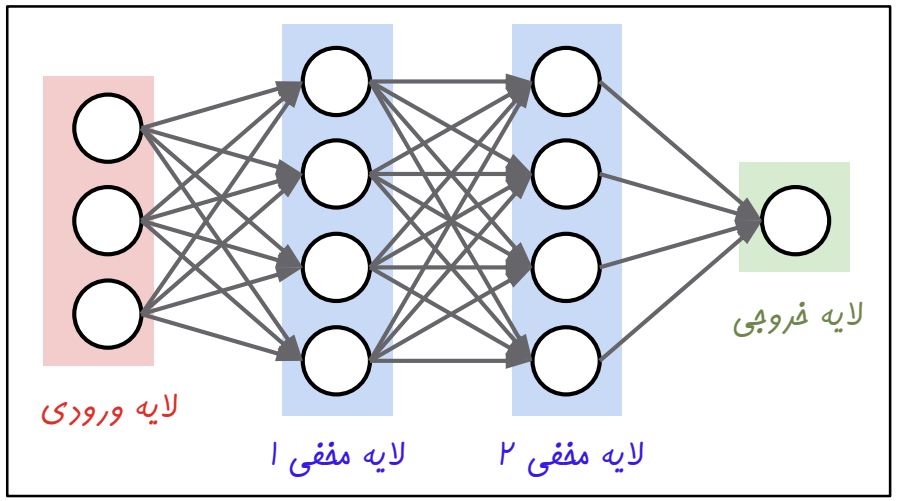
شبکه‌های عصبی: آموزش

سید ناصر رضوی www.snrazavi.ir

۱۳۹۶

یادآوری: بهینه‌سازی

- گرادیان کاهشی. [با دسته‌های کوچک]



حلقه:

۱. یک دسته از داده‌ها را به طور تصادفی **انتخاب** کن.

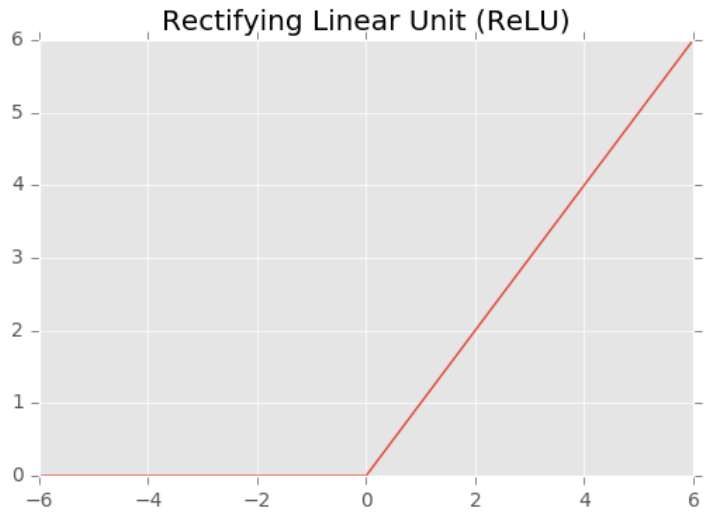
۲. **محاسبات رو به جلو** را در طول گراف انجام بده و مقدار تابع هزینه را محاسبه کن.

۳. **محاسبات رو به عقب** را به منظور محاسبه گرادیان‌ها انجام بده.

۴. مقدار پارامترها را با استفاده از گرادیان‌های محاسبه شده **به روز رسانی** کن.

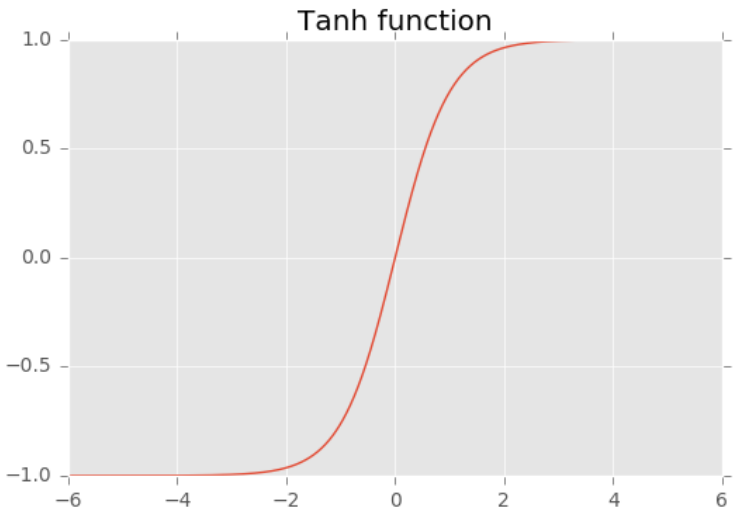
یادآوری: توابع فعالیت

ReLU



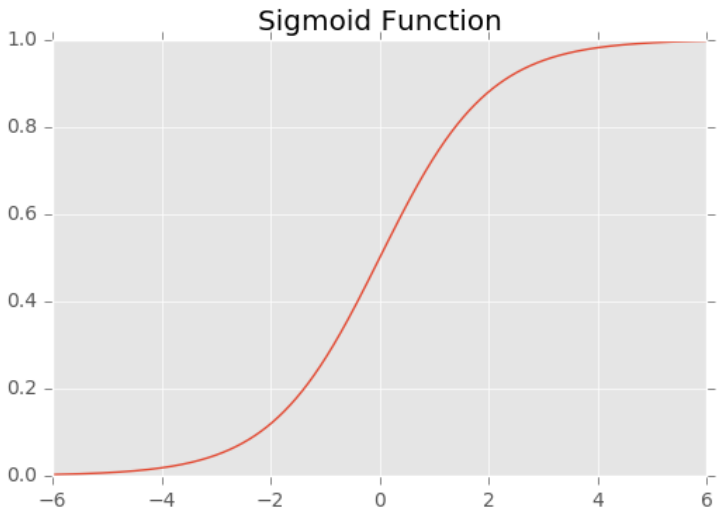
$$\max(0, x)$$

تانژانت هایپربولیک



$$\tanh(x)$$

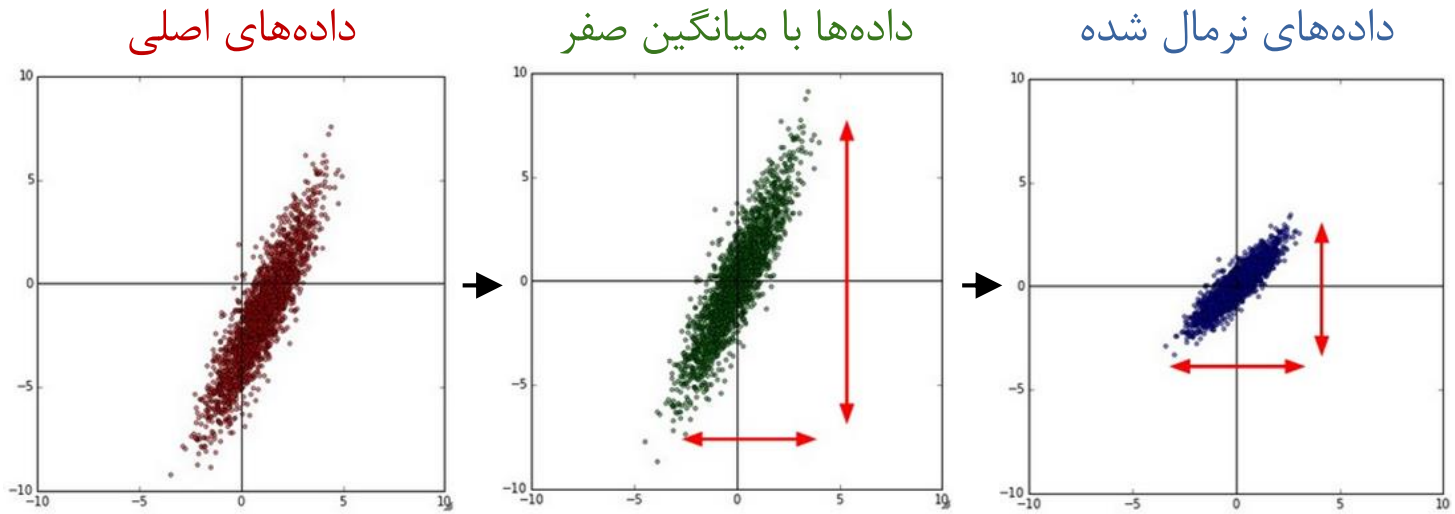
سیگموید

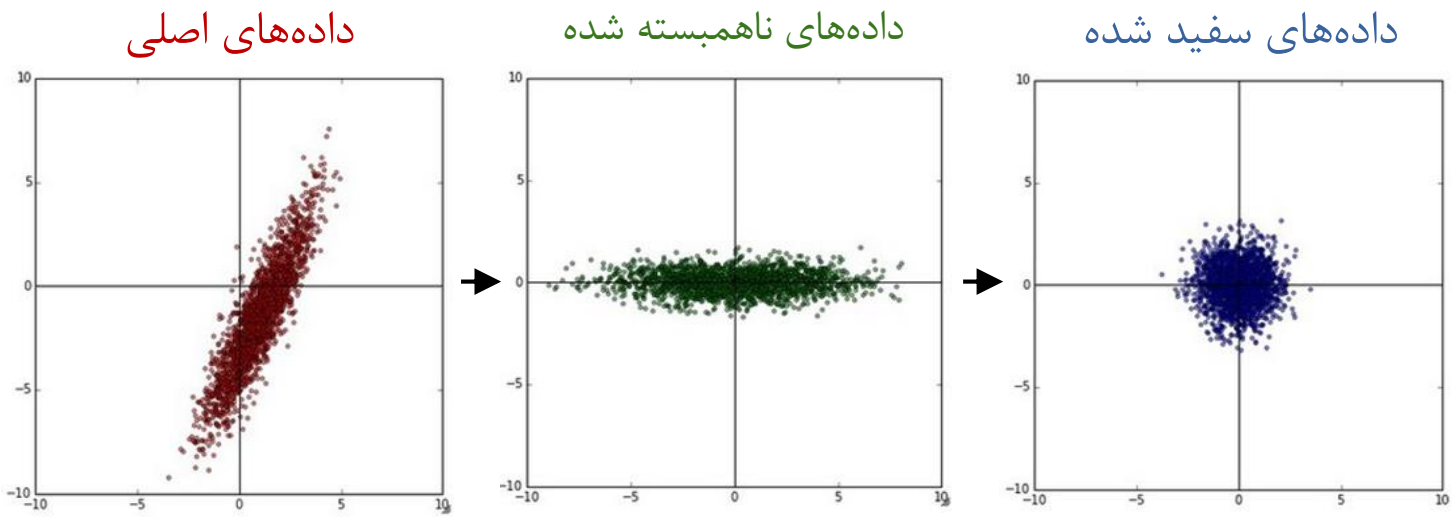


$$\sigma(x) = 1/(1 + e^{-x})$$

یادآوری: پیش‌پردازش داده‌ها

- پیش‌پردازش داده‌ها.

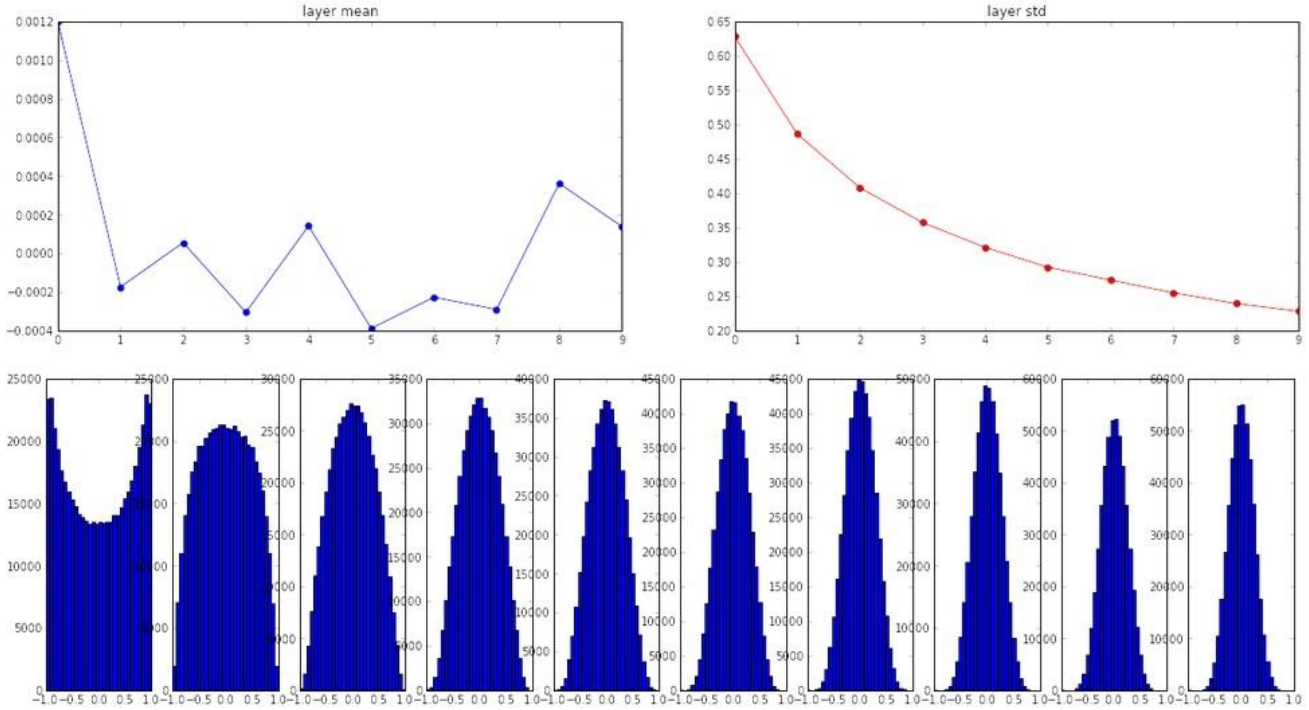




یادآوری: مقداردهی اولیه به وزن‌ها

- وزن‌دهی زاویر. [Gloret et al., 2010]

```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean 0.001198 and std 0.627953
hidden layer 2 had mean -0.000175 and std 0.486051
hidden layer 3 had mean 0.000055 and std 0.407723
hidden layer 4 had mean -0.000306 and std 0.357108
hidden layer 5 had mean 0.000142 and std 0.320917
hidden layer 6 had mean -0.000389 and std 0.292116
hidden layer 7 had mean -0.000228 and std 0.273387
hidden layer 8 had mean -0.000291 and std 0.254935
hidden layer 9 had mean 0.000361 and std 0.239266
hidden layer 10 had mean 0.000139 and std 0.228008
```




```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in)
```

یادآوری: نرمال‌سازی دسته‌ای

- نرمال‌سازی دسته‌ای.

  - جریان گرادیان را در شبکه بهبود می‌بخشد.
  - امکان استفاده از مقادیر بزرگ‌تر را برای نرخ یادگیری فراهم می‌کند.
  - وابستگی به روش مقداردهی اولیه وزن‌ها را کاهش می‌دهد.
  - به عنوان نوعی تنظیم کننده عمل می‌کند و در نتیجه نیاز به استفاده از dropout را کاهش می‌دهد.

نرمال‌سازی کنید:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

سپس اجازه دهید شبکه در صورت نیاز میانگین و واریانس را تغییر دهد:

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}$$

گام چهارم: آموزش مقدماتی

- به طور معمول، با یک ضریب تنظیم بسیار کوچک شروع می‌کنیم.
- سعی می‌کنیم یک مقدار برای نرخ یادگیری بیابیم، به گونه‌ای که باعث شود مقدار تابع هزینه در هر تکرار کاهش یابد.

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e-6, verbose=True)
```

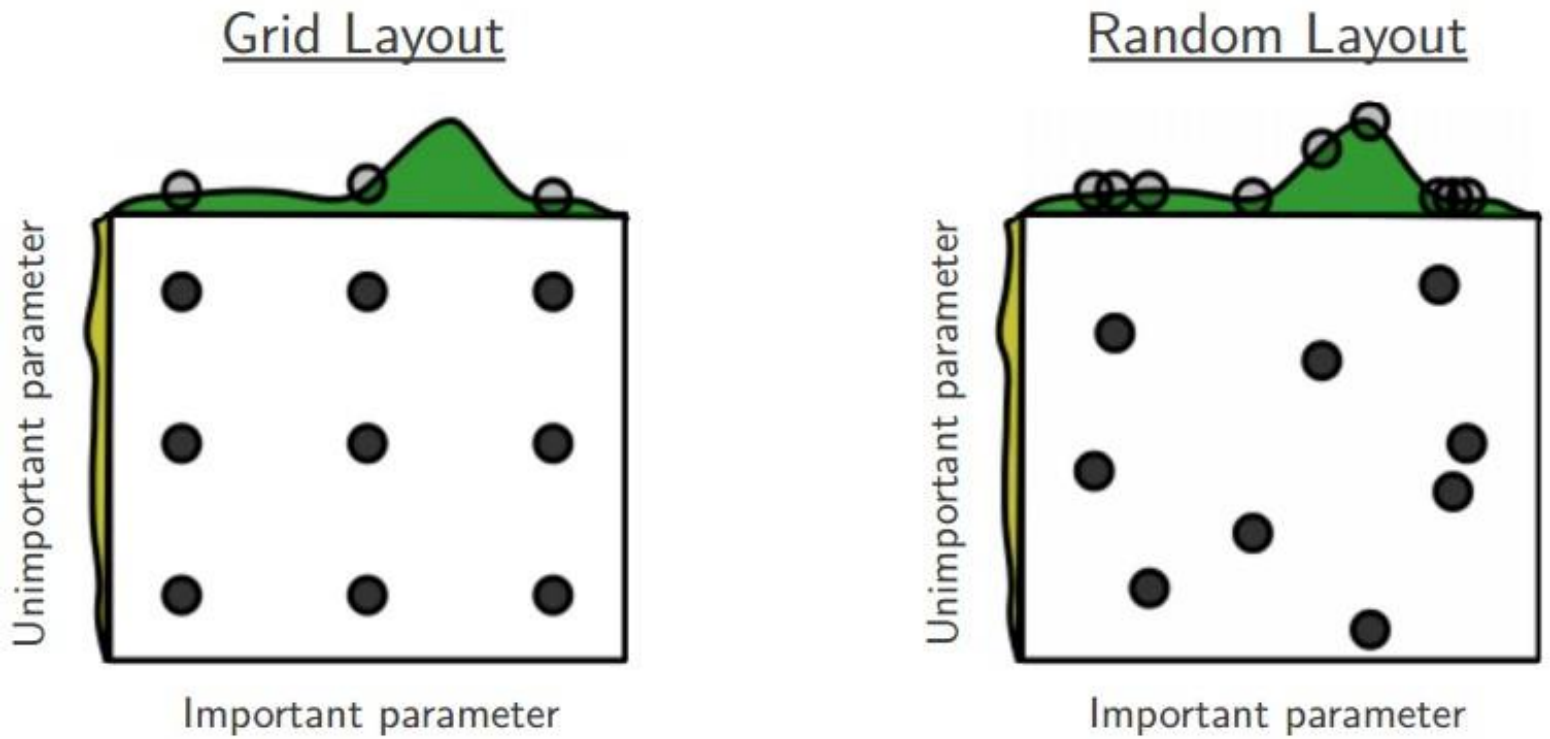
```
Finished epoch 1 / 10: cost 2.302576, train: 0.080000, val 0.103000, lr 1.000000e-06
Finished epoch 2 / 10: cost 2.302582, train: 0.121000, val 0.124000, lr 1.000000e-06
Finished epoch 3 / 10: cost 2.302558, train: 0.119000, val 0.138000, lr 1.000000e-06
Finished epoch 4 / 10: cost 2.302519, train: 0.127000, val 0.151000, lr 1.000000e-06
Finished epoch 5 / 10: cost 2.302517, train: 0.158000, val 0.171000, lr 1.000000e-06
Finished epoch 6 / 10: cost 2.302518, train: 0.179000, val 0.172000, lr 1.000000e-06
Finished epoch 7 / 10: cost 2.302466, train: 0.180000, val 0.176000, lr 1.000000e-06
Finished epoch 8 / 10: cost 2.302452, train: 0.175000, val 0.185000, lr 1.000000e-06
Finished epoch 9 / 10: cost 2.302459, train: 0.206000, val 0.192000, lr 1.000000e-06
Finished epoch 10 / 10: cost 2.302420, train: 0.190000, val 0.192000, lr 1.000000e-06
finished optimization. best validation accuracy: 0.192000
```

مقدار تابع هزینه بسیار کم تغییر می‌کند!

کاهش نیافتن تابع هزینه:

نرخ یادگیری بیش از حد کوچک است!

یادآوری: بهینه‌سازی ابرپارامترها (جستجوی تصادفی یا منظم)



[Bergstra and Bengio, 2012]

فهرست مطالب

- روش‌های به روز رسانی پارامترها.
- زمان‌بندی نرخ یادگیری.
- بررسی گرادیان.
- تنظیم. [حذف تصادفی]
- ارزیابی.

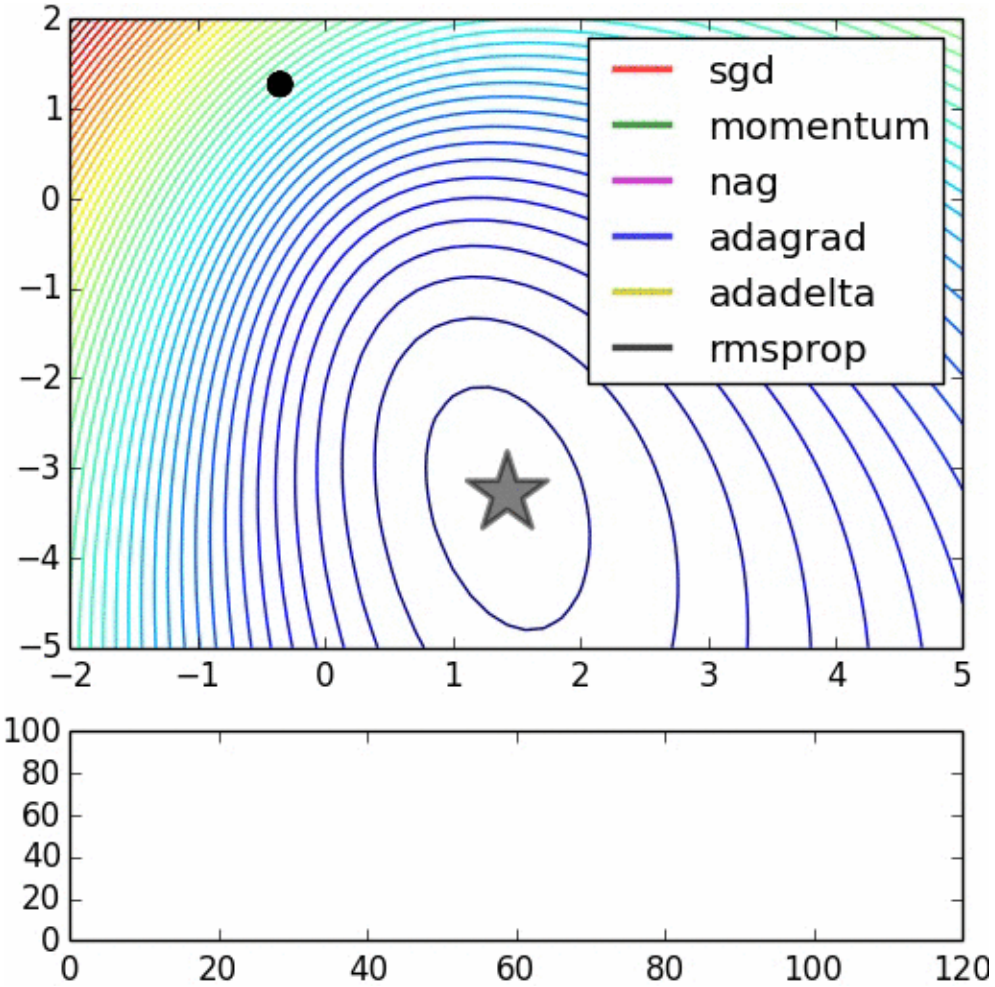
به روز رسانی پارامترها

آموزش یک شبکه عصبی: حلقه اصلی


```
while True:
    data_batch = dataset.sample_data_batch()
    loss = network.forward(data_batch)
    dx = network.backward()
    x += -learning_rate * dx
```

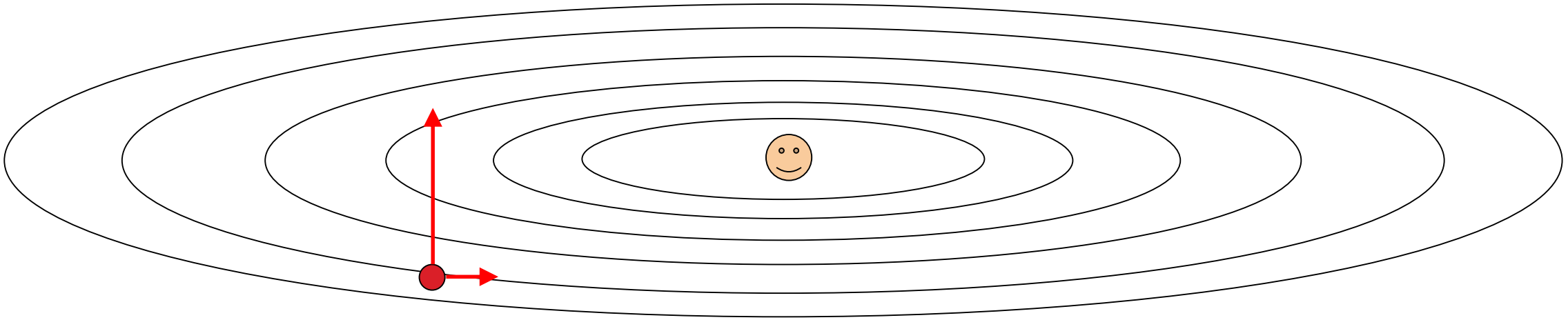
قاعده به روز رسانی: گرادیان کاهشی ساده

به روز رسانی پارامترها



به روز رسانی پارامترها

- تابع هزینه. در جهت عمودی دارای شیب تند و در جهت افقی دارای شیب ملایم.

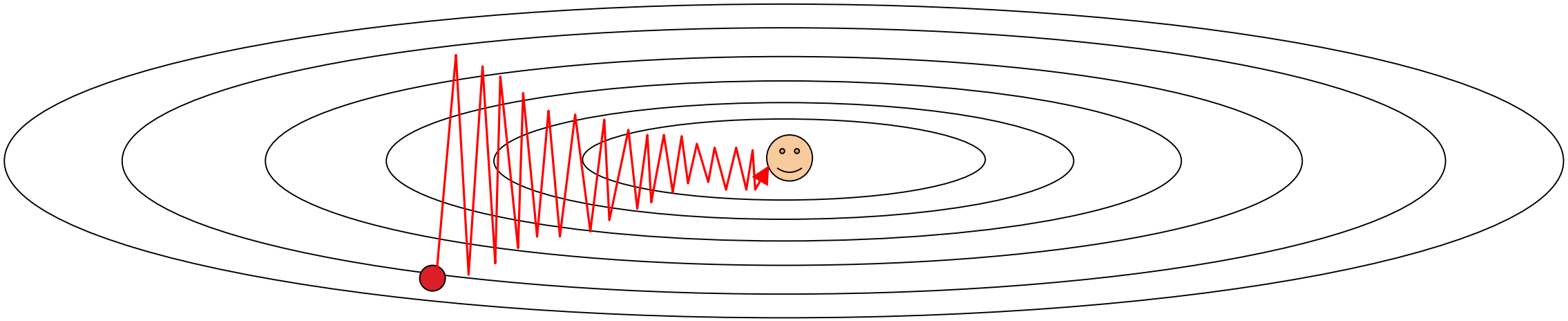


در صورت استفاده از گرادیان کاهشی، مسیر همگرایی به سمت نقطه کمینه چگونه خواهد بود؟

به روز رسانی پارامترها

- تابع هزینه. در جهت عمودی دارای شیب تند و در جهت افقی دارای شیب ملایم.

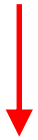
اجرای دمو



در جهت افقی پیشروی بسیار آهسته و در جهت عمودی دارای حرکات نامنظم خواهد بود!

روش ممنتوم


```
# Gradient descent update
x += -learning_rate * dx
```

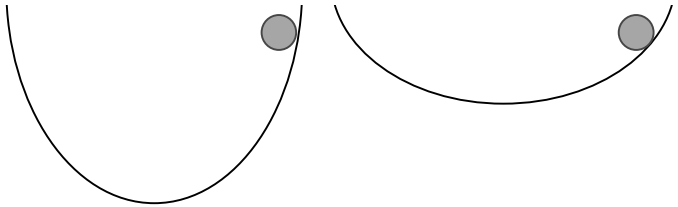


```
# Momentum update
v = mu * v - learning_rate * dx  # integrate velocity
x += v  # integrate position
```

- تفسیر فیزیکی. به عنوان یک توپ که بر روی سطح تابع هزینه به سمت پایین حرکت می‌کند + اصطکاک (ضریب mu)
  - مقدار ضریب mu: معمولاً ۰/۵، ۰/۹ یا ۰/۹۹

روش ممنتوم

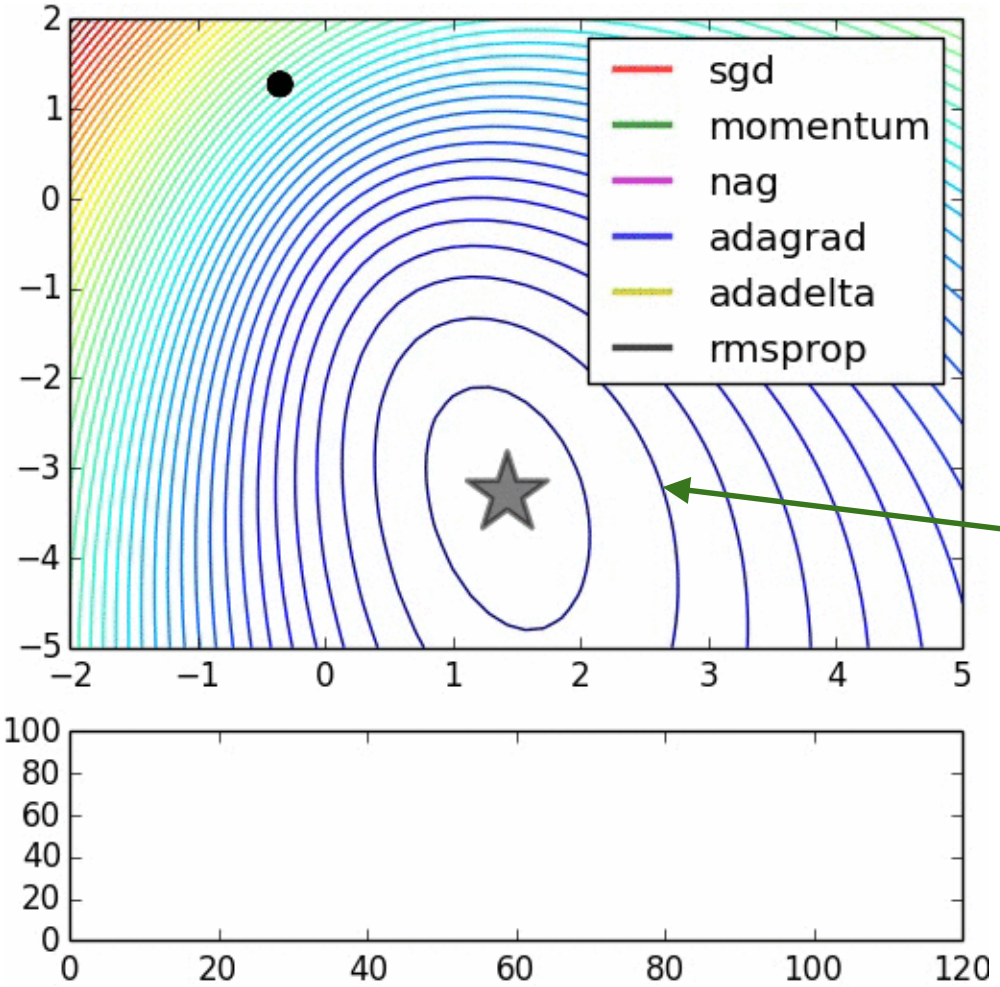

```
# Gradient descent update
x += -learning_rate * dx
```



```
# Momentum update
v = mu * v - learning_rate * dx  # integrate velocity
x += v  # integrate position
```

- در جهت‌های پرشیب به دلیل تغییر سریع علامت، به تدریج سرعت کاهش می‌یابد.
- در جهت‌های کم‌شیب، به دلیل هم جهت بودن بردار سرعت و بردار گریان، به تدریج سرعت افزایش می‌یابد.

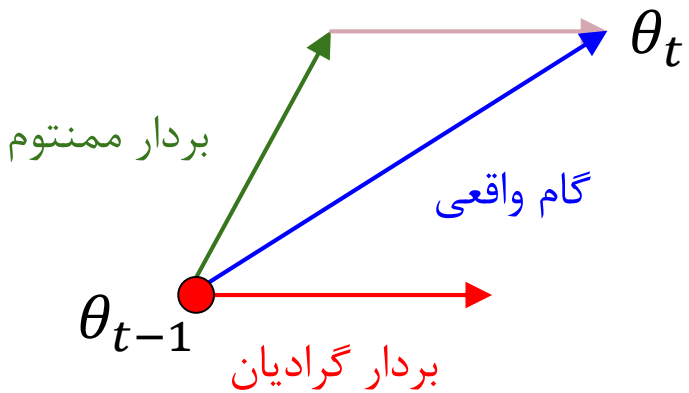
روش ممنتوم



توجه کنید که در روش ممنتوم اگرچه از هدف عبور می‌کنیم اما این روش در کل نسبت به گرادیان کاهشی سریع‌تر همگرا می‌شود.

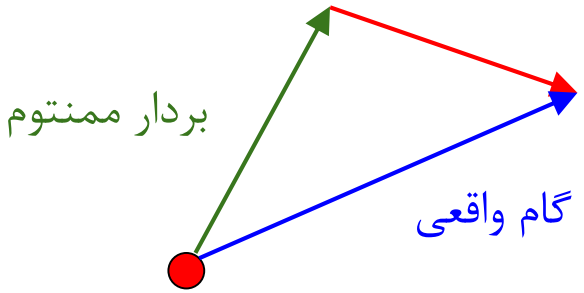
روش ممنتوم «نستروف»

روش ممنتوم معمولی




```
# Momentum update
v = mu * v - learning_rate * dx  # integrate velocity
x += v  # integrate position
```

روش ممنتوم نستروف



گرادیان «پیش‌بینی شده» (اندکی متفاوت با گرادیان اصلی)

نستروف: تنها تفاوت ...

$$v_t = \mu v_{t-1} - \epsilon \nabla f(\theta_{t-1} + \mu v_{t-1})$$

$$\theta_t = \theta_{t-1} + v_t$$

روش ممنتوم «نستروف»

$$v_t = \mu v_{t-1} - \epsilon \nabla f(\theta_{t-1} + \mu v_{t-1})$$

$$\theta_t = \theta_{t-1} + v_t$$

نسبتاً نامناسب ...

به طور معمول بردارهای زیر را داریم:

$$\theta_{t-1}, \nabla f(\theta_{t-1})$$

با یک تغییر متغیر و بازنویسی مشکل حل می‌شود:

$$\phi_{t-1} = \theta_{t-1} + \mu v_{t-1}$$

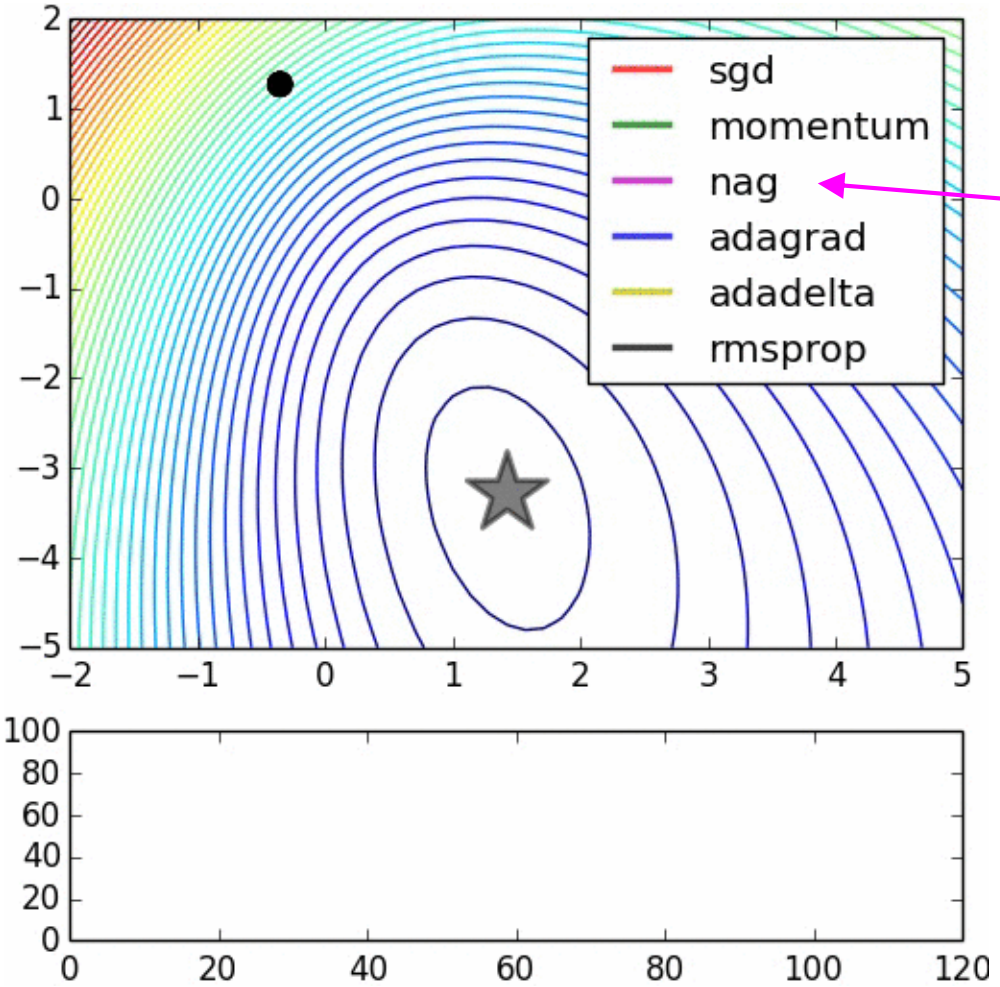
$$v_t = \mu v_{t-1} - \epsilon \nabla f(\phi_{t-1})$$

$$\phi_t = \phi_{t-1} - \mu v_{t-1} + (1 + \mu) v_t$$

$$\begin{aligned}\phi_t &= \theta_t + \mu v_t = \theta_{t-1} + v_t + \mu v_t \\ &= \phi_{t-1} - \mu v_{t-1} + (1 + \mu) v_t\end{aligned}$$

```
# Nesterov Momentum update rewrite
v_prev = v
v = mu * v - learning_rate * dx
x += -mu * v_prev + (1 + mu) * v
```

روش ممنتوم «نستروف»



Nesterov Accelerated Gradient

روش AdaGrad

- مقیاس‌بندی گرادیان‌ها در هر بعد بر اساس مجموع مربعات مقادیر قبلی گرادیان در همان بعد.

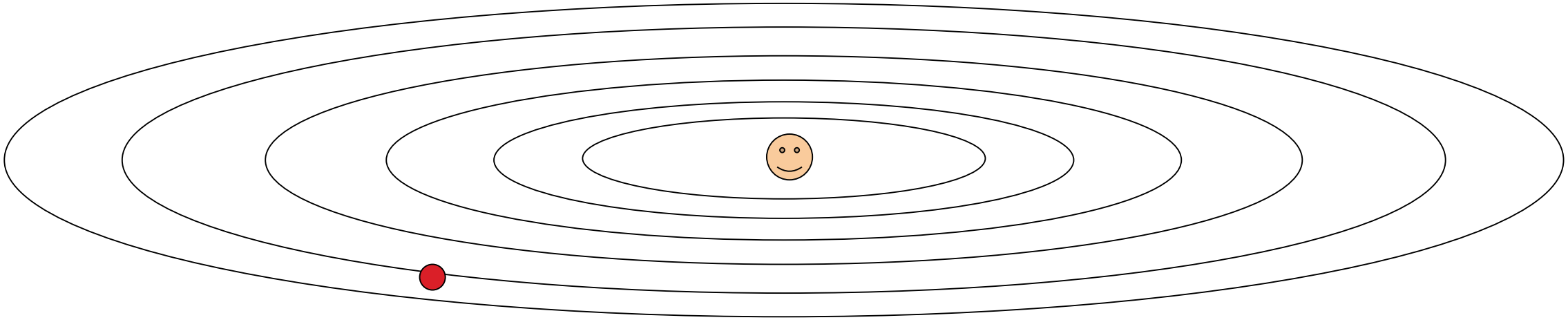
```
# AdaGrad update
cache += dx ** 2
x += -learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

[Duchi et al., 2011]

روش AdaGrad

- در صورت استفاده از این روش، همگرایی چگونه خواهد بود؟

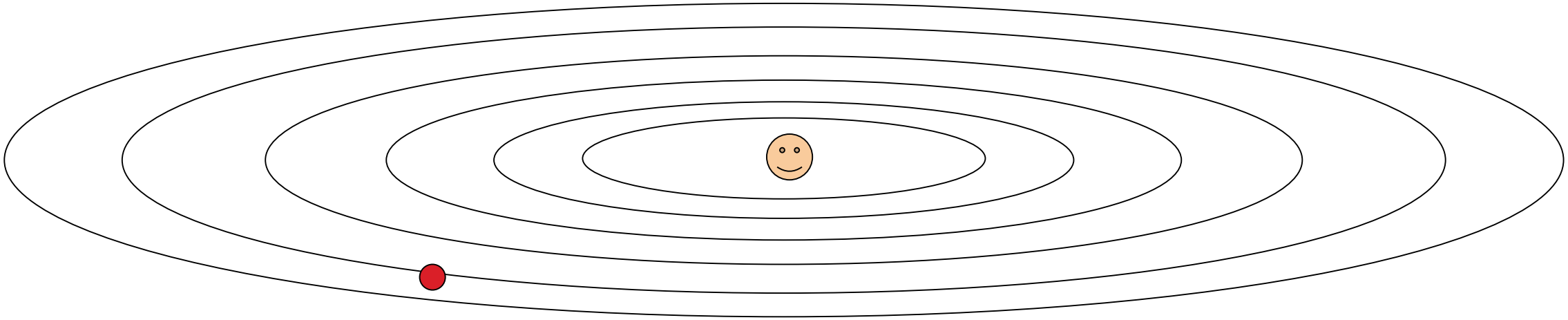
```
# AdaGrad update
cache += dx ** 2
x += -learning_rate * dx / (np.sqrt(cache) + 1e-7)
```



روش AdaGrad

- در یک مدت زمان طولانی برای نرخ یادگیری چه اتفاقی می‌افتد؟

```
# AdaGrad update
cache += dx ** 2
x += -learning_rate * dx / (np.sqrt(cache) + 1e-7)
```



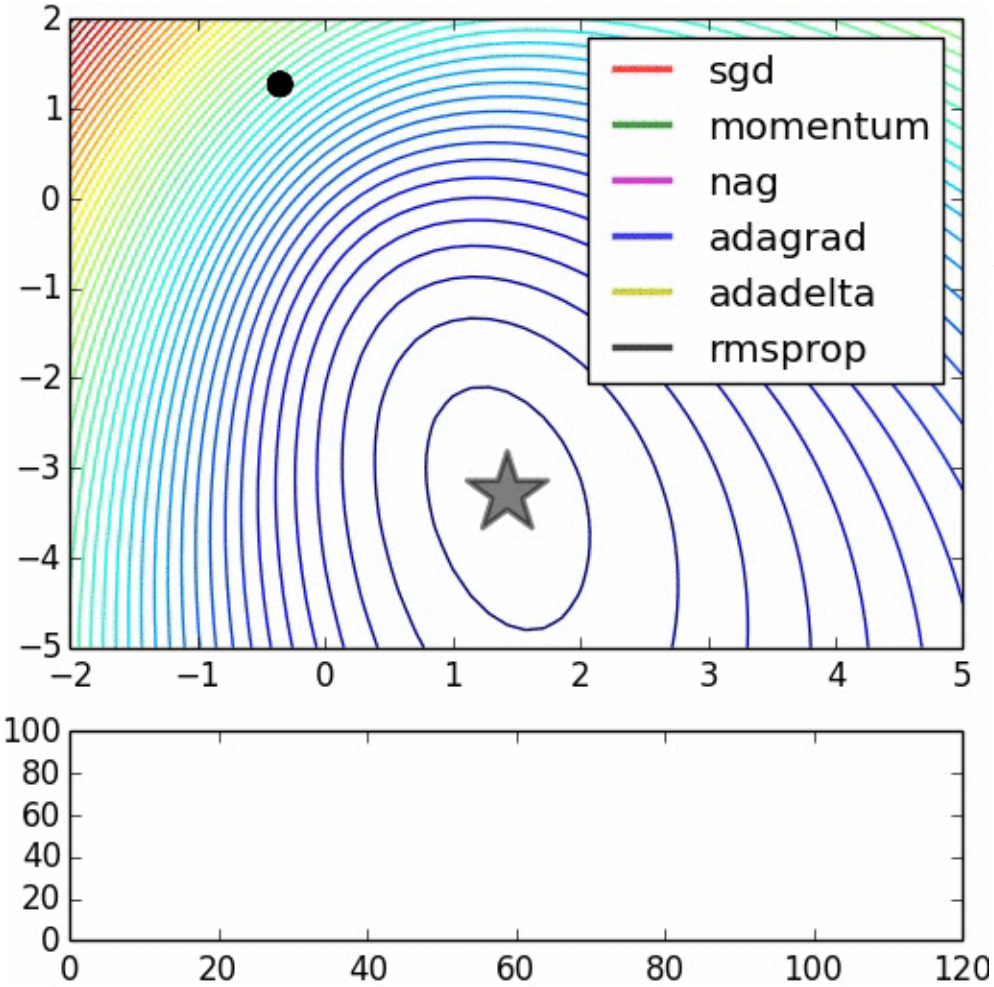
روش RMSProp


```
# AdaGrad update
cache += dx ** 2
x += -learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

```
# RMSProp update
cache = decay_rate * cache + (1 - decay_rate) * dx ** 2
x += -learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

[Tieleman and Hinton, 2012]

روش‌های AdaGrad و RMSProb



adagrad
rmsprop

روش Adam


```
# Adam update
m = beta1 * m + (1 - beta1) * dx          # first moment
v = beta2 * v + (1 - beta2) * dx ** 2     # second moment
x += -learning_rate * m / (np.sqrt(v) + 1e-7)
```

- تا حدی شبیه به روش RMSProp به همراه ممنتوم

```
# RMSProp update
cache = decay_rate * cache + (1 - decay_rate) * dx ** 2
x += -learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

[Kingma and Ba, 2014]

روش Adam

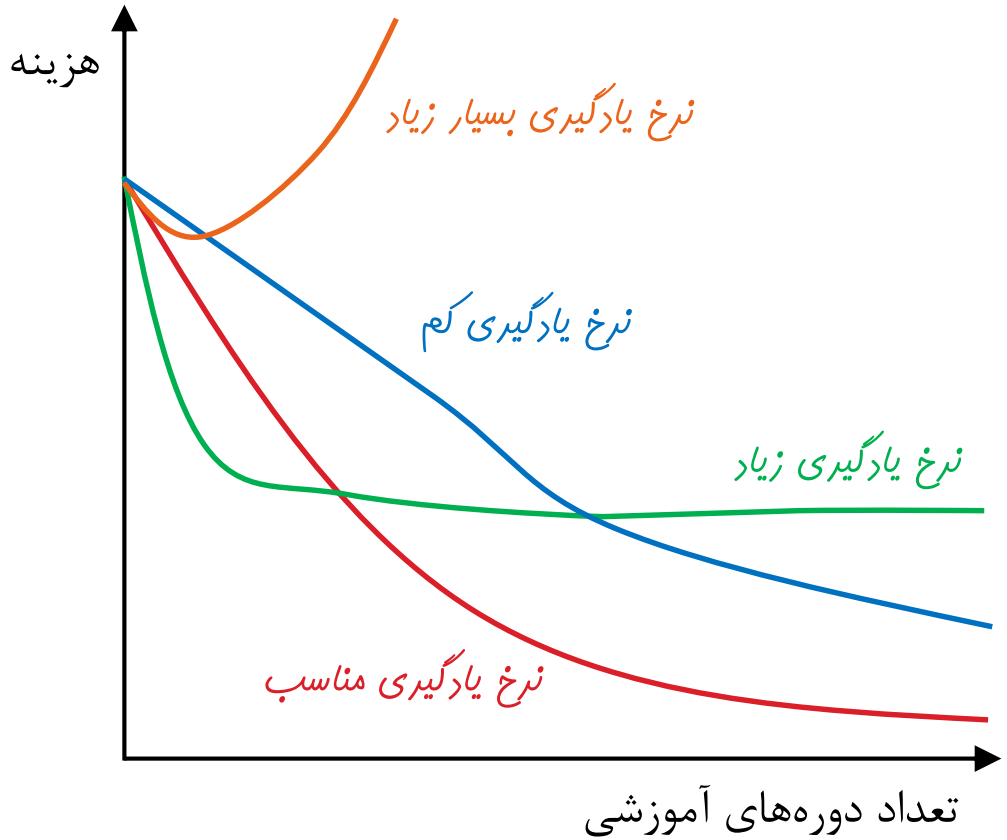

```
# Adam update
m, v =  #... initialize caches to zeros
for t in xrange(1, big_number):
    dx =  #... evaluate gradient
    m  = beta1 * m + (1 - beta1) * dx         # first moment
    v  = beta2 * v + (1 - beta2) * dx ** 2    # second moment
    mb = m / (1 - beta1 ** t)  # correct bias
    vb = v / (1 - beta2 ** t)  # correct bias
    x += -learning_rate * mb / (np.sqrt(vb) + 1e-7)
```

- دلیل تصحیح بایاس‌ها این است که m و v در ابتدا صفر هستند و مدتی زمان نیاز است تا به مقدار مناسبی برسند.

[Kingma and Ba, 2014]

نرخ یادگیری

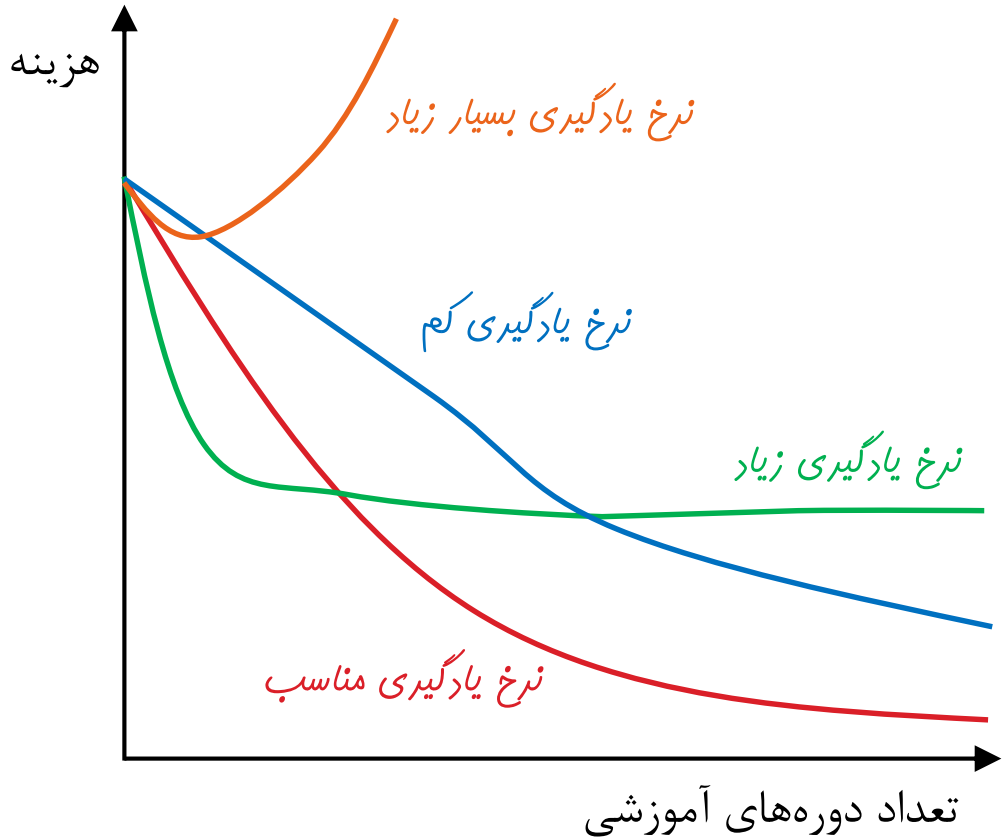
- در تمام روش‌های به روز رسانی بیان شده، مقدار **نرخ یادگیری** به عنوان یک ابرپارامتر باید به دقت تعیین شود.



بهترین مقدار برای نرخ یادگیری کدام است؟

نرخ یادگیری

- در تمام روش‌های به روز رسانی بیان شده، مقدار **نرخ یادگیری** به عنوان یک ابرپارامتر باید به دقت تعیین شود.



⇐ کاهش نرخ یادگیری در طول زمان!

کاهش مرحله‌ای:

مثلاً پس از هر چند دوره آموزش نرخ یادگیری را نصف کن.

کاهش نمایی:

$$\alpha = \alpha_0 e^{-kt}$$

کاهش متناسب با معکوس t:

$$\alpha = \alpha_0 / (1 + kt)$$

روش‌های بهینه‌سازی درجه دوم

- بسط تیلور مرتبه دوم.

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \boldsymbol{H} (\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

- پس از حل این رابطه برای یافتن نقطه بحرانی، روش به روز رسانی نیوتون را به دست می‌آوریم:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

محاسبه مشتق و برابر قرار دادن آن با صفر

این روش به روز رسانی چه ویژگی جالبی دارد؟

روش‌های بهینه‌سازی درجه دوم

- بسط تیلور مرتبه دوم.

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \boldsymbol{H} (\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

- پس از حل این رابطه برای یافتن نقطه بحرانی، روش به روز رسانی نیوتون را به دست می‌آوریم:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

توجه: هیچ ابرپارامتری وجود ندارد! (مانند نرخ یادگیری)

چرا در عمل استفاده از این روش برای آموزش شبکه‌های عصبی عمیق ممکن نیست؟

روش‌های بهینه‌سازی درجه دوم

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

- روش‌های شبه نیوتونی. [BFGS پرطرفدارترین]

به جای محاسبه معکوس ماتریس هسین، معکوس ماتریس هسین را در طول زمان و با به روز رسانی‌های مرتبه اول تقریب بزن. [پیچیدگی زمانی درجه دوم به جای پیچیدگی زمانی درجه سوم]

- روش L-BFGS. [BFGS با حافظه محدود]

تمام ماتریس هسین را ذخیره (ایجاد) نکن.

روش‌های بهینه‌سازی درجه دوم

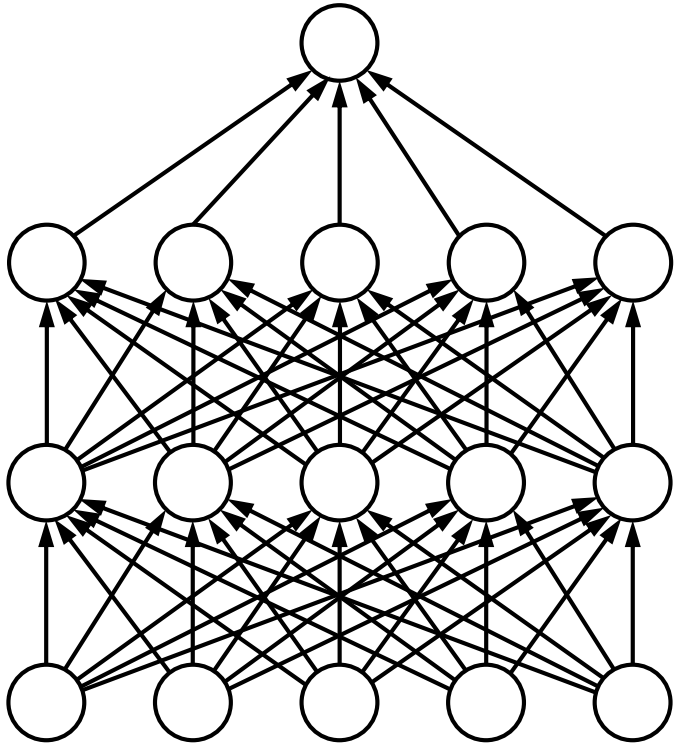
در عمل:

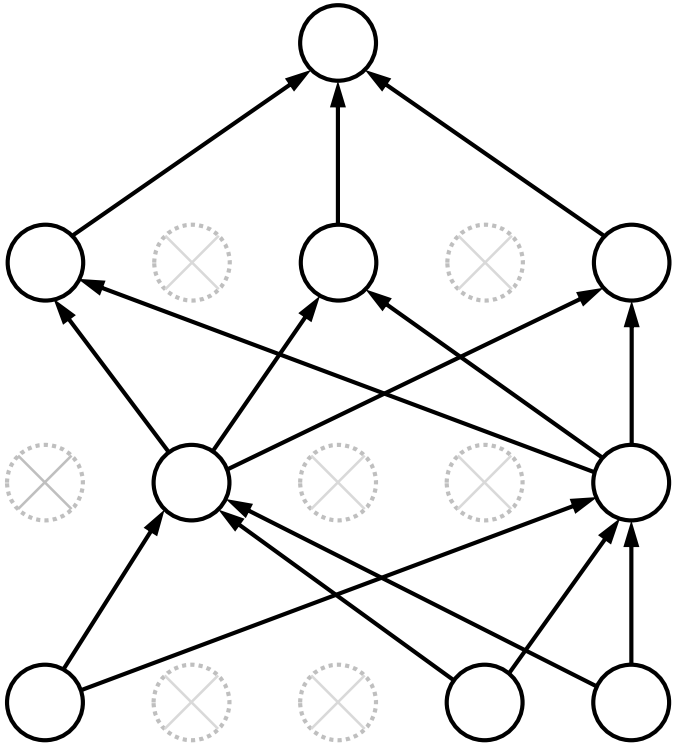
- روش Adam به عنوان انتخاب پیش‌فرض، در بیشتر موارد به خوبی کار می‌کند.
- اگر امکان به روز رسانی با استفاده از تمام داده‌ها به طور کامل در هر تکرار برای شما وجود دارد، روش L-BFGS را امتحان کنید. [در این صورت یادتان باشد که باید تمامی منابع نویز را غیر فعال کنید]

تنظیم: حذف تصادفی

تنظیم: حذف تصادفی

- در محاسبات رو به جلو، به صورت تصادفی خروجی بعضی از نورون‌ها را صفر کن.

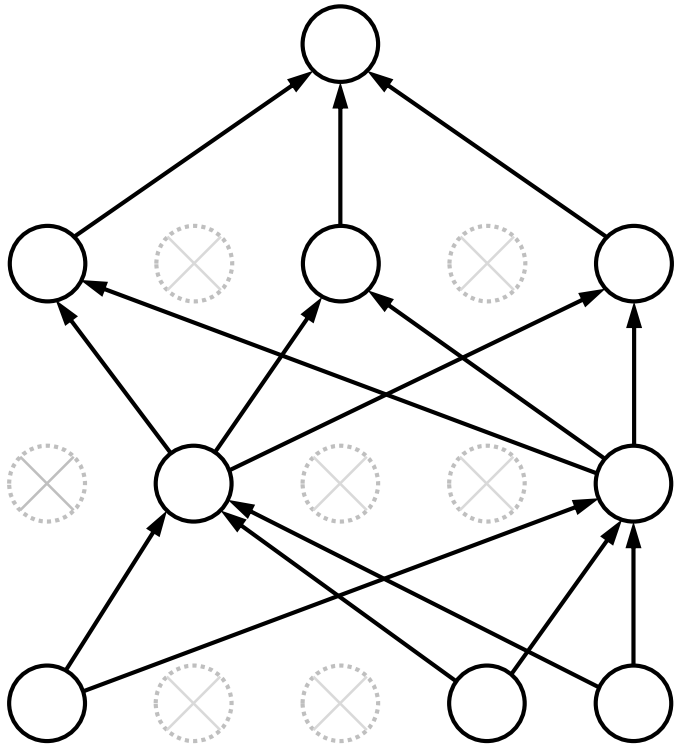




[Srivastava et al., 2014]

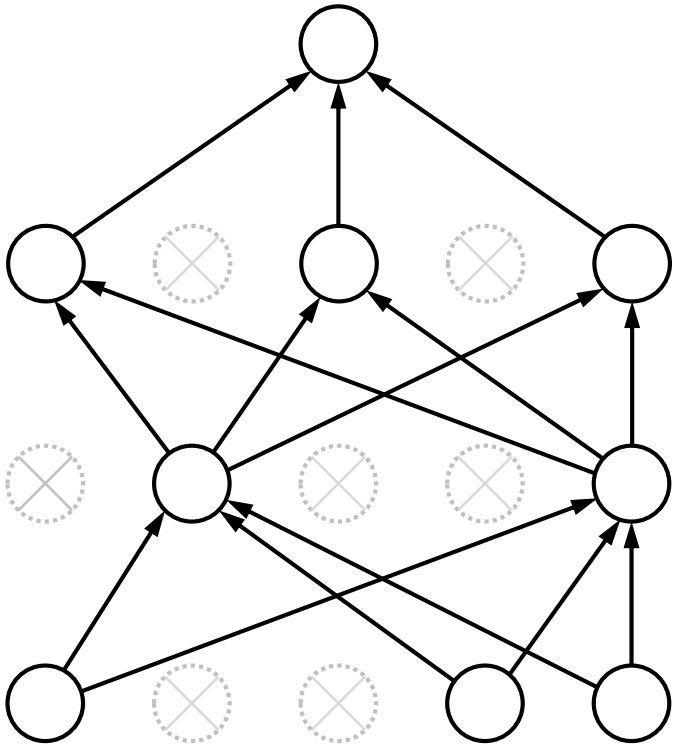
تنظیم: حذف تصادفی


```
p = 0.5  # probability of keeping a unit active

def train_step(X):
    """ X contains the data """

    # forward pass for a 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = np.random.rand(*H1.shape) < p  # first dropout mask
    H1 *= U1  # drop!

    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = np.random.rand(*H2.shape) < p # second dropout mask
    H2 *= U2  # drop!

    out = np.dot(W3, H2) + b3

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)
```

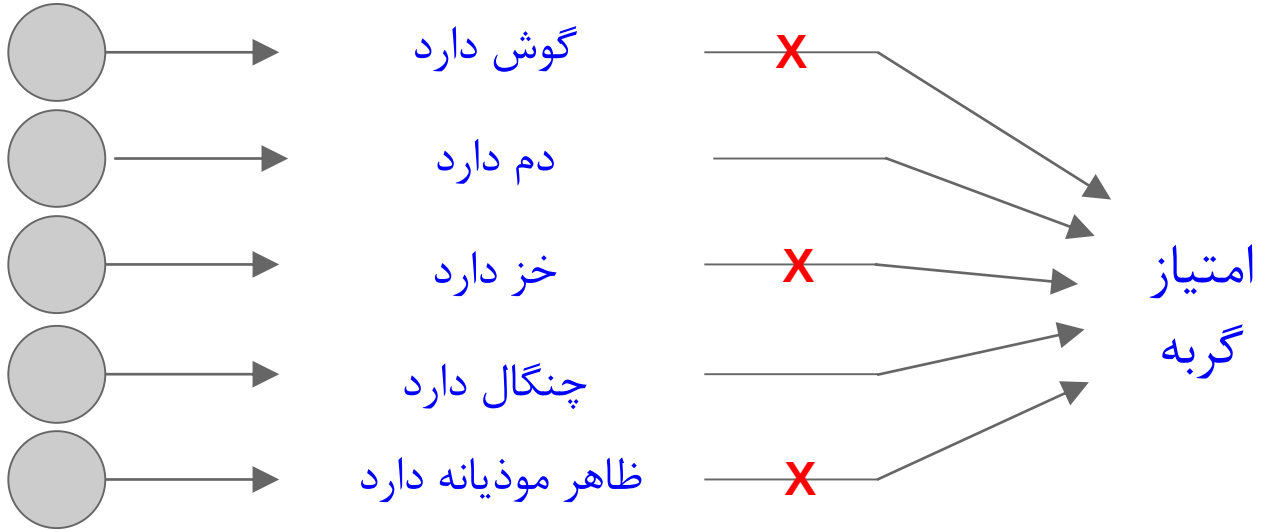


تنظیم: حذف تصادفی

- چگونه ممکن است حذف تصادفی برخی نورون‌ها ایده خوبی باشد؟

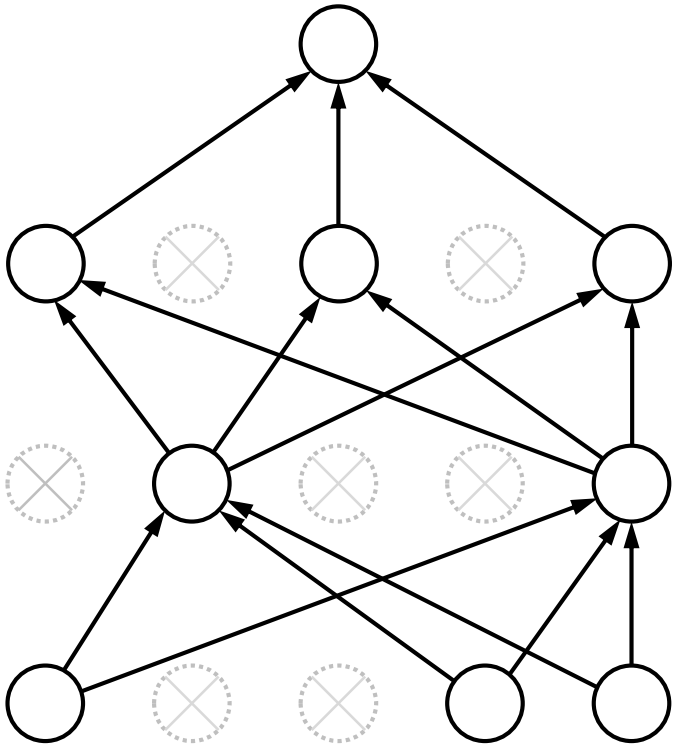


اجبار شبکه به داشتن بازنمایی دارای افزونگی



تنظیم: حذف تصادفی

- چگونه ممکن است حذف تصادفی برخی نورون‌ها ایده خوبی باشد؟

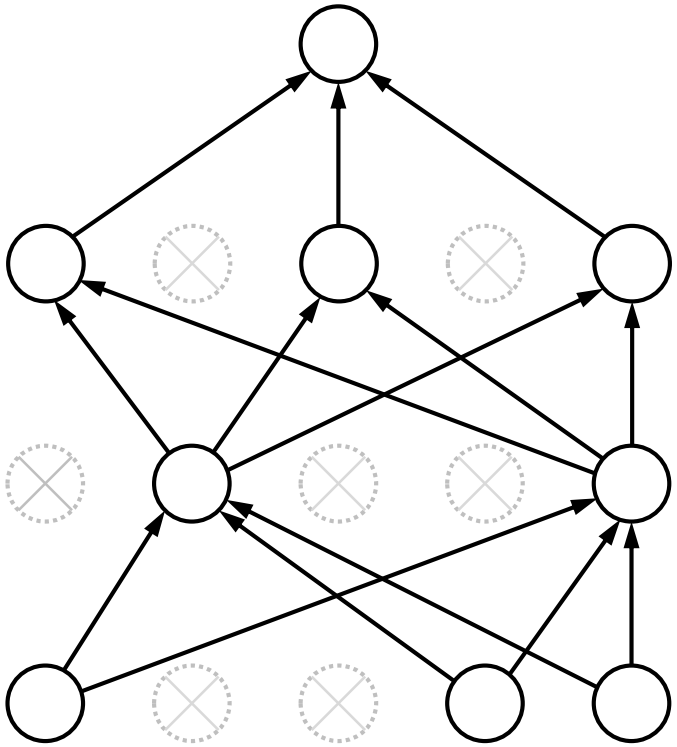


یک تفسیر دیگر:

با حذف تصادفی، یک مجموعه بزرگ از مدل‌ها آموزش داده می‌شوند (که دارای پارامترهای مشترک هستند)

حذف تصادفی در زمان پیش‌بینی...

- چگونه ممکن است حذف تصادفی برخی نورون‌ها ایده خوبی باشد؟



ایده‌آل:

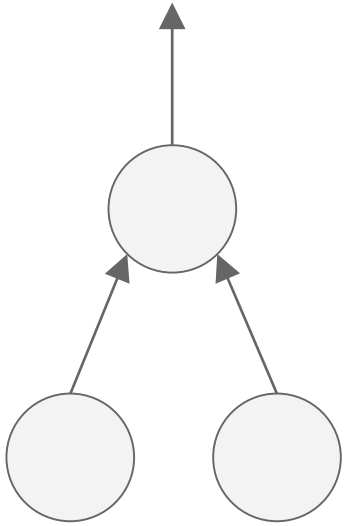
جمع کردن همه نویزها

تقریب مونت-کارلو:

انجام محاسبات رو به جلو به تعداد زیاد با ماسک‌های متفاوت و میانگین‌گیری از نتایج به دست آمده.

حذف تصادفی در زمان پیش‌بینی...

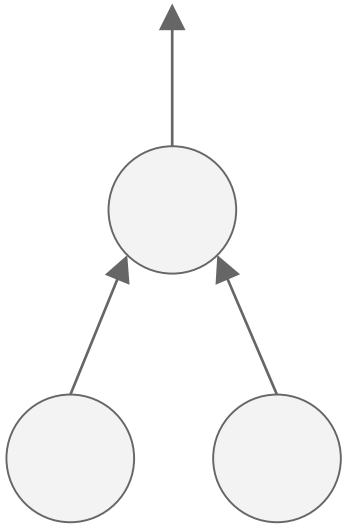
- می‌توان این کار را با یک مرحله از محاسبات رو به جلو انجام داد! (تقریب)
  - با فعال نگاه داشتن همه نورون‌ها (بدون حذف تصادفی)



(می‌توان نشان داد این کار تقریبی از ارزیابی اجتماع همه مدل‌ها است)

حذف تصادفی در زمان پیش‌بینی...

- می‌توان این کار را با یک مرحله از محاسبات رو به جلو انجام داد! (تقریب)
  - با فعال نگاه داشتن همه نورون‌ها (بدون حذف تصادفی)

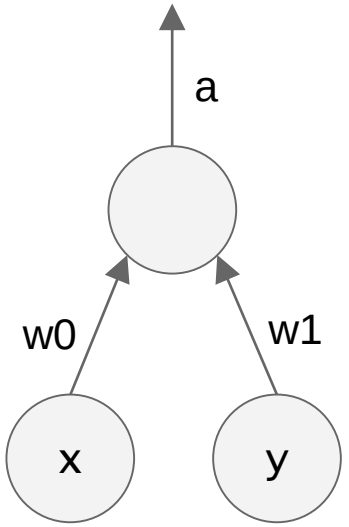


فرض کنید با فعال بودن همه ورودی‌ها در زمان پیش‌بینی، خروجی این نورون برابر با a باشد.

خروجی این نورون در زمان آموزش به طور متوسط چه مقدار است؟ (اگر $p = 0.5$)

حذف تصادفی در زمان پیش‌بینی...

- می‌توان این کار را با یک مرحله از محاسبات رو به جلو انجام داد! (تقریب)
  - با فعال نگاه داشتن همه نورون‌ها (بدون حذف تصادفی)



در زمان پیش‌بینی:

$$a = w0*x + w1*y$$

در زمان آموزش:

$$E[a] = \tfrac{1}{4} * (w0*0 + w1*0 + w0*0 + w1*y + w0*x + w1*0 + w0*x + w1*y)$$

$$= \tfrac{1}{4} * (2\ w0*x + 2\ w1*y)$$

$$= \tfrac{1}{2} * \mathbf{(w0*x + w1*y)}$$

اگر $p = 0.5$ باشد، خروجی در زمان پیش‌بینی ۲ برابر خروجی در زمان آموزش خواهد بود!

⇐ برای جبران این مسئله باید مقدار فعالیت نورون در زمان پیش‌بینی در ضریب $1/2$ ضرب شود.

حذف تصادفی در زمان پیش‌بینی...


```
def predict(X):
    # ensemble forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) * p  # NOTE: scale the activations
    H2 = np.maximum(0, np.dot(W2, H1) + b2) * p  # NOTE: scale the activations
    out = np.dot(W3, H2) + b3
```

- در زمان پیش‌بینی، همه نورون‌ها همواره فعال هستند:
- بنابراین باید به گونه‌ای مقادیر فعالیت نورون‌ها را تغییر دهیم که:
- خروجی در زمان پیش‌بینی = متوسط خروجی در زمان آموزش

حذف تصادفی

```python
p = 0.5  # probability of keeping a unit active

def train_step(X):
    """ X contains the data """

    # forward pass for a 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = np.random.rand(*H1.shape) < p  # first dropout mask!
    H1 *= U1  # drop!

    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = np.random.rand(*H2.shape) < p # second dropout mask!
    H2 *= U2  # drop!

    out = np.dot(W3, H2) + b3

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)

def predict(X):
    # ensemble forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) * p  # NOTE: scale the activations
    H2 = np.maximum(0, np.dot(W2, H1) + b2) * p  # NOTE: scale the activations
    out = np.dot(W3, H2) + b3
```

حذف تصادفی در زمان آموزش

مقیاس‌بندی در زمان پیش‌بینی

این پیاده‌سازی توصیه نمی‌شود!

حذف تصادفی

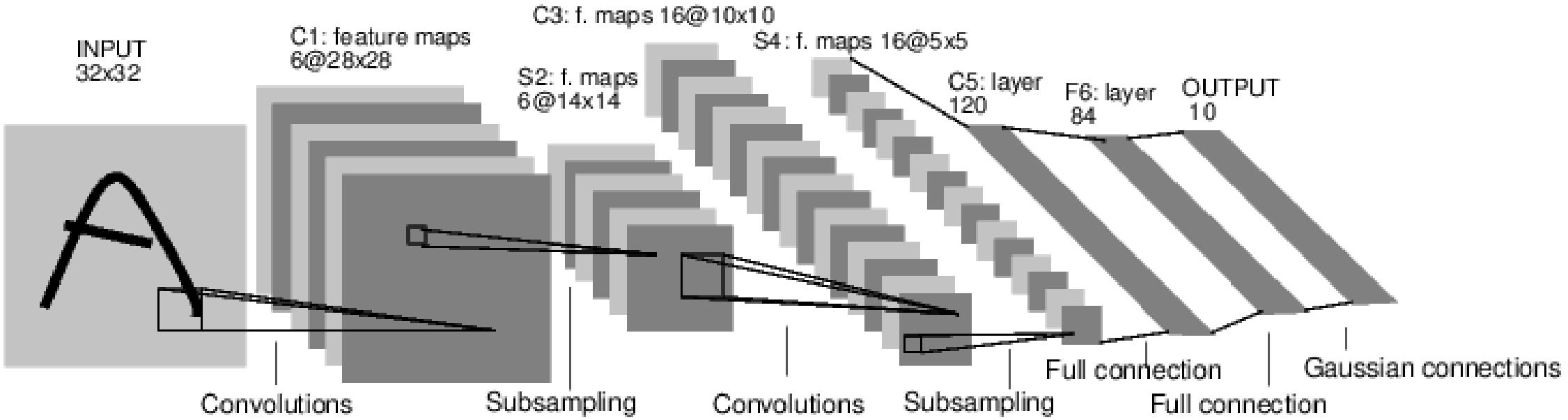
پیاده‌سازی متداول

```
p = 0.5  # probability of keeping a unit active

def train_step(X):
    """ X contains the data """

    # forward pass for a 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = (np.random.rand(*H1.shape) < p) / p  # first dropout mask! Notice /p
    H1 *= U1  # drop!

    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask! Notice /p
    H2 *= U2  # drop!

    out = np.dot(W3, H2) + b3

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)

def predict(X):
    # ensemble forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1)  # NOTE: no scaling necessary
    H2 = np.maximum(0, np.dot(W2, H1) + b2)  # NOTE: no scaling necessary
    out = np.dot(W3, H2) + b3
```

پیش‌بینی بدون تغییر!

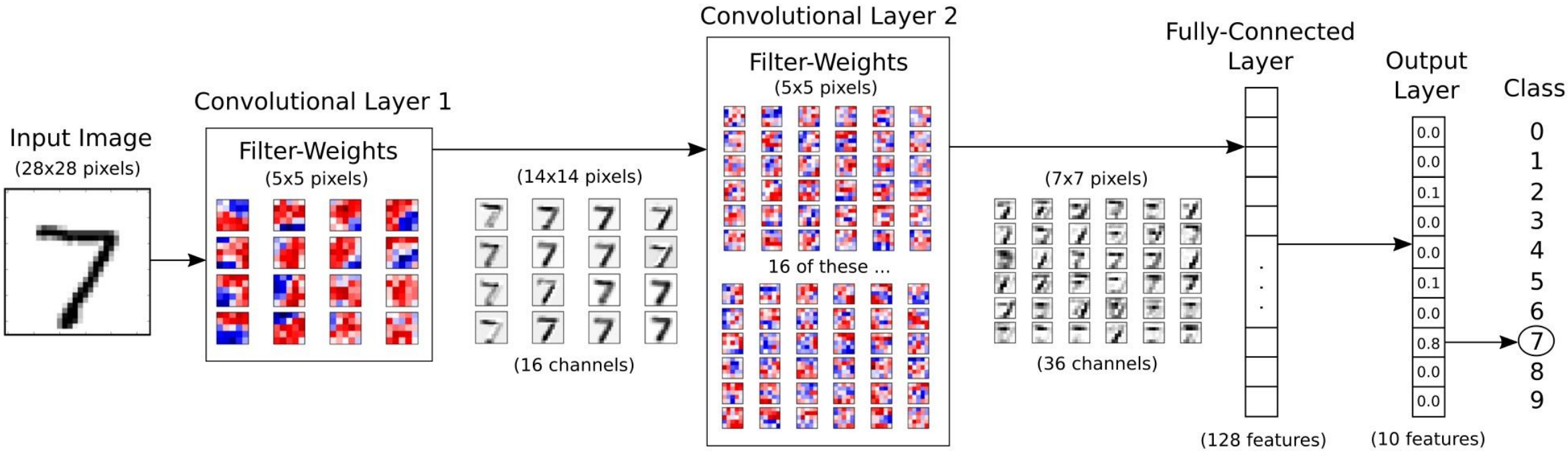
شبکه‌های عصبی کانولوشنال

شبکه‌های عصبی کانولوشنال



[LeNet-5, LeCun 1980]

شبکه‌های عصبی کانولوشنال

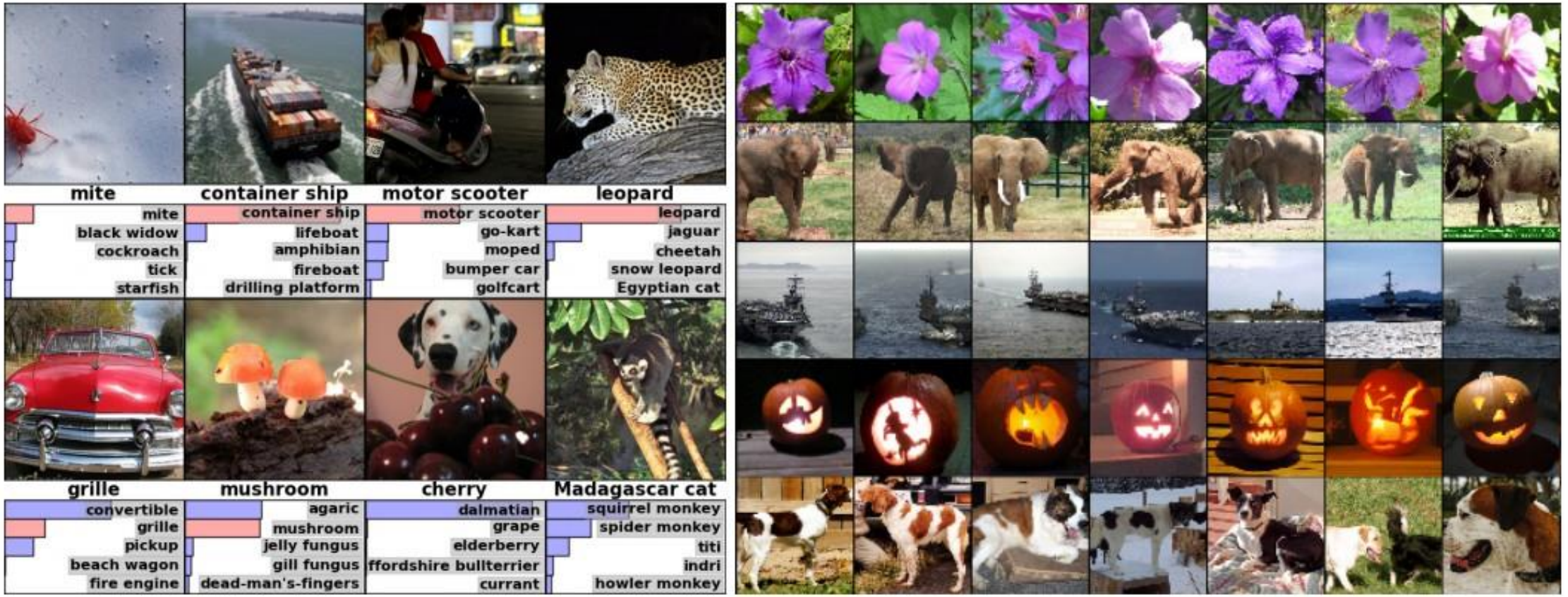


[LeNet-5, LeCun 1980]

شبکه‌های عصبی کانولوشنال

دسته‌بندی

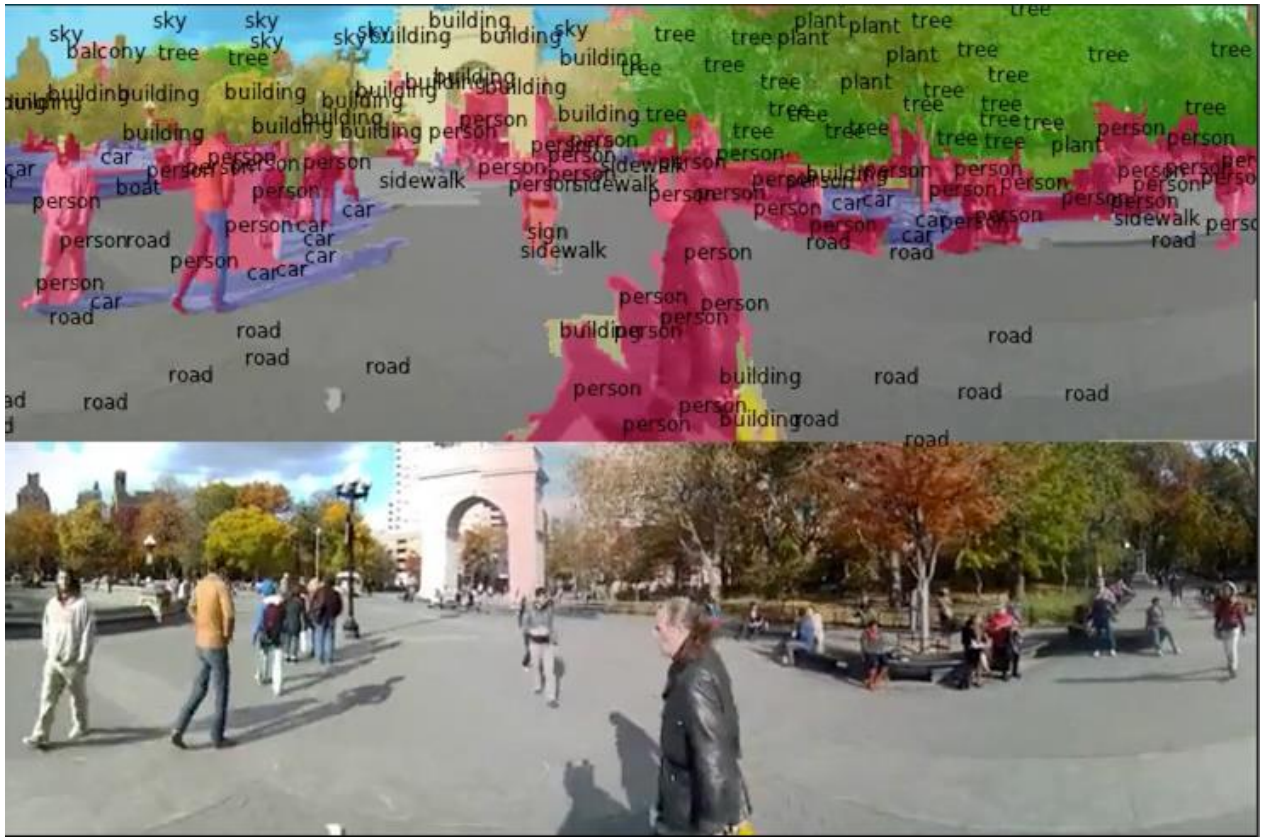
بازیابی

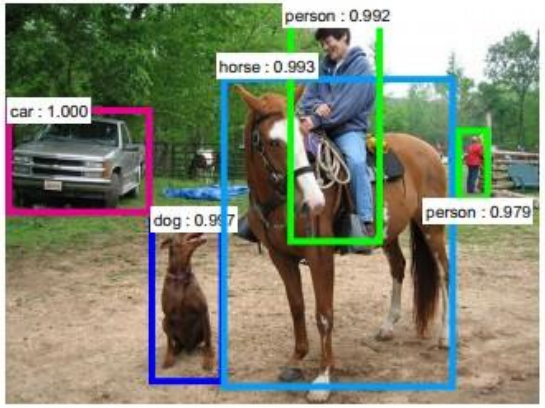


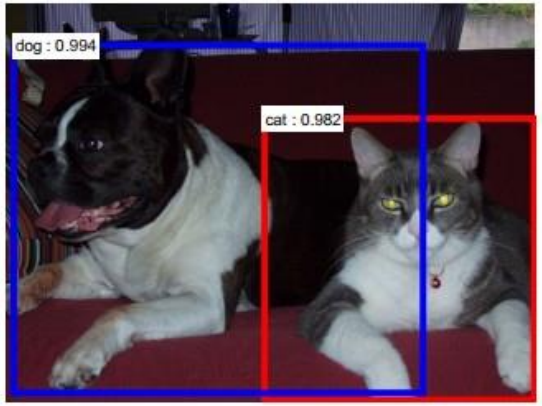
شبکه‌های عصبی کانولوشنال

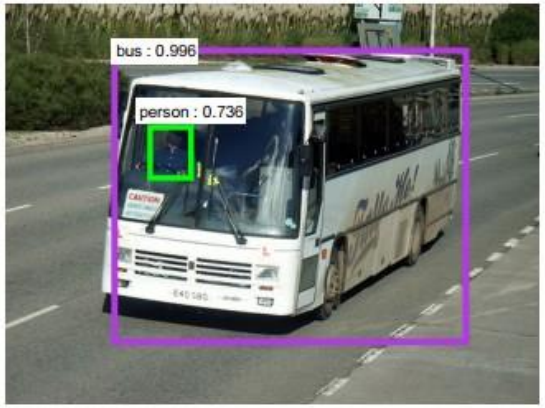
قطعه‌بندی

شناسایی





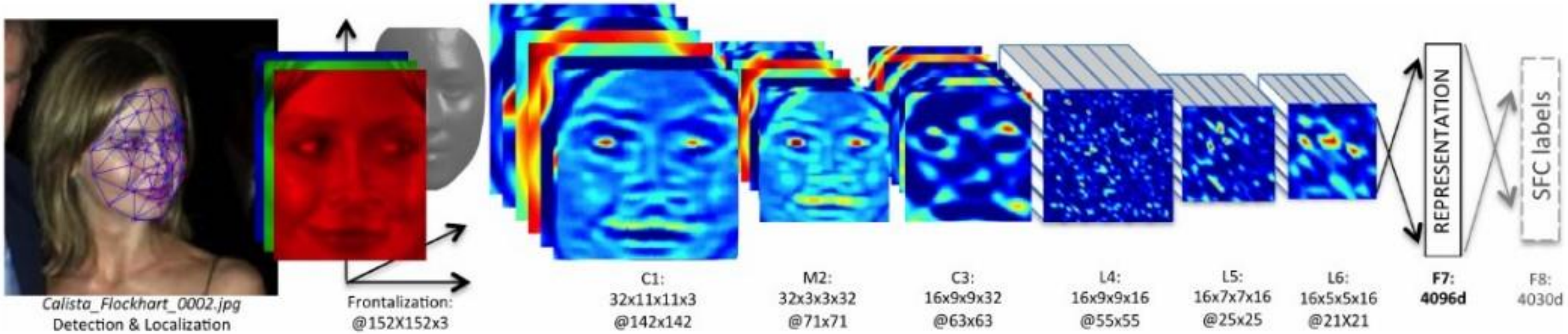


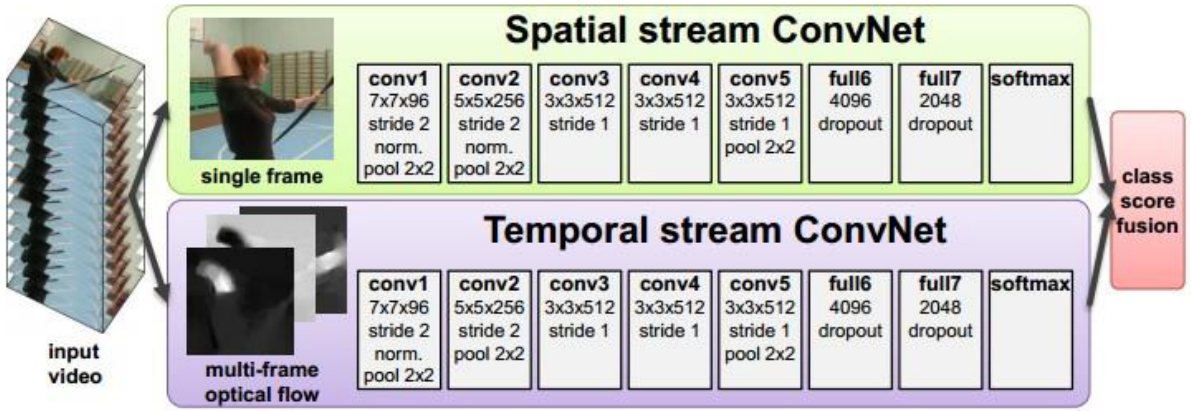


شبکه‌های عصبی کانولوشنال

خودروی بدون راننده

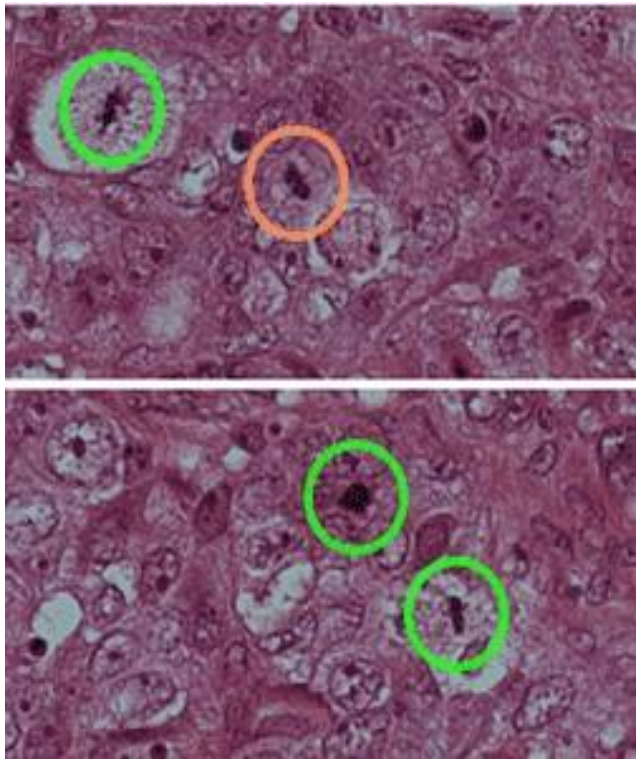
شبکه‌های عصبی کانولوشنال



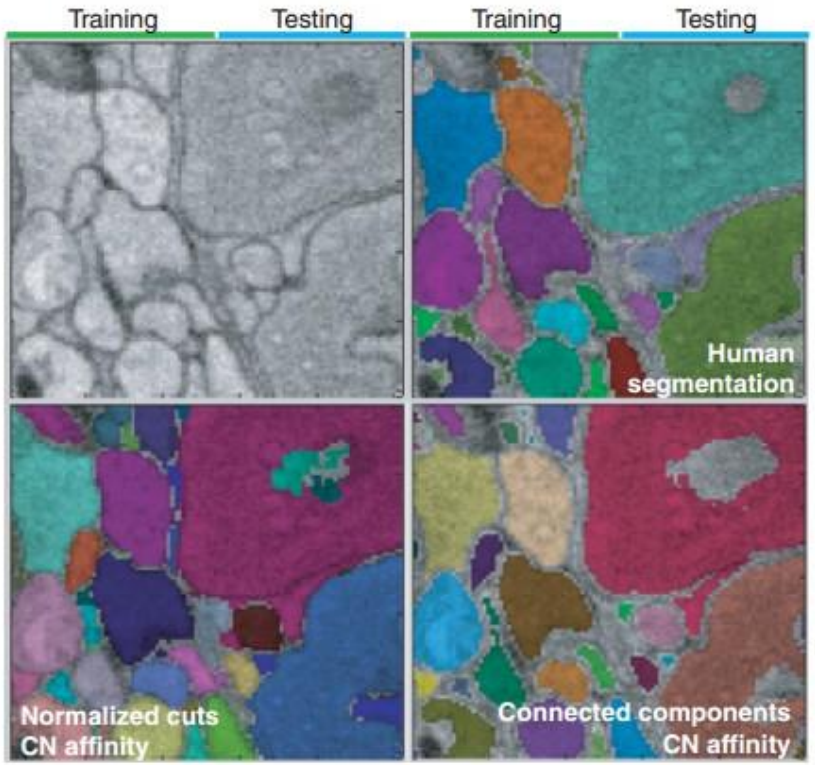


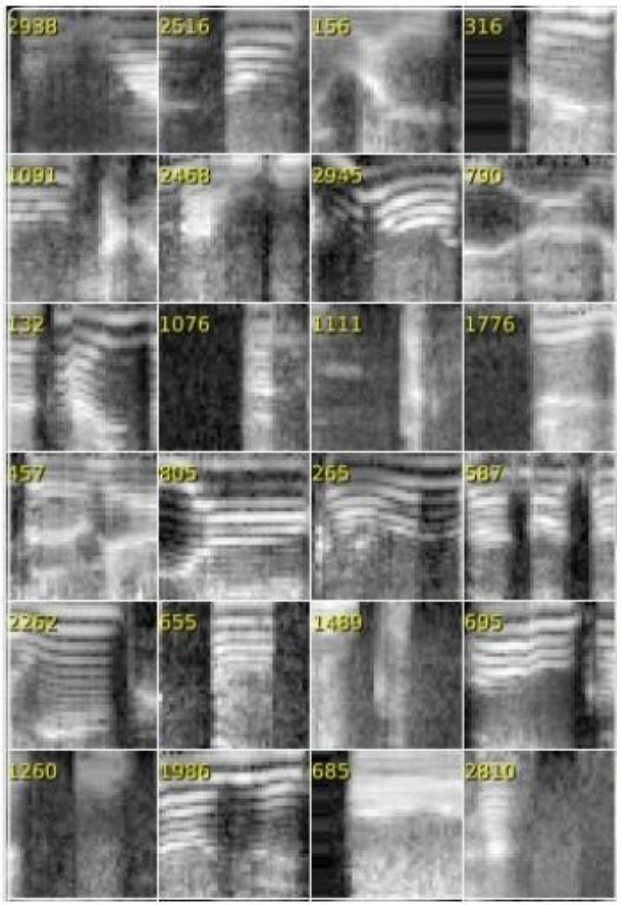
شبکه‌های عصبی کانولوشنال

شبکه‌های عصبی کانولوشنال



شبکه‌های عصبی کانولوشنال



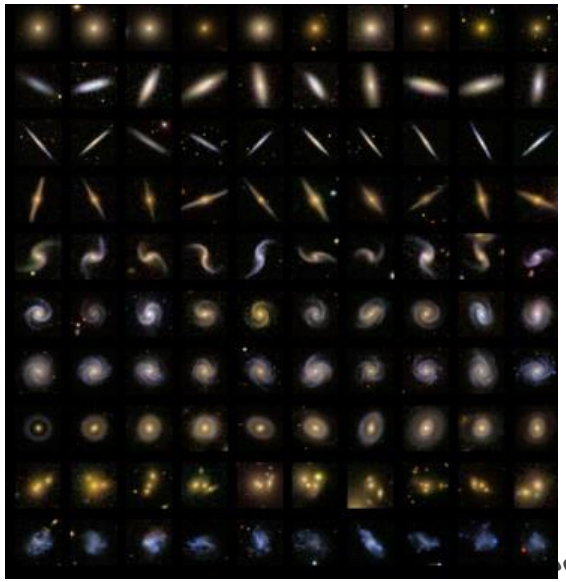


I caught this movie on the Sci-Fi channel recently. It actually turned out to be pretty decent as far as B-list horror/suspense films go. **Two guys (one naive and one loud mouthed a **) take a road trip to stop a wedding but have the worst possible luck when a maniac in a freaky, make-shift tank/truck hybrid decides to play cat-and-mouse with them.** Things are further complicated when they pick up a ridiculously whorish hitchhiker. What makes this film unique is that the combination of comedy and terror actually work in this movie, unlike so many others. The two guys are likable enough and there are some good chase/suspense scenes. Nice pacing and comic timing make this movie more than passable for the horror/slasher buff. **Definitely worth checking out.**

I just saw this on a local independent station in the New York City area. **The cast showed promise but when I saw the director, George Cosmotos, I became suspicious. And sure enough, it was every bit as bad, every bit as pointless and stupid as every George Cosmotos movie I ever saw.** He's like a stupid man's Michael Bey – with all the awfulness that accolade promises. There's no point to the conspiracy, no burning issues that urge the conspirators on. We are left to ourselves to connect the dots from one bit of graffiti on various walls in the film to the next. Thus, the current budget crisis, the war in Iraq, Islamic extremism, the fate of social security, 47 million Americans without health care, stagnating wages, and the death of the middle class are all subsumed by the sheer terror of graffiti. A truly, stunningly idiotic film.

Graphics is far from the best part of the game. **This is the number one best TH game in the series.** Next to Underground. **It deserves strong love. It is an insane game.** There are massive levels, massive unlockable characters... it's just a massive game. **Waste your money on this game. This is the kind of money that is wasted properly.** And even though graphics suck, thats doesn't make a game good. Actually, the graphics were good at the time. Today the graphics are crap. WHO CARES? As they say in Canada, This is the fun game, aye. (You get to go to Canada in THPS3) Well, I don't know if they say that, but they might. who knows. Well, Canadian people do. Wait a minute, I'm getting off topic. This game rocks. Buy it, play it, enjoy it, love it. It's PURE BRILLIANCE.

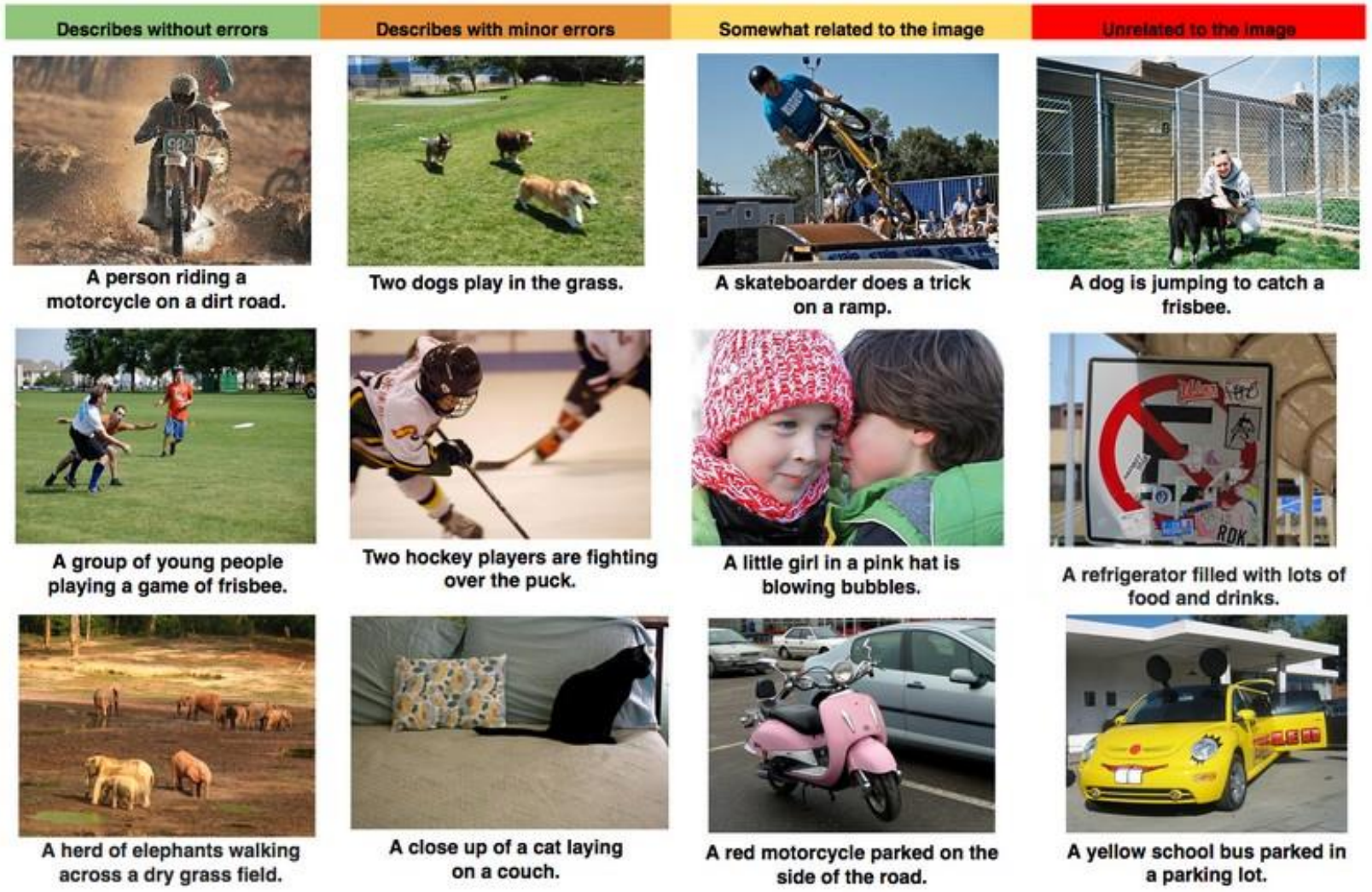
The first was good and original. I was a not bad horror/comedy movie. So I heard a second one was made and I had to watch it . What really makes this movie work is Judd Nelson's character and the sometimes clever script. **A pretty good script for a person who wrote the Final Destination films and the direction was okay.** Sometimes there's scenes where it looks like it was filmed using a home video camera with a grainy - look. Great made - for - TV movie. **It was worth the rental and probably worth buying just to get that nice eerie feeling and watch Judd Nelson's Stanley doing what he does best.** I suggest newcomers to watch the first one before watching the sequel, just so you'll have an idea what Stanley is like and get a little history background.



شبکه‌های عصبی کانولوشنال

شبکه‌های عصبی کانولوشنال

عنوان‌بندی تصاویر



شبکه‌های عصبی کانولوشنال