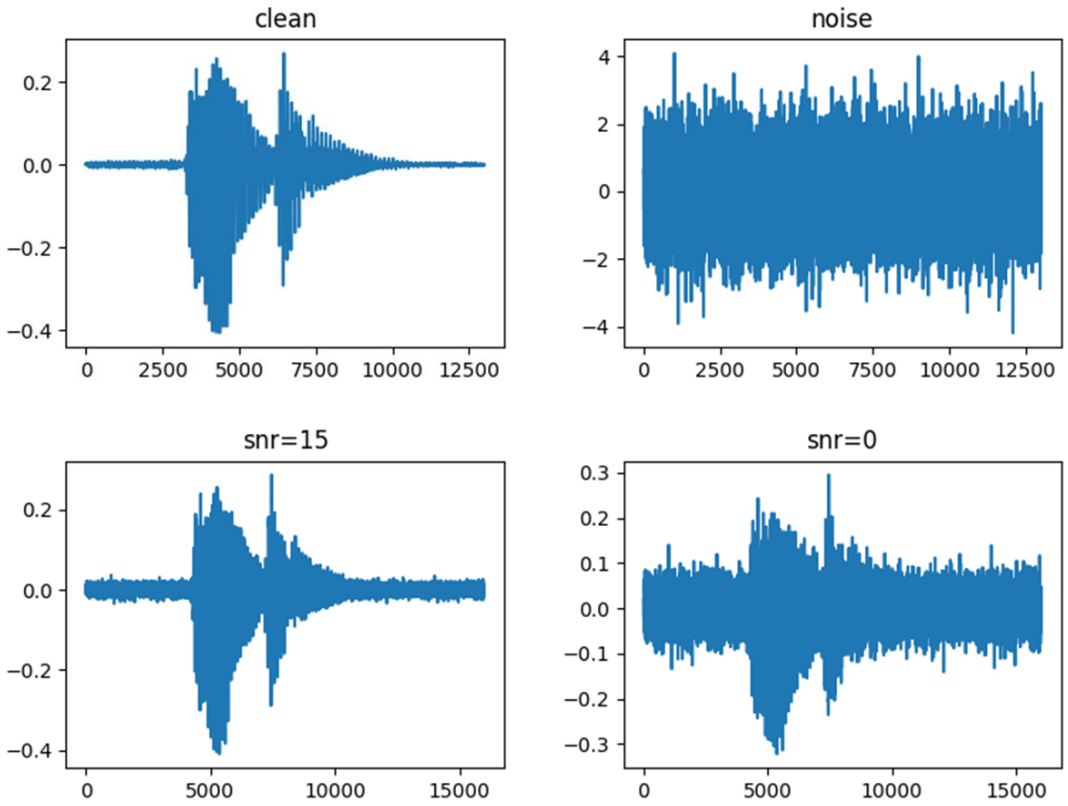

بسم الله الرحمن الرحيم

منظم‌سازی

Regularization

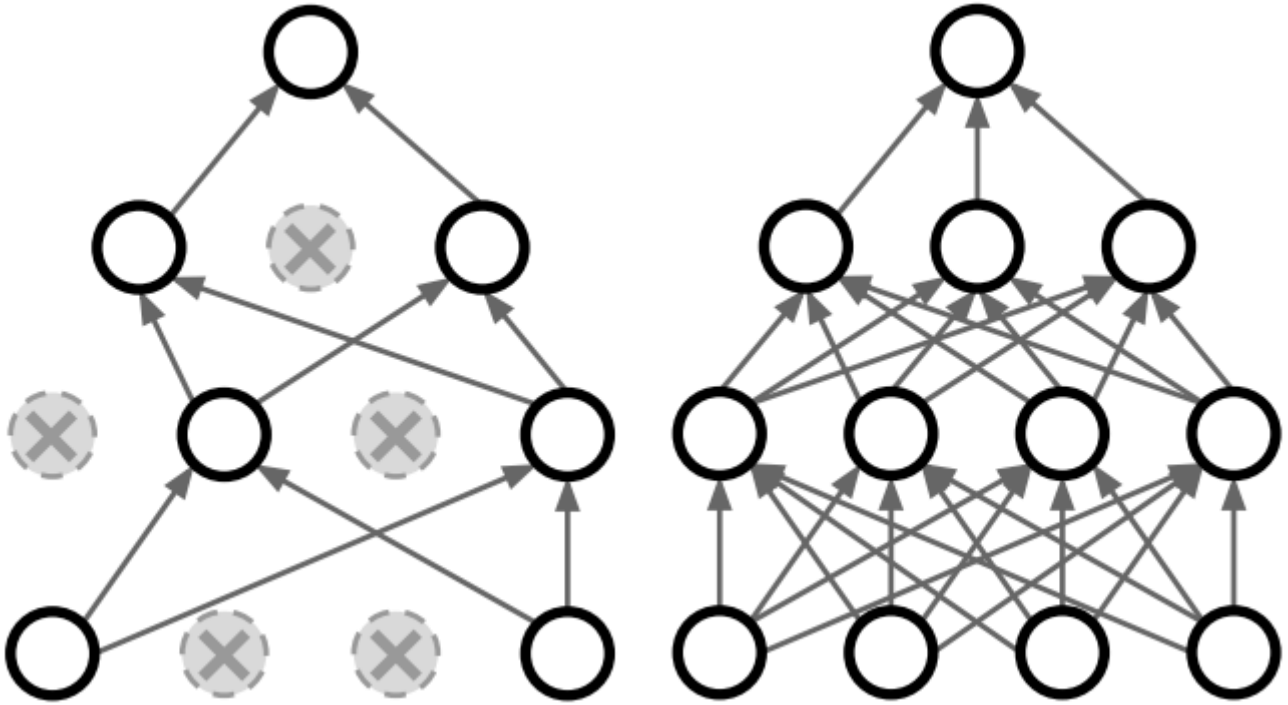
داده‌افزایی: افزودن نویز

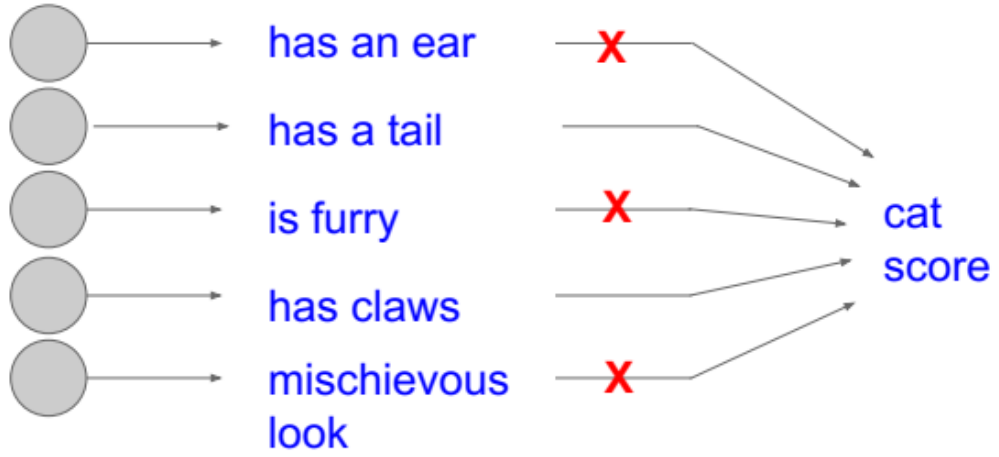
- افزودن نویز در ورودی به یک شبکه عصبی می‌تواند به عنوان نوعی داده‌افزایی در نظر گرفته شود
- برای بسیاری از مسائل دسته‌بندی و حتی برخی از مسائل رگرسیون، با افزودن مقدار محدودی نویز به ورودی همچنان می‌توان همان خروجی را توقع داشت
- افزودن نویز می‌تواند در لایه‌های میانی شبکه نیز انجام شود



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

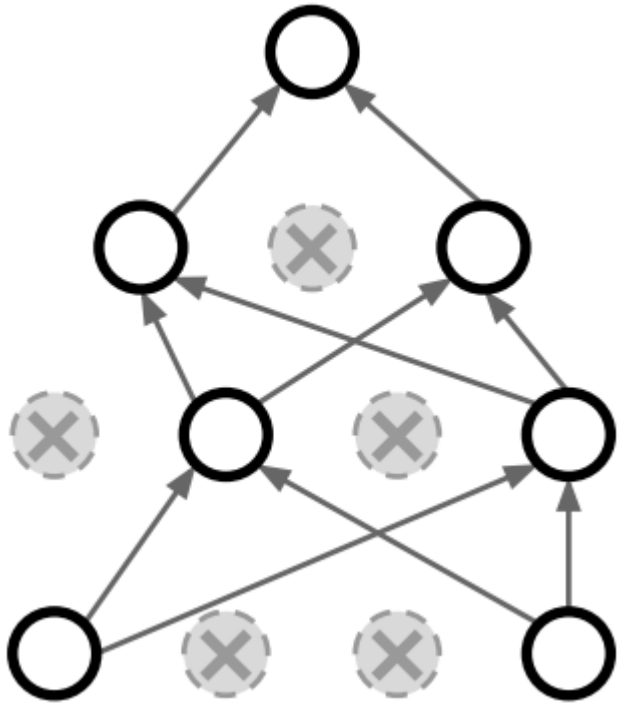




Dropout

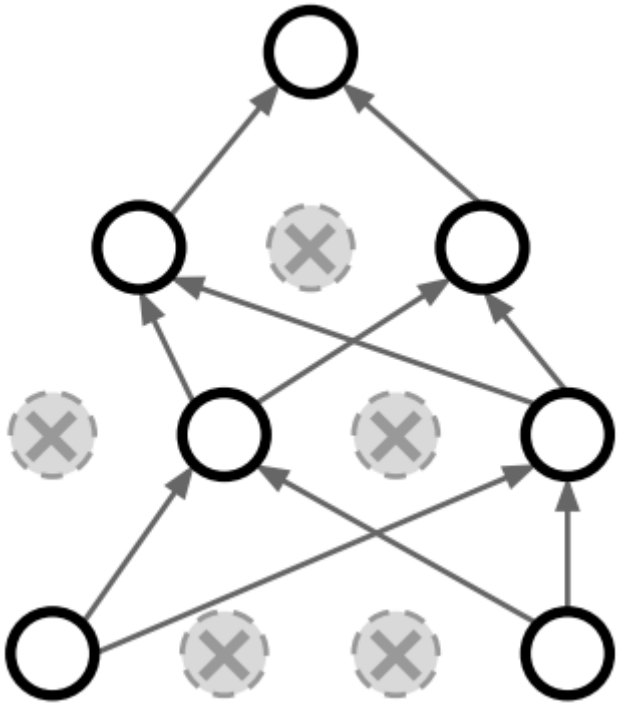
```
p = 0.5 # probability of keeping a unit active. higher = less dropout
def train_step(X):
  """ X contains the data """
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0,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

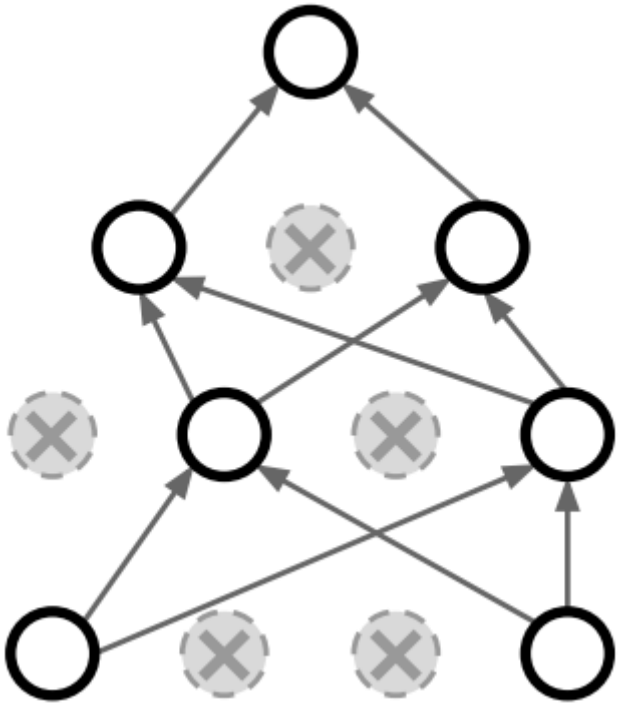


Dropout

- 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

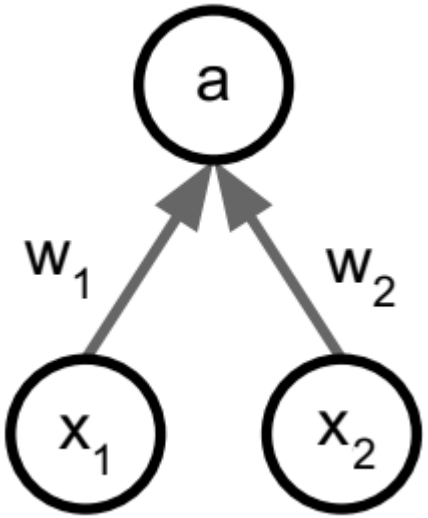


Dropout

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

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

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



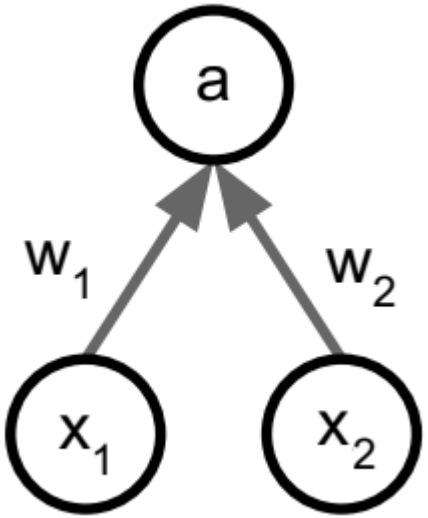
- در زمان تست، تمام نورون‌ها فعال هستند اما خروجی هر نورون را در ضریب p ضرب می‌کنیم

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

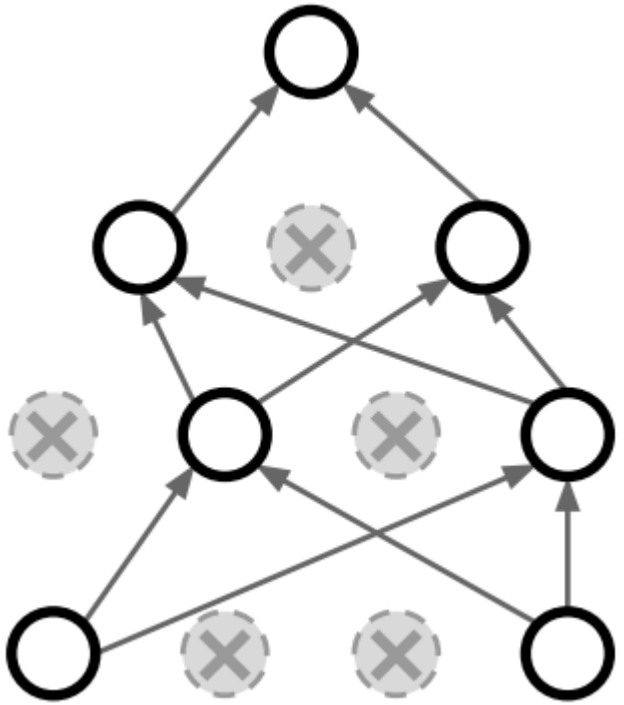
```
p = 0.5 # probability of keeping a unit active. higher = less dropout
def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) # first dropout mask.
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1)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

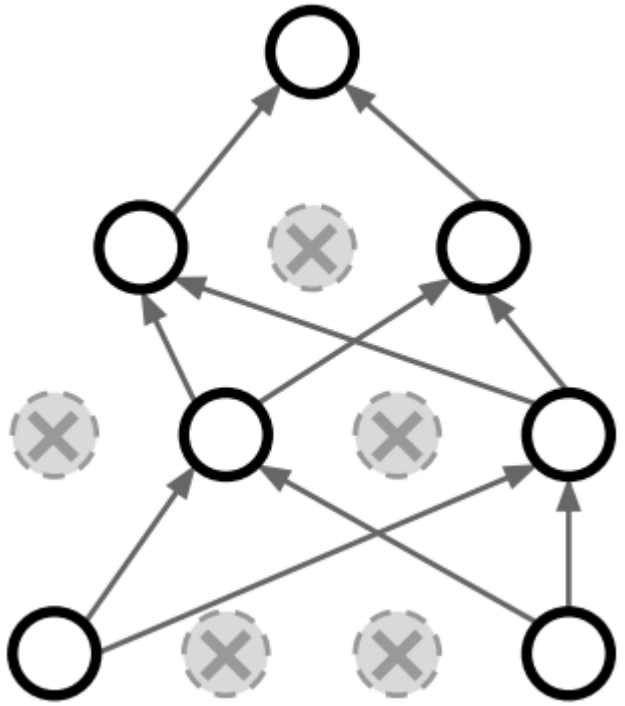
```
p = 0.5 # probability of keeping a unit active. higher = less dropout
def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.maximum(0,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

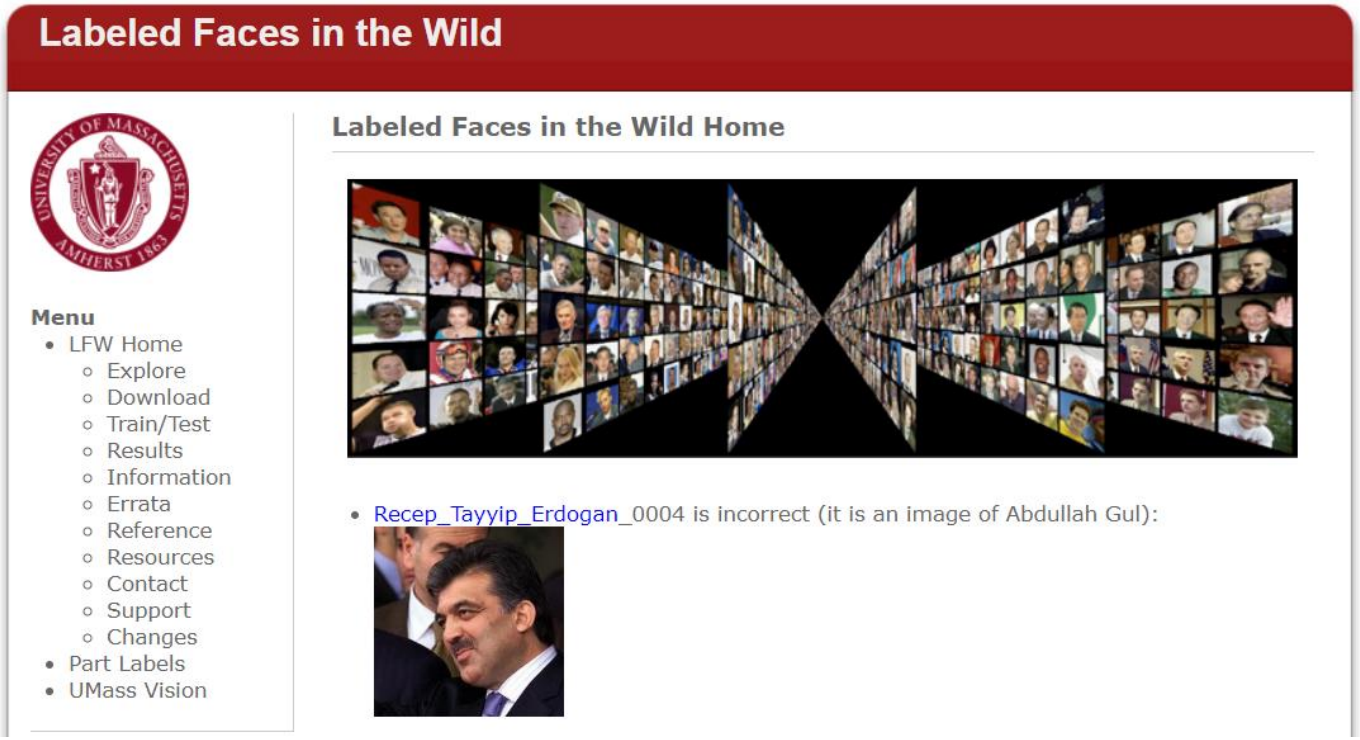
drop in forward pass

test time is unchanged



افزودن نویز در خروجی مطلوب

- اکثر مجموعه‌های داده مقداری اشتباه در برچسب‌های y دارند
- ماکزیمم کردن $\log p(y|x)$ زمانیکه y اشتباه است می‌تواند خیلی مضر باشد



افزودن نویز در خروجی مطلوب

- اکثر مجموعه‌های داده مقداری اشتباه در برچسب‌های y دارند
- ماکزیمم کردن $\log p(y|x)$ زمانیکه y اشتباه است می‌تواند خیلی مضر باشد
- می‌توانیم فرض کنیم برچسب موجود در مجموعه داده با احتمال $1-\epsilon$ درست است که ϵ یک عدد کوچک است
- با استفاده از Label Smoothing، بجای آنکه خروجی مطلوب برای دسته‌بند را مقادیر سخت ۰ و ۱ قرار دهیم، از مقادیر نرم شده $\epsilon/(k-1)$ و $1-\epsilon$ استفاده می‌کنیم

1.0	0.0	0.0

0.90	0.05	0.05

- از این مقادیر در تابع ضرر cross entropy استاندارد استفاده می‌کنیم