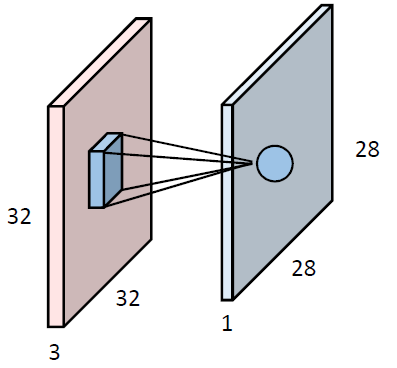
컴퓨터비전특론 5th week summary

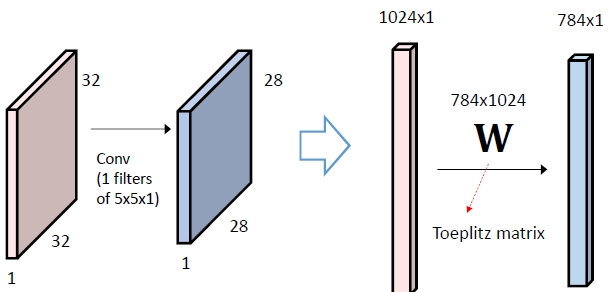
[Lec06: Convolutional Neural Networks] 이어서

**The brain/neuron view of CONV Layer**

32 x 32 image가 convolution layer를 거쳐 28 x 28 feature map을 output으로 낼 때, feature map을 출력하는 하나의 neuron은 filter와 곱해지는 input image의 일부를 가지고만 만들어진다. 또한 feature map의 모든 neuron은 같은 파라미터를 공유한다.

만약 5개의 filter가 사용된다면 input image의 한 부분에 5개의 다른 필터가 곱해져 output feature map의 전체 구조는 28 x 28 x 5가 된다.

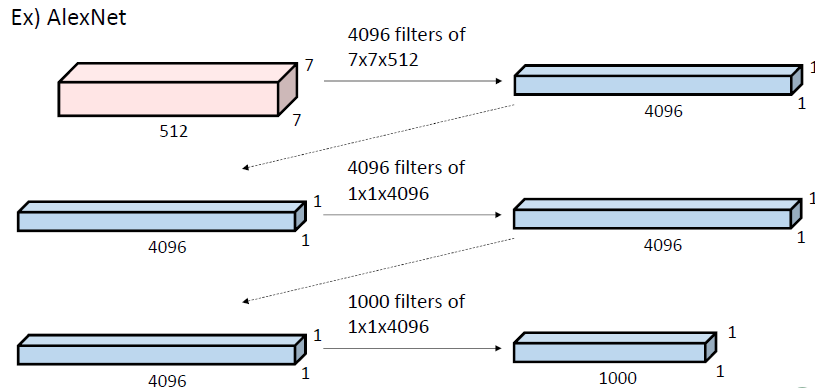
**FC Layer vs Convolutional Layer**

- Conv Layer implemented as FC Layer

Input과 output은 FC에서 사용하는 vector처럼 width와 height를 곱한 크기의 벡터로 표현한다.

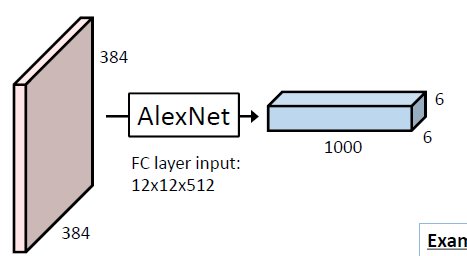
Weight의 경우, 한 행에서 25개의 값(filter가 5x5x1이므로 25개)만 nonzero 값을 가지고 나머지 원소는 0이며, filter weight이 matrix의 한 행을 내려갈 때마다 shift되어 나타내는 형태인 Toeplitz matrix 형태의 weight matrix를 사용한다.

- FC Layer implemented as Conv Layer



FC Layer를 Conv Layer처럼 사용하는 변환은 실제 모델에서도 많이 쓰인다. 그 예로 AlexNet에서는 7x7x512의 input을 받는 모델의 첫번째 FC Layer를 필터 크기가 7인 conv layer로 바꾼다. 7x7 크기의 필터 4096개를 사용하여 output의 크기는 1x1x4096이 된다. 두번째 FC layer에서도 같은 방식을 사용하고, 마지막 layer에서는 필터 개수를 1000개로 하여 1000x1x1의 output이 출력된다. 각 변환에서 FC layer의 가중치가 conv layer의 필터로 변환되는 것과 같다. 이러한 변환을 통해 별도의 이미지 슬라이딩 없이 한 번의 forward pass 만으로 계산을 가능하게 한다.

예를 들어 밑의 예시는 384x384 이미지를 32 stride 간격으로 224x224만큼 잘라 잘려진 각각의 이미지를 conv layer에 적용한 것과 같은 결과를 보여준다. 그러나 original conv layer를 통과했을 때보다 훨씬 계산량이 적다.



**Pooling layer**

주로 feature map의 크기를 줄이는 역할. 주로 한 filter에서 가장 큰 값을 선택해서 가져오는 maxpooling이 많이 쓰인다.

Input size가 , pooling을 위한 window size(=spatial extent)를 , stride를 라고 할 때 pooling을 거친 후의 output size는 다음과 같다.

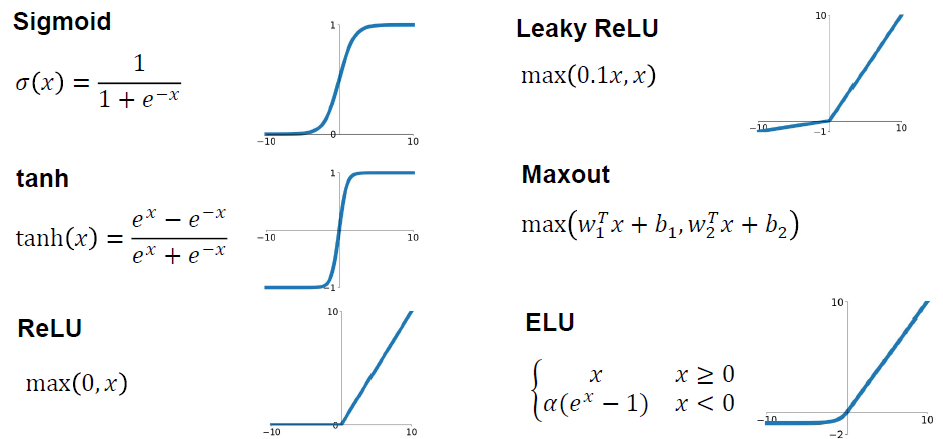
보통 또는 를 사용한다.

Max pooling layer가 추가될 경우, backpropagation 과정에서 highest value만 gradient가 계산되며 그 외의 값들은 gradient가 0이 된다.

최근의 모델들은 pooling layer를 사용하기 보다 stride를 큰 값을 사용함으로써 feature map의 size를 줄이는 것을 더 사용하는 추세다.

[Lec07: Training Neural Networks Part 1]

**Activation Functions**



Sigmoid function

- Input은 , output은 의 범위를 가진다.

- x의 값이 커지면 1 또는 0으로 수렴하는 값을 가져 gradient가 0에 매우 가까운 값을 가지는 gradient vanishing 문제가 발생한다.

-Outgoing gradient를 계산할 때 와 가 곱해지는데, sigmoid function의 output은 0과 1 사이의 값이므로 sigmoid를 반복해서 계산해야 하는 모델 구조라면 결국 gradient가 0에 가까워진다. 따라서 직선에 가까운 부분에 해당하는 x값이 아니더라도 gradient vanishing 문제가 발생한다.

- 또한 출력값의 범위로 인해 평균이 0이 될 수 없다. 값이 모두 양수일 경우 gradient를 계산하면 항상 모두 양수 또는 모두 음수 값이므로 gradient 계산 시 zig zag path를 갖게 된다. 이 경우 두 방향으로만 gradient가 계산된다.

- 이러한 문제점들로 인해 모델 중간에서는 sigmoid 함수를 거의 사용하지 않고, output이 0과 1 사이의 값으로 표현되어야 할 때 모델 구조의 마지막 layer로 주로 사용된다.

Tanh(x) Function

Output의 범위가 로 zero centered지만, -1과 1로 수렴하는 형태를 가져 gradient vanishing 문제가 발생한다. 따라서 잘 사용하지 않는 함수이다.

Rectified Linear Unit (ReLU) function

- 계산의 효율성이 높아 sigmoid와 tanh보다 계산속도가 빠르다.

- 일 때 saturate 되지 않는다.

- zero-centered output이 아니다.

- x가 0 이하면 gradient는 무조건 0으로 계산된다. 이러한 문제를 해결하기 위해 일 때의 값이 0.01이 되도록 사용하는 경우도 있다.

- 가장 많이 사용되는 activation function이다.

Leaky ReLU

- 에서는 0.1x, 일때는 x를 출력하여 saturate되지 않고, ReLU 함수처럼 계산이 효율적이다.

- Zero mean은 아니지만 음수와 양수가 어느정도 섞인 결과를 얻을 수 있어 gradient가 0이 아닌 값을 가질 수 있다.

\* Parametric Rectifier (PReLU) : 를 training 가능하게 하여 에 대한 backprop이 가능함.

Exponential Linear Units (ELU)

- Output이 zero mean에 가까운 평균값을 가지며, x<0에서 수렴하기 때문에 Leaky ReLU보다 noise에 대해 더 강하다.

- 계산식에 exponential이 있어 계산이 오래 걸린다.

Maxout Function

ReLU와 Leaky ReLU의 일반화된 버전이지만, 계산량이 2배여서 잘 사용하지 않는다.

**Data Preprocessing**

Axis-independent normalization

각 변수들의 값의 범위를 scaling하여 같게 하는 역할.

Input data 에 대해

이때 S의 식은 의 모든 component가 독립이라는 가정 하에 성립한다. 독립이 아닐 경우 x의 covariance matrix가 S로 사용된다.

Decorrelation using PCA

Feature들간의 상관관계를 없애는 역할을 한다. (상관관계: 두 변수간의 선형적 관계. 변수 x와 y가 있을 때 x가 증가함에 따라 y도 증가하거나, x가 증가함에 따라 y는 감소하는 경향)

Whitening

Feature들이 동일한 분포를 갖게 만드는 역할을 한다.

**Weight Initialization**

- Weight를 0으로 initialization 할 경우: 모든 gradient 값이 0으로 계산 -> 학습이 되지 않는다.

- Random initialization: 평균이 0, 표준편차가 0.01인 정규분포를 따르는 값으로 random하게 지정한다.

-> layer 계산이 많아질수록 분산이 0이 가까워져서 backpropagation이 불가능 해진다.

- Xavier initialization: 를 표준편차로 사용한다.

-> layer를 통과하기 이전과 통과한 후의 분산이 같도록 계산되지만, 이러한 계산은 activation function은 고려되지 않아 ReLU를 사용하면 문제가 발생한다.

- ReLU initialization: 를 표준편차로 사용한다.