یادگیری عمیق

مدرس: محمدرضا محمدی

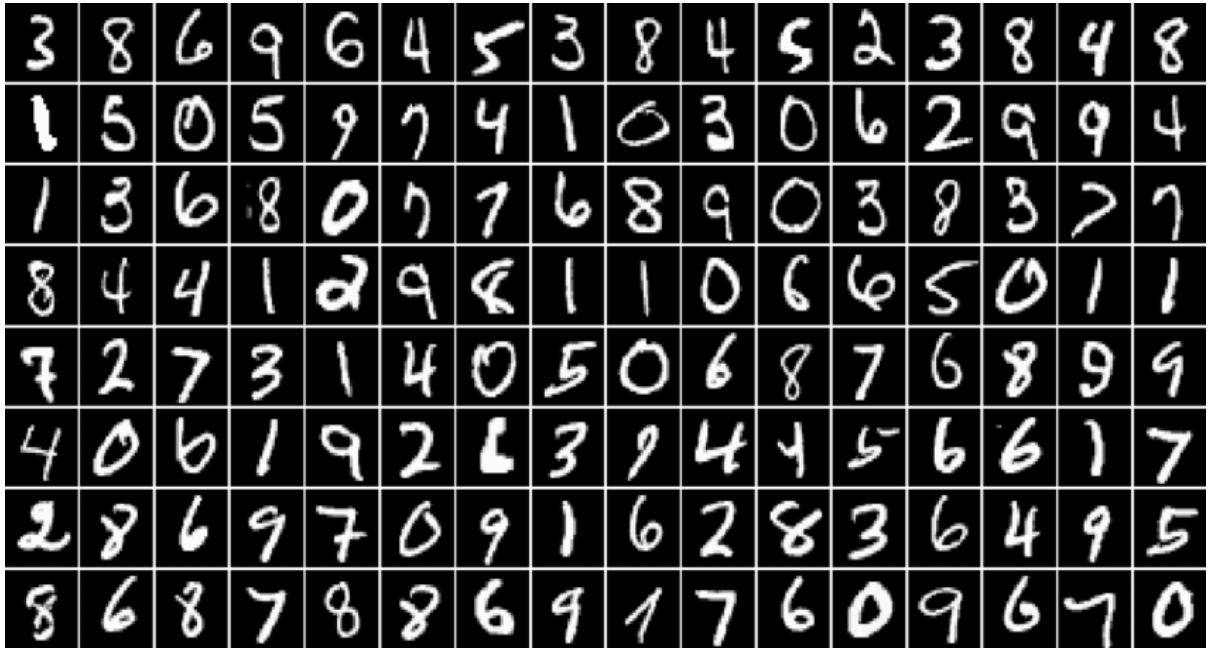
زمستان ۱۴۰۱

منظم‌سازی

Regularization

بیش‌برازش

- در سال ۲۰۱۷ گروهی از پژوهشگران نشان دادند که شبکه‌های عمیق با ظرفیت یادگیری بسیار بالا، می‌توانند حتی تصاویر با برچسب‌های تصادفی را آموزش ببینند
- علی‌رغم عدم وجود الگوی واقعی که ورودی‌ها را به خروجی‌ها مرتبط کند، پارامترهای شبکه عصبی می‌تواند بهینه شود
- برای حالت ۱۰ کلاسه تا ۹۰٪ فاصله بین آموزش و آزمون می‌تواند وجود داشته باشد

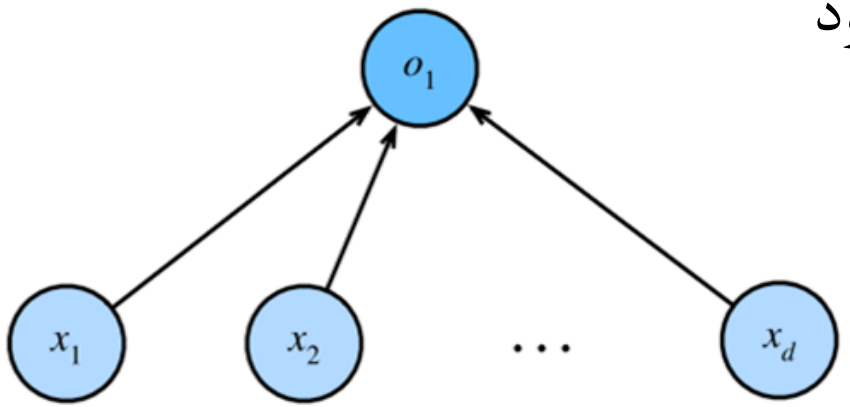


جریمه اندازه پارامترها

- به منظور محدود کردن پیچیدگی مدل، می‌توانیم محدودیت‌هایی بر روی پارامترهای مدل اعمال کنیم
- تابعی که خروجی آن یک عدد ثابت برای همه ورودی‌ها باشد، ساده‌ترین تابع است و می‌توانیم میزان پیچیدگی یک تابع را در مقایسه با آن بسنجیم
- فرض کنید $L(\mathbf{w},b)$ یکی از توابع ضرر باشد که خطای پیش‌بینی $\hat{y}^{(i)}$ها را محاسبه می‌کند
- تابع ضرر جدید با اعمال جریمه بر روی پارامترها:

$$\tilde{L}(\mathbf{w},b) = L(\mathbf{w},b) + \lambda\,\Omega(\mathbf{w})$$

- معمولا محدودیتی روی بایاس‌ها (به خصوص لایه آخر) اعمال نمی‌شود
- به λ ثابت منظم‌سازی گفته می‌شود
- با چه رابطه‌ای فاصله تا صفر را اندازه بگیریم؟



منظم‌سازی پارامتر L2

- اگر بجای L2 از فاصله اقلیدوسی استفاده کنیم محاسبه مشتق پیچیده می‌شود

$$\Omega(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2$$

$$\tilde{L}(\mathbf{w}, b) = L(\mathbf{w}, b) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

- این استراتژی وزن‌های شبکه را به سمت مبدا نزدیک می‌کند

- گرادیان تابع ضرر منظم‌شده:

$$\nabla_{\mathbf{w}}\tilde{L}(\mathbf{w}, b) = \nabla_{\boldsymbol{w}} L(\mathbf{w}, b) + \lambda\mathbf{w}$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta\big(\lambda\mathbf{w} + \nabla_{\mathbf{w}} L(\mathbf{w}, b)\big)$$

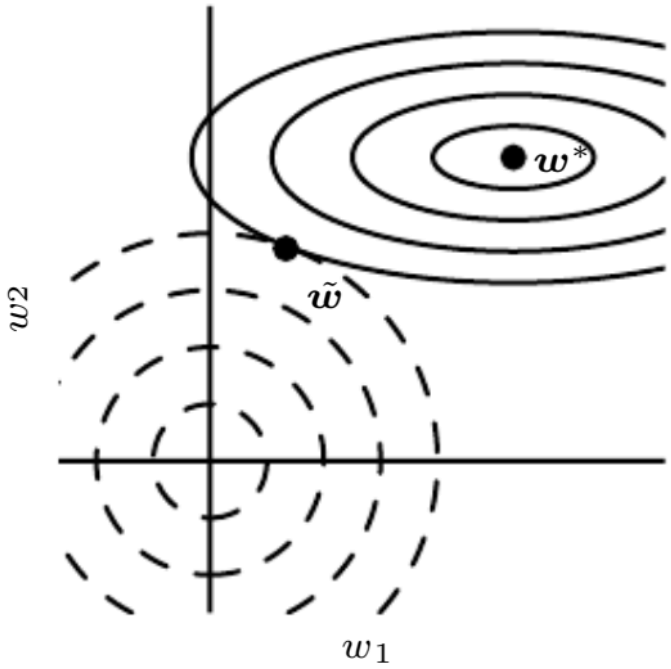
$$\mathbf{w} \leftarrow (1 - \eta\lambda)\mathbf{w} - \eta\nabla_{\mathbf{w}} L(\mathbf{w}, b)$$

- به این روش، Weight Decay هم گفته می‌شود

منظم‌سازی پارامتر L2

$$\Omega(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2$$

$$\mathbf{w} \leftarrow (1 - \eta\lambda)\mathbf{w} - \eta\nabla_{\mathbf{w}} L(\mathbf{w}, b)$$



- قبل از انجام به‌روزرسانی معمولی مبتنی بر گرادیان، بردار وزن را در هر گام با یک ضریب ثابت کاهش می‌دهد
- وزن‌هایی که تغییر کمتری در تابع ضرر ایجاد می‌کنند، اهمیت کمتری دارند و بیشتر کاهش می‌یابند

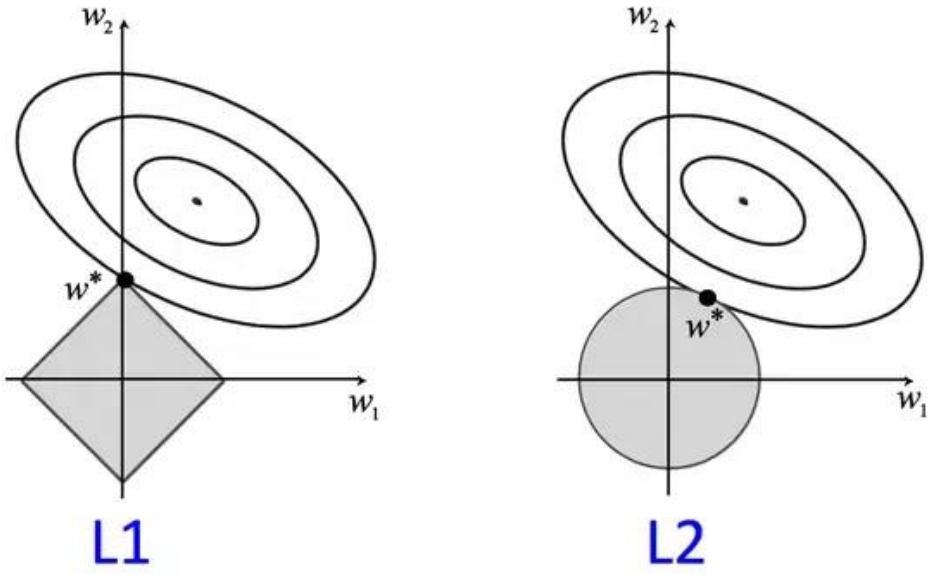
منظم‌سازی پارامتر L1

- منظم‌سازی L1 بر روی پارامترهای شبکه $\mathbf{w}$ به صورت زیر تعریف می‌شود

$$\Omega(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_i |w_i|$$

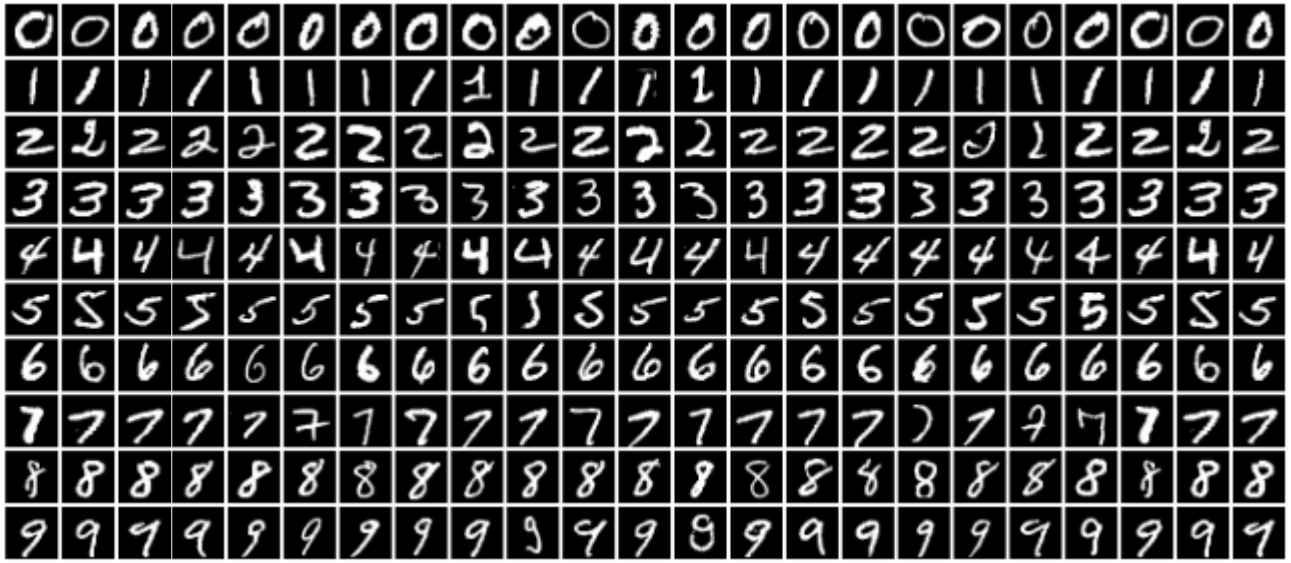
$$\tilde{L}(\mathbf{w}, b) = L(\mathbf{w}, b) + \lambda \|\boldsymbol{w}\|_1$$

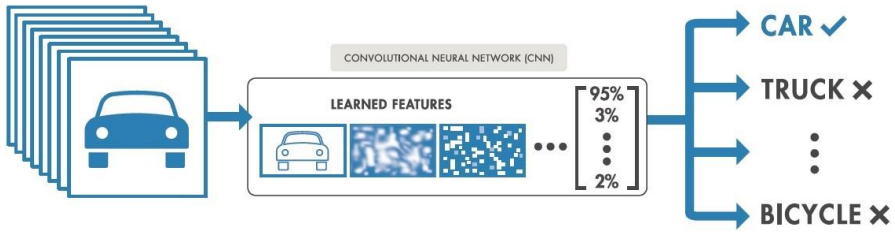
$$\nabla_{\mathbf{w}} \tilde{L}(\mathbf{w}, b) = \nabla_{\mathbf{w}} L(\mathbf{w}, b) + \lambda \operatorname{sign}(\mathbf{w})$$



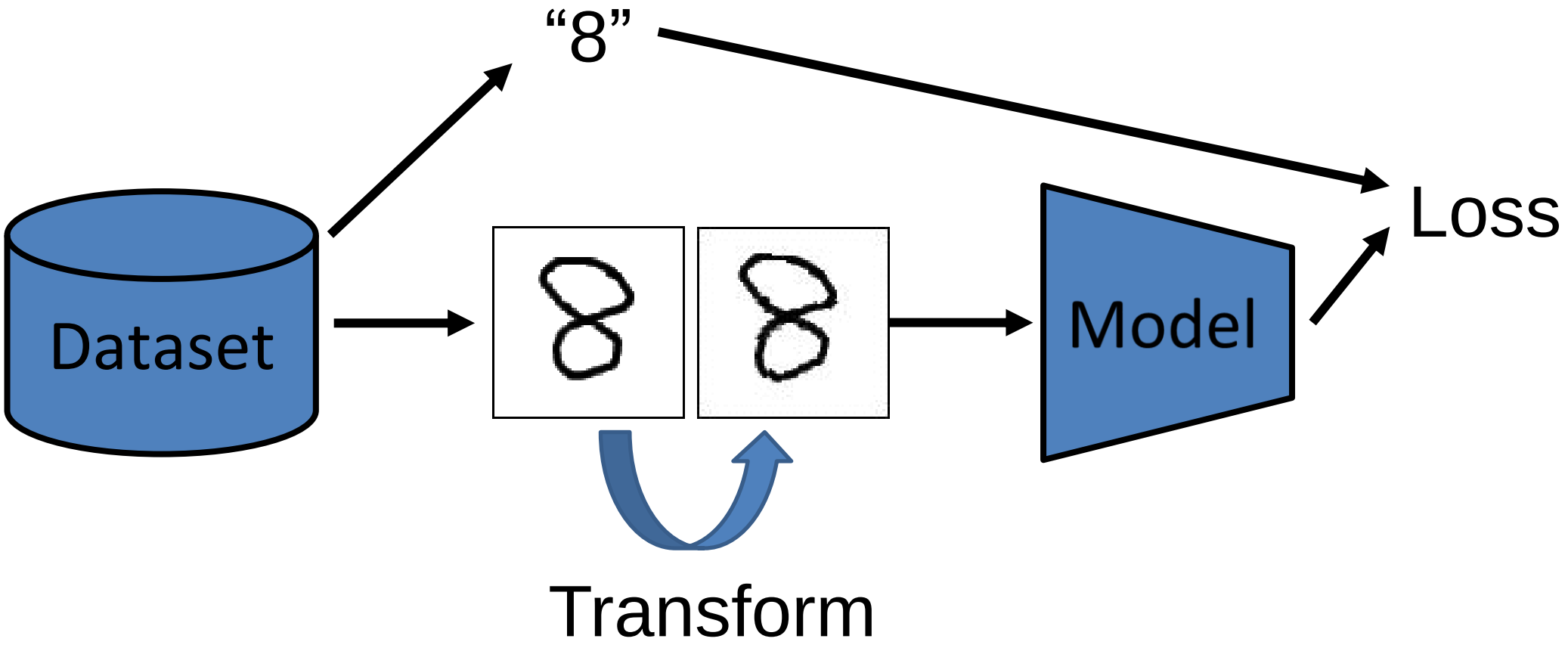
داده‌افزایی (Augmentation)

- بهترین راه برای افزایش قدرت تعمیم‌دهی یک الگوریتم یادگیری ماشین با ظرفیت یادگیری بالا، آموزش آن بر روی داده‌های بیشتر است
- جمع‌آوری داده معمولا فرآیند دشوار و خسته‌کننده‌ای است
- می‌توانیم داده‌های ساختگی بسازیم و به داده‌های آموزشی اضافه کنیم
- این کار برای مسئله دسته‌بندی راحت‌ترین است
- می‌توانیم جفت‌های (x, y) جدید را به سادگی و تنها با تبدیل x بسازیم

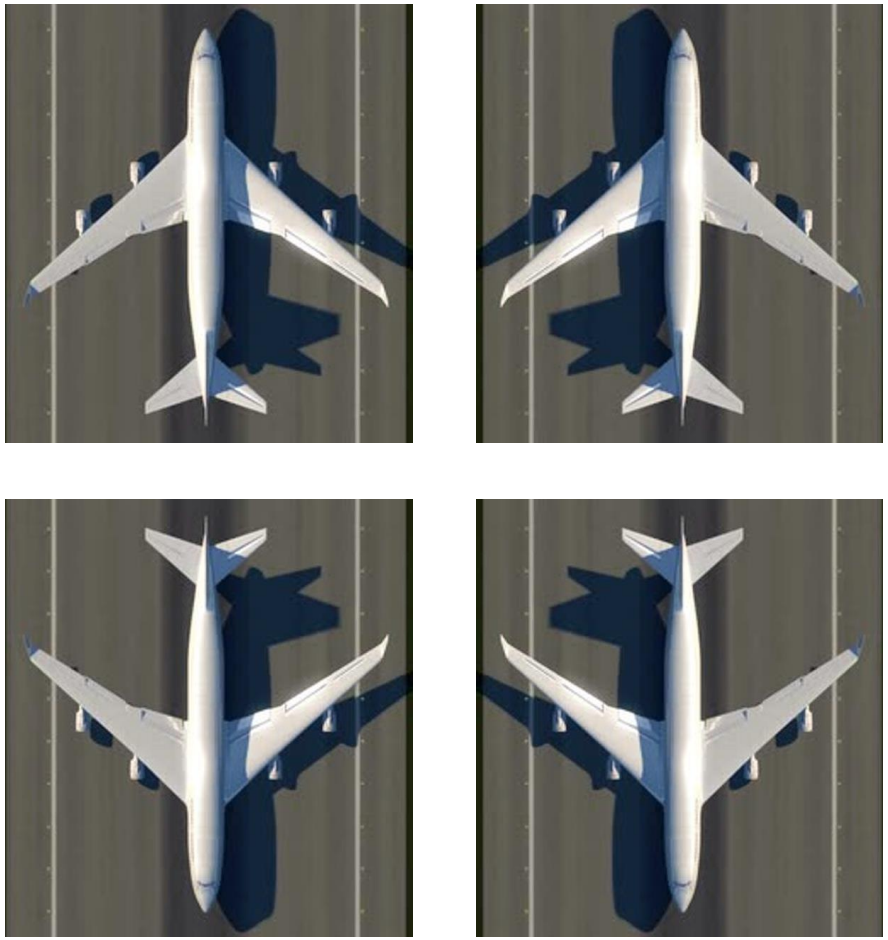




داده‌افزایی



داده‌افزایی: flip

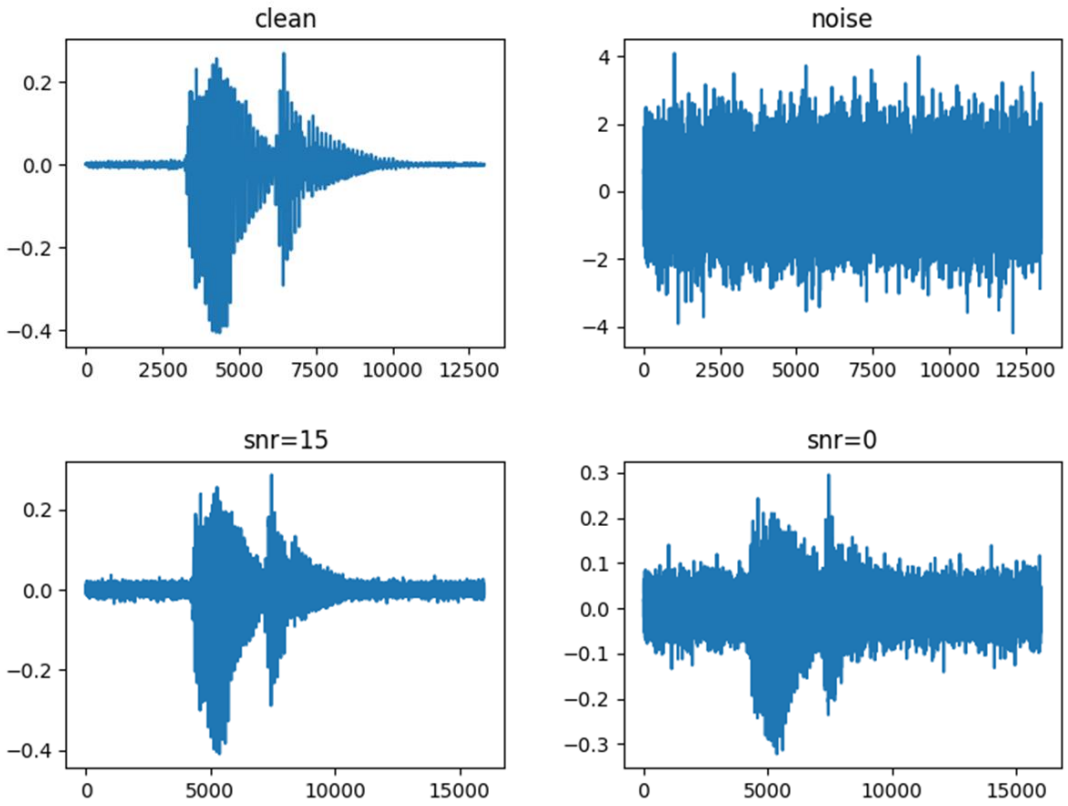


b d

p q

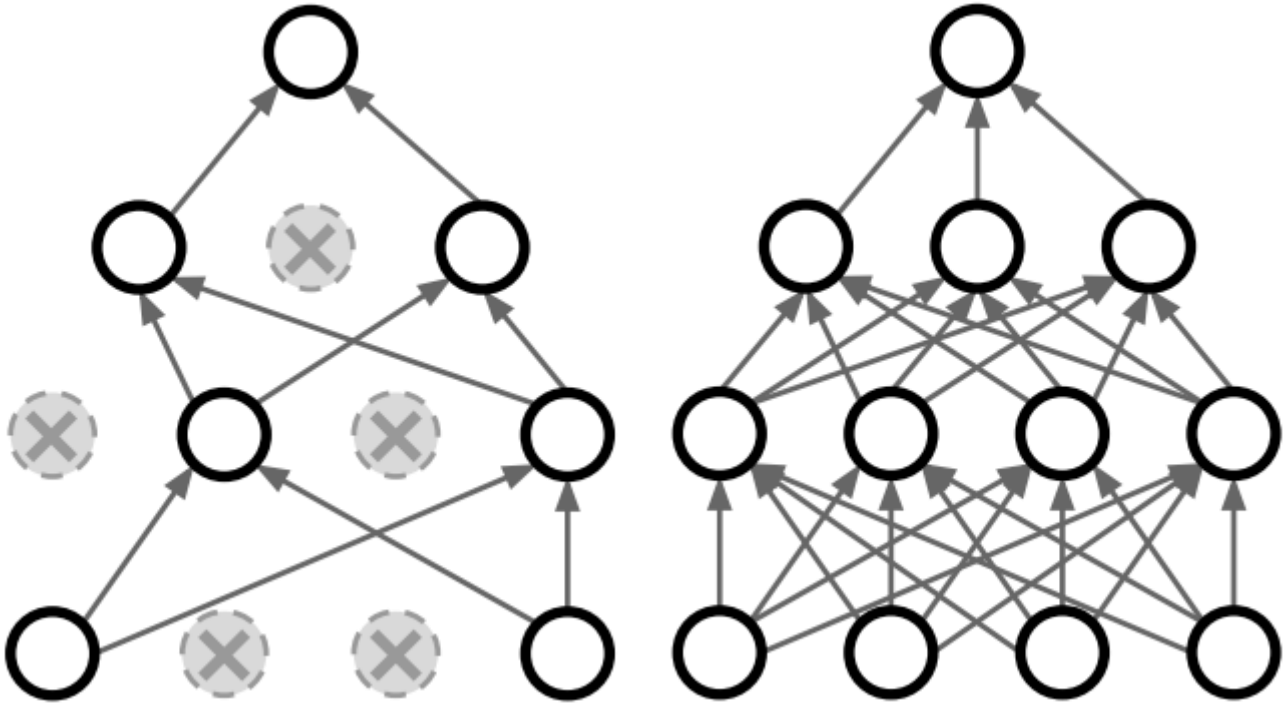
داده‌افزایی: افزودن نویز

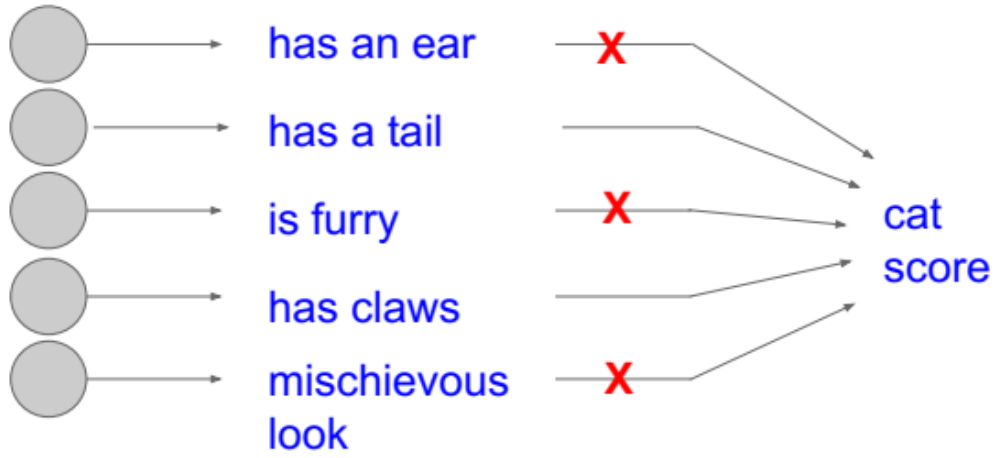
- افزودن نویز در ورودی به یک شبکه عصبی می‌تواند به عنوان نوعی داده‌افزایی در نظر گرفته شود
- برای بسیاری از مسائل دسته‌بندی و حتی برخی از مسائل رگرسیون، با افزودن مقدار محدودی نویز به ورودی همچنان می‌توان همان خروجی را توقع داشت
- افزودن نویز می‌تواند در لایه‌های میانی شبکه نیز انجام شود



Dropout

- در هر تکرار، به صورت تصادفی مقدار تعدادی نورون را صفر می‌کند
  - مشابه با نویز ضرب‌شونده با مقادیر باینری است
- احتمال حذف هر واحد یک اَبَرپارامتر است
  - مقدار 0.5 متداول است

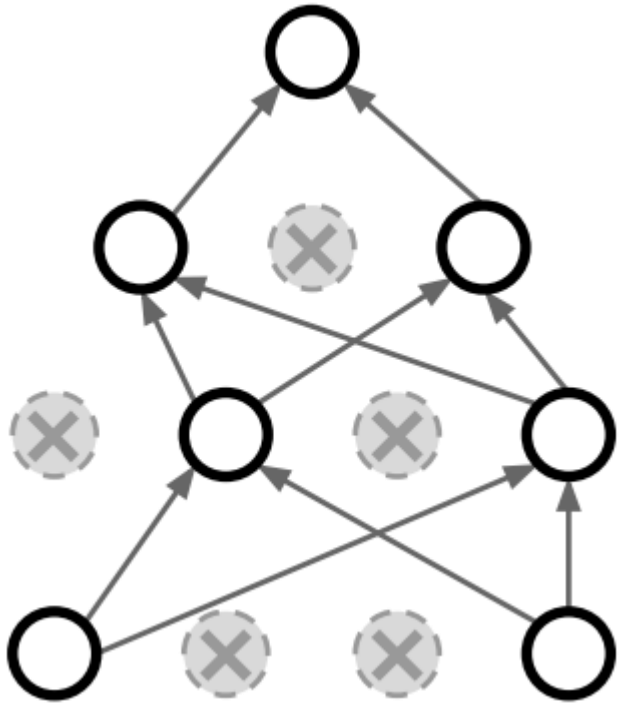




Dropout

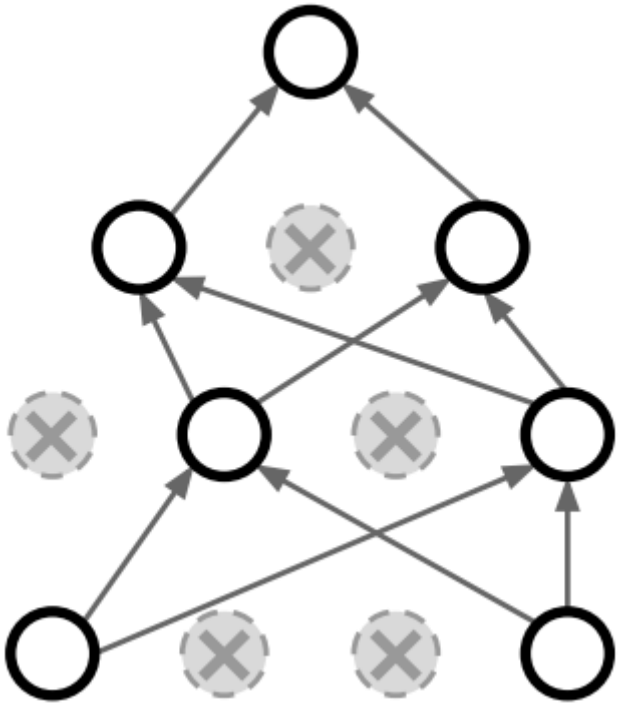
```
p = 0.5 # probability of keeping a unit active. higher = less dropout
def train_step(X):
  """ X contains the data """
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, 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)
```



Dropout

- با استفاده از Dropout، یک مجموعه (ensemble) بزرگ از مدل‌ها آموزش می‌بینند که دارای پارامترهای مشترک هستند
  - هر ماسک باینری یک مدل است
- یک لایه FC با ۴۰۹۶ واحد دارای $۲^{۴۰۹۶} \approx ۱۰^{۱۲۳۳}$ ماسک متفاوت است!
- در زمان تست از کدام ماسک استفاده کنیم؟
  - اگر در زمان تست هم ماسک تصادفی انتخاب شود، خروجی تصادفی می‌شود

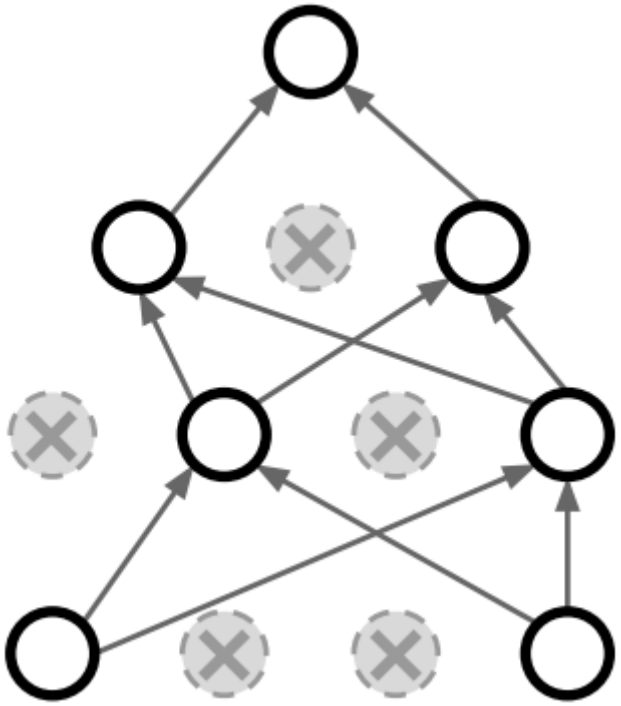


Dropout

- می‌توانیم در زمان تست امید ریاضی خروجی را محاسبه کنیم
  - میانگین وزن‌دار خروجی به ازای تمام ماسک‌های ممکن

$$y = f(x) = E_z[f(x,z)] = \sum_z p(z) f(x,z)$$

- این محاسبات بسیار سنگین است

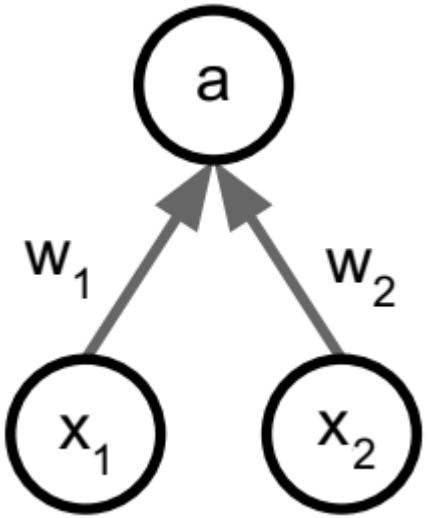


Dropout

$$y = f(x) = E_z[f(x,z)] = \sum_z p(z) f(x,z)$$

- یک نورون ساده با دو ورودی را در نظر بگیرید:

$$E[a] = p^2(w_1x_1 + w_2x_2) + p(1-p)(w_1x_1 + w_20)$$
$$+ (1-p)p(w_10 + w_2x_2) + (1-p)^2(w_10 + w_20)$$
$$= p(w_1x_1 + w_2x_2)$$



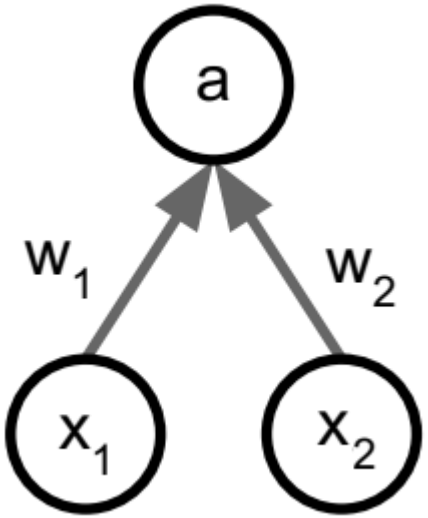
- در زمان تست، تمام نورون‌ها فعال هستند اما خروجی هر نورون را در ضریب p ضرب می‌کنیم

Dropout

$$y = f(x) = E_z[f(x,z)] = \sum_z p(z) f(x,z)$$

- یک نورون ساده با دو ورودی را در نظر بگیرید:

$$E[a] = p^2(w_1x_1 + w_2x_2) + p(1-p)(w_1x_1 + w_20)$$
$$+ (1-p)p(w_10 + w_2x_2) + (1-p)^2(w_10 + w_20)$$
$$= p(w_1x_1 + w_2x_2)$$




```
def predict(X):
  # ensembled 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
```

Dropout

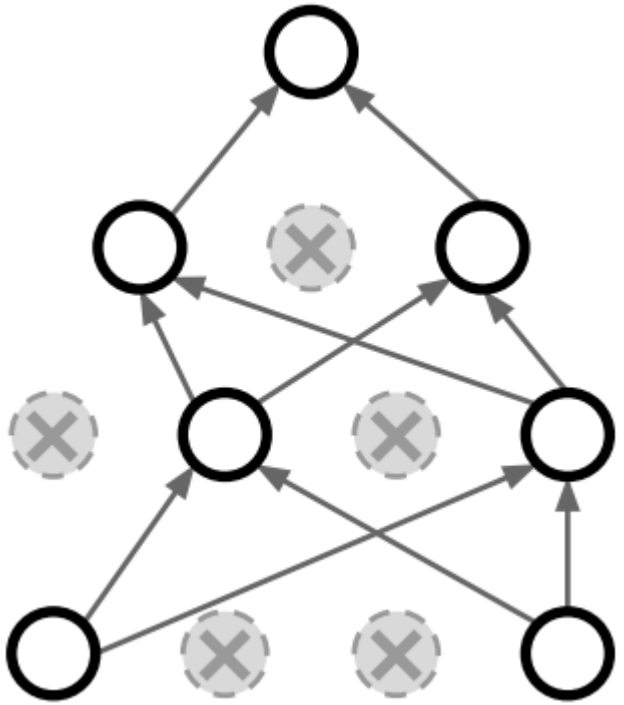
```
p = 0.5 # probability of keeping a unit active. higher = less dropout
def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) # first dropout mask.
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + 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):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # scale the activations
  out = np.dot(W3, H2) + b3
```

drop in forward pas

scale at test time



Inverted Dropout

```
p = 0.5 # probability of keeping a unit active. higher = less dropout
def train_step(X):
  # forward pass for example 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):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

drop in forward pass

test time is unchanged