بهینه‌سازی

سید ناصر رضوی www.snrazavi.ir
۱۳۹۷

یادآوری

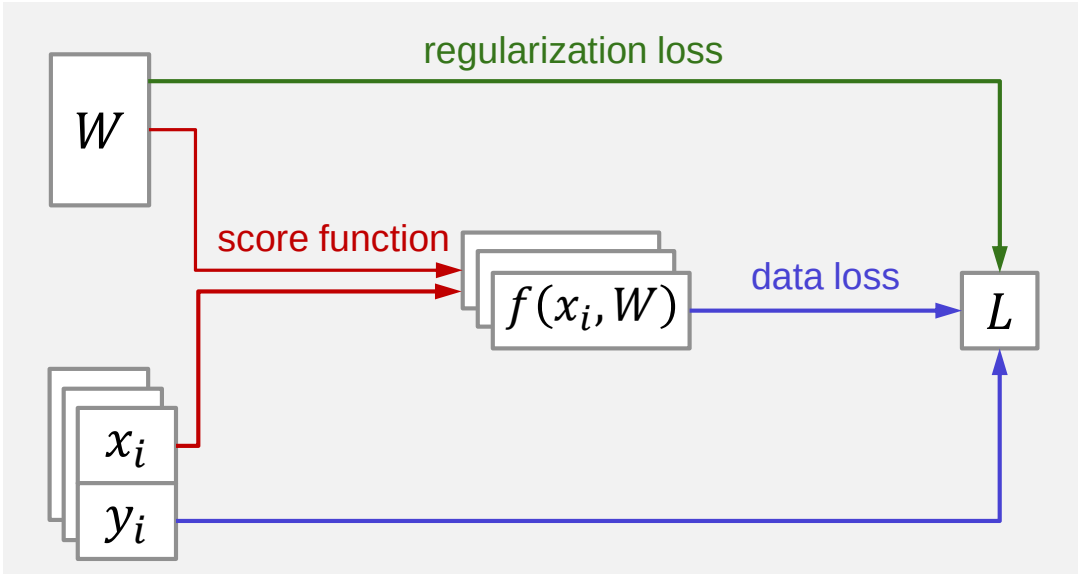
- یک مجموعه از داده‌های آموزشی به صورت (x,y)
- یک تابع امتیاز: $s = f(x;W) = Wx$
- یک تابع هزینه:

هدف. کمینه‌سازی تابع هزینه به منظور یافتن مقادیر W

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$ cross-entropy loss (Softmax)

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$ hinge loss (SVM)

$$L = \left(\frac{1}{N}\sum_{i=1}^{N} L_i\right) + \lambda R(W)$$ Full loss



استراتژی اول: جستجوی تصادفی (یک ایده بسیار بد)


```
best_loss = float("inf")
for i in xrange(1000):
    W = np.random.randn(10, 3073) * 0.0001
    loss = L(X_train, y_train, W)
    if loss < best_loss:
        best_loss = loss
        best_W = W
    print "In attempt %d the loss was %f, best %f" % (i, loss, best_loss)
```

```
In attempt 0 the loss was 9.401632, best 9.401632
In attempt 1 the loss was 8.959668, best 8.959668
In attempt 2 the loss was 9.044034, best 8.959668
In attempt 3 the loss was 9.278948, best 8.959668
In attempt 4 the loss was 8.857370, best 8.857370
In attempt 5 the loss was 8.943151, best 8.857370
```

یک نمونه از اجرای الگوریتم جستجوی تصادفی

آزمایش جستجوی تصادفی


```
# assume X_test is (3073, 10000), y_test is (10000, 1)
scores = np.dot(best_W, X_test)
# find the index with max score in each column (the predicted class)
y_pred = np.argmax(scores, axis=0)
# calculate accuracy (fraction of predictions that are correct)
accuracy = np.mean(y_pred == y_test)
# returns 0.1555 as accuracy
```

دقت ۱۵/۵ درصد! خیلی بد نیست!

(اما دقت بهترین راه‌حل در حدود ۹۵ درصد است)

گرادیان کاهشی

استراتژی دوم: دنبال کردن شیب

- مشتق تابع. در یک فضای ۱- بُعدی

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- در یک فضای چند بُعدی، گرادیان تابع برداری است از مشتق‌های جزئی.

محاسبه عددی گرادیان

W	$W + h$		dW
0.34	0.34	+ 0.0001	?
-1.11	-1.11		?
0.78	0.78		?
0.12	0.12		?
0.55	0.55		?
2.81	2.81		?
-3.10	-3.10		?
-1.50	-1.50		?
0.33	0.33		?
...	...		...

Loss 1.25347

Loss 1.25322

محاسبه عددی گرادیان

W
0.34
-1.11
0.78
0.12
0.55
2.81
-3.10
-1.50
0.33
...

Loss 1.25347

$W + h$
0.34
-1.11
0.78
0.12
0.55
2.81
-3.10
-1.50
0.33
...

+ 0.0001

Loss 1.25322

dW
-2.50
?
?
?
.
?
...

(1.25322 - 1.25347) / 0.0001

= -2.5

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

محاسبه عددی گرادیان

W	$W + h$		dW
0.34	0.34		-2.50
-1.11	-1.11	+ 0.0001	?
0.78	0.78		?
0.12	0.12		?
0.55	0.55		?
2.81	2.81		?
-3.10	-3.10		?
-1.50	-1.50		?
0.33	0.33		?
...	...		...

Loss 1.25347

Loss 1.25353

محاسبه عددی گرادیان

W
0.34
-1.11
0.78
0.12
0.55
2.81
-3.10
-1.50
0.33
...

Loss 1.25347

$W + h$
0.34
-1.11
0.78
0.12
0.55
2.81
-3.10
-1.50
0.33
...

Loss 1.25353

+ 0.0001

dW
-2.50
0.60
?
?
.
?
...

(1.25353 - 1.25347) / 0.0001

= 0.6

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

محاسبه عددی گرادیان

W	$W + h$		dW
0.34	0.34		-2.50
-1.11	-1.11		0.60
0.78	0.78	+ 0.0001	?
0.12	0.12		?
0.55	0.55		?
2.81	2.81		?
-3.10	-3.10		?
-1.50	-1.50		?
0.33	0.33		?
...	...		...

Loss 1.25347

Loss 1.25347

محاسبه عددی گرادیان

W
0.34
-1.11
0.78
0.12
0.55
2.81
-3.10
-1.50
0.33
...

Loss 1.25347

$W + h$
0.34
-1.11
0.78
0.12
0.55
2.81
-3.10
-1.50
0.33
...

Loss 1.25347

+ 0.0001

dW
-2.50
0.60
0.00
?
.
?
...

(1.25347 - 1.25347) / 0.0001

= 0.0

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

محاسبه عددی گرادیان


```
def eval_numerical_gradient(f, x):

    fx = f(x)
    grad = np.zeros(x.shape)
    h = 0.00001

    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
    while not it.finished:

        ix = it.multi_index
        old_value = x[ix]
        x[ix] += h
        fxh = f(x)  # evaluate f(x + h)
        x[ix] = old_value

        grad[ix] = (fxh - fx) / h  # compute the partial derivative
        it.iternext()  # step to next dimension

return grad
```

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- معایب.
  - تقریبی
  - بسیار زمان‌بر

محاسبه گرادیان به صورت تحلیلی

- تابع هزینه یک تابع از پارامترهای W است.
  - محاسبه گرادیان به صورت عددی چندان هوشمندانه به نظر نمی‌رسد.

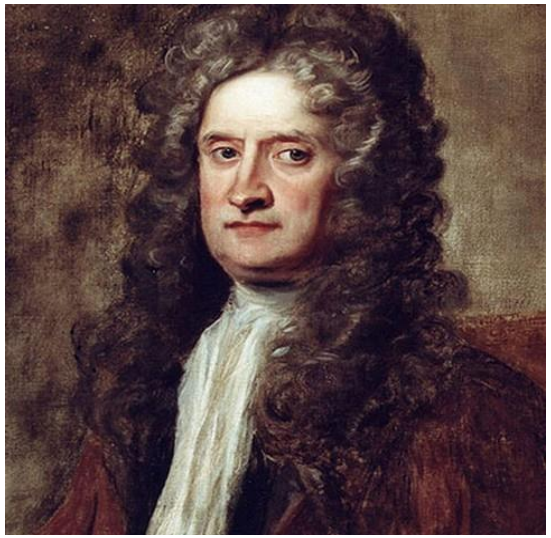
$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

$$\nabla_W L = \cdots$$

ویلهلم لایب‌نیتس (۱۶۴۶ – ۱۷۱۶)



ایزاک نیوتن (۱۶۴۲ – ۱۷۲۷)

محاسبه گرادیان به صورت تحلیلی

- تابع هزینه یک تابع از پارامترهای W است.
  - محاسبه گرادیان به صورت عددی چندان هوشمندانه به نظر نمی‌رسد.

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

$$\nabla_W L = \cdots$$

محاسبه گرادیان به صورت تحلیلی

W

0.34
-1.11
0.78
0.12
0.55
2.81
-3.10
-1.50
0.33
...

Loss 1.25347

$$\nabla_W L = \cdots$$

تابعی از داده‌ها و پارامترهای W

dW

-2.50
0.60
0.00
0.20
0.70
-0.50
1.10
1.30
-2.10
...

بررسی گرادیان

- به طور خلاصه.
  - گرادیان عددی: تقریبی، زمان‌بر، پیاده‌سازی آسان!
  - گرادیان تحلیلی: دقیق، سریع، امکان بروز خطا در پیاده‌سازی!

- در عمل.
  - همیشه از گرادیان تحلیلی استفاده می‌کنیم.
  - اما به منظور اطمینان از صحت پیاده‌سازی، گرادیان تحلیلی را با گرادیان عددی مقایسه می‌کنیم.

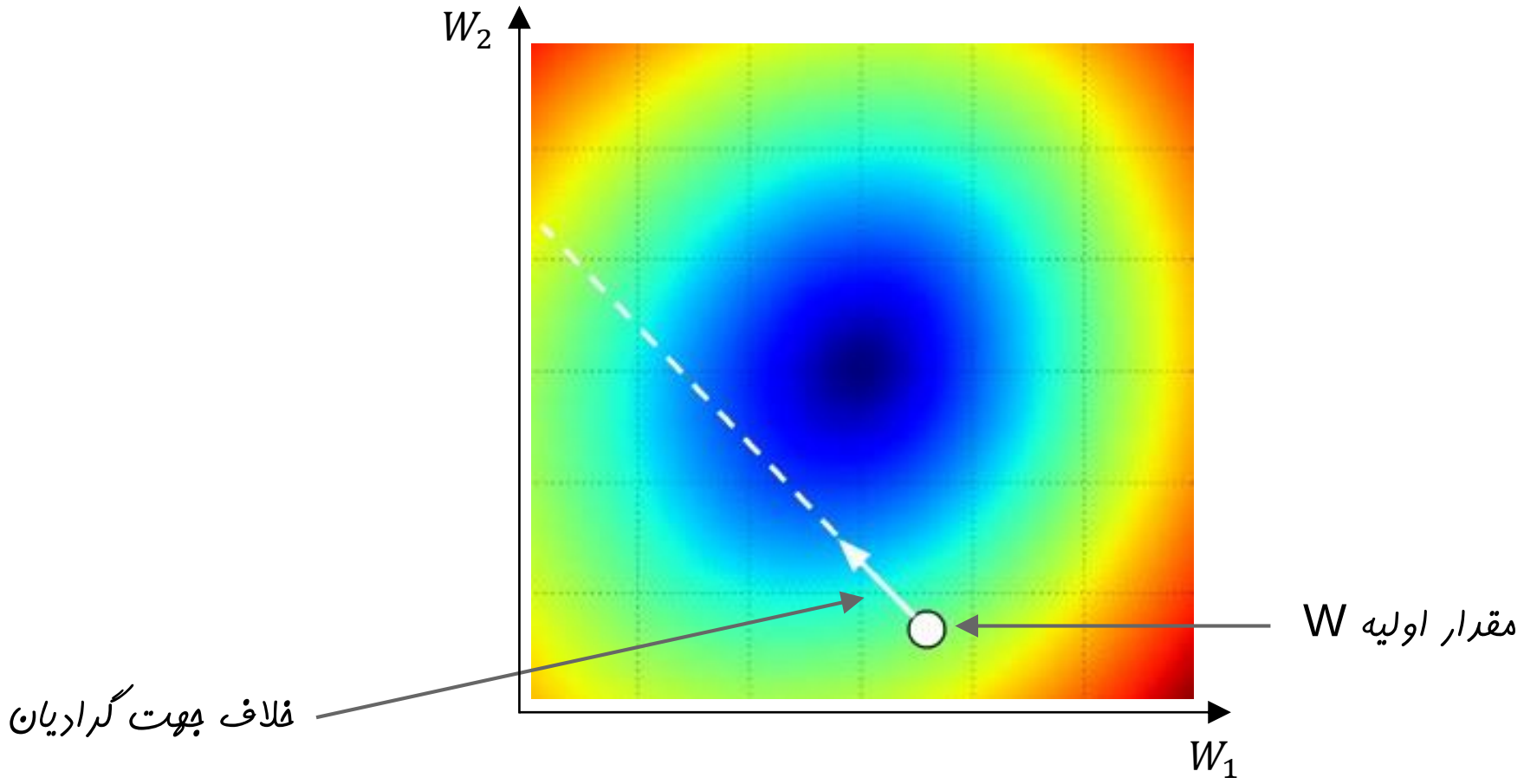
بررسی گرادیان

الگوریتم گرادیان کاهشی


```
# Vanilla Gradient Descent

while True:
    gradient = evaluate_gradient(loss_fun, data, weights)
    weights += -step_size * gradient  # weight update
```

الگوریتم گرادیان کاهشی



یک نسخه کاراتر از الگوریتم گرادیان کاهشی

- گرادیان کاهشی دسته‌ای.
  - برای محاسبه گرادیان تابع هزینه در هر تکرار، تنها از بخش کوچکی از داده‌های آموزشی استفاده کن.

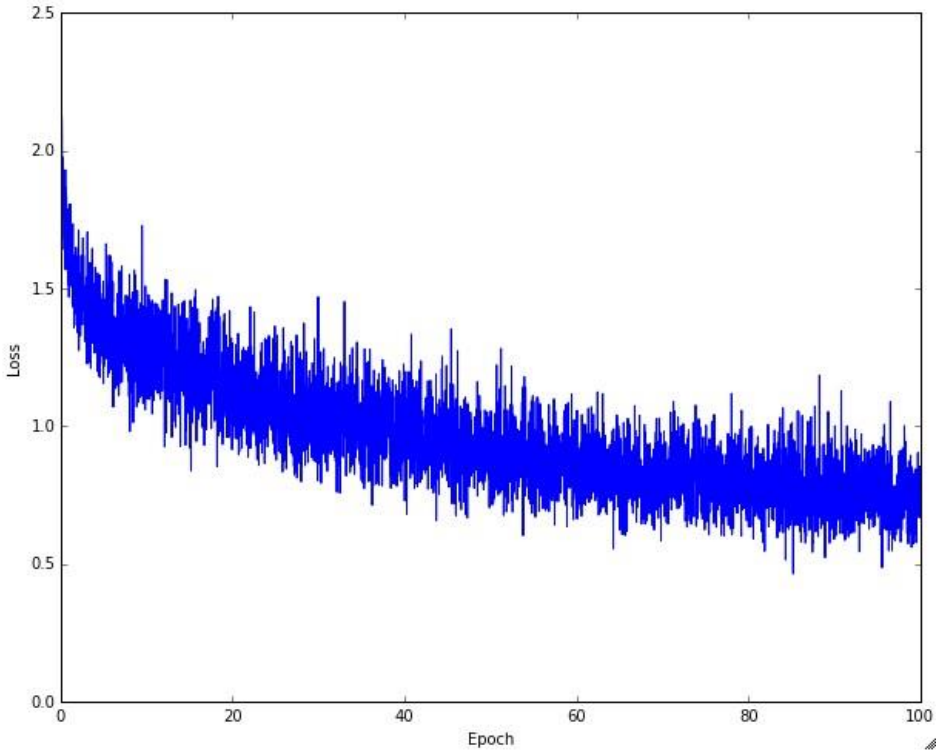
```
# Mini-batch Gradient Descent

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

  - مقادیر متداول برای اندازه دسته: ۳۲، ۶۴، ۱۲۸ و ۲۵۶.

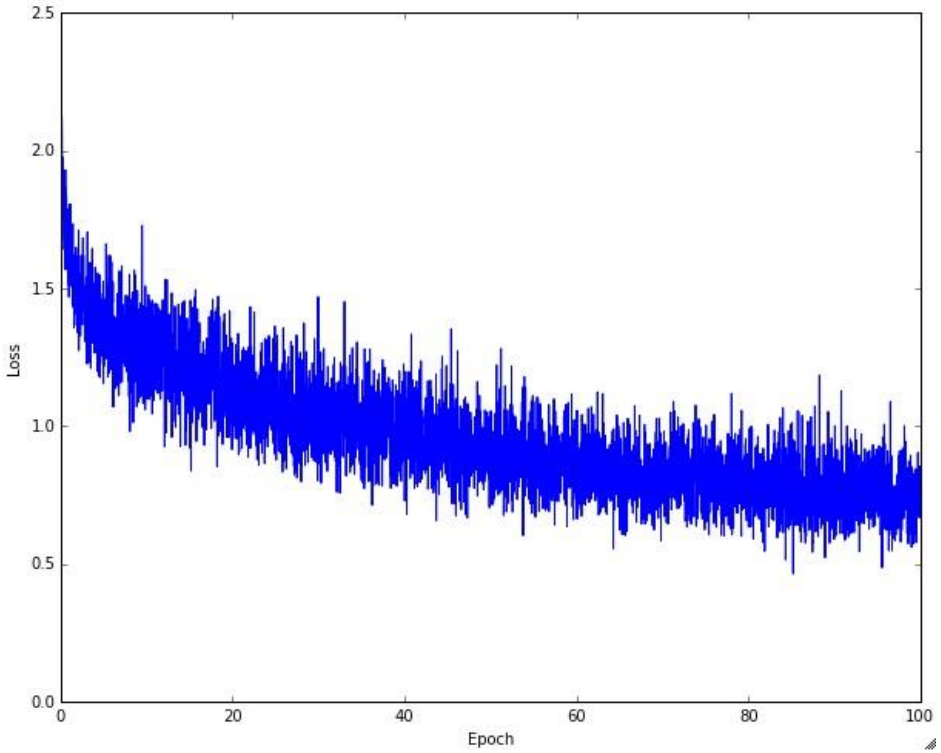
گرادیان کاهشی دسته‌ای

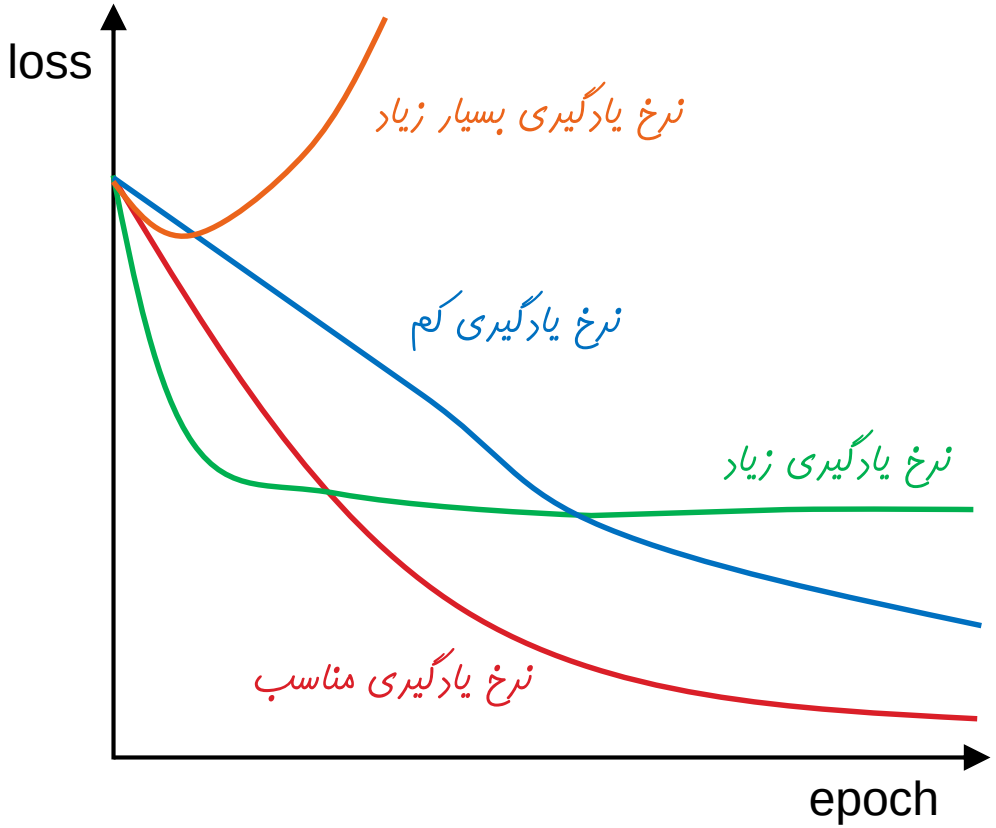
- نمونه اجرای الگوریتم گرادیان کاهشی دسته‌ای به منظور بهینه‌سازی وزن‌های یک شبکه عصبی.



هزینه در طول زمان کاهش می‌یابد.

تأثیر نرخ یادگیری (اندازه گام)





یک نسخه کاراتر از الگوریتم گرادیان کاهشی

- گرادیان کاهشی دسته‌ای.
  - برای محاسبه گرادیان تابع هزینه در هر تکرار، تنها از بخش کوچکی از داده‌های آموزشی استفاده کن.

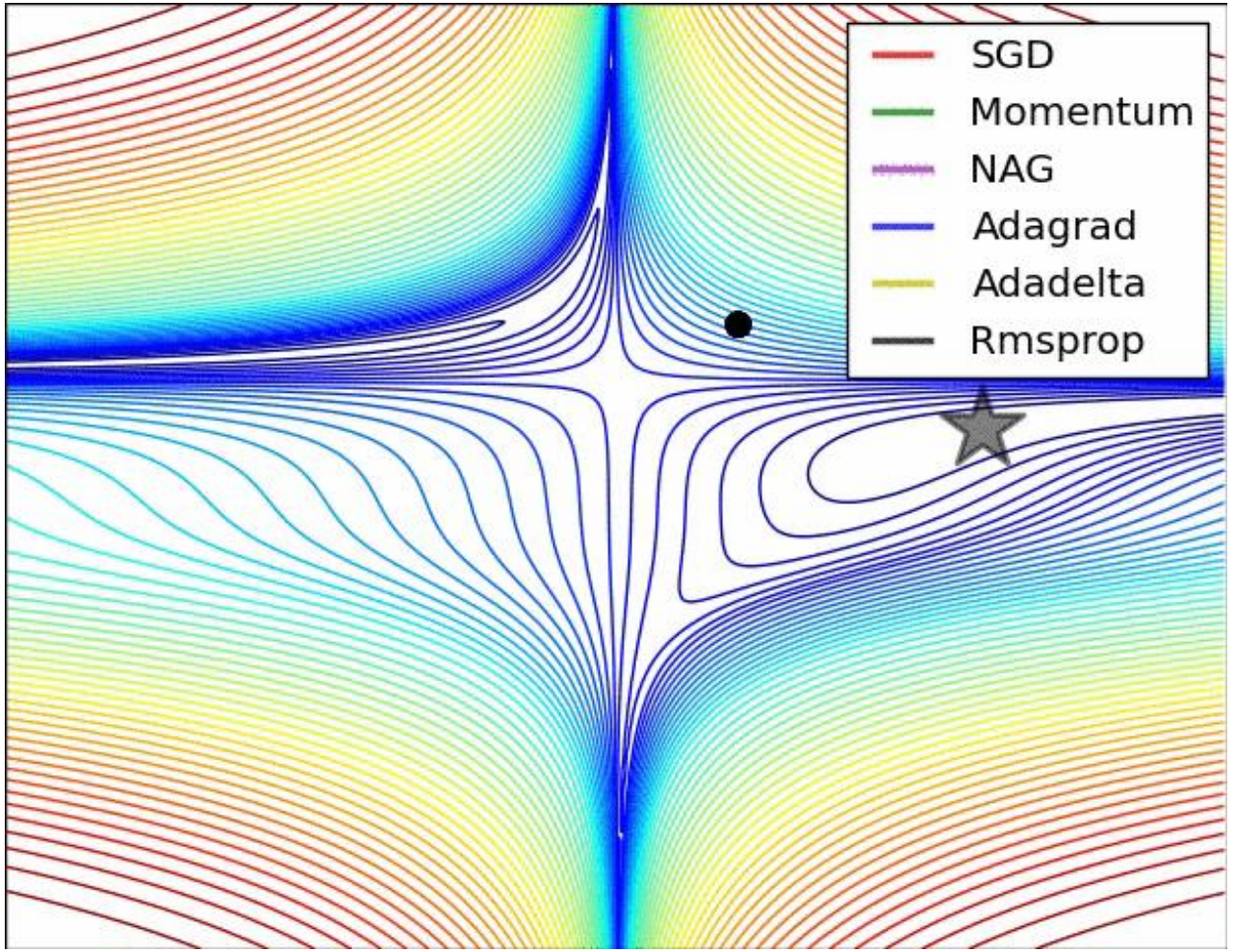
```
# Mini-batch Gradient Descent

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

- مقادیر متداول برای اندازه دسته: ۳۲، ۶۴، ۱۲۸ و ۲۵۶.

روش‌های دیگر به روز رسانی مقدار وزن‌ها.
ممنتوم، آداگراد، آدام و غیره. [با ما باشید]

به روز رسانی مقدار وزن‌ها



جلسه آینده

- الگوریتم پس انتشار خطا!
- آشنایی با شبکه‌های عصبی (بخش اول)