Mathematics of Deep Learning - III

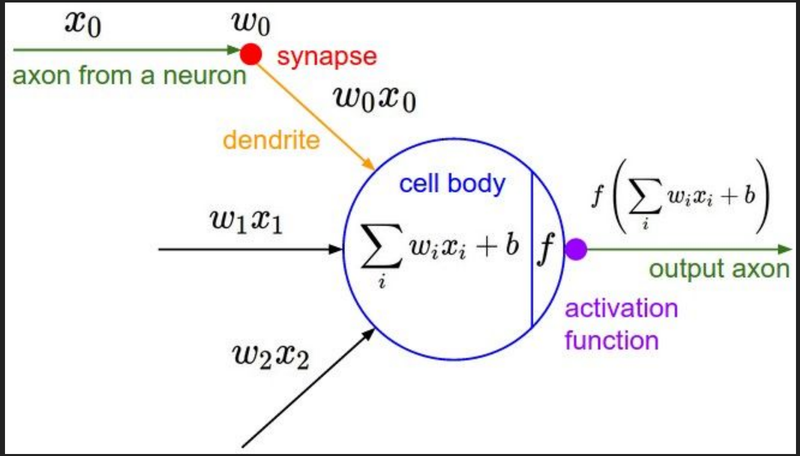
Amir Hajian

July 2019

aisc

Recap

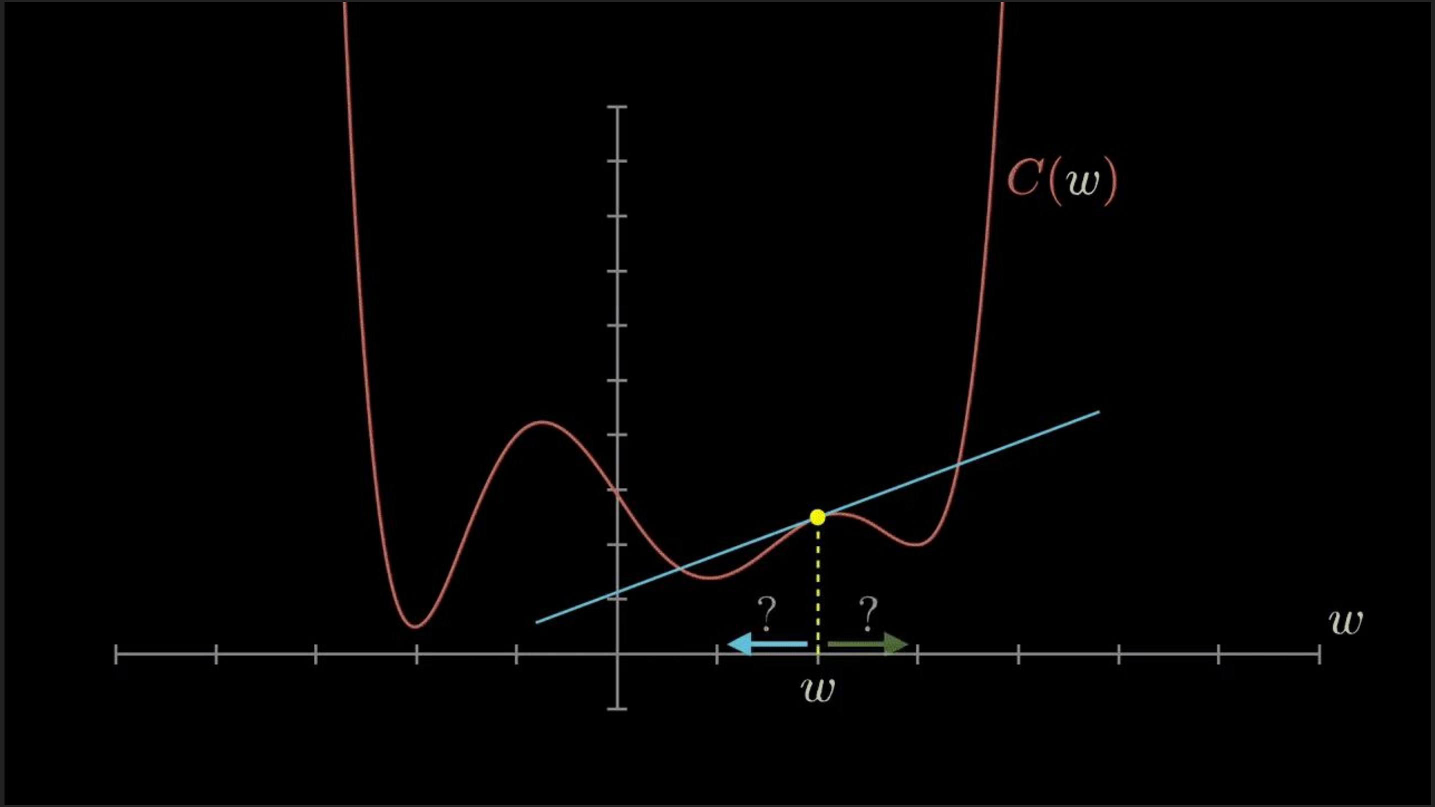
- It is all about matrices, vectors and scalars.



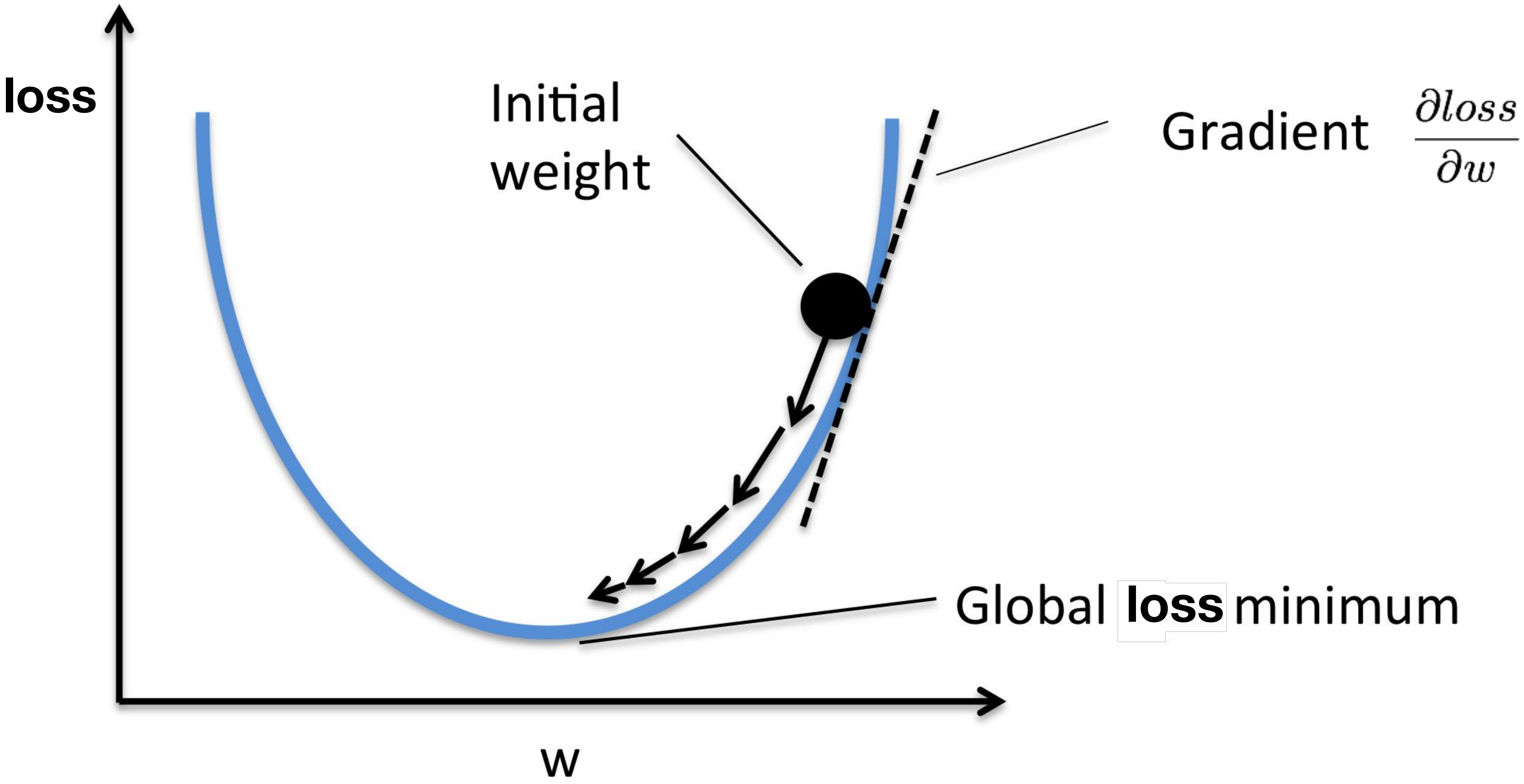
Mathematics of finding the solution

-

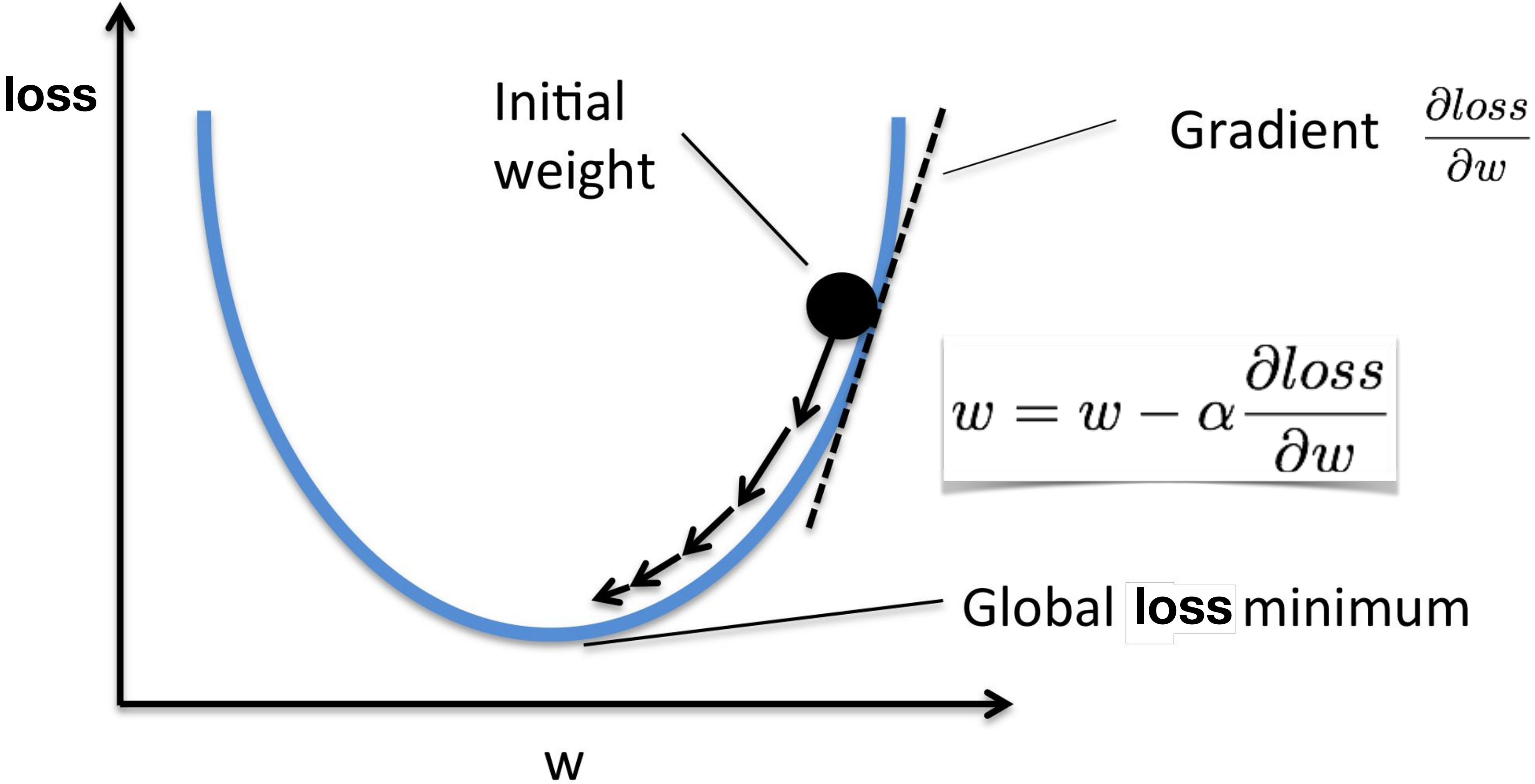
How Neural Networks Learn?



Gradient descent algorithm



Gradient descent algorithm



Derivative

$$loss = (\hat{y} - y)^2 = (x * w - y)^2$$

$$w = w - \alpha \frac{\partial loss}{\partial w}$$

Derivative

$$loss = (\hat{y} - y)^2 = (x * w - y)^2$$

$$w = w - \alpha \frac{\partial loss}{\partial w}$$

$$\frac{\partial loss}{\partial w} = ?$$

Derivative

$$loss = (\hat{y} - y)^2 = (x * w - y)^2$$

$$\frac{\partial loss}{\partial w} = ?$$

Derivative

$$loss = (\hat{y} - y)^2 = (x * w - y)^2$$

$$\frac{\partial loss}{\partial w} = ?$$

YOUR INPUT:
$f(w) =$

$$(xw - y)^2$$

Simplify **Roots/zeros**

FIRST DERIVATIVE:
$\frac{\mathrm{d}}{\mathrm{d}w}[f(w)] = f'(w) =$

The steps of calculation are displayed.
Move the mouse over a derivative $\frac{\mathrm{d}}{\mathrm{d}w}[\ldots]$ or tap it in order to show its calculation.

$$\frac{\mathrm{d}}{\mathrm{d}w}\left[(xw - y)^2\right]$$

$$= 2(xw - y) \cdot \frac{\mathrm{d}}{\mathrm{d}w}[xw - y]$$

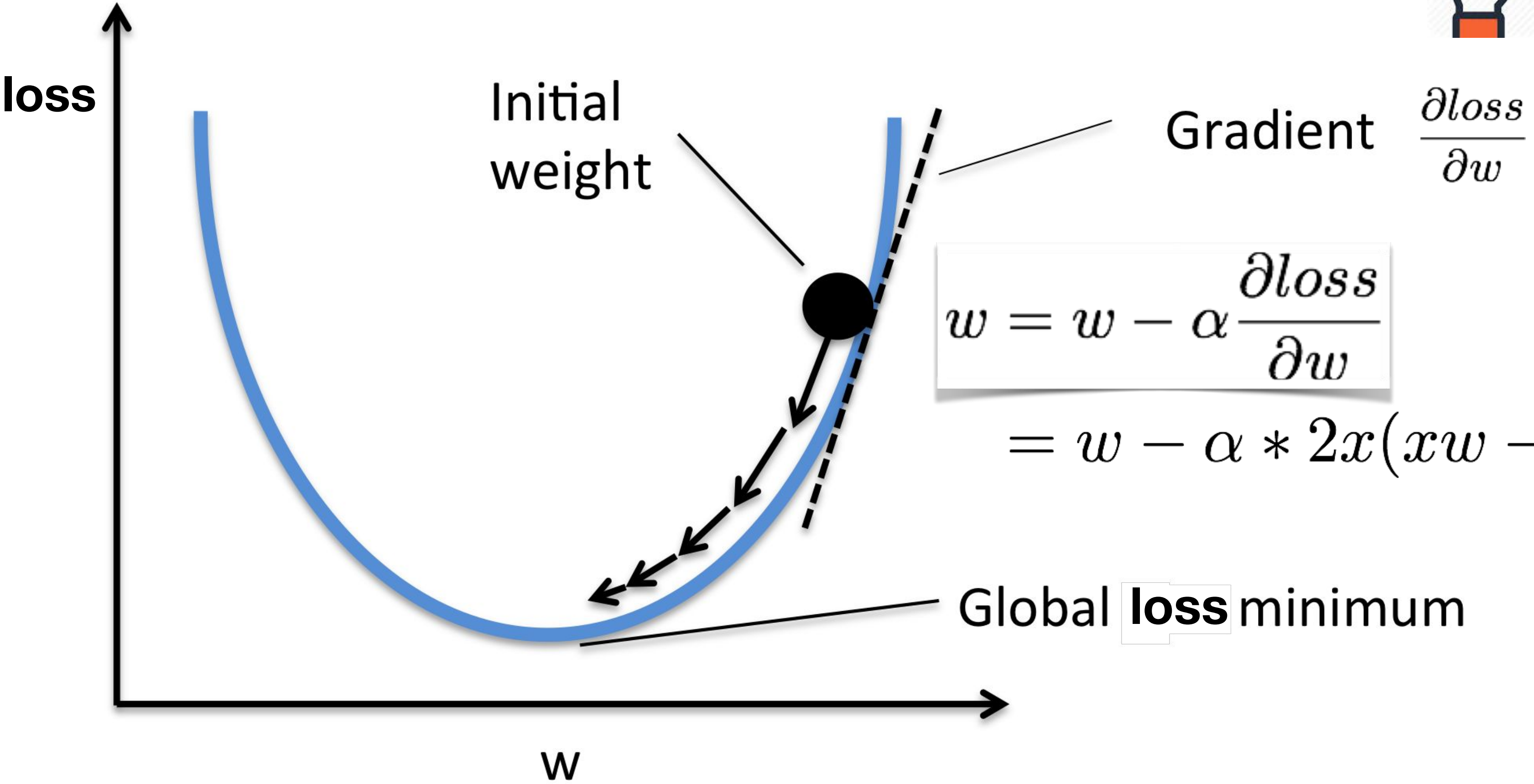
$$= 2\left(x \cdot \frac{\mathrm{d}}{\mathrm{d}w}[w] + \frac{\mathrm{d}}{\mathrm{d}w}[-y]\right)(xw - y)$$

$$= 2(1x + 0)(xw - y)$$

$$= 2x(xw - y)$$

https://www.derivative-calculator.net/

Let's implement!



$$w = w - \alpha \frac{\partial loss}{\partial w}$$

$$= w - \alpha * 2x(xw - y)$$

Data, Model, Loss, and Gradient

```
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

w = 1.0  # a random guess: random value

# our model forward pass
def forward(x):
   return x * w


# Loss function
def loss(x, y):
   y_pred = forward(x)
   return (y_pred - y) * (y_pred - y)


# compute gradient
def gradient(x, y):  # d_loss/d_w
   return 2 * x * (x * w - y)
```

$$2x(xw - y)$$

Training: updating weight

```python
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

w = 1.0  # a random guess: random value

# our model forward pass
def forward(x):
    return x * w


# Loss function
def loss(x, y):
    y_pred = forward(x)
    return (y_pred - y) * (y_pred - y)


# compute gradient
def gradient(x, y):  # d_loss/d_w
    return 2 * x * (x * w - y)
```

```python
# Before training
print("predict (before training)",  4, forward(4))

# Training loop
for epoch in range(100):
    for x_val, y_val in zip(x_data, y_data):
        grad = gradient(x_val, y_val)
        w = w - 0.01 * grad
        print("\tgrad: ", x_val, y_val, grad)
        l = loss(x_val, y_val)

    print("progress:", epoch, "w=", w, "loss=", l)

# After training
print("predict (after training)",  "4 hours", forward(4))
```

Output

(from gradient numeric computation)

```
predict (before training) 4 4.0
	grad:  1.0 2.0 -2.0
	grad:  2.0 4.0 -7.84
	grad:  3.0 6.0 -16.23
progress: 0 w= 1.26 loss= 4.92
	grad:  1.0 2.0 -1.48
	grad:  2.0 4.0 -5.8
	grad:  3.0 6.0 -12.0
progress: 1 w= 1.45 loss= 2.69
	grad:  1.0 2.0 -1.09
	grad:  2.0 4.0 -4.29
	grad:  3.0 6.0 -8.87
progress: 2 w= 1.6 loss= 1.47
	grad:  1.0 2.0 -0.81
	grad:  2.0 4.0 -3.17
	grad:  3.0 6.0 -6.56
..
progress: 7 w= 1.91 loss= 0.07
	grad:  1.0 2.0 -0.18
	grad:  2.0 4.0 -0.7
	grad:  3.0 6.0 -1.45
progress: 8 w= 1.93 loss= 0.04
	grad:  1.0 2.0 -0.13
	grad:  2.0 4.0 -0.52
	grad:  3.0 6.0 -1.07
progress: 9 w= 1.95 loss= 0.02
predict (after training) 4 hours 7.80
```

```python
# Before training
print("predict (before training)",  4, forward(4))

# Training loop
for epoch in range(100):
   for x_val, y_val in zip(x_data, y_data):
       grad = gradient(x_val, y_val)
       w = w - 0.01 * grad
       print("\tgrad: ", x_val, y_val, grad)
       l = loss(x_val, y_val)

   print("progress:", epoch, "w=", w, "loss=", l)

# After training
print("predict (after training)",  "4 hours", forward(4))
```

Hands-on: Do it yourself

- Use this notebook to fit a function to training data using gradient descent:

  https://colab.research.google.com/drive/18Kfyw2aw4n4TvL49u6Mu5B20yUClhwfZ

Computing gradient in simple network

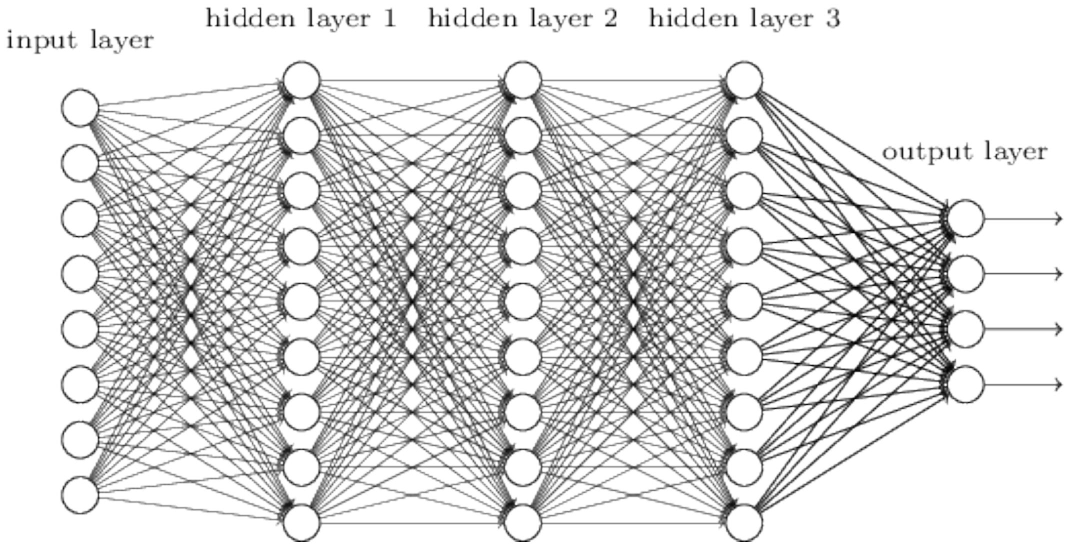
$$x \longrightarrow \boxed{\text{Linear}} \longrightarrow \hat{y}$$

Gradient of **loss**
with respect to ***w***

$$\frac{\partial loss}{\partial w} = ?$$

```python
# compute gradient
def gradient(x, y):  # d_loss/d_w
    return 2 * x * (x * w - y)
```

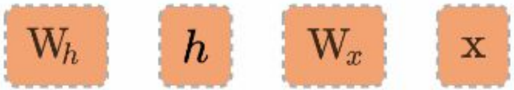
Complicated network?



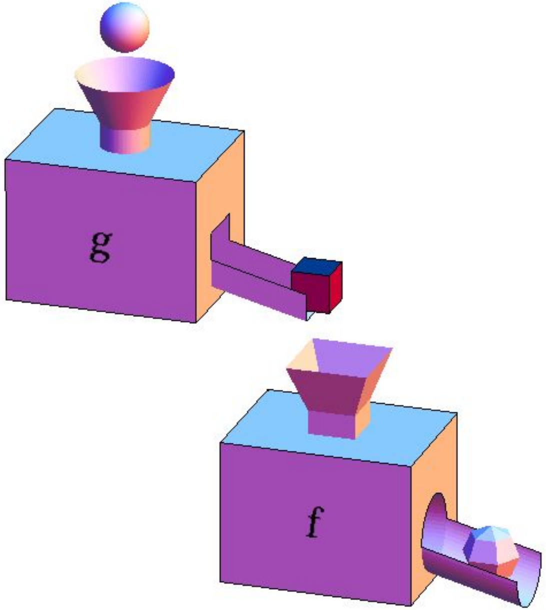
Gradient of **loss** with respect to ***w***

$$\frac{\partial loss}{\partial w} = ?$$

Better way? Computational graph + chain rule



Chain Rule

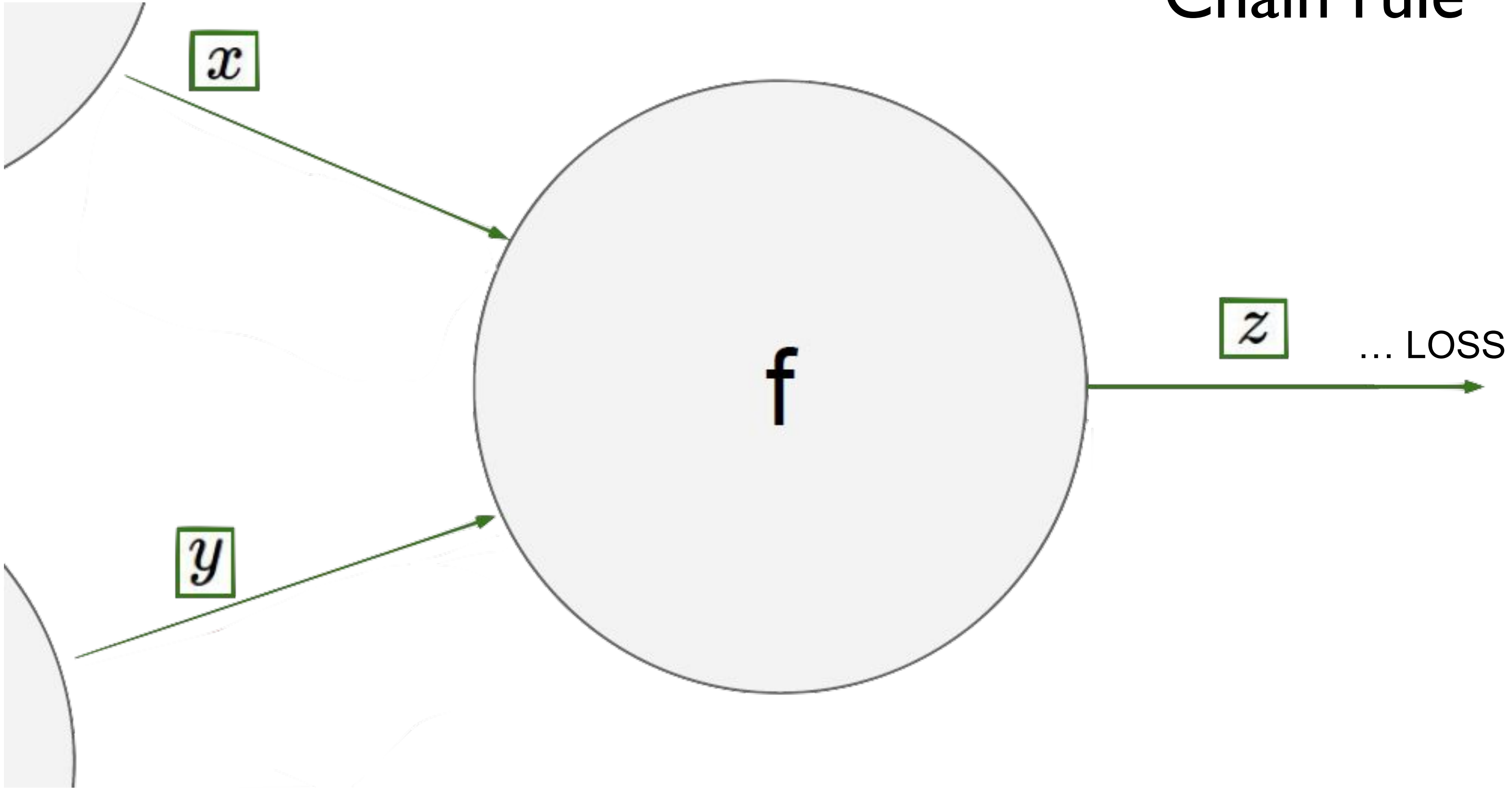


$$f = f(g)\ ; \quad g = g(x)$$

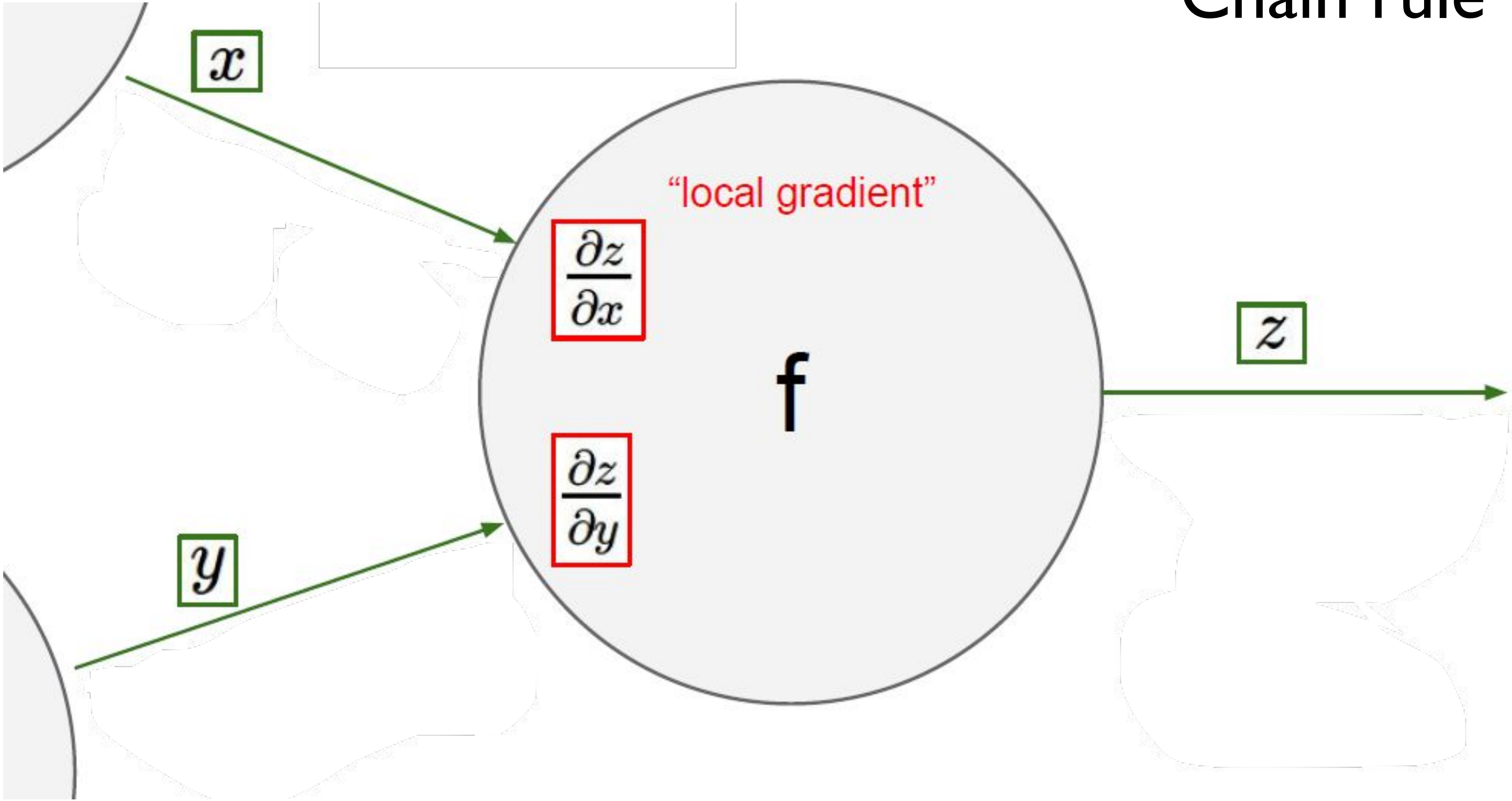
$$\frac{df}{dx} = \frac{df}{dg}\,\frac{dg}{dx}$$

http://mathinsight.org/image/function_machines_composed

Chain rule

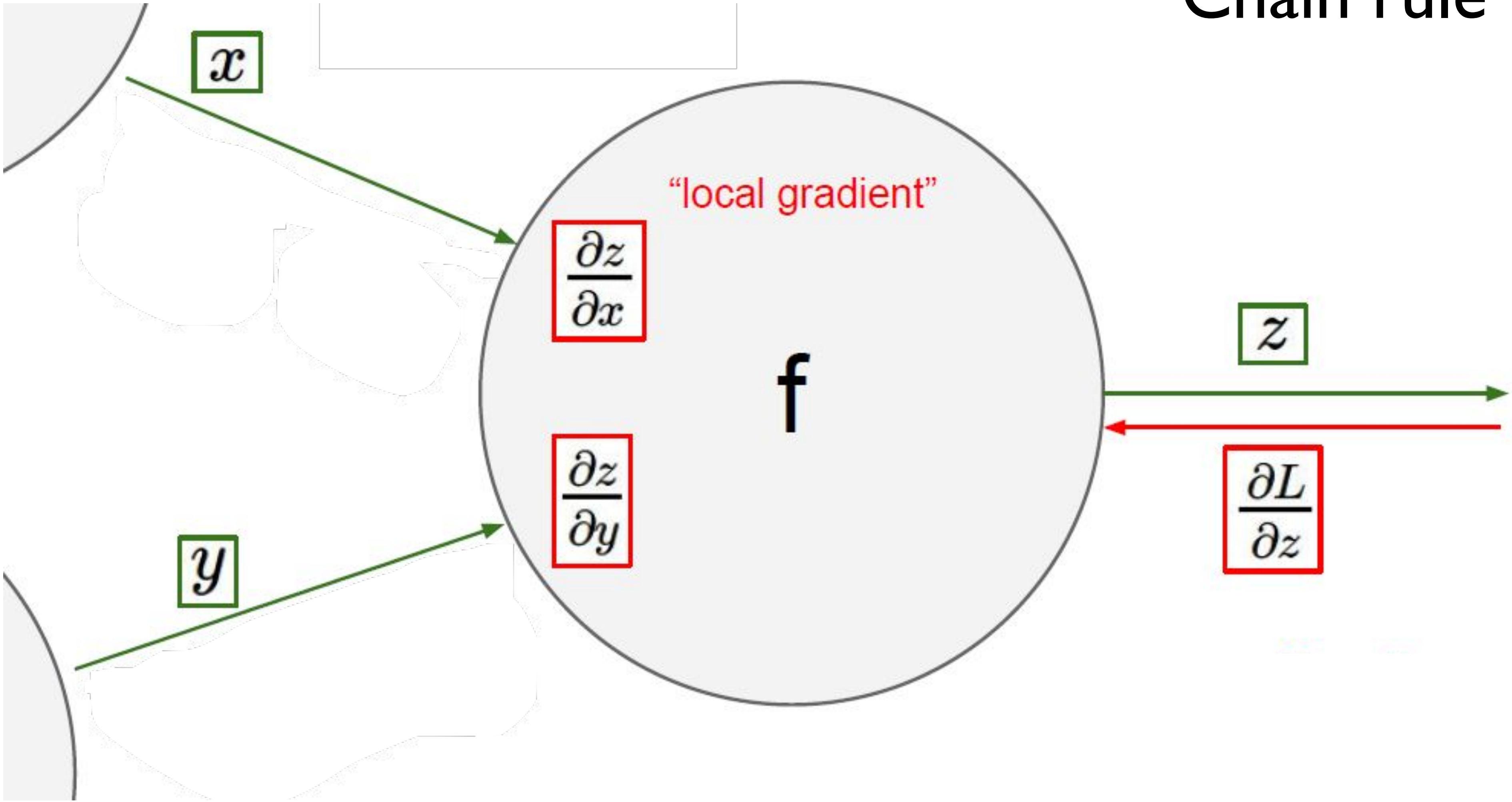


Chain rule

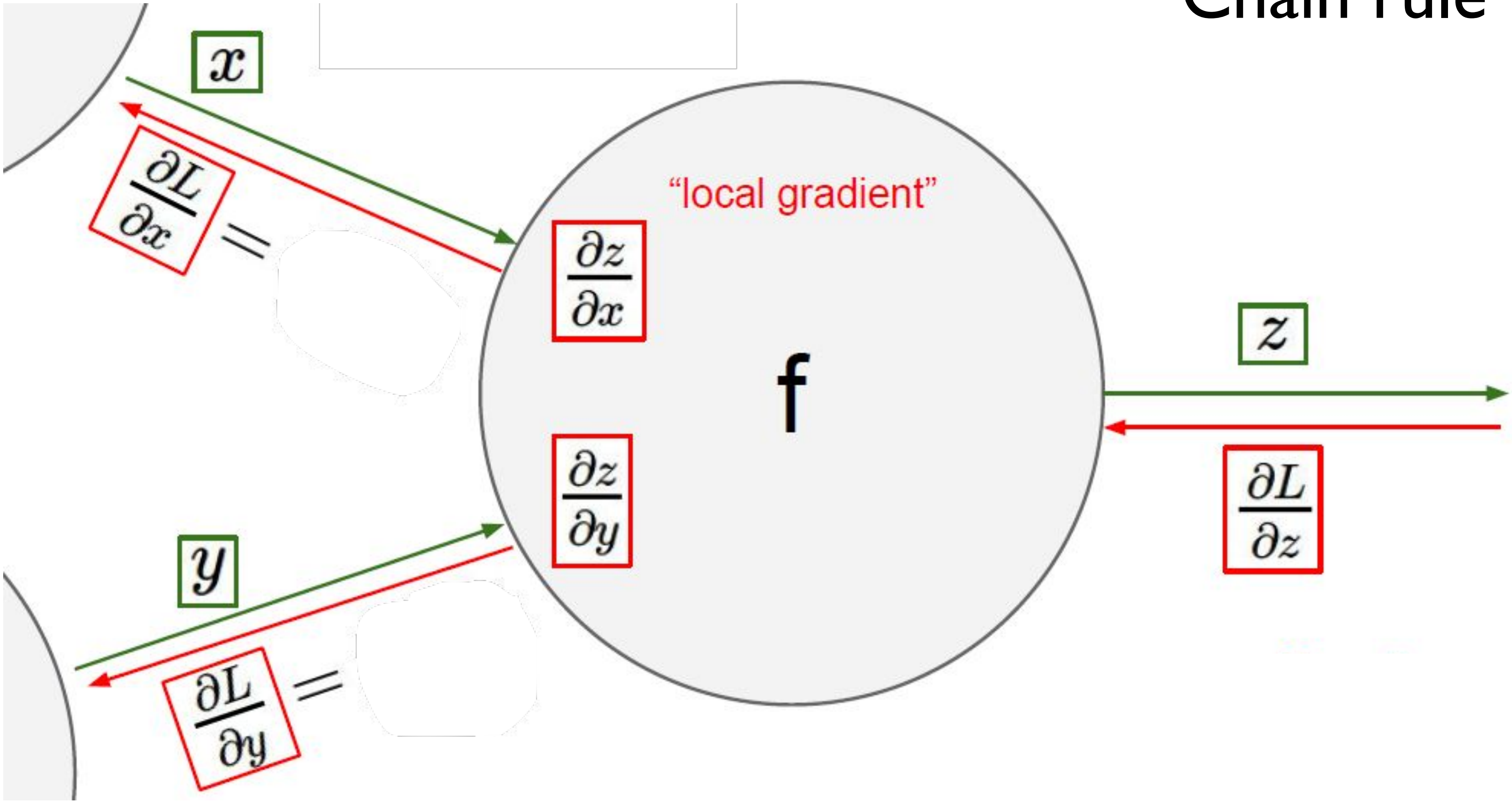


x
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
y

Chain rule

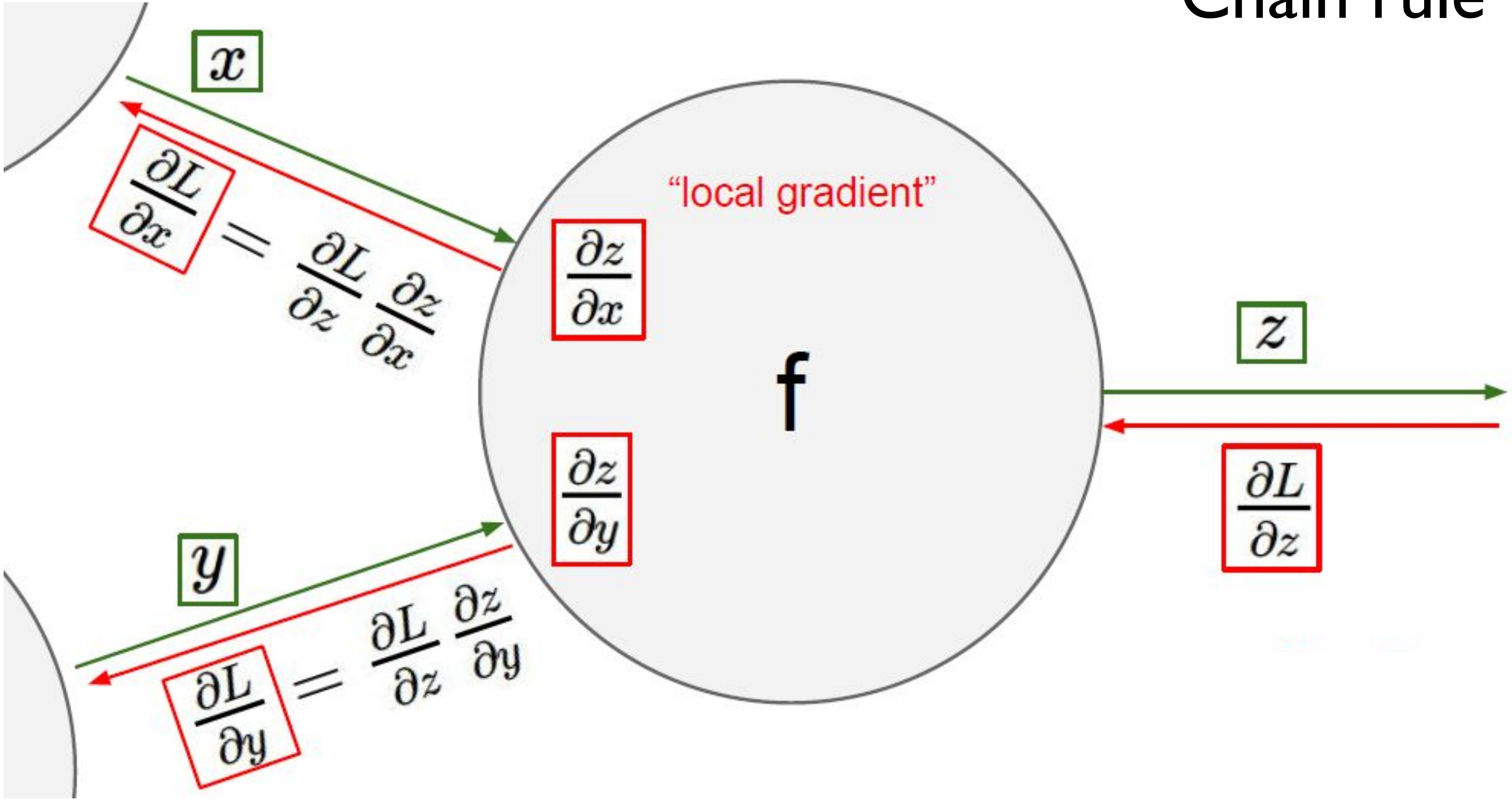

x
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y

Chain rule



x
$\frac{\partial L}{\partial x} =$
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} =$

Chain rule



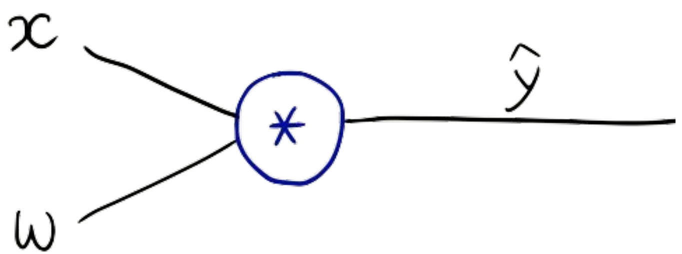
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$

Computational graph

$$\hat{y} = x * w$$

Computational graph

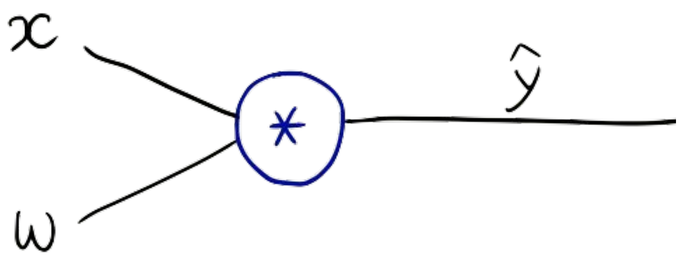
$$\hat{y} = x * w$$



Computational graph

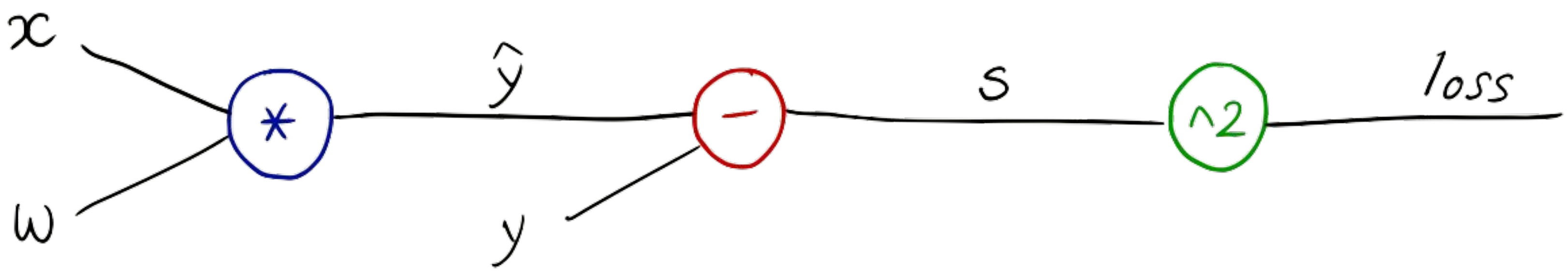
$$\hat{y} = x * w$$

$$loss = (\hat{y} - y)^2 = (x * w - y)^2$$

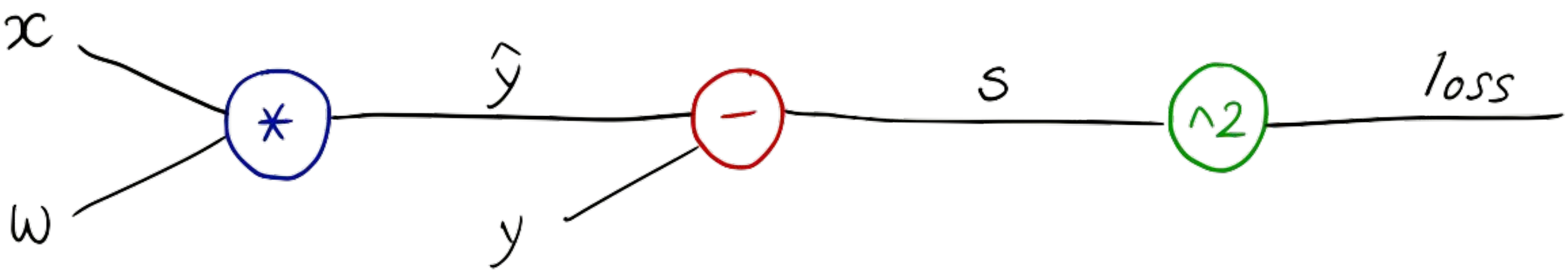


Computational graph

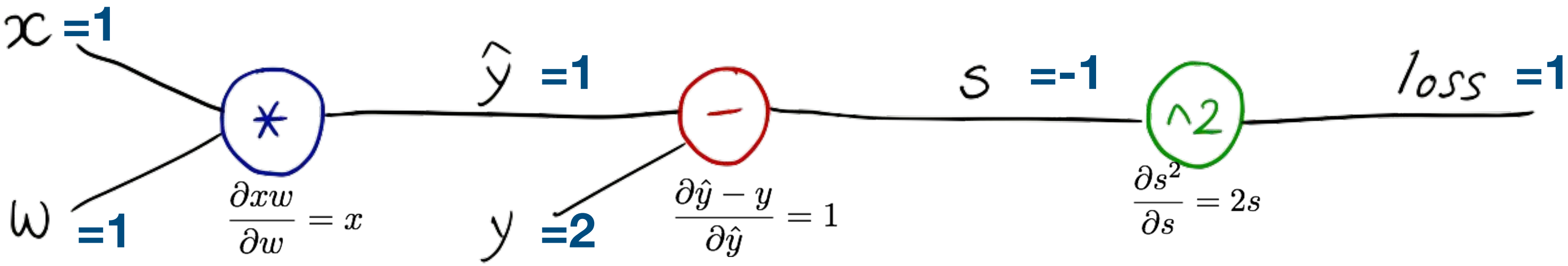
$$\hat{y} = x * w \qquad\qquad loss = (\hat{y} - y)^2 = (x * w - y)^2$$



Forward pass x=1, y = 2 where w=1

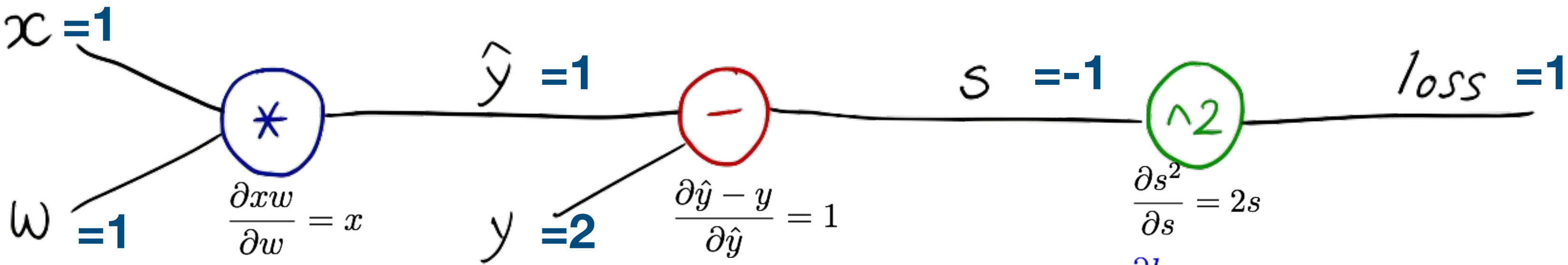


❷ Backward propagation



$$\frac{\partial loss}{\partial w} =$$

❷ Backward propagation



$$\frac{\partial xw}{\partial w} = x$$

$$\frac{\partial \hat{y} - y}{\partial \hat{y}} = 1$$

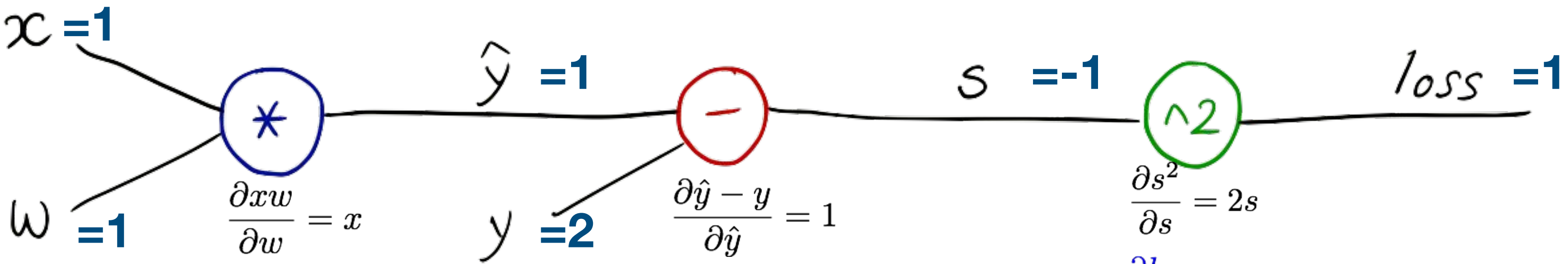
$$\frac{\partial s^2}{\partial s} = 2s$$

$$\frac{\partial loss}{\partial s} = 2s = -2$$

$$\frac{\partial loss}{\partial \hat{y}} = \frac{\partial loss}{\partial s}\frac{\partial s}{\partial \hat{y}} = -2 * 1 = -2$$

$$\frac{\partial loss}{\partial w} = \frac{\partial loss}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w} = -2 * x = -2 * 1 = -2$$

❷ Backward propagation



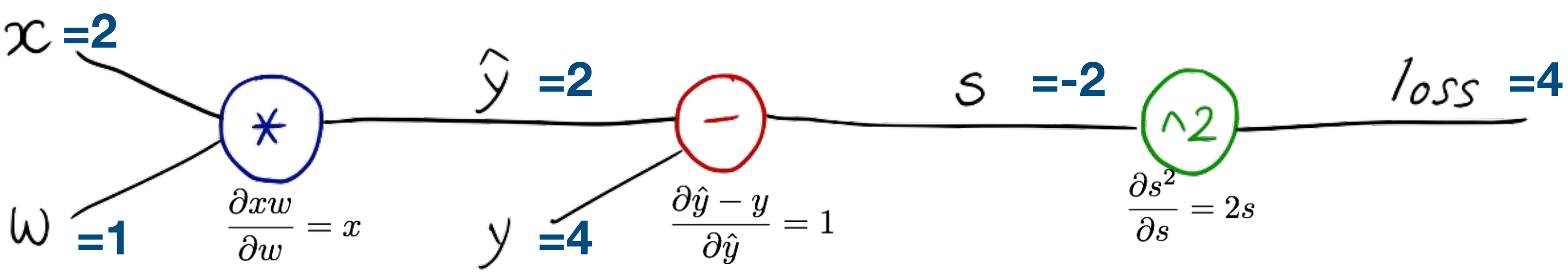
$$\frac{\partial loss}{\partial s} = 2s = -2$$

$$\frac{\partial loss}{\partial \hat{y}} = \frac{\partial loss}{\partial s}\frac{\partial s}{\partial \hat{y}} = -2 * 1 = -2$$

$$\frac{\partial loss}{\partial w} = \frac{\partial loss}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w} = -2 * x = -2 * 1 = -2$$

```
    grad:  1.0 2.0 -2.0
    grad:  2.0 4.0 -7.84
    grad:  3.0 6.0 -16.2288
progress: 0 4.919240100095999
```

Exercise 4-1: x = 2, y=4, w=1



$$\frac{\partial loss}{\partial w} =$$

Hands-on: Do it yourself

- Use this notebook to experiment with autograd in pytorch:

Hands-on: Do it yourself

- Use this notebook to fit a function to training data using PyTorch Autograd and backpropagation

  https://colab.research.google.com/drive/1sBkR95yLyxL9V4uyAaw4BwWWryf44C1m

Automatic Differentiation

Hands-on: autograd

```
# Create a Rank-2 tensor of all ones

x = torch.ones(2, 2, requires_grad=True)

print(x)

# Define y to be a function of x

y = x+2

# And z to be a function of y (and hence x):

z = 3*y*y

out = z.mean()

print(z, out)
```

```
# Now backprop:

out.backward()

# print gradients d(out)/dx

print(x.grad)
```

You should have got a matrix of `4.5`. Let's call the `out` *Tensor* "o". You have that $o = \frac{1}{4}\sum_i z_i$, $z_i = 3(x_i + 2)^2$ and $z_i\big|_{x_i=1} = 27$. Therefore, $\frac{\partial o}{\partial x_i} = \frac{3}{2}(x_i + 2)$, hence $\frac{\partial o}{\partial x_i}\big|_{x_i=1} = \frac{9}{2} = 4.5$.

Assignments

Reading:

- Gradient Descent Demystified (Excellent)

Programming

- Learn to work with ConvNets. Follow these tutorials to learn how to use ConvNets for various tasks in PyTorch
  - Beginner: Training a classifier
  - Advanced: Gated ConvNets for Neural NLP

Optional:

- Experiment with fast.ai image segmentation library:
  - Learn about it here and here
  - Try it as is
  - Build your own image segmentation either by using annotated data or by using pixel annotation tool.

Discussion Points