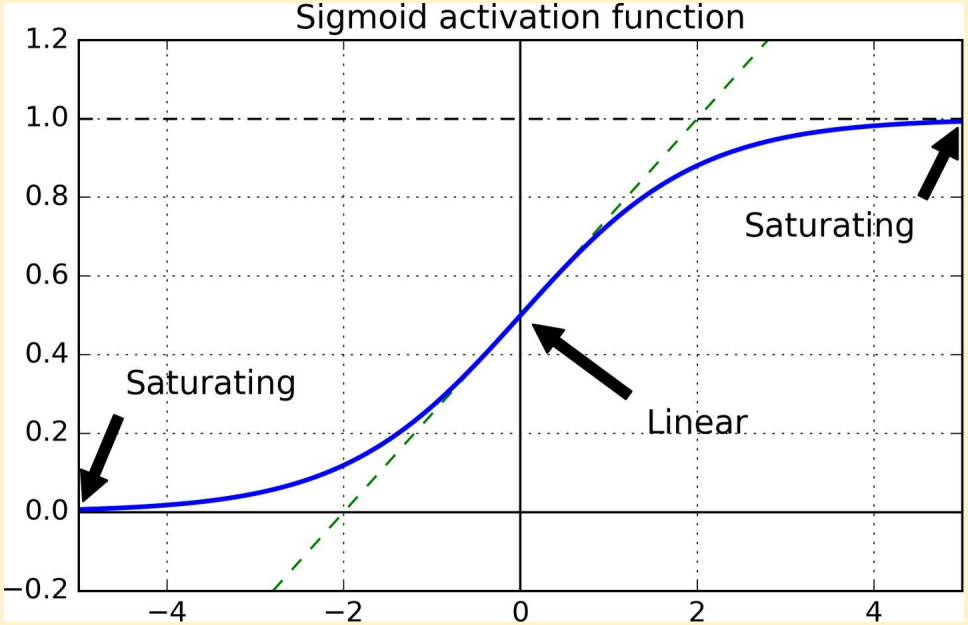
심층훈련망의 속도 개선

1. 가중치 설정의 초기화전략 개선
2. 좋은 활성화함수 사용
3. 배치 정규화 사용
4. 전이학습
5. 더 빠른 옵티마이저 사용

1. Vanishing/Exploding Gradients Problem

(1) 불안정한 그래디언트는 층마다 학습 속도를 다르게 만들어 DNN 훈련을 어렵게 만든다

(2) 그렇다면 왜 그래디언트 소실/폭발 문제가 발생하는가?



From "Understanding the Difficulty of Training Deep Feedforward Neural Networks"(Xavier Glorot, Yoshua Bengio, 2010)

시그모이드 함수와 정규분포의 가중치 초기화 방법을 사용했을 때 출력의 분산이 입력의 분산보다 크다.

따라서 레이어가 많아지면 가장 높은 층에서 활성화 함수가 0이나 1으로 수렴하게 됨.

로지스틱 함수의 도함수가 $\delta(1-\delta)$를 만족하기 때문에 역전파가 진행되면 아래층에 값이 도달 X

시그모이드 함수를 활성화 함수로 사용할 때 그래디언트 소실 문제가 발생

Solution (1) 새로운 함수/초기화방법 사용

(1) Glorot Initialization (or Xavier initialization)

Normal distribution with mean 0 and variance $\sigma^2 = \frac{1}{fan_{\text{avg}}}$

Or a uniform distribution between $-r$ and $+r$, with $r = \sqrt{\frac{3}{fan_{\text{avg}}}}$

$fan_{avg} = (fan_{in} + fan_{out})/2$ (fan_{in}은 입력층의 개수, fan_{out}은 출력층의 개수)

(2) Glorot Initialization 과 다른 초기화 전략 비교

Initialization	Activation functions	σ^2 (Normal)
Glorot	None, tanh, logistic, softmax	$1 / fan_{\text{avg}}$
He	ReLU and variants	$2 / fan_{\text{in}}$
LeCun	SELU	$1 / fan_{\text{in}}$

Solution (2) Batch Normalization

각 레이어를 통과하기 전에 모델을 정규화하고, 레이어의 출력값에 두 개의 파라미터를 이용하여 (곱연산을 통한 스케일 조정, +alpha를 통한 이동) 결괏값의 스케일을 조정함. (총 두 개의 파라미터를 사용함)

1. $\mu_B = \frac{1}{m_B}\sum_{i=1}^{m_B}\mathbf{x}^{(i)}$
2. $\sigma_B{}^2 = \frac{1}{m_B}\sum_{i=1}^{m_B}\left(\mathbf{x}^{(i)} - \mu_B\right)^2$
3. $\widehat{\mathbf{x}}^{(i)} = \frac{\mathbf{x}^{(i)} - \mu_B}{\sqrt{\sigma_B{}^2 + \varepsilon}}$
4. $\mathbf{z}^{(i)} = \gamma \otimes \widehat{\mathbf{x}}^{(i)} + \beta$

1. μB = 미니배치 B 벡터의 평균, m_B = 미니배치 B의 크기
2. σ_B = 미니배치 B 벡터의 표준편차
3. $\hat{x}^{(i)}$ = 정규화된 벡터의 개별 입력값
4. γ = 해당 층의 출력 스케일 벡터,

⊗ = 개별곱연산, β = output shift parameter, ε = 0이 되는 것을 막기 위한 smoothing term, $z^{(i)}$ = 해당 층의 최종 배치 정규화 출력

[장점]
- 수렴성을 가진 활성화함수(tanh, sigmoid)를 사용하면서 그래디언트 소실 문제를 해결
- 학습률을 높여도 괜찮기 때문에 학습과정의 속도가 높아짐

[단점]
- 모델의 복잡도가 높아지고 각 층마다 계산이 추가되어 전체적인 예측 시간을 증가시킴

=> 학습이 끝난 뒤 이전 층과 배치정규화를 합치는 것으로 실행속도 저하를 막을 수 있음

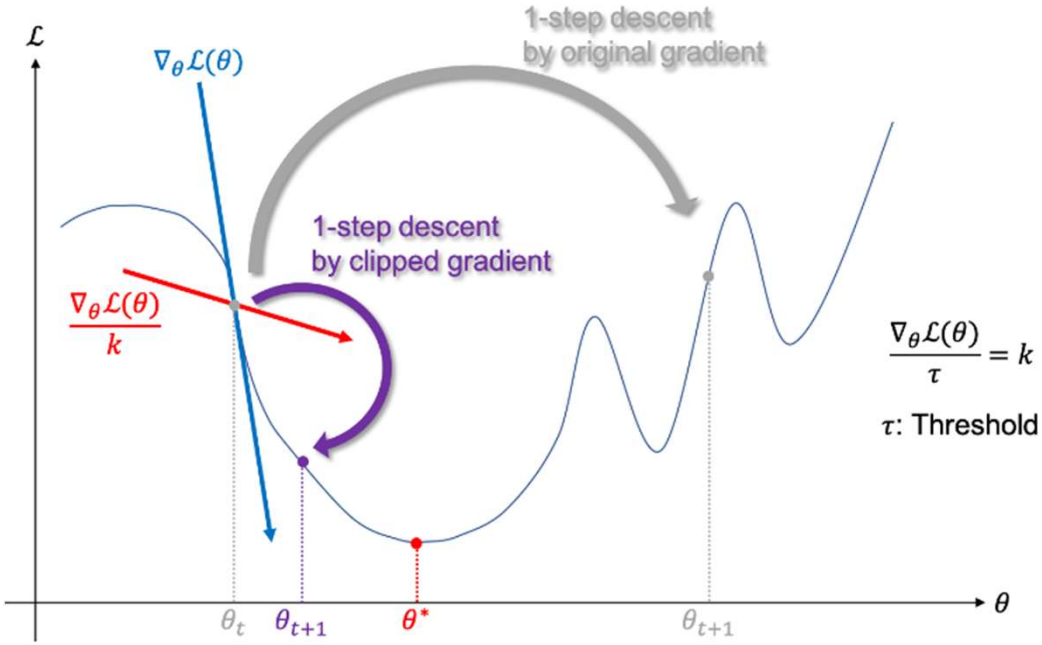
Solution (3) Gradient Clipping

- 역전파 시 임계값을 넘어서지 못하도록 그래디언트를 잘라냄
- 순환신경망에서 자주 사용 (배치 정규화를 사용하기 어렵기 때문에)

1. 그래디언트를 자라내는 기준(threshold)을 설정
2. 임계값을 넘어설 때 그래디언트 벡터의 모든 원소를 -1과 1 사이로 클리핑
3. 그러나 만약 그래디언트 벡터가 [0.9, 100]이면 클리핑 후 벡터는 [0.9, 1]이 되고 그래디언트 벡터의 방향이 변경됨
4. 클리핑을 통해 그래디언트의 방향이 바뀌는 문제를 없애려면 clipnorm(L2값)을 지정해 전체 그래디언트 벡터를 클리핑함.

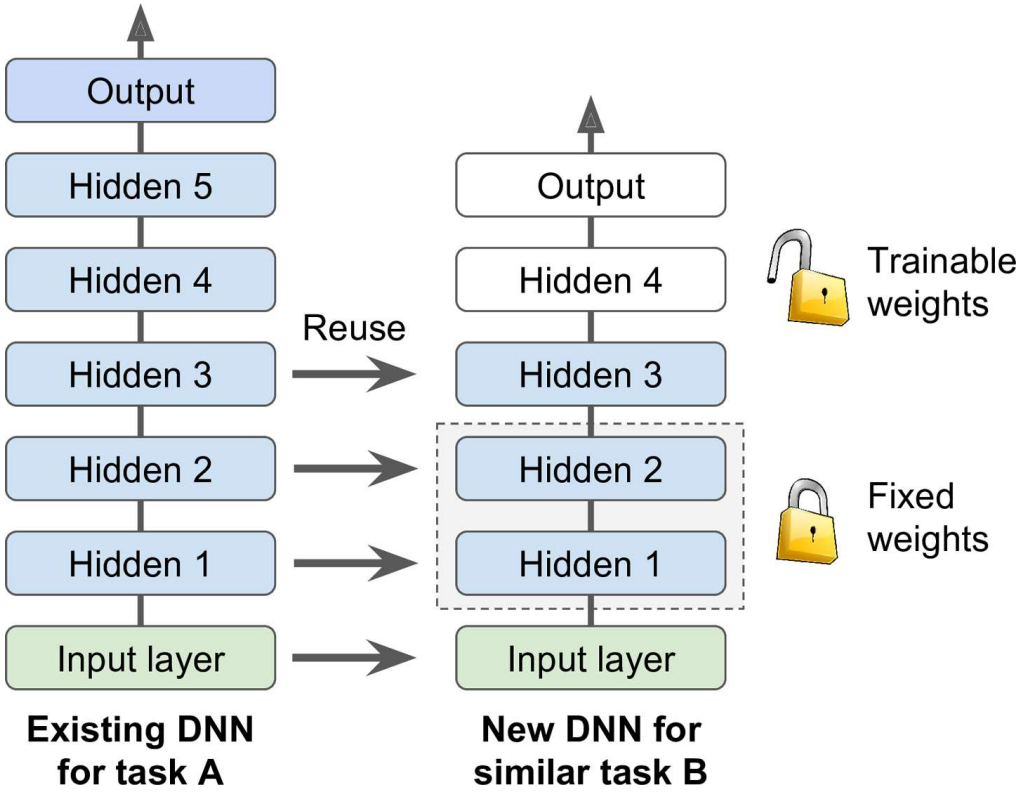
$$\frac{\partial \epsilon}{\partial \theta} \leftarrow \begin{cases} \frac{\text{threshold}}{\|\hat{g}\|}\hat{g} & \text{if } \|\hat{g}\| \geq \text{threshold} \\ \hat{g} & \text{otherwise} \end{cases}$$

$$\text{where } \hat{g} = \frac{\partial \epsilon}{\partial \theta}.$$



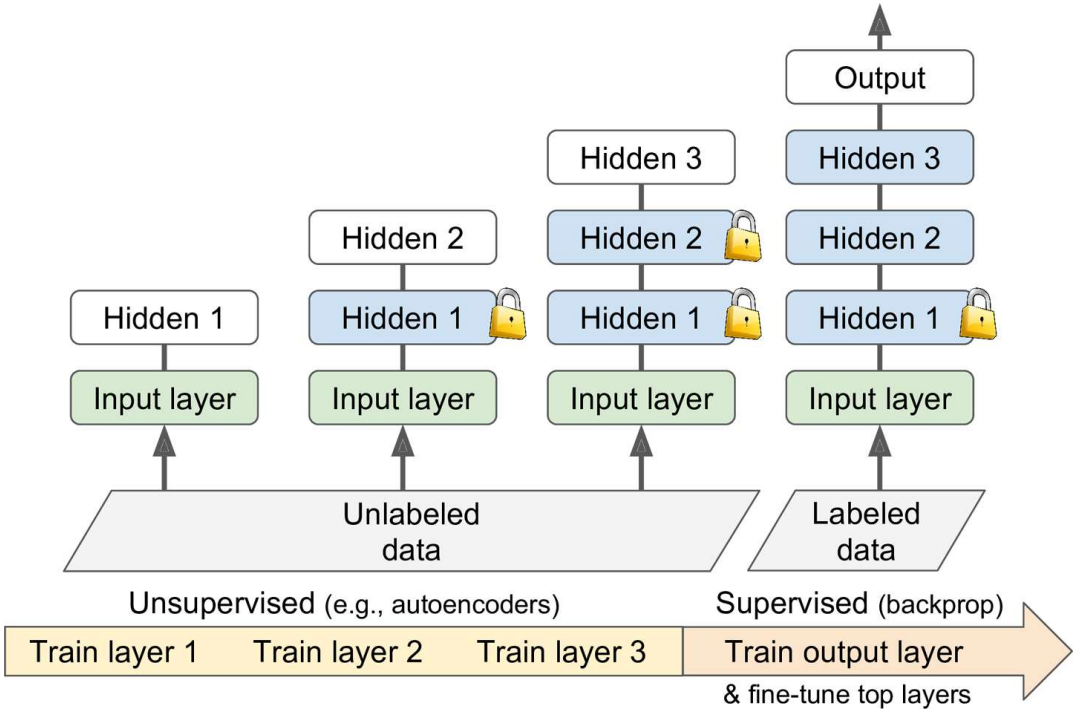
2. Reusing Pretrained Layers (전이학습)

- 하위 신경망을 재사용하여 학습속도와 학습에 필요한 데이터의 수를 줄임
- 원본 신경망에서 재사용할 층은 동결하고 상위 은닉층의 동결은 해제하여 가중치를 조정함
- 재사용할 은닉층의 적절한 개수를 찾을 때까지 작업 반복



2-1 Unsupervised Pretraining

- 전이학습에서 비슷한 데이터에 대한 학습 모델을 찾을 수 없을 때 사용
- 레이블이 없는 데이터에 비지도학습 (오토인코더 or GAN) 모델을 적용하고 그 하위층을 재사용함
- 과거에는 하나의 층을 훈련하고 그 층을 동결한 뒤 새로운 층을 하나 더 얹어 학습시키는 방법을 활용
- 현재는 전체 비지도모델을 학습시키고 필요한 층을 동결시킨 뒤 은닉층을 얹는 방법 활용



* 과거의 방법 (Restricted Boltsmann machine)은 왼쪽 그림의 순서로 진행되고, 현재는 1번에서 곧바로 3번 순서로 넘어가는 방식

**** Pretraining on an Auxiliary Task**

레이블된 훈련 데이터가 없을 때 보조 작업의 첫번째 신경망을 훈련시켜 문제를 해결할 수 있음

Ex. NLP에서 수백만 개 텍스트로 이루어진 코퍼스를 다운로드하여 레이블된 데이터 형성 -> 문장의 일부 단어를 지웠을 때 그것을 예측하는 모델을 훈련시키기

3-1 Momentum Optimization

: 훈련속도를 높일 수 있는 방법 중 하나로 이전 그래디언트 값에 비례한 momentum 값을 현재 그래디언트에 더하여, 좁고 긴 골짜기에서도 최적점에 도달할 때까지 빠르게 내려가도록 함

<기본 그래디언트 업데이트 식>

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta)$$

<모멘텀 알고리즘 업데이트 식>

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta_t} J(\theta_t)$$
$$\theta_{t+1} = \theta_t - v_t$$

1. Momentum gradient 식에서 γ는 0(높은 마찰저항)과 1(마찰저항 없음) 사이의 값으로 모멘텀이 너무 커지는 것을 막기 위한 하이퍼파라미터. 일반적인 γ는 0.9에 해당
2. 그래디언트 세타가 일정할 때 terminal velocity(가장 큰 가중치 업데이트 사이즈)는 1/(1-γ)에 해당. 예를 들어 γ가 0.9이면 terminal velocity는 10으로 일반 그래디언트를 사용할 때보다 10배 빠르게 진행됨
3. 배치정규화를 사용하지 않는 DNN에서 상위 layer는 스케일이 매우 다른 입력값들을 받을 때가 있는데, momentum Algorithm을 사용하면 이러한 문제를 해결하고 local optima를 탈출할 수 있게 도와줌
4. γ는 friction(마찰력)과 같은 작용을 한다. 옵티마이저가 최적값을 뛰어넘었을 때 최적값 근처에서 진동하는 것을 방지하고 빠르게 수렴하도록 함

(시뮬레이션 : https://distill.pub/2017/momentum)

3-2 NAG (Nesterov Accelerated Gradient)

: momentum Optimizer는 현재 방향으로 계속 나아가버리는 (gradien의 방향을 바꾸지 못하는) 문제가 존재할 수 있다. NAG는 현재 위치에서의 gradien를 사용하지 않고 다음 시점의 gradient를 사용하여 momentum 벡터가 올바른 방향을 더 빨리 찾을 수 있도록 돕는다

<모멘텀 알고리즘 업데이트 식>

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta_t} J(\theta_t)$$
$$\theta_{t+1} = \theta_t - v_t$$

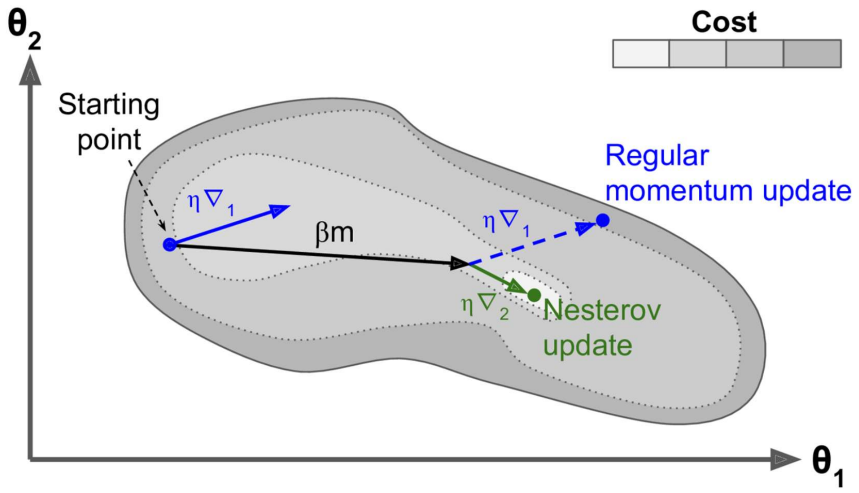
<NAG 알고리즘 업데이트 식>

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta_t} (\theta_t - \gamma v_{t-1})$$
$$\theta_{t+1} = \theta_t - v_t$$

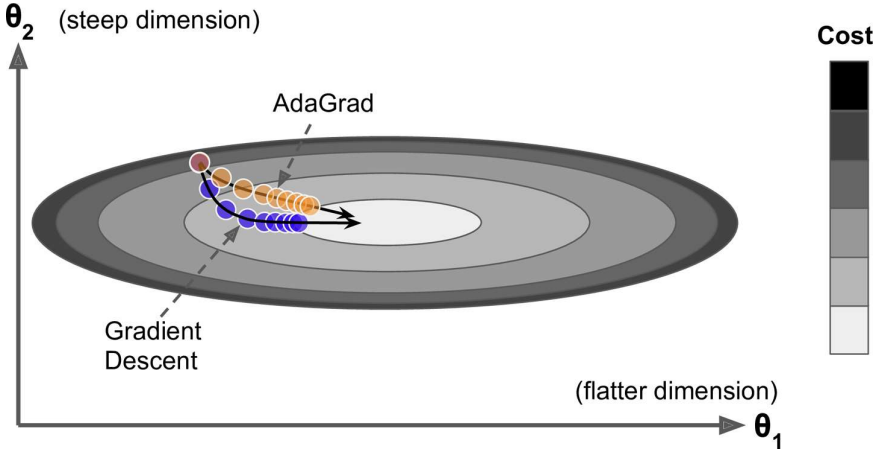
[Momentum과 NAG 비교]

오른쪽 그림에서 Momentum update는 momentum을 더하기 이전 단계에서 계산한 그래디언트를 이용하지만, Nesterov update는 momentum을 더한 이후 단계에서의 gradient를 계산하기 때문에 전역 최적값으로 더 정확하게 접근할 수 있다.

이 때문에 NAG는 momentum 방식보다 훈련속도가 빠르다.



3-3 AdaGrad, RMSProp



Vanilla Gradient Descent일 때 그래디언트는 더 가파른 차원을 따라 내려가게 된다 (가파른 차원은 그래디언트가 더 크게 생성되기 때문에 방향이 가파른 쪽을 먼저 따라가게 됨)

이 경우 global optima에 도착하는 시간이 늦어질 수 있기 때문에 이미 많이 움직인 곳(가파른 곳)에 대한 제약이 필요하다.

<AdaGrad 업데이트 식>

$$G_t = G_{t-1} + (\nabla_\theta J(\theta_t))^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_\theta J(\theta_t)$$

<RMSProp 업데이트 식>

$$G = \gamma G + (1-\gamma)(\nabla_\theta J(\theta_t))^2$$

$$\theta = \theta - \frac{\eta}{\sqrt{G + \epsilon}} \cdot \nabla_\theta J(\theta_t)$$

1.그래디언트 벡터의 제곱을 element-wise로 계산하여 Gt에 저장한다. 이 벡터 원소에 루트를 씌워 각 그래디언트 벡터 원소를 나누어 스케일을 조정한다. 이로써 그래디언트 벡터가 한 방향을 따라 가는 현상을 막을 수 있다.
2.AdaGrad는 간단한 2차 방정식에는 잘 작동하지만 깊은 신경망을 훈련할 때는 학습률이 너무 빠르게 감소되어 global optima에 도착하기 전에 멈추는 경우가 있다.

3.그래디언트가 무한히 커지는 문제를 막기 위해 decay rate 감마를 도입한 RMSProp를 고안

3-5 Adam and Nadam, AdaMax

<Adam 알고리즘 업데이트 식>

1. $\mathbf{m} \leftarrow \beta_1 \mathbf{m} - (1 - \beta_1) \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
2. $\mathbf{s} \leftarrow \beta_2 \mathbf{s} + (1 - \beta_2) \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
3. $\widehat{\mathbf{m}} \leftarrow \dfrac{\mathbf{m}}{1 - {\beta_1}^t}$
4. $\hat{\mathbf{s}} \leftarrow \dfrac{\mathbf{s}}{1 - {\beta_2}^t}$
5. $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \widehat{\mathbf{m}} \oslash \sqrt{\hat{\mathbf{s}} + \varepsilon}$

- AdaGrad + Momentum 개념
- 3,4는 초기 m, s 값이 0에 가깝기 때문에 해당 값을 증폭시키는 단계로 활용됨
- (1~4의 과정을 거치면 m과 s 앞에 붙은 지수는 beta/(1-beta)가 됨

<Nadam 알고리즘 업데이트 식>

$$g_t = \nabla_{\theta_t} J(\theta_t)$$
$$m_t = \gamma m_{t-1} + \eta g_t$$
$$\theta_{t+1} = \theta_t - (\gamma m_t + \eta g_t)$$
$$\therefore \theta_{t+1} = \theta_t - m_{t+1}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

- Adam 알고리즘에 Nesterov 방법 적용
- 그래디언트 업데이트에 이전 시점(t-1)이 아닌 현재 시점(t)을 적용하여 계산

<AdaMax 알고리즘 업데이트 식>

$$g_{ij}^{(t)} = \beta_2^p g_{ij}^{(t-1)} + (1 - \beta_2^p) \left| \frac{\partial L}{\partial w_{ij}^{(t)}} \right|^p = (1 - \beta_2^p) \sum_{i=1}^{t} \beta_2^{p(t-i)} \left| \frac{\partial L}{\partial w_{ij}^{(t)}} \right|^p$$

- L2 norm을 적용
- Adam이 동작하지 않을 때 시도할 수 있는 옵티마이저

4-1 l1 and l2 Regularization

- L2 규제 : 각 가중치 제곱의 합에 규제 강도(Regularization Strength) λ를 곱한다. 그 값을 손실함수에 더한다. λ를 크게 하면 가중치가 더 많이 감소되고(규제를 중요시함), λ를 작게 하면 가중치가 증가한다(규제를 중요시하지 않음). 가중치를 갱신할 때, 손실함수의 미분값을 이전 가중치에서 빼서 다음 가중치를 계산한다. 따라서 가중치가 크면 손실 함수가 커지고, 다음 가중치가 크게 감소된다.

$$L_{regularized}(\mathbb{W};\mathbb{X},\mathbb{Y}) = L(\mathbb{W};\mathbb{X},\mathbb{Y}) + \lambda||\mathbb{W}||_2^2$$

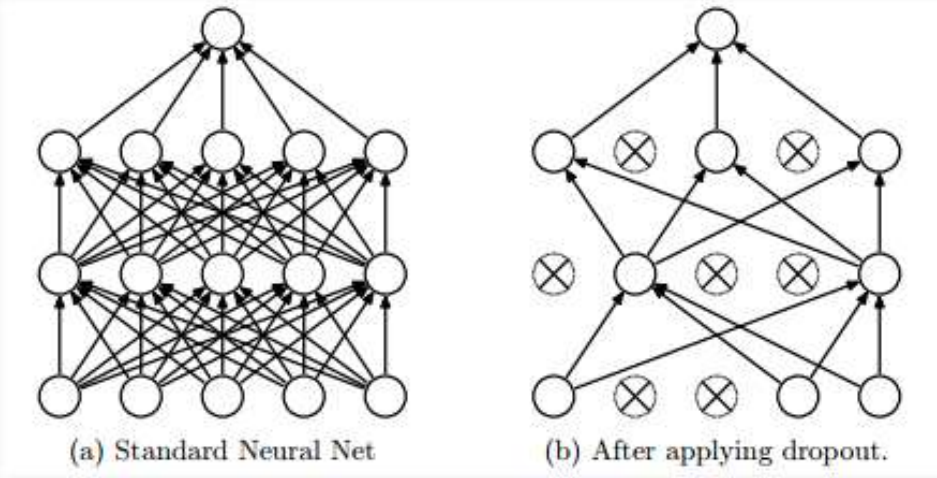
$$where||\mathbb{W}||_2^2 = w_1^2 + w_2^2 + \ldots + w_n^2$$

- L1 규제 : L1 규제는 가중치의 제곱의 합이 아닌 가중치의 합을 더한 값에 규제 강도(Regularization Strength) λ를 곱하여 오차에 더한다. 이렇게 하면 L2 규제와는 달리 어떤 가중치는 실제로 0이 된다. 즉, 모델에서 완전히 제외되는 특성이 생기는 것이다. 일부 계수를 0으로 만듦으로써 모델을 이해하기 쉬워지고, 모델의 가장 중요한 특성이 무엇인지 드러나는 것이다. 그러나 L2 규제가 L1 규제에 비해 더 안정적이라 일반적으로는 L2규제가 더 많이 사용된다.

$$L_{regularized}(\mathbb{W};\mathbb{X},\mathbb{Y}) = L(\mathbb{W};\mathbb{X},\mathbb{Y}) + \lambda||\mathbb{W}||$$

$$,where||\mathbb{W}|| = |w_1| + |w_2| + \ldots + |w_n|$$

4-2 Dropout



- 훈련 스텝에서 입력 뉴런의 일부를 확률적으로 무시
각 뉴런의 드롭아웃 확률 p를 dropout rate로 지칭하며,
10%와 50%사이의 값으로 설정.
(RNN은 20~30%, CNN은 40~50%)

- 훈련 이후의 뉴런에는 드롭아웃을 적용하지 않음 (즉 test set 입력할 때는 드롭아웃이 적용되지 않음)

1.일반적으로 가장 상위 레이어부터 세번째 레이어까지만 드롭아웃을 적용함

2.특정 뉴런이 과대한 영향력을 가지는 것을 방지하여 과대적합 문제를 해결

3.테스트 이후에는 드롭아웃을 적용하지 않기 때문에 연결 가중치가 더 낮아지도록 keep probability (1-p)를 곱해줘야 함

ex) p=50% 이면 신경망은 훈련 때보다 두 배 많은 입력값을 받게 되므로, 훈련 상태와 비슷하게 만들기 위해 연결가중치에 0.5를 곱해줘야 함

4.과소적합시 드롭아웃을 줄이고, 과대적합시 드롭아웃을 늘리고, 층이 적을 시에는 드롭아웃을 줄이는 식으로 적절하게 튜닝 가능

5.드롭아웃은 훈련속도를 느리게 만드는 경향이 존재

4-3 Max-Norm Regularization

- 각 뉴런에 대한 연결 가중치 w가 $\| \mathbf{w} \|_2 \leq r,$ 의 식을 만족하도록 함
- 여기서 감마는 Max-Norm 정규화의 하이퍼 파라미터이고, 바깥 식은 l2 규제(제곱)를 뜻함.
- 매 훈련스텝이 끝날 때마다 가중치의 l2 식을 구하여 특정 값을 넘어갈 때 가중치의 값을 조정함
- 이는 불안정한 그래디언트를 완화하는 데 도움이 되고, 배치 정규화를 사용하지 않았을 경우 효과적임