RNN ( Recurrent Neural Network) and CNN ( Convolutional Neural Network)

RNN(Recurrent Neural Networks)은 다른 신경망과 어떻게 다른가?

RNN은 이름에서 알 수 있는 것처럼 신경망(Neural Network)의 일종입니다. 일반적인 신경망을 Feed-forward neural networks(FFNets)라고 하는데요. FFNets에서는 데이터를 트레이닝 셋과 테스트 셋으로 나누어서 관리를 하고 트레이닝 셋을 통해서 신경망의 가중치를 학습시키고 이 결과를 테스트셋을 통해서 확인을 하는 방식입니다.

​

FFNets에서는 데이터를 입력하면 입력층에서 은닉층까지 연산이 차근차근 진행되고 출력이 나가게 됩니다. 이 과정에서 입력 데이터는 모든 노드를 딱 한 번씩만 지나가게 됩니다. 데이터가 노드를 한 번만 지나가게 된다는 것은 데이터의 순서 즉 시간적인 측면을 고려하지 않는 구조라는 의미입니다. 데이터들의 시간 순서를 무시하고 현재 주어진 데이터를 통해서 독립적으로 학습을 합니다.

​

하지만 RNN의 경우는 은닉층의 결과가 다시 같은 은닉층의 입력으로 들어가도록 연결되어 있습니다. 이런 특성이 RNN이 순서 또는 시간이라는 측면을 고려할 수 있는 특징을 가져다주게 됩니다. RNN에서 Recurrent는 '반복되는, 되풀이되는'이라는 의미를 가지고 있는데 이 이름은 은닉층의 결과가 다시 은닉층으로 들어가게 되는 특성에서 나온 듯 합니다.

RNN은 어디서 자주 사용하는가?

이렇게 순서적인 측면을 고려할 수 있는 RNN을 어디에서 사용하면 좋을까요? 순서적인 측면을 고려해서 판단할 수 있다는 특성은 특히 Sequence data를 다루는데 도움이 될 수 있습니다.

​

Sequence data의 대표적인 형태로는 '문장'과 같은 데이터가 있습니다. 문장의 단어 같은 경우 현재의 단어만으로 의미를 해석하는 것이 아니라 앞 단어와의 관계를 통해서 현재 단어의 의미를 해석할 수 있습니다. 하지만 개별 데이터를 독립적으로 학습하는 FFNets의 경우는 Sequence data를 처리하는데 어려움이 있습니다.

​

이외에도 유전자, 손글씨, 음성 신호, 센서가 감지한 데이터, 주가 등의 배열(squence, 또는 시계열 데이터)를 처리하는데 자주 활용된다고 합니다. 사진에 캡션을 달아주는 기능 역시 RNN을 활용할 수 있습니다.

​

심지어는 RNN을 통해서 영화 대본을 짜는 실험도 있었다고 하네요(내용은 이해하기 어렵지만 AI가 만들었다고 하니 뭔가 흥미롭습니다).

<https://www.youtube.com/watch?v=LY7x2Ihqjmc>

[](https://www.youtube.com/watch?v=LY7x2Ihqjmc)

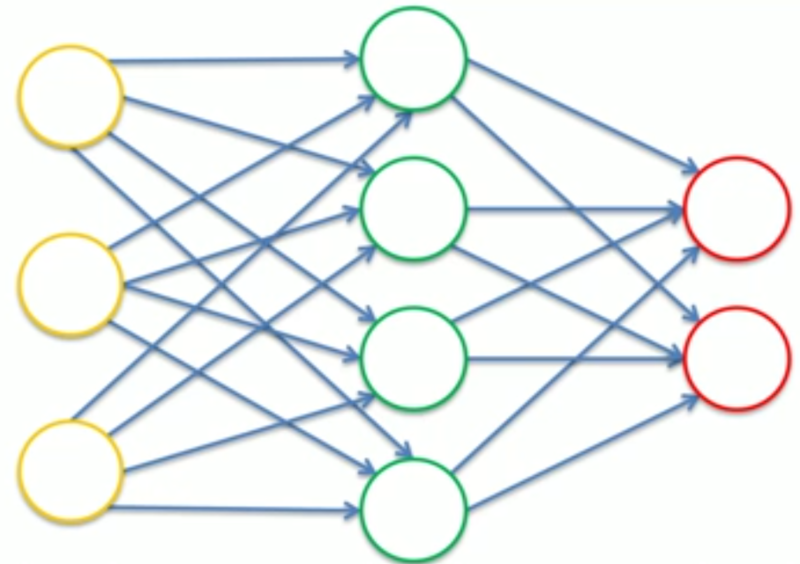
**[Sunspring | A Sci-Fi Short Film Starring Thomas Middleditch](https://www.youtube.com/watch?v=LY7x2Ihqjmc" \t "_blank)**

[In the wake of Google's AI Go victory, filmmaker Oscar Sharp turned to his technologist collaborator Ross Goodwin to build a machine that could write screenp...](https://www.youtube.com/watch?v=LY7x2Ihqjmc" \t "_blank)

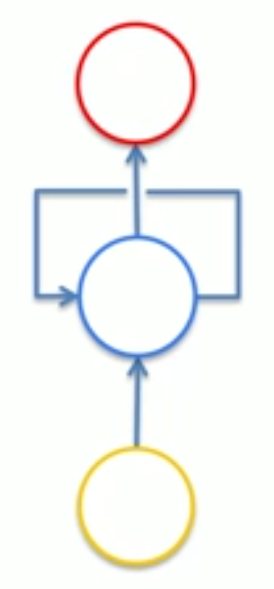
[www.youtube.com](https://www.youtube.com/watch?v=LY7x2Ihqjmc" \t "_blank)

RNN은 어떻게 생겼나?

아래는 일반적인 신경망을 그림으로 표현한 것입니다.

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

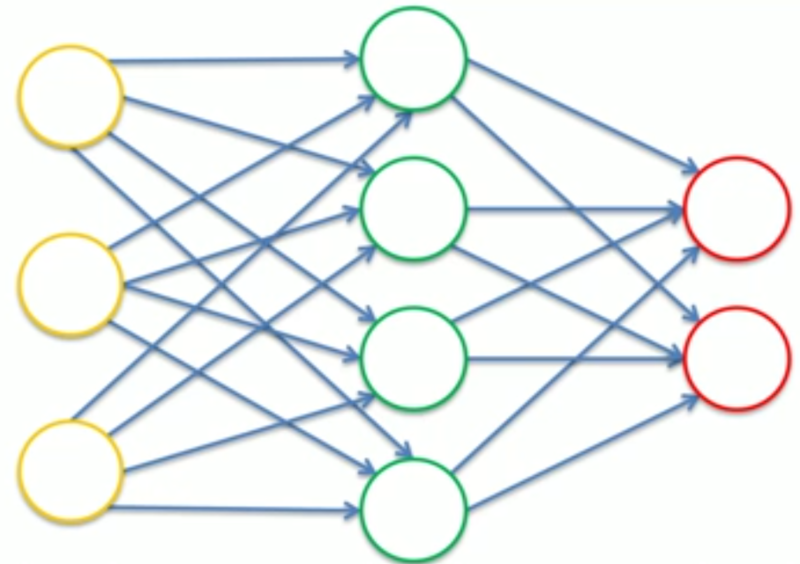
그럼 RNN을 그림으로 표현한 것을 볼까요?

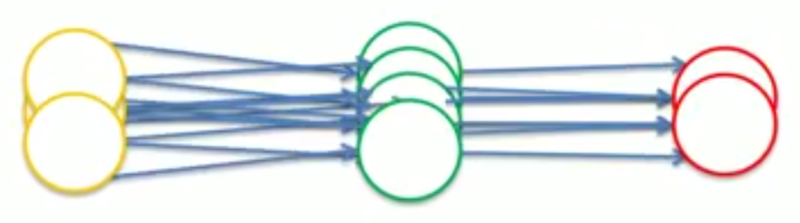
[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

​

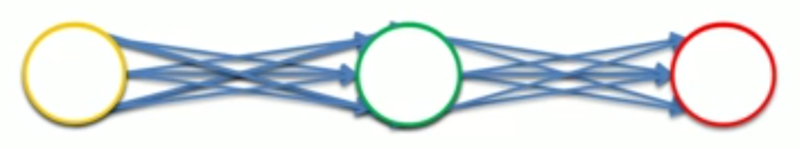
뭔가 굉장히 단순해 졌네요. 하지만 이 그림만으로는 RNN을 직관적으로 이해하기는 어려운 것 같습니다. 그럼 조금 더 자세히 살펴보겠습니다.

​

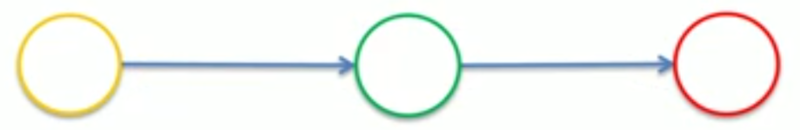
[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

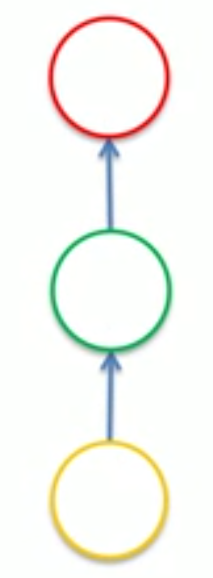
​

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

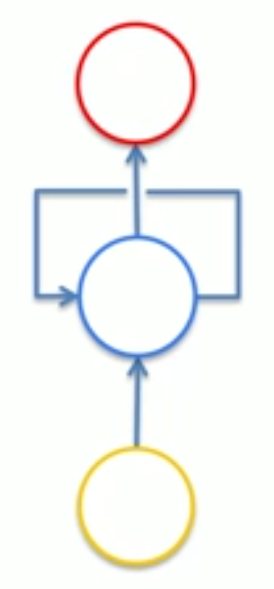
​

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

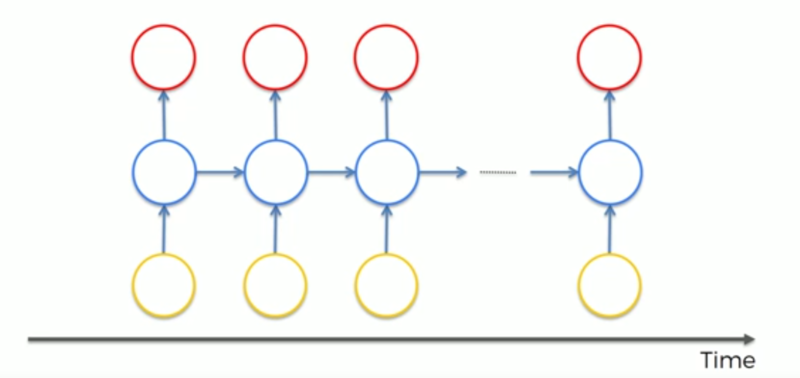
눈치 채셨죠? 하나의 신경망을 단순하게 표현해서 날씬하게 만들었네요. 이제 세웠습니다.

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

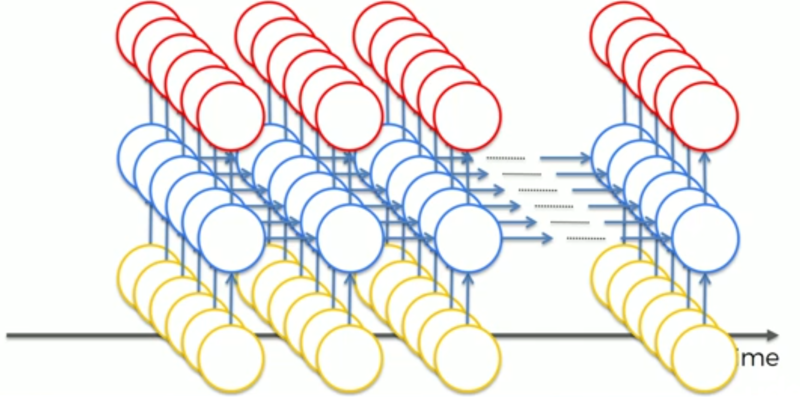
그런데 은닉층을 감싸고 있는 화살표는 무엇을 의미하는 것일까요? 앞에서 RNN의 특성이 은닉층의 출력이 다시 은닉층에 입력되는 구조로 되어 있어서 Recurrent Neural Networks라는 이름이 붙었다고 했었습니다. 은닉층의 출력이 다시 같은 은닉층으로 반복해서(Recurrnt) 입력되는 것을 표현하고 있습니다.

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

이 그림을 시간축을 중심으로 다시 표현하면 다음과 같습니다.

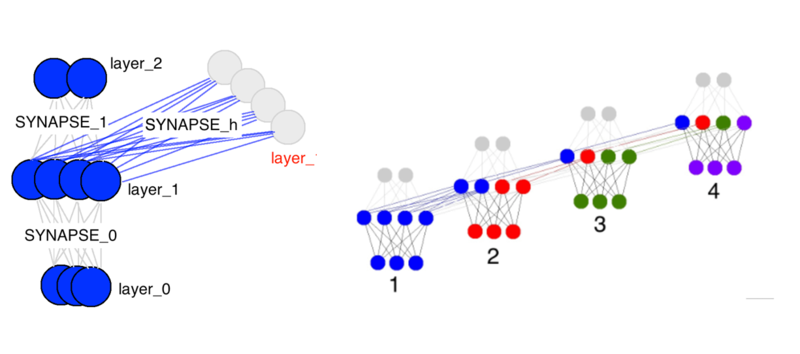
[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

실제로는 이런 느낌인 것을 더 간략하게 표현한 것이겠지요.

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

​

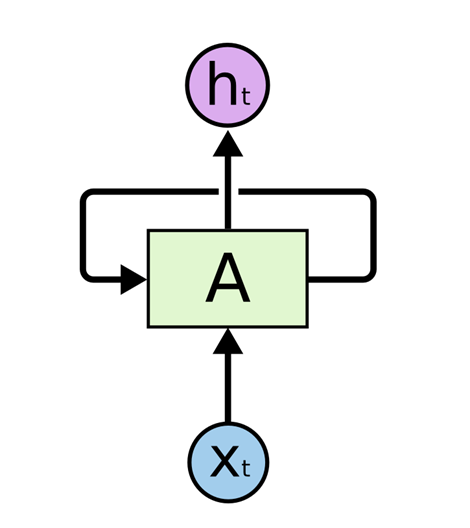
이 그림이 훨씬 더 개념을 잘 이해할 수 있네요.

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

<http://jaejunyoo.blogspot.com/2017/06/anyone-can-learn-to-code-LSTM-RNN-Python.html>

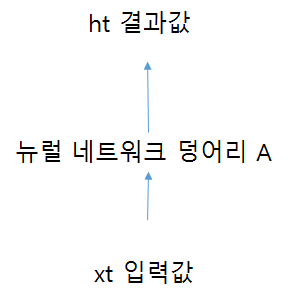
비슷한 다른 그림도 한 번 살펴보겠습니다.

**(아래 내용은 이 링크(**[**http://colah.github.io/posts/2015-08-Understanding-LSTMs/**](http://colah.github.io/posts/2015-08-Understanding-LSTMs/)**)의 내용을 번역 및 요약 한 것입니다)**

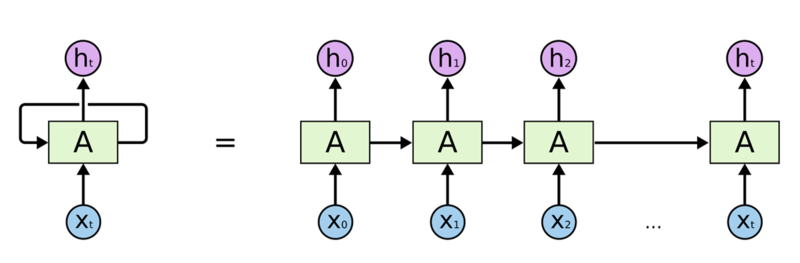
[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

마찬가지로 신경망을 단순하게 표현을 하고 반복되는 구조를 표현하고 있습니다. 이런 구조로 표현을 할 수 있겠죠.

​

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

이걸 아까처럼 펼쳐보겠습니다.

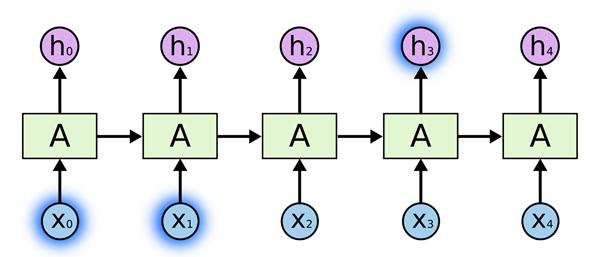
[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

이 그림에는 표시되어 있지 않지만 마찬가지로 시간출을 중심으로 펼쳐진 그림이라고 할 수 있습니다. 이 그림을 통해서 RNN이 하나의 네트워크가 여러 개 복사된 형태로 구성되어 있고 각 네트워크는 다음 단계로 정보를 넘겨주고 있는 형태라는 것을 알 수 있습니다.

​

RNN의 장기 의존성(Long-Term Dependency)의 문제점

이제 순서를 고려한 처리가 가능한 RNN이 있으니 모든 sequence data를 가뿐히 처리할 수 있을 것 같지만 여전히 문제가 남아있었습니다.

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

위 그림은 제공된 데이터와 참고해야정보의 입력 위치가 비교적 가까운 상황을 표현하고 있습니다.

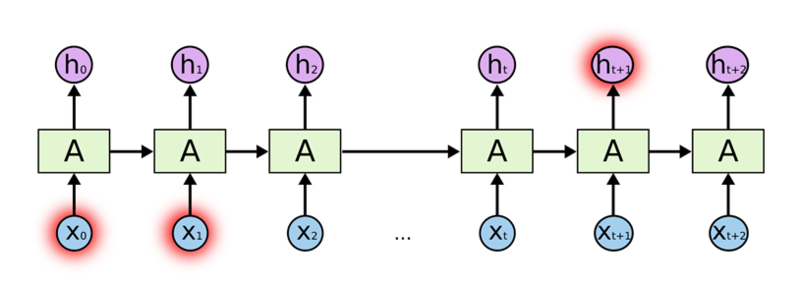
​

예를 들어서

"the clouds are in the sky"라는 문장이 있을 때

"the clouds are in the"라는 문장이 있다면 뒤 단어가 "sky"일 확률이 훨씬 높습니다. 더 이상의 문맥이 필요 없다고 할 수 있겠죠.

따라서 제공된 데이터와 참고해야 할 정보의 입력 위치 차이(Gap)가 크지 않습니다. 이런 상황만 계속된다면 아무런 문제가 없겠죠. 하지만 이 차이(Gap)이 큰 상황에서 문제가 발생합니다.

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

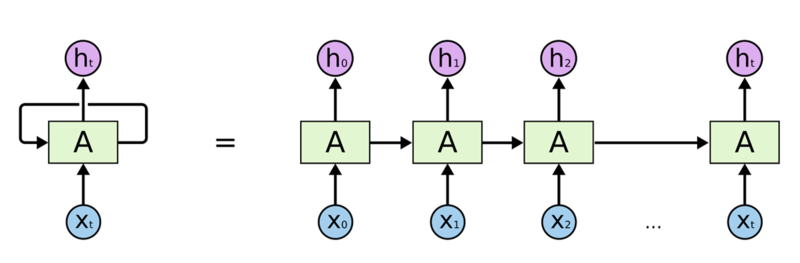
예를 들어서 "I grew up in France... I speak fluent French"라는 문장에서 "French"를 예측한다고 했을 때 France가 필요한 정보라고 할 수 있습니다.

[https://mblogthumb-phinf.pstatic.net/MjAxODA3MDJfMTk0/MDAxNTMwNTM3NDUxMTky.H3pc45yTAG8QC1_uf1oGBMoC2hHs7Ivz-Eq79ua0O4og.GTdjfmPM2mv_E-ujsHG10mvSw5CQDG6ULNUB1r2NvQMg.PNG.magnking/image.png?type=w800](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

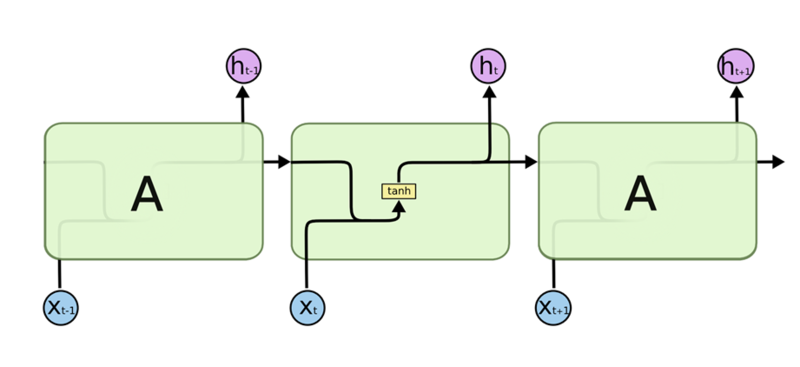
제공된 데이터와 참고해야정보의 입력 위치 차이(Gap)이 크지 않다면 문제가 없겠지만 위치 차이가 커지는 상황이 자주 발생합니다. 문장표현의 순서상 갭이 커질 수록 RNN은 두 정보의 문맥을 연결하기 힘들어지고 실제로 성능도 저하되게 됩니다. 하지만 지금 RNN이 매우 여러 분야에서 활발하게 사용되고 있는 것을 보면 분명 좋은 해결책이 등장한 것을 알 수 있습니다. 실제로 RNN의 성과들은 대부분 LSTM을 통해서 이루어졌다고 합니다.

LSTM이란 무언인가?

LSTM에 대해서 알아보기 위해서 일반적인 RNN의구조를 다시 한번 살펴보겠습니다.

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

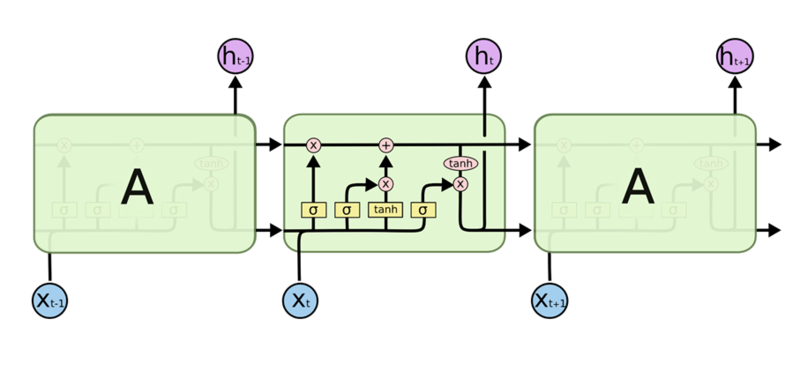
이미 보신 이 그림을 조금 다르게 표현하면 다음과 같습니다.

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

위 그림은 싱글 레이어(단일 tanh 레이어)를 가지고 반복되는 표준 RNN 모듈을 표현하고 있습니다. 연속이 되어 있다는 의미에서 체인 구조라고 부르기도 하는 것 같습니다.

​

그럼 LSTM 네트워크는 어떻게 생겼을까요?

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

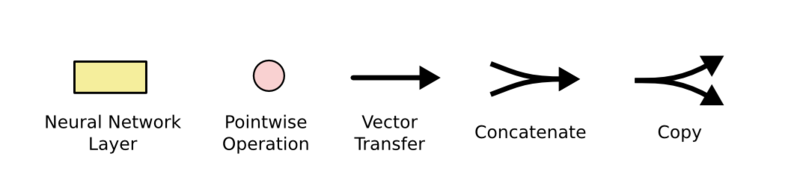
표준 RNN은 싱글레이어였지만 LSTM 네트워크는 훨씬 복잡한 구조를 가지고 있는 것을 확인할 수 있습니다.

​

LSTM에는 상호작용을 하는 4개의 레이어가 존재합니다. 이 구조가 반복되는 모듈을 가지는 것이 LSTM입니다. 기존 RNN의 문제점이 입력된 데이터와 참고해야 할 데이터의 위치 차이가 커질 때 문맥을 연결하기 힘들어진다는 것이었습니다. 그렇다면 이 상호작용 구조가 이런 부분을 해결하는데 도움을 준다는 것을 예측해 볼 수 있습니다. 그럼 LSTM을 조금 더 자세히 알아보도록 하겠습니다.

기호 살펴보기

먼저 그림에 표현된 그림의 기호를 살펴보겠습니다.

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

노란색 박스는 뉴럴 네트워크의 레이어 단위를 표현하고 있습니다. 각각 입력층과 출력층이 있습니다.

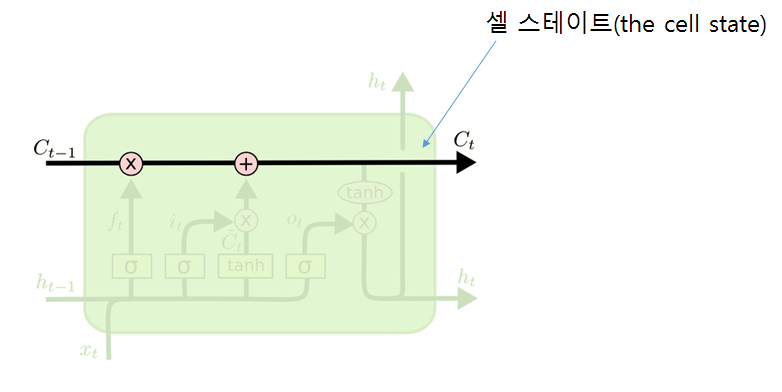
분홍색 원은 점단위의 연산을 표현하고 있습니다. 점단위의 연산은 예를 들면 벡터의 더하기 연산 등이 있다고 합니다.

각 라인은 온전한 vector를 포함하고 있고 각 출력값은 다른 노드의 입력값으로 들어가고 있습니다.

하나로 합쳐지는 네 번째 기호는 데이터들에 변화가 일어나는 것은 아니고 한 곳에 합쳐져서 같이 이동하는 것을 의미합니다.

화살표가 두 개로 나눠지는 다섯 번째 기호의 경우는 결과 값이 복사되는 것을 의미합니다.

셀 스테이트(the cell state)

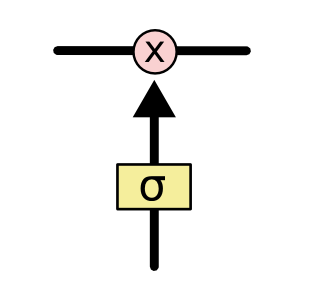
[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

셀 스테이트는 LSTM의 핵심적인 부분입니다. 이 셀스테이트는 하나의 컨베이어 벨트처럼 천체 체인을 통과합니다. 이 구조를 통해서 정보는 큰 변함 없이 계속 다음 단계에 전달이 될 수 있습니다.

​

LSTM은 이 셀 스테이트에 신중하게 정제된 구조를 가진 게이트(gate)라는 요소를 활용해서 정보를 더하거나 제거하는 기능을 수행합니다. 게이트(Gates)들은 선택적으로 정보들이 흘러들어갈 수 있도록 만드는 장치를 의미합니다. 게이트들은 시그모이드 신경망 레이어(sigmoid neural net layer)와 점단위 곱하기 연산으로 구성되어 있습니다.

시그모이드 레이어

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

시그모이드 레이어(The sigmoid layer)는 0 혹은 1의 값을 출력합니다. 각 구성요소가 영향을 주게 될지를 결정해주는 역할을 합니다. 0이라는 값을 가지게 되면 해당 구성요소가 매래의 결과에 아무런 영향을 주지 않게 됩니다. 1이라는 값을 가지게 되면 해당 구성요소가 확실히 미래의 예측결과에 영향을 주도록 데이터가 흘러가게 합니다.

셀 스테이트를 보호하고 컨트롤 하기 위한 세 가지 게이트들

셀 스테이트를 보호하고 컨트롤하기 위해서 세 가지 게이트들을 이용하고 있습니다.

​

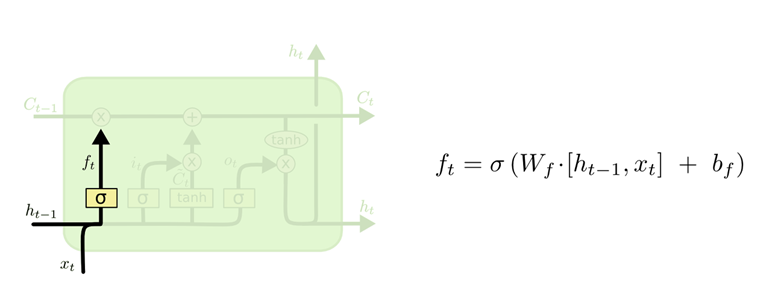
forget gate layer(시그모이드 레이어)

input gate layer(시그모이드 레이어)

tanh layer

LSTM 스텝 1 셀 스테이트에서 어떤 정보를 버릴지 선택하는 과정

forget gate layer(시그모이드 레이어)

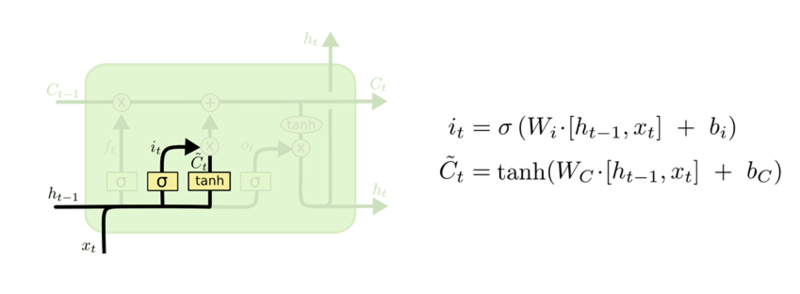
[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

forget gate layer는 0과 1 사이의 출력값을 가지는 ht-1과 xt를 입력 값으로 받습니다. 여기서 xt는 새로운 입력값이고 ht1은 이전 은닉층에서 입력되는 값입니다.

​

출력값이 1인 경우 완전히 이 값을 유지하게 되고 출력값이 0이 될 경우 완전히 이 값을 버리는 것이 됩니다.

LSTM 스텝 2 새로운 정보가 셀 스테이트에 저장될지를 결정하는 단계

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

**input gate layer(시그모이드 레이어)**

어떤 값을 우리가 업데이트할 지를 결정하는 역할을 합니다.

​

**tanh layer**

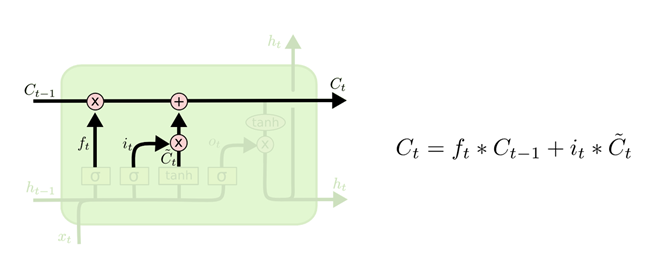
셀 스테이트에 더해질 수 있는 새로운 후보 값

[https://mblogthumb-phinf.pstatic.net/MjAxODA3MDJfMjg5/MDAxNTMwNTM3NzQ4MDAy.O2JJdZItSeWVgSTk_O1K0jQvB-cF4kD6J5nHhKjCiqwg.fC51iG9HxjGNroKQnJezyO7kWtrVs98-35YzT9UN_vYg.PNG.magnking/image.png?type=w800](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

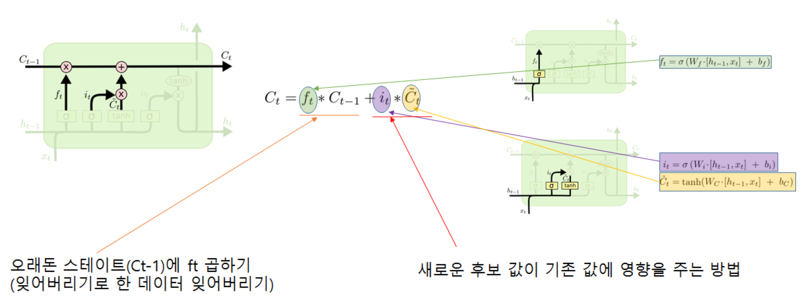
를 만들어 냅니다.

​

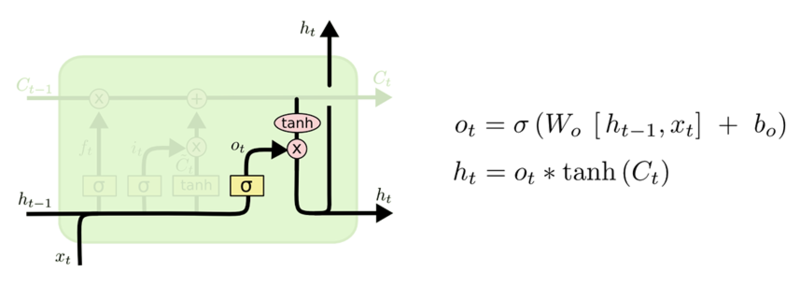
LSTM 스텝 3 오래된 셀 스테이트(Ct-1)를 새로운 스테이트인 Ct로 업데이트

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

앞의 단계에서 얻은 값들을 토대로 셀 스테이트를 업데이트 합니다.

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

LSTM 스텝 4 어떤 출력값을 출력할지 결정

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

1 어떤 값을 출력할지 결정하는 시그모이드 레이어

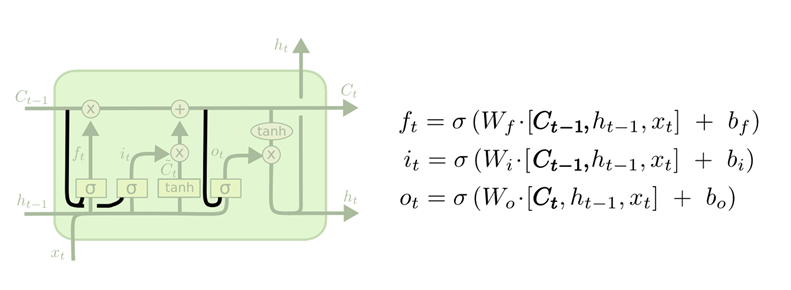
2 셀 스테이트의 값이 tanh 함수를 거쳐서 -1과 1 사이의 값으로 출력

3 1의 시그모이드 레이어에서 얻은 값과 2dml tanh함수를 거쳐 얻은 값을 곱해서 출력

​

이상이 표준적인 LSTM 방식입니다. 이외에도 여러 변칙 패턴들이 등장했고 마찬가지로 좋은 성능을 보여주고 있다고 합니다.

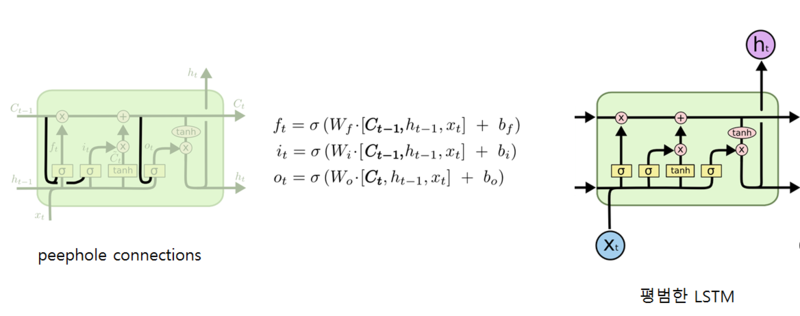
LSTM의 변칙 패턴 1

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

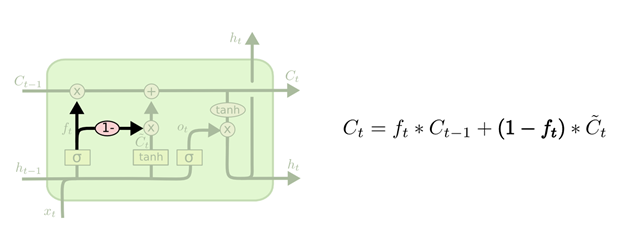
게이트가 셀 스테이트 자체를 입력값으로 받고 있는 것을 확인할 수 있습니다.

​

<비교>

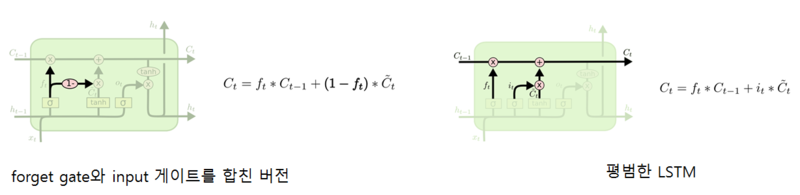
[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

LSTM의 변칙 패턴 2

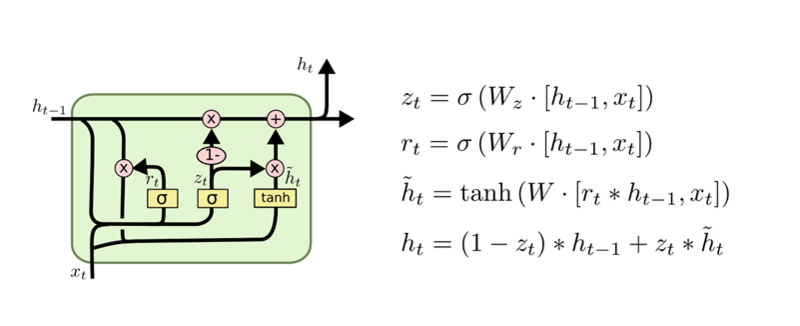
[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

forget gate와 input 게이트를 합친 버전입니다. 어떤 값을 잊어버리고 어떤 값을 더할지 과정을 따로 수행하지 않고 이를 동시에 결정하는 방식입니다. 새로운 값이 제공될 때만 이전 값을 잊어버리게 됩니다.

<비교>

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

LSTM의 변칙 패턴3

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

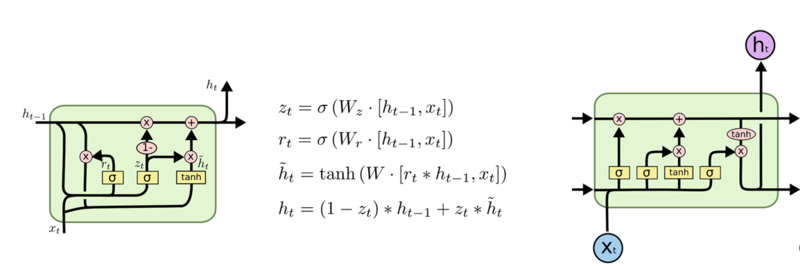
the gated Recurrent Unit(GRU)

​

GRU는 forget gate와 input gate를 하나의 업데이트 게이트로 통일했고 셀 스테이트와 히든 스테이트를 하나로 합쳤습니다.

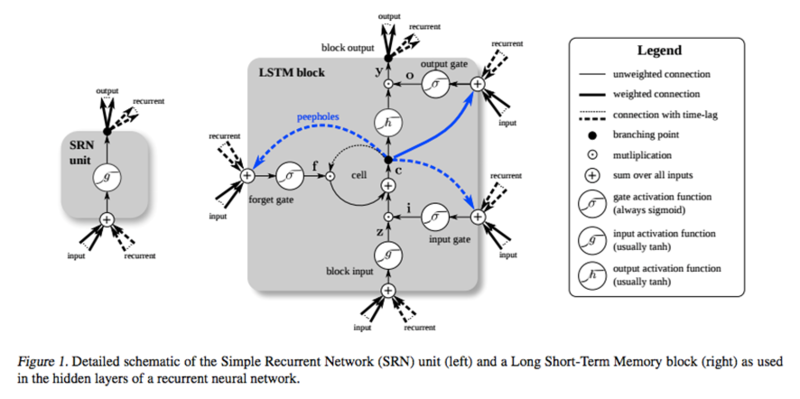
​

<비교>

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

다른 그림으로 보는 LSTM

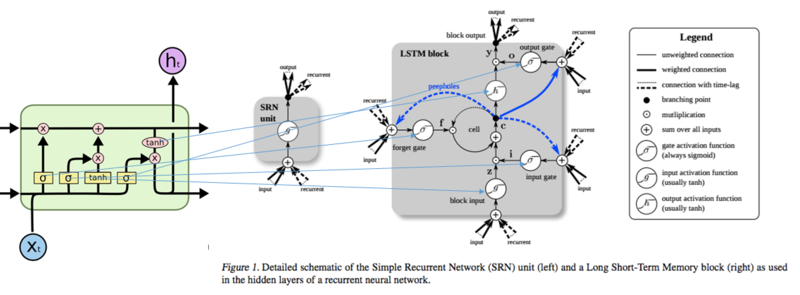
​

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

조금 더 복잡해 보이지만 같은 내용을 표현하고 있습니다.

​

<비교>

[](https://m.blog.naver.com/PostView.nhn?blogId=magnking&logNo=221311273459&proxyReferer=https%3A%2F%2Fwww.google.com%2F)

CNN

**평소 무엇인가를 쉽게 설명하는 능력이 있다고 생각해서  , CNN (convolutional neural network) 도 그렇게 해볼까 했는데 역시 무리. 쉽게 설명한다는것은 그것에 대해 확실한 이해를 가지고 있다고 생각될때 가능한것인데 아직 CNN 라는 풍랑에서 표류중이기 때문에 대신해서 좋은 해외 블로그글을 번역하고자 한다. 일반 소프트웨어 엔지니어 입장에서 딥러닝을 활용하기 위해서 수학적인 이해는 필요 없다고 생각하며, 대신 직관적인 이해는 반드시 해야하는데 이 글은 좋은 지침이 될거 같다.**

이 글을 이해하기 위해서 선수학습으로 다음의 것을 알고 있으면 좋다. (물론 이것들도 직관적인 이해만 하면 된다. 옆에서 누가 설명해주면 1~2시간이면 족할 내용이지만 아마 직접 책을 통해 공부한다면 꽤나 오래 걸릴수도 있겠다.)

- 퍼셉트론

- 하강경사법

- Overfitting

- 오류역전파 알고리즘

- 소벨마스크

- 선형회귀/로지스틱 회귀

- Sigmoid

(오토인코더나,RBM 같은 어려운것들은 나중에~)

\* 딥러닝 처음 접한 사람을 위한 팁

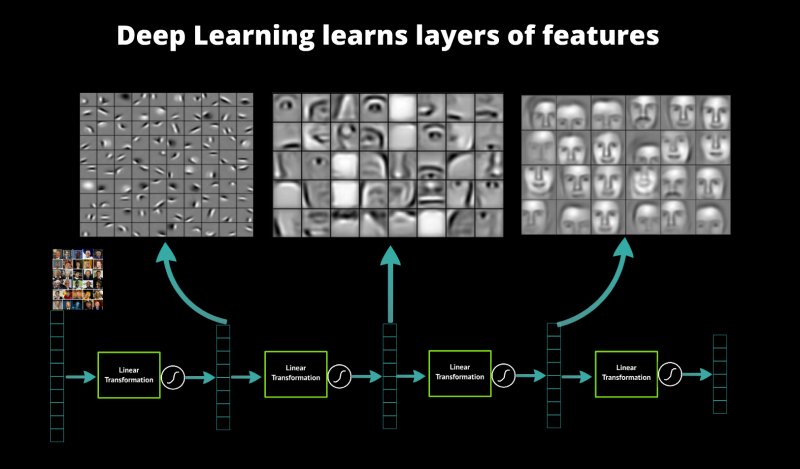
이 포스트에서 설명할 CNN 은 딥러닝은 한 종류로 주로 이미지를 인식하는데 사용됩니다. ( 음성 및 1차원 타임시리즈 데이타도 가능) 2012년 세계적인 이미지 인식 경연 대회 (ILSVRC) 에서 세계 유수의 기관을 제치고 난데없이 큰 격차로 캐나다의 토론토 대학의 슈퍼비전이 우승하게 되는데 그때 사용된 방법이 CNN 에 기반합니다. (그 동안은 SIFT,HOG방식등 )

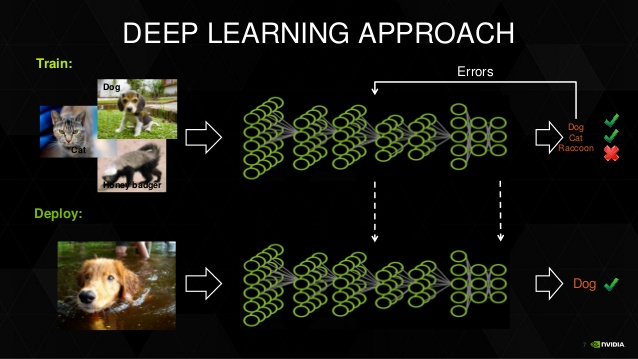
이 대회는 천만장의 이미지데이터를 기계학습으로 하급하고 15만장의 이미지를 사용해서 테스트하여 정답율을 측정합니다.

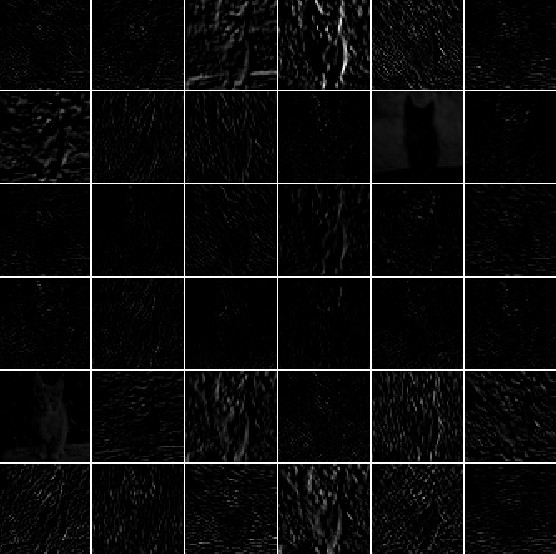
즉 고양이 이미지를 고양이로 인식하면 성공~

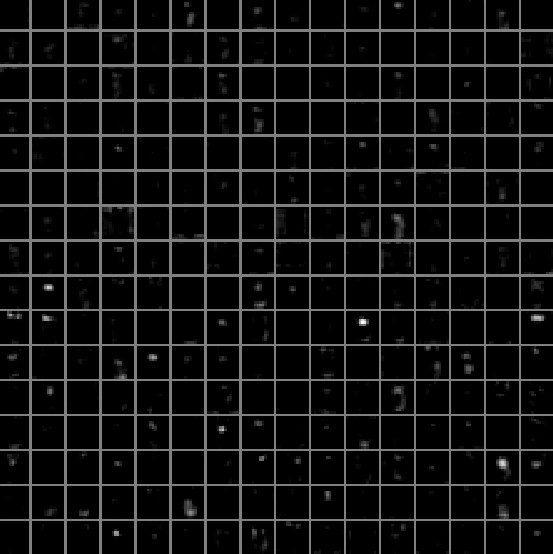
이 고양이 인식에는 **'특징표현 학습(feature**representation learning)**'** 이라는 엄청난 발명이 사용되었는데, **컴퓨터 스스로가 특징표현을 만들어 내는것**이다.

다음은 구글의 고양이 연구 및 기타 자료들로써 어느정도 직관적인 이해를 할 수 있을것이다.  아래층에서는 점이나 엣지등의 이미지에 자주 검출되는 '모양' 을 인식하는것 뿐이지만 위로 가면서 원이나 삼각등의 모형을 인식할수 있으며 그 위로는 얼굴과 같은 형상을 얻게되며, 새로운 이미지를 분류할때 저러한  형상이 나올 확율이 높으면  = 고양이다 라고 분류 하는것이다.









위에것은 CONV 1 레이어이고 아래것은 CONV 5 레이어이다. AlexNet 아키텍쳐에서 고양이 이미지를 학습한것이며 각각의 박스는 각 필터들과 연관된 엑티베이션 맵을 보여준다.  액티베이션은 Sparse (대부분 0 이며 위 이미지에서는 검정으로 보여진다.) 이고 대부분 local 이다.

딥러닝에서 Convolution 을  이해해보자.

2015-03-26 by [Tim Dettmers](http://timdettmers.com/author/tim-dettmers/)

현재 컨볼루션 은 아마도 딥러닝에 관해서  가장 중요한 개념일것이다. 대부분의 기계학습의 전면에  컨볼루션 과 컨볼루션 nets 은 딥러닝을  스타덤에 올려놓았다. 근데 무엇이 컨볼루션을 그렇게  강력하게 만들까? 어떻게 작동할까?  이런 질문에 대해 이 블로그에서 다른 컨셉들과 비교하며 설명할것이다. 컨볼루션에 대한 직관적인 이해를 할수 있는데  도움이 될것이다.

이미 몇몇의 컨볼루션 관련 블로그들이 있지만, 그것들은 굳이 필요없는 (이해하는데 전혀 도움이 된다는 방식의) 수학적 상세표현을 통해 매우 큰 혼동만 주고 있다고 생각한다.  이 블로그에도 수학표현이 없다고 말하지는 않겠으나, 적어도 나는 그것들을 모든사람이 이해할수있는 이미지와 함께 표현하여 개념적 이해를 도울것이다. 이 블로그의 첫번째 장에서는 누구나 컨볼루션과 Convolutional Neural Network 를 직관적으로 이해할수 있도록 하는게 목표이다. 두번째 장에서는 좀 더 깊숙한 개념을 설명하여 연구자나 깊이있는 이해를 하고 싶어하는 사람들에게 도움이 되는 글들로 채울것이다.

컨볼루션이 무엇인가?

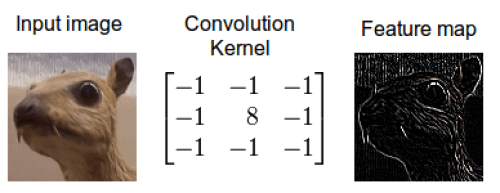
전체 블로그는 이 질문에 대한 정확한 대답을 드리기 위해 채워져 있다. 먼저 방향설정을 잡아보자. 그래 컨볼루션의 대략적인 의미는 무엇인가?

먼저 당신은 정보를 섞는것으로서 컨볼루션을 상상할수 있다. 2개의 양동이에 어떤 정보가 가득 차있고, 그것을 하나의 양동이에 쏟아넣는것을 상상해보자. 그리고 어떤 특정한 룰 따라서 섞어보자. 각각의 양동이는 그 자신의 레시피를 가지고 있고,  그것을 통해 어떻게 정보들이 하나의 양동이에 서로서로 섞이는지 알려준다. 즉  컨볼루션은 2개의 정보가 서로 섞이는 순서가 있는 절차이다. (역주 : 두 벡터의 내적을 생각해보자)

컨볼루션은 또한 수학적으로 표현될 수 있다. 사실, 더하기,곱하기,미분하기등과 같은 수학적인 연산이다. 복잡한 방정식을 간단하게 하기 위한 좋은 도구가 될 수 있다. 컨볼루션들은 물리학이나 엔지니어링에서 중요하게 사용되는데, 그런 복잡한 방정식을 간소화 할 경우가 많기 때문이다. 두번째 장에서  이러한 두 분야에서의 관계와 통합에 대해 알아 볼 예정이다. 그러나 지금은 현실적인 관점에서만 컨볼션을 살펴볼것이다.

어떻게 컨볼루션을 이미지에 적용할 수 있을까?

우리가 이미지에 컨볼루션을 적용할때, 우리는 2차원으로 생각해 볼수 있다. 즉 너비와 높이를 가진 이미지.우리가 2개의 양동이를 섞을때, 첫번째 양동이에는 원본 이미지 (3차원 행렬의 픽셀 전체) 가 들어가고, 빨강,녹생,파랑의 색상 채널들이 하나의 행렬이 된다. (역주: 3가지 색상을 하나의 그레이스케일로 줄여도 된다) 하나의 픽셀은 각각의 색상 채널에서 0~255 사이의 정수로 구성된다. 두번째 양동이에는 컨볼루션 커널이 있고, 이것은 실수의 단일 행렬로 이루어져 있고 이것은  어떻게 원본 이미지와 커널을 컨볼루션 연산에 의해서 섞는지에 대한 레시피로써 크기와 패턴들이 구성된다. 이 커널의 출력은 '피쳐 맵' 이라고 불리는 이미지이다. 커널 하나당 각각의 색상 채널에는 각각의 피처맵이 생길것이다. (역주: 2번째 양동이, 즉 커널종류가 다양할 수록 다양한 피처맵이 생긴다.)

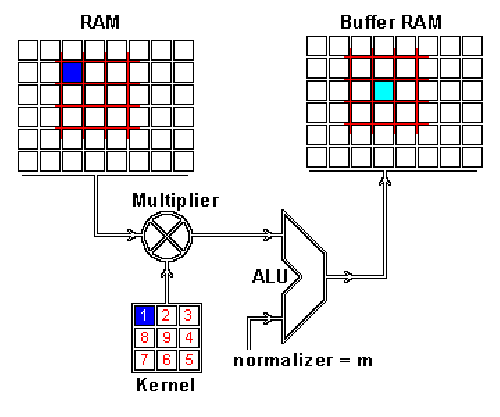
[](http://i1.wp.com/timdettmers.com/wp-content/uploads/2015/03/convolution.png)

원본이미지 와 경계선 추출 커널 (두번째 양동이) 을 섞어서 만들어진 피처맵.

(역주:  CNN 은 이 커널을 자동으로 만들어 준다는게 핵심이다)

이 두가지 정보들을 컨볼루션을 통해 엮었다.  여기서 컨볼루션을 적용하는 한가지 방식은 이미지 패치를 원본이미지로 부터 커널 사이즈 만큼 가져와서 이미지 패치 와 컨볼루션 커널을 연산한다. (역주:  위에 원본 이미지가 100\*100 사이즈 이고 3\*3 크기 컨볼루션  커널을 행렬 곱 할때, 원본 이미지에서 커널과 곱할 부분을 떼어내게 되는데 이 떼어낸것을 패치라고 한다.)

하나의 연산의 합이 피쳐맵에서 하나의 픽셀에 사상된다. 피처맵의 하나의 픽셀이 계산되고 난 후에는 이미지 패치가 하나씩 오른쪽으로 이동하면서 새로운 패치 정의되고 그 패치는 커널과 연산하여 새로운 픽셀이 계산되어진다. 이 절차를 아래 이미지를 통해 이해해 보자.

[](http://i2.wp.com/timdettmers.com/wp-content/uploads/2015/03/aa-convolution-02.gif)

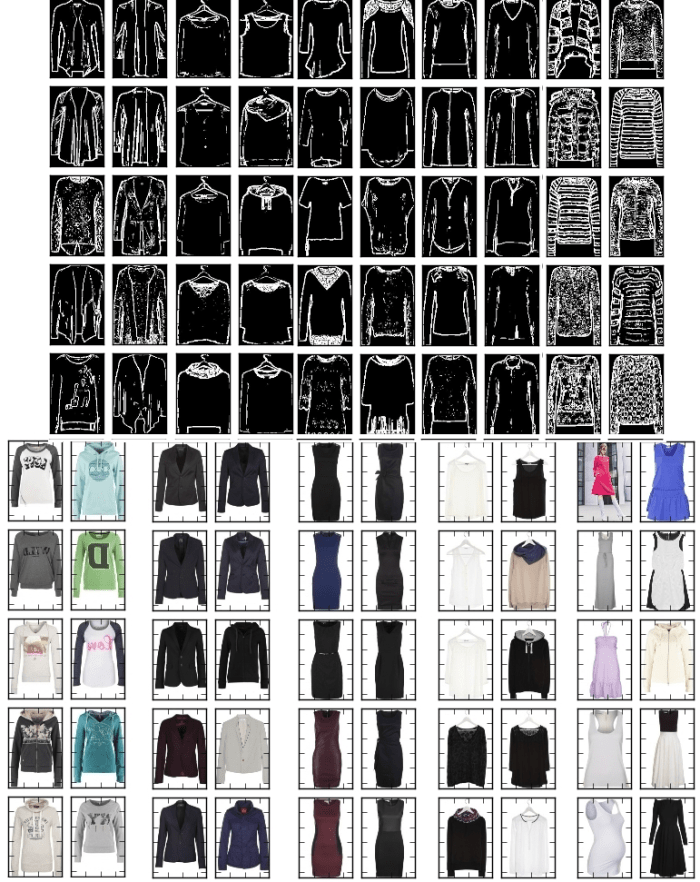
(역주: 이렇게 커널이 패치와 연산될때, 패딩/슬라이딩값들을 조절한다던가, 커다란 이미지를 1/4 같이  줄이는 풀링이라든지 , 아예 DropOut 시킨다던가 레이어간에 부분적으로만 연결한다던가, 파라미터들을 그룹핑하여 공유한다던가 하는 것들이 있는데 전체적인 직관력을 기르고 난후에 접근하시면 됩니다. 일단은 원본이미지에서 어떤 필터를 중간에 껴서 새로운 이미지 집합을 만들어낸다는것에 집중하면 될거 같습니다.)

왜 이미지의 컨볼루션이 머신러닝에서 중요한가?

이미지에는 다양하게 추출될수있는 정보들이 포함되어있다. 좋은 예로 내가 참여한 프로젝트를 살펴보자. Burda Bootcamp 는 빠른속도의 학생들이 기술적으로 위험한 프로젝트들을 매우 짧은 기간안에 만들기위한 해커튼 스타일의 환경의 프로토 타이핑 랩인데 우리는 11개의 프로덕트를 2달만에 만들었다. 하나의 프로젝트에서 나는 딥 오토엔코더와 함께 패션 이미지 탐색을 만들길 원했다. 당신은 패션 아이템의 이미지를 업로드하고 오토엔코더는 비슷한 스타일의 옷을 포함한 이미지를 찾아야했다.

지금 만약 당신이 옷의 스타일에 관한 차이점에 대해서 설명할때 그 옷의 색깔은 그것을 하기에 별로 유용하진 않을것이다. 또한 상표의 엠블램같은것 또한 마찬가지이다. 가장 중요한것은 옷의 윤곽/모양그 자체일것인데 일반적으로 브라우스의 모양은 셔츠,자켓등의 모양과는 아주 다르다. 그래서 만약 우리가 필요없는 정보를 추출하는 필터를 설계하면 그것은 필요없는 것들에 의해 구분되어지는 참사를 막을 수 있을것이다. 결국 우리는 좀 더 쉽게 커널과 함께 컨볼루션을 수행할 것이다.

나의 동료는 데이타를 전처리하고 소벨 윤곽선 검출기를 적용하였다. (첫번째 그림에서 윤곽선 검출한것과 비슷) 객체의 모양의 외곽선을 제외하고 이미지로부터 모드것을 제거하는 필터이며, 이것은 왜 컨볼루션의 어플리케이션이 종종 필터라고 불리는지 말해준다. 그리고 커널들도 필터라고도 불리운다. 결과로 만들어진 피쳐맵은 당신이 다양한 종류의 옷을 구분할때 매우 큰 도움을 줄것이다. 아래 이미지들을 보자.

[](http://i1.wp.com/timdettmers.com/wp-content/uploads/2015/03/autoencoder_fashion_features_and_results.png)

이런 종류의 절차를 사용하는것이  -  입력을 받고, 변환하고 , 변형된 이미지를 알고리즘으로 먹이고 - 피처 엔지니어링 이라고 불린다. 피쳐 엔지니어링은 매우 어려운데, 이런 기술을 익히기 위해 당신을 도울 리소스는  별로 없기 때문이다.  결과적으로 아주 소수의 사람들이 피쳐 엔지니어링을 넓은 범위의 업무에서 잘 적용할 수 있는데. 그냥 피쳐 엔지니어링을 잘하는 방법을 살펴봄으로써 느껴보자.  [the most important skill to score well in Kaggle competitions](http://blog.kaggle.com/2014/08/01/learning-from-the-best/).

피쳐 엔지니어링은 너무 어렵다. 각각의 데이타 타입과 각각 문제의 타입에 대해 다른 피쳐들이 적합하기 때문이다. 이미지 업무에 관한 피처 엔지니어링의 지식은 타임시리즈 데이터에는 쓸모가 없을때가 많으며, 심지어 우리는 비슷한 이미지 업무에 대해서도 좋은 피처들을 엔지니어링하기 쉽지 않다. 이미지 안의 객체들이 우리가 무엇을 하려고하는지에 관해서 다르게 결정되기 때문이다. 굉장히 오랜기간의 경험이 필요한 일이다. 그래서 피처 엔지니어링은 매우 어려우며 당신 앞에 닥치게된 업무는  빈바닥에서 시작해서 새롭게 시작해야 한다는것을 의미한다. 그러나 그러나 !!!!  그런 커널들이 대부분의 업무에 맞춰서 자동적으로 적합하게 찾아진다면 어떨까??

CNN (Convolutional Neural Network)  으로

우리의 커널이 정확하게 고정된 값을 갖게하는 대신해서 (역주: 엔지니어가 직접 커널을 선택해주는것을 대신해서) 컨볼루셔널 넷이 정확하게 그것을 한다.  우리의 컨볼루션넷을 학습 시킴으로써 , 커널은 점점 더 주어진 이미지(혹은 주어진 피처맵) 를 필터링을 잘 하게될것이다. 이 프로세스는 자동적으로 이루어지므로 피처 학습이라고 불리운다. 피처 학습은 자동적으로 각각의 목적에 대해 생성된다. 우리는 단지 새로운 목적과 부합한 필터를 찾기위해 우리의 네트워크를 간단하게 학습시키기만 하면 된다. (역주 : 이미 CaffeNet, GoogleNet 등의 아키텍처 모델들이 존재함. 물론 파인튜닝도 필요하고, 전혀 다른 도메인이라면 어려운 도전이 필요합니다)  이것이 컨볼루션 넷을 강력하게 만드는 것이라고 볼수있습니다. 피쳐엔지니어링은 더 이상 어렵지 않아요. 자동적으로 만들어지니깐!

보통 우리는 컨볼루션 넷에서 싱글 커널을 학습하진 않습니다. 대신해 우리는 다층의 커널들의 계층에 대해 학습하게 되는데 예를들어 32\*16\*16 의 커널이 256\*256 이미지에 적용되면 32 개의 피처맵들을 241\*241 사이즈로 생성하게 됩니다. 자동적으로 우리는 32개의 새로운 피처 (우리의 목적에 부합되는 형태를 가진) 들을 학습하게되며 이러한 피처들은 다음 커널의 입력으로 사용됩니다. (즉 여러개의 커널을 거침. 인간의 뇌가 여러 레이어의 신경망으로 되어 있는것 처럼 ) 일단 계층적인 특징(feature)들을 배우게 되면 우리는 간단히 그것들을 fully connected 로 통과하게 하고 , 이미지에서 특정 클래스(고양이, 자전거) 로의 분류를 위해  그것들을 합칩니다.  (역주 : 특정 표현들이 얼마만큼 존재하냐에 따라서 확율적으로 선택해줍니다. 소프트맥스라는 회귀방법이 주로 사용됩니다.)

이것들이 개념적인 수준에서 CNN 을 이해할수있는 거의 모든 것 이며 , 2편에서는 좀 더 상세한 얘기들을 해 볼 것 입니다.

출처: <https://hamait.tistory.com/535> [HAMA 블로그]