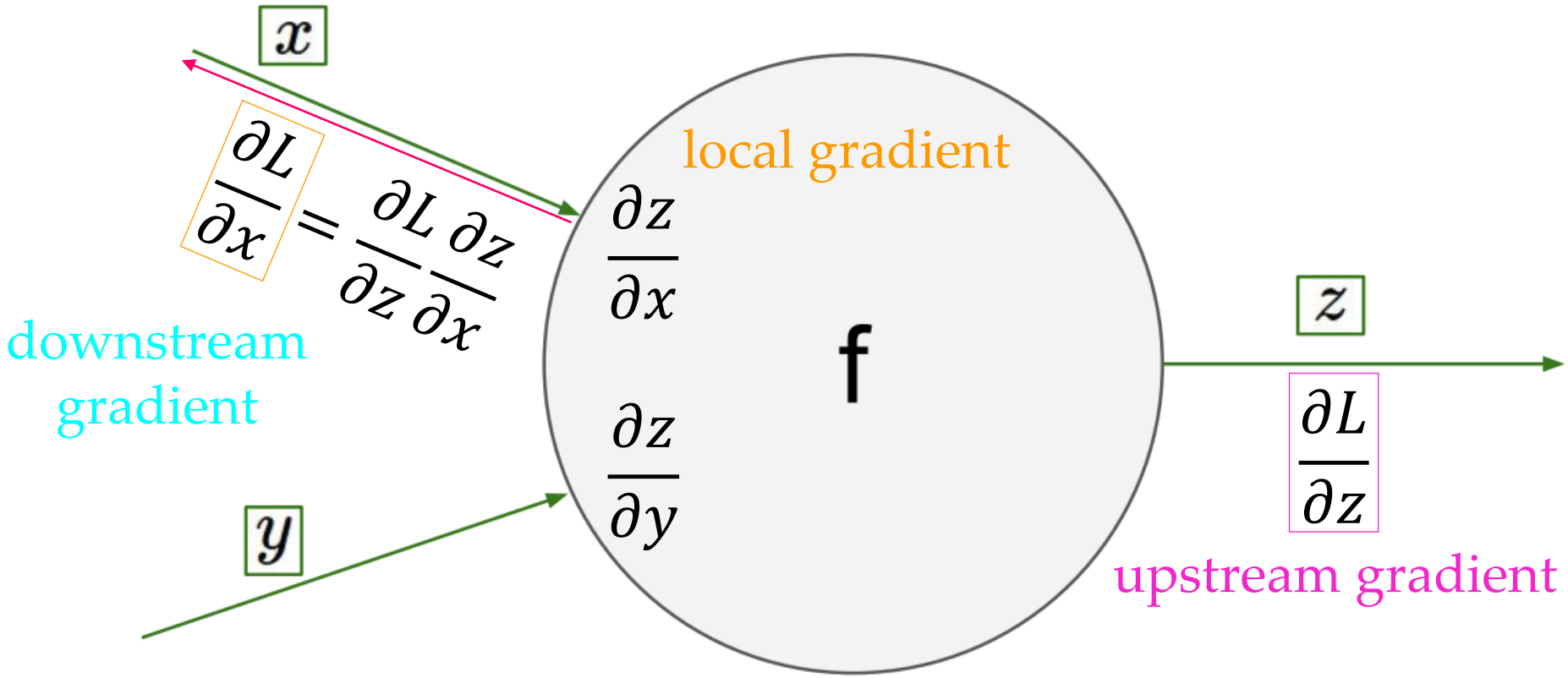

بسم الله الرحمن الرحيم

مبانی یادگیری عمیق

مدرس: محمدرضا محمدی

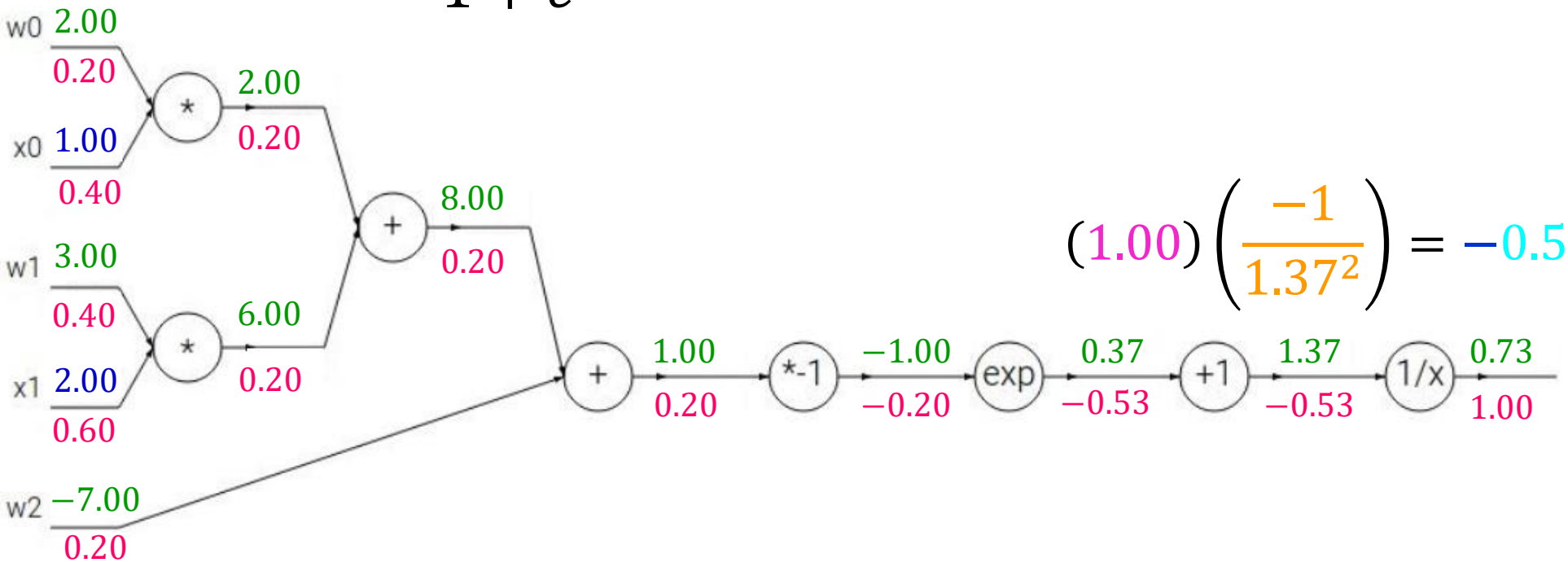
۱۴۰۰

پس‌انتشار (Backpropagation)



مثال: Linear + Sigmoid

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

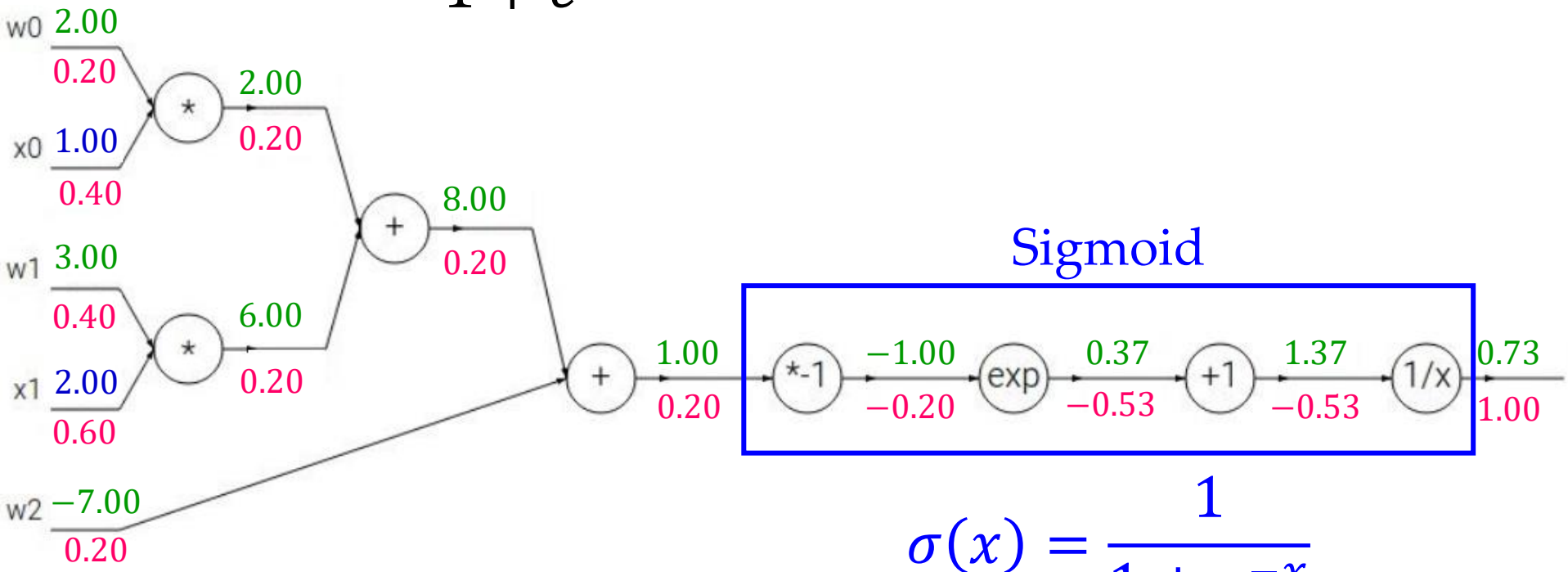


$$(1.00)\left(\frac{-1}{1.37^2}\right) = -0.53$$

$$\frac{\partial(ax)}{\partial x} = a \qquad \frac{\partial(c+x)}{\partial x} = 1 \qquad \frac{\partial(e^x)}{\partial x} = e^x \qquad \frac{\partial(1/x)}{\partial x} = \frac{-1}{x^2}$$

مثال: Linear + Sigmoid

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

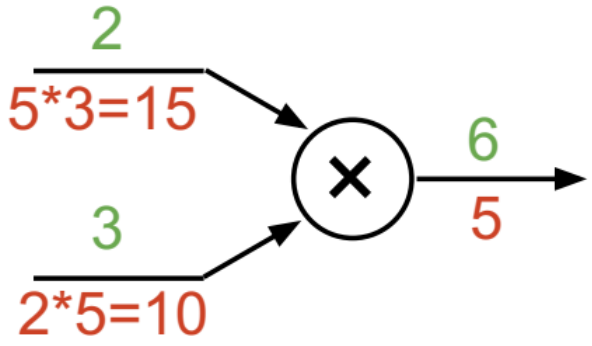


$$\sigma(x) = \frac{1}{1+e^{-x}}$$

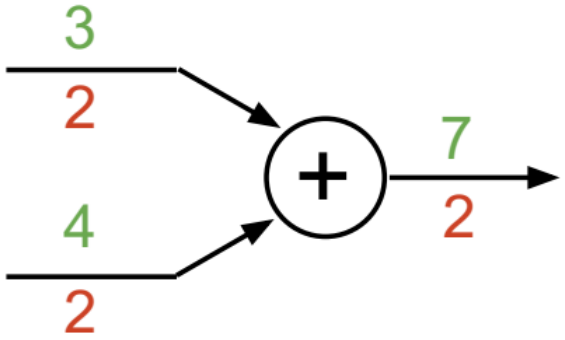
$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1-1+e^{-x}}{1+e^{-x}}\frac{1}{1+e^{-x}} = \big(1-\sigma(x)\big)\sigma(x)$$

نمونه‌هایی از جریان گرادیان

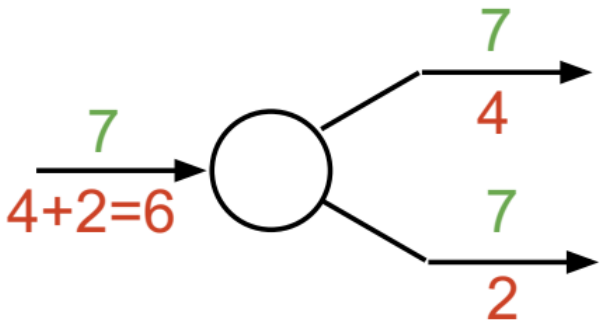
گیت **ضرب**: مبادله‌گر گرادیان



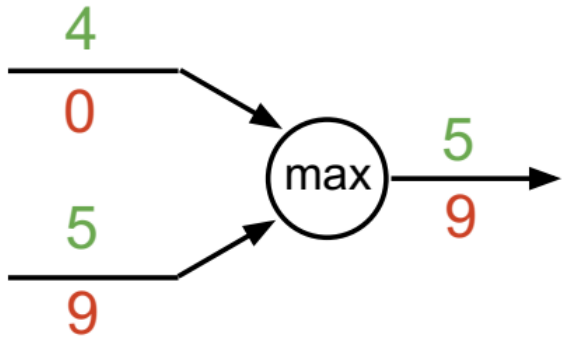
گیت **جمع**: توزیع‌کننده گرادیان



گیت **کپی**: جمع‌کننده گرادیان

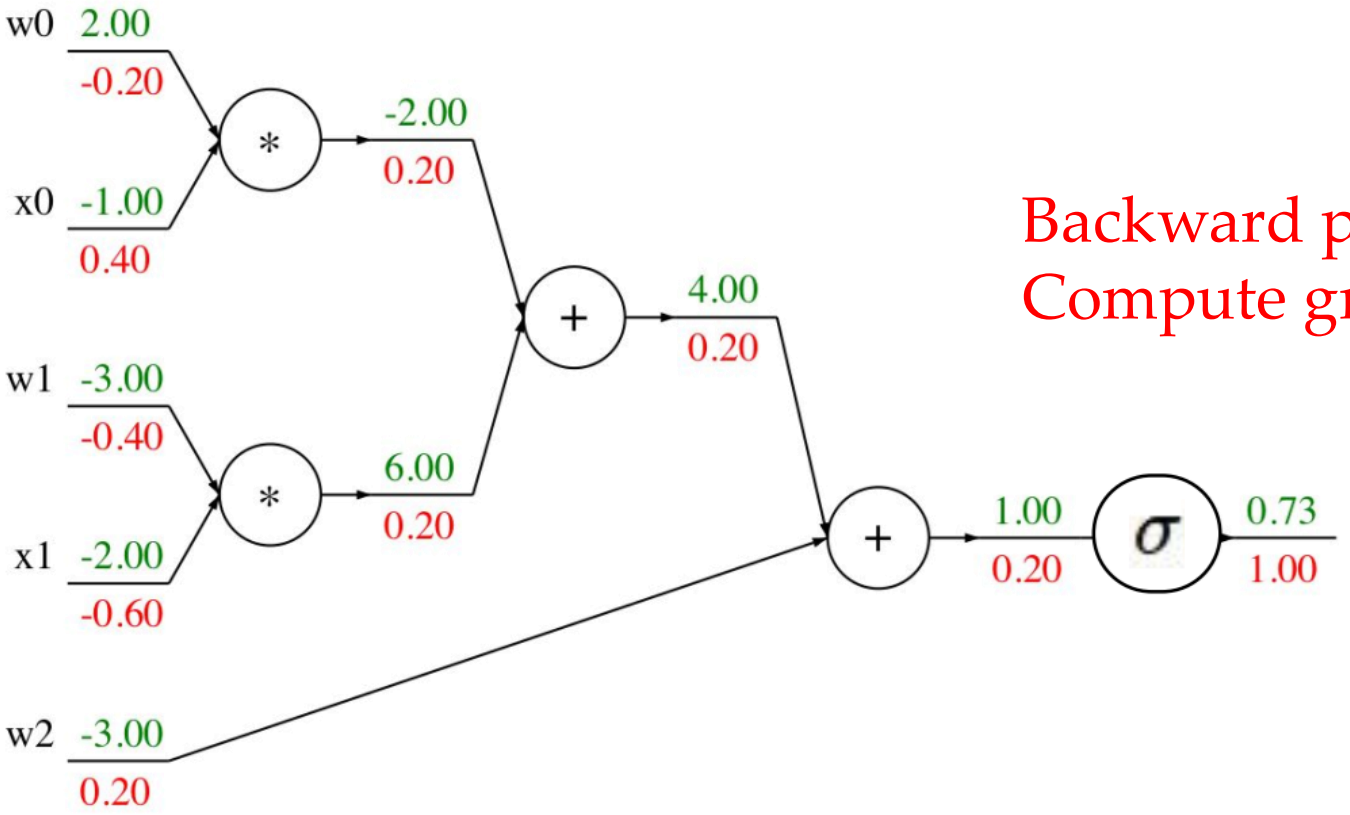


گیت **بیشینه**: روتر گرادیان



Forward pass:
Compute output

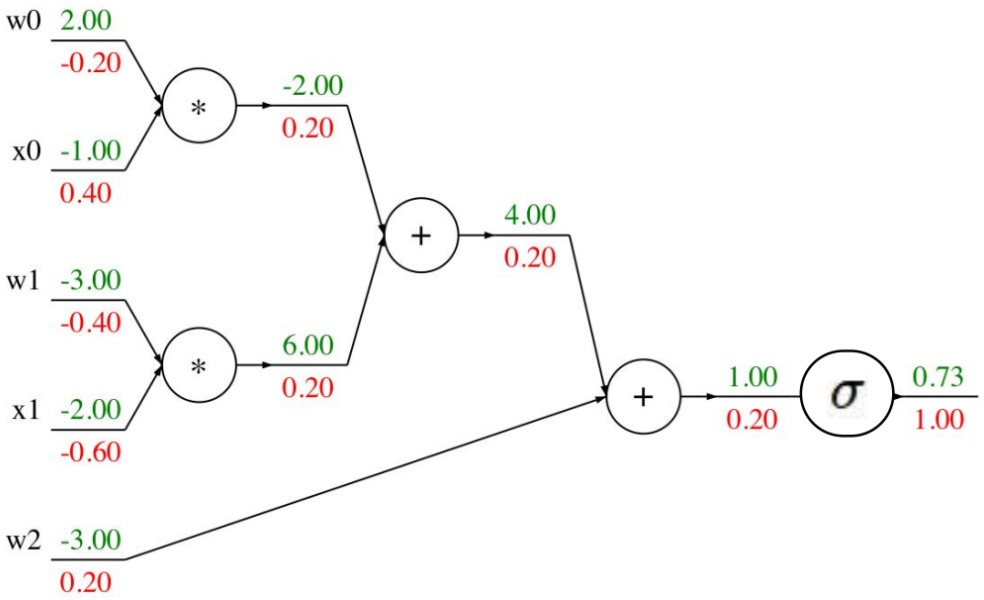
```
s0 = w0 * x0
s1 = w1 * x1
s2 = s0 + s1
s3 = s2 + w2
L = sigmoid(s3)
```



Backward pass:
Compute grads

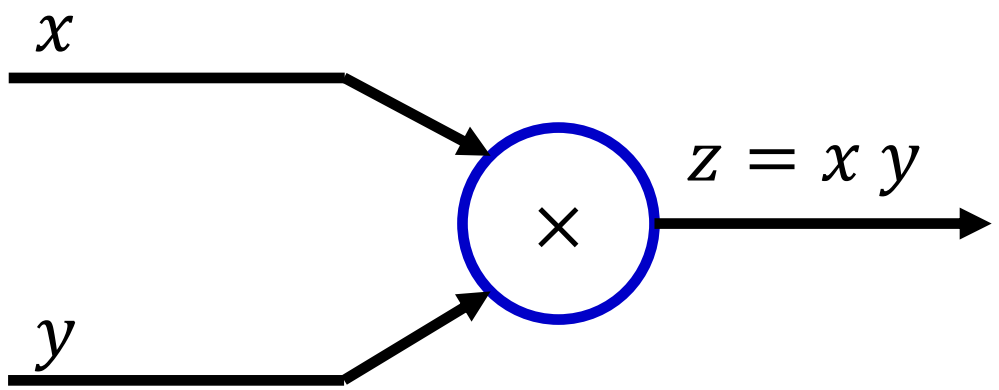
```
grad_L = 1.0
grad_s3 = grad_L * (1 - L) * L
grad_w2 = grad_s3
grad_s2 = grad_s3
grad_w0 = grad_s2
grad_s1 = grad_s2
grad_w1 = grad_s1 * x1
grad_x1 = grad_s1 * w1
grad_w0 = grad_s0 * x0
grad_x0 = grad_s0 * w0
```

پیاده‌سازی ماژولار




```python
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

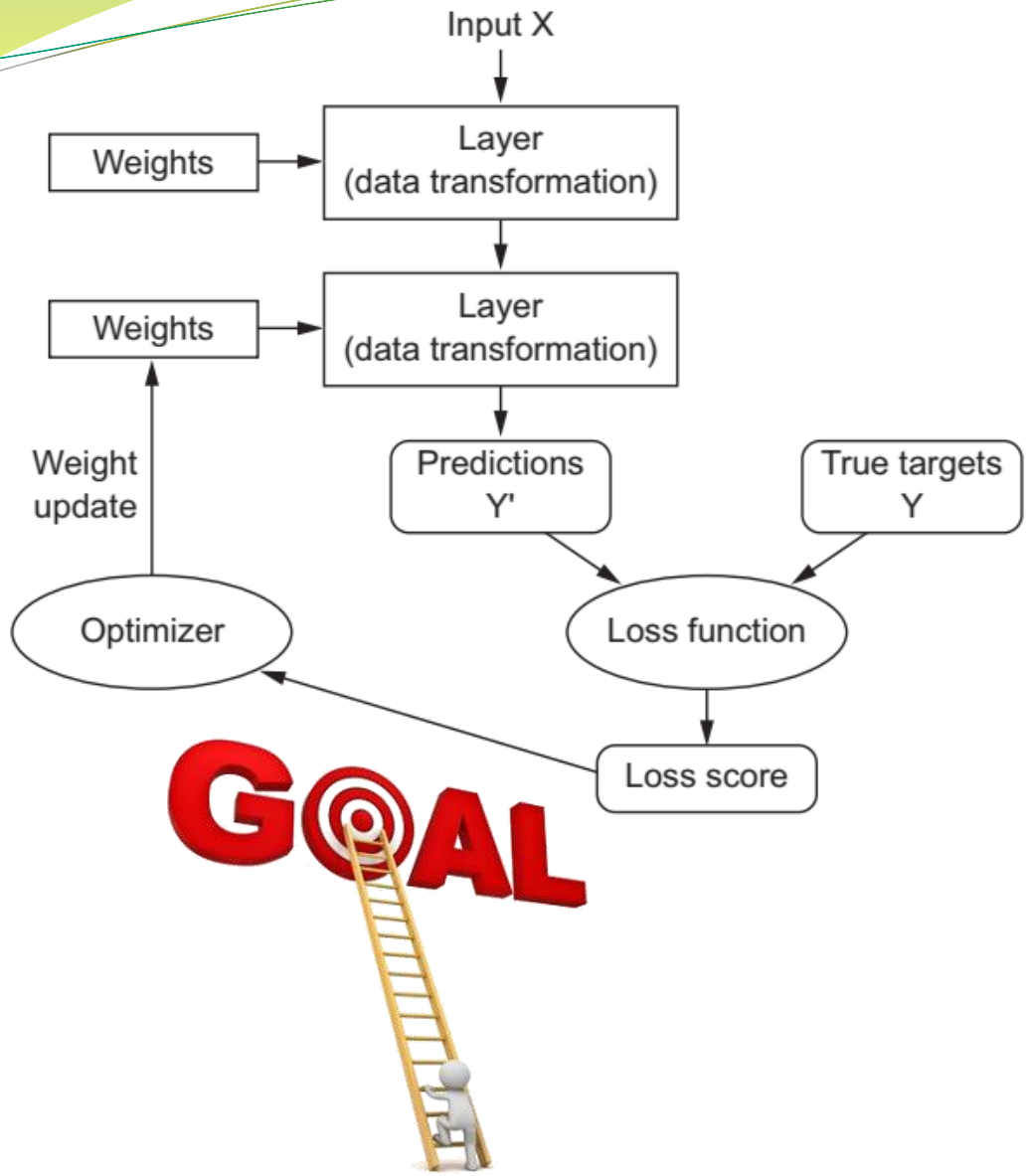
گیت ضرب




```
class MultuplyGate(object):
  """
  x,y are scalars
  """
  def forward(x,y):
    z = x*y
    self.x = x  # Cache
    self.y = y  # Cache
    # We cache x and y because we know that the derivatives contains them.
    return z
  def backward(dz):
    dx = self.y * dz          #self.y is dx
    dy = self.x * dz
    return [dx, dy]
```

تابع ضرر و تابع فعال‌سازی

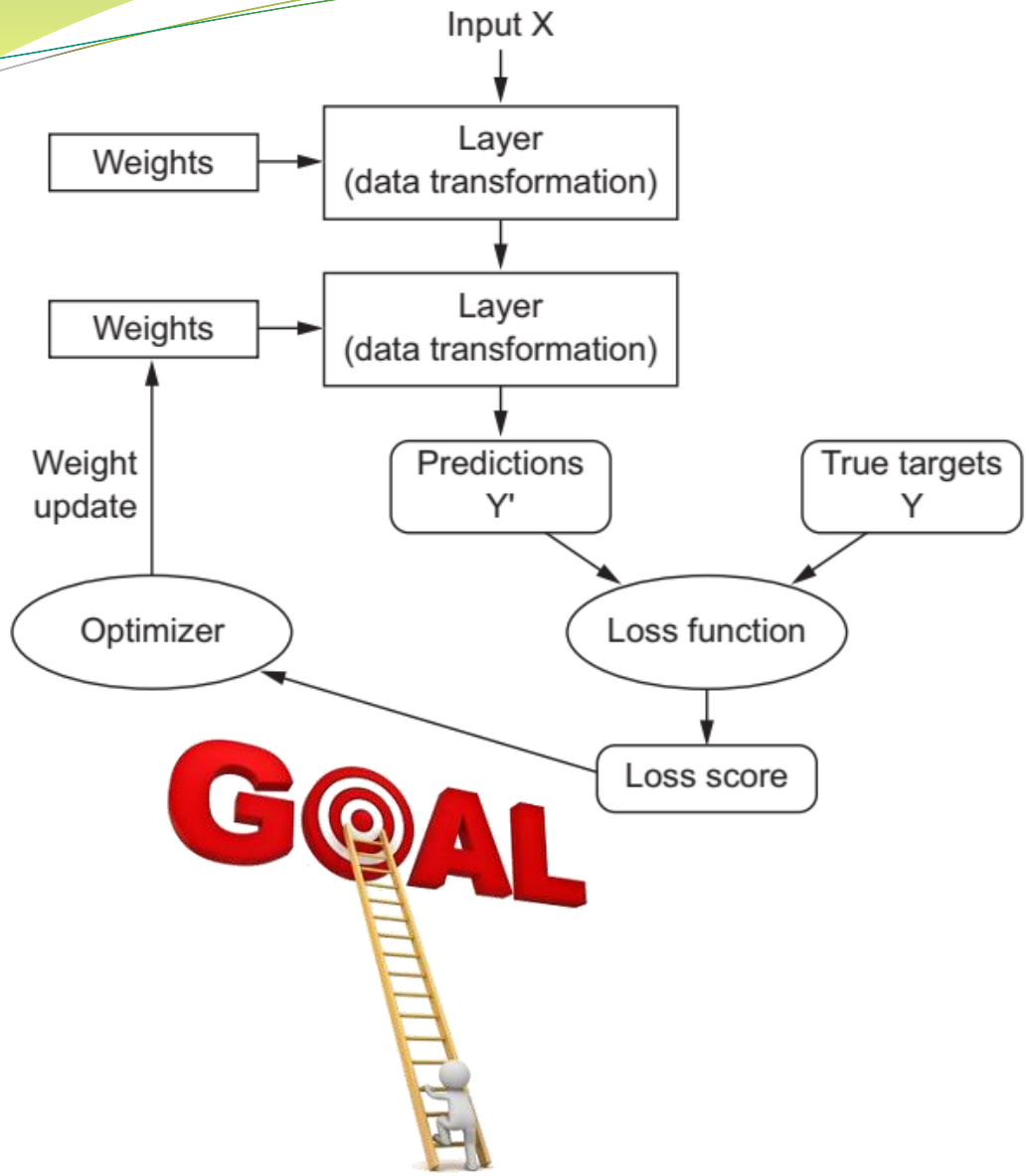
یادگیری عمیق



- تابع فعال‌سازی لایه آخر
  - این تابع قیدهای مفیدی را بر روی خروجی شبکه اعمال می‌کند

- تابع ضرر
  - این تابع باید منطبق بر مسئله مورد نظر و تابع فعال‌سازی لایه آخر باشد
  - بسیاری از اوقات نمی‌توان معیار هدف را به طور مستقیم بهینه کرد
  - لازم است بتواند برای یک minibatch به درستی محاسبه شود و مشتق‌پذیر باشد

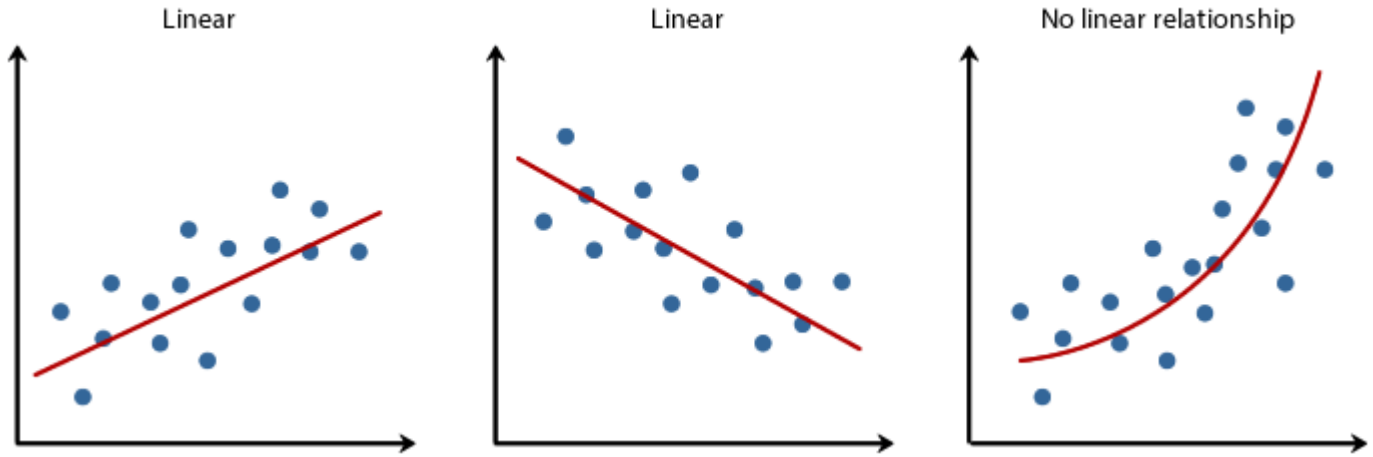
تابع ضرر



- با چه معادله‌ای y و y' را مقایسه کنیم؟
- روش‌های یادگیری ماشین عموما مبتنی بر تئوری احتمالات هستند
- به طور معمول، از رویکرد Maximum Likelihood استفاده می‌شود
  - می‌خواهیم پارامترهای مدل را طوری تنظیم کنیم که احتمال مشاهده xها از yها ماکزیمم باشد

رگرسیون

- شامل تخمین یک مقدار پیوسته است
  - به عنوان مثال، پیش‌بینی دمای فردا بر اساس داده‌های هواشناسی
- می‌توانیم از تابع فعال‌سازی خطی در لایه آخر استفاده کنیم
  - تابع ضرر مناسب چیست؟
  - با فرض توزیع نرمال خطا، میانگین مربعات خطا



$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{data}} \|\boldsymbol{y} - f(\boldsymbol{x};\boldsymbol{\theta})\|^2$$