
بسم الله الرحمن الرحيم

مبانی بینایی کامپیوتر

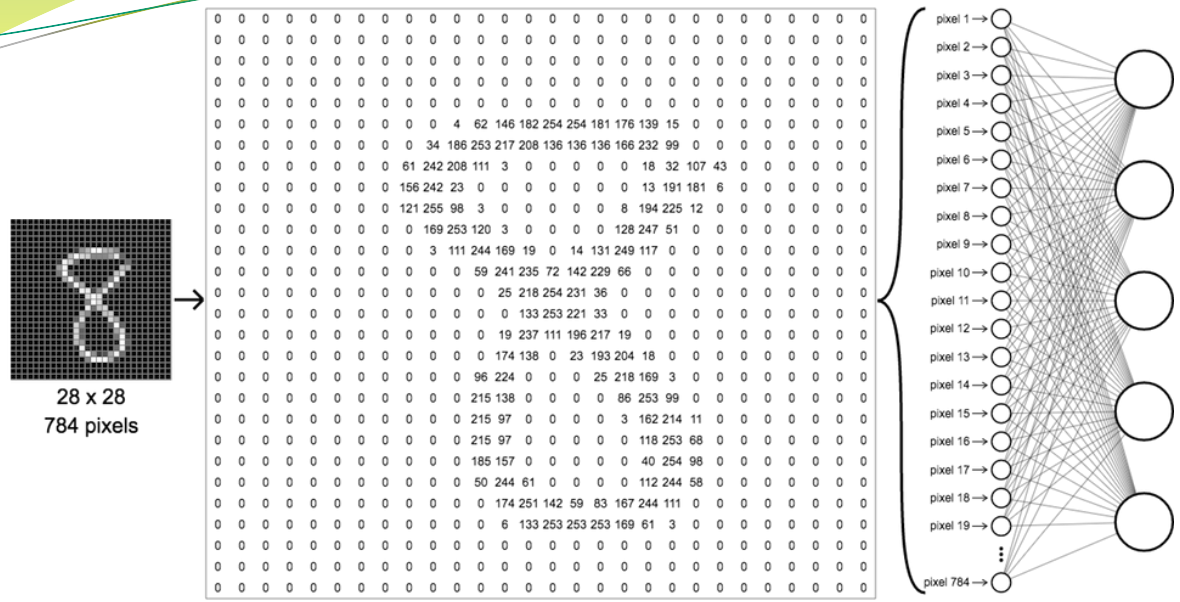
مدرس: محمدرضا محمدی

بهار ۱۴۰۳

یادگیری ویژگی

Feature Learning

یادگیری ماشین



- یادگیری ماشین از ۳ گام اصلی تشکیل می‌شود:
  - انتخاب مدل

$$y = f(x|\theta)$$

  - انتخاب معیار ارزیابی

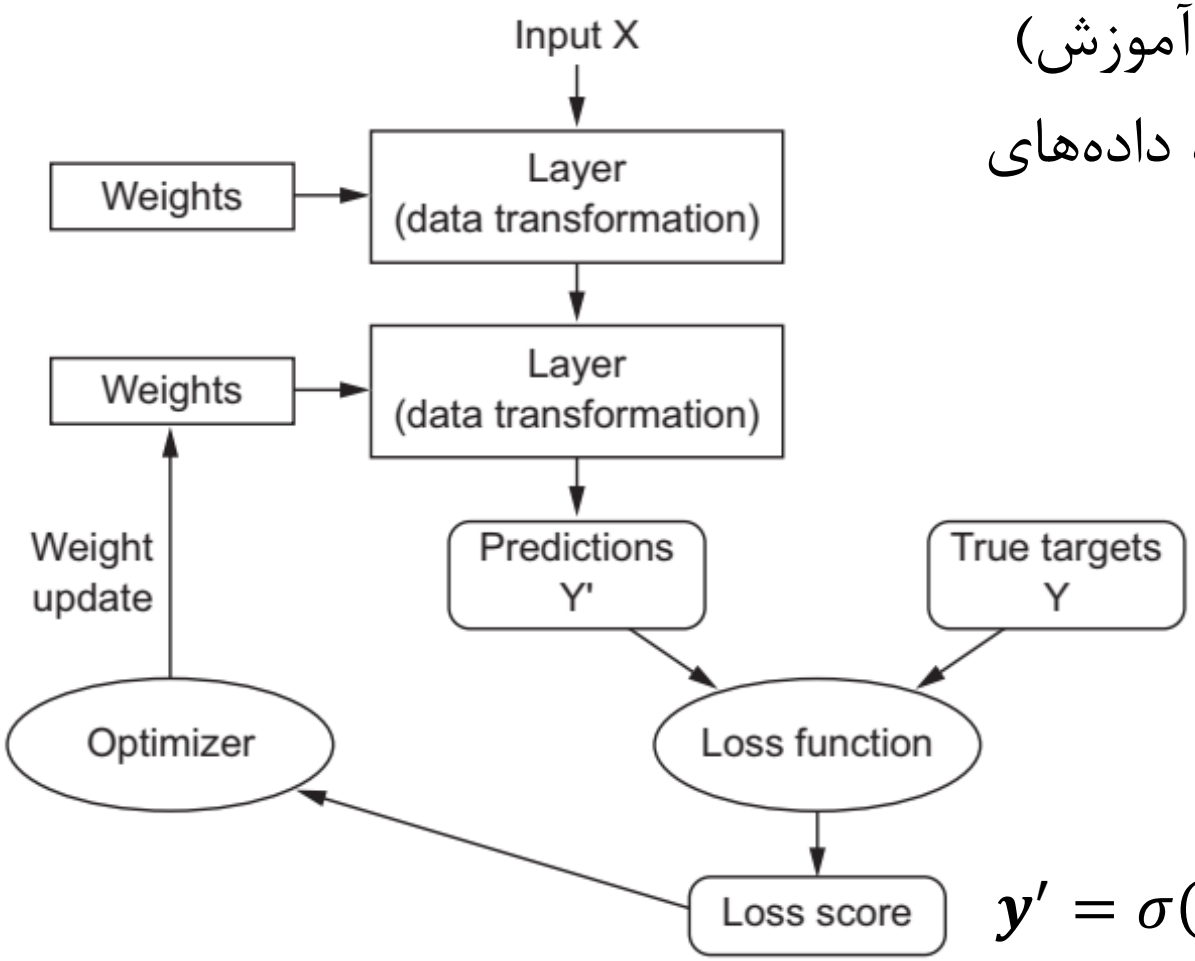
$$loss = compare\left(y_{true}, y_{pred} = f(x|\theta)\right)$$

  - بهینه‌سازی

$$\theta^* = \min_{\theta} \quad loss\left(y_{true}, f(x)\right)$$

بهینه‌سازی

- Wها و $\boldsymbol{b}$ها وزن‌های هر لایه هستند (پارامترهای قابل آموزش)
- این وزن‌ها شامل اطلاعاتی هستند که شبکه از مشاهده داده‌های آموزشی آموخته است
- چگونه مقادیر بهینه را بدست بیاوریم؟



$$\boldsymbol{y}' = \sigma(W_2\sigma(W_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2)$$

رویکرد ۱: جستجوی تصادفی

- پاسخ بسیار ضعیف است!
- دقت نهایی تنها ۱۵.۵٪ است

```
# assume X_train is the data where each column is an example (e.g. 3073 x 50,000)
# assume Y_train are the labels (e.g. 1D array of 50,000)
# assume the function L evaluates the loss function

bestloss = float("inf") # Python assigns the highest possible float value
for num in range(1000):
  W = np.random.randn(10, 3073) * 0.0001 # generate random parameters
  loss = L(X_train, Y_train, W) # get the loss over the entire training set
  if loss < bestloss: # keep track of the best solution
    bestloss = loss
    bestW = W
  print 'in attempt %d the loss was %f, best %f' % (num, loss, bestloss)

# prints:
# 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
# in attempt 6 the loss was 8.605604, best 8.605604
# ... (trunctated: continues for 1000 lines)
```

```
# Assume X_test is [3073 x 10000], Y_test [10000 x 1]
scores = Wbest.dot(Xte_cols) # 10 x 10000, the class scores for all test examples
# find the index with max score in each column (the predicted class)
Yte_predict = np.argmax(scores, axis = 0)
# and calculate accuracy (fraction of predictions that are correct)
np.mean(Yte_predict == Yte)
# returns 0.1555
```

رویکرد ۲: جستجوی محلی تصادفی

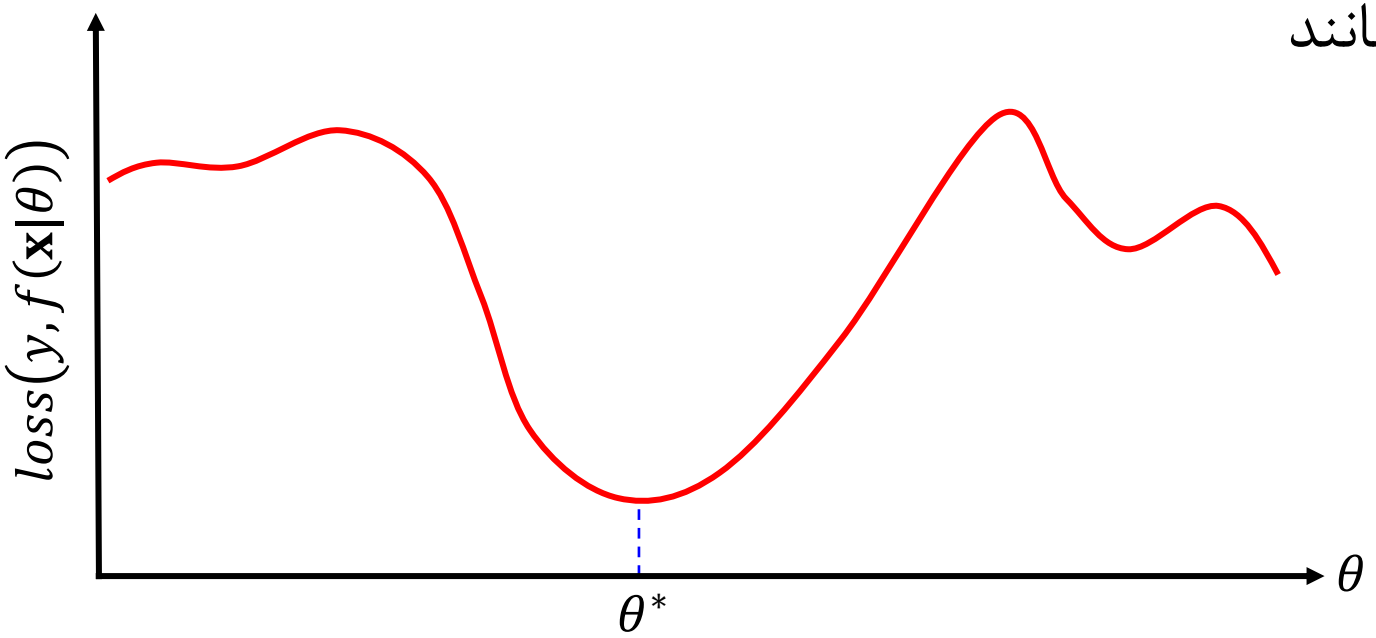
- در هر مرحله بهترین وزن‌ها را ذخیره می‌کنیم و جستجو را با یک گام محدود در اطراف آن انجام می‌دهیم
- با استفاده از این تغییر ساده، دقت بر روی داده‌های آزمون به ۲۱.۴٪ افزایش می‌یابد
- همچنان دقت خیلی پائین است!

```
W = np.random.randn(10, 3073) * 0.001 # generate random starting W
bestloss = float("inf")
for i in range(1000):
  step_size = 0.0001
  Wtry = W + np.random.randn(10, 3073) * step_size
  loss = L(Xtr_cols, Ytr, Wtry)
  if loss < bestloss:
    W = Wtry
    bestloss = loss
  print 'iter %d loss is %f' % (i, bestloss)
```

بهینه‌سازی

$$\theta^* = \min_{\theta} loss\big(y, f(\mathbf{x}|\theta)\big)$$

- در حالت‌هایی که فضای جستجو کوچک باشد می‌توان تمام فضا را جستجو کرد
- اگر فضای جستجو پیوسته اما ساده باشد، می‌توانیم مشتق بگیریم و مساوی با صفر قرار دهیم
- در غیر این صورت، باید از روش‌های تقریبی مانند الگوریتم گرادیان کاهشی استفاده کنیم



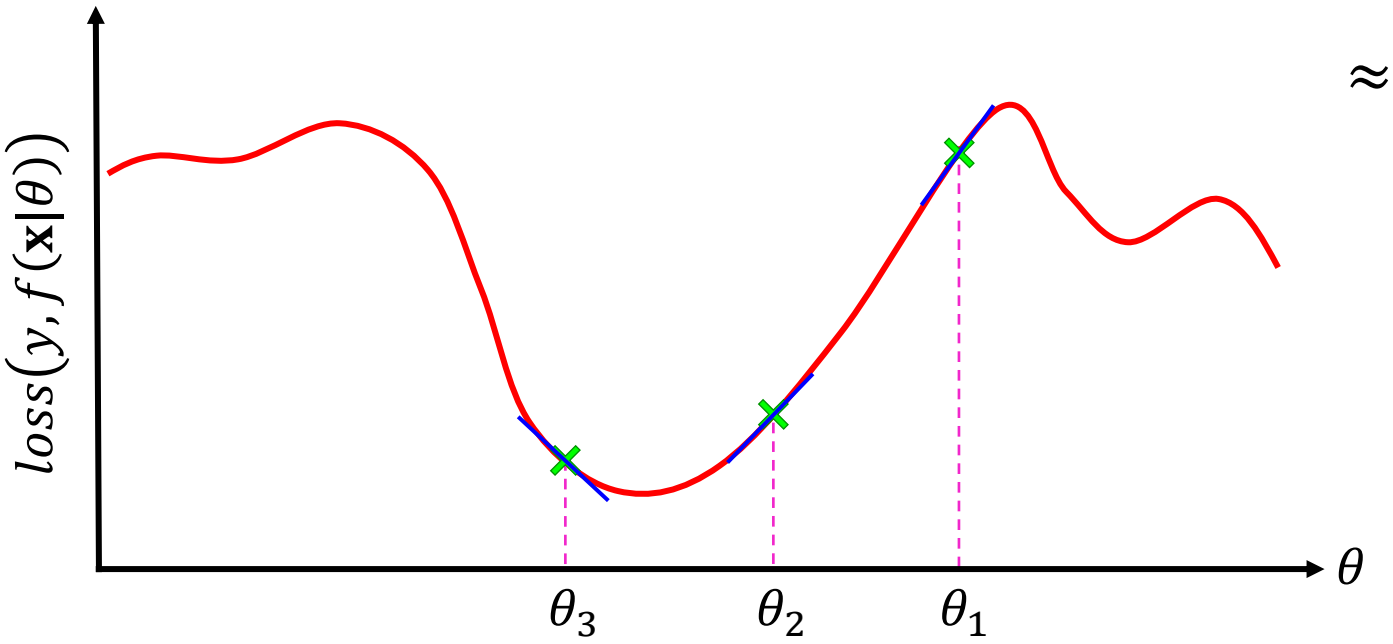
گرادیان کاهشی (Gradient Descent)

- با یک نقطه اولیه شروع می‌کنیم و در هر گام در جهتی حرکت می‌کنیم که منجر به کاهش تابع شود

$$L(\theta + \Delta\theta) = L(\theta) + \Delta\theta \frac{\partial L(\theta)}{\partial \theta} + \frac{(\Delta\theta)^2}{2!} \frac{\partial^2 L(\theta)}{\partial \theta^2} + \cdots$$

$$\approx L(\theta) + \Delta\theta \frac{\partial L(\theta)}{\partial \theta}$$

- در خلاف جهت گرادیان حرکت می‌کنیم

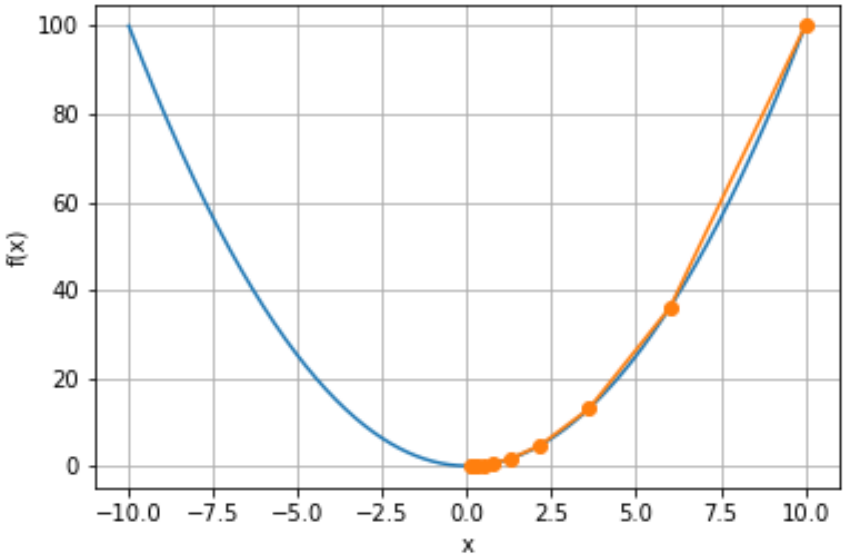


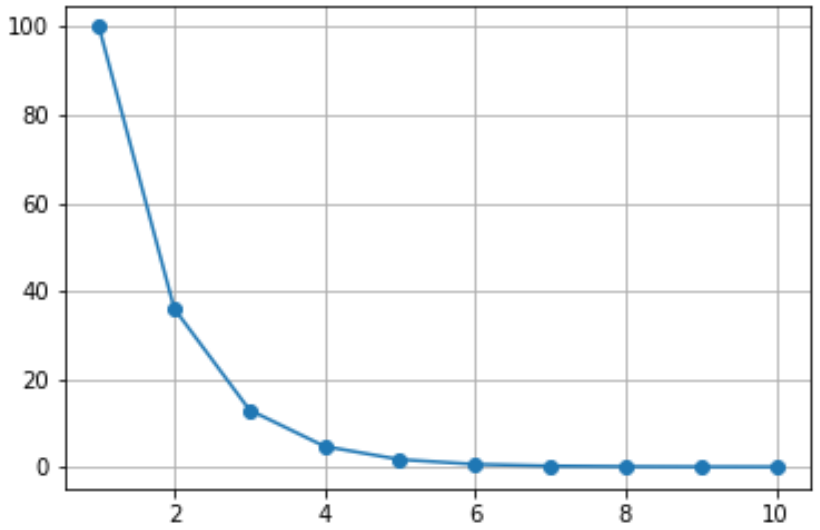
گرادیان کاهشی (Gradient Descent)

```
# Vanilla Gradient Descent

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

$$\eta = 0.20$$



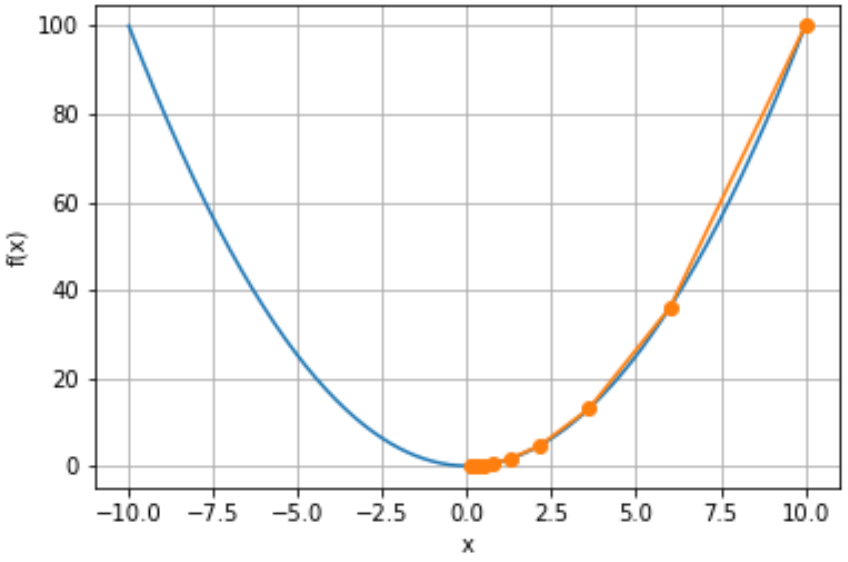


گرادیان کاهشی (Gradient Descent)

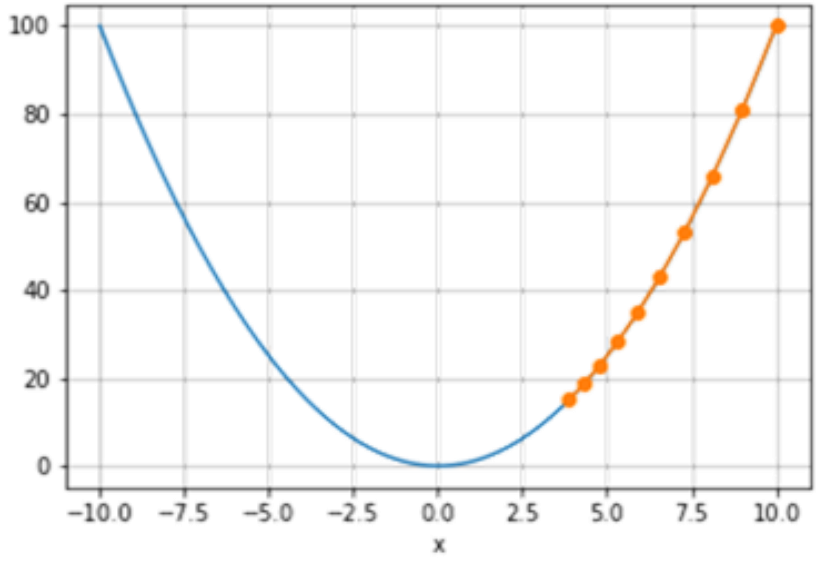
```
# Vanilla Gradient Descent

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

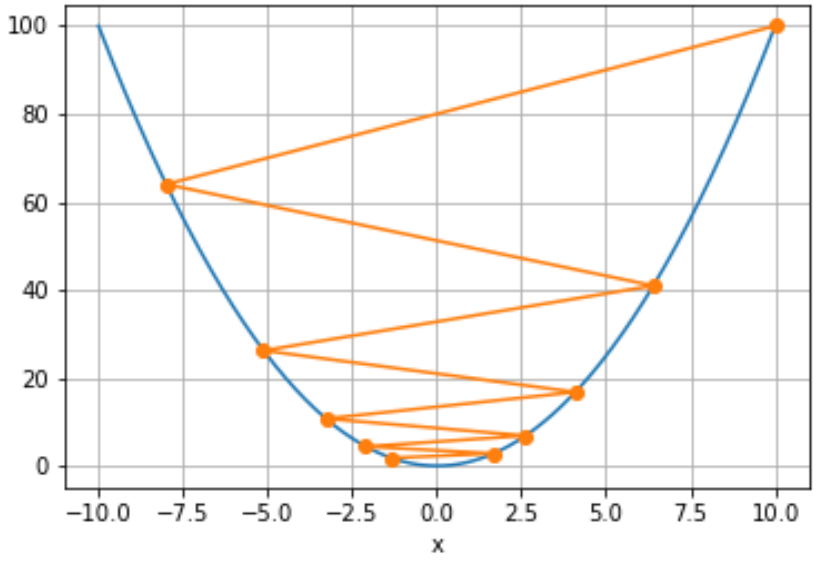
$\eta = 0.20$



$\eta = 0.05$

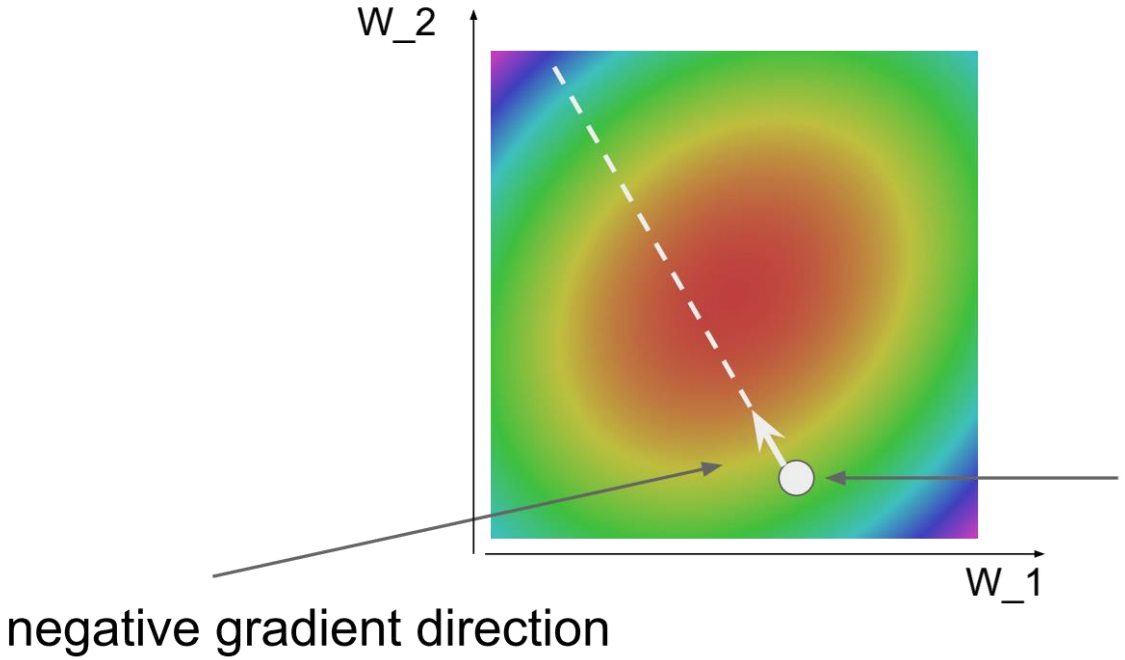


$\eta = 0.90$



گرادیان کاهشی (Gradient Descent)

- در چند بعد، گرادیان تعریف می‌شود که برداری است شامل مشتق‌های جزئی در هر بُعد
- شیب در هر جهت دلخواه برابر است با ضرب داخلی جهت با بردار گرادیان
- جهت تندترین کاهش تابع برابر با منفی گرادیان است



$$\nabla f(\mathbf{x}) = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_d}\right]^{\mathrm{T}}$$

$$f(\mathbf{x} + \boldsymbol{\epsilon}) = f(\mathbf{x}) + \boldsymbol{\epsilon}^{\mathrm{T}} \nabla f(\mathbf{x}) + \mathcal{O}(\|\boldsymbol{\epsilon}\|^2)$$

$$\mathbf{x} \leftarrow \mathbf{x} - \eta \nabla f(\mathbf{x})$$

گرادیان کاهشی تصادفی (SGD)

- در یادگیری عمیق، تابع هدف به طور معمول میانگین تابع ضرر برای تمام نمونه‌های مجموعه داده آموزشی است

$$f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x}) \Rightarrow \nabla f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\mathbf{x})$$

- این محاسبات به صورت خطی با بزرگ شدن مجموعه داده افزایش می‌یابد
- در روش SGD، در هر تکرار گرادیان تنها برای یک نمونه محاسبه می‌شود

$$\mathbf{x} \leftarrow \mathbf{x} - \eta \nabla f_i(\mathbf{x})$$

- هزینه هر گام به مقدار ثابت $\mathcal{O}(1)$ می‌رسد
- گرادیان تصادفی $\nabla f_i(\mathbf{x})$ یک تخمین بدون بایاس از $\nabla f(\mathbf{x})$ است

$$\mathbb{E}_i[\nabla f_i(\mathbf{x})] = \nabla f(\mathbf{x})$$

گرادیان کاهشی تصادفی (SGD)

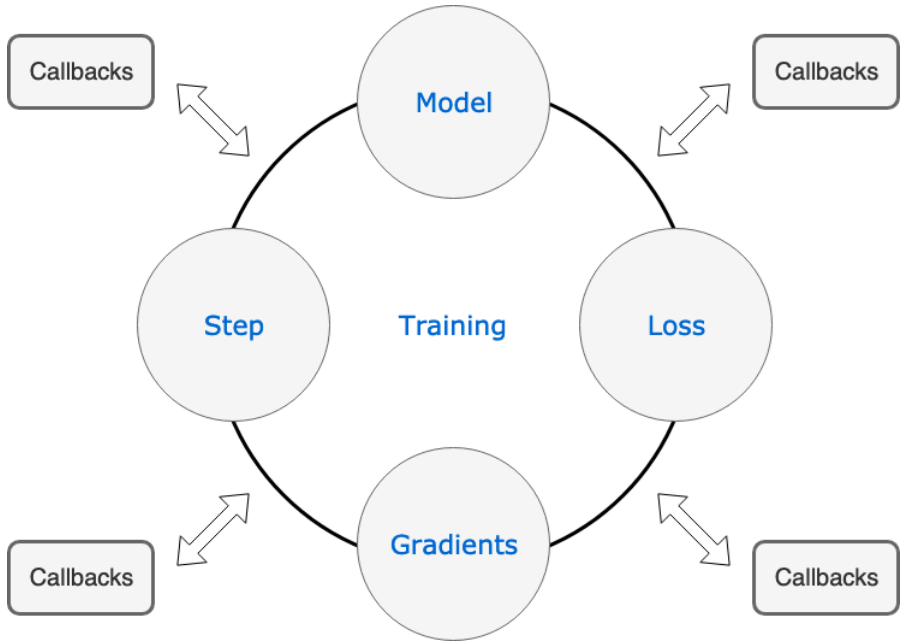
- بجای ۱ نمونه، می‌توان $\nabla f(\mathbf{x})$ آن را با استفاده از یک minibatch از نمونه‌ها تقریب زد
  - ۳۲/۶۴/۱۲۸ متداول هستند

```
# 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
```

چرخه آموزش

- این تنظیم تدریجی، که آموزش نیز نامیده می شود، پایه یادگیری در بسیاری از الگوریتم‌های یادگیری ماشین است
  - یک batch از نمونه‌های آموزشی x و خروجی‌های مربوطه y انتخاب می‌شود
  - شبکه بر روی x اعمال می‌شود (forward pass) تا y_{pred} بدست بیاید
  - تابع ضرر شبکه برای این batch از مقایسه y و y_{pred} محاسبه می‌شود
  - تمام وزن‌های شبکه به گونه‌ای به روز می‌شوند که مقدار ضرر برای این batch کمی کاهش بیابد



شبیه‌سازی

- کتابخانه TensorFlow یکی از ابزارهای قدرتمند در حوزه یادگیری عمیق است که امکانات زیادی را در اختیار کاربران قرار می‌دهد و مورد استقبال بسیاری از پژوهشگران است
- در این دوره ما از بخش Keras در TensorFlow برای شبیه‌سازی استفاده می‌کنیم
- برای اجرای کدها از سرویس رایگان Google Colab استفاده می‌کنیم

Example: https://colab.research.google.com/drive/1P6zdJ3XrO5CEKBOMTFNzAr5JOowt5V0g?usp=sharing