بسم الله الرحمن الرحیم

# یادگیری عمیق

مدرس: محمدرضا محمدی

زمستان ۱۴۰۱

# الگوریتم‌های بهینه‌سازی

# Optimization Algorithms

# SGD + Momentum

### SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x -= learning_rate * dx
```

### SGD + Momentum

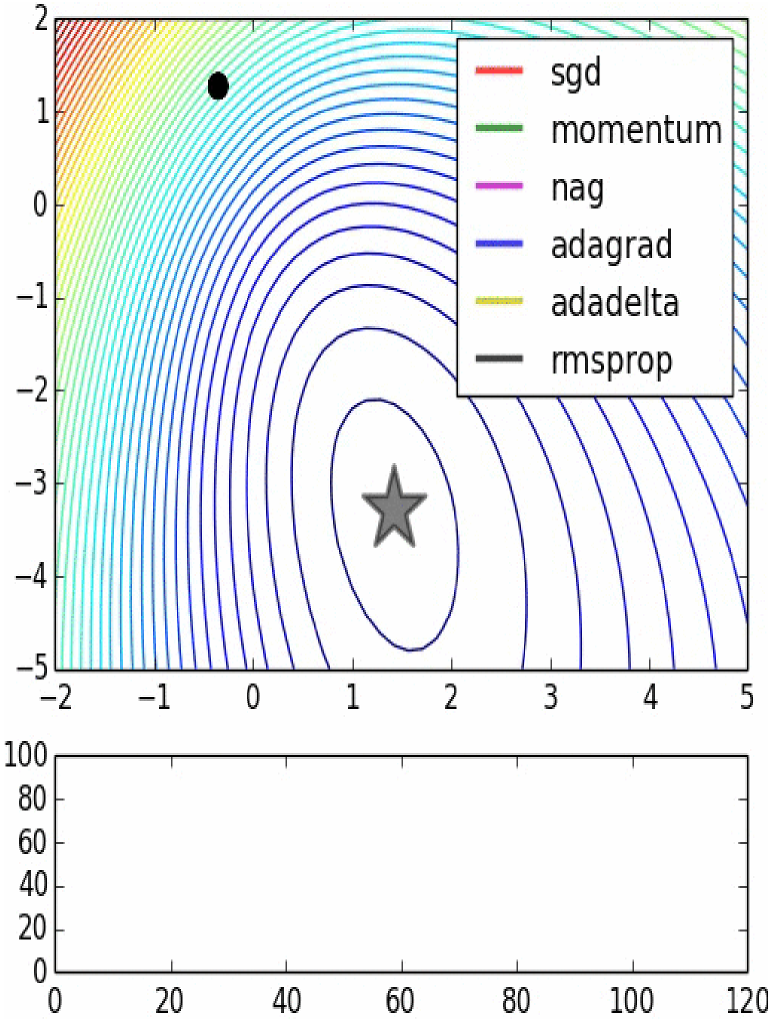
$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x -= learning_rate * vx
```

- با معادل فرض کردن گرادیان با شتاب، "سرعت" محاسبه می‌شود
- پارامتر $\rho$ اصطکاک را شبیه‌سازی می‌کند
  - به طور معمول ۰.۹ یا ۰.۹۹ است
- یک پیاده‌سازی معادل هم به صورت روبرو است

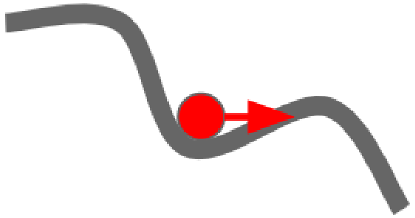
$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t)$$
$$x_{t+1} = x_t + v_{t+1}$$

# SGD + Momentum

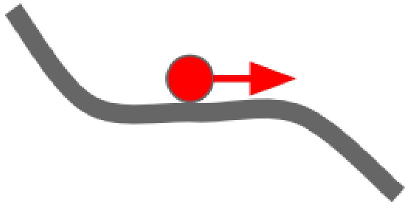



$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

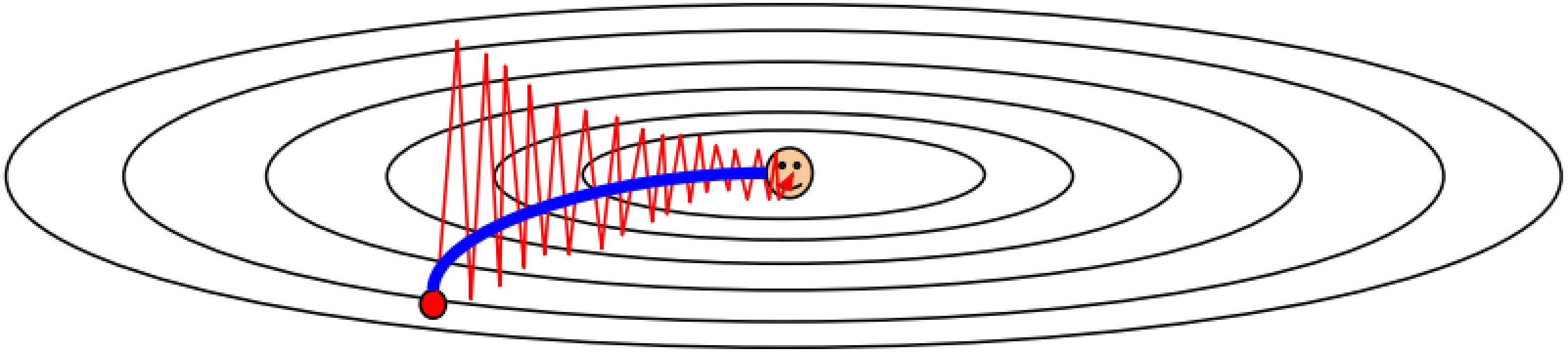
Local Minima



Saddle points



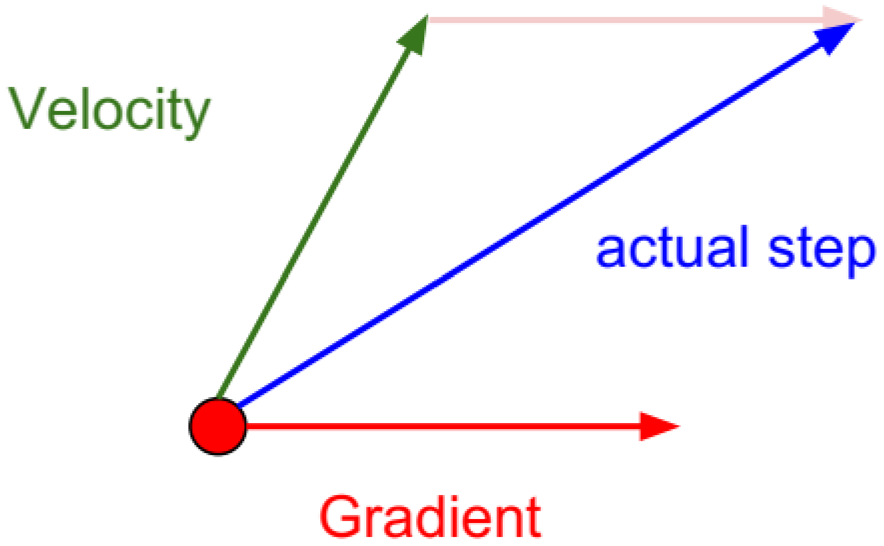
Poor Conditioning

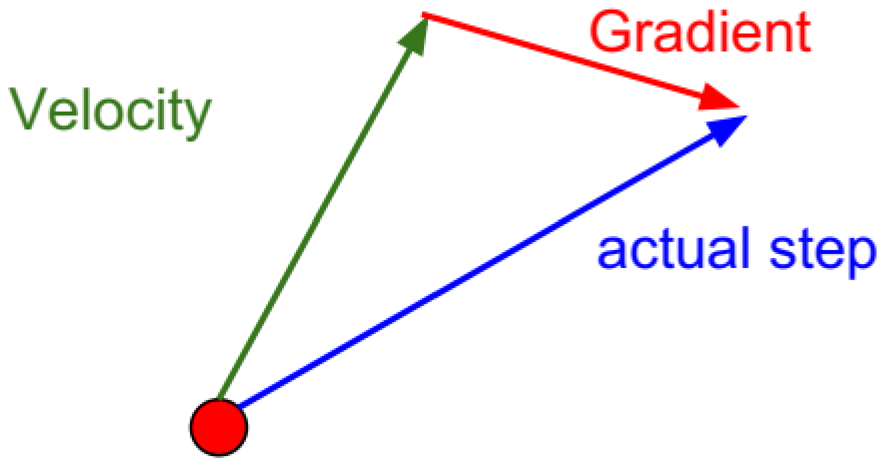


# Nesterov Momentum

Nesterov Momentum: به جلو نگاه می‌کند. گرادیان را در نقطه‌ای محاسبه می‌کند که اگر با همین سرعت حرکت کند به آنجا می‌رسد

Momentum: گرادیان و سرعت در نقطه فعلی را با هم ترکیب می‌کند تا گام به‌روزرسانی وزن‌ها را بدست بیاورد



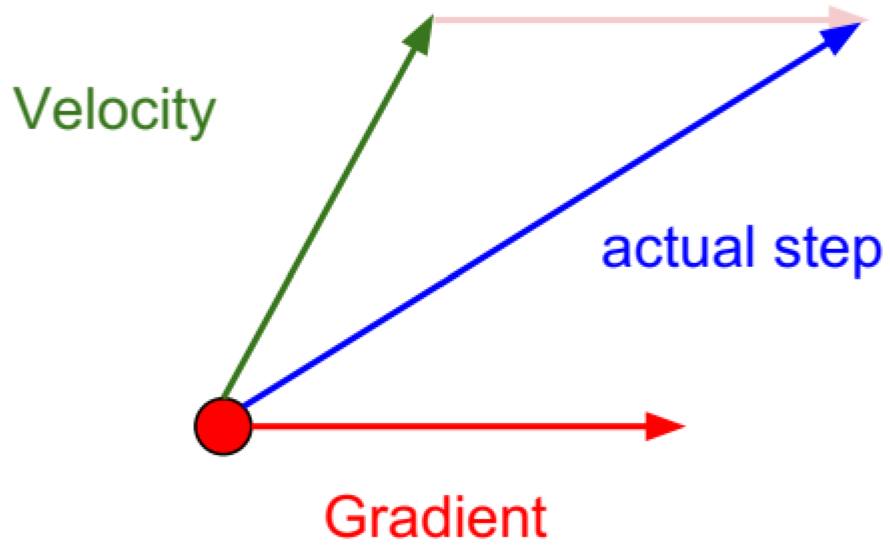



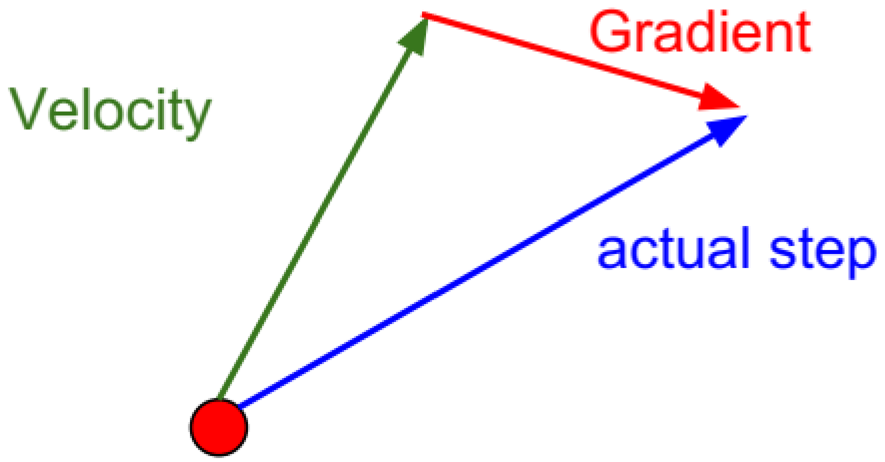
# Nesterov Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t)$$
$$x_{t+1} = x_t + v_{t+1}$$

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t \boxed{+ \rho v_t})$$
$$x_{t+1} = x_t + v_{t+1}$$

- هزینه محاسباتی دارد زیرا باید گرادیان را در یک نقطه دیگر محاسبه کند



# Nesterov Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t)$$
$$x_{t+1} = x_t + v_{t+1}$$

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t \boxed{+ \rho v_t})$$
$$x_{t+1} = x_t + v_{t+1}$$

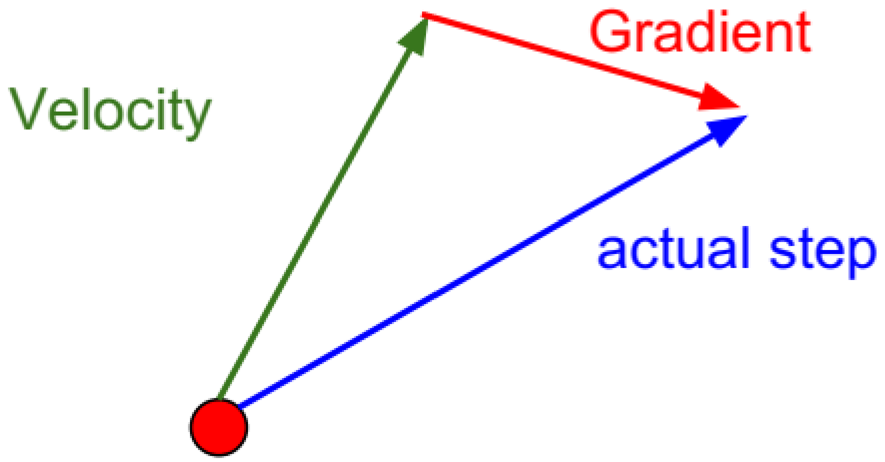
- تغییر متغیر می‌دهیم: $\tilde{x}_t \triangleq x_t + \rho v_t$

$$v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x}_t)$$

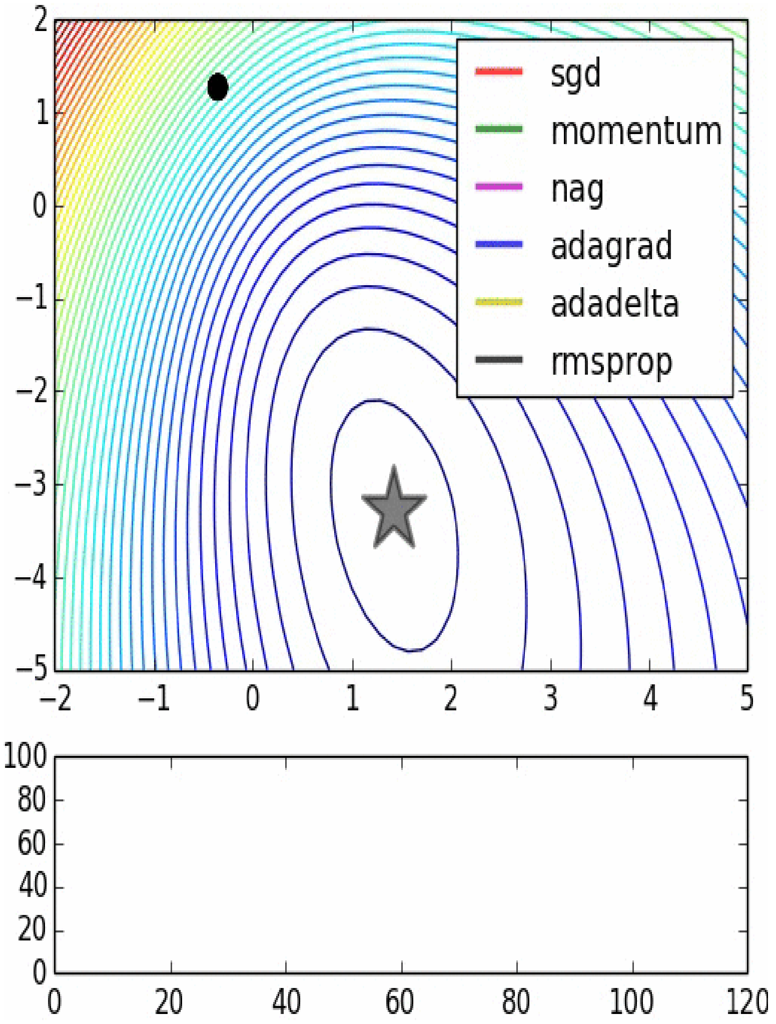
$$\tilde{x}_{t+1} - \rho v_{t+1} = \tilde{x}_t - \rho v_t + v_{t+1}$$

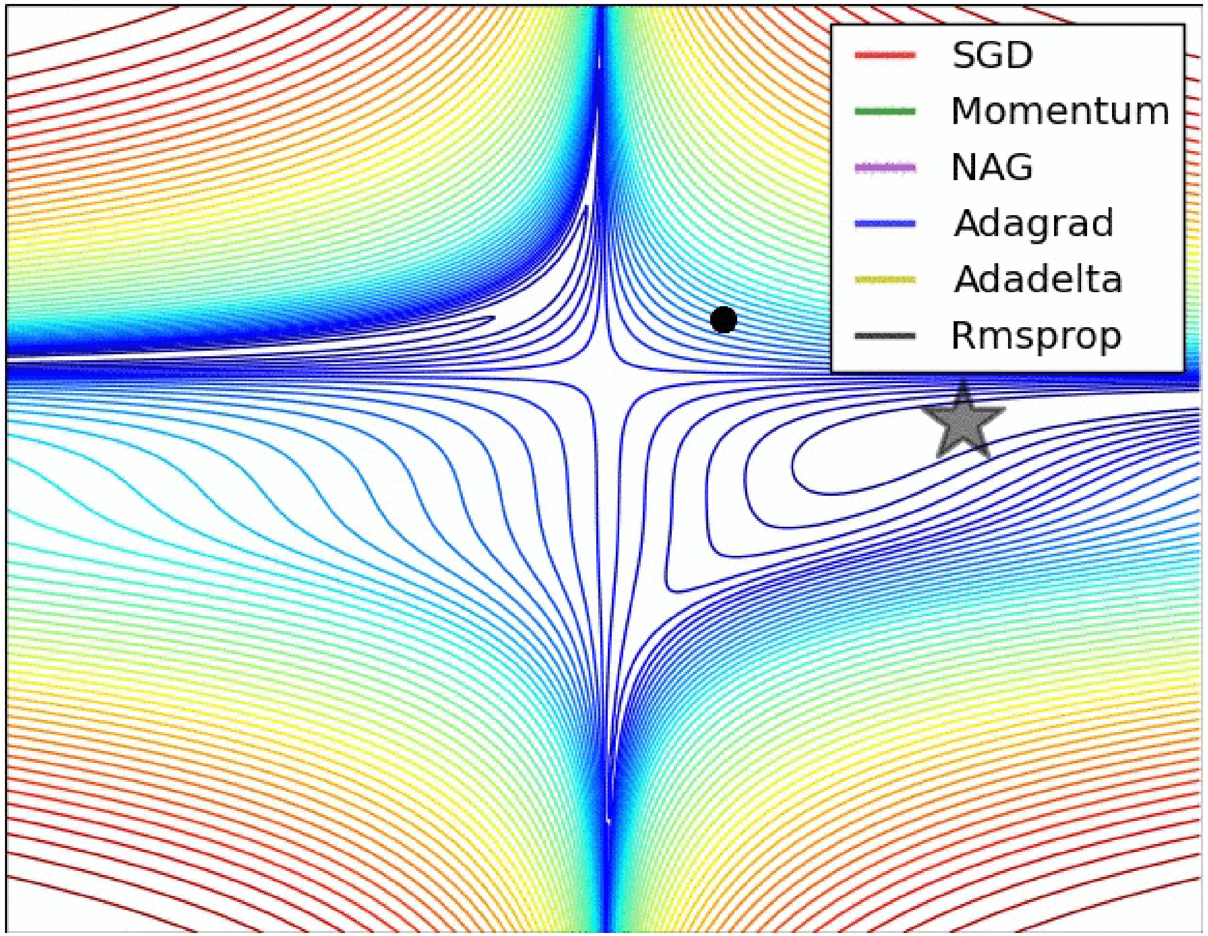
$$\tilde{x}_{t+1} = \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t)$$

```
dx = compute_gradient(x)
old_v = v
v = rho * v - learning_rate * dx
x += -rho * old_v + (1 + rho) * v
```



# Nesterov Momentum

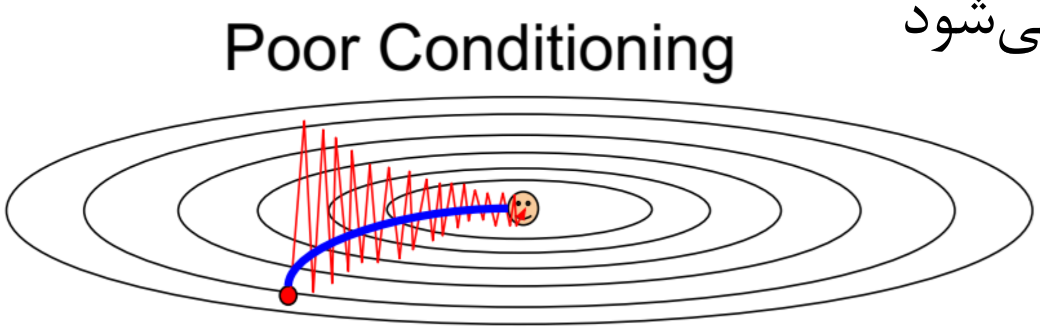



(image credits to Alec Radford)

# AdaGrad

- هر کدام از مولفه‌های گرادیان را در یک ضریب مستقل که بر اساس مقادیر گذشته بدست می‌آید ضرب می‌کند
- "نرخ یادگیری بر پارامتر" یا "نرخ یادگیری تطبیقی" نامیده می‌شود
  - تغییر در جهت عمیق کند می‌شود
  - تغییر در جهت مسطح شتاب می‌گیرد
  - نرخ آموزش در طول زمان به سمت صفر کاهش می‌یابد




```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

# RMSProp

```python
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

```python
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

# Adam (تقریبا)

```python
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Momentum

AdaGrad / RMSProp

- گشتاورهای اول و دوم در گام‌های اولیه خیلی کوچک خواهند بود

# Adam

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

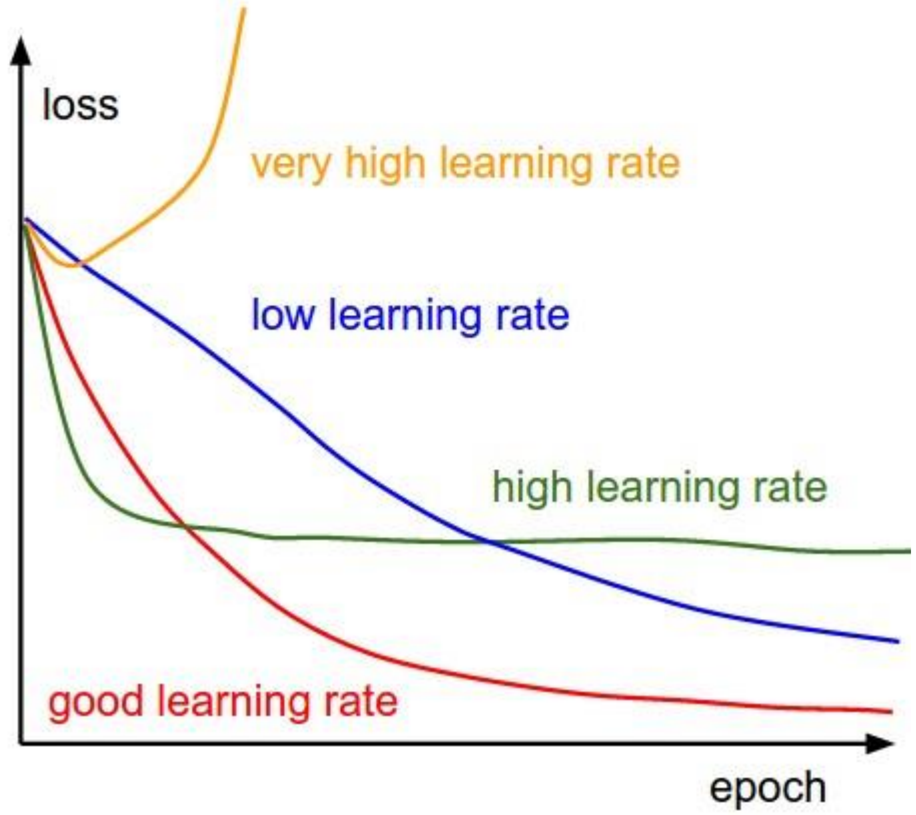
Bias correction

AdaGrad / RMSProp

- بهینه‌ساز Adam با پارامترهای beta1 = 0.9 و beta2 = 0.999 و نرخ آموزش برابر با 1e-3 یا 5e-4 یک نقطه شروع خوب برای بسیاری از مدل‌ها است

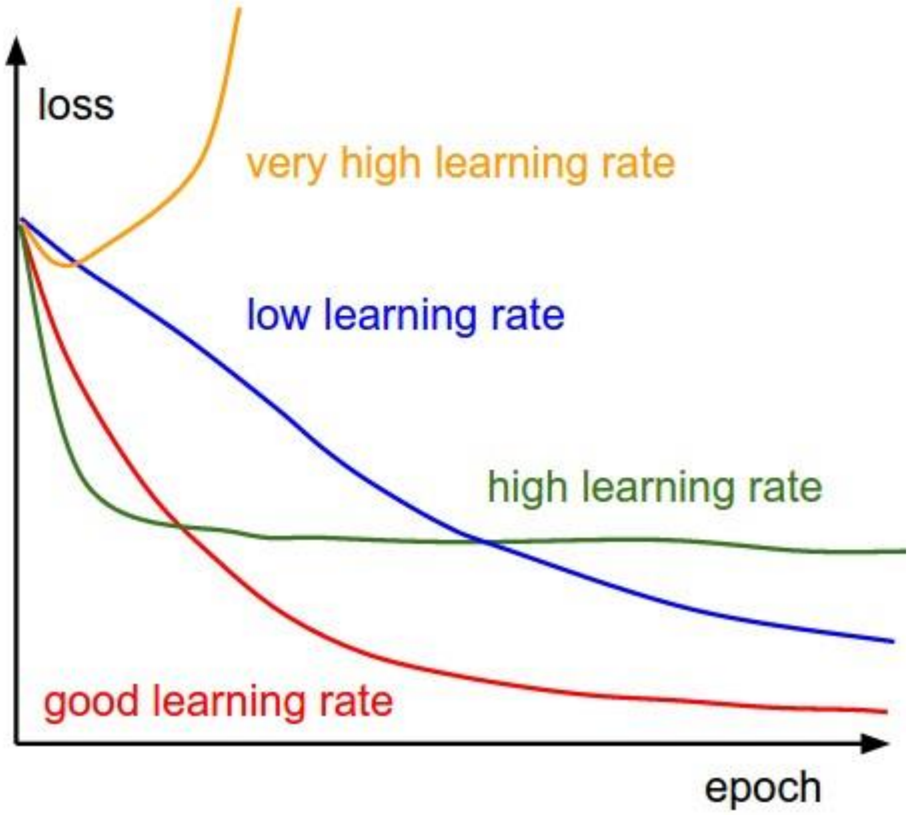
# نرخ آموزش

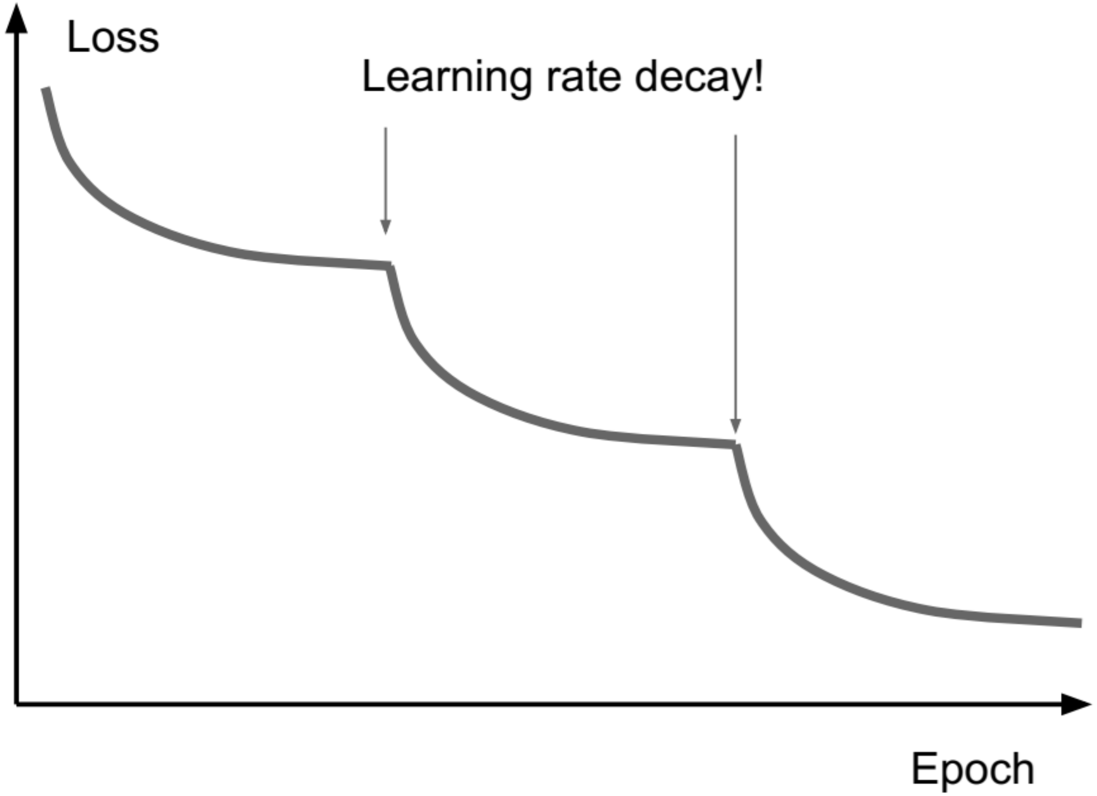
- تمام بهینه‌سازهای SGD، SGD+Momentum، Adagrad، RMSProp و Adam دارای اَبَرپارامتر نرخ آموزش هستند
- می‌توانیم نرخ آموزش را در طول زمان کاهش دهیم
  - step decay:
    - نرخ آموزش بعد از چند epoch در ضریبی مانند ۰.۵ ضرب می‌شود
  - exponential decay:
    - $\alpha = \alpha_0 e^{-kt}$
  - 1/t decay:
    - $\alpha = \alpha_0/(1 + kt)$



# کاهش نرخ آموزش

- برای SGD+Momentum حیاتی‌تر است، برای Adam کمتر رایج است



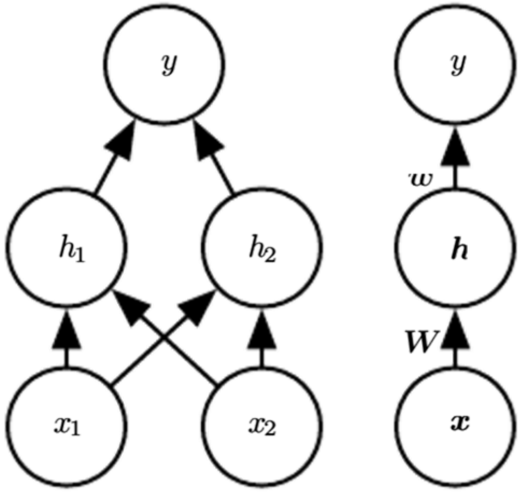
# محاسبه گرادیان

$$h_i = g\left(\boldsymbol{x}^T \boldsymbol{W}_{:,i} + c_i\right) = f^{(1)}(\boldsymbol{x})$$

$$y = f^{(2)}\left(f^{(1)}(\boldsymbol{x})\right) = f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^T \max\{0, \boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{c}\} + b$$

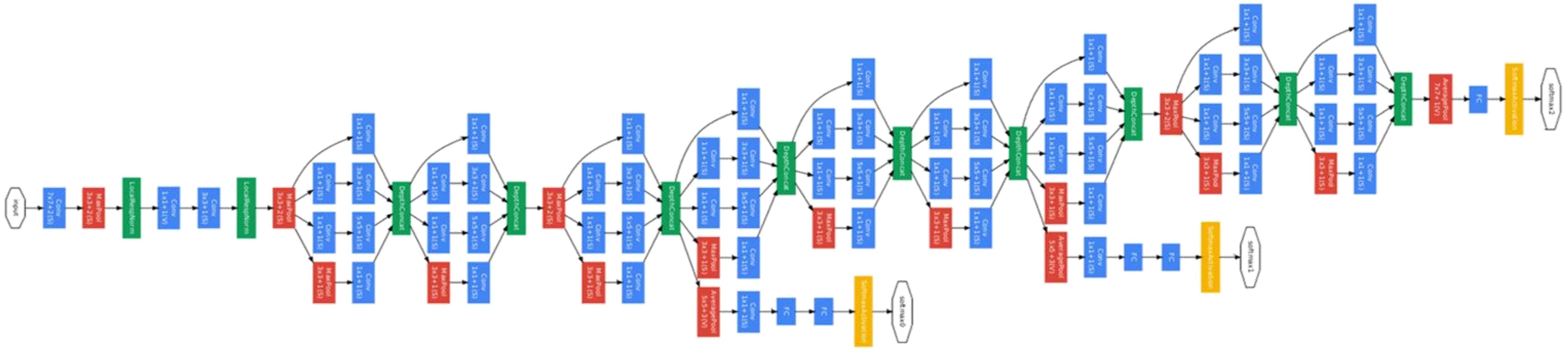
$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum \left(f^*(\boldsymbol{x}) - f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b)\right)^2$$

- اگر بتوانیم $\frac{\partial J}{\partial \boldsymbol{W}}$ ، $\frac{\partial J}{\partial \boldsymbol{c}}$ ، $\frac{\partial J}{\partial \boldsymbol{w}}$ و $\frac{\partial J}{\partial b}$ را محاسبه کنیم، می‌توانیم پارامترهای مدل را آموزش بدهیم
- اگر بخواهیم تابع ضرر را تغییر بدهیم؟
  - نیاز است تا معادلات دوباره از ابتدا استخراج شوند



# محاسبه گرادیان

- بسیار خسته‌کننده است و نیاز به محاسبات ماتریسی فوق‌العاده زیادی دارد



# پس‌انتشار (Backpropagation)

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

e.g., $x=-2,\ y=5,\ z=-4$

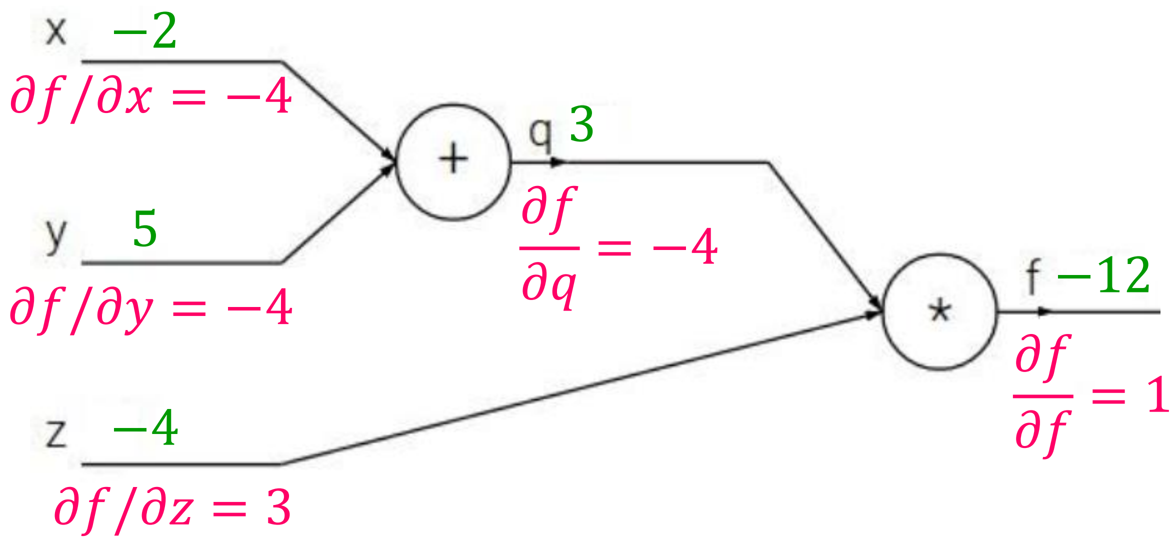
$q = x + y \qquad \Rightarrow f = qz$

we want $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ and $\frac{\partial f}{\partial z}$

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

Chain rule

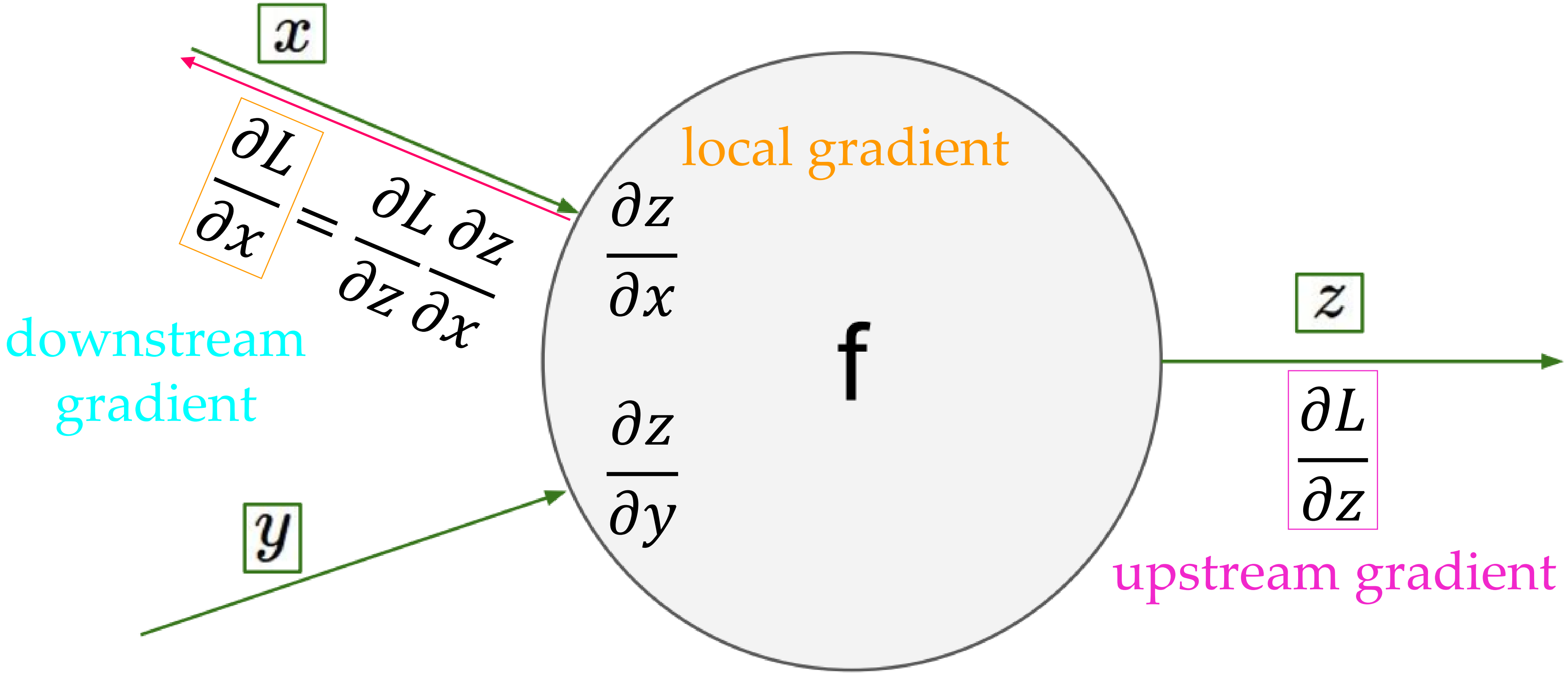
$$\frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1 \qquad \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} \qquad \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$





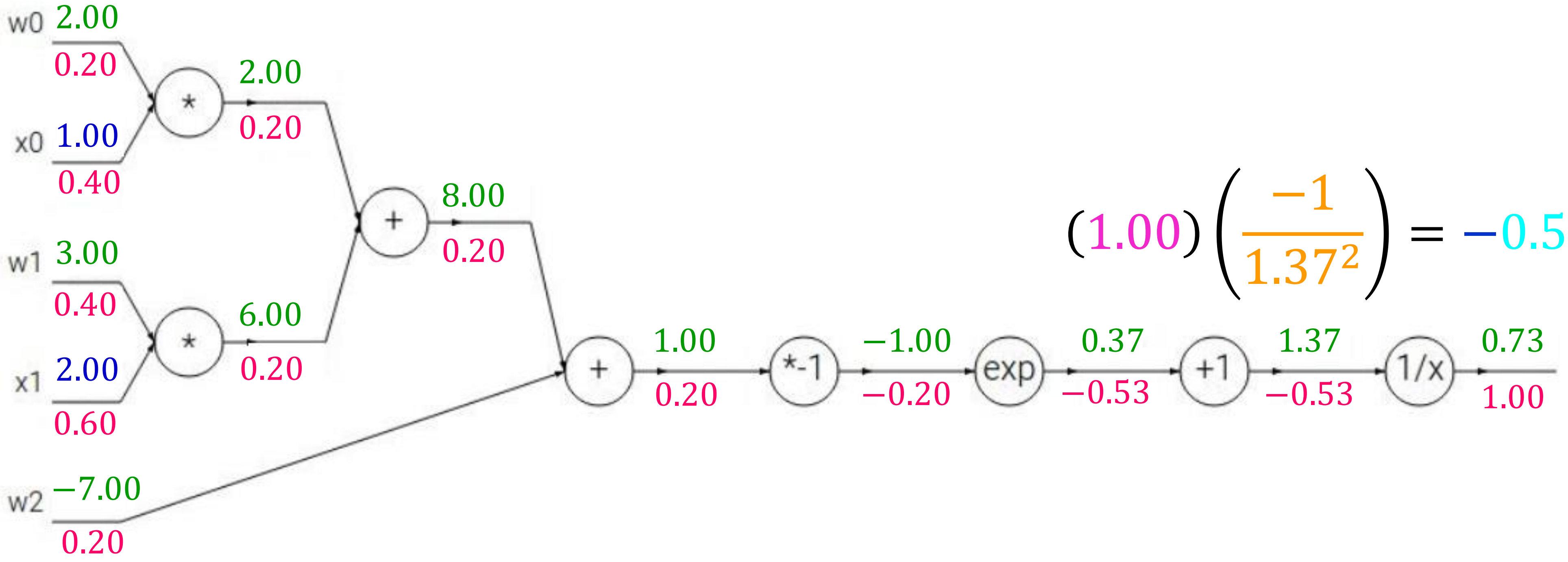

Based on cs231n by Fei-Fei Li & Justin Johnson & Serena Yeung https://cs231n.github.io/optimization-2/

# پس‌انتشار (Backpropagation)



# مثال: Linear + Sigmoid

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

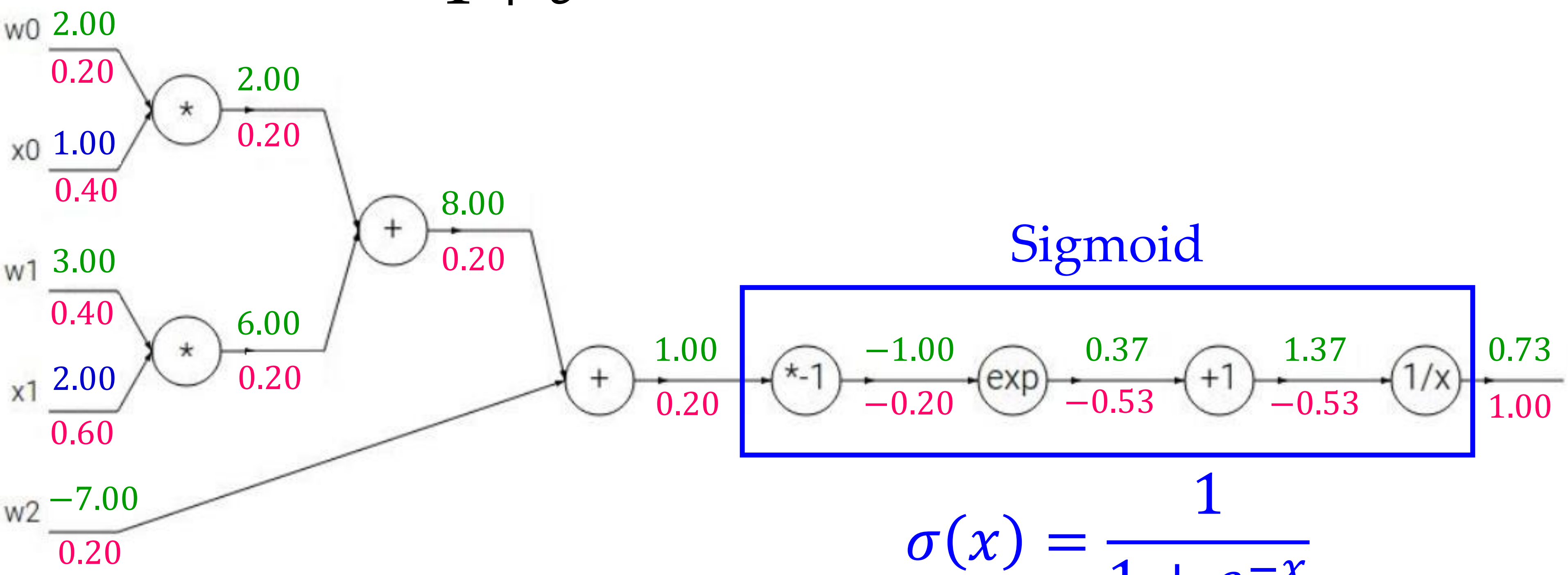



$$(1.00)\left(\frac{-1}{1.37^2}\right) = -0.53$$

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

# مثال: Linear + Sigmoid

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




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

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