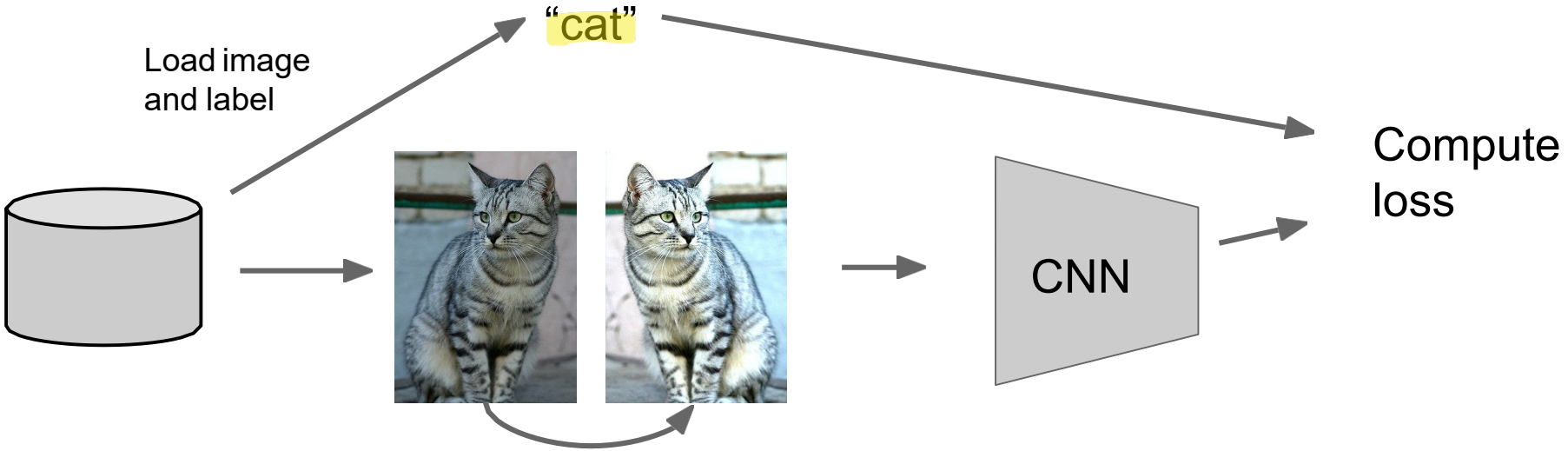
2021 Spring

Artificial Intelligence & Deep Learning

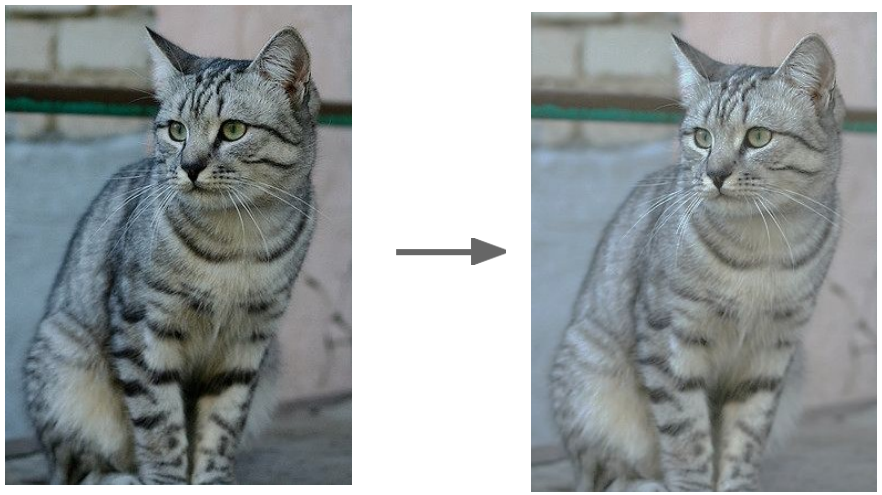
Prof. Minsuk Koo

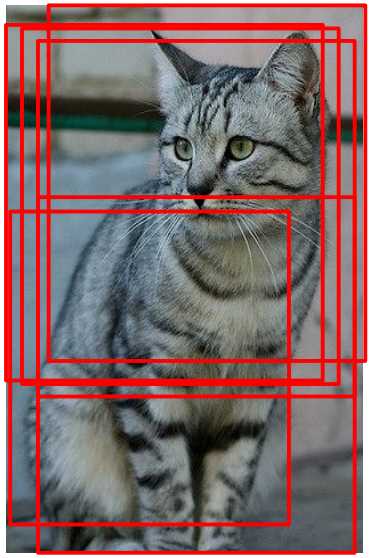
Department of Computer Science & Engineering
Incheon National University

5.4.3 데이터 확대



Transform image

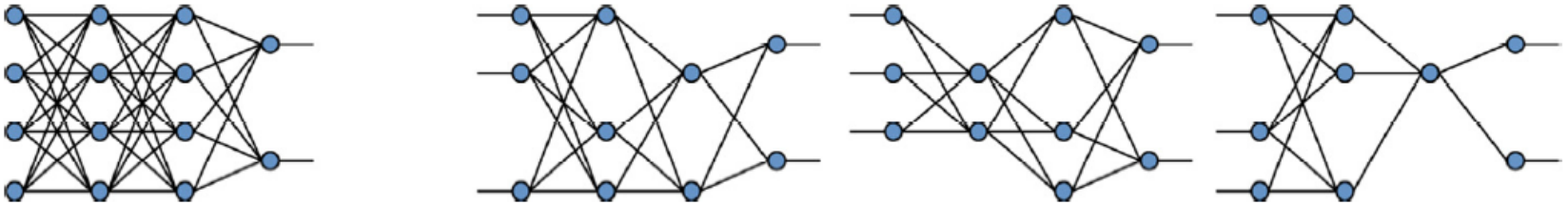




From cs231n Stanford Univ.

5.4.4 드롭아웃

- 드롭아웃 규제 기법
  - 입력층과 은닉층의 노드 중 일정 비율을 임의로 선택하여 제거
  - 남은 부분 신경망을 학습
  - 많은 부분 신경망을 만들고, 예측 단계에서 앙상블 결합하는 기법으로 볼 수 있음



(a) 원래 신경망(4–4–4–2 구조) (b) 드롭아웃된 3개의 신경망 예시

그림 5–27 드롭아웃된 신경망

  - 많은 부분 신경망을 학습하고, 저장하고, 앙상블 결합하는 데 따른 계산 시간과 메모리 공간 측면의 부담

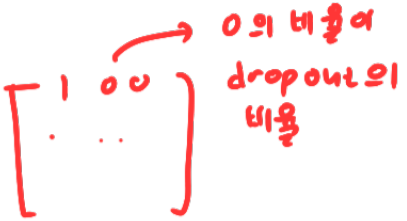
5.4.4 드롭아웃

- 실제로는 가중치 공유 사용
  - 하나의 신경망(하나의 가중치 집합)에 드롭아웃을 적용함([알고리즘 5-8])

알고리즘 5-8 드롭아웃을 채택한 기계 학습 알고리즘

입력: 드롭아웃 비율 p_{input}, p_{hidden}
출력: 최적해 $\hat{\Theta}$

1 난수를 생성하여 초기해 Θ를 설정한다.
2 while (! 멈춤 조건) // 수렴 조건
3 미니배치 $\mathbb{B}$를 샘플링한다.
4 for (i=1 to $|\mathbb{B}|$) // $\mathbb{B}$의 샘플 각각에 대해
5 입력층은 p_{input}, 은닉층은 p_{hidden} 비율로 드롭아웃을 수행한다.
6 드롭아웃된 부분 신경망 $\Theta_i^{dropout}$로 전방 계산을 한다.
7 오류 역전파를 이용하여 $\Theta_i^{dropout}$를 위한 그레이디언트 $\nabla_i^{dropout}$를 구한다.
8 $\nabla_1^{dropout}, \nabla_2^{dropout}, \cdots, \nabla_{|\mathbb{B}|}^{dropout}$의 평균 $\nabla_{ave}^{dropout}$를 계산한다.
9 $\Theta = \Theta - \rho\nabla_{ave}^{dropout}$ // 가중치 갱신
10 $\hat{\Theta} = \Theta$



앙상블 효과

5.4.4 드롭아웃

- 라인 6의 전방 계산

l번째 은닉층의 j번째 노드의 연산:

$$z_j^l = \tau_l(s_j^l)$$

이때 $s_j^l = \mathbf{u}_j^l \mathbf{z}^{l-1}$

$\Rightarrow$

드롭아웃 적용:

$$z_j^l = \tau_l(s_j^l)$$

이때 $\begin{cases} \tilde{\mathbf{z}}^{l-1} = \mathbf{z}^{l-1} \odot \boldsymbol{\pi}^{l-1} \\ s_j^l = \mathbf{u}_j^l \tilde{\mathbf{z}}^{l-1} \end{cases}$ (5.35)

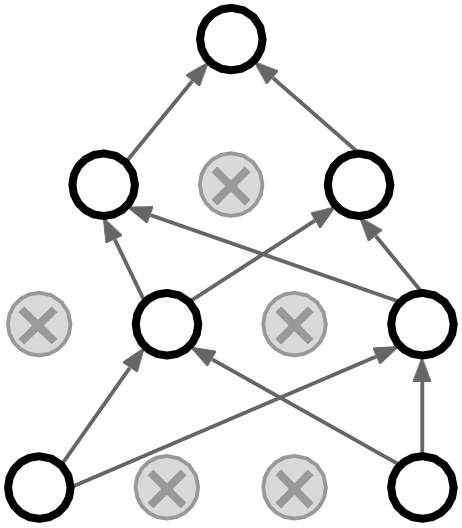
Dropout 처리!

- 불린 배열 $\boldsymbol{\pi}$에 노드 제거 여부를 표시
- $\boldsymbol{\pi}$는 샘플마다 독립적으로 정하는데, 난수로 설정함
- 보통 입력층 제거 비율 $P_{input} = 0.2$, 은닉층 제거 비율 $P_{hidden} = 0.5$로 설정

5.4.4 드롭아웃

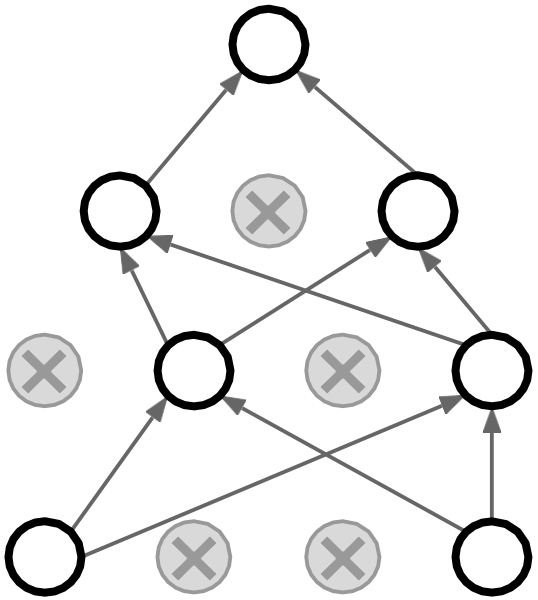
```python
p = 0.5 # probability of keeping a unit active. higher = less dropout

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)
```

Example forward pass with a 3-layer network using dropout



From cs231n Stanford Univ.

5.4.4 드롭아웃



Dropout is training a large ensemble of models (that share parameters).

Each binary mask is one model

From cs231n Stanford Univ.

5.4.4 드롭아웃

예측 단계

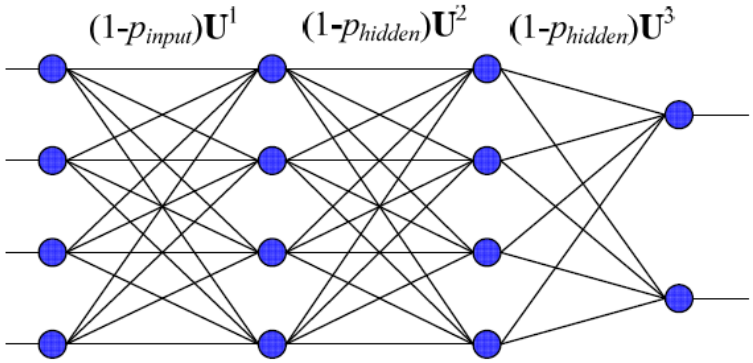


그림 5-28 드롭아웃의 예측 단계

- 앙상블 효과 모방
  - 가중치에 생존 비율 (1-드롭아웃 비율)을 곱하여 전방 계산
  - 학습 과정에서 가중치가 (1-드롭아웃 비율)만큼만 참여했기 때문

메모리와 계산 효율

- 추가 메모리는 불린 배열 **π**, 추가 계산은 작음
- 실제 부담은 신경망의 크기에서 옴: 보통 은닉 노드 수를 $\frac{1}{P_{hidden}}$만큼 늘림

5.4.4 드롭아웃

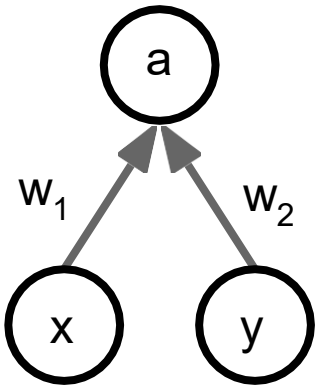
Output (label) Input (image)

Dropout makes our output random! $y = f_W(x, z)$ Random mask

Want to "average out" the randomness at test-time

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

But this integral seems hard …



At test time we have: $E[a] = w_1 x + w_2 y$

uring training we have:

$$E[a] = \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) = \frac{1}{2}(w_1 x + w_2 y)$$

확률

At test time, **multiply** by dropout probability

From cs231n Stanford Univ.

5.4.4 드롭아웃

```
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
```

At test time all neurons are active always
=> We must scale the activations so that for each neuron:
output at test time = expected output at training time

From cs231n Stanford Univ.

5.4.4 드롭아웃

```
""" Vanilla Dropout: Not recommended implementation (see notes below) """

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)

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 Summary

drop in forward pass

scale at test time

From cs231n Stanford Univ.

5.4.4 드롭아웃

More common: "Inverted dropout"

```python
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
```

test time is unchanged!

From cs231n Stanford Univ.

5.4.4 드롭아웃 -code

```
...

# nn layers
linear1 = torch.nn.Linear(784, 512, bias=True)
linear2 = torch.nn.Linear(512, 512, bias=True)
linear3 = torch.nn.Linear(512, 512, bias=True)
linear4 = torch.nn.Linear(512, 512, bias=True)
linear5 = torch.nn.Linear(512, 10, bias=True)
relu = torch.nn.ReLU()
dropout = torch.nn.Dropout(p=drop_prob)

# model
model = torch.nn.Sequential(linear1, relu, dropout,
                            linear2, relu, dropout,
                            linear3, relu, dropout,
                            linear4, relu, dropout,
                            linear5).to(device)

...
```

```
Epoch: 0001 cost = 0.309925616
Epoch: 0002 cost = 0.143516496
Epoch: 0003 cost = 0.113396436
Epoch: 0004 cost = 0.092770174
Epoch: 0005 cost = 0.081650071
Epoch: 0006 cost = 0.073365353
Epoch: 0007 cost = 0.070349611
Epoch: 0008 cost = 0.061270669
Epoch: 0009 cost = 0.060892191
Epoch: 0010 cost = 0.054064836
Epoch: 0011 cost = 0.051594462
Epoch: 0012 cost = 0.048855171
Epoch: 0013 cost = 0.043751985
Epoch: 0014 cost = 0.044706535
Epoch: 0015 cost = 0.044633854
Learning finished
Accuracy: 0.9771999716758728
```

5.4.4 드롭아웃 -code

```
...
total_batch = len(data_loader)
model.train()    # set the model to train mode (dropout=True)
for epoch in range(training_epochs):
...


...
# Test model and check accuracy
with torch.no_grad():
    model.eval()    # set the model to evaluation mode (dropout=False)
...
```

test

model.train() & model.eval()

regular: Dropconnect

- Sets the module in training/evaluation mode.
- This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. `Dropout`, `BatchNorm`, etc.

https://pytorch.org/docs/stable/nn.html?highlight=eval#torch.nn.Module.eval

5.4.5 앙상블 기법

- 앙상블
  - 서로 다른 여러 개의 모델을 결합하여 일반화 오류를 줄이는 기법
  - 현대 기계 학습은 앙상블도 규제로 여김
- 두 가지 일
  - 서로 다른 예측기를 학습하는 일
    - 예, 서로 다른 구조의 신경망 여러 개를 학습 또는 같은 구조를 사용하되 서로 다른 초깃값과 하이퍼 매개변수를 설정하고 학습
    - 예, 배깅(훈련집합을 여러 번 샘플링하여 서로 다른 훈련집합을 구성)
    - 예, 부스팅(i번째 예측기가 틀린 샘플을 $i+1$번째 예측기가 잘 인식하도록 연계성을 고려)
  - 학습된 예측기를 결합하는 일
    - 주로 투표 방식을 사용
- 자세한 내용은 12장