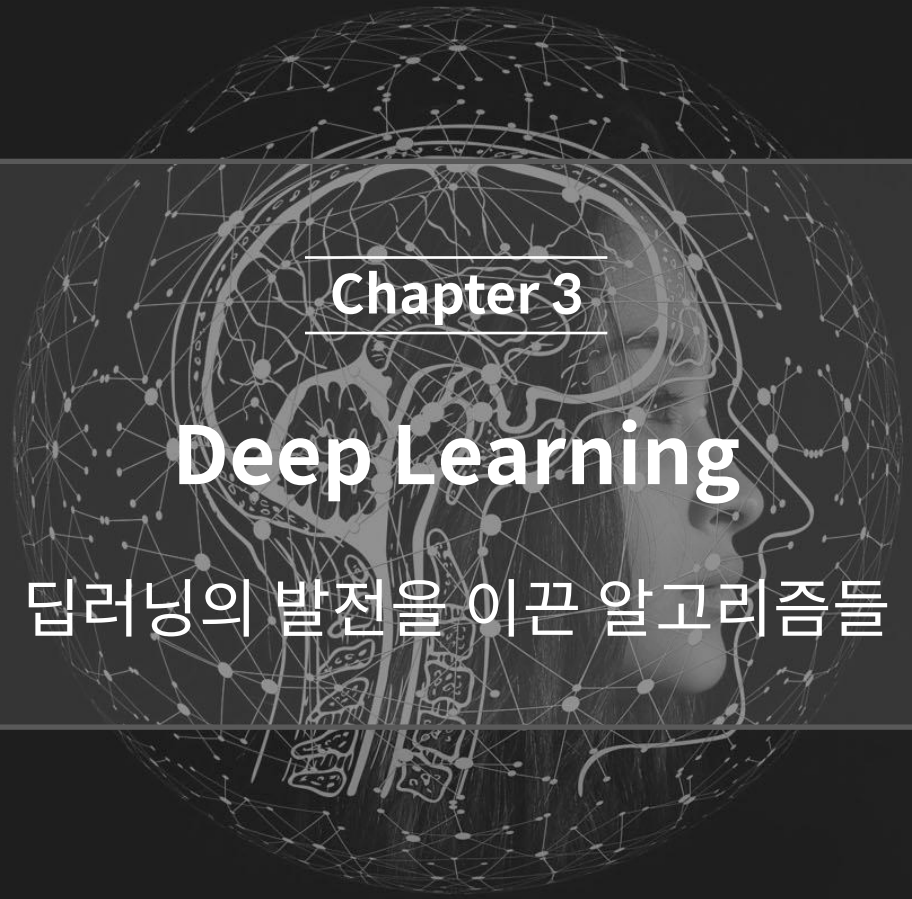


Chapter 3

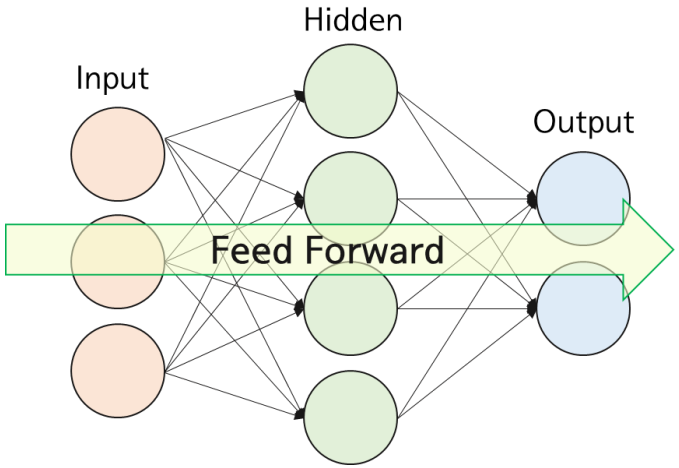
Deep Learning

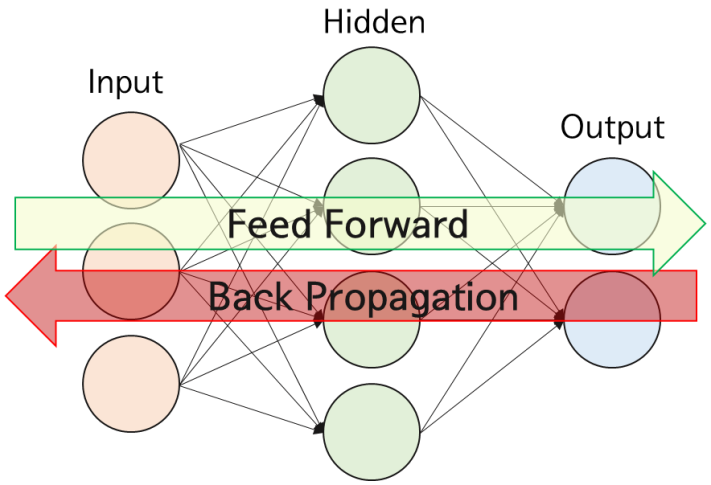
딥러닝의 발전을 이끈 알고리즘들

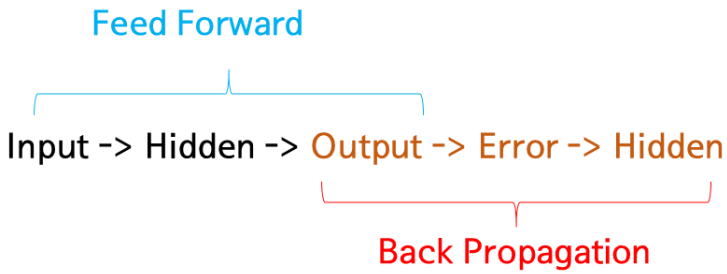
MLP

Multi Layer Perceptron 학습

- Feed Forward : 신경망의 Input에서 Weight와 Hidden을 거쳐 Output을 내보내는 과정
- Back Propagation : Feed Forward를 통해서 Input에서 Output까지 계산하며 Weight를 Update하는 과정







신경망 모형의 단점

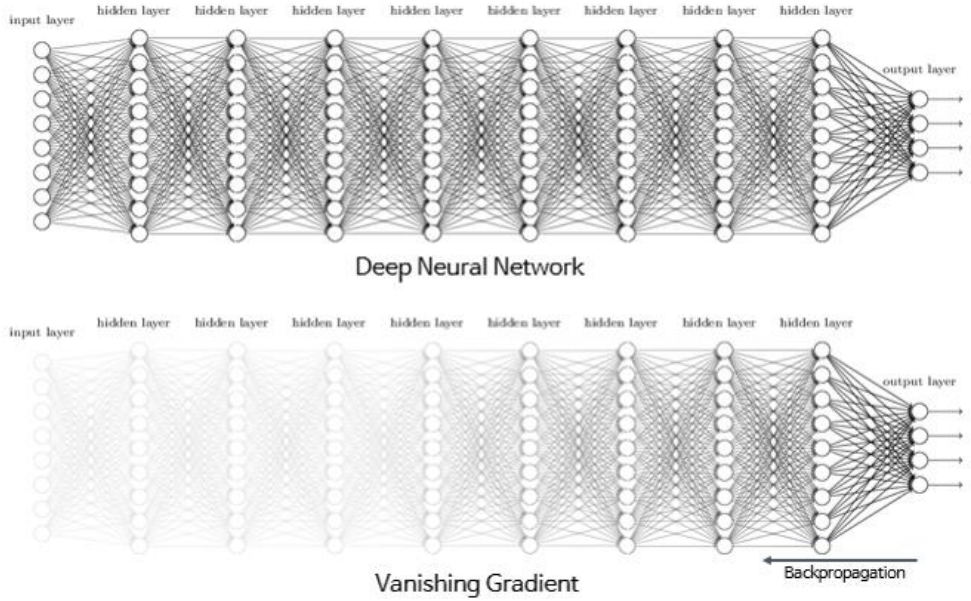
Universal Approximation Theorem

- 신경망 이론 중에 가장 유명하고 신경망 학습의 특성을 잘 나타내 주는 이론
- Hidden layer가 하나 이상인 신경망은 학습데이터 내에서 어떠한 함수 든 근사 (Approximation)시킬 수 있다
- 기본적인 MLP이상의 신경망은 학습 데이터 내에서 어떠한 모델이든 만들 수 있다라는 것
- 내가 가지고 있는 학습 데이터 내에서는 어떠한 함수 든 만들 수 있지만, 실제 데이터에는 잘 맞는다라는 보장은 하지 못하는 것
- 결국에 Overfitting의 개념과 이어짐. 학습 데이터내에서 완벽하게 맞춘다라는 것은 이전에 서술했던 것처럼 Overfitting이 굉장히 심해질 수 밖에 없다는 것을 뜻함

신경망 모형의 단점

Gradient Vanishing Problem

- Hidden Layer가 깊어지면 깊어질 수록 앞의 Gradient를 구해주기 위해서는 이 Sigmoid의 미분한 값을 계속 곱해 주어야 함
- Layer가 깊어지면 깊어질수록 앞 부분의 Weight의 Gradient는 큰 변화가 없어지고, Weight의 변화가 거의 일어나지 않는 현상
- Layer가 깊으면 깊을수록 Gradient Vanishing이 일어나면서 신경망이 가지는 장점이 발휘 되지 못함



출처 : https://excelsior-cjh.tistory.com/177

딥러닝의 정의

Deep Learning의 정의

- 2개 이상의 Hidden Layer를 가지는 다층 신경망 (Deep Neural Network; DNN)
- Graphical Representation Learning

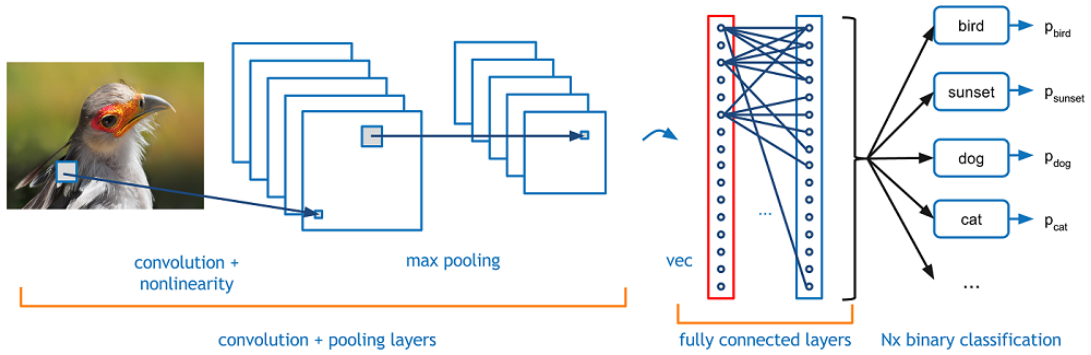
Deep Learning이 발전하게 된 계기

- Overfitting과 Gradient Vanishing을 완화 시킬 수 있는 알고리즘이 발전 한 것
- Graphics Processing Unit (GPU)를 신경망의 연산에 사용할 수 있게 되면서 고속 연산이 가능해짐
- Big Data시대의 도래

딥러닝의 종류

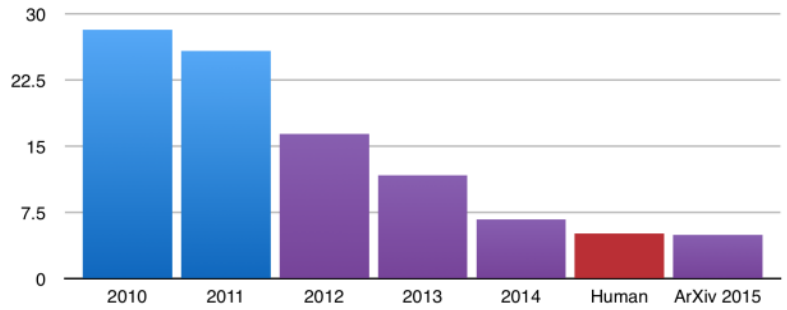
Deep Learning의 종류

- 이미지 처리에 많이 쓰이는 Convolutional Neural Network(CNN)



기본적인 CNN 구조

고양이와 강아지를 분류하는 모델



년도 별 ImageNet데이터 분류 모델 성능

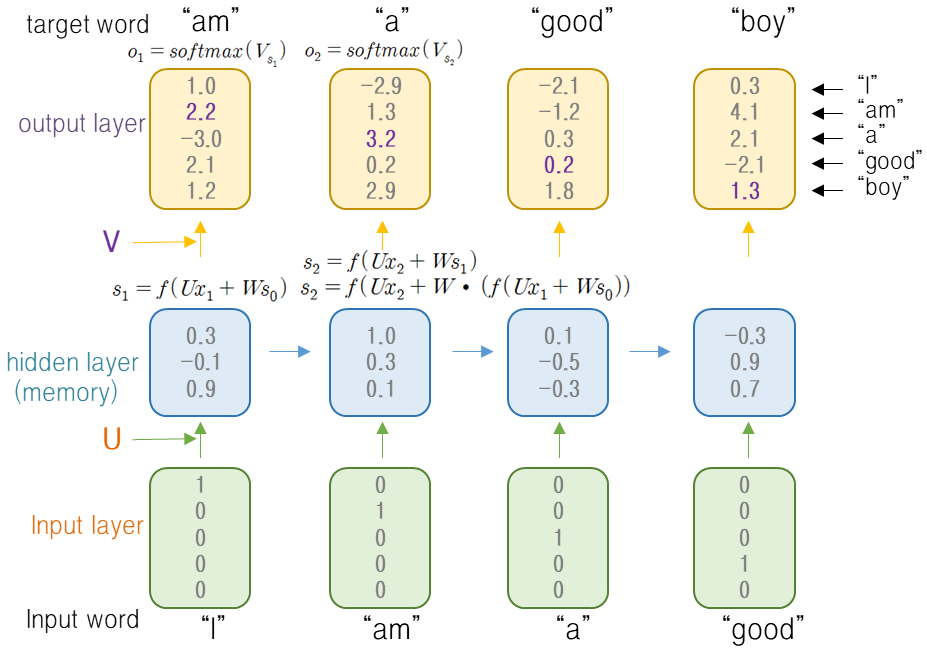
출처 : https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

출처 : https://excelsior-cjh.tistory.com/177

딥러닝의 종류

Deep Learning의 종류

- 자연어 처리에 많이 쓰이는 Recurrent Neural Network(RNN)



기본적인 RNN 구조

딥러닝의 정의

Deep Learning의 정의

- 2개 이상의 Hidden Layer를 가지는 다층 신경망 (Deep Neural Network; DNN)
- Graphical Representation Learning

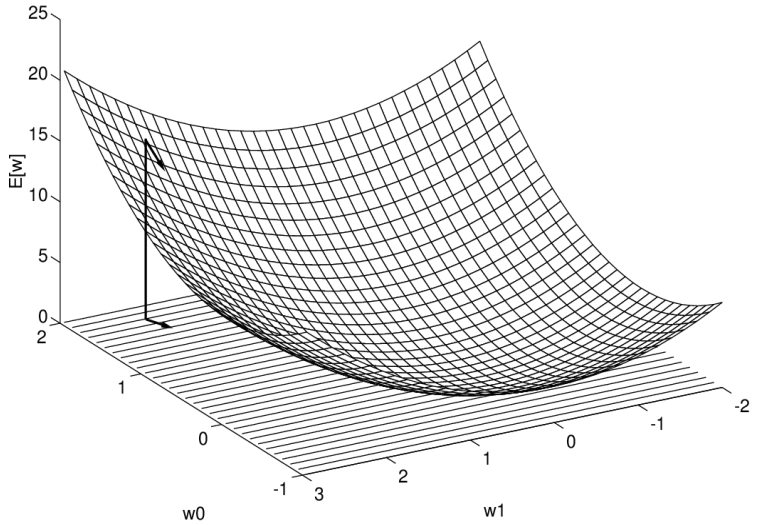
Deep Learning이 발전하게 된 계기

- Overfitting과 Gradient Vanishing을 완화 시킬 수 있는 알고리즘이 발전 한 것
- Graphics Processing Unit (GPU)를 신경망의 연산에 사용할 수 있게 되면서 이를 해결하게 된
- Big Data시대의 도래

MLP Back Propagation

Gradient Descent Method

- 신경망 모형 같은 경우는 앞서 서술하였듯이 매우 복잡한 모형.
- Hidden Layer가 깊어지면 깊어질수록, Hidden Node가 많아지면 많아질수록 더욱 복잡 해짐. 그렇기 때문에, 신경망 모형에서 MSE를 신경망 모형의 Weight로 미분해서 기울기가 0이 지점을 찾을 수 없음
- MSE를 신경망 모형의 Weight로 미분하여 기울기를 감소시켜 최소가 되는 지점을 찾아갑니다. 다시 말해, 한번에 기울기가 최소가 되는 지점을 찾기 어렵기 때문에, 산 정상에서 차근차근 내려오면서 길을 찾듯이 기울기를 조금씩 구해 MSE가 낮아지는 지점을 찾아 감



신경망 모형의 단점

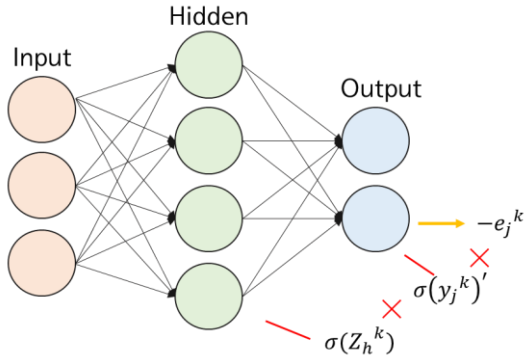
Gradient Vanishing Problem

- 기울기가 사라지는 현상을 의미

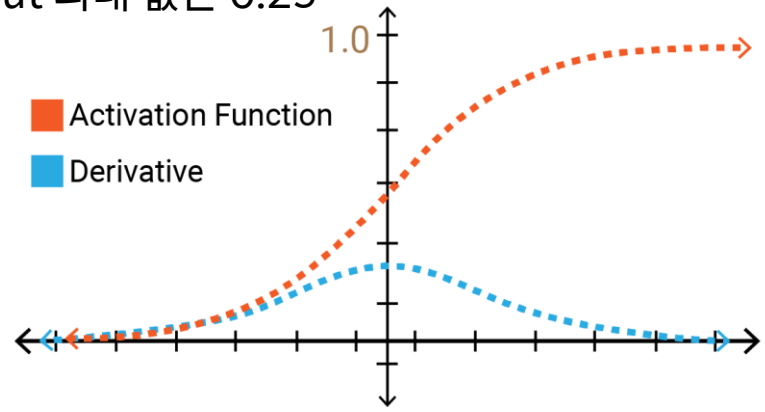
Back Propagation 계산 - Weight의 기울기는 Error부터 전파 되어 옴

$$\therefore \frac{\partial \varepsilon_k}{\partial w_{hj}{}^k} = \frac{\partial \varepsilon_k}{\partial \sigma(y_j{}^k)} \frac{\partial \sigma(y_j{}^k)}{\partial y_j{}^k} \frac{\partial y_j{}^k}{\partial w_{hj}{}^k}$$

$$-\left(t_j^k - \sigma\left(y_j{}^k\right)\right) = e_j{}^k \quad \sigma(y_j{}^k)\left(1 - \sigma\left(y_j{}^k\right)\right) \qquad \sigma(Z_h{}^k)$$



- Sigmoid의 미분한 값의 Output 최대 값은 0.25

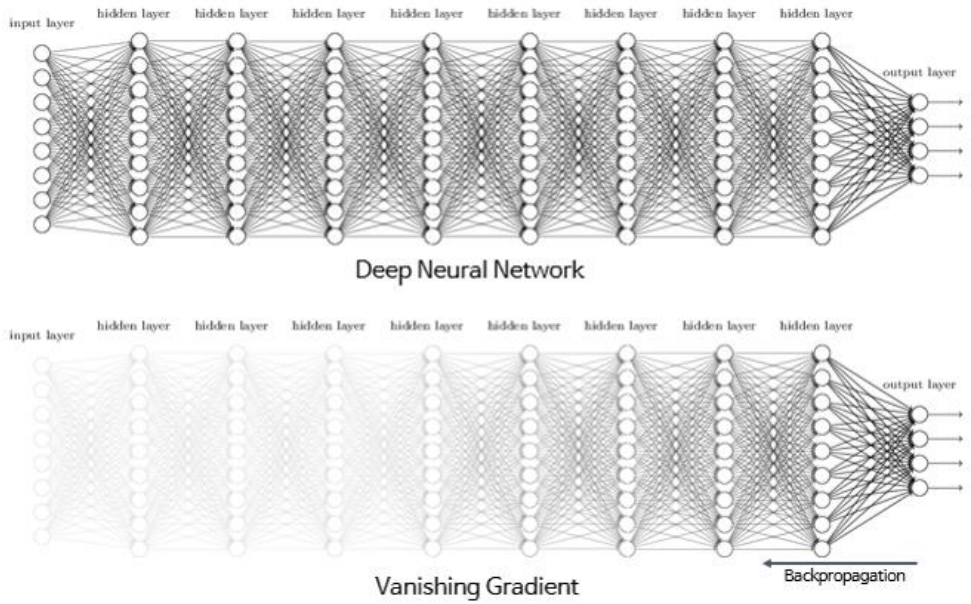


출처 : https://codeodysseys.com/posts/gradient-calculation/

신경망 모형의 단점

Gradient Vanishing Problem

- Hidden Layer가 깊어지면 깊어질 수록 앞의 Gradient를 구해주기 위해서는 이 Sigmoid의 미분한 값을 계속 곱해 주어야 함
- Layer가 깊어지면 깊어질수록 앞 부분의 Weight의 Gradient는 큰 변화가 없어지고, Weight의 변화가 거의 일어나지 않는 현상
- Layer가 깊으면 깊을수록 Gradient Vanishing이 일어나면서 신경망이 가지는 장점이 발휘 되지 못함



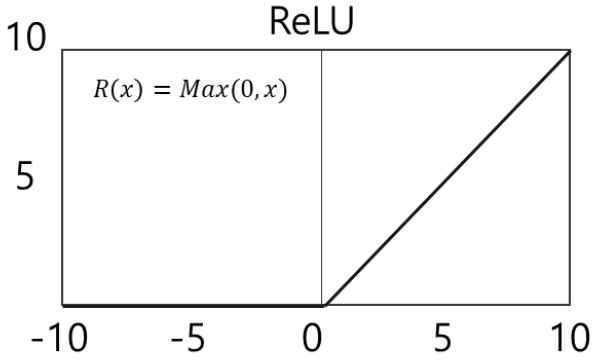
출처 : https://excelsior-cjh.tistory.com/177

새로운 Activation Function

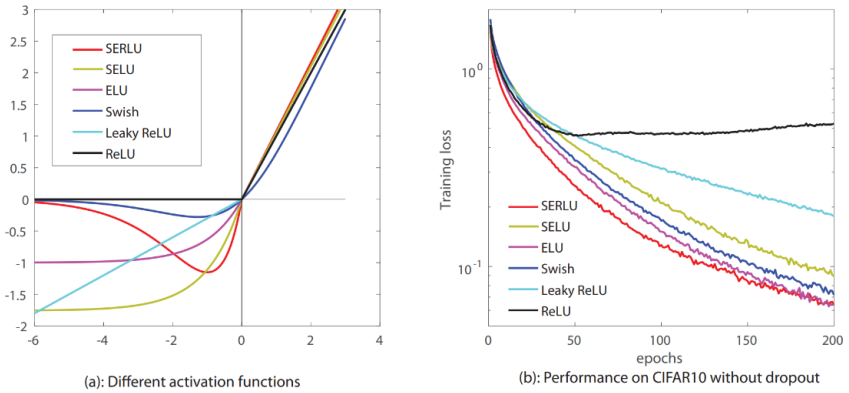
새로운 Activation Function – ReLU (rectified linear unit)

ReLU: R(z)=max(0, z)

- 0이하는 0, 0이상은 z값을 가짐
- Back Propagation 과정 중에서, 곱해지는 Activation 미분 값이 0 또는 1이 되기 때문에, 아예 없애거나 완전히 살리거나로 해석할 수 있음
- Gradient vanishing (exploding) 을 완화시킴
- 미분 계산이 간단함
- 학습 속도가 빠름 (빠르게 수렴)
- ReLU의 응용버전인 Leaky ReLU, Parametric ReLU가 있음



ReLU함수 형태



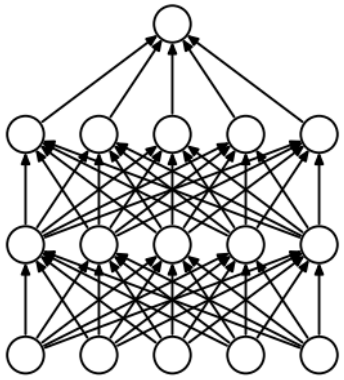
다양한 Activation Function

출처: https://towardsdatascience.com/a-comprehensive-guide-on-activation-functions-b45ed37a4fa5

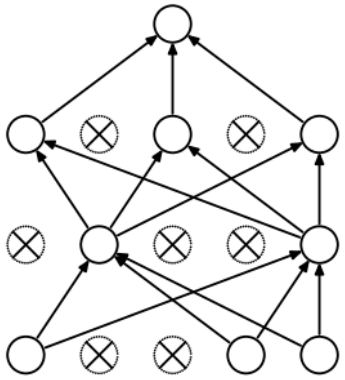
DropOut

DropOut

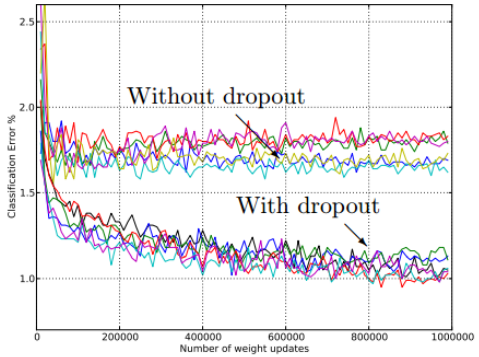
- 신경망의 학습과정 중에 Layer의 Node를 Random하게 Drop함으로써, Generalization효과를 가져오게 하는 테크닉
- Dropout을 적용한 다는 것은 Weight Matrix에 랜덤하게 일부 Column에 0을 집어넣어 연산을 한다는 것
- Epoch마다 랜덤하게 Dropout시킴
- 처음 제안된 이후로도 현재까지 기본적으로 신경망을 디자인할 때에 범용적으로 많이 사용되고 있는 테크닉 (ReLU + DropOut)



일반적인 신경망 구조



Drop-out이 적용된 신경망



MNIST에 적용한 Drop-out

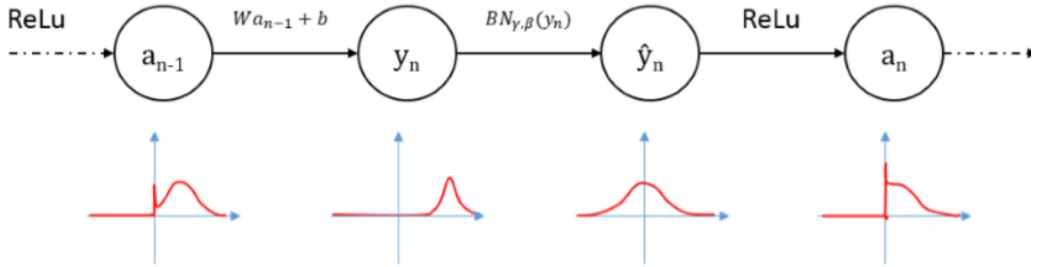
Drop-out의 robust한 성질을 보이기 위해 다양한 아키텍쳐를 비교

출처 : Dropout: A Simple Way to Prevent Neural Networks from Overfitting
(http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Batch Normalization

Batch Normalization

- Internal covariance shift : 각 layer마다 input 분포가 달라 짐에 따라 학습 속도가 느려지는 현상



- Batch Normalization
  - Input 분포를 scale시키고 shift 시킴으로써 layer의 분포를 normalization시킴
  - 학습 속도를 향상 시켜 주고 gradient vanishing problem도 해결
  - 분포를 정규화 시켜 비선형 활성 함수의 의미를 살리는 개념이라고

Scale

$$BN(h;\gamma,\beta)=\beta+\gamma\frac{h-E(h)}{\sqrt{(Var(h)+\epsilon)}}$$

Shift

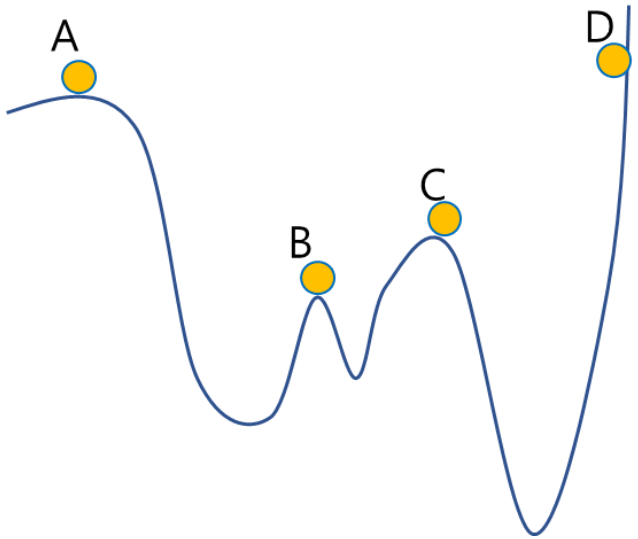
β 와 γ는 BP과정을 통해 학습

출처 : https://guillaumebrg.wordpress.com/2016/02/28/dogs-vs-cats-adding-batch-normalization-5-1-error-rate/

Initialization

Initialization

- 신경망은 처음에 Weight를 랜덤하게 초기화 시키고 Loss가 최소화가 되는 부분을 찾아감.
- 이전에는 기본적으로 초기 분포로 Uniform Distribution이나 Normal Distribution을 사용.
- Weight를 랜덤하게 초기화 시키면서 아래 그림에서 신경망의 초기 Loss가 달라짐
- 신경망을 어떻게 초기화 시키느냐에 따라 학습 속도가 달라 질 수 있음.



Loss function의 다양한 초기 값

Initialization

LeCun Initialization

- LeCun이라는 Convolutional Neural Network의 창시자의 이름에서 따온 기법
- 초기 분포가 다음과 같은 분포를 따르도록 weight를 초기화 시키는 것

LeCun Normal Initialization

$$W \sim N\left(0, Var(W)\right)$$

$$Var(W) = \sqrt{\frac{1}{n_{in}}}$$

n_{in} = 이전 layer의 node수

LeCun Uniform Initialization

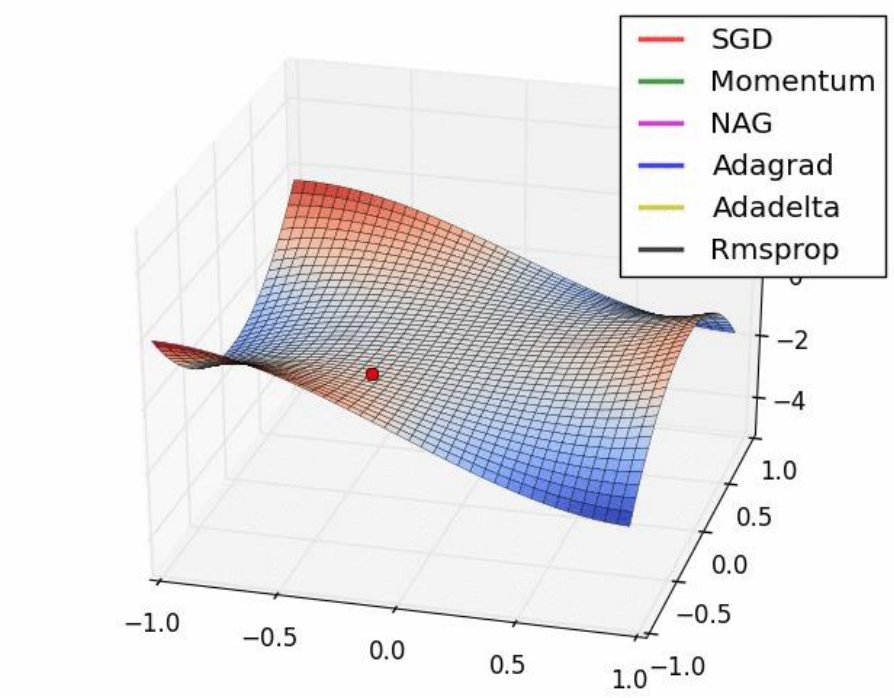
$$W \sim U\left(-\sqrt{\frac{1}{n_{in}}}, +\sqrt{\frac{1}{n_{in}}}\right)$$

- He Initialization, Xavier Initialization등 다양한 Initialization 기법 존재

Optimizer

다양한 Optimizer

- Batch단위로 Back Propagation하는 과정을 Stochastic Gradient Descent (SGD)라고 하며, 이러한 과정을 Optimization라고 함.
- SGD외에 SGD의 단점을 보완하기 위한 다양한 Optimizer가 존재



- SGD : 전부다 한번에 gradient를 구하기 어렵기 때문에, 조금씩 보고 gradient를 구하겠다. (batch)
- Momentum : 내려오던 방향으로 관성을 주자
- NAG : 관성방향으로 먼저 움직이고 스텝
- Adagrad : 안 가본 곳은 많이 움직이고 가본 곳은 조금씩 움직임
- AdaDelta : 움직임이 멈추는 것을 막음
- RMSProp : Adagrad의 update 버젼
- Adam : RMSProp + Momentum

Optimizer

다양한 Optimizer

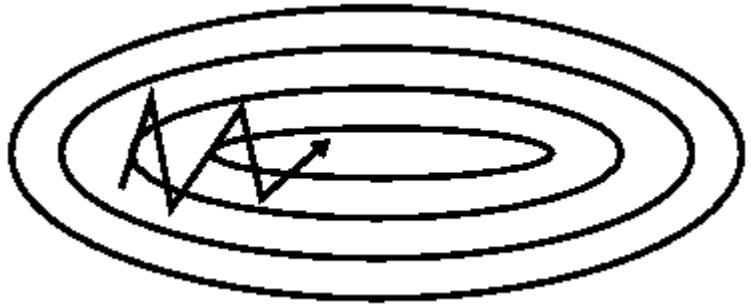
- SGD : 전부다 한번에 gradient를 구하기 어렵기 때문에, 조금씩 보고 gradient를 구하겠다. (batch)

$$\theta = \theta - \eta \nabla_\theta J(\theta)$$

- Momentum : 내려오던 방향으로 관성을 주자

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

SGD의 해를 찾는 모습



SGD에 Momentum이 적용되어 해를 찾는 모습

출처 : http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html

Optimizer

다양한 Optimizer

- Nesterov Accelerated Gradient (NAG) : Momentum으로 이동을 한 이후에, Gradient를 구하여 이동하는 방식으로

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

- Adaptive Gradient (Adagrad) : Adagrad는 가보지 않은 곳은 많이 움직이고 가본곳은 조금씩 움직이는 개념을 가지는 방법

$$G_t = G_{t-1} - (\nabla_\theta J(\theta_t))^2$$
$$\theta_{t+1} \leftarrow \theta_t - \frac{\eta}{\sqrt{G+\epsilon}} \nabla_\theta J(\theta_t)$$

- RMSProp : Adagrad의 단점을 보완한 방법. Adagrad는 학습이 오래 진행 될수록 G_t부분이 계속 증가하여 Step Size가 작아진다라는 단점이 있는데, RMSProp은 G가 무한히 커지지 않도록 지수평균을 내어 계산

$$G = \gamma G + (1 - \gamma)\left(\nabla_\theta J(\theta_t)\right)^2$$
$$\theta_{t+1} \leftarrow \theta_t - \frac{\eta}{\sqrt{G+\epsilon}} \nabla_\theta J(\theta_t)$$

Optimizer

다양한 Optimizer

- Adaptive Delta (Adadelta) : Adadelta또한 Adagrad의 단점을 보완한 방법. Gradient를 구하여 움직이는데 Gradient의 양이 너무 적어지면 움직임이 멈춰질 수 있음. 이를 방지하기 위한 방법이 Adadelta

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

$$\Delta_\theta = \frac{\sqrt{s + \epsilon}}{\sqrt{G + \epsilon}} \nabla_\theta J(\theta_t)$$

$$\theta_{t+1} \leftarrow \theta_t - \Delta_\theta$$

$$s = \gamma s + (1 - \gamma)\Delta_\theta^2$$

- Adaptive Moment Estimation (Adam) : Adam은 딥러닝 모델을 디자인 할 때에, 기본적으로 가장 많이 사용하는 Optimizer로서 RMSProp과 Momentum방식의 특징을 결합한 방법

Optimizer

다양한 Optimizer

Rectified Adam optimizer (RAdam)

- Adam뿐만 아니라, 대부분의 Optimizer들은 학습 초기에 Bad Local Optimum에 수렴 해 버릴 수도 있는 단점이 존재
- 학습 초기에 Gradient가 매우 작아 져서 학습이 더 이상 일어나지 않는 현상이 발생하는 것
- RAdam은 이러한 Adaptive Learning Rate Term의 분산을 교정 (Rectify)하는 Optimizer

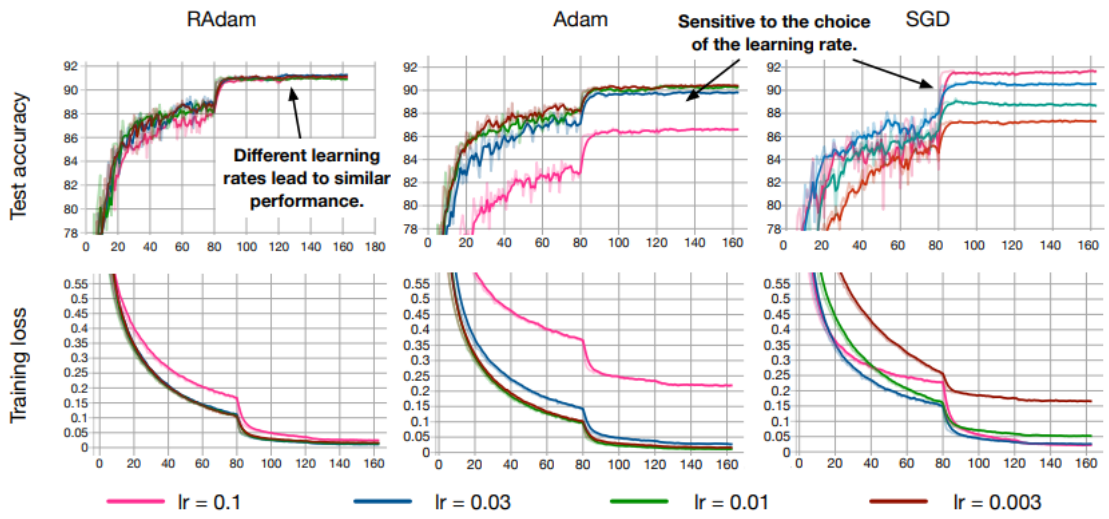


Figure 6: Performance of RAdam, Adam and SGD with different learning rates on CIFAR10. X-axis is the number of epochs.

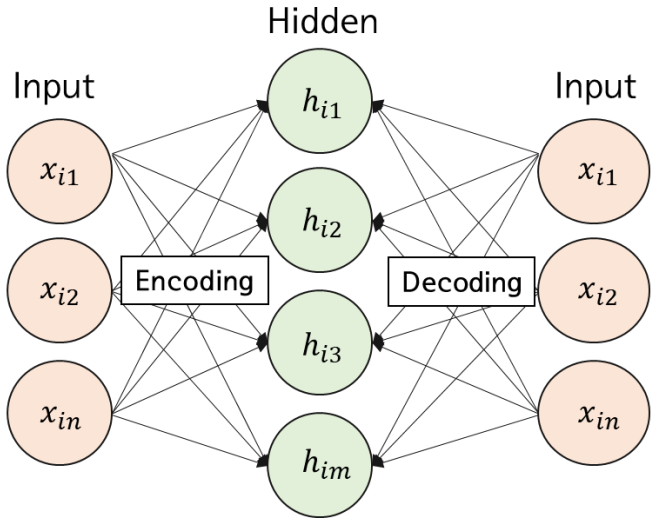
출처 : https://arxiv.org/pdf/1908.03265.pdf

AutoEncoder

AutoEncoder (AE)

- **입력층 값을 출력층에서 그대로 예측**하는 것을 목적으로 구성된 인공 신경망 모형 (Bengio et al., 2007)으로 **데이터 셋을 압축적으로 표현**하여 다층 신경망의 가중치 학습 문제를 해결

  - Autoencoder의 구조는 오른쪽 그림과 같음
  - Y(출력) = X(입력) 로 자기 자신을 학습시킴
  - Autoencoder를 이용하여 학습된 가중치를 다층 인공 신경망의 초기 가중치로 사용
  - 출력층에서 를 계산해서 $\hat{X}$ 를 얻고 error ($\hat{X}$-X) 계산
  - Error가 최소화 되는 방향으로 BP를 통해 weight update



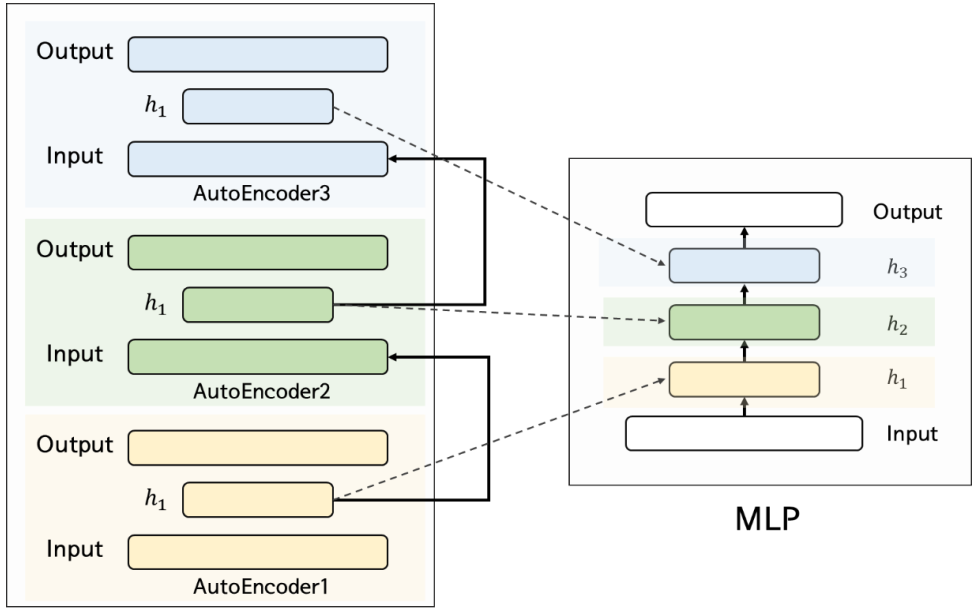
AutoEncoder

Stacked AutoEncoder (SAE)

- **AE를 아래 그림과 같이 쌓아 올린 모델**
- **AE의 새로운 Feature가 Feature로서의 의미가 있다면 이를 쌓아 올려서 학습시키면 더 좋은 학습 모델을 만들 수 있을 것**

1. Input Data로 AE1을 학습
2. ①에서 학습된 모형의 Hidden Layer를 Input으로 하여 AE2를 학습
3. ②과정을 원하는 만큼 반복
4. ①~③에서 학습된 Hidden Layer를 쌓아 올림
5. 마지막 Layer를 Softamx와 같은 Classification기능이 있는 Output Layer 추가
6. Fine-tuning으로 전체 다층 신경망을 재학습

- SAE는 '좋은 Feature를 가지는 Hidden Layer를 쌓아서 네트워크를 학습시키면 더 좋은 모델을 만들 수 있을 것이다' 컨셉을 가지는 모델
- 미리 학습시키는 모델을 Pre-Trained Model이라고 하며, 이 모델을 재학습 시키는 과정을 Fine-tuning이라고 함



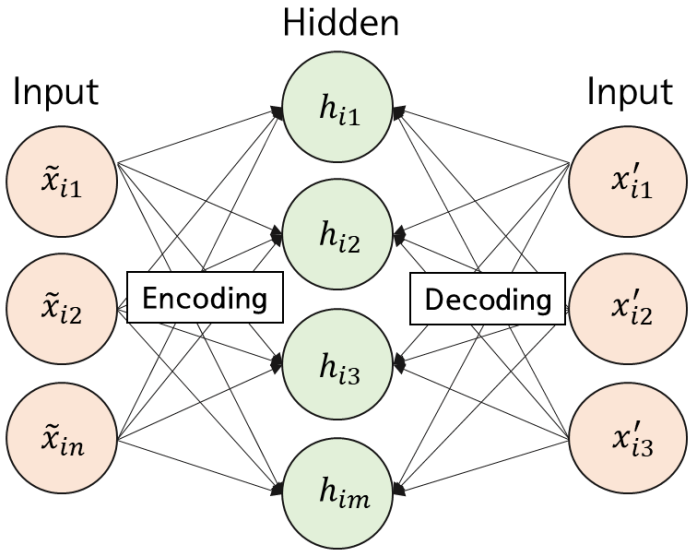
Stacked AutoEncoder

AutoEncoder

Denoising AutoEncoder (DAE)

- Good representation ≒ robust feature
- Input data에 noise를 주어 학습 시킴으로써, 어떠한 데이터가 input으로 오든 간에 강건한 (robust) 모형을 만듦
- "안개가 낀 상황에서 운전을 연습을 해두면, 실전에 도움이 됨"

"극한 상황에서 훈련을 해야 (training), 실전 (test)에 도움이 됨"



1) Corruption step $\tilde{\boldsymbol{x}} = \boldsymbol{x} + noise$
2) Hidden representation $\boldsymbol{h} = sigmoid(\boldsymbol{W}\tilde{\boldsymbol{x}} + b)$
3) Reconstruct $\boldsymbol{x}' = \boldsymbol{W}'\boldsymbol{h} + b$
4) Minimize error $\boldsymbol{x} - \boldsymbol{x}'$