# Activation Functions

## 활성화함수란 신경회로망에서 각 뉴런들에 들어오는 입력신호의 합을 출력신호로 변환시키는 함수이다. 따라서 딥러닝에서 어떤 활성함수가 사용되었는지에 따라 Vanishing gradient 문제로 학습 정도가 달라지기도 한다. 인공신경망은 사람의 신경구조를 모방하여 만들어진 것인데, 가장 기본개념은 여러 개의 신호가 들어오면 이를 조합해서 다음 신호를 결정한다는 것이다. 다음 신호가 존재하는지 또는 존재하지 않는지 판단하는 것을 넘어 얼마나 세게 신호를 전달할지 정하는 것이 활성함수이다. 활성함수에는 선형함수와 비선형 함수(non-linear)가 있다.

Logistic Regression 로지스틱 회기분석은 Generalized linear model 일반화된 선형 함수의 여러 케이스 중의 하나이다. 이 로지스틱 회기분석의 목적은 어떠한 사건이 발생할 확률을 알고자 하는 것이다. 선형 함수이므로 어떤 사건이 발생할 확률 y는 y= ax+b로 나타낼 수 있는 것이다. 하지만 직관적으로 알 수 있듯이 선형함수의 문제점은, x의 값에 따라 확률 y가 100%가 넘는 경우가 발생한다는 것이다. 따라서 선형함수의 장점인 이해 및 계산이 쉬운 방법을 유지하면서도 확률의 공리에 어긋나지 않는 방법을 사용하는 로지스틱 함수를 연결함수로 사용하게 되었다.

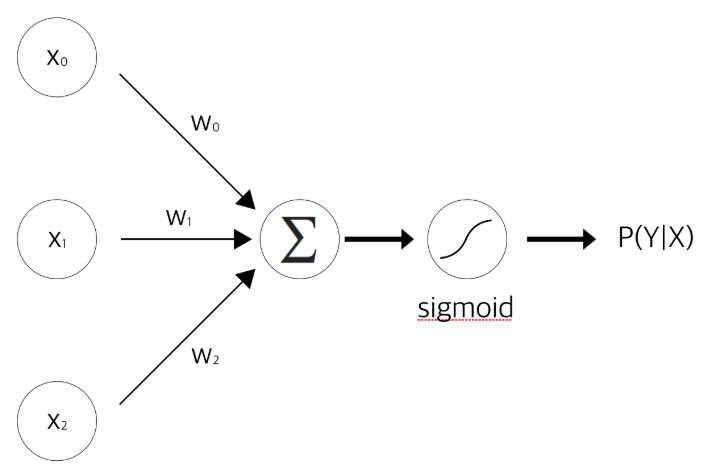
\*비선형 함수 및 multi hidden layer

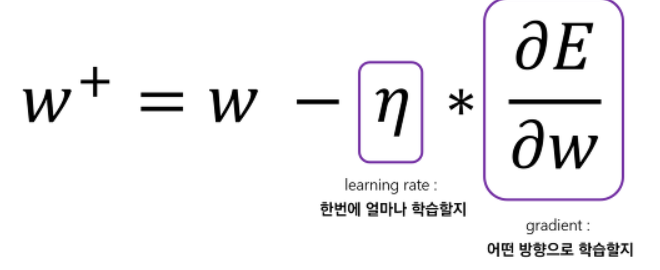
복잡한 현상을 선형함수로 linearization하는 것은 결과 해석에 도움이 되지 않기 때문에 non linearlization 을 해야 할 때가 많다. 이처럼 비선형의 분류를 하기 위하여는 첫째, 1개 이상의 hiddenlayer 가 존재해야 하고 둘째, 비선형의 활성화함수가 필요하다. 왜냐하면 만약 활성화함수가 선형이라면 각 뉴런의 결과값은 선형결합이 된다. 따라서 아무리 다중 층을 쌓는다 하더라도 선형결합이 되기 때문에 한계를 벗어나지 못한다. 따라서 비선형활성화함수가 필요하다.

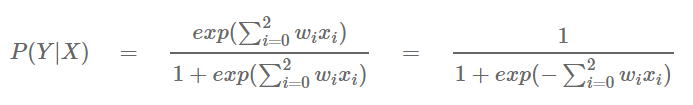
Hidden layer의 개수가 중요한 이유는 결국 logistic regression 도 선형 모델이기 때문이다. 왜냐하면 결과값이 특정값 이상이면 1로 분류되고, 그렇지 않은 경우 0으로 분류되는 이분법이 적용되기 때문이다. 따라서 층이 1개인 경우 비록 비선형의 활성화함수를 사용하더라도, 비선형 함수의 지수를 풀어내면 결국 선형 결합의 결과값에 대한 분류가 되므로 목적했던 비선형의 분류를 할 수 없다. 결국, 비선형의 활성화함수를 사용하되 다수의 hiddenlayers 를 사용해야 한다. 나아가 hidden layer 의 개수가 늘어날수록 좀더 비선형 데이터에 적합한 분류가 가능하다. 다만 overfitting 의 문제가 있기 때문에 과제마다 적절한 층이 필요하다.

로지스틱 함수는 대표적으로 Sigmoid가 있다.

g(x)=1/(1+e^x) , P(y|x) = e^bx/(1+e^bx) – 위 함수를 이용하면 결과값의 범위가 0~1까지 제한되어 확률값의 예측에 사용할 수 있게 된다.

위의 그림은 hiddenlayer가 없는 단순한 형태의 feed forward neural network 형태이다. 즉 Xn 을 가중치 Wn 곱해서 더한 값이 Sigmoid 라는연결함수를 거치면 y라는 사건이 일어날 확률 P가 되는 것이다. Weight는 인공신경망에서 각 뉴런에 곱해지는 모수를 추정하는 과정이다. (parameter estimate) 따라서 Sigmoid를 활성 함수로 사용하는 다중층의 인공신경망은 각 층의 학습과정에서 weight라는 모수를 학습을 통해 추정하게 된다. 아래와 같은 식으로 새로운 weight가 도출된다. 따라서 Gradient 는 activation function 의 도함수에 의해 결정됨을 알 수 있다.





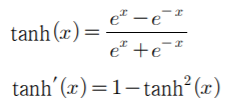
Sigmoid Function ; logistic 함수라고도 불린다. 수식은 다음과 같다. 신경회로망에서 많이 쓰이는 함수이다. 하지만 어느정도 숫자 이상이거나 이하일 경우 Backpropagation 을 할 때 Gradient 값이 작아지는 현상이 나타난다. 그결과 학습률이 떨어지고, 미분 수치가 작아 계산이 복잡하게 되는 문제점이 있다. (Gradient Vanishing현상) 또한 sigmoid function의 범위는 [0,1] 로서 모두 0이상의 값을 가지고 이는 w의 학습시 허용되는 방향에 제약이 되므로 학습속도가 늦거나 수렴이 어렵게 된다. 이로 인해 깊은 층의 신경회로망을 만들 경우 Sigmoid 만 사용하기는 어렵다. Softmax 같은 분류과정에서 사용되거나 적은 층의 다중 층에 사용된다. 이러한 문제점은 함수값이 0에 대해 대칭인 하이퍼볼릭탄젠트 함수로 극복할 수 있을 것이다.

[https://pozalabs.github.io/Activation\_Function/#](https://pozalabs.github.io/Activation_Function/)

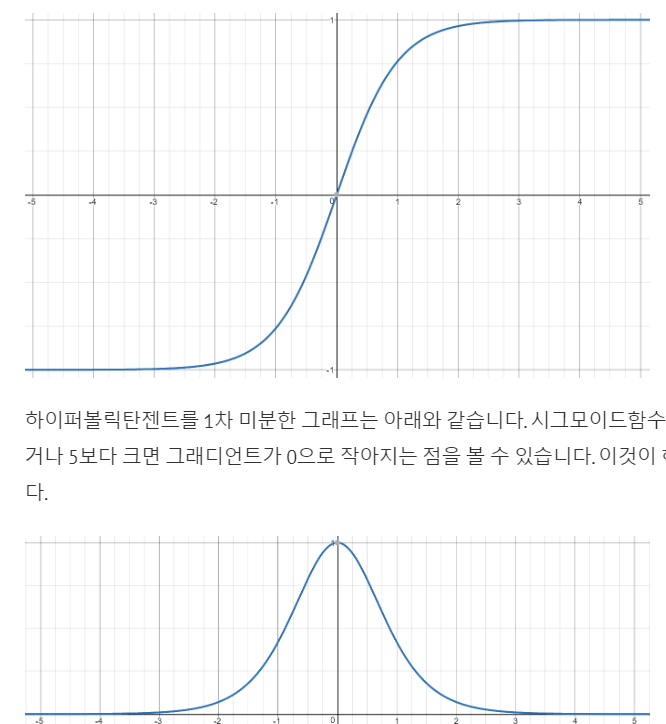
<https://www.slideshare.net/jasonhong56/gomguard-yolo>

<http://gomguard.tistory.com/183> Vanishing Gradient 정의 및 원인

<https://ratsgo.github.io/deep%20learning/2017/04/22/NNtricks/> 활성화 함수 설명

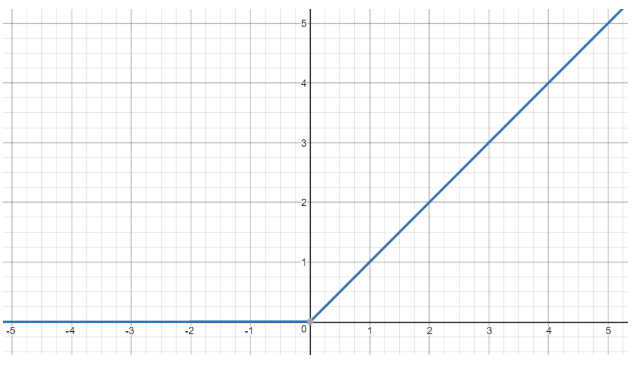
\* Hyperbolic Tangent Function

Sigmoid 함수에서 크기와 위치가 조절된 함수이다. 그 범위는 [-1.1] 이고, 0을 기준으로대칭된점이 특징이다. 이 때문에 Sigmoid 보다 학습의 수렴 속도가 더 빠르다. 그러나 여전히 Sigmoid 함수가 갖는 단점을 가지고 있다. 1차 미분 하였을 때, 시그모이드 함수와 마찬가지로 x가 [-5,5]범위를 벗어나게 되면 Gradient 가 0으로 작아지기 때문이다.



\*Relu (rectified linear units)

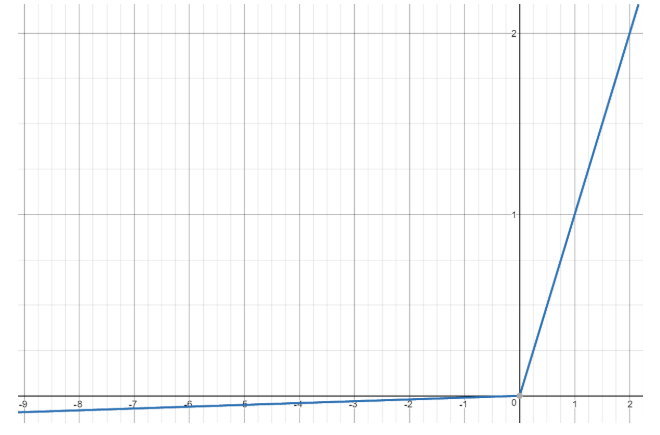
Relu 함수는 최근 들어 가장 많이 쓰이는 활성화 함수 이다. 음의 값에서는 0 양의 값에서는 x 를 갖는 함수이다. Y=max(x,0); relu unit



Relu 함수는 Fully connected layer 뿐만 아니라 Convolution layer에서도 사용되는데, Con layer 에서 각각의 픽셀들을 뉴런들로 보고, 이 뉴런들을 Relu 함수를 통과시키기 때문이다. Relu 함수의 장점은 Gradient vanishing 현상이 없다는 것과 선형이므로 계산이 매우 효율적이며 수렴 속도가 Sigmoid 에 비해 훨씬 빠르다는 점이다. 다만 tanh와 달리 zero centered 하지 않아 gradient descent 할 때 정확한 방향으로 가지 못할 위험이 있다. Relu함수의 linear한 식을 거친 웨이트는 또다른 레이어의 입력값이 되기 때문이다. 또한 음의 값을 항상 0으로 만들기 때문에 음의 값을 갖는 뉴런은 업데이트가 안될 수 있지만 컬러 이미지는 거의 양의 값을 가지므로 이러한 단점은 큰 문제가 되지 않는다. 그러나 다른 활성화함수 처럼 파라미터들이 업데이트가 되지 않는다는 단점을 가지고 있다. 이를 보완하기 위해 leaky Relu라고 하여 Relu 의 변형 함수가 있고, 또다른 Elu 라는 변형 함수도 최근에 많이 쓰이고 있다.

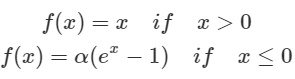
\*leaky Relu

F(x)=max(0.01x, x) 식을 갖는 Leaky Relu 는 x가 음수일 때 Gradient 가 0.01을 갖는다는 점이 특징이다.

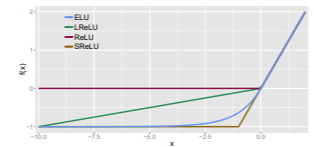


\*Expotential Linear Units ( ELU) https://arxiv.org/pdf/1511.07289.pdf

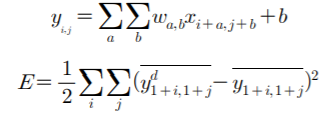
Elu는 Relu의 특성을 공유하며 Gradient 가 vanishing 되지 않는 장점이 있다. ELU는 [Clevert et al., 2015](http://arxiv.org/abs/1511.07289)(Cornell univeristy) 에 의해 나온 비교적 최신 방법이다. ElU 는 Relu의 threshold 를 -1 로 낮춘 함수를 exp 함수를 이용하여 근사한 것이다. 그 특징으로는, Relu의 장점을 모두 포함하며, x가 음수일 때Gradient가 소멸하는 문제를 해결하였고, 출력값이 zero centered 에 가깝다.( 합수값의 중심이 0이 아닌 것이 문제가 되는 이유는 어떤 뉴런의 입력값이 모두 양수일 때, 파라미터의 Gradient 는 입력값에 의해 영향을 받아 파라미터의 부호가 모두 같게 되고, 이는 gradient descent를 할 때 정확한 방향으로 가지 못하고 지그재그로 수렴하는 문제가 발생하기 때문이다) 하지만 단점으로는 Relu, Leaky relu 와는 달리 exp함수를 계산해야 하는 Cost 가 든다.



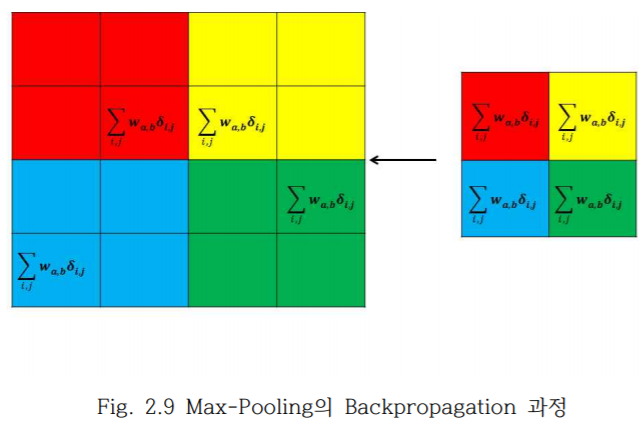
\*함수 비교

ReLU, Leaky Relu (a=0.1) shifted ReLU, ELU (a=0.1) 

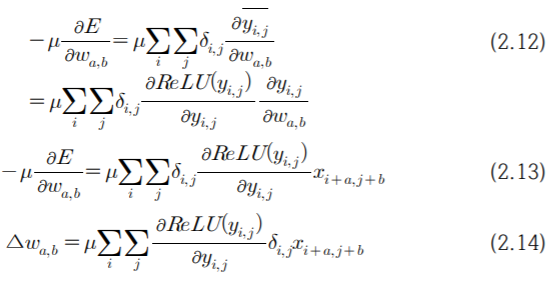
# BackPropagation

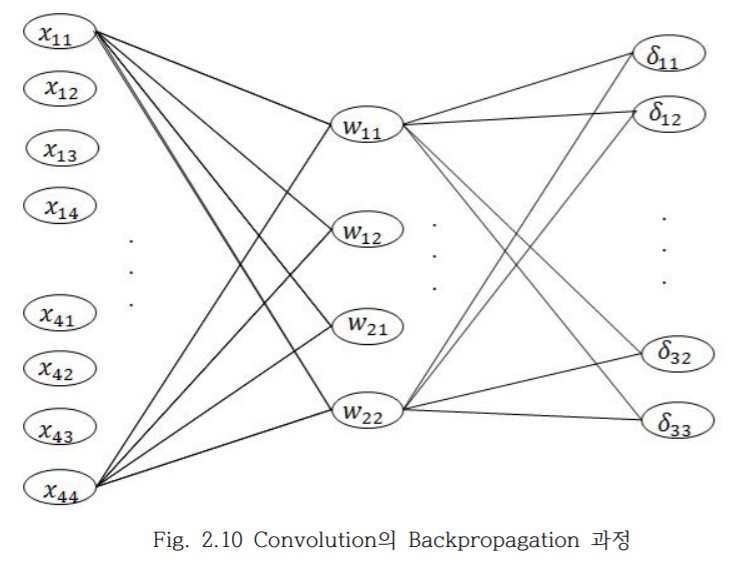
위의 필터링 과정을 거친 다음 Fully connected layer 을 통과하게 된다. Fully connected layer는 CNN의 back propagation 과정 중 하나이다. 기존의 신경회로망에서 Error는 다음과 같은 식으로 계산된다.  w는 웨이트, b는 bias, ybar는 활성화함수를 통과한 y. 이를 바탕으로 Pooling layer 와 Convolutional layer 가 각각 어떻게 back propagation 하는지 본다.

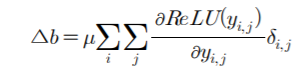
Back propagation은 역과정을 의미하기 때문에 Pooling 과정을 먼저 간략히 설명하겠다. Max pooling은 필터를 통과한 각 영역들의 최댓값이 통과하므로 backpropagation 과정을 통해 다시 최댓값이면 1을 아니면 0을 곱해준다. 그러면 max pooling의 back propagation 과정은 다시 2\*2 에서 4\*4 로 커지게 된다. 즉 그림과 같이 원래의 픽셀 최댓값 자리로 들어가게 되는 것이다.



마찬가지로 convolutional layer 에서 backpropagation 과정은 convolutional filter을 통과하는 각 픽셀을 일종의 노드로 본다. 4\*4 입력에 2\*2 필터를 씌웠을 때 역으로 4\*4 의 Gradient 가 적용되는 것을 그림과 식으로 나타내면 다음과 같다.





이처럼 식 2.1.2를 통해 식 13,14,15를 얻을 수 있고, 식과 같이 밑의 첨자들이 이미지 필터의 행과 열의 위치를 알려준다. 즉 이를 통해 이미지와 필터의 크기와 픽셀의 위치도 알 수 있다.

# Fully connected layer

## Fully connected 레이어 내의 뉴런들은 이전 레이어의 모든 액티베이션들과 연결되어 있다. 그러므로 Fully connected레이어의 액티베이션은 매트릭스 곱을 한 뒤 바이어스를 더해 구할 수 있다. FC layer 와 컨벌루션 레이어의 차이는 컨벌루션 레이어는 일부 영역에만 연결되어 있고 컨벌루션 볼륨의 많은 뉴런들이 파라미터를 공유한다는 것이다. 두 레이어 모두 내적 연산을 하므로 함수 형태는 동일하다. 따라서 컨벌루션 레이어를 FC레이어로 변환하는 것이 가능하다.

\*실험의 컴퓨터 환경

본 연구에서는 학습을 위해 다음 스펙의 컴퓨터를 사용하였다.

CPU/ RAM/ GPU/ SSD

OS; window 10 , Google colab은 jupyter 환경과 유사하여 사용하는데 어려움이 없었다. 속도도 window 환경에서의 Anaconda 보다 매우 빨랐다. 따라서 보통 효율적인 알고리즘은 시간과 정확도를 따지는데, 본 논문에서는 정확도만 측정하였다. Tensorflow는 딥러닝을 쉽게 하기 위한 툴인데 Keras 는 tensorflow 의 여러 라이브러리 중 상위단계에 속한 것으로 Tensorflow 에서 함수의 사용을 쉽게 하도록 도와준다.



# 코드에 대한 설명

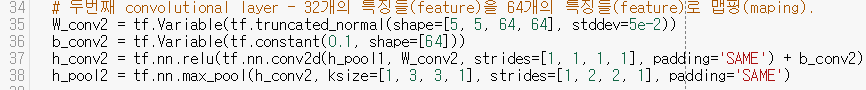
## 인풋 데이터는 CIfar10 으로서 10개의 카테고리를 갖는 여러 이미지들이 들어있다. INPUT 이미지는 가로32,세로32, RGB체널을 가지므로 입력의 크기는 [32\*32\*3] 이다.

## 

## Convolutional layer 는 입력 이미지의 영역과 연결되어 있으며, 이 연결된 영역과 자신의 가중치의 내적연산 (dot product)를 계산한다. 즉 Padding 1을 하여 3\*3 필터를 64개 Convolutional layer1에 통과시킨다. 같은 크기의 Convolutional layer2와 Padding 2을 통과시켰다. 그 결과 볼륨은 [32\*32\*64]의 크기를 갖게 되었다. (하나의 grayscale 이미지를 64개의 특징들 feature로 매핑하기로 설계).

## 

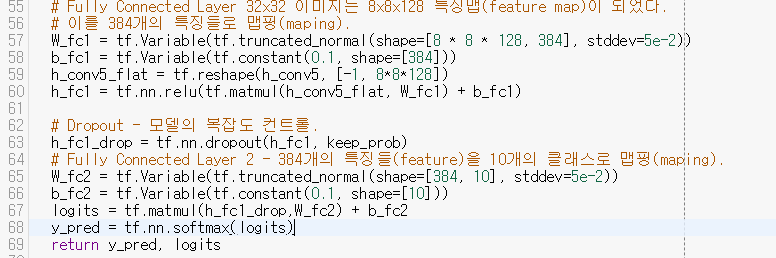
## Relu layer 는 각 요소에 적용되는 액티베이션 함수(activation function)이다. 이 레이어는 볼륨의 크기를 변화시키지 않는다. Pooling layer 는 가로 세로에 대해 down sampling을 수행해 [16\*16\*64] 와 같이 볼륨을 반으로 줄인다. 볼륨을 줄이는 과정에서 Pooling 시에 Max 값을 대표값으로 하거나, avg 값을 대표값으로 하여 볼륨을 줄여 보았다. 두번의 Pooling layer을 사용할 경우 이미지 사이즈는 8\*8 이 된다.



Convolutional layer 3 이후도 이전의 레이어 들과 같이 이미지 사이즈가 줄지 않게 하고 128개의 필터를 통과하는 것으로 구성된다. 더 이상의 Pooling layer 가 없기 때문에 사이즈가 유지된다. 다만 Stride를 1\*1 로 하여 더 촘촘히 검토하도록 하였다.

## 

## 다섯번째 convolutional layer 을 통과한 이미지는 128개의 특징을 갖는 8\*8 사이즈의 이미지가되었다. 이것을 FC(fully connected)layer에 일렬로 만들어 넣으면 8\*8\*128 개의 노드를 가지게 된다. 이것을 384개의 특징들로 매핑을 하고, 최종적으로 클래스 점수를 계산해 [1\*1\*10]의 크기를 갖는 볼륨을 출력한다. 10개의 숫자들은 10개 카테고리에 대한 클래스 점수를 나타내 어떤 카테고리에 있는지 판단해준다. 즉 모든 노드는 마지막 output에서 클래스 개수에 따라 Softmax로 이미지를 분류한다.



코드 중 dropout은 학습 시킬 때 신경회로망 전체가 아닌 일부만 학습시키는 것을 말한다. Dropout은 신경회로망이 커질수록 over fitting 문제가 생기는데 이를 보완하는데 사용된다. Overfitting은 특정 뉴런의 w 웨이트 값이 커지면서 다른 뉴런들의 학습이 제대로 되지 않는 현상을 말한다. 하지만 dropout으로 일부만 학습시키더라도 이 과정을 무작위로 반복하면 평균의 효과를 얻기 때문에 특정 뉴런의 w 웨이트 값이 커지지 않도록 해준다.



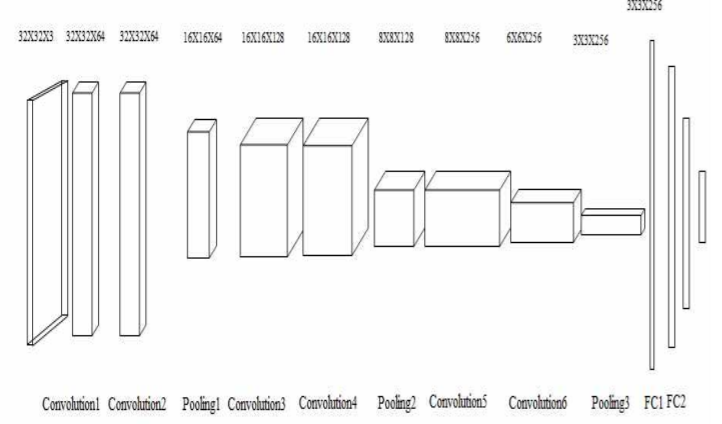


Figure 2 Test Accuracy

위의 그래프 들은 각 실험대상의 정확도/10을 보여준다. Cifar10은 70%의 정확도를 보였다. 생각보다 낮은 정확도 이지만 Epoch 가 40번뿐 이었다는 점이 낮은 정확도를 야기한 것으로 판단된다. 학습할 기회가 적었기 때문이다.

Figure 3. Training Accuracy

Figure 4. Pooling layer

<http://solarisailab.com/archives/2325>

<https://bi.snu.ac.kr/Courses/ML2016/LectureNote/LectureNote_ch4.pdf>

Relu는 활성화함수를 hyperbolic tangent, logistic fuction 대신에 Rectifier 을 적용한 뉴런이며 Rectified linear unit 과 Drop out 을 적용하여 신경망의 성능을 더욱 높일 수 있다. Relu를 적용함으로써 학습 과정에서 문제가 되던 Vanishing gradient problem 이 완화되었다. 또한 dropout은 학습 시 임의의 확률로 각 뉴런의 출력을 강제로 0으로 만듦으로서 학습과 추론 과정에서 활성화되는 뉴런을 다르게 한다. 드롭아웃은 뉴런들의 co adaptation을 방지하여 모델이 이미 주어진 데이터만을 잘 설명하고 새로운 데이터를 설명하지 못하는 현상(overfitting)을 방지하는데 도움을 준다.

여러 Activation function을살펴본 결과, 가장 먼저 Relu를 사용하고 변형된 버전인 Leaky relu, ELU를 사용하는 것이 바람직 하다고 판단했다.