[2, 3, 5], [1, 2, 9]])
```

```
array([[ 2,  3,  5],
       [ 1,  2,  9]])
```

```python
np.random.rand(2, 2)
```

```
array([[ 0.0374, -0.0936],
       [ 0.3135, -0.6961]])
```

```python
a = np.random.randn(3, 5)
a.shape
```

```
(3, 5)
```

Matrix operations

```
a = torch.rand((2, 2))
b = torch.rand((2, 2))
```

```
tensor([[-0.6110,  0.0145],
        [ 1.3583, -0.0921]])
tensor([[ 0.0673,  0.6419],
        [-0.0734,  0.3283]])
```

```
torch.matmul(a, b)
```

```
tensor([[-0.0422, -0.3875],
        [ 0.0981,  0.8417]])
```

```
a = np.random.rand(2, 2)
b = np.random.rand(2, 2)
```

```
array([[-0.6110,  0.0145],
        [ 1.3583, -0.0921]])
array([[ 0.0673,  0.6419],
        [-0.0734,  0.3283]])
```

```
np.dot(a, b)
```

```
array([[-0.0422, -0.3875],
       [ 0.0981,  0.8417]])
```

Matrix operations

```
a * b
```

```
tensor([[-0.0411,  0.0093],
        [-0.0998, -0.0302]])
```

```
np.multiply(a, b)
```

```
array([[-0.0411,  0.0093],
       [-0.0998, -0.0302]])
```

Zeros and Ones

```
a_torch = torch.zeros(2, 2)
```

```
tensor([[0., 0.],
        [0., 0.])
```

```
a_numpy = np.zeros((2, 2))
```

```
array([[0., 0.],
       [0., 0.]])
```

```
b_torch = torch.ones(2, 2)
```

```
tensor([[1., 1.],
        [1., 1.])
```

```
b_numpy = np.ones((2, 2))
```

```
array([[1., 1.],
       [1., 1.]])
```

```
c_torch = torch.eye(2)
```

```
tensor([[1., 0.],
        [0., 1.]
```

```
c_numpy = np.identity(2)
```

```
array([[1., 0.],
       [0., 1.]])
```

PyTorch to NumPy and vice versa

```
d_torch = torch.from_numpy(c_numpy)
```

```
tensor([[1., 0.],
        [0., 1.],
        dtype=torch.float64)
```

```
d = c_torch.numpy()
```

```
array([[1., 0.],
       [0., 1.]])
```

Summary

```
torch.matmul(a, b)    # multiples torch tensors a and b

*                     # element-wise multiplication between two torch tensors

torch.eye(n)          # creates an identity torch tensor with shape (n, n)

torch.zeros(n, m)     # creates a torch tensor of zeros with shape (n, m)

torch.ones(n, m)      # creates a torch tensor of ones with shape (n, m)

torch.rand(n, m)      # creates a random torch tensor with shape (n, m)

torch.tensor(l)       # creates a torch tensor based on list l
```

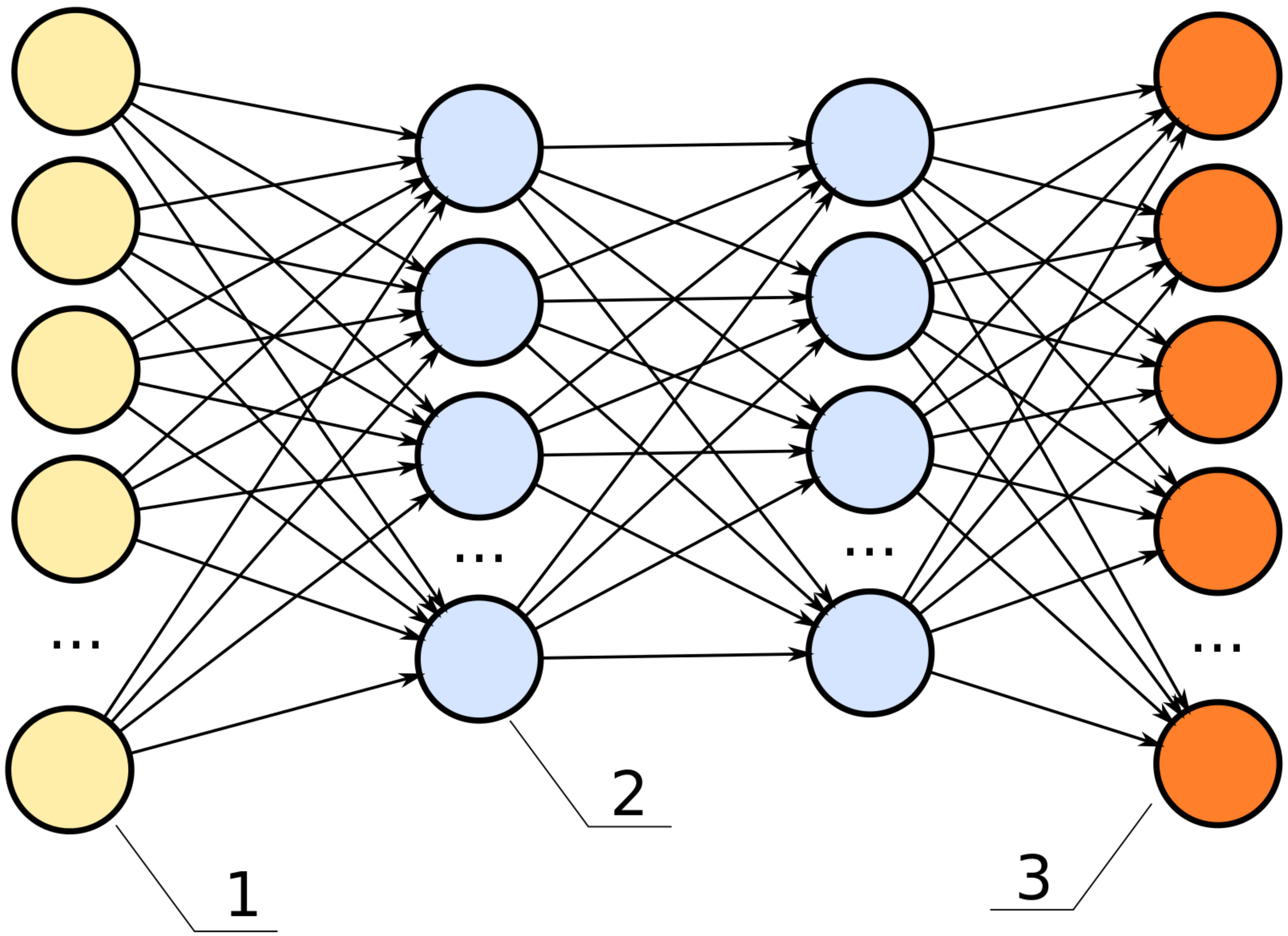
Let's practice

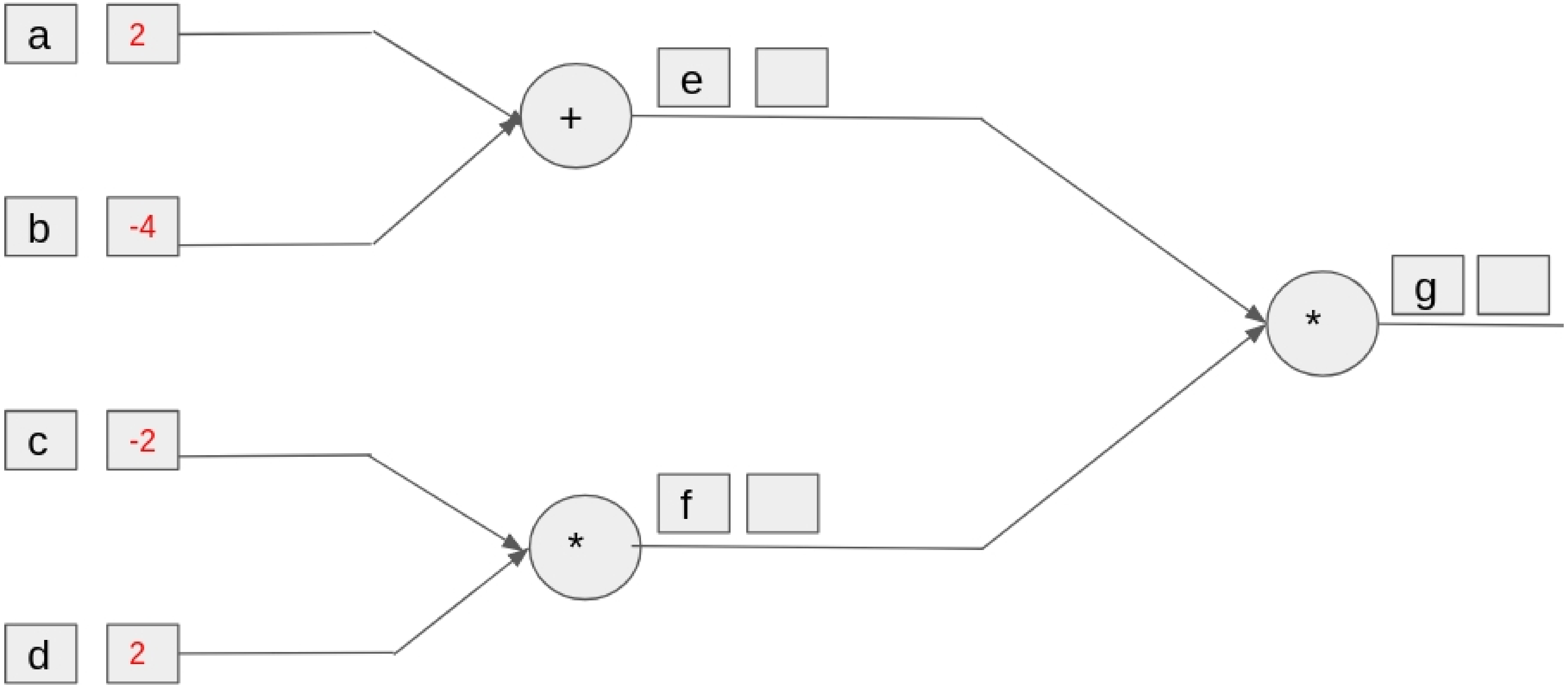
INTRODUCTION TO DEEP LEARNING WITH PYTORCH

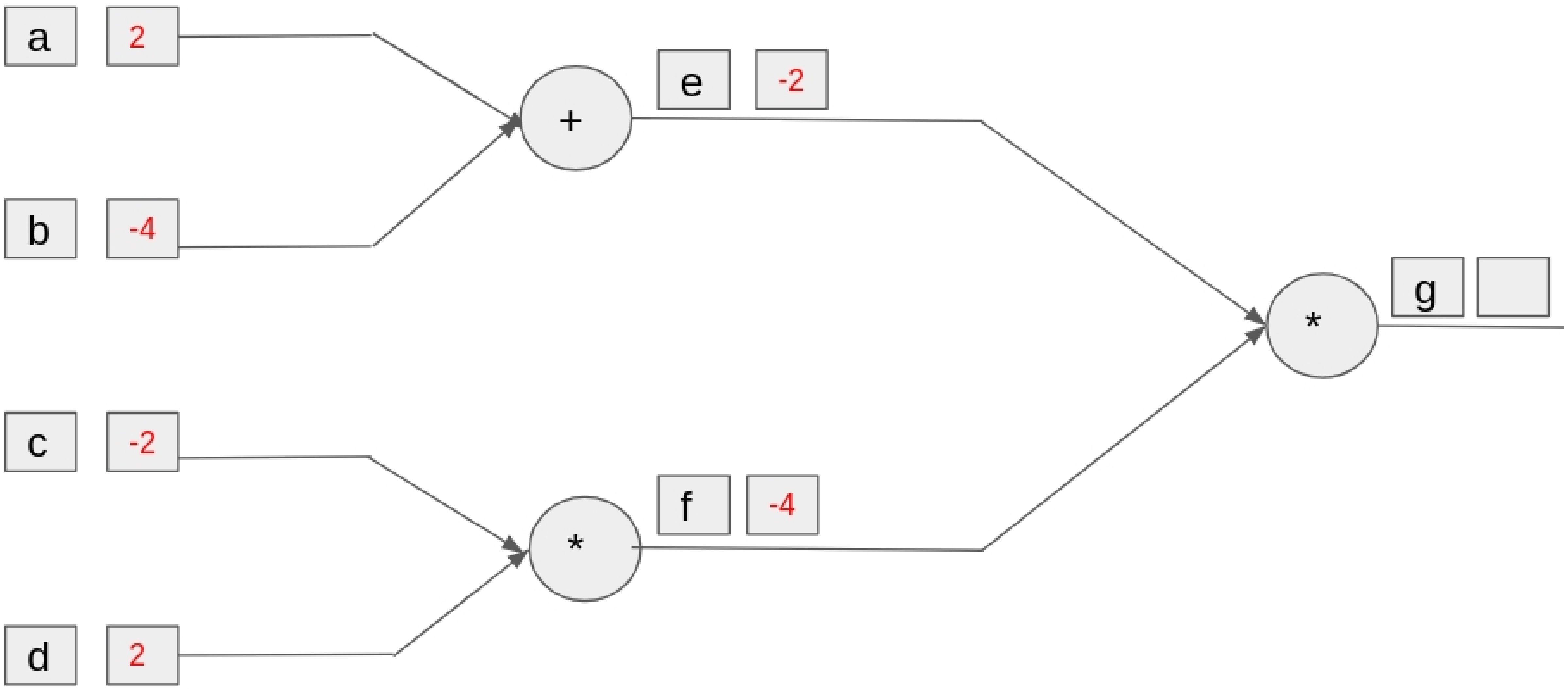
Forward propagation

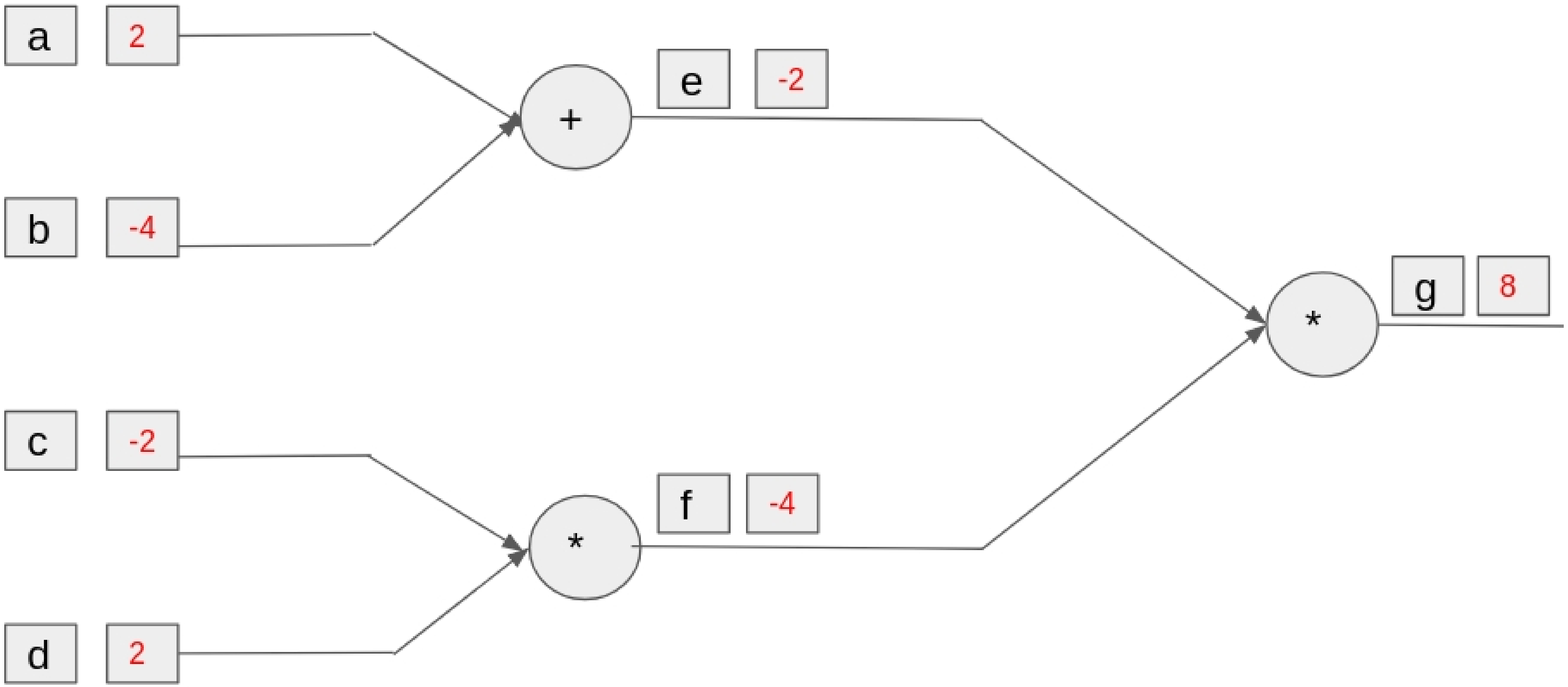
INTRODUCTION TO DEEP LEARNING WITH PYTORCH

Ismail Elezi
Ph.D. Student of Deep Learning


...
...
...
...
1
2
3


a
2
b
-4
c
-2
d
2
+
e
*
f
*
g


a
2
b
-4
c
-2
d
2
+
e
-2
*
f
-4
*
g


a
2
b
-4
+
e
-2
c
-2
d
2
*
f
-4
*
g
8

PyTorch implementation

```python
import torch

a = torch.Tensor([2])
b = torch.Tensor([-4])
c = torch.Tensor([-2])
d = torch.Tensor([2])
```

```python
e = a + b
f = c * d
```

```python
g = e * f
print(e, f, g)
```

```
tensor([-2.]), tensor([-4.]), tensor([8.])
```

Let's practice!

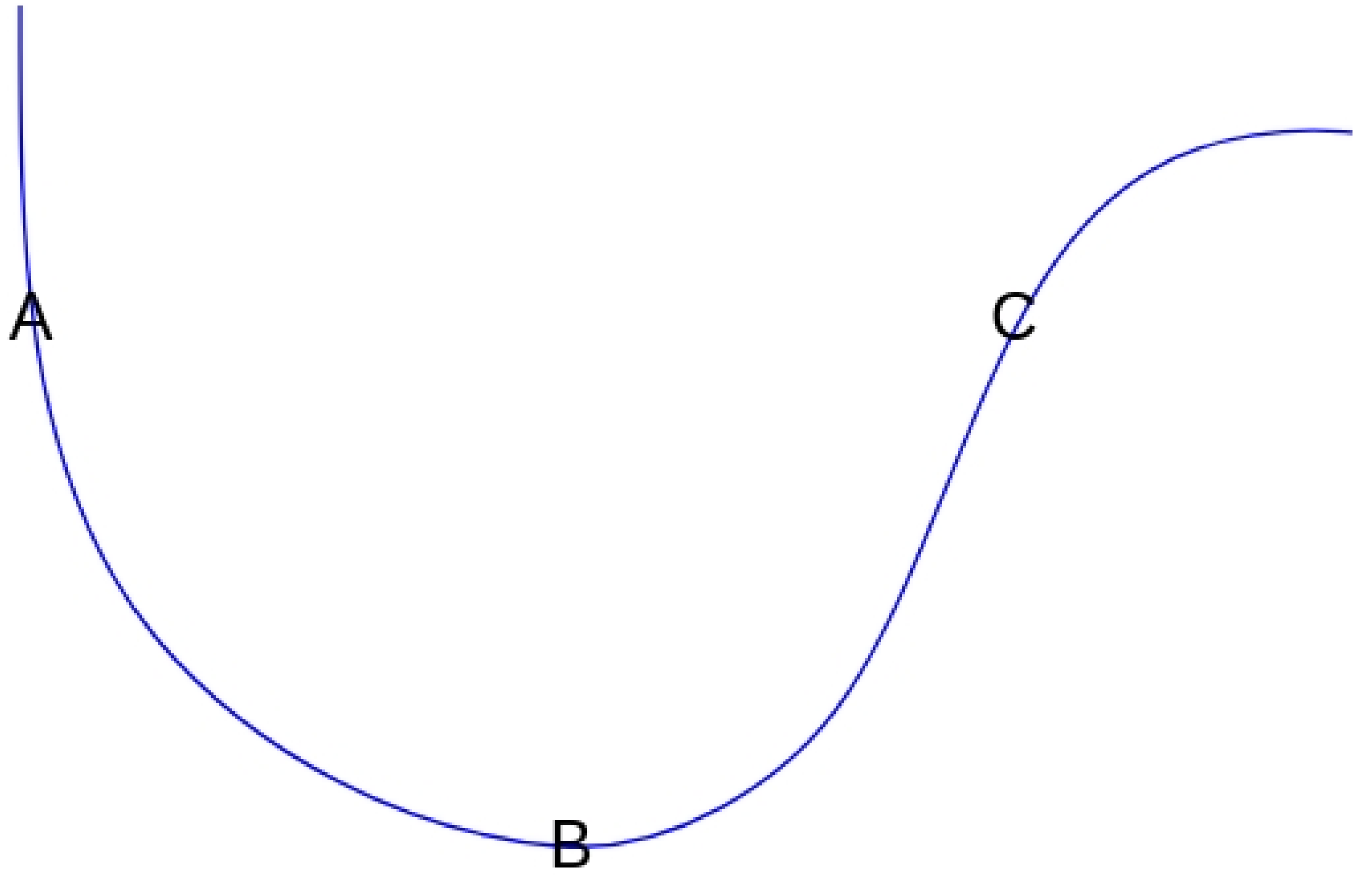
INTRODUCTION TO DEEP LEARNING WITH PYTORCH

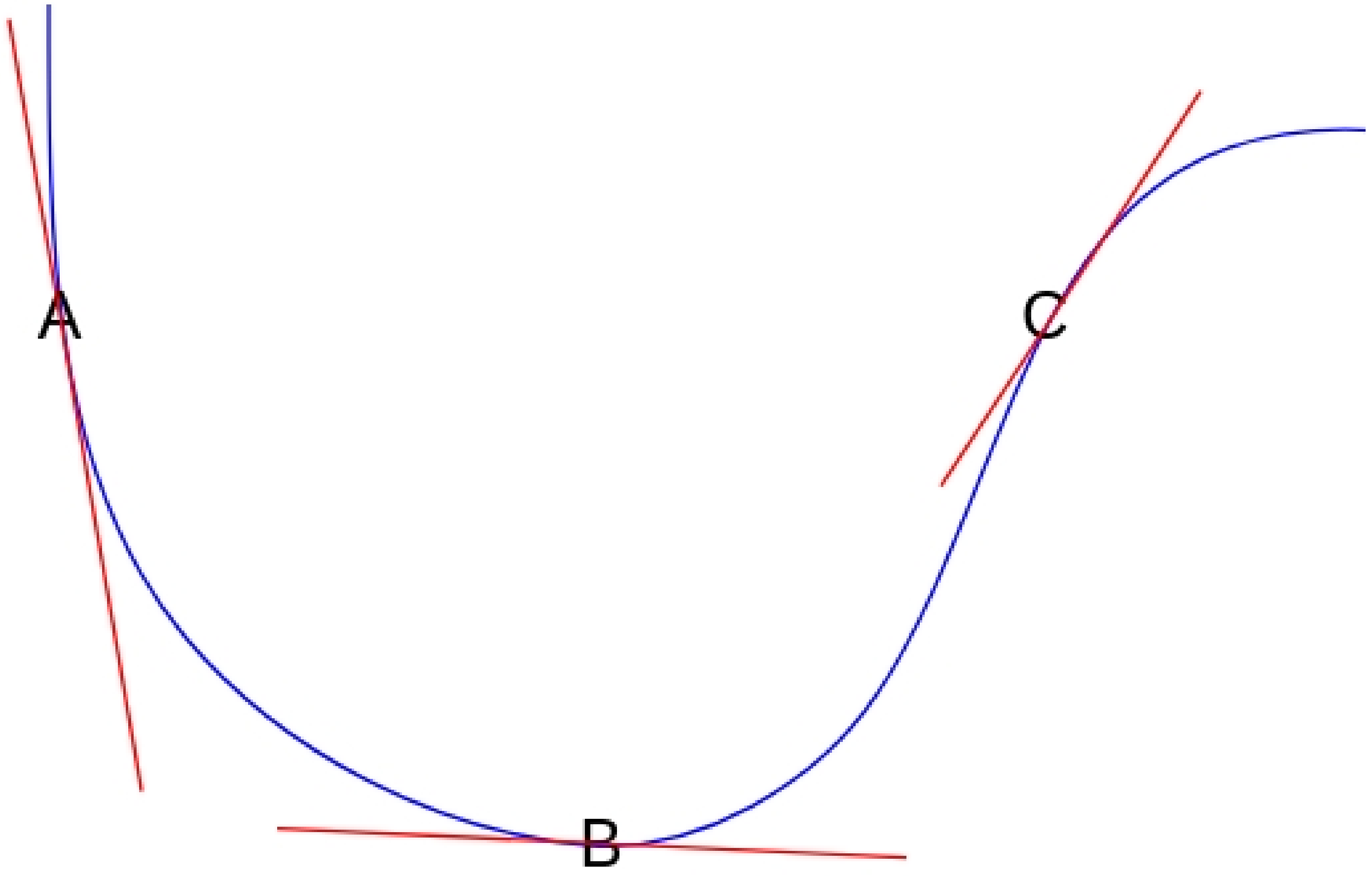
Backpropagation by auto-differentiation

INTRODUCTION TO DEEP LEARNING WITH PYTORCH

Ismail Elezi
Ph.D. Student of Deep Learning

Derivatives





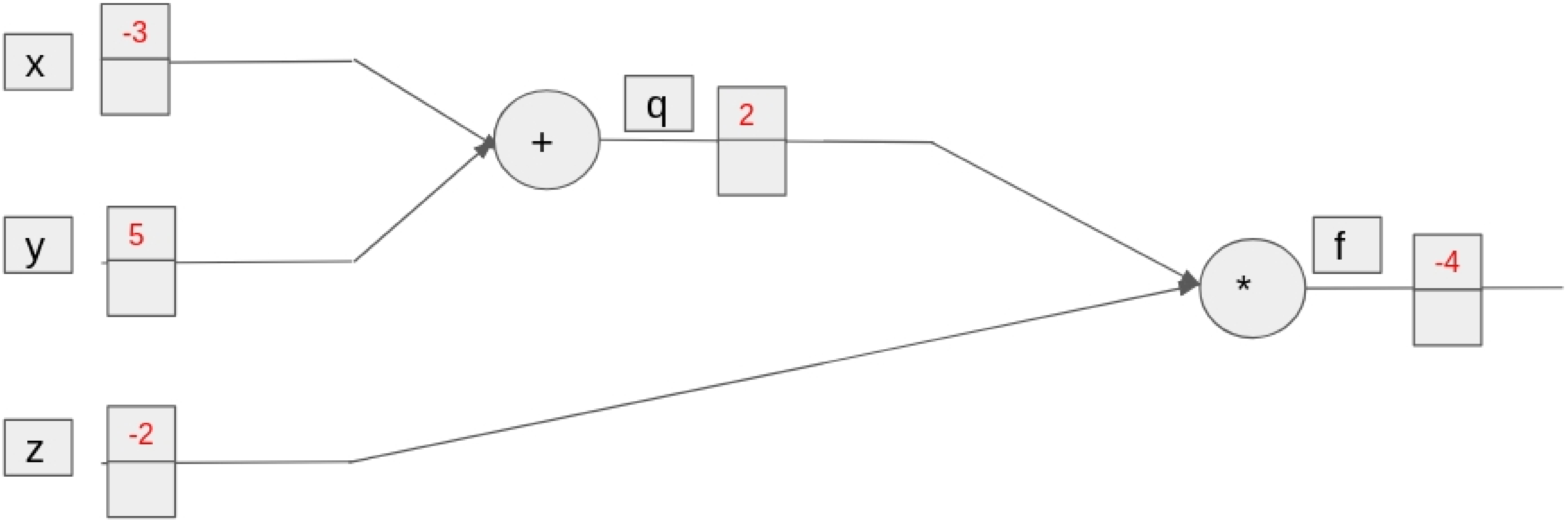
Derivative Rules

Interaction	Overall Change
Addition	$(f+g)' = f' + g'$
Multiplication	$(f \cdot g)' = f \cdot dg + g \cdot df$
Powers	$(x^n)' = \frac{d}{dx}x^n = nx^{n-1}$
Inverse	$\left(\frac{1}{x}\right)' = -\frac{1}{x^2}$
Division	$\left(\frac{f}{g}\right)' = \left(df \cdot \frac{1}{g}\right) + \left(\frac{-1}{g^2}dg \cdot f\right)$

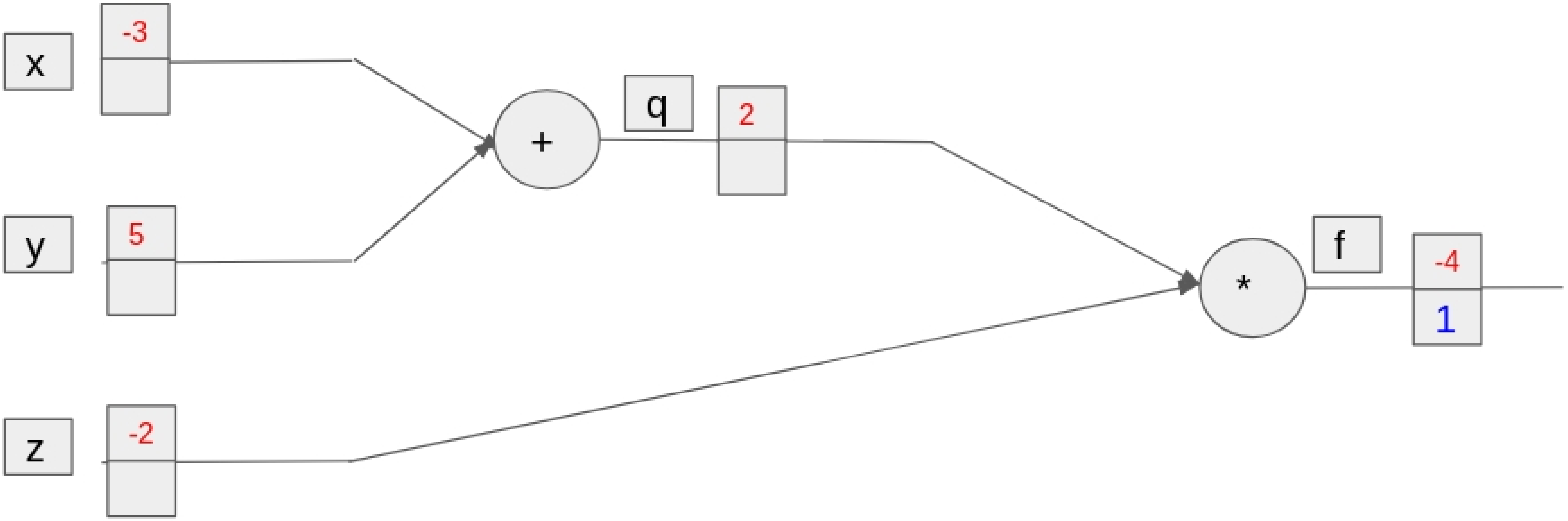
$$\frac{d}{dx}\left[(f(x))^n\right] = n(f(x))^{n-1} \cdot f'(x)$$

$$\frac{d}{dx}\left[f(g(x))\right] = f'(g(x))g'(x)$$

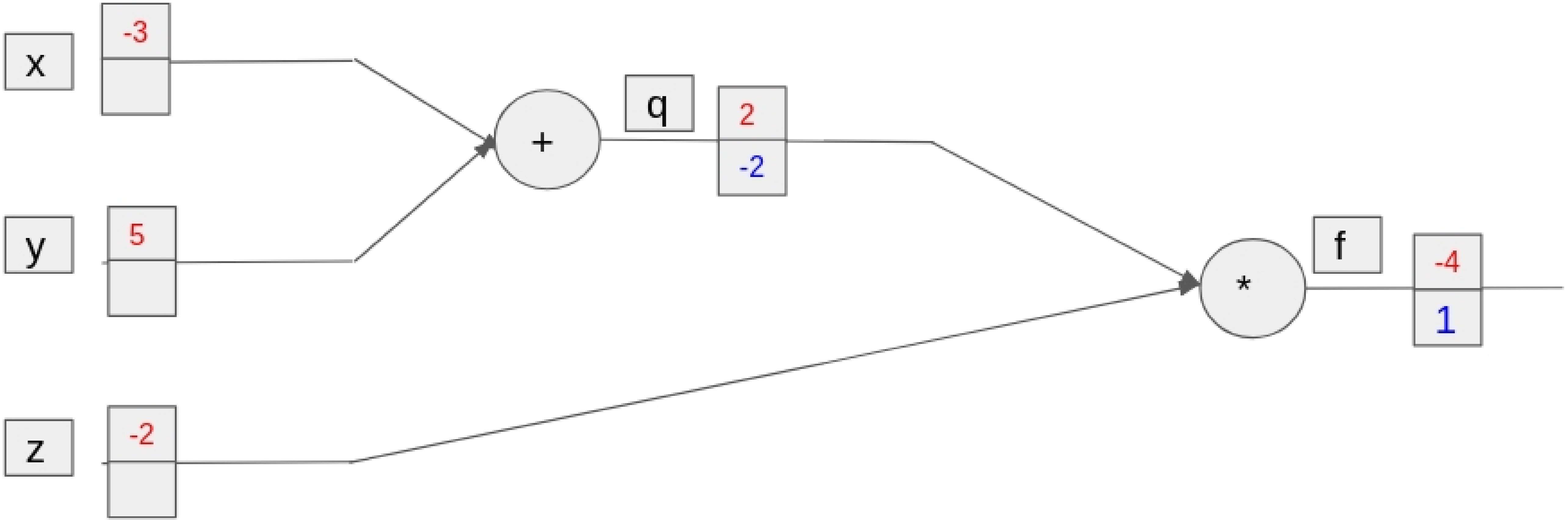
Derivative Example - Forward Pass



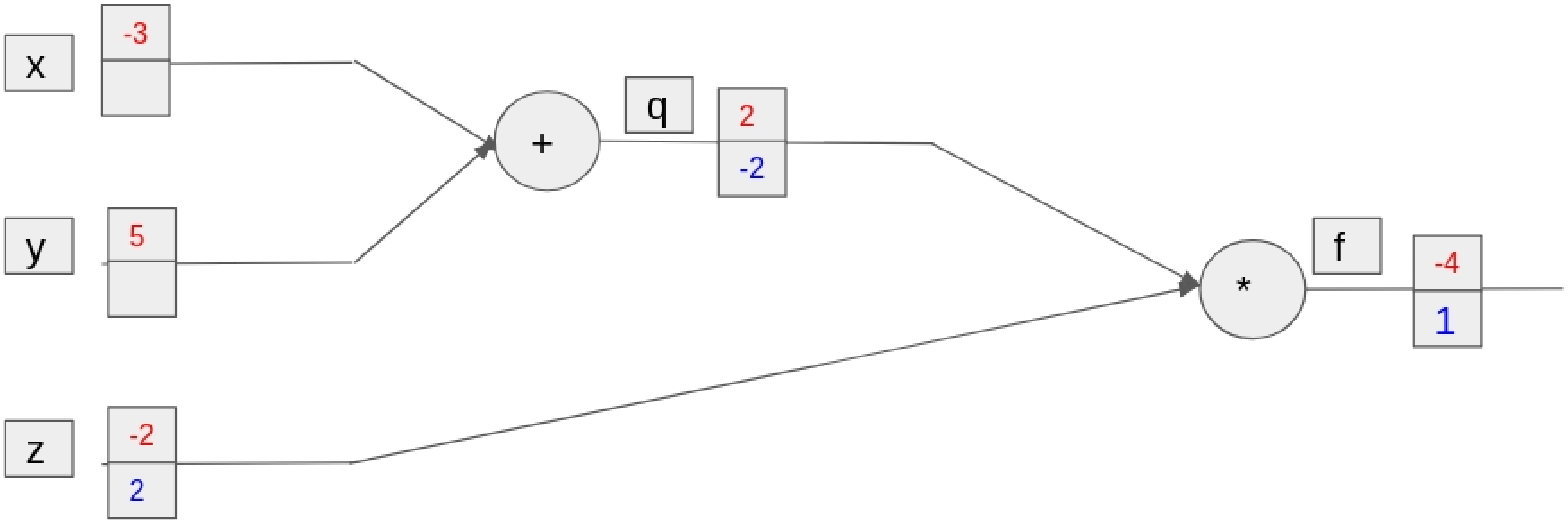
Derivative Example - Backward Pass



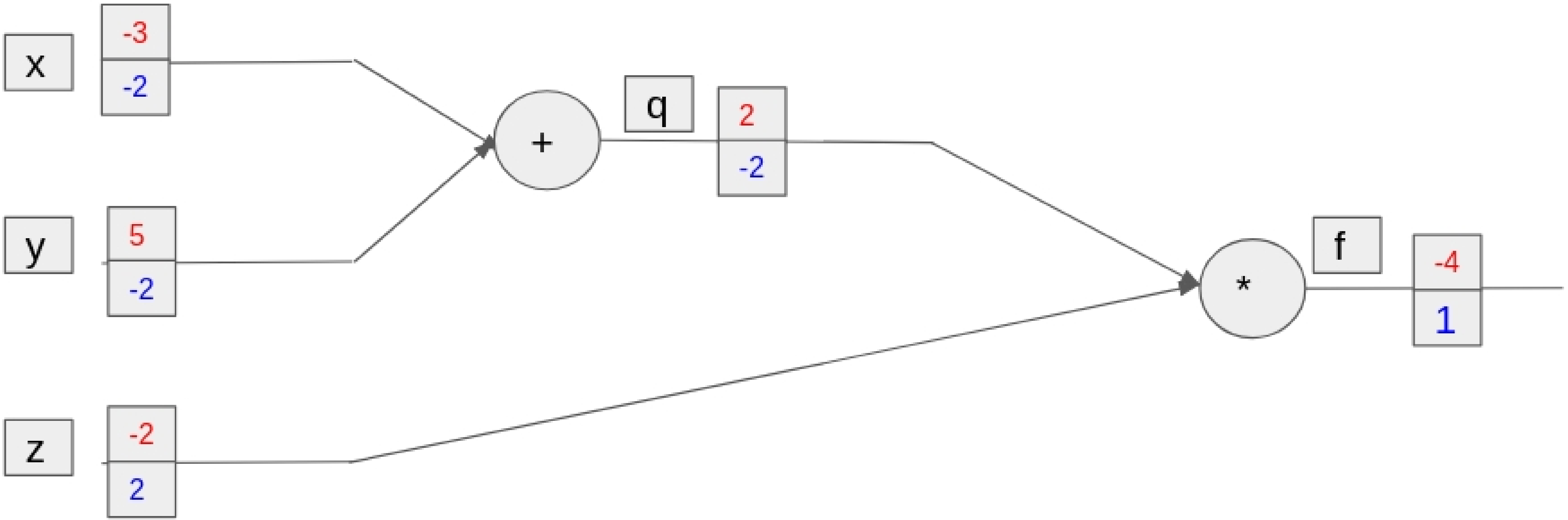
Derivative Example - Backward Pass



Derivative Example - Backward Pass



Derivative Example - Backward Pass



Backpropagation in PyTorch

```python
import torch

x = torch.tensor(-3., requires_grad=True)
y = torch.tensor(5., requires_grad=True)
z = torch.tensor(-2., requires_grad=True)

q = x + y
f = q * z

f.backward()


print("Gradient of z is: " + str(z.grad))
print("Gradient of y is: " + str(y.grad))
print("Gradient of x is: " + str(x.grad))
```

```
Gradient of z is: tensor(2.)
Gradient of y is: tensor(-2.)
Gradient of x is: tensor(-2.)
```

Let's practice

Introduction to Neural Networks

INTRODUCTION TO DEEP LEARNING WITH PYTORCH

Ismail Elezi
Ph.D. Student of Deep Learning

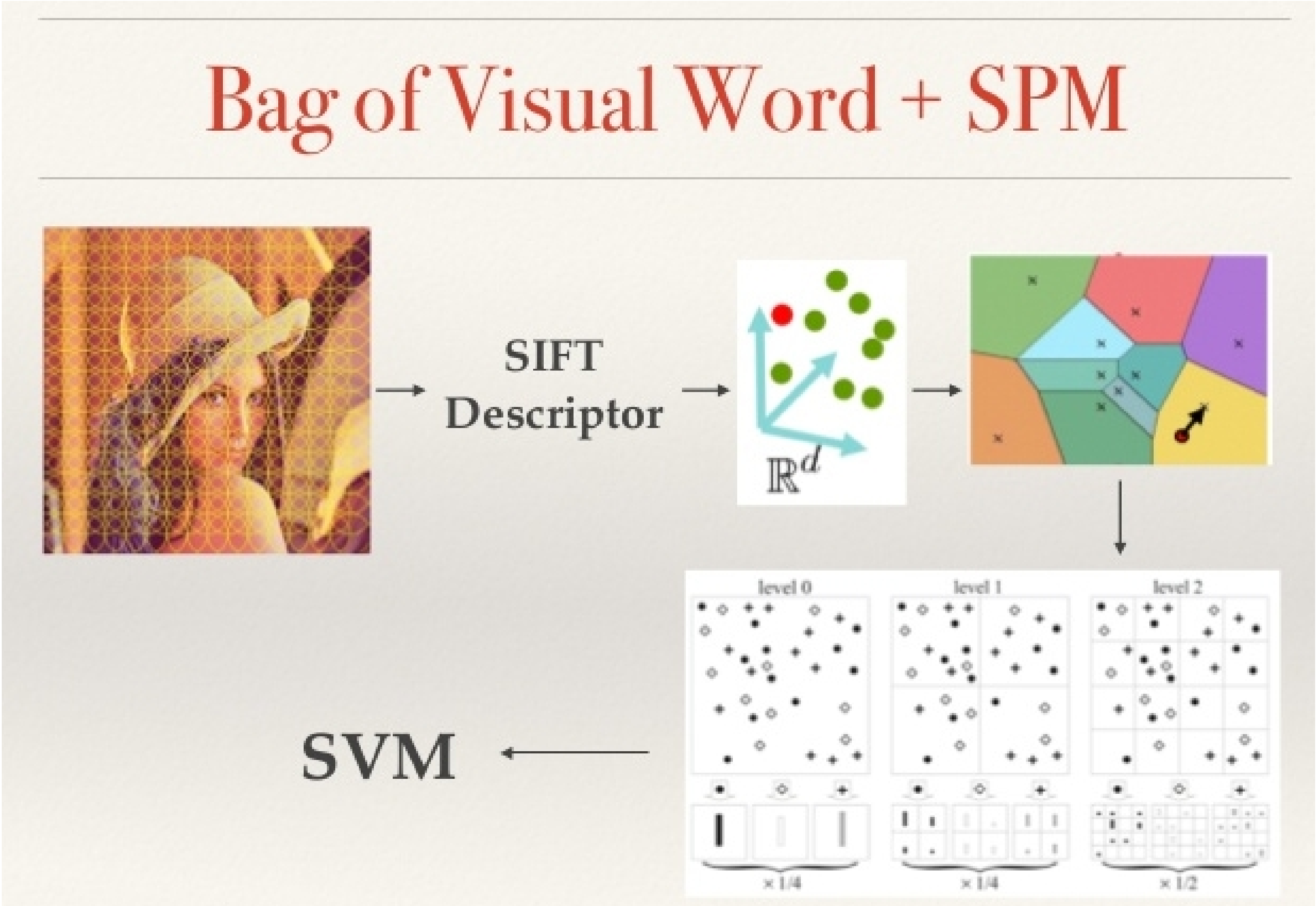
Other classifiers

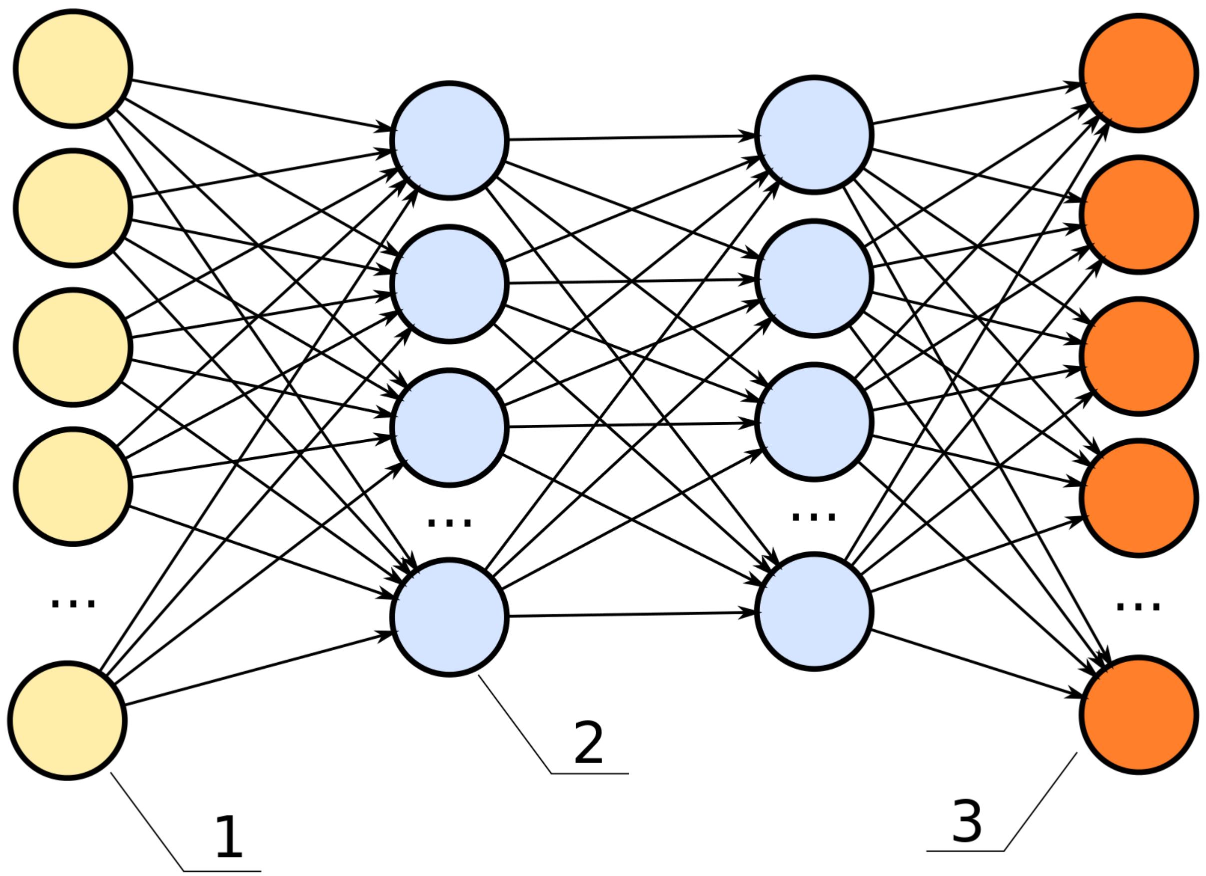
- k-Nearest Neighbour
- Logistic/Linear Regression
- Random Forests
- Gradient Boosted Trees
- Support Vector Machines
- ...

Incanter Dataset

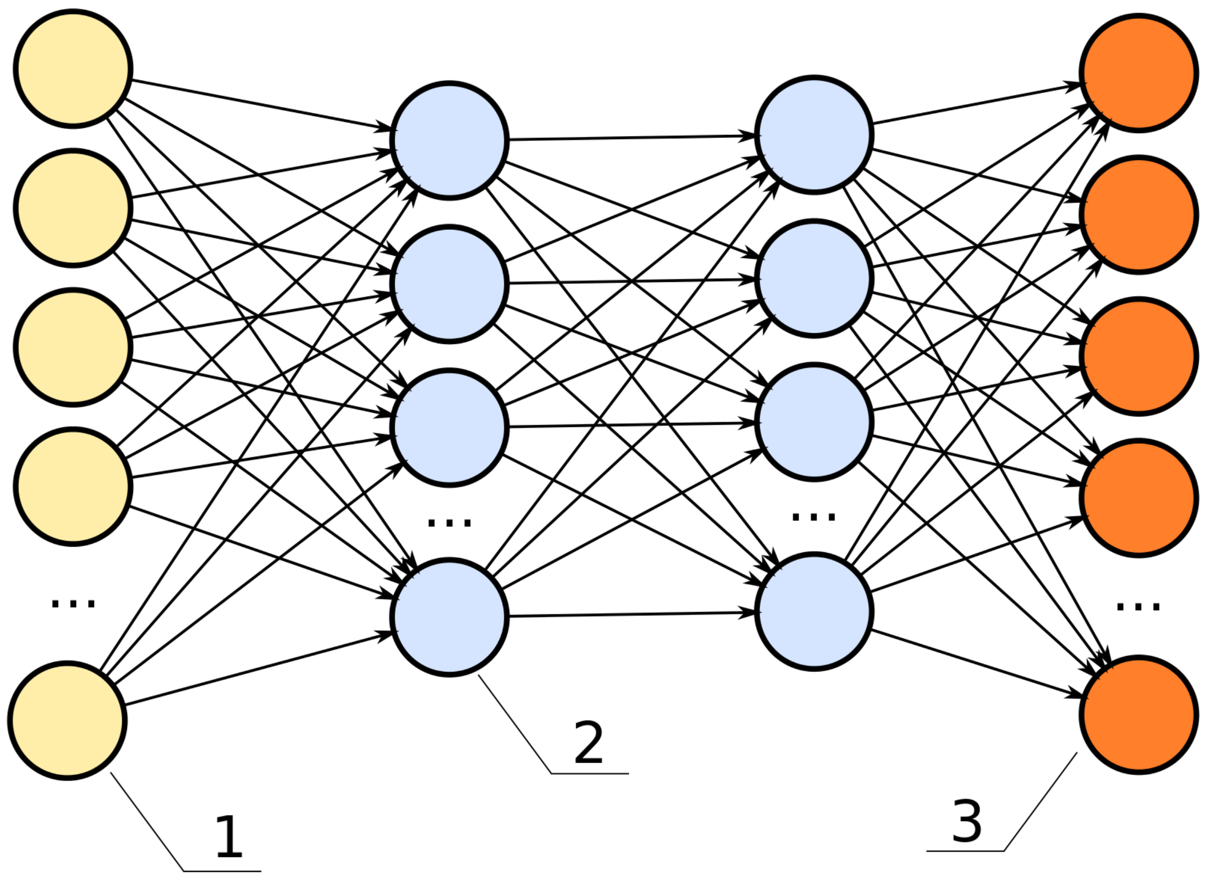
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa
5.4	3.4	1.7	0.2	setosa
5.1	3.7	1.5	0.4	setosa

ANN vs other classifiers





Fully connected neural networks




```python
import torch

input_layer = torch.rand(10)

w1 = torch.rand(10, 20)
w2 = torch.rand(20, 20)
w3 = torch.rand(20, 4)
h1 = torch.matmul(input_layer, w1)
h2 = torch.matmul(h1, w2)
output_layer = torch.matmul(h2, w3)
print(output_layer)
```

```
tensor([413.8647, 286.5770,
        361.8974, 294.0240])
```

Building a neural network - PyTorch style

```python
import torch
import torch.nn as nn


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(10, 20)
        self.fc2 = nn.Linear(20, 20)
        self.output = nn.Linear(20, 4)

    def forward(self, x):
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.output(x)
        return x
```

```python
input_layer = torch.rand(10)
net = Net()
result = net(input_layer)
```

Let's practice!

INTRODUCTION TO DEEP LEARNING WITH PYTORCH