
بسم الله الرحمن الرحيم

مبانی یادگیری عمیق

مدرس: محمدرضا محمدی

۱۴۰۰

گرادیان کاهشی تصادفی (SGD)

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(s_i, y_i)$$

$$\nabla_W L = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(s_i, y_i)$$

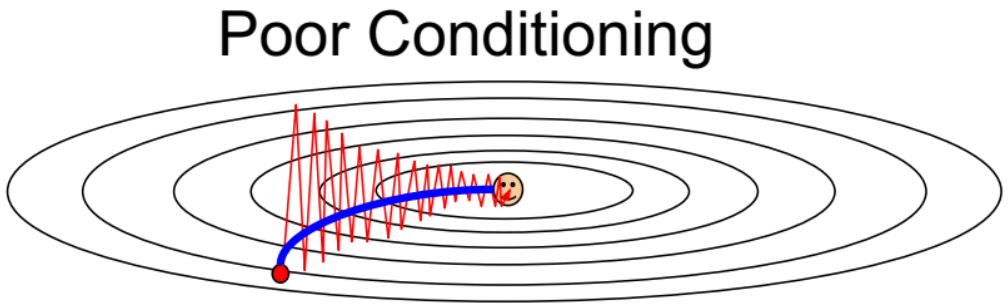
$$s_i = f(x_i, W)$$

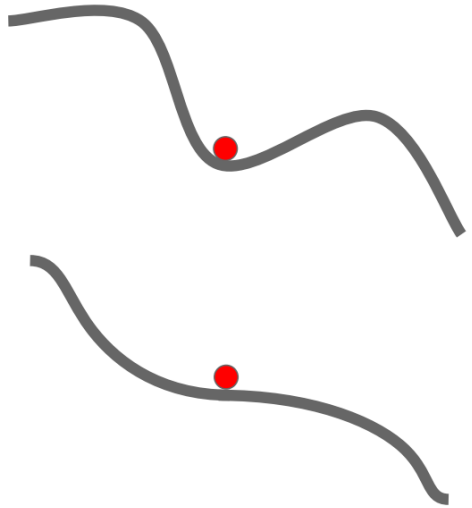
$$W = W - \eta \nabla_W L$$

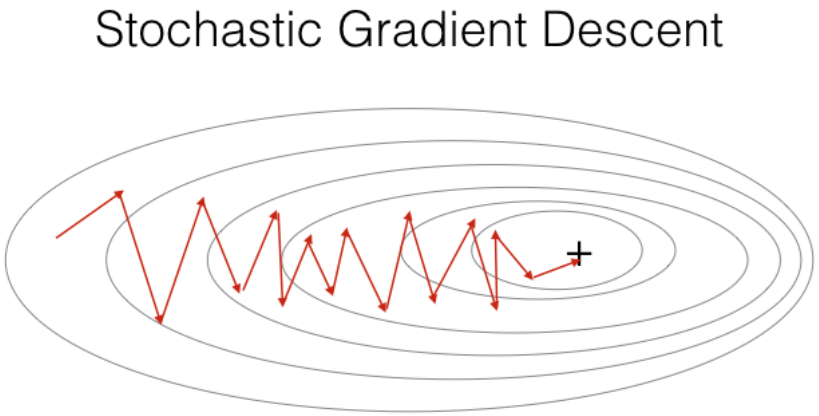
- محاسبه مجموع کامل اگر N بزرگ باشد بسیار پر هزینه است
- آن را با استفاده از یک minibatch از نمونه‌ها تقریب می‌زنیم
  - ۳۲/۶۴/۱۲۸ متداول هستند

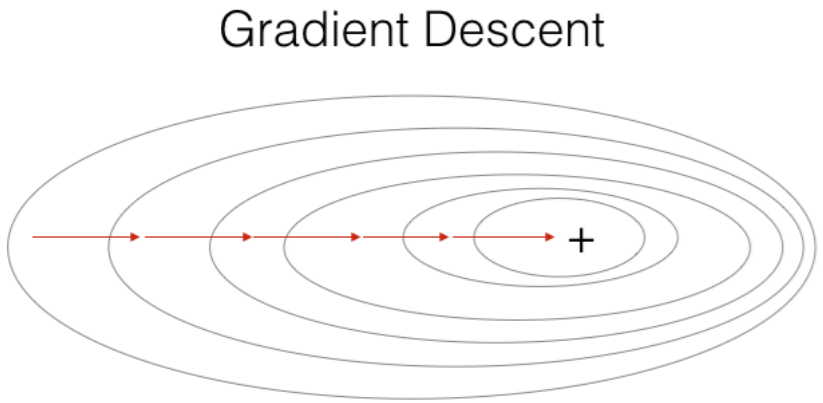
```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

مشکلات SGD


Poor Conditioning




Stochastic Gradient Descent


Gradient Descent

SGD + Momentum

- تابع ضرر می‌تواند مشابه با ارتفاع یک زمینه تپه‌ای در نظر گرفته شود
- مقداردهی اولیه تصادفی برای پارامترها معادل با قرار دادن یک توپ با سرعت اولیه صفر در یک مکان است
- فرآیند بهینه‌سازی می‌تواند معادل با حرکت توپ به سمت عمیق‌ترین نقطه این زمین در اثر جاذبه مدل‌سازی شود

SGD + Momentum

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
```

SGD

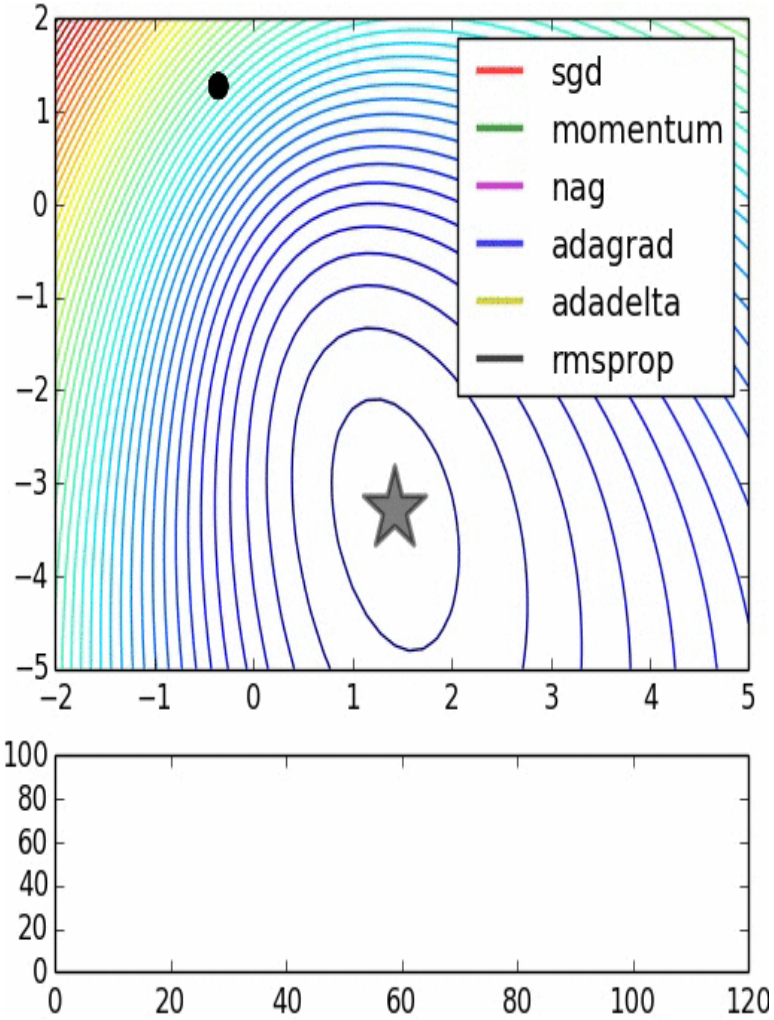
$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

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

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

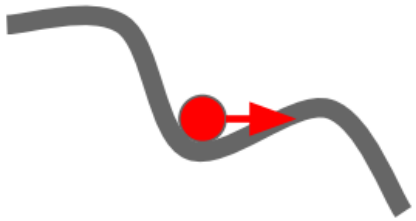
$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t)$$
$$x_{t+1} = x_t + v_{t+1}$$

SGD + Momentum

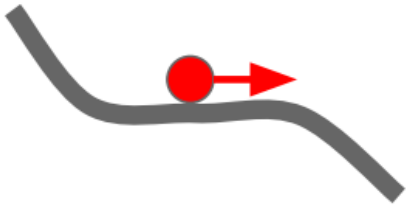


$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

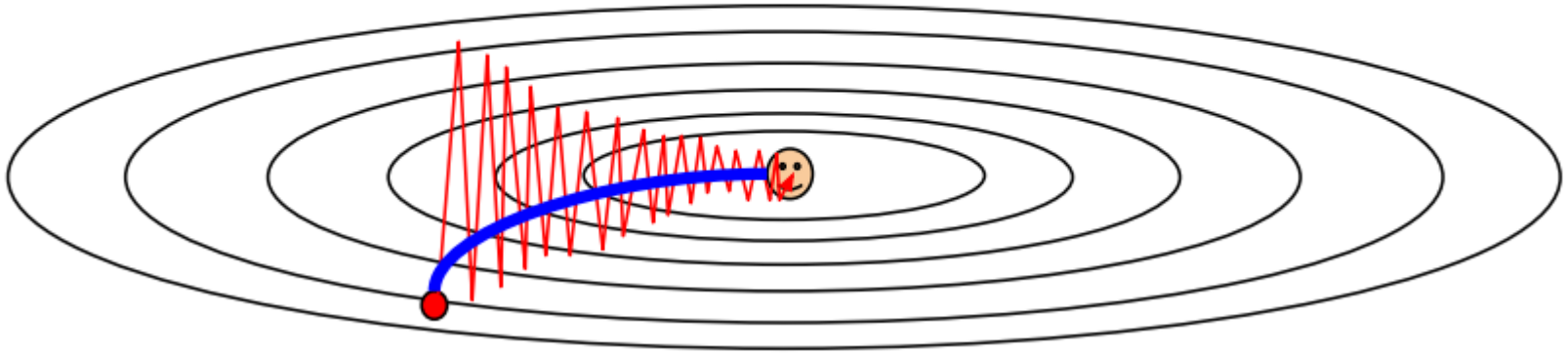
Local Minima



Saddle points

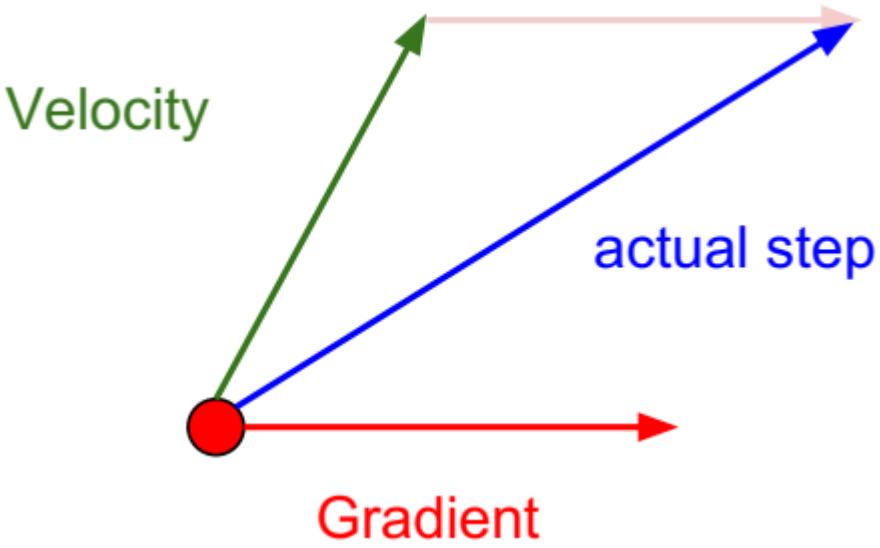


Poor Conditioning

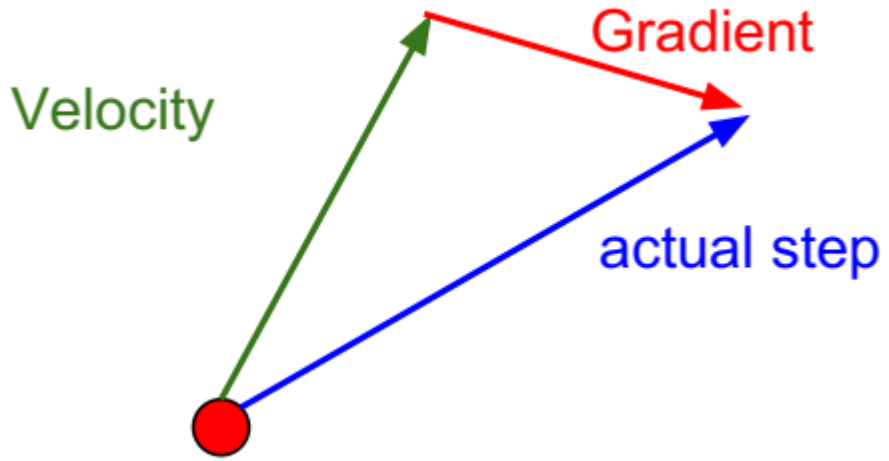


Nesterov Momentum

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



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

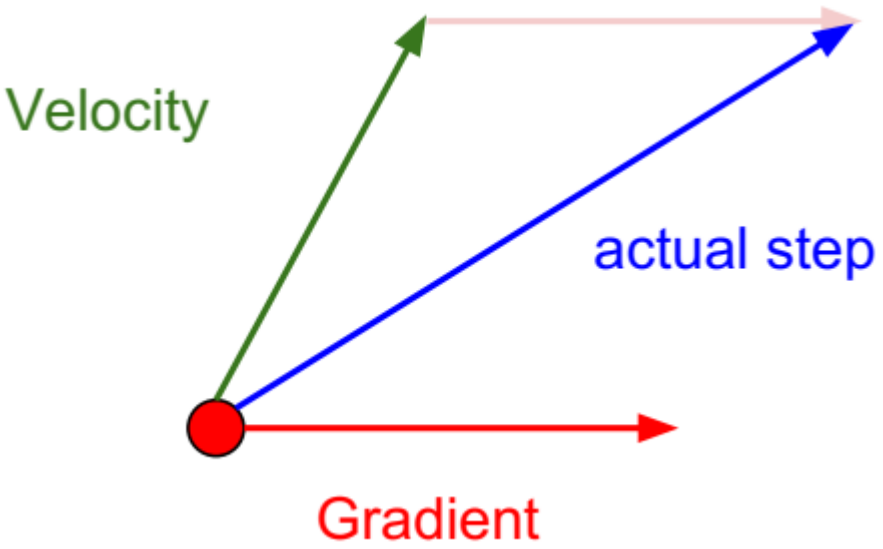


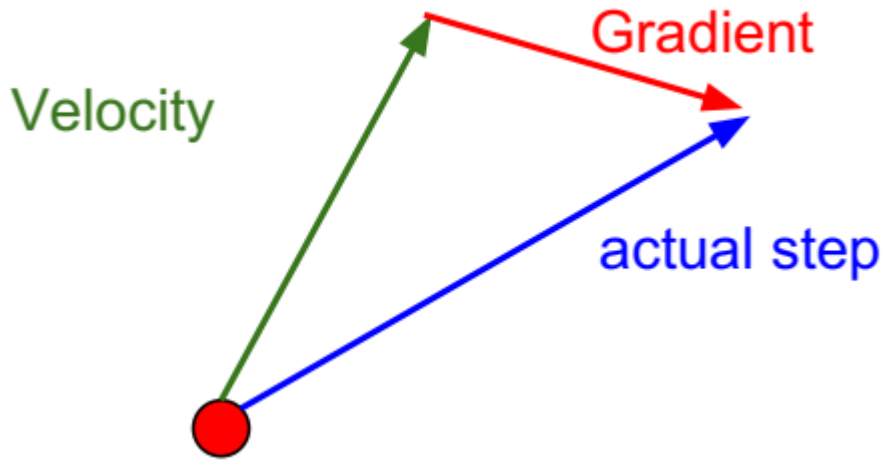
Nesterov Momentum

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

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

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





Nesterov Momentum

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

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

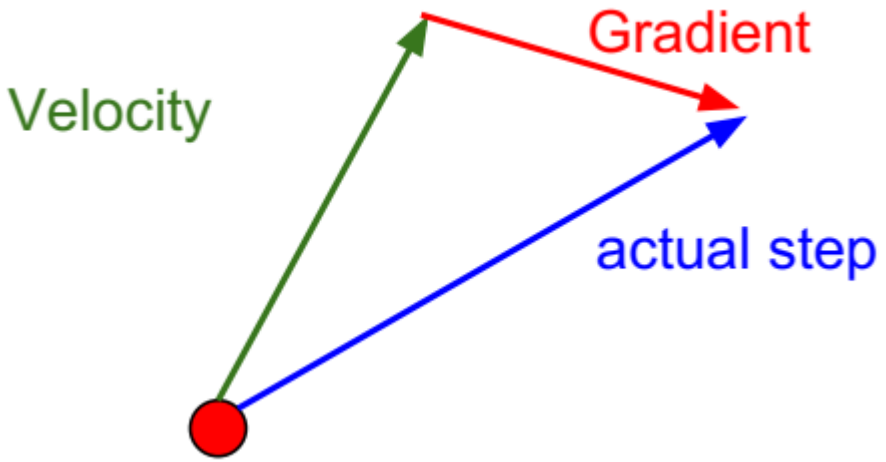
- تغییر متغیر می‌دهیم: $\tilde{x}_t \triangleq x_t + \rho v_t$

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

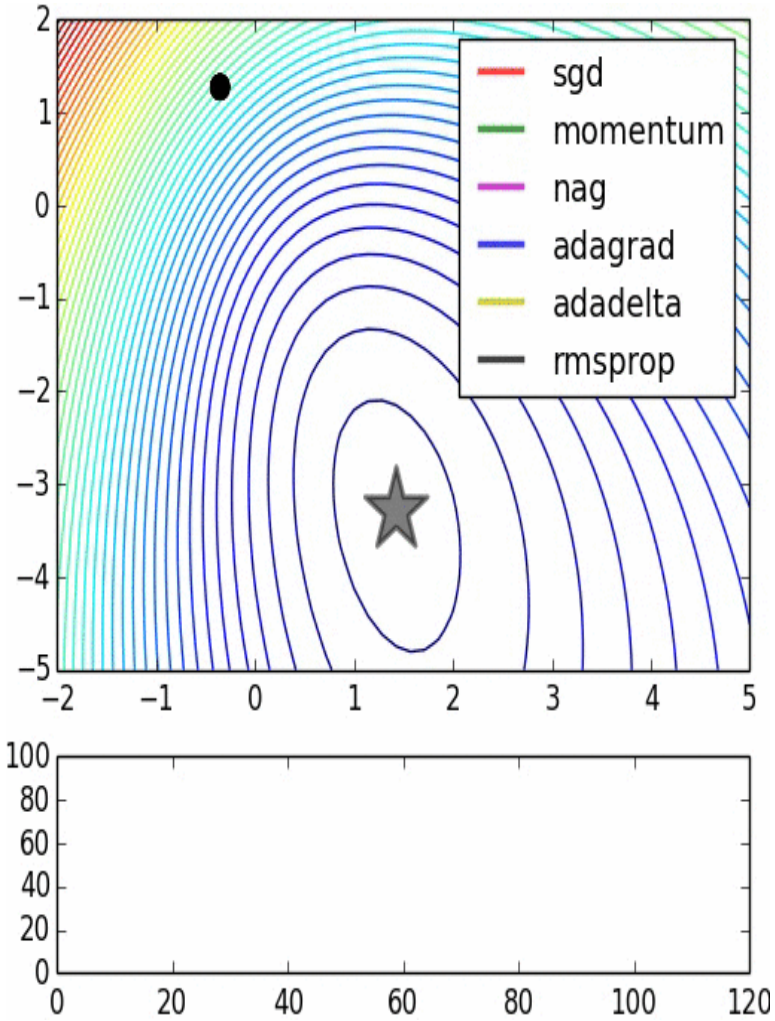
$$\tilde{x}_{t+1} - \rho v_{t+1} = \tilde{x}_t - \rho v_t + v_{t+1}$$

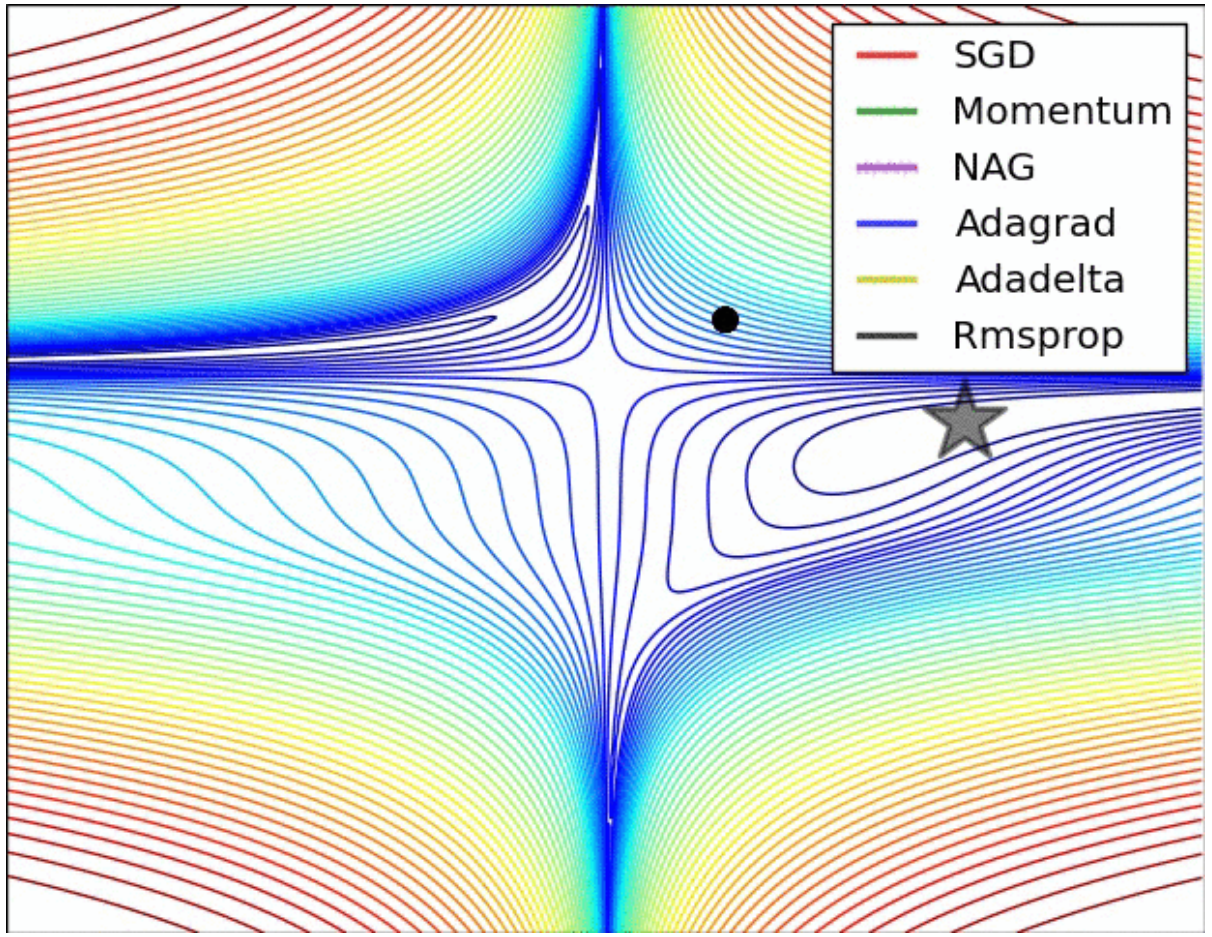
$$\tilde{x}_{t+1} = \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t)$$

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



Nesterov Momentum

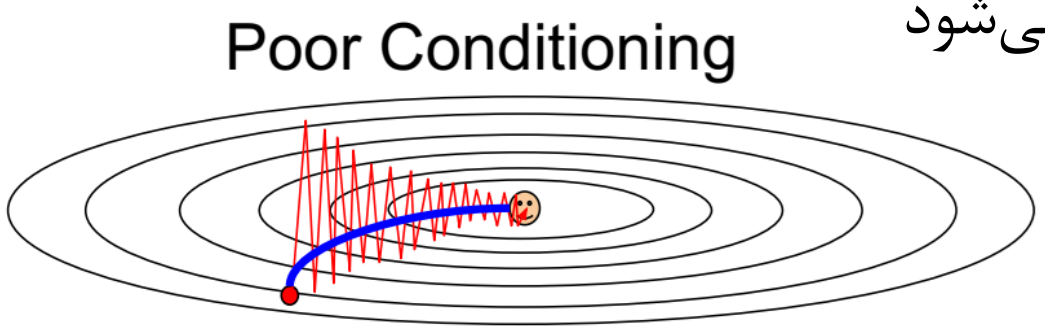




(image credits to Alec Radford)

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

```
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)
```

```
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)

```
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