7강: 신경망 파트2 - 데이터 및 모델 설정

데이터 전처리1: 평균 제거

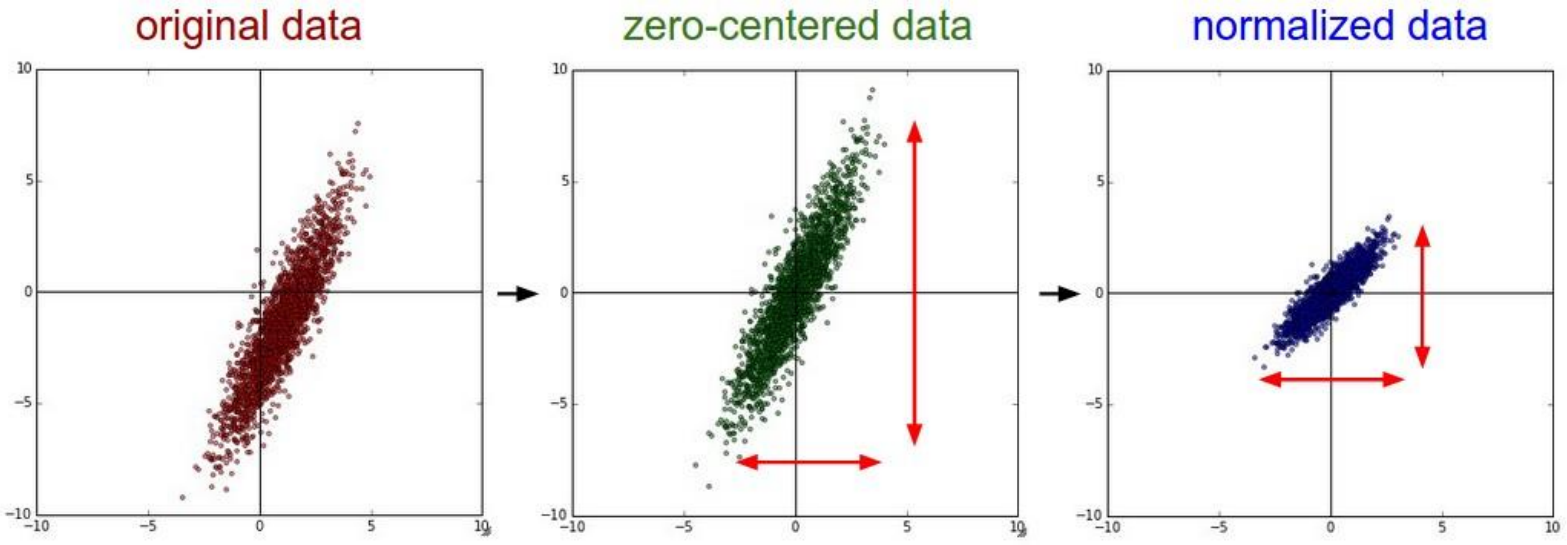
- 데이터 행렬 X
  - (N,D) 배열
- 가장 일반적인 전처리 형태는 평균 제거
- 데이터의 각 픽셀에서 평균을 뺌
  - X -= np.mean(X, axis=0)
- 이미지의 경우, 모든 픽셀에 대한 평균(단일 값)을 빼줄 수 있음
  - X -= np.mean(X)
- 또는, 세 가지 색상 채널 각각에 대한 평균을 따로 구하여 빼줄 수 있음

데이터 전처리2: 정규화

- 정규화는 데이터 차원을 정규화하여 대략적으로 동일한 스케일을 갖도록 함
- 일반적으로 사용되는 정규화 방법 중 하나는 각 차원을 표준편차로 나누는 것
  - 먼저 중심이 0이 되어있을 때 해야함
  - 각 픽셀별 표준편차가 1인 분포가 되도록 하는 것임
  - X /= np.std(X, axis=0) 으로 수행 가능

데이터 전처리2: 정규화

- 또 다른 정규화 방법은, 각 차원을 정규화하여 해당 차원에서 최소값은 -1, 최대값은 1이 되도록 하는 것
- 이 정규화는 입력 픽셀들이 서로 다른 스케일(또는 단위)를 가지지만 학습 알고리즘에 대해 대략 동일한 중요성을 가져야한다고 믿을 경우에 의미있음
- 이미지의 경우 상대적 스케일이 이미 대략 동일(0~255)하니 위 정규화가 꼭 필요하지 않음



데이터 전처리3: PCA & 화이트닝(whitening)

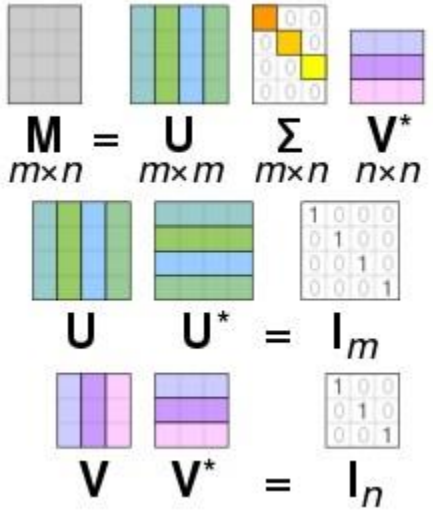
- PCA와 화이트닝은 또 다른 전처리 방식
- 데이터를 0이 중심이 되도록 한 후 공분산 행렬을 계산

$$Cov(X, Y) = E[(X - \mu_X)(Y - \mu_Y)]$$

```
# Assume input data matrix X of size [N x D]
X -= np.mean(X, axis = 0) # zero-center the data (important)
cov = np.dot(X.T, X) / X.shape[0] # get the data covariance matrix
```

- 공분산 행렬의 SVD 분해 계산 가능

```
U,S,V = np.linalg.svd(cov)
```



데이터 전처리3: PCA & 화이트닝(whitening)

```
U,S,V = np.linalg.svd(cov)
```

- U의 열은 고유벡터, S는 특이값의 1차원 배열
- 데이터를 비상관화하기 위해 원래의 데이터를 고유 벡터에 정사영

```
Xrot = np.dot(X, U) # decorrelate the data
```

- U의 열들은 orthonormal 벡터들이므로 기저 벡터(basis vector)로 간주
- 따라서, 이 투영은 고유벡터에 새로운 축으로 회전하는 것에 해당함
- Xrot의 공분산 행렬을 만약 계산한다면 대각행렬이 될 것임

데이터 전처리3: PCA & 화이트닝(whitening)

- np.linalg.svd의 좋은 성질 중 하나는 반환값 U에서 고육벡터 열이 고유값 크기에 의해 순서대로 정렬된다는 것
- 이를 활용하여 분산이 큰 상위 몇 개의 고유벡터만을 사용함으로써 데이터의 차원을 줄일 수 있음
- 이는 주성분 분석 또는 principal component analysis (PCA) 차원 축소라고 불림

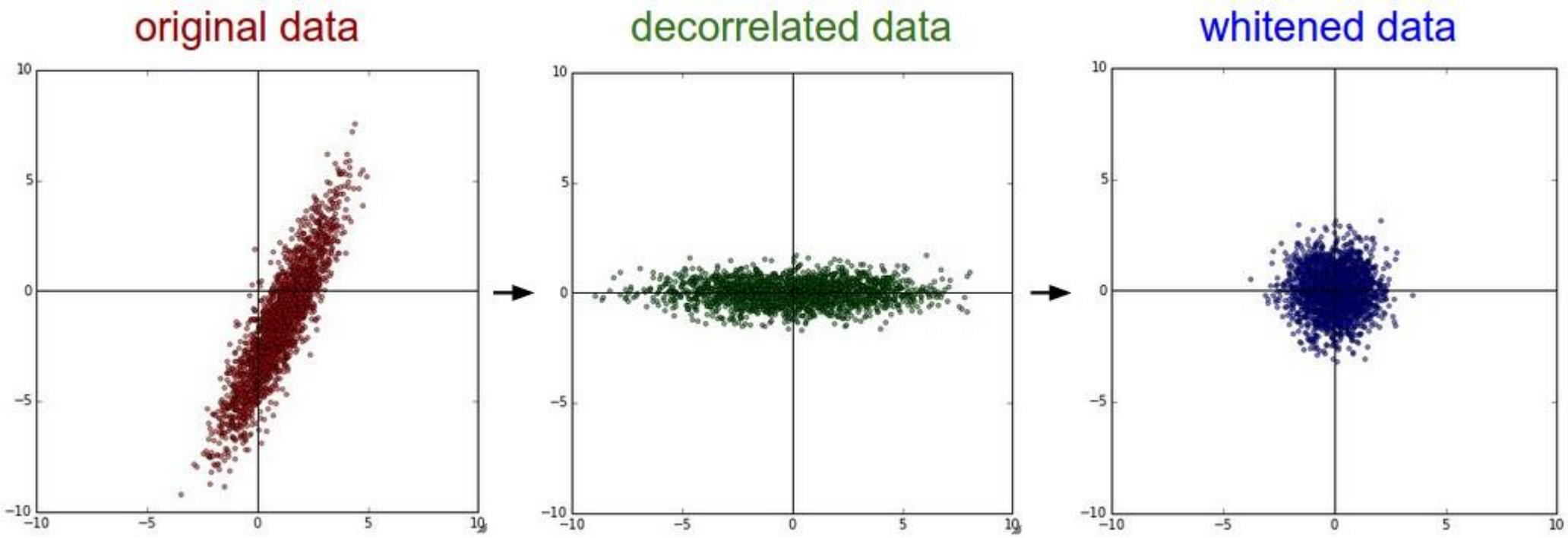
```
Xrot_reduced = np.dot(X, U[:,:100]) # Xrot_reduced becomes [N x 100]
```

- NxD 크기의 큰 데이터셋이 가장 큰 분산을 가진 100개 차원의 데이터셋 Nx100으로 줄어듦
- PCA로 축소된 데이터셋에 대해 신경망 훈련 및 분류를 실행하면, 공간과 시간을 모두 절약하면서도 매우 좋은 성능을 얻는 경우가 많음

데이터 전처리3: PCA & 화이트닝(whitening)

- 화이트닝 연산은 데이터를 고유 기저로 가져가고, 모든 차원을 고유값으로 나누어 스케일링을 정규화함

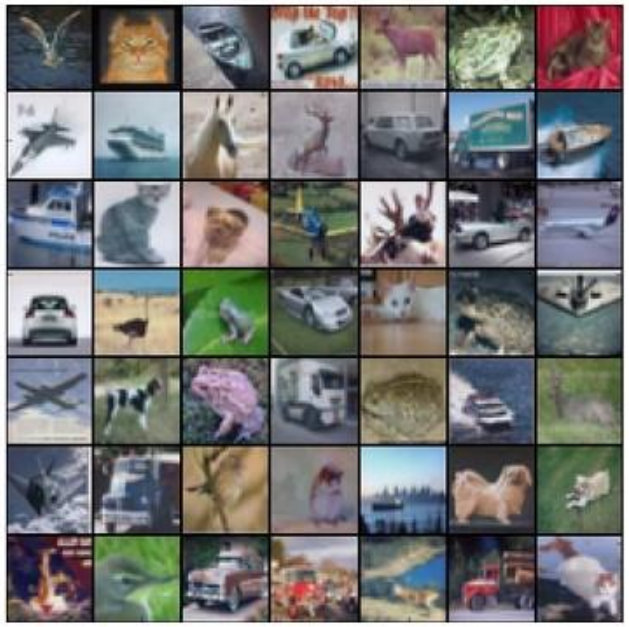
```
# whiten the data:
# divide by the eigenvalues (which are square roots of the singular values)
Xwhite = Xrot / np.sqrt(S + 1e-5)
```



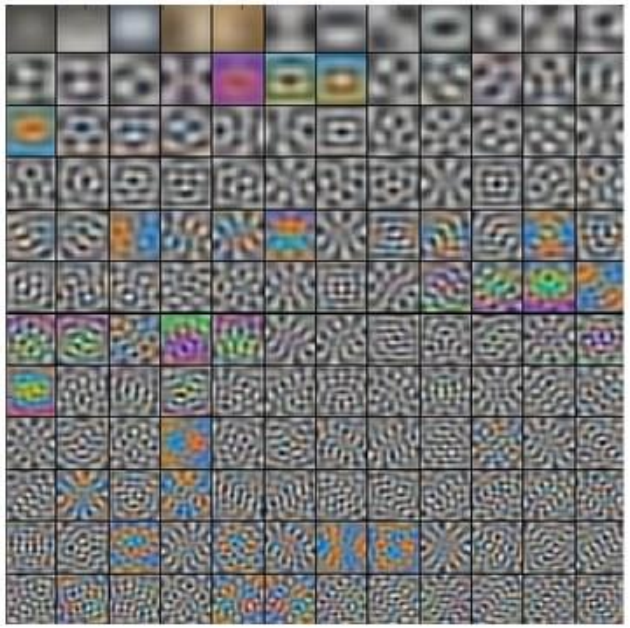
CIFAR-10의 PCA/화이트닝 적용 예

- CIFAR-10 훈련세트: 50000x3072, 공분산 행렬: 3072x3072
- 작은 분산을 설명하던 고주파수가 무시되고, 대부분의 분산을 설명하는 저주파수가 과장됨

original images

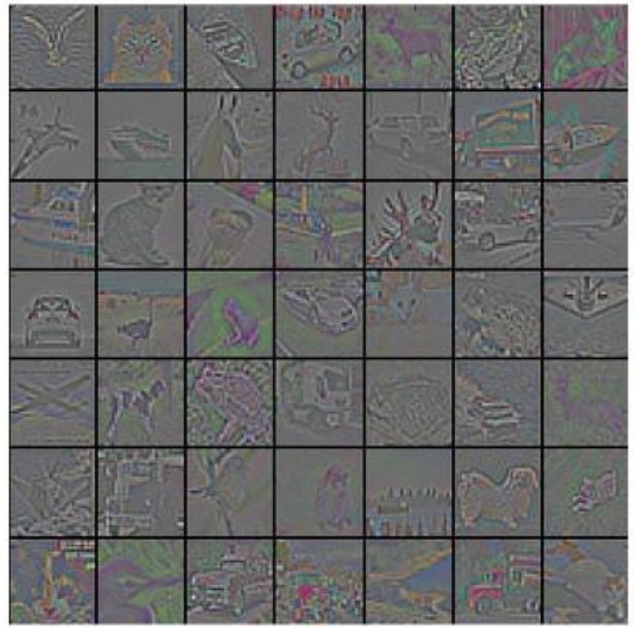


top 144 eigenvectors



reduced images

whitened images



전처리 관련 유의사항

- PCA/화이트닝 전처리 기법은 컨볼루션 신경망에서는 사용되지 않음
- 그러나 데이터를 0 중심으로 만드는 것은 매우 중요하며, 모든 픽셀 정규화도 흔히 볼 수 있음
- 모든 전처리 통계값(ex 평균)들은 훈련 데이터셋에 대해서만 계산되어야하며, 그 후 검증/테스트 세트에 적용되어야함
- 즉, 평균은 훈련세트에 대해서만 계산되고, 그 후 훈련세트, 검증세트, 테스트세트 각각에서 동일한 평균 값을 빼주어야함

가중치 벡터 초기화

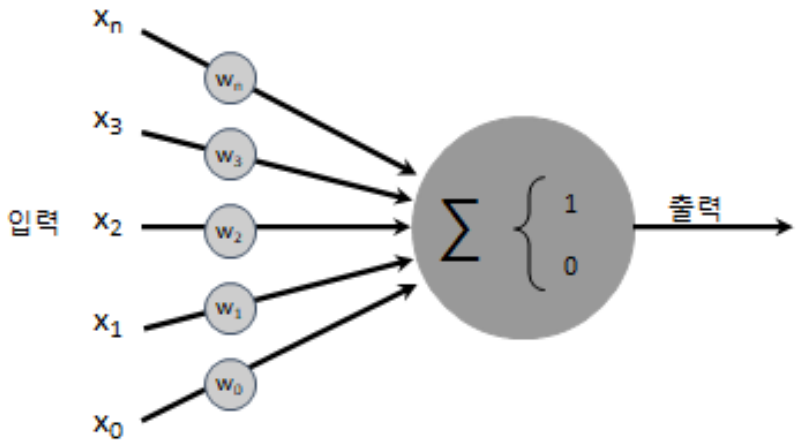
- 초기 가중치를 0으로 설정하는 것을 생각해볼 수 있음
- 그러나 이는 실수인데, 그 이유는 신경망 모든 뉴런이 같은 출력을 계산하면, 그들은 모두 역전파 동안 같은 그래디언트를 계산하고 완전히 동일한 파라미터 업데이트를 받게 되기 때문
- 즉, 가중치가 같게 초기화되면 뉴런들 사이 비대칭성의 원천이 없어짐

작은 랜덤 값으로 설정

- 가중치를 작은 수로 초기화하는 것이 일반적 (대칭성 파괴)
- 모든 뉴런이 처음에는 랜덤이고 고유하므로, 각자 서로 다르게 파라미터 업데이트됨
- W = 0.01*np.random.randn(D,H)
- 균일 분포(uniform distribution)에서 추출한 작은 수를 사용하는 것도 가능하지만, 이것이 실제 최종 성능에 큰 영향을 미치지는 않음

입력 수에 따라 $1/\sqrt{n}$ 으로 분산 조정

- 랜덤으로 초기화된 뉴런에서 출력의 분산이 입력수에 비례하여 증가한다는 문제가 발생
- 뉴런의 출력 분산이 1이 되도록 가중치 벡터를 입력 수의 제곱근으로 스케일링함
  - w = np.random.randn(n) / sqrt(n)
- 최근 권장사항
  - w = np.random.randn(n) * sqrt(2.0/n)



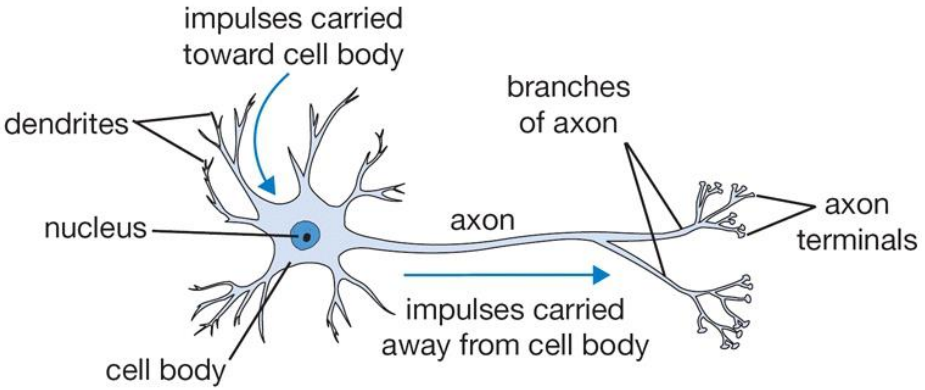
인공뉴런(Artificial Neuron)

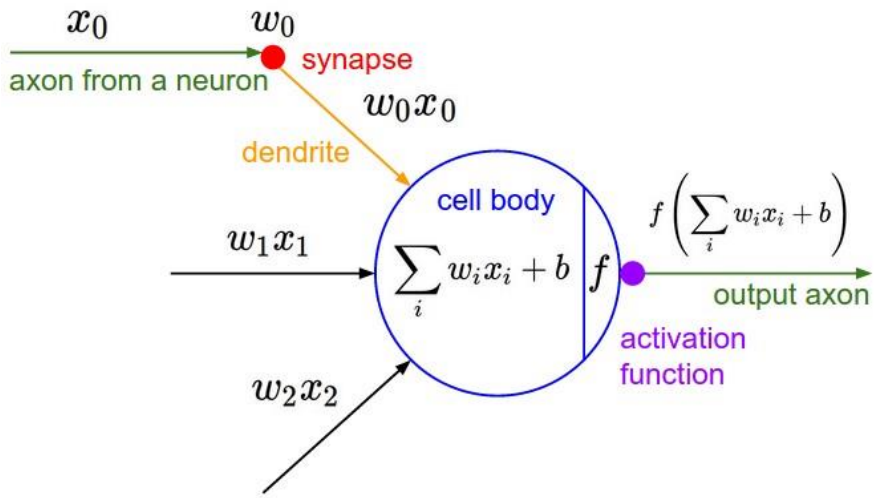
편향 초기화

- 편향(bias)은 0으로 초기화하는 것이 일반적인 방법임
- ReLU 유닛을 초기에 활성화하여 그래디언트 전파가 보장되도록 하기 위해 0.01과 같은 작은 상수 값으로 편향을 초기화하는 경우도 있지만 일반적이지 않음

배치 정규화(batch normalization)

- Ioffe, Szegedy에 의해 개발된 기법은 신경망 초기화 문제의 고민을 덜어줌
- 훈련 시작 시 신경망 전체의 활성화를 명시적으로 단위 가우시안 분포가 되도록 강제
- 일반적으로 완전연결계층 바로 뒤와 활성화 함수 앞에 배치 정규화를 삽입
- 신경망에서 배치정규화 사용은 매우 일반적인 관행이 됨
- 배치정규화는 나쁜 초기화에 대해 견고함





L2 정규화(L2 regularization)

- 신경망의 용량을 제어하여 과적합을 방지하는 여러 가지 방법 중 대표적인 방법이 L2 정규화
- 목표 함수(손실 함수)에서 모든 파라미터의 제곱에 페널티를 가함
  - $R(W) = \sum_k \sum_l \frac{1}{2} W_{k,l}^2$

$$L = \underbrace{\frac{1}{N} \sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$$

- L2 정규화는 뾰족한 가중치 벡터에 강한 페널티를 주고, 분산된 가중치 벡터를 선호함
- 즉, 신경망이 모든 입력을 약간씩 사용하는 것을 장려

L1 정규화(L1 regularization)

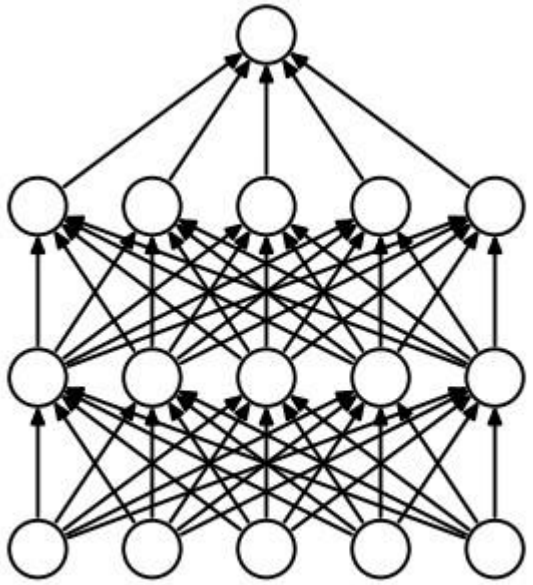
- L1 정규화는 각 가중치 w에 $\lambda|w|$ 항을 목표 함수에 추가함
- L1 정규화와 L2 정규화를 결합하는 것도 가능
  - $\lambda_1|w| + \lambda_2 w^2$ (Elastic net regularization)
- L1 정규화는 최적화 과정에서 가중치 벡터가 희소(sparse)해지도록 만드는 특징이 있음
- 즉, L1 정규화 사용시 뉴런은 가장 중요한 입력의 희소한 부분집합만 사용
- 반면, L2 정규화에서의 최종 가중치 벡터는 일반적으로 퍼져있고 작은 수
- 명확한 특성 선택에 관심이 없다면, L2 정규화가 L1에 비해 일반적으로 우수한 성능을 제공

최대 노름 제약 (max norm constraints)

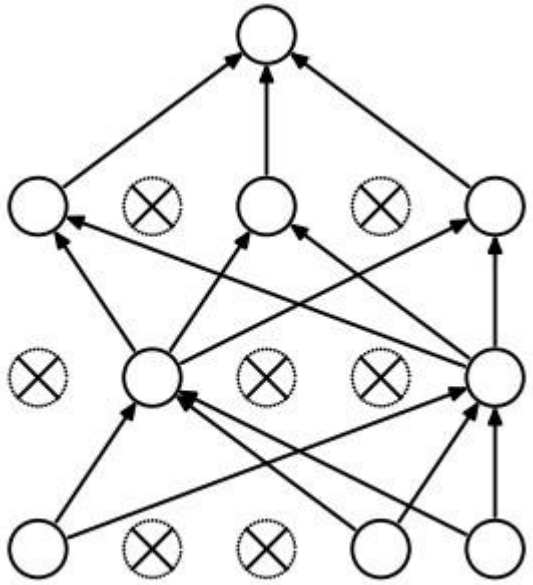
- 최대 노름 제약은 또 다른 형태의 정규화
- 각 뉴런에 대한 가중치 벡터 크기에 절대적인 상한선을 설정
- 모든 뉴런 가중치 벡터가 $\left|\left|\vec{w}\right|\right|_2 < c$를 만족하도록 제약 적용
- c의 전형적인 값은 3이나 4정도
- 학습률이 너무 높게 설정되어도 네트워크가 확 커지지 않는다는 것이 장점

드롭아웃(dropout)

- 최근 연구된 드롭아웃은 L1, L2, 최대 노름 정규화를 보완함
- 훈련 도중 드롭아웃은 뉴런을 확률 p로만 활성 상태로 유지하거나, 혹은 0으로 설정



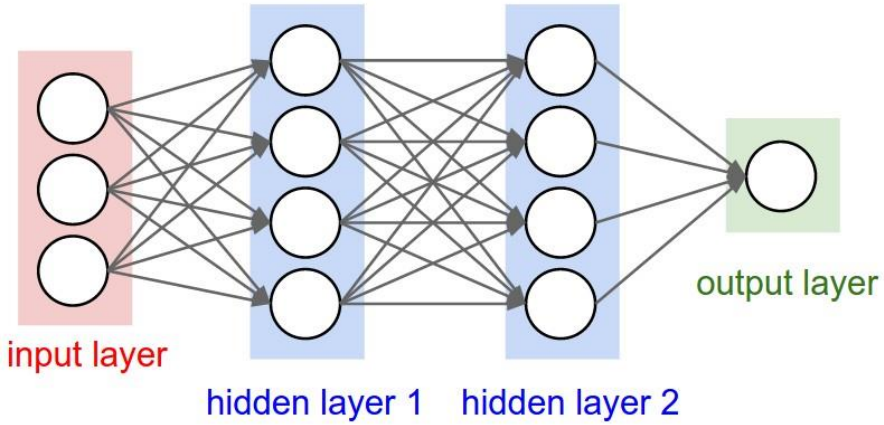
(a) Standard Neural Net



(b) After applying dropout.

드롭아웃(dropout)

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



드롭아웃(dropout)

- 예측 함수에서는 은닉 계층 출력에 p를 곱해야함
- 테스트 시점에서 뉴런이 모든 입력을 볼 수 있게 되는데, 테스트 시점의 뉴런 출력이 훈련 시점의 예상 출력과 동일하도록 하기 위함임
- 드롭아웃을 적용하지 않을 때에 비해 드롭아웃을 적용하면 뉴런의 출력이 x -> px로 줄어들기 때문에 테스트 시에 p를 곱함
- 테스트 시간에 p를 곱해야한다는 단점이 있음

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

드롭아웃(dropout)

```
"""
Inverted Dropout: Recommended implementation example.
We drop and scale at train time and don't do anything at test time.
"""

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 /
  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
```

편향 정규화 / 레이어별 정규화

- 편향 정규화
  - 편향을 정규화하는 것은 일반적이지 않음
  - 그렇지만 실제 애플리케이션에서 적절한 데이터 전처리와 함께 편향을 정규화하는 것은 거의 성능을 악화시키지 않는다고 함
  - 필요한 경우에는 편향 정규화를 사용할 수도 있음
- 레이어별 정규화
  - 다른 레이어를 다른 정도로 정규화하는 것은 매우 흔하지 않음

실전에서의 정규화

- 가장 흔한 방법은 교차검증된 L2 정규화 강도 사용
- 드롭아웃과 결합하는 것이 일반적
- p=0.5가 기본적으로 사용하는 확률 값이지만, 검증 과정에서 조정 가능