CNN Basic

AI – 스쿨

2022. 06. 08.

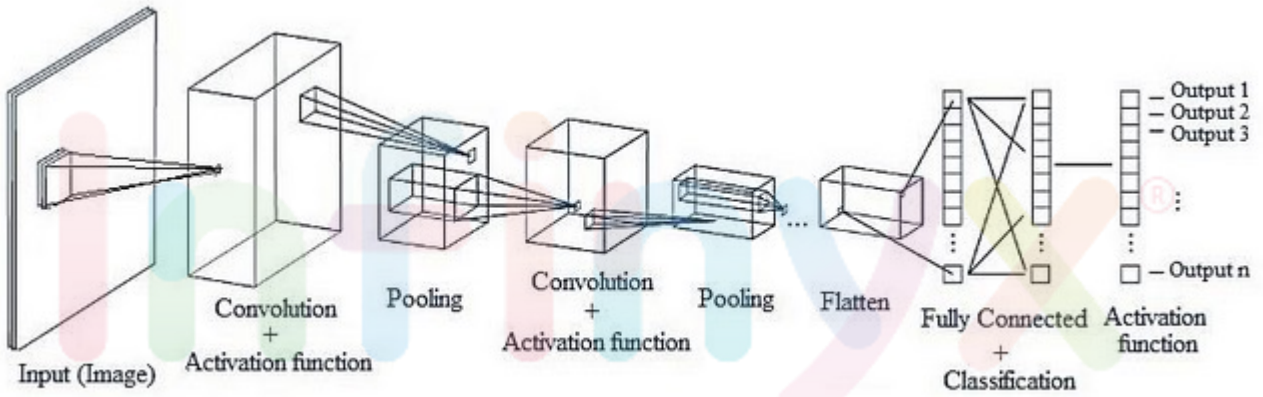
Infinyx®

I

CNN Basic

1. CNN 전체적인 네트워크 구조

CNN 전체적인 네트워크 구조

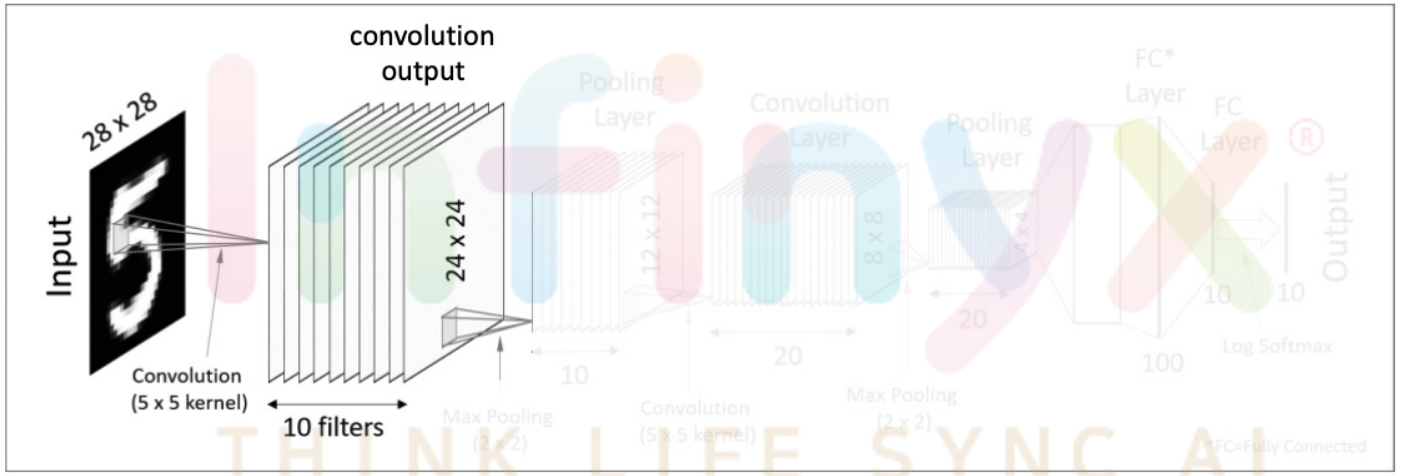


CNN의 구조는 기존의 완전연결계층(Fully-Connected Layer)과는 다르게 구성되어 있습니다.
완전연결계층(또는 Dense Layer이라고도 합니다)에서는 이전 계층의 모든 뉴런과 결합 되어있는 Affine계층으로 구현했지만, CNN는 *Convolutional Layer*과 *Pooling Layer*들을 활성화 함수 앞뒤에 배치하여 만들어집니다.

이렇게 설명하면 너무 추상적 이조 ? 순서대로 각 계층을 살펴 볼게요 !!

CNN 전체적인 네트워크 구조

1. 첫번째 Convolutional Layer



우선 우리에게 주어진 입력값은 28x28크기를 가진 이미지입니다. 이 이미지를 대상으로 여러개의 필터(커널)을 사용하여 결과값(feature mapping이라고도 합니다)을 얻습니다. 즉 한개의 28x28 이미지 입력값에 10개의 5x5필터를 사용하여 10의 24x24 matrics, 즉 convolution 결과값을 만들어 냈습니다. 그 후 이렇게 도출해낸 결과값에 Activation function(예를 들면 ReLU function)을 적용합니다. 이렇게 첫번째 Convolutional Layer이 완성되었습니다. 우리는 여기서 한 Convolutional Layer는 Convolution처리와 Activation function으로 구성되어 있다는 것을 알 수 있습니다.

A Convolutional Layer = convolution + activation

CNN 전체적인 네트워크 구조

1. 잠깐 ? 왜 활성함수(Activation function)을 쓰고 이건 무엇일까 ?

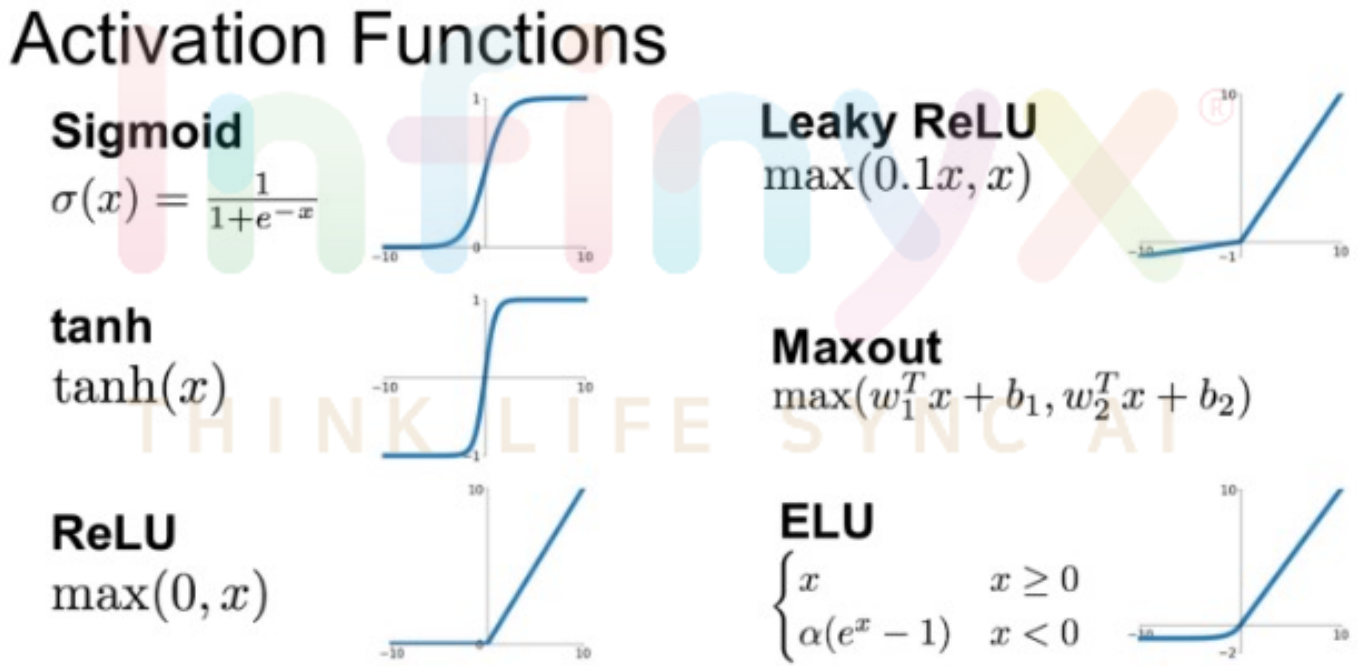
간단히 말하자면 선형함수(linear function)인 convolution에 비선형성(nonlinearity)를 추가하기 위해 사용하는 것입니다.

* 따라서 MLP(Multiple layer perceptron)는 단지 linear layer를 여러개 쌓는 개념이 아닌 활성화 함수를 이용한 non-linear 시스템을 여러 layer로 쌓는 개념이다.

- https://medium.com/@snaily16/what-why-and-which-activation-functions-b2bf748c0441

CNN 전체적인 네트워크 구조

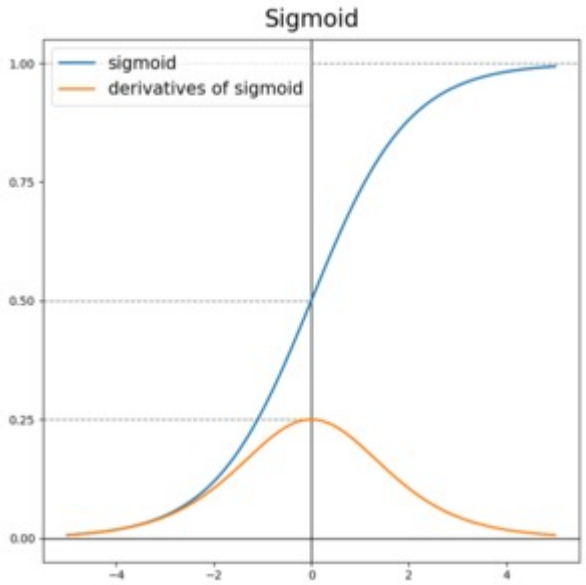
1. 활성화함수(Activation Function)



활성화함수를 사용하면 왜 입력값에 대한 출력값을 비선형으로 만들 수 있는지는 함수의 생김새만 봐도 명확하다.

CNN 전체적인 네트워크 구조

1. 활성화함수(Activation Function) - 시그모이드



Sigmoid

- $a = \frac{1}{1+e^{-z}}$

output값을 0에서 1사이로 만들어준다. 데이터의 평균은 0.5를 갖게된다.

그림에서 시그모이드 함수의 기울기를 보면 알 수 있듯이 input 값이 어느정도 크거나 작으면 기울기가 아주 작아진다.

이로인해 생기는 문제점은 vanishing gradient현상이 있다.

Vanishing gradient
이렇게 시그모이드로 여러 layer를 쌓았다고 가정하자. 그러면 출력층에서 멀어질수록 기울기가 거의 0인 몇몇 노드에 의해서 점점 역전파해갈수록, 즉 입력층 쪽으로갈수록 대부분의 노드에서 기울기가 0이되어 결국 gradient가 거의 완전히 사라지고만다. 결국 입력층쪽 노드들은 기울기가 사라지므로 학습이 되지 않게 된다.

CNN 전체적인 네트워크 구조

1. 활성화함수(Activation Function) - 시그모이드

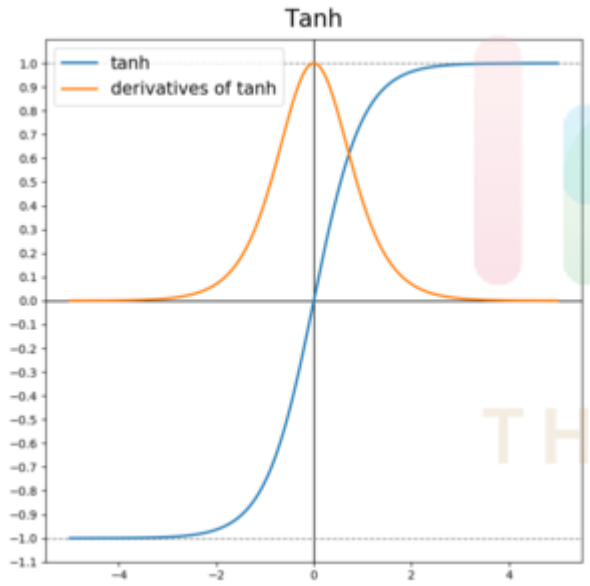
시그모이드를 사용하는 경우
대부분의 경우에서 시그모이드함수는 좋지 않기때문에 사용하지 않는다. 그러나 유일한 예외가 있는데

binary classification경우 출력층 노드가 1개이므로 이 노드에서 0~1사이의 값을 가져야 마지막에 cast를 통해(ex. 0.5이상이면 1, 미만이면 0) 1혹은 0값을 output으로 받을 수 있다.
따라서 이때는 시그모이드를 사용한다.

요약
장점) binary classification의 출력층 노드에서 0~1사이의 값을 만들고 싶을때 사용한다.
단점) Vanishing gradient - input값이 너무 크거나 작아지면 기울기가 거의 0이된다.

CNN 전체적인 네트워크 구조

1. 활성화함수(Activation Function) –Thah(Hyperbolic Tangent)-하이퍼볼릭 탄젠트



Tanh

- $a = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

그림에서 보면 알 수 있듯이 시그모이드 함수와 거의 유사하다. 차이는 -1~1값을 가지고 데이터의 평균이 0이라는 점이다.

데이터의 평균이 0.5가 아닌 0이라는 유일한 차이밖에 없지만 대부분의 경우에서 시그모이드보다 Tanh가 성능이 더 좋다.

그러나 시그모이드와 마찬가지로 Vanishing gradient라는 단점이 있다.

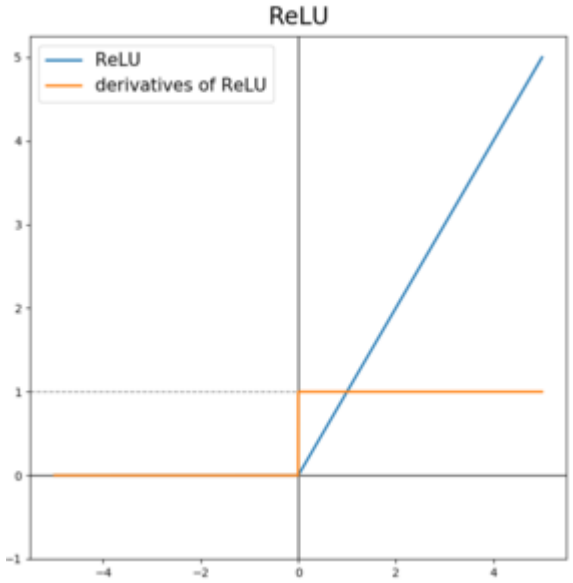
결론

장점) output데이터의 평균이 0으로써 시그모이드보다 대부분의 경우에서 학습이 더 잘 된다.

단점) 시그모이드와 마찬가지로 Vanishing gradient현상이 일어난다.

CNN 전체적인 네트워크 구조

1. 활성화함수(Activation Function) –ReLU



ReLU

- $a = max(0, z)$

대부분의 경우 일반적으로 ReLU의 성능이 가장 좋기때문에 ReLU를 사용한다.

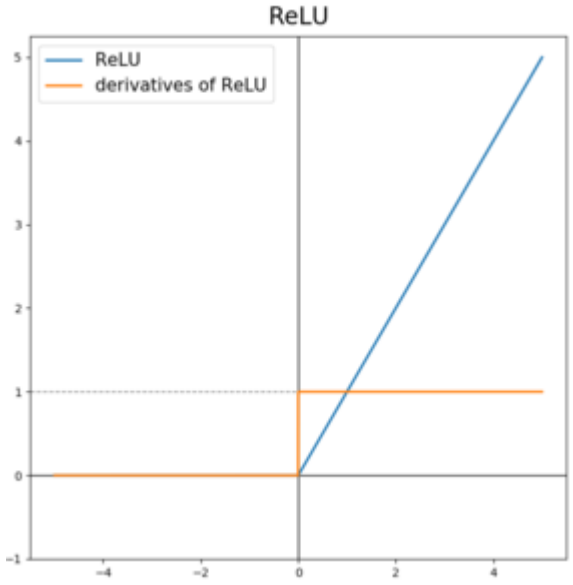
"Hidden layer에서 어떤 활성화 함수를 사용할지 모르겠으면 ReLU를 사용하면 된다" - Andrew ng 앤드류 응

상식

요슈아 벤지오, 얀 르쿤, 제프리 힌튼과 더불어 세계 4대 AI 석학으로 꼽히는 앤드류 응 스탠퍼드 교수가 데이터 품질 개선을 위한 캠페인을 실시한다. 그가 설립한 스타트업 '랜딩AI(Landing.AI)'

CNN 전체적인 네트워크 구조

1. 활성화함수(Activation Function) –ReLU



ReLU

- $a = max(0, z)$

대부분의 input값에 대해 기울기가 0이 아니기 때문에 학습이 빨리 된다. 학습을 느리게하는 원인이 gradient가 0이 되는 것인데 이를 대부분의 경우에서 막아주기 때문에 시그모이드, Tanh같은 함수보다 학습이 빠르다.

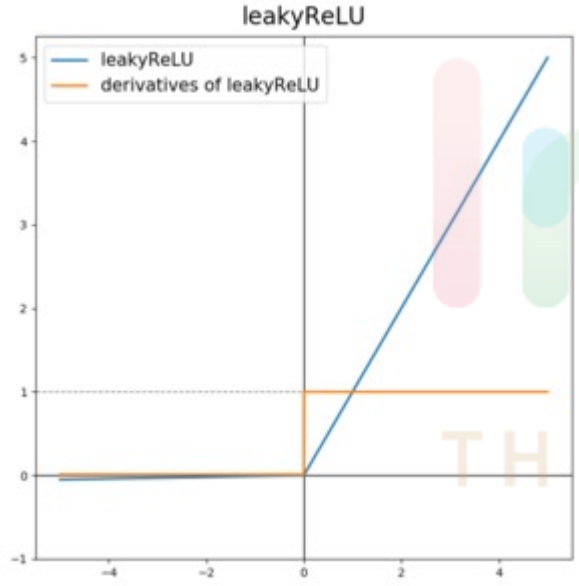
그림을 보면 input이 0보다 작을 경우 기울기가 0이기 때문에 대부분의 경우에서 기울기가 0이 되는것을 막아주는게 납득이 안 될수 있지만 실제로 hidden layer에서 대부분 노드의 z값은 0보다 크기 때문에 기울기가 0이 되는 경우가 많지 않다.

단점으로는 위에서 언급했듯이 z가 음수일때 기울기가 0이라는 것이지만 실제로는 거의 무시할 수 있는 수준으로 학습이 잘 되기 때문에 단점이라 할 수도 없다.

장점) 대부분의 경우에서 기울기가 0이 되는 것을 막아주기 때문에 학습이 아주 빠르게 잘 되다.
hidden layer에서 활성화 함수 뭐 써야할지 모르겠으면 그냥 ReLU를 쓰면 된다.

CNN 전체적인 네트워크 구조

1. 활성화함수(Activation Function) – leaky ReLU



leaky ReLU

- $a = max(0.01z, z)$

ReLU와 유일한 차이점으로는 max(0, z)가 아닌 max(0.01z, z)라는 점이다.

즉, input값인 z가 음수일 경우 기울기가 0이 아닌 0.01값을 갖게 된다.

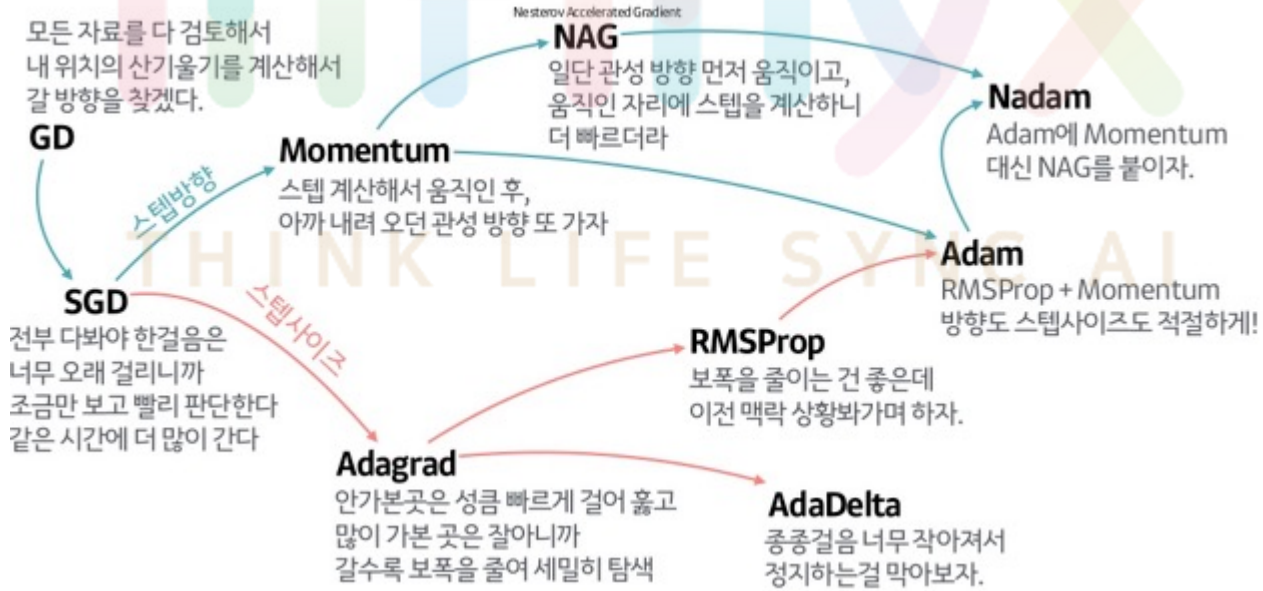
leaky ReLU를 일반적으로 많이 쓰진 않지만 ReLU보다 학습이 더 잘 되긴 한다.

결론
장점) z가 음수일때 기울기가 0이 아닌 0.01을 갖게 하므로 ReLU보다 학습이 더 잘 된다.

CNN 전체적인 네트워크 구조

1. Optimizer 종류 및 정리

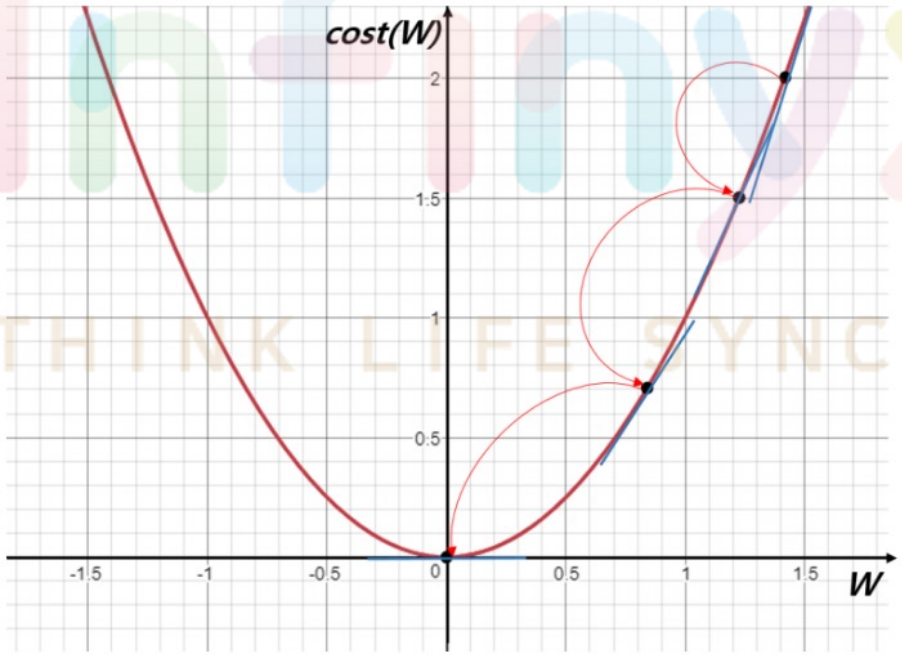
옵티마이저는 학습 데이터(Train data)셋을 이용하여 모델을 학습 할 때 데이터의 실제 결과와 모델이 예측한 결과를 기반으로 잘 줄일 수 있게 만들어주는 역할을 한다.



CNN 전체적인 네트워크 구조

1. Optimizer – Gradient descent(GD)

가장 기본이 되는 optimizer 알고리즘으로 경사를 따라 내려가면서 W를 update시킨다.



$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

CNN 전체적인 네트워크 구조

1. Optimizer – Gradient descent(GD)

GD 를 사용하는 이유 ?

왜 이렇게 기울기를 사용하여 step별로 update를 시키는것일까?
애초에 cost(W)식을 미분하여 0인 점을 찾으면 되는게 아닌가??
-> 두 가지 이유 때문에 closed form(닫힌형식)으로 해결하지 못한다고 알고 있다.

1. 대부분의 non-linear regression문제는 closed form solution이 존재하지 않다.

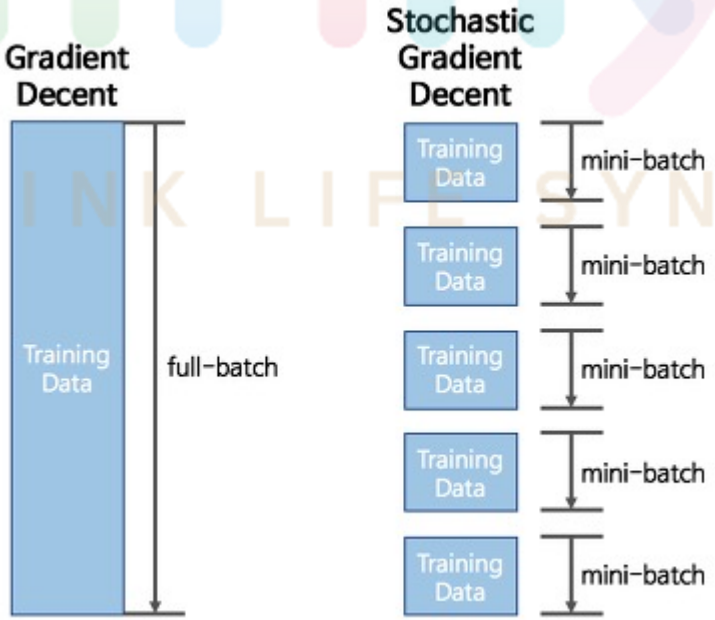
2. closed form solution이 존재해도 수많은 parameter가 있을때는 GD로 해결하는 것이 계산적으로도 더 효율적이다.

CNN 전체적인 네트워크 구조

1. Optimizer – Stochastic gradient decent(SGD)

full-batch가 아닌 mini batch로 학습을 진행하는 것 -> 내일 PPT 추가

(* batch로 학습하는 이유 : full-batch로 epoch마다 weight를 수정하지 않고 빠르게 mini-batch로 weight를 수정하면서 학습하기 위해)

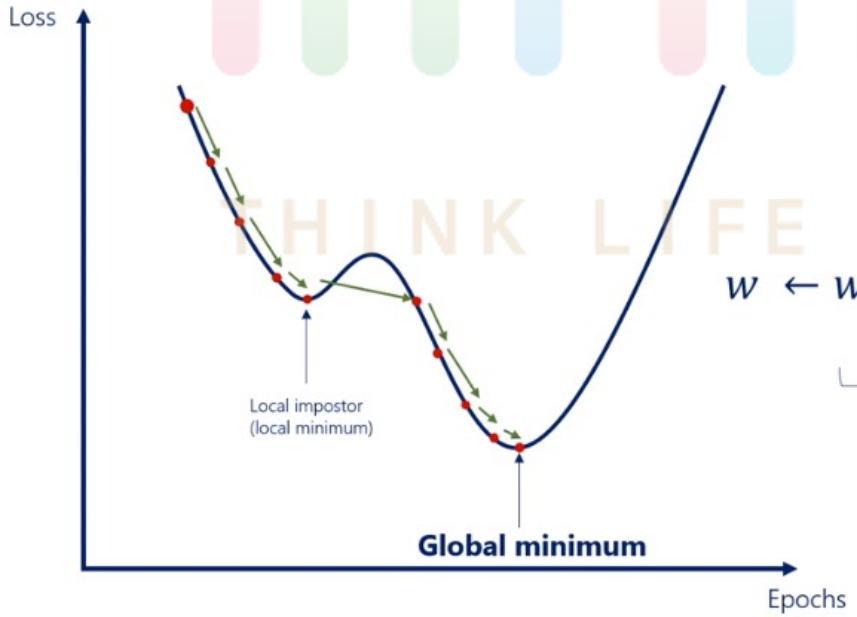


CNN 전체적인 네트워크 구조

1. Optimizer – Stochastic gradient decent(SGD)

Momentum 개념

SGD에 momentum개념을 추가한 것이다.
현재 batch로만 학습하는 것이 아니라 이전의 batch 학습결과도 반영한다.



$$w \leftarrow w(t) - \eta \frac{\partial L}{\partial w}(t) - \eta \frac{\partial L}{\partial w}(t-1)$$

Current update

Update a moment ago

CNN 전체적인 네트워크 구조

1. Optimizer – Stochastic gradient decent(SGD)

Momentum 개념

모멘텀이란 무엇입니까? 모멘텀을 설명하는 쉬운 방법은 물리학적 비유를 통한 것입니다. 작은 언덕은 볼이 멈추지 않는 평평한 표면에 도달 할 때까지 롤링을 계속하지 않을 것입니다. 작은 언덕은 지역 최소값이고, 큰 계곡은 전역 최소값입니다.

운동량이 없으면 공은 원하는 최종 목적지에 도달하지 못합니다. 속도가 전혀 떨어지지 않고 작은 언덕에서 멈추었을 것입니다. 지금까지 모멘텀을 고려하지 않았습니다. 즉, 작은 언덕에 빠질 가능성이있는 알고리즘을 만들었습니다.

그래서 알고리즘에 운동량을 추가하는 방법에 대해 알아보아야 합니다.

규칙은 운동량을 포함한 w에 손실의 기울기를 활용합니다.

지금까지 내려간 속도를 고려할 것입니다. 공이 빨리 굴러가는 경우 운동량은 높고 그렇지 않으면 운동량이 낮습니다. 볼이 얼마나 빨리 움직이는 지 알아내는 가장 좋은 방법은 전에 공이 얼마나 빨리 굴렀는지를 확인하는 것입니다. 이는 머신 러닝에 채택 된 방법이기도합니다.

CNN 전체적인 네트워크 구조

1. Optimizer – AdaGrad

학습을 통해 크게 변동이 있었던 가중치에 대해서는 학습률을 감소시키고 학습을 통해 아직 가중치의 변동이 별로 없었던 가중치는 학습률을 증가시켜서 학습이 되게끔 한다.

$$h \leftarrow h + \frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}$$

$$W \leftarrow W - \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W}$$

기존 SGD에서의 notation에서 h가 추가되었는데 h는 가중치 기울기 제곱들을 더해간다.
따라서 가중치 값에 많은 변동이 있었던 가중치는 점점 학습률을 감소시키게 된다.
(* AdaGrad는 무한히 학습하면 어느 순간 h가 너무 커져서 학습이 아예 안될 수 있다. 이를 RMSProp에서 개선한다.)

CNN 전체적인 네트워크 구조

1. Optimizer – RMSProp

AdaGrad는 간단한 convex function에서 잘 동작하지만, 복잡한 다차원 곡면 function에서는 global minimum에 도달하기 전에 학습률이 0에 수렴할 수 있다. 따라서 RMSProp에서는 이를 보완하였다.

a) 가중치 기울기를 단순 누적시키는 것이 아니라 최신 기울기들이 더 반영되도록 한다.

$$h_i \leftarrow \rho h_{i-1} + (1-\rho)\frac{\partial L_i}{\partial W} \odot \frac{\partial L_i}{\partial W}$$

AdaGrad의 h에 hyper parameter p가 추가되었다. p가 작을수록 가장 최신의 기울기를 더 크게 반영한다. Momentum에서와 반대로 이번 batch에 의한 최신 기울기가 크게 반영되도록 한다. (누적 h보다 최신 기울기에 초점)

즉, 그동안 가중치에 큰 변동사항이 있었어도 이번 batch로 인한 기울기가 완만하면 학습률을 높인다.

CNN 전체적인 네트워크 구조

1. Optimizer – RMSProp

-> (minimum과 같은 극점 근처에서 학습 속도가 느려지는 문제와 local minimum에 수렴하는 문제를 해결)
* 최신 batch가 이전 batch중 특정 1개보다 더 크게 반영된다는 점은 같다.
(Momentum에서는 이번 batch에 의해 크게 좌지우지 되지 않도록 한다. / 지금까지 누적된 momentum에 초점)

b) hyper parameter p를 추가하여 h가 무한히 커지지 않게 한다.
p*h + (1-p)g^2 연산을 하므로 h가 가질 수 있는 값이 max{g^2} 으로 bounded된다는 것을 알 수 있다.

CNN 전체적인 네트워크 구조

1. Optimizer – Adam

Momentum과 RMSProp를 융합한 방법이다. 먼저 Adam optimizer의 수도코드는 아래와 같다.

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

https://arxiv.org/pdf/1412.6980.pdf

CNN 전체적인 네트워크 구조

1. Optimizer – Adam

Momentum과 Adagrad는 각각 v와 h가 처음에 0으로 초기화되면 W가 학습 초반에 0으로 biased되는 문제가 있다.

$$v \leftarrow \alpha v - \eta \frac{\partial L}{\partial W}$$
$$W \leftarrow W + v$$

Momentum notation1

$$h_i \leftarrow \rho h_{i-1} + (1-\rho)\frac{\partial L_i}{\partial W} \odot \frac{\partial L_i}{\partial W}$$

RMSProp notation

이런 잘못된 초기화로 인해 0으로 biased되는 문제를 해결하기 위해 Adam이 고안되었다.
위 수도코드에서 (**m**=Momentum에서의 v, **v**=RMSProp에서의 h, **g**=W의 기울기) 라고 보면된다.

CNN 전체적인 네트워크 구조

1. Optimizer – Adam

아래 예시들을 보면서 어떻게 Adam이 위 문제들을 해결했는지 보자.

$$m_1 \leftarrow \beta_1 m_0 + (1-\beta_1) g_1$$

$$\widehat{m_1} \leftarrow \frac{m_1}{1-\beta_1^1} = \frac{\beta_1 m_0}{1-\beta_1^1} + \frac{(1-\beta_1) g_1}{1-\beta_1^1}$$

$$= 0 + g_1 (\because m_0 = 0)$$

기존 Momentum에서는 m0가 0으로 초기화되면 m1=0.1g1(∵β1 = 0.9) 으로써 0으로 biased된다.

그러나 Adam에서는 m/(1-β)로 m을 update하기 때문에 m1=g1이 되어 0으로 biased(편향)됨을 해결하였다.

CNN 전체적인 네트워크 구조

1. Optimizer – Adam

$$m_2 \leftarrow \beta_1 m_1 + (1-\beta_1) g_2$$

$$\widehat{m_2} \leftarrow \frac{m_2}{1-\beta_1^2} = \frac{\beta_1 m_1}{1-\beta_1^2} + \frac{(1-\beta_1) g_2}{1-\beta_1^2}$$

$$= 4.73 m_1 + 0.52 g_2 (\because \beta_1 = 0.9)$$

또한 위 m2가 update되는 식을 보면 알 수 있듯이 최신 기울기 한 개에 의해 update가 좌지우지 되지 않도록 하여 Momentum의 성질을 그대로 따르고 있다.

위 수도코드를 보면 RMSProp 방식인 **v** parameter 역시 동일한 방식으로 0으로 biased됨을 해결하고 있다.

CNN 전체적인 네트워크 구조

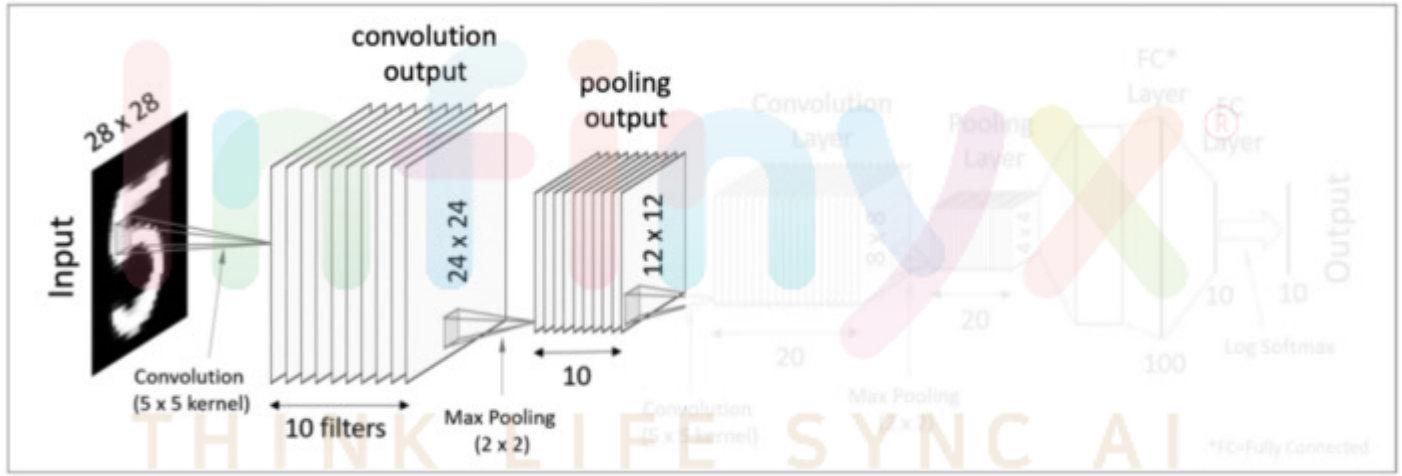
1. Optimizer – Adam

$$\theta_t \leftarrow \theta_{t-1} - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$$

마지막 실질적인 **W** update 식을 보면 RMSProp에서와 같이 v를 루트를씌워서 나눠주고 거기에 Momentum을 곱하는 형식으로 update하고 있다.

CNN 전체적인 네트워크 구조

2. Pooling Layer



그 다음은 Pooling Layer입니다. 우선 **Pooling**이라는 개념이 무엇인지 짚고 넘어가야 합니다.

CNN 전체적인 네트워크 구조

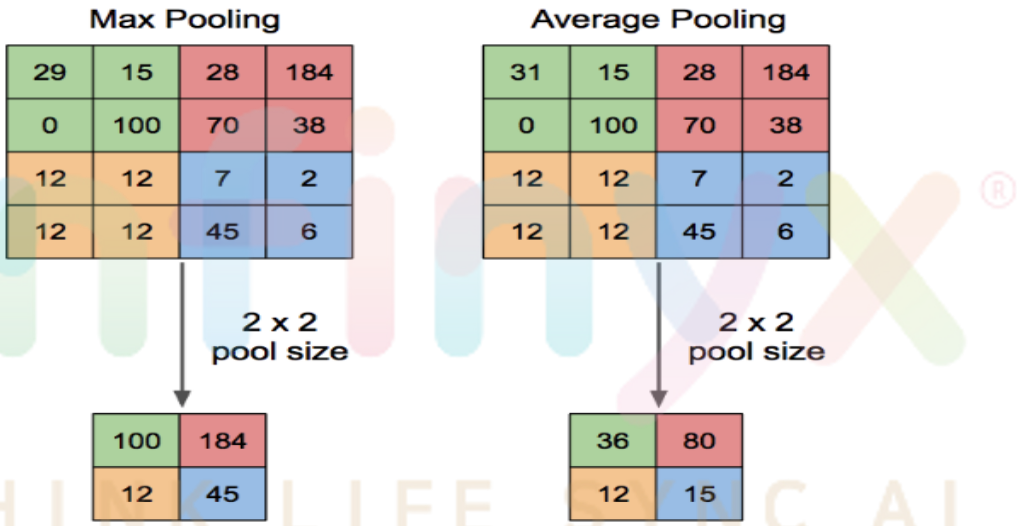
2. Pooling Layer

이 전 단계에서 convolution 과정을 통해 많은 수의 결과값(이미지)들을 생성했습니다. 하지만 위와 같이 한 개의 이미지에서 10개의 이미지 결과값이 도출되어버리면 값이 너무 많아졌다는 것이 문제가 됩니다.

때문에 고안된 방법이 Pooling이라는 과정입니다. *Pooling은 각 결과값(feature map)의 dimentionality를 축소해 주는 것을 목적으로* 둡니다. 즉 correlation이 낮은 부분을 삭제(?)하여 각 결과값을 크기(dimension)을 줄이는 과정입니다.

CNN 전체적인 네트워크 구조

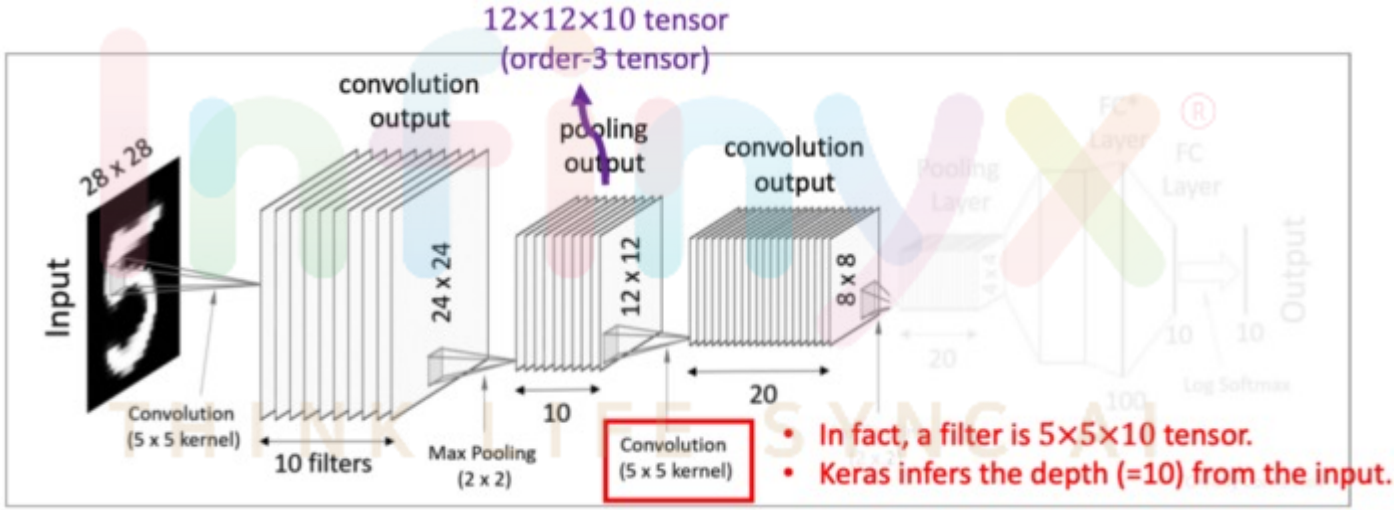
2. Pooling Layer



Pooling에는 대표적으로 두가지 방법으로 Max pooling과 Average pooling이 있습니다. 위 이미지와 같이 Pool의 크기가 2x2인 경우 2x2크기의 matrix에서 가장 큰 값(max)이나 평균값(average)를 가져와 결과값의 크기를 반으로 줄여주게 됩니다.

CNN 전체적인 네트워크 구조

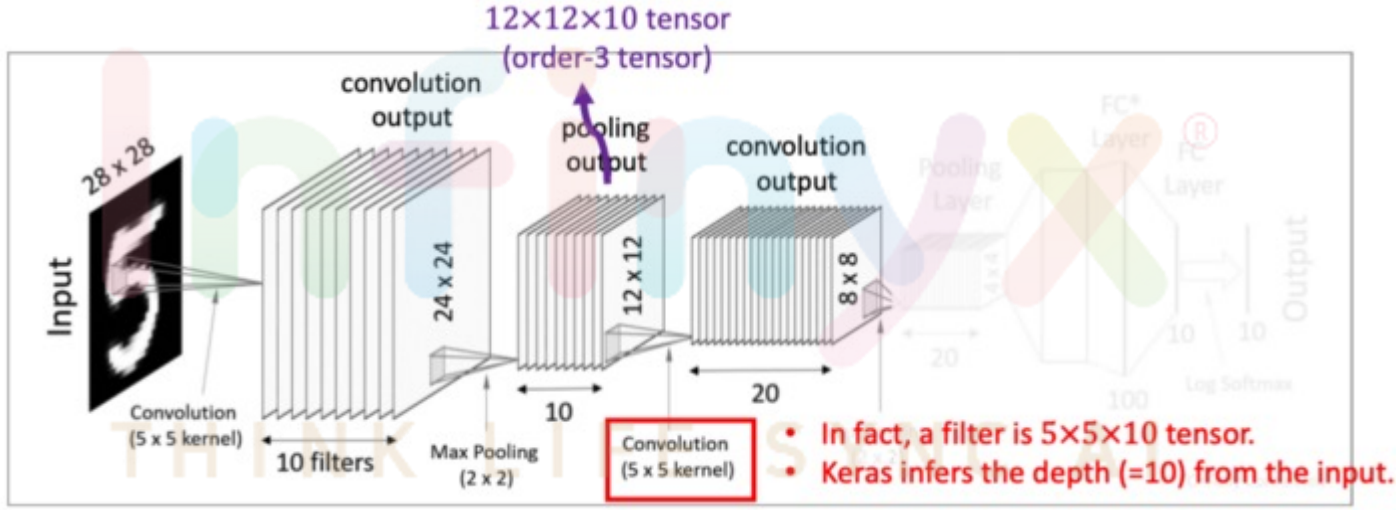
2. 첫번째 Pooling Layer



자, 이제 다시 본론으로 돌아가 그 위의 전체 네트워크 이미지로 돌아가면 Pooling Layer의 결과값이 열개의 12x12 matrics가 된 것을 확인할 수 있습니다.

CNN 전체적인 네트워크 구조

2. 두번째 Convolutional Layer

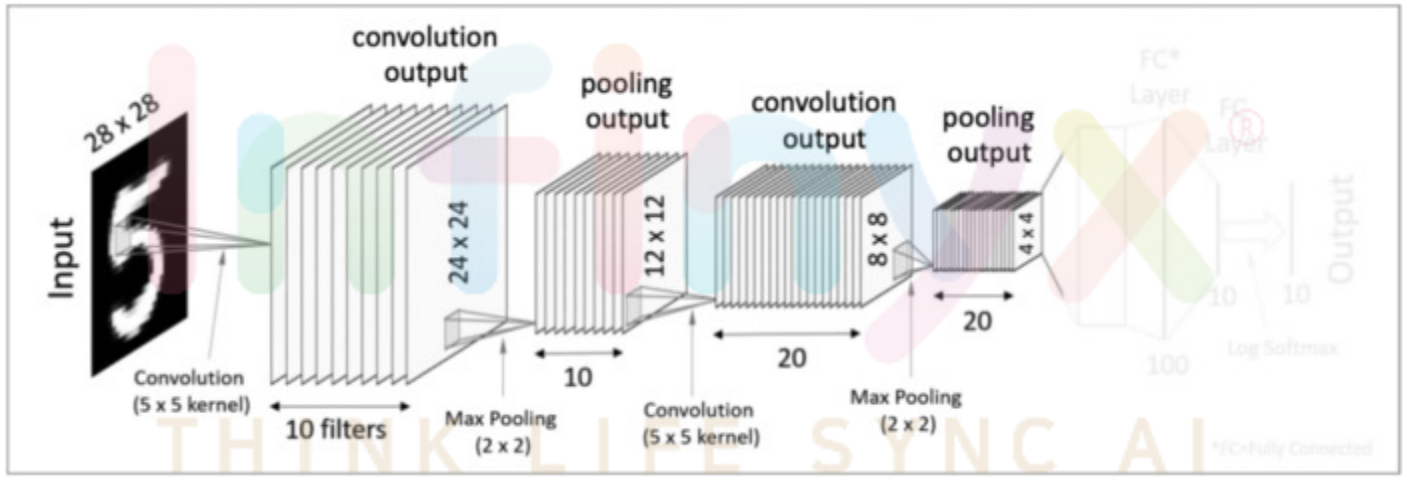


이번 Convolutional layer에서는 텐서 convolution을 적용합니다.

이전의 pooling layer에서 얻어낸 12x12x10 텐서(order-3 tensor)를 대상으로 5x5x10크기의 텐서필터 20개를 사용해 줍니다. 그렇게 되면 각각 8x8크기를 가진 결과값 20개를 얻어낼 수 있습니다.

CNN 전체적인 네트워크 구조

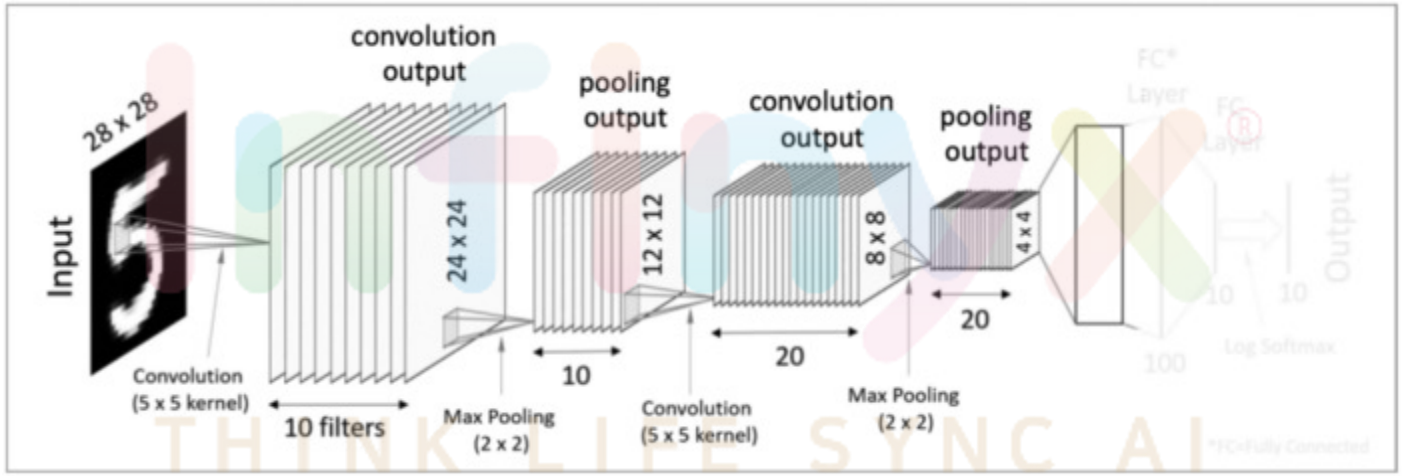
2. 두번째 Pooling Layer



두번째 Pooling layer입니다. 전과 똑같은 방식으로 Pooling과정을 처리해주면 더 크기가 작아진 20개의 4x4 결과값을 얻습니다.

CNN 전체적인 네트워크 구조

2. Flatten(Vectorization)

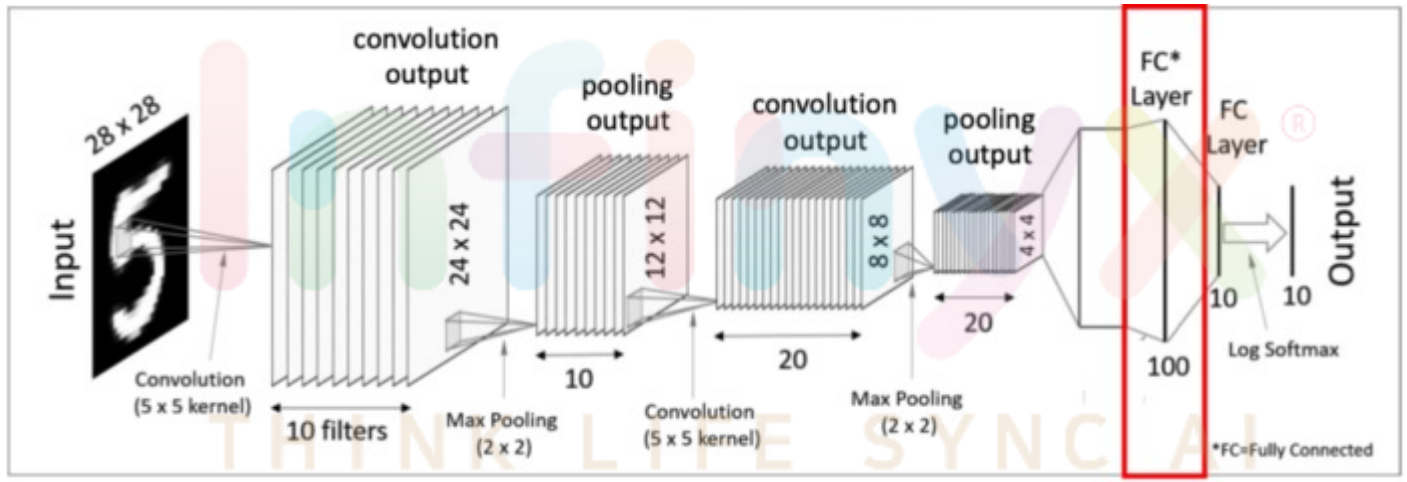


그 후 이 4x4x20의 텐서를 일자 형태의 데이터로 쭉 펼쳐준다고 생각해 봅시다. 이 과정을 *Flatten* 또는 *Vectorization* 이라고 합니다. 쉽게 말해 각 세로줄을 일렬로 쭉 세워두는 것입니다. 그러면 이것은 320-dimension을 가진 벡터(vector)형태가 됩니다.

그렇다면 **왜** 이렇게 1차원 데이터로 변형해도 상관이 없을까요? 이 전에 두번째 pooling layer에서 얻어낸 4x4크기의 이미지들은 이미지 자체라기 보다는 입력된 이미지에서 얻어온 특이점 데이터가 됩니다. 즉 1차원의 벡터 데이터로 변형시켜주어도 무관한 상태가 된다는 의미입니다.

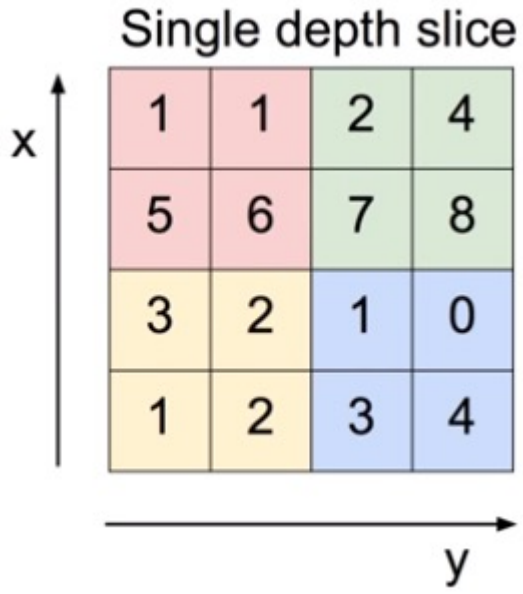
CNN 전체적인 네트워크 구조

2. Fully Connected Layers(Dense Layers)



이제 마지막으로 하나 혹은 하나 이상의 Fully-Connected Layer를 적용시키고 마지막에 Softmax activation function을 적용해주면 드디어 최종 결과물을 출력하게 됩니다.

과제 – MAX Pool and Average Pool 계산



조건 : 2x2 filters and stride 2

Max Pool and Average Pool 손으로 계산하여 제출

제출 시간 6/8일 오후 6시까지


감사합니다.
THINK LIFE SYNC AI
Infinyx