딥러닝
Deep Learning

이건명
충북대학교 소프트웨어학과

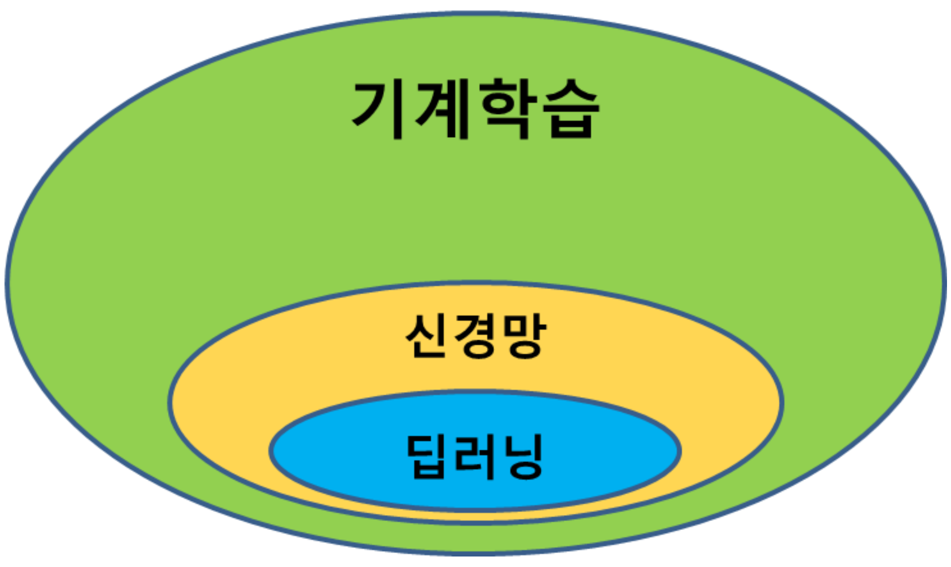
학습 내용

- 딥러닝 신경망의 특성에 대해서 알아본다.
- 기울기 소멸 문제와 해결 방안에 대해서 알아본다.
- 가중치 초기화 방법에 대해서 알아본다.
- 신경망의 과적합 완화 기법들에 대해서 알아본다.
- 신경망 학습에 사용되는 경사 하강법의 여러가지 변형 형태에 대해서 알아본다.

1. 딥러닝

❖ **딥러닝**(deep learning)

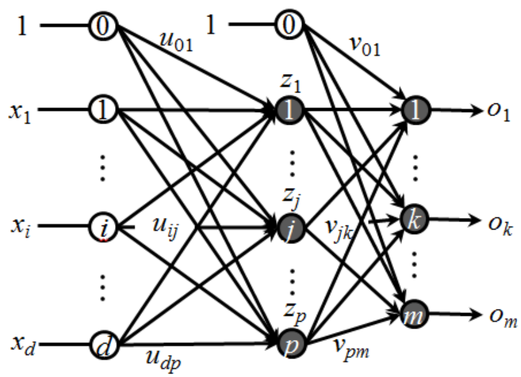
- 비교적 최근에 개발된 **딥러닝 신경망 모델 기반**의 **기계 학습 기법**을 차별화하여 일컫는 말
- **다수의 층**을 갖는 **신경망** 구조 사용
- 복잡한 구조의 신경망을 학습시키기 위해 **많은 데이터**와 **많은 컴퓨팅 자원** 사용
- 다양한 분야에서 기존 기계학습 방법보다 훨씬 **뛰어난 성능**을 보이면서 많은 관심



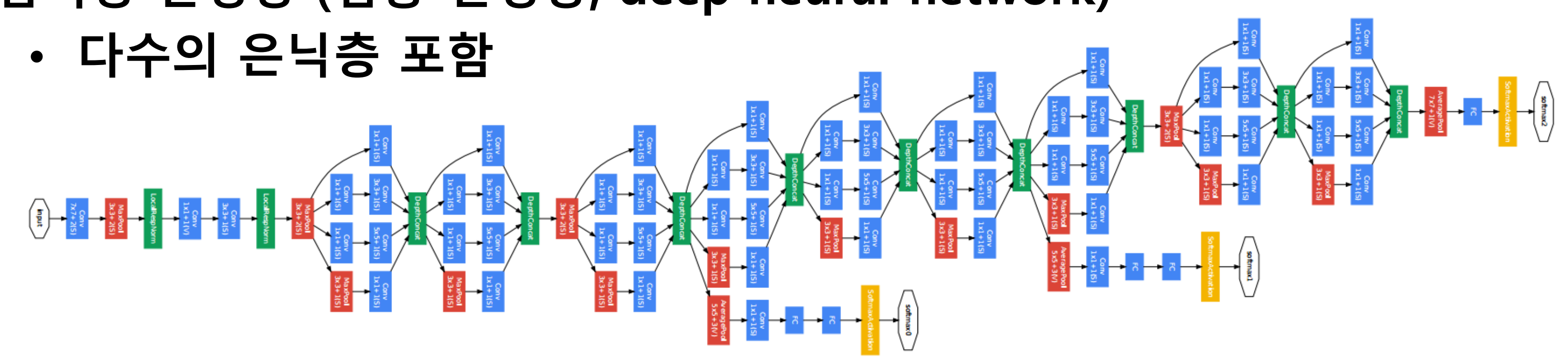
딥러닝

❖ **딥러닝**(deep learning)

- **일반 신경망**
  - **소수의 은닉층 포함**



- **딥러닝 신경망 (심층 신경망, deep neural network)**
  - **다수의 은닉층 포함**



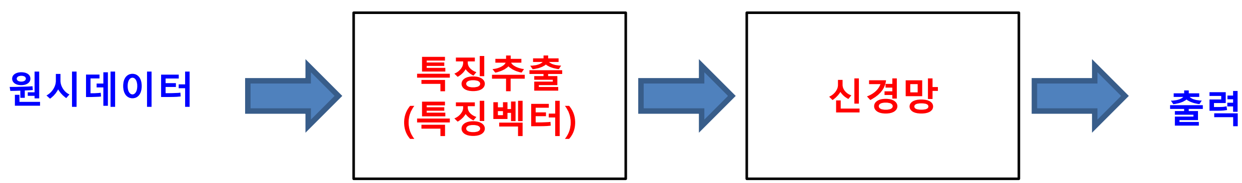
기계학습, 이건명

출전: googLeNet (google)

딥러닝

❖ 일반 신경망과 딥러닝 신경망

- **일반 신경망 모델**
  - 원시 데이터(original data)에서 직접 특징(**handcrafted feature**)을 추출해서 만든 **특징 벡터**(feature vector)를 신경망 모델 학습을 위한 입력으로 사용
  - 특징 벡터들의 품질에 영향

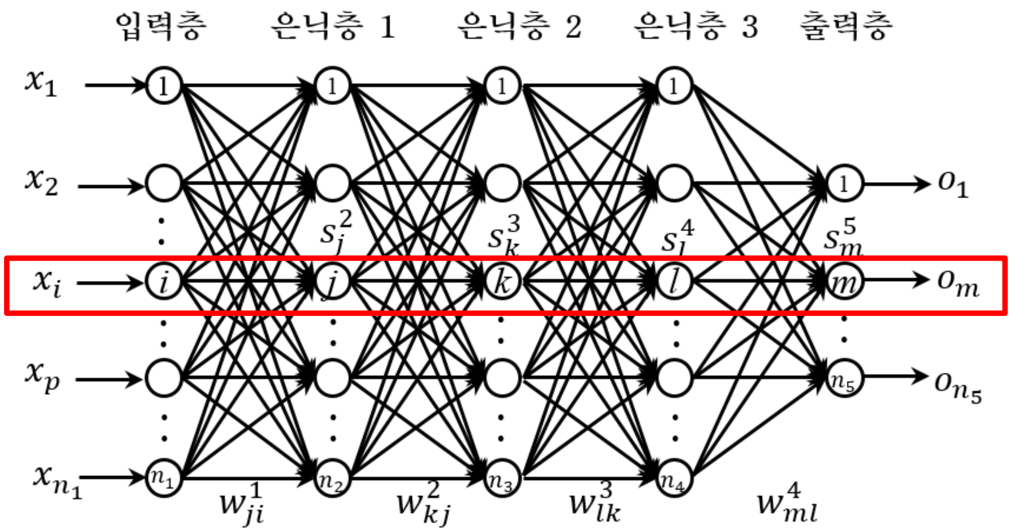


- **딥러닝 신경망**
  - **특징추출**과 **문제 해결을 위한 모델**의 **학습**을 함께 수행
  - 데이터로부터 문제해결에 **효과적인 특징**을 학습을 통해 추출
    ➔ 우수한 성능

2. 기울기 소멸 문제

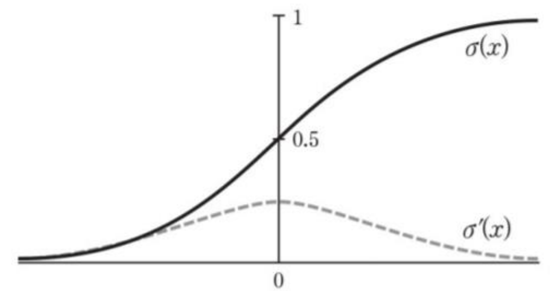
❖ 기울기 소멸 문제(Vanishing gradient problem)

- 은닉층이 많은 다층 퍼셉트론에서, 출력층에서 아래 층으로 갈 수록 전달되는 오차가 크게 줄어들어, 학습이 되지 않는 현상

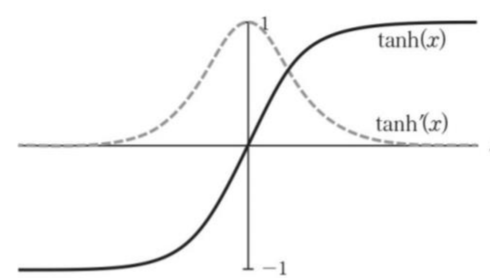


$$E = \frac{1}{2}\sum_{m=1}^{n_5}(y_m - o_m)^2$$

$$\frac{\partial E}{\partial w_{ji}^1} = \sum_{m=1}^{n_5}\sum_{l=1}^{n_4}\sum_{k=1}^{n_3}\sum_{j=1}^{n_2}(y_m - o_m)f'(s_m^5)\,w_{ml}^4 f'(s_l^4) w_{lk}^3 f'(s_k^3)\,w_{kj}^2 f'(s_j^2)\,x_i$$



시그모이드(sigmoid)

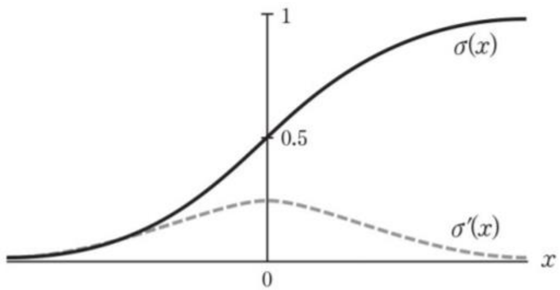


쌍곡 탄젠트(hypertangent)

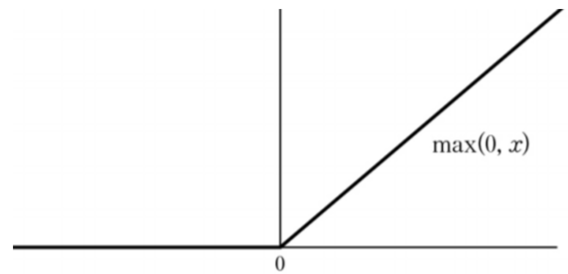
기울기 소멸 문제

❖ 기울기 소멸 문제 완화

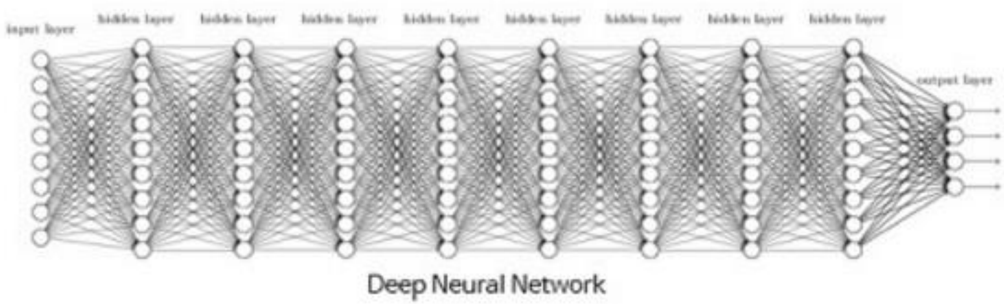
- 시그모이드나 쌍곡 탄젠트 대신 **ReLU(Rectified Linear Unit)** 함수 사용



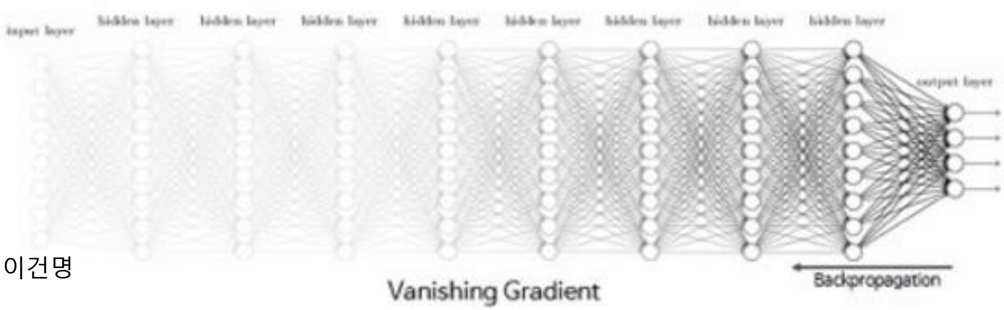
ReLU




```
def ReLU(x):
    return np.maximum(0,x)
```



ReLU 사용



시그모이드 사용

https://www.kaggle.com/getting-started/118228

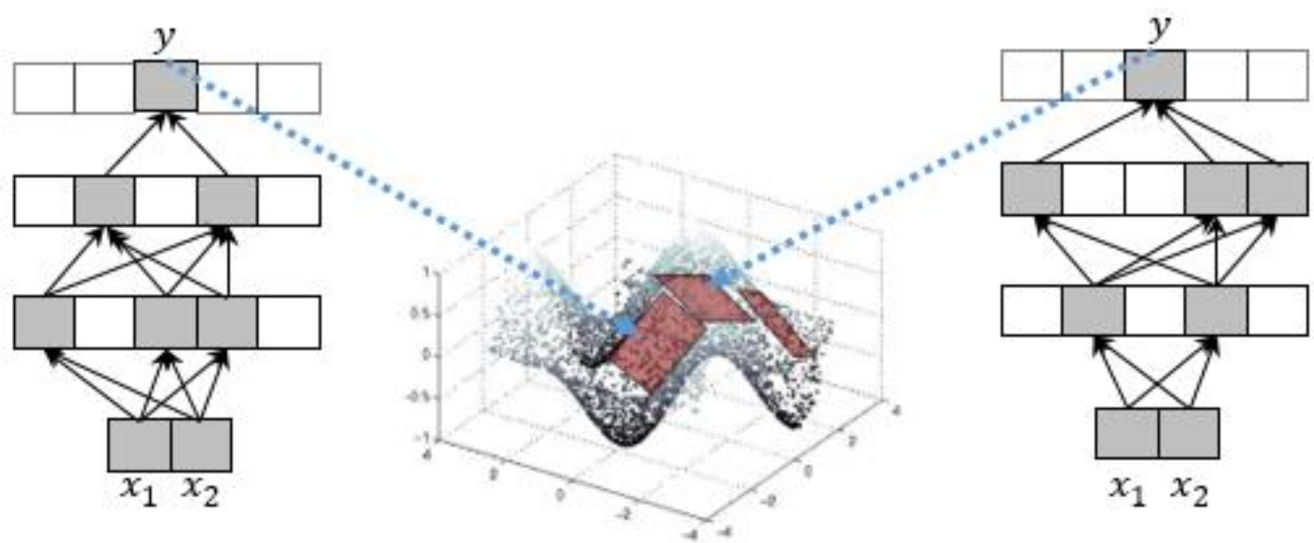
기울기 소멸 문제

❖ ReLU 함수 사용과 함수 근사

- 함수를 부분적인 평면 타일들로 근사(approximate)하는 형태
- 출력이 0이상인 것들에 의해 계산되는 결과
  - 입력의 선형변환(입력과 가중치 행렬의 곱으로 표현)의 결과

$$y = [\ ]_{1\times 2} \cdot [\ ]_{2\times 3} \cdot [\ ]_{3\times 2}\, \boldsymbol{x}$$

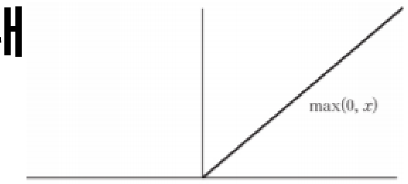
$$y = [\ ]_{1\times 3} \cdot [\ ]_{3\times 2} \cdot [\ ]_{2\times 2}\, \boldsymbol{x}$$



기울기 소멸 문제

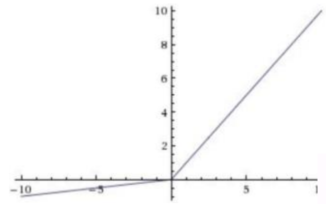
❖ ReLU와 변형된 형태

- **ReLU**



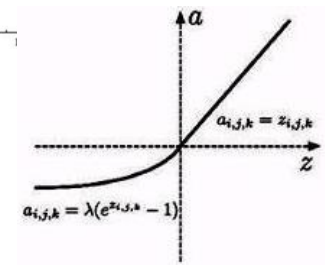
- **누수 ReLU(Leaky ReLU)**

$$f(x) = \max(\alpha x, x)$$



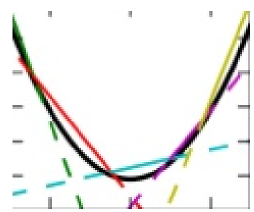
- **ELU (exponential Linear Unit)**

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{otherwise} \end{cases}$$



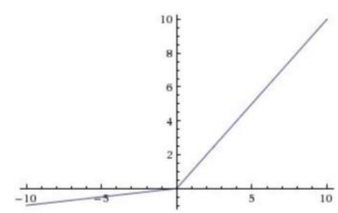
- **Maxout**

$$f(\boldsymbol{x}) = \max_{i \in \{1,\ldots,k\}} \{\boldsymbol{w}_i^\top \boldsymbol{x} + b_i\}$$



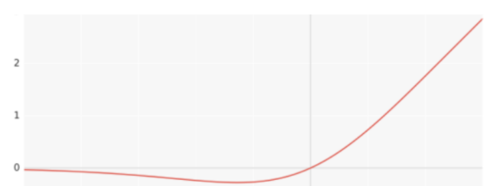
- **PReLU** (parameteric ReLU)

$$f(x) = \max(\alpha x, x)$$ α : 학습되는 파라미터

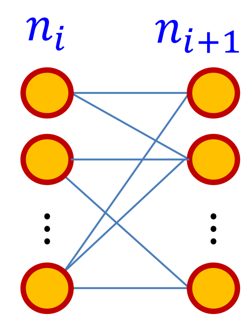


- **Swish**

$$f(x) = x \cdot sigmoid(x)$$



3. 가중치 초기화



❖ 가중치 최기화

- 신경망의 성능에 큰 영향을 주는 요소
- 보통 가중치의 초기값으로 0에 가까운 무작위 값 사용

❖ 개선된 가중치 초기화 방법

- 각 노드의 **입력 노드 개수** n_i와 **출력 노드의 개수** n_{i+1}를 사용하는 방법
  - **균등 분포 초기화**

    $$U\left[-\sqrt{\frac{6}{n_i+n_{i+1}}}, \sqrt{\frac{6}{n_i+n_{i+1}}}\right]$$ 에서 무작위로 선택

```
sd = np.sqrt(6/(layer_size[i+1]+ layer_size[i]))
W = np.random.uniform(-sd, sd, layer_size[i]*
      layer_size[i-1]).reshape(layer_size[i+1], layer_size[i])
```

  - **제이비어**(Xavier) **초기화**

    $$\frac{N(0,1)}{\sqrt{n_i}}$$ 에서 무작위로 선택 $N(0,1)$: 표준 정규분포

```
W = np.random.randn(layer_size[i+1], layer_size[i])
      * np.sqrt(1/layer_size[i])
```

  - **허**(He) **초기화**

    $$\frac{N(0,1)}{\sqrt{n_i/2}}$$ 에서 무작위로 선택

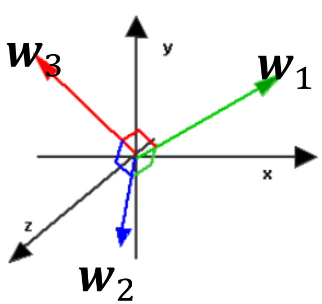
```
W = np.random.randn(layer_size[i+1], layer_size[i])
      * np.sqrt(2/layer_size[i])
```

가중치 초기화

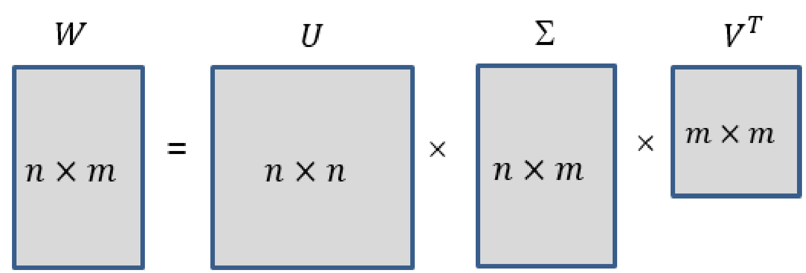
❖ 개선된 가중치 초기화 방법 – cont.

- **제한적 볼츠만 머신**(Restricted Boltzmann Machine, RBM) **이용 방법**
  - 입력 데이터에 대해 제한적 볼츠만 머신(5.3.1절에서 소개)을 학습시킨 결과의 가중치를 초기값으로 사용

- **인접 층간의 가중치**를 **직교하는 벡터**로 초기화
  - 가중치 행렬 W을 표준 정규분포 $N(0,1)$에서 무작위 추출하여 채움
  - 가중치 행렬 W을 특이값 분해(SVD, 부록 B.2.2 참고)하여, **직교하는 벡터를 초기 가중치**로 사용
    - **특이값 분해**
      - » 행렬 W를 $W = U\Sigma V^T$로 분해하는 행렬곱으로 표현 방법
      - » 여기에서 U, V는 각 열의 서로 직교하는 직교행렬



직교

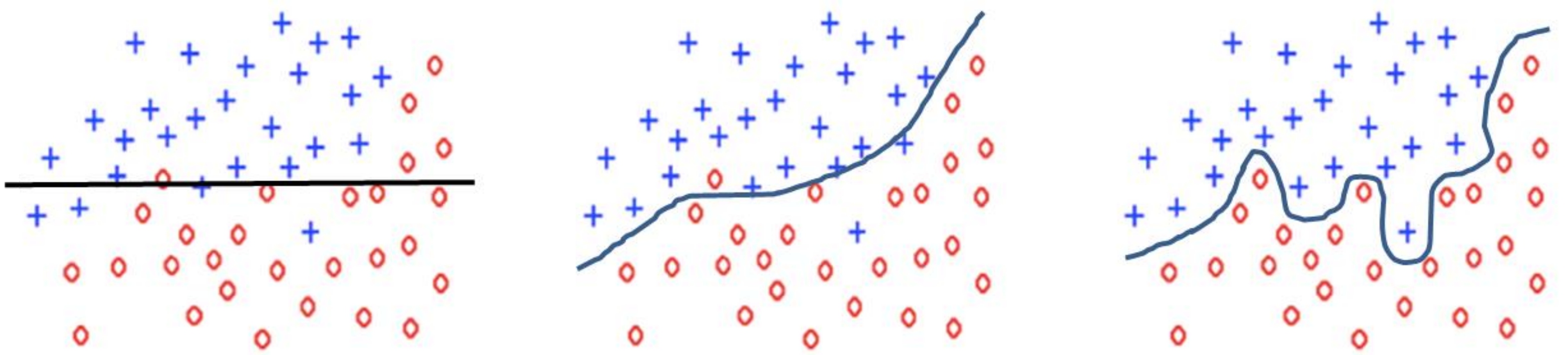


특이값 분해

4. 과적합 문제

❖ 과적합(Overfitting)

- 모델이 학습 데이터에 지나치게 맞추어진 상태
- 데이터에는 잡음이나 오류가 포함
  - 과적합된 모델은 학습되지 않는 데이터에 대해 성능 저하



부적합(underfitting) 정적합(good fitting) 과적합(overfitting)

- **과적합 문제 완화 기법**
  - 규제화
  - 드롭아웃
  - 배치 정규화

과적합 문제

1. 규제화(Regularization) 기법

- 오차 함수를 오차(error) 항과 모델 복잡도(model complexity) 항으로 정의
  - 모델이 복잡해 지면 과적합이 될 수 있으므로, 모델 복잡도를 벌점(penalty) 항으로 추가

오차 함수= (오차 항) +α(모델 복잡도 항)

α : 상대적인 반영비율을 조정

- **신경망 학습의 모델 복잡도**
  - 절대값이 큰 가중치에 **벌점**(penalty) 부여

$$\boldsymbol{w} = (w_1, w_2, \cdots, w_n)$$

  - 모델 복잡도 정의

$\sum_{i=1}^{n} w_i^2$ 또는 $\sum_{i=1}^{n} |w_i|$

cf. ridge/lasso regression

과적합 문제

2. 드롭아웃(Dropout) 기법

- 학습할 때 일정 확률로 노드들을 무작위로 선택하여, **선택된 노드의 앞뒤로 연결된 가중치 연결선은 없는 것**으로 간주
- **미니배치**(mini-batch)나 **학습주기**(epoch) 마다 드롭아웃 할 즉, 없는 것으로 간주할 노드들을 새롭게 선택하여 학습
- 추론을 할 때는 드롭아웃을 하지 않고 **전체 학습된 신경망을 사용**하여 출력 계산

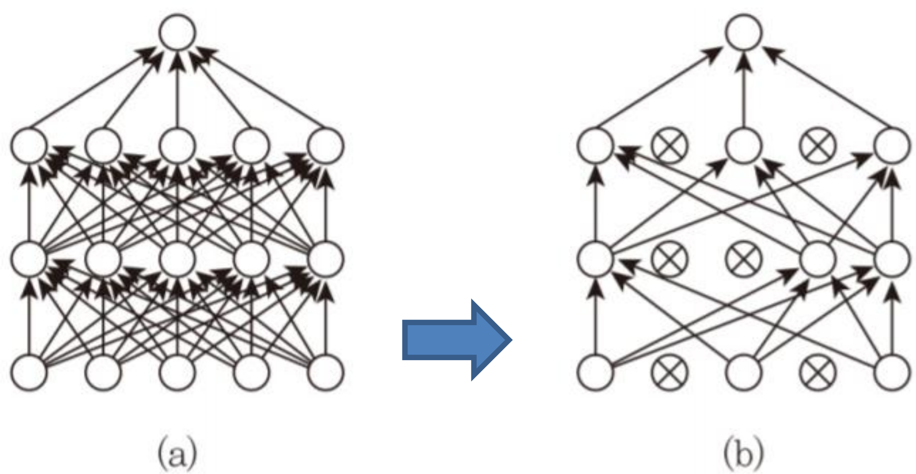


그림 5.6 드롭아웃 방법

(b)는 (a)의 신경망에 드롭아웃을 적용한 결과를 나타냄

과적합 문제

❖ 학습주기(epoch, 에포크)

- **전체 데이터**에 대해서 신경망 모델을 **한번의 학습 과정**을 완료하는 것

❖ 배치(batch)

- 신경망의 **가중치**를 **한번 수정**할 때 **사용**되는 **데이터**
- **배치 크기(batch size)**
  - **하나의 배치**에 **포함**되는 **데이터의 개수**

❖ iteration(반복)

- **한 번의 학습주기**를 **완료**하기 위해 수행되는 **배치**의 **처리 회수**
  - **iteration 개수 = (전체 데이터 개수)/(배치 크기)**

과적합 문제

- ❖ **학습 데이터 단위에 따른 가중치 갱신 전략**
  - **확률적 갱신** (stochastic update, stochastic gradient descent)
    - 한번에 **하나의 학습 데이터**에 대한 **그레디언트**를 계산하여 가중치 갱신
    - 가중치의 변동이 심하여 성능 개선 저하
  - **배치 갱신** (batch update, batch gradient descent)
    - **전체 학습 데이터**에 대한 **그레디언트의 평균**을 구하여 가중치 갱신
    - 느린 학습
  - **미니배치 갱신** (mini-batch update, mini-batch gradient descent)
    - **일정 개수의 학습 데이터**, **미니배치**에 대해 그레디언트의 평균을 구하여 가중치 갱신
      - 예. 10개 데이터로 구성된 미니배치의 그레디언드

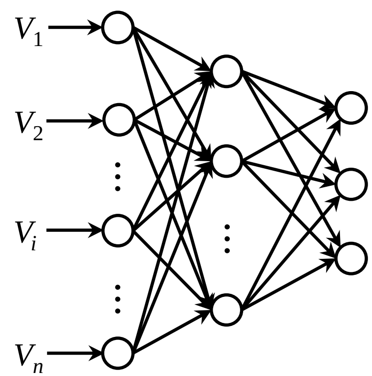
$$\nabla g = \frac{1}{10}\sum_{i=1}^{10}\nabla g_i$$

    - 과적합 문제 완화에 도움

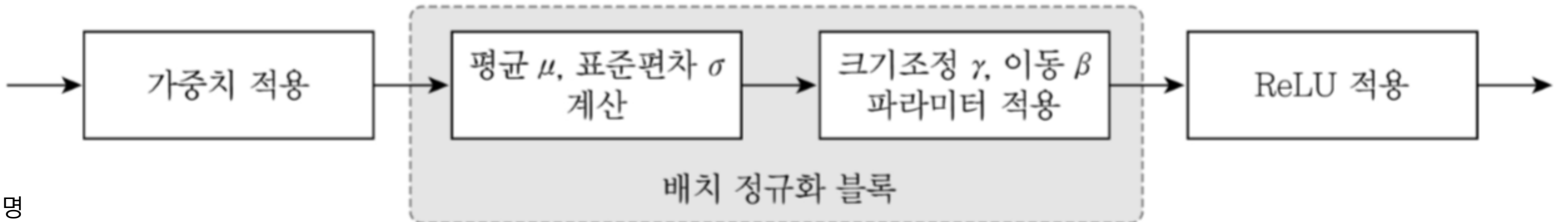
과적합 문제

3. 배치 정규화(Batch normalization) 기법

- **내부 공변량 이동(internal covariate shift) 문제**
  - 오차 역전파 알고리즘 적용 학습
  - 이전 층들의 학습에 의해 이들 층의 가중치가 바뀌게 되면, 현재 층에 전달되는 데이터의 분포가 현재 층이 학습했던 시점의 분포와 차이 발생 ➔ 학습 속도 저하



- **배치 정규화**
  - 신경망의 **각 층**에서 **미니배치** B의 각 데이터에 가중치 연산을 적용한 결과인 x_i의 **분포**를 **정규화**하는 것



과적합 문제

❖ 배치 정규화 기법 – cont.

- **미니배치** B에 대해
  1. $\boldsymbol{x}_i$의 평균 $\boldsymbol{\mu}_B$가 **0**이 되고 표준편차 $\boldsymbol{\sigma}_B$는 **I**가 되도록 변환
  2. 크기조정(scaling) 파라미터 γ와 이동(shift) 파라미터 β 적용
  3. 변환된 데이터 $\boldsymbol{y}_i$ 생성

- 가중치 연산 결과의 미니 배치 : $B = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_m\}$
- 배치 정규화 적용 결과 : $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_m\}$

미니배치의 평균 : $\boldsymbol{\mu}_B = \frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}_i$

미니배치의 분산 : $\boldsymbol{\sigma}_B^2 = \frac{1}{m}\sum_{i-1}^{m}(\boldsymbol{x}_i - \boldsymbol{\mu}_B)^2$

정규화 : $\hat{\boldsymbol{x}}_i = \frac{\boldsymbol{x}_i - \boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^2 + \epsilon}}$

크기조정 및 이동변환 : $\boldsymbol{y}_i = \gamma\hat{\boldsymbol{x}}_i + \beta$

γ, β : 학습 대상 파라미터

5. 가중치 학습 기법(optimizer)

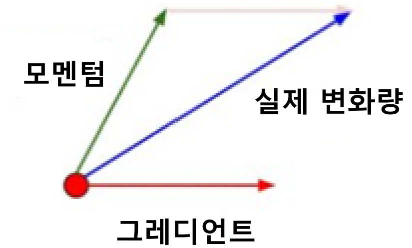
❖ 경사 하강법(Gradient descent method)

$$w^{(t+1)} = w^{(t)} - \eta \frac{\partial E(w^{(t)})}{\partial w}$$

❖ 모멘텀 사용 경사 하강법(Gradient descent method with Momentum

$$\Delta^{(t)} = \alpha \Delta^{(t-1)} + \eta \frac{\partial E(w^{(t)})}{\partial w}$$

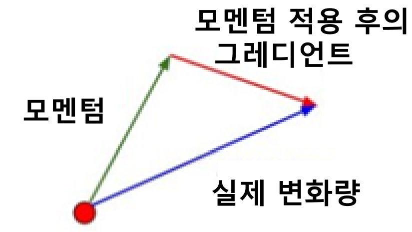
$$w^{(t+1)} = w^{(t)} - \Delta^{(t)}$$



❖ NAG(Nesterov accelerated gradient)

$$\Delta^{(t)} = \alpha \Delta^{(t-1)} + \eta \frac{\partial E(w^{(t)} - \alpha \Delta^{(t-1)})}{\partial w}$$

$$w^{(t+1)} = w^{(t)} - \Delta^{(t)}$$



가중치 학습 기법

❖ **Adagrad** (Adaptive Gradient Algorithm)

- 가중치 별도 다른 학습율 사용

$$g_i^{(t)} = \frac{\partial E(\boldsymbol{w}^{(t)})}{\partial w_i} \qquad G_i^{(t)} = G_i^{(t-1)} + \left(g_i^{(t)}\right)^2$$

$$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta}{\sqrt{G_i^{(t)} + \epsilon}} g_i^{(t)}$$

❖ **Adadelta**

- Adagrad의 확장
- 과거 그레디언트의 영향을 점점 줄이면서 그레디언트 제곱합 계산

$$E[g_i^2]_t = \gamma E[g_i^2]_{t-1} + (1-\gamma)\left(g_i^{(t)}\right)^2 \qquad RMS[g_i]^{(t)} = \sqrt{E[g_i^2] + \epsilon}$$

$$E[w_i^2]_t = \gamma E[w_i^2]_{t-1} + (1-\gamma)\left(\frac{\eta}{RMS[g_i]^{(t)}} g_i^{(t)}\right)^2 \quad RMS[w_i]^{(t)} = \sqrt{E[w_i^2] + \epsilon}$$

$$w_i^{(t+1)} = w_i^{(t)} - \frac{RMS[w_i]^{(t-1)}}{RMS[g_i]^{(t)}} g_i^{(t)}$$

가중치 학습 기법

❖ **RMSprop**

- 가중치별 다른 학습율 사용
- 결합된 그레디언트 제곱의 합의 제곱근을 학습율로 사용

$$E[g_i^2]_t = \gamma E[g_i^2]_{t-1} + (1-\gamma)\left(g_i^{(t)}\right)^2$$

$$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta}{\sqrt{E[g_i^2]^{(t)} + \epsilon}} g_i^{(t)}$$

❖ **ADAM (Adaptive Moment Estimation)**

- 가중치별 다른 학습율 사용
- 그레디언트의 1차, 2차 모멘텀 사용

$$m^{(t)} = \beta_1 m^{(t-1)} + (1-\beta_1) g_i^{(t)}$$

$$v^{(t)} = \beta_2 v^{(t-1)} + (1-\beta_2)\left(g_i^{(t)}\right)^2$$

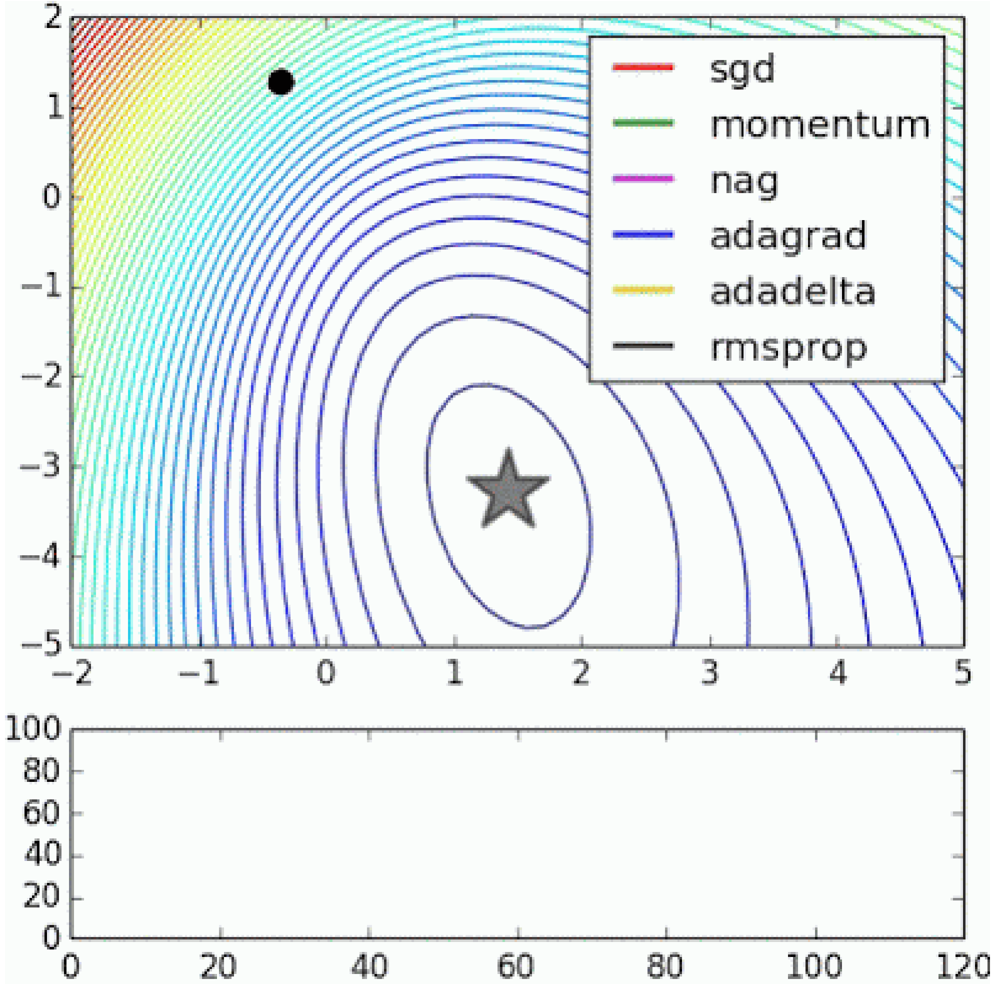
$$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta}{\sqrt{\hat{v}^{(t)} + \epsilon}} \hat{m}^{(t)}$$

$$\hat{m}^{(t)} = \frac{m^{(t)}}{1-\beta_1^{(t)}}$$

$$\hat{v}^{(t)} = \frac{v^{(t)}}{1-\beta_2^{(t)}}$$

가중치 학습 기법

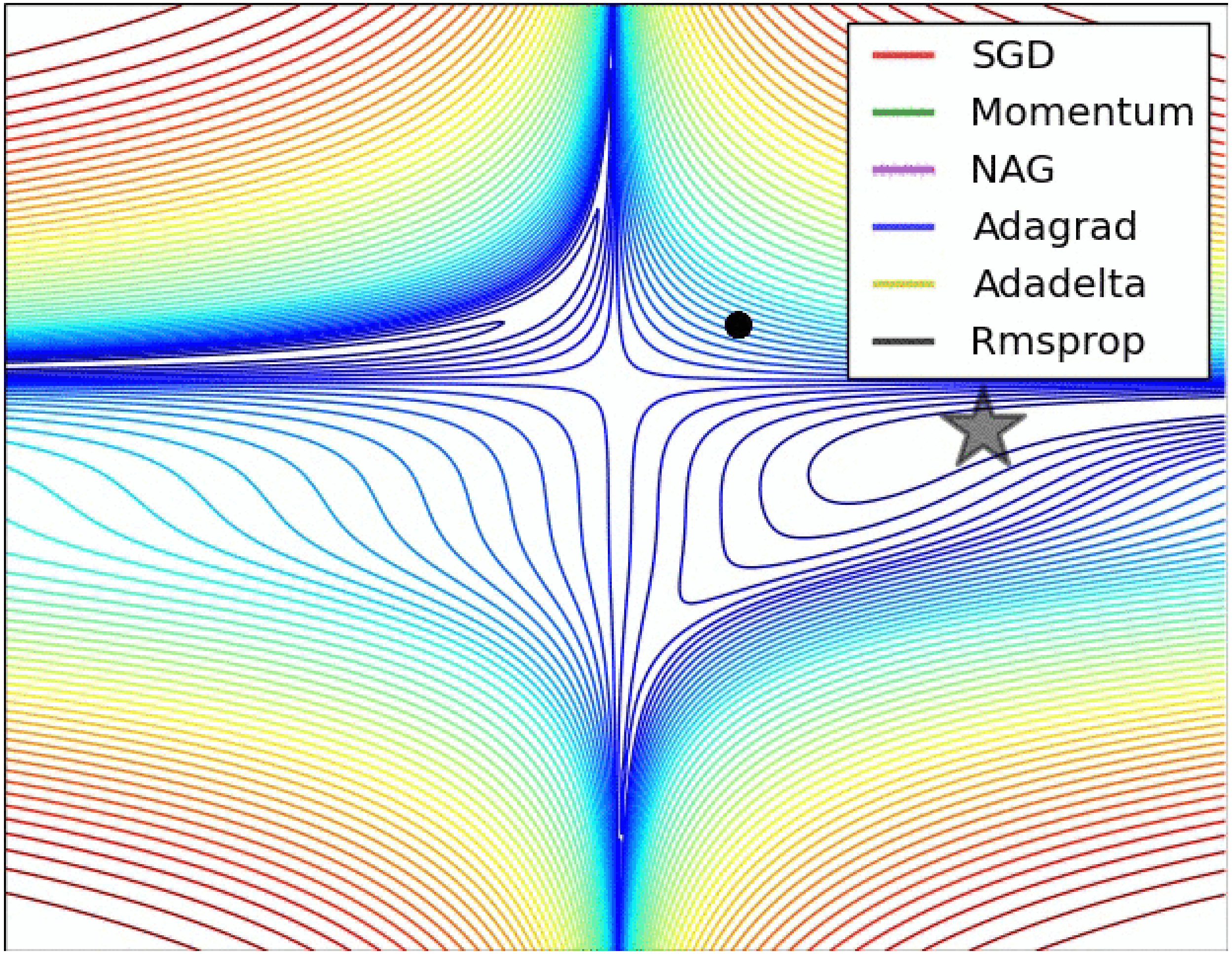
❖ **여러 경사 하강법의 동작 형태**



http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

가중치 학습 기법

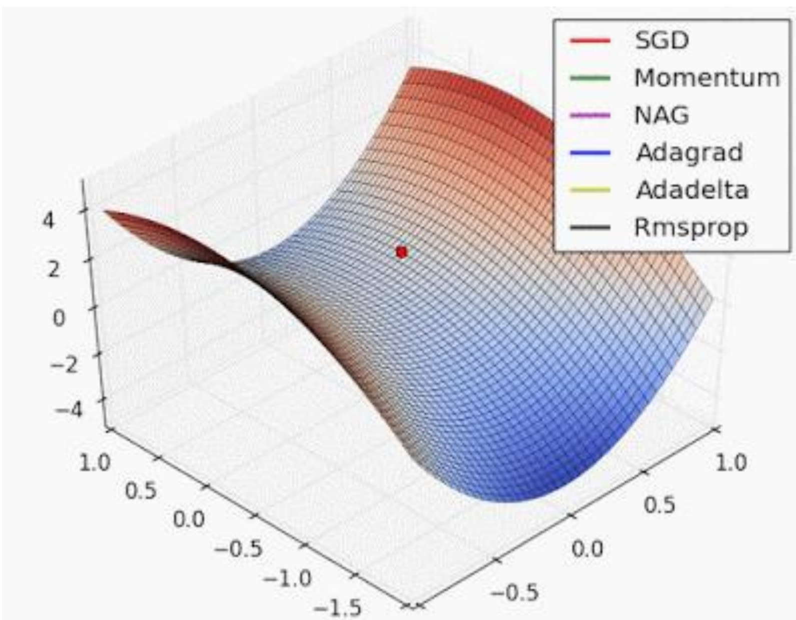
❖ 여러 경사 하강법의 동작 형태

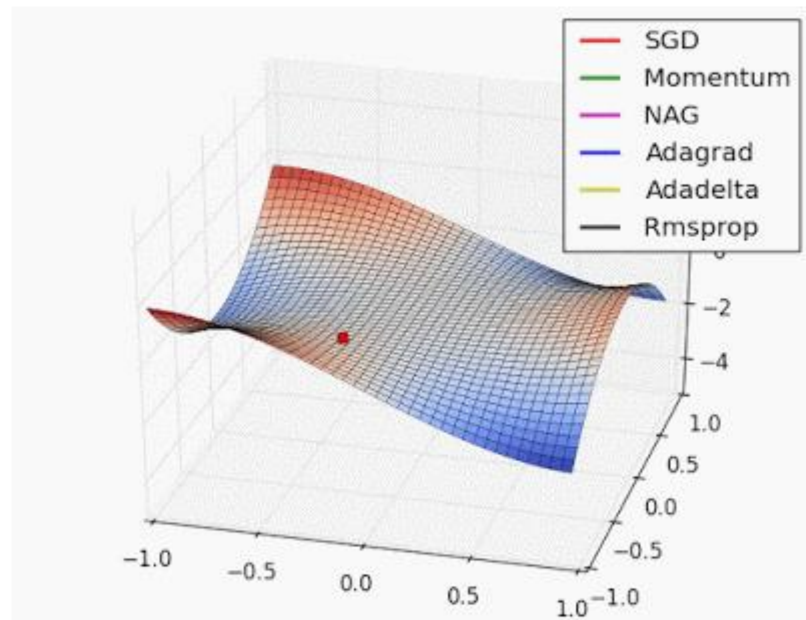


http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

가중치 학습 기법

❖ 여러 경사 하강법의 동작 형태





http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

Quiz

1. 전통적인 신경망에서 은닉층의 개수를 많이 늘리기 어려웠던 것은 활성화 함수 때문이다. (O,X)
2. 딥러닝 모델은 문제 해결에 적합한 특징 추출을 모델에서 학습으로 찾을 수 있는 능력이 있다. (O,X)
3. LeRU 함수를 사용하면 기울기 소멸 문제를 완전히 해결할 수 있다. (O,X)
4. 신경망 모델의 가중치를 초기화하는 방법에 따라 학습된 모델의 성능이 달라질 수 있다. (O,X)
5. 배치 정규화는 신경망의 각 노드의 출력값이 정규 분포를 갖도록 한다. (O,X)