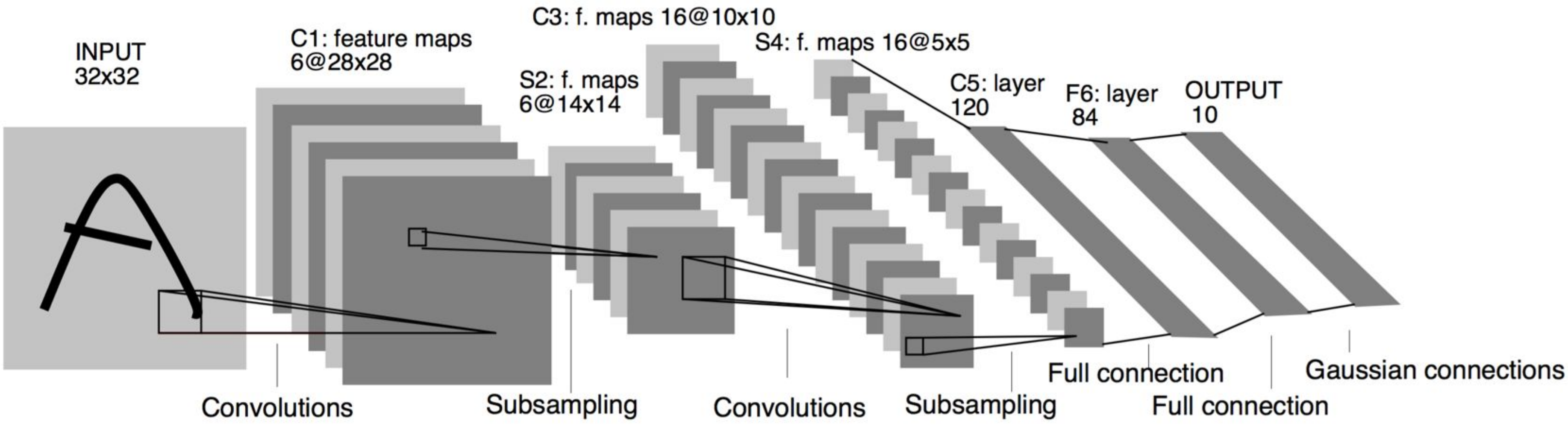
Tensorflow day 3

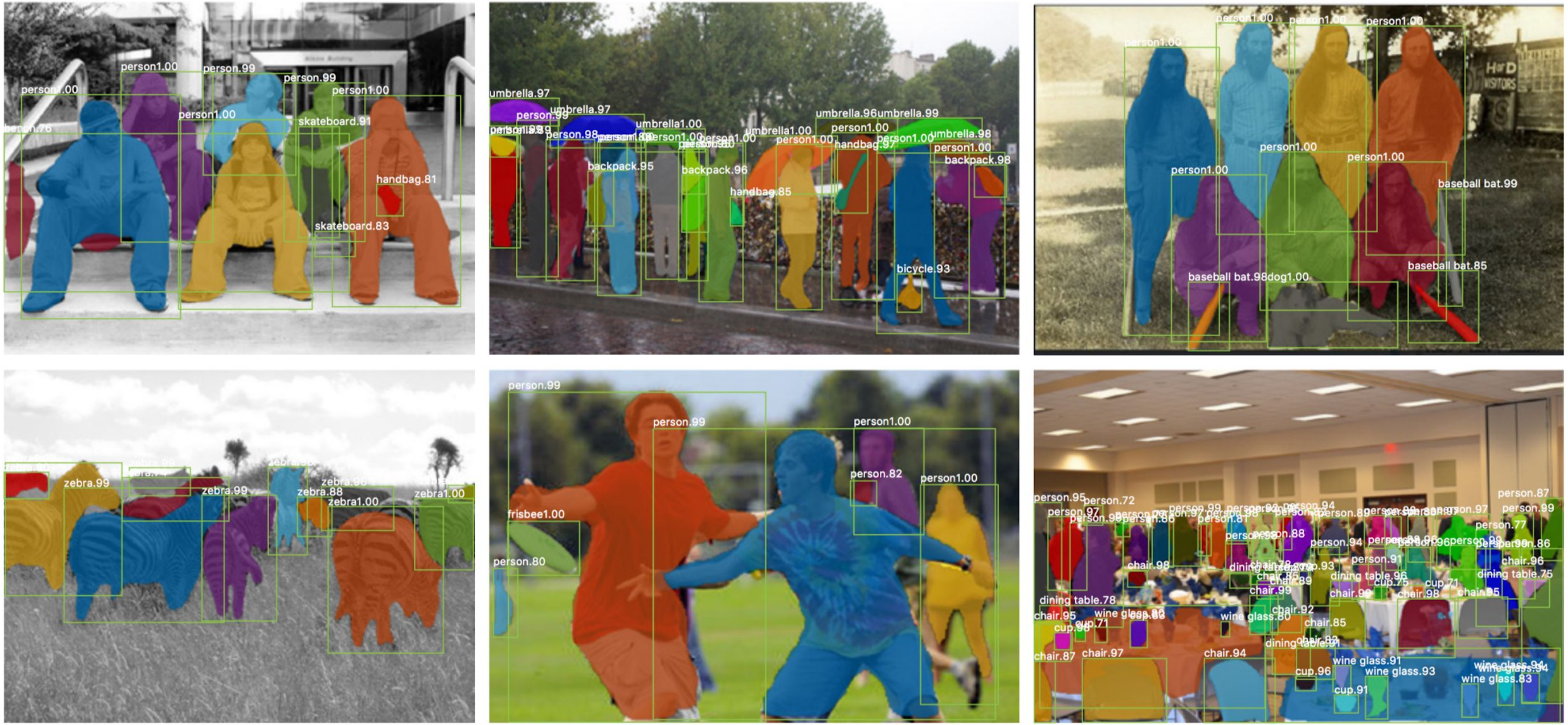
모두의 연구소 Lab. Rubato
소준섭

Convolutional Neural Networks



Gradient-based learning applied to document recognition
[LeCun, Bottou, Bengio, Haffner 1998]

CNNs - Object Detection / Segmentation



He, et. al., Mask R-CNN (https://arxiv.org/abs/1703.06870) / https://www.youtube.com/watch?v=OOT3UIXZztE

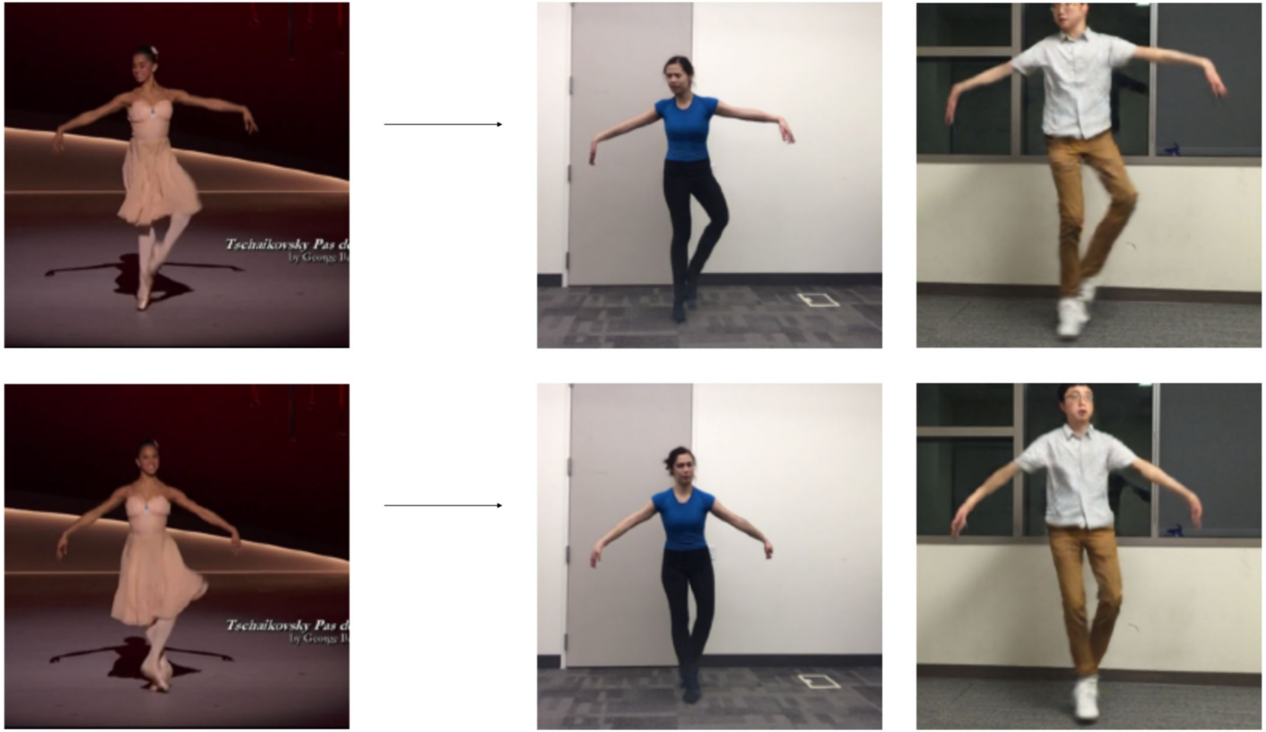
CNNs - Pose Estimation

Cao, et. al., Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields (https://arxiv.org/abs/1611.08050) / https://www.youtube.com/watch?v=pW6nZXeWlGM

CNNs - Dense Pose

Guler, et. al., DensePose: Dense Human Pose Estimation In The Wild / https://www.youtube.com/watch?v=Dhkd_bAwwMc&t=67s

CNNs - Everybody Dance Now



Caroline Chan, et. al., Everybody Dance Now / https://www.youtube.com/watch?v=PCBTZh41Ris

CNNs - Image Captioning

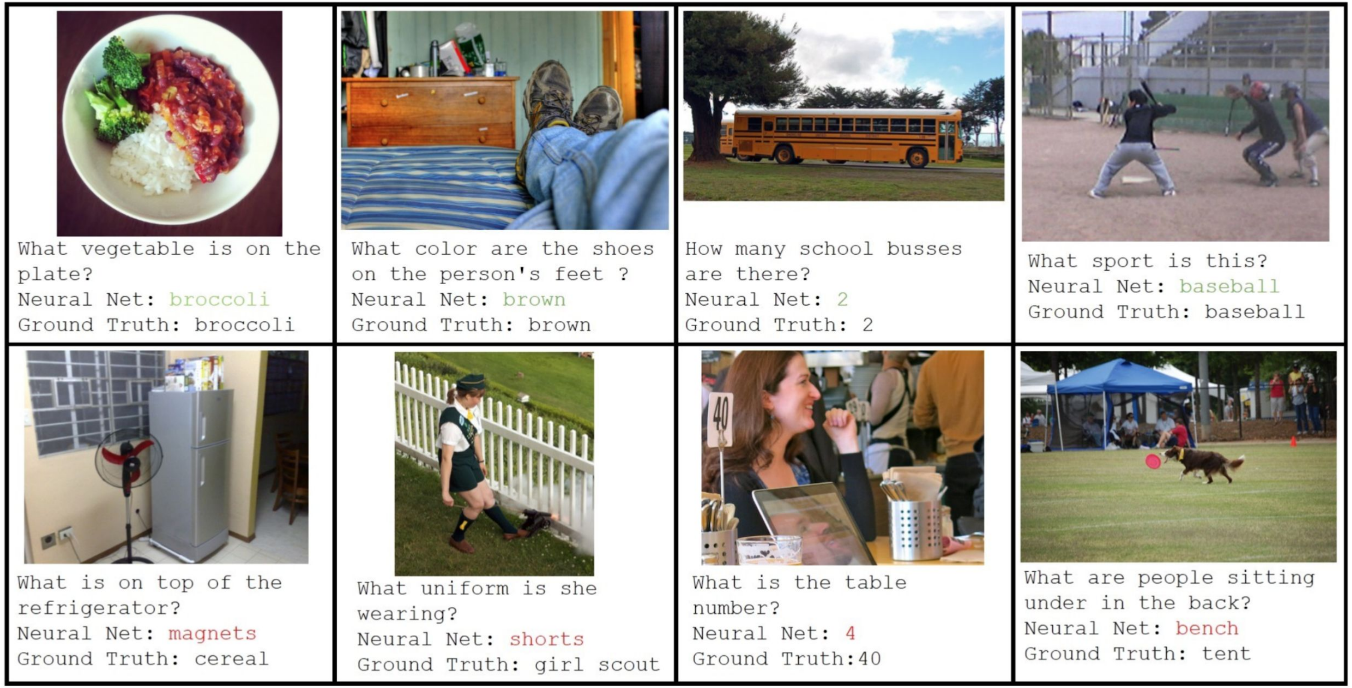
man in black shirt is playing guitar.

construction worker in orange safety vest is working on road.

two young girls are playing with lego toy.

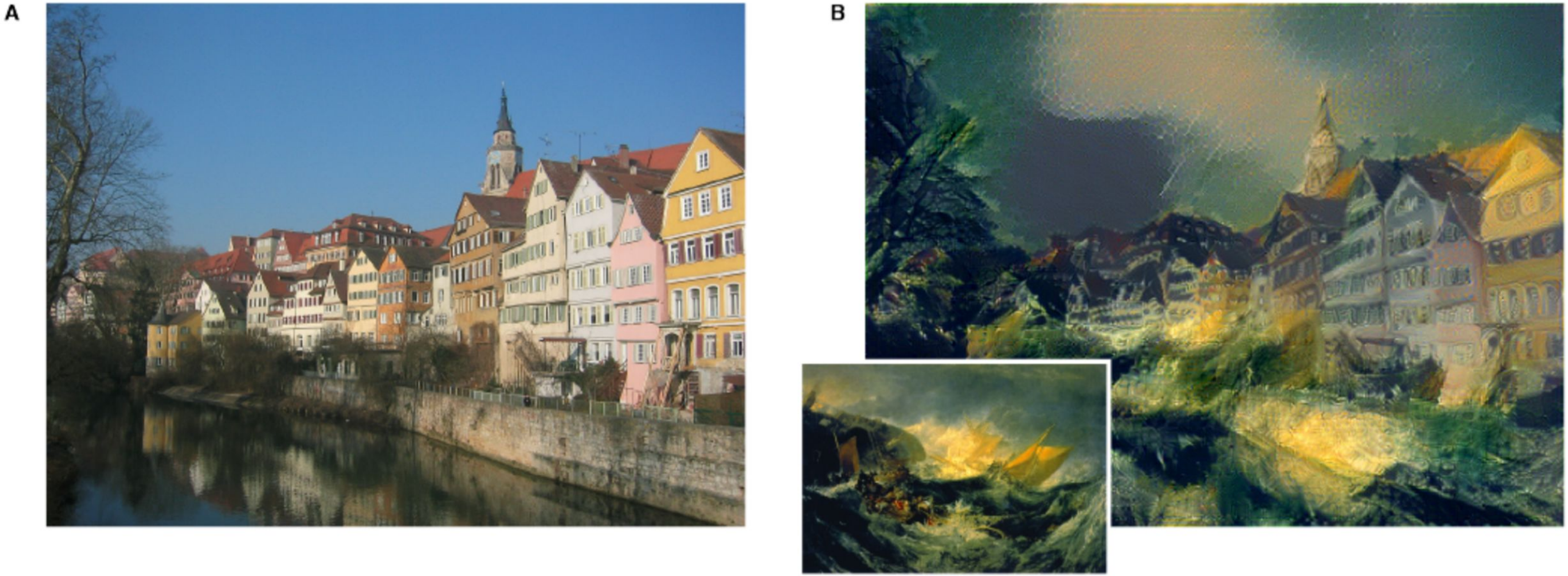
Karpathy, et. al, Deep Visual-Semantic Alignments for Generating Image Descriptions (https://arxiv.org/abs/1412.2306) / https://www.youtube.com/watch?v=8BFzu9m52sc

CNNs - Video Question & Answering



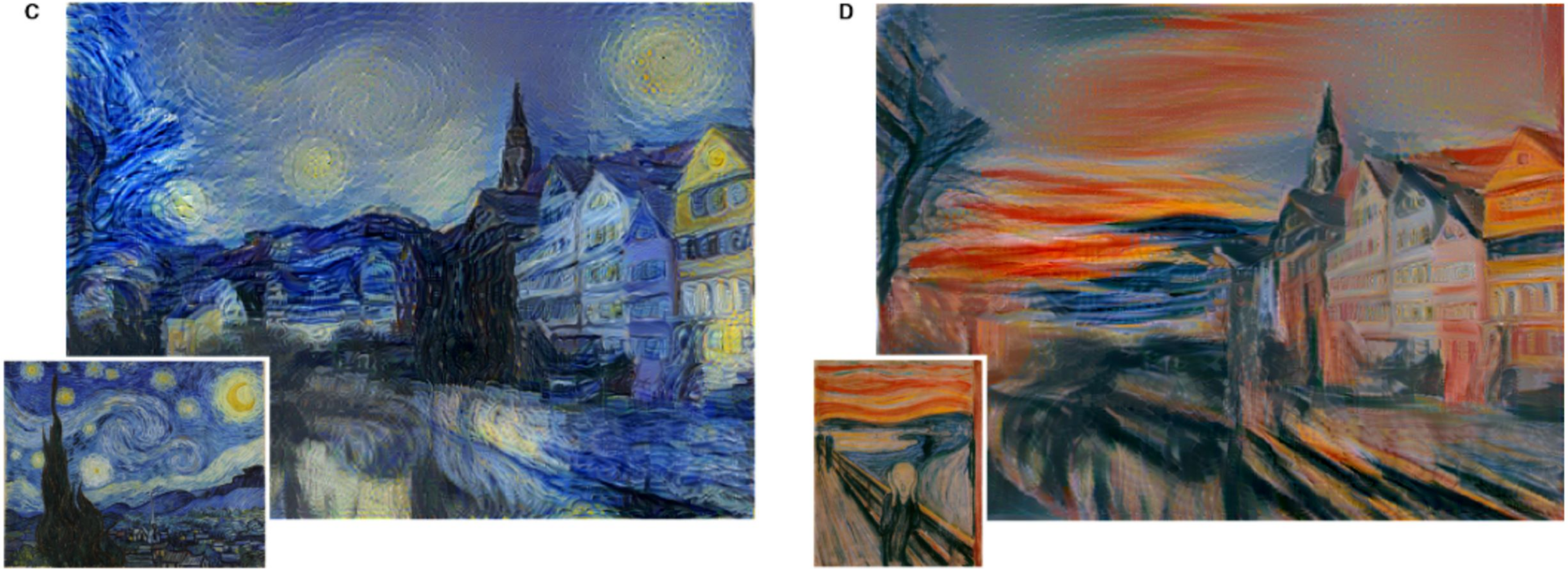
https://avisingh599.github.io/deeplearning/visual-qa/

CNNs - Style Transfer



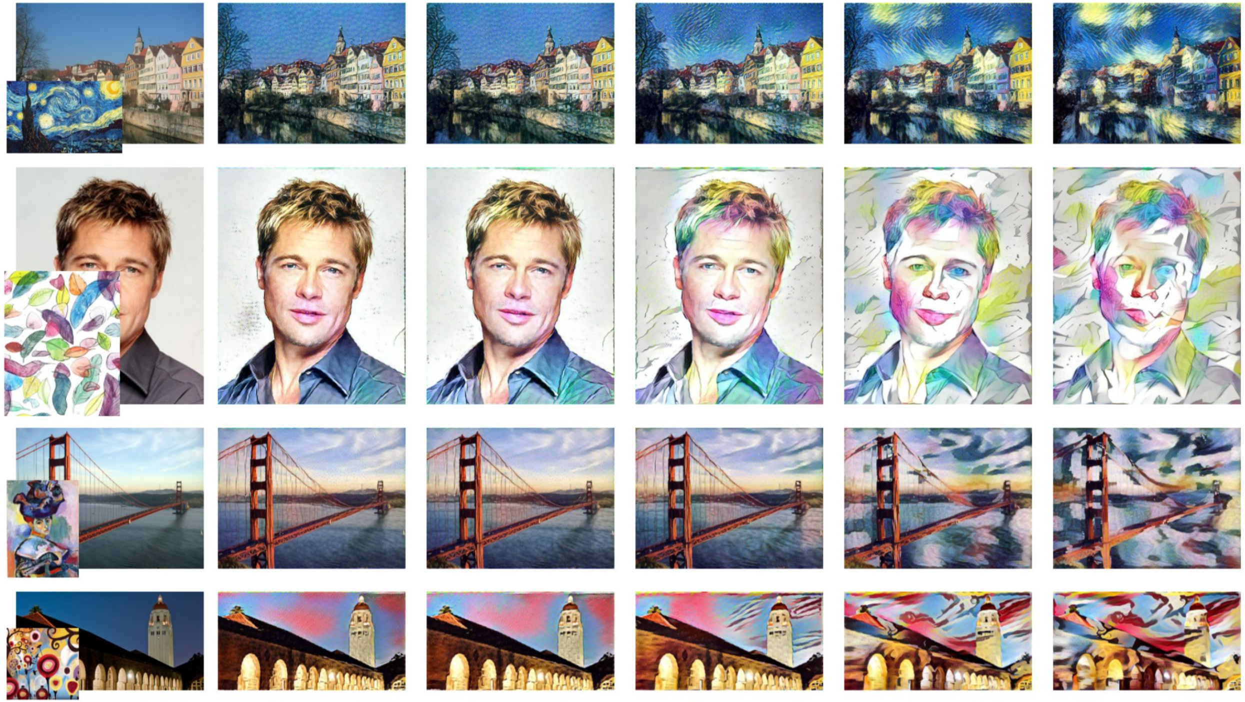
Leon A. Gatys, et. al., A Neural Algorithm of Artistic Style / https://www.youtube.com/watch?v=tZZrDN3qAk8

CNNs - Style Transfer



Leon A. Gatys, et. al., A Neural Algorithm of Artistic Style / https://www.youtube.com/watch?v=tZZrDN3qAk8

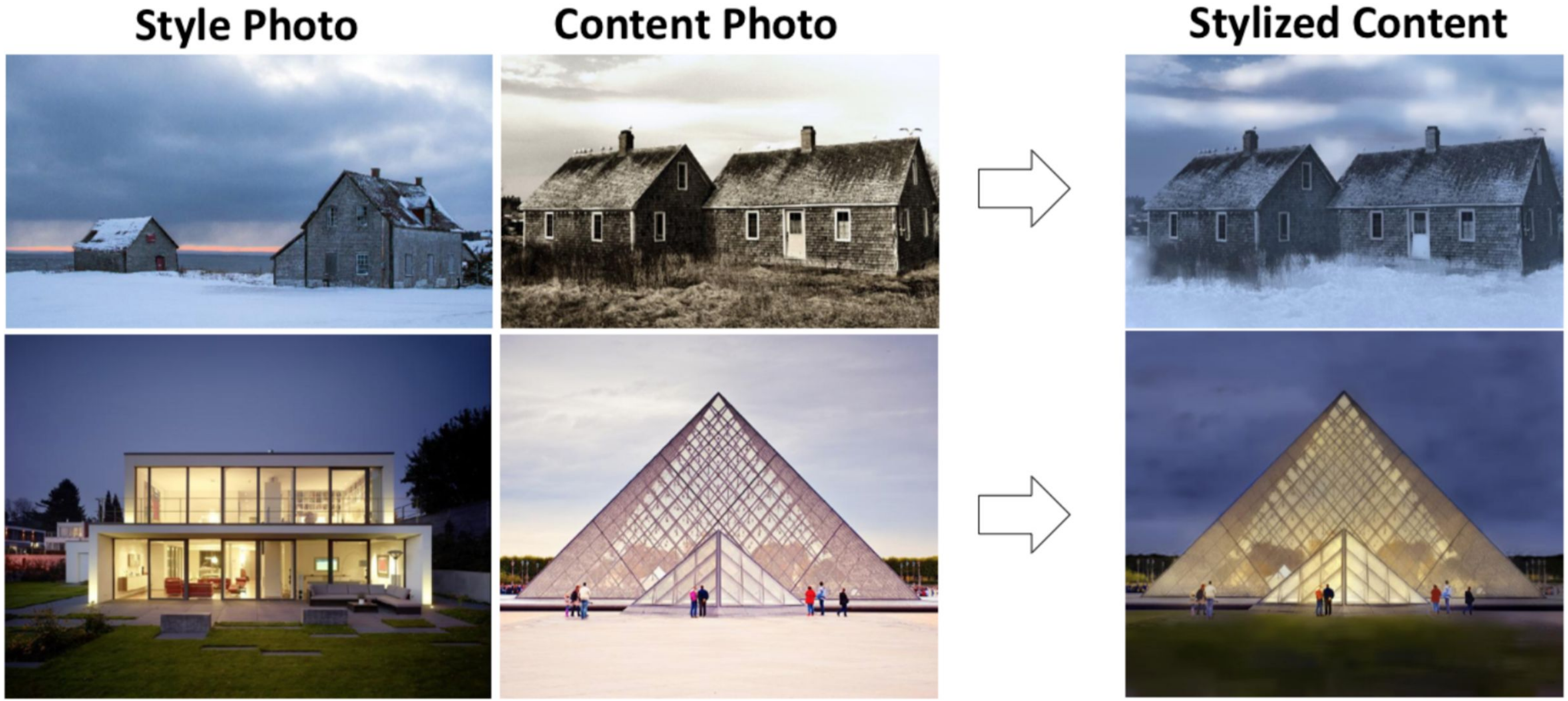
CNNs - Style Transfer



(a) Content / Style (b) $\gamma = 0.1$ (c) $\gamma = 0.2$ (d) $\gamma = 1.0$ (e) $\gamma = 5.0$ (f) $\gamma = 10.0$

Li, et. al., Demystifying Neural Style Transfer (https://arxiv.org/abs/1701.01036) / https://www.youtube.com/watch?v=Ut5WYGi5yRU

CNNs - Fast Photo Style Transfer



https://github.com/nvidia/fastphotostyle

CNNs - Fast Photo Style Transfer

https://github.com/luanfujun/deep-painterly-harmonization

CNNs - Fast Photo Style Transfer

https://www.youtube.com/watch?v=Dx59bskG8dc&t=24s

CNNs - Image colorization

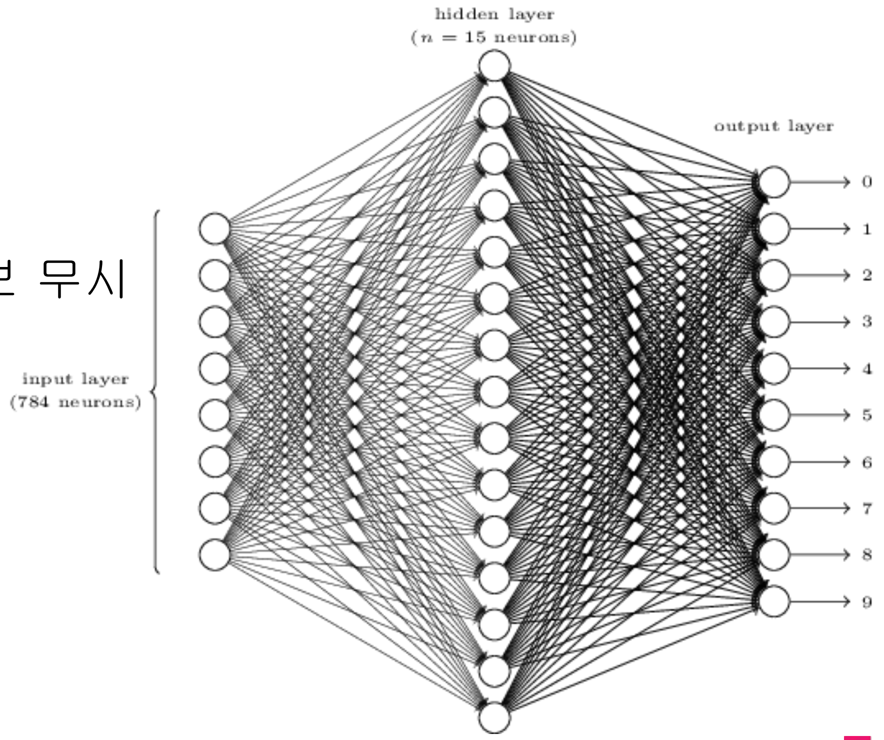
(a) Colorado National Park, 1941 (b) Textile Mill, June 1937 (c) Berry Field, June 1909 (d) Hamilton, 1936

Iizuka, et. al., Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification

Convolution Neural Networks

- Fully connected layers의 단점
  - 데이터 형상의 무시

    이미지의 공간적(spatial)한 정보 무시
  - 학습해야 할 가중치(W)가 많다



Convolution filters

박스

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

가우시안

.0000	.0000	.0002	.0000	.0000
.0000	.0113	.0837	.0113	.0000
.0002	.0837	.6187	.0837	.0002
.0000	.0113	.0837	.0113	.0000
.0000	.0000	.0002	.0000	.0000

샤프닝

0	-1	0
-1	5	-1
0	-1	0

수평 에지

1	1	1
0	0	0
-1	-1	-1

수직 에지

1	0	-1
1	0	-1
1	0	-1

모션

.0304	.0501	0	0	0
.0501	.1771	.0519	0	0
0	.0519	.1771	.0519	0
0	0	.0519	.1771	.0501
0	0	0	.0501	.0304

(a) 원래 영상과 여러 가지 마스크들

> 박스

> 가우시안

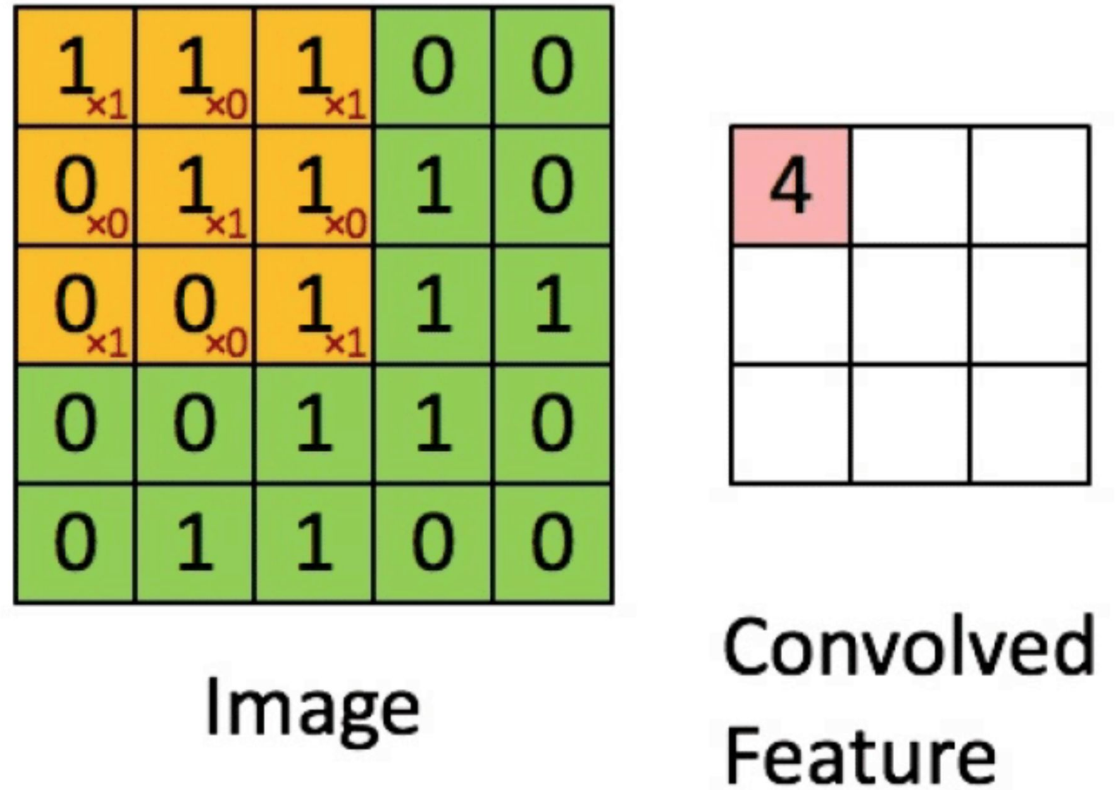
> 샤프닝

> 수평 에지

> 수직 에지

> 모션

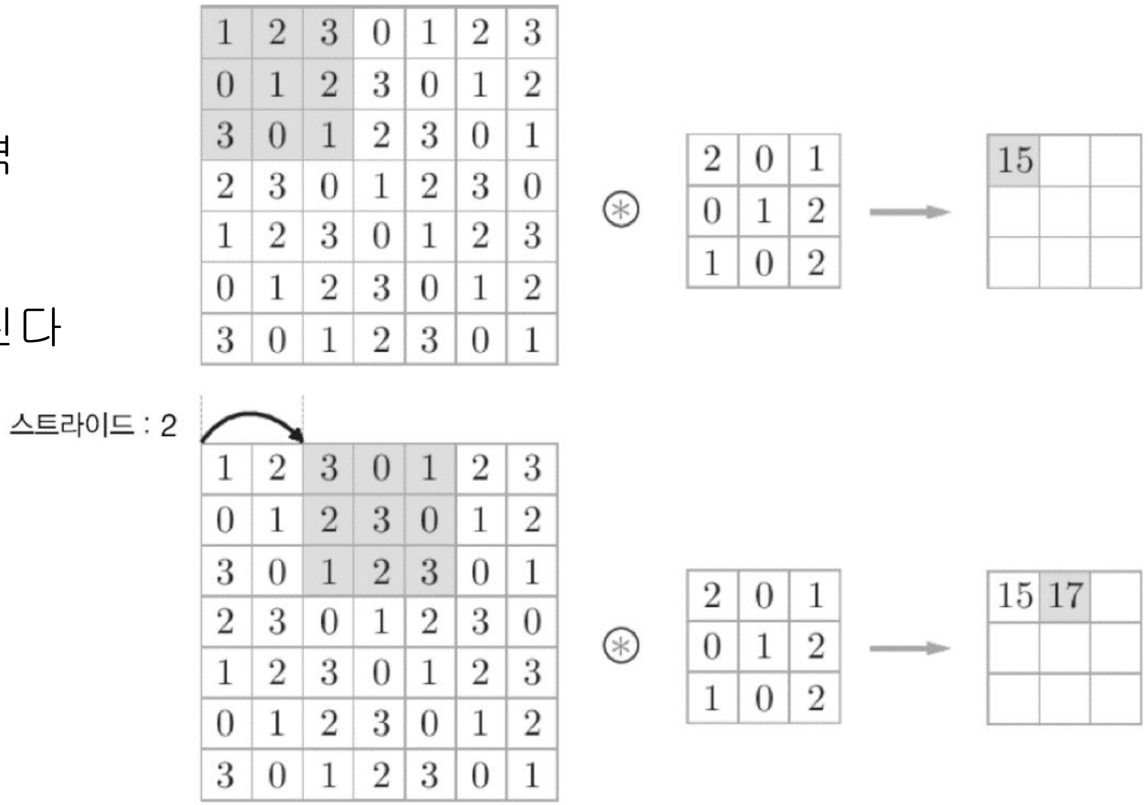
Convolution Neural Networks



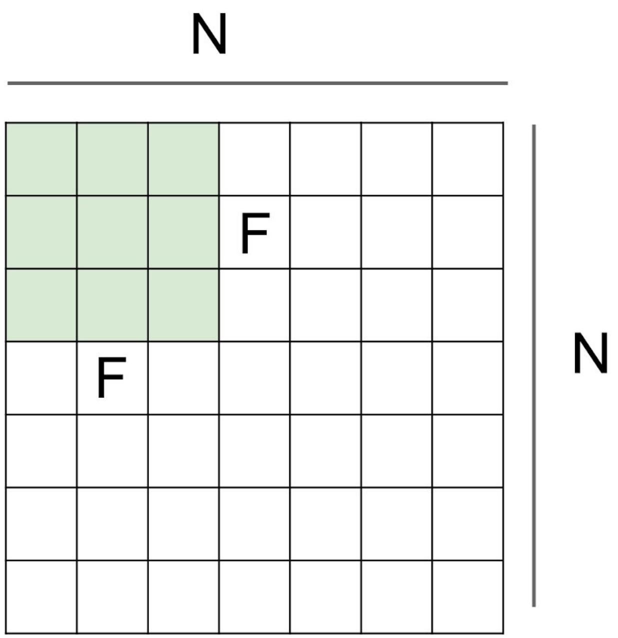
https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks

CNNs - Stride

- 필터를 적용하는 위치 간격
    - 스트라이드의 간격에 따라 출력 값이 달라진다



https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks



Output size:

(N - F) / stride + 1

e.g. N = 7, F = 3:

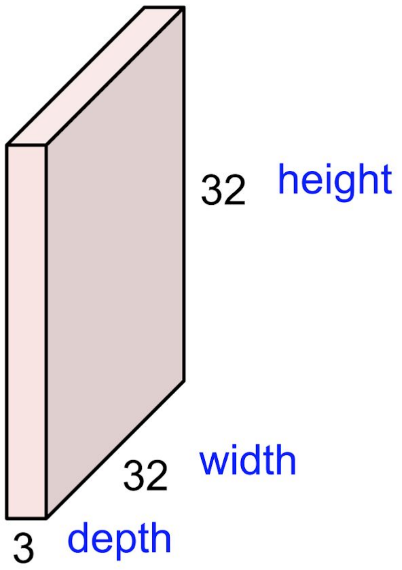
stride 1 => (7 - 3)/1 + 1 = 5

stride 2 => (7 - 3)/2 + 1 = 3

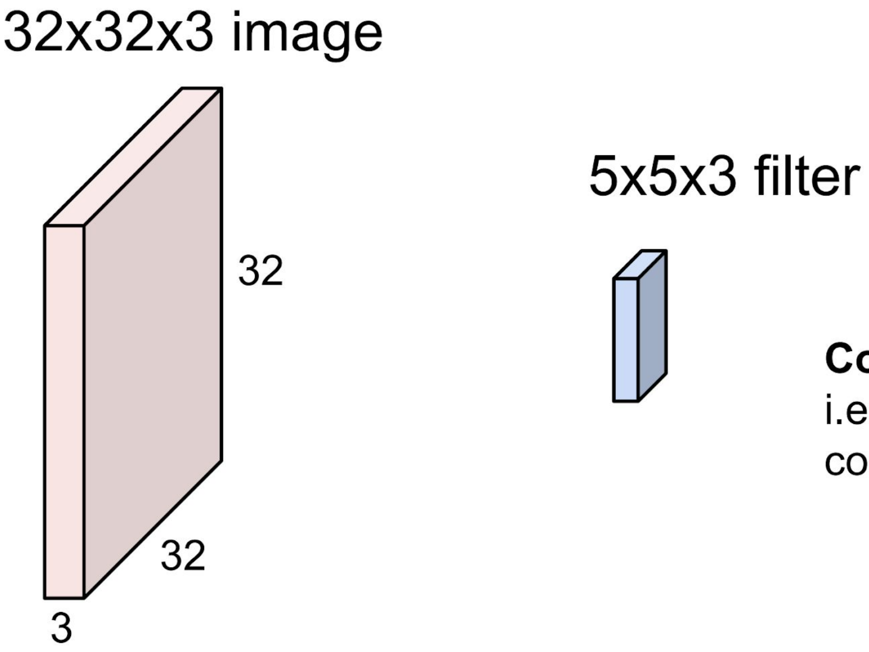
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

Convolution Layer

32x32x3 image -> preserve spatial structure



Convolution Layer

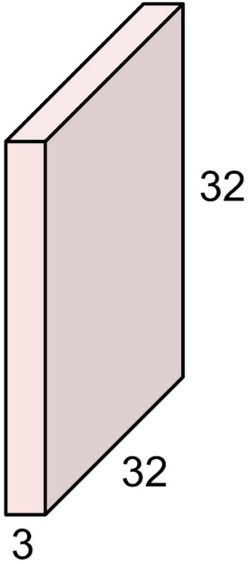


Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

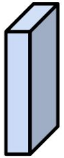
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

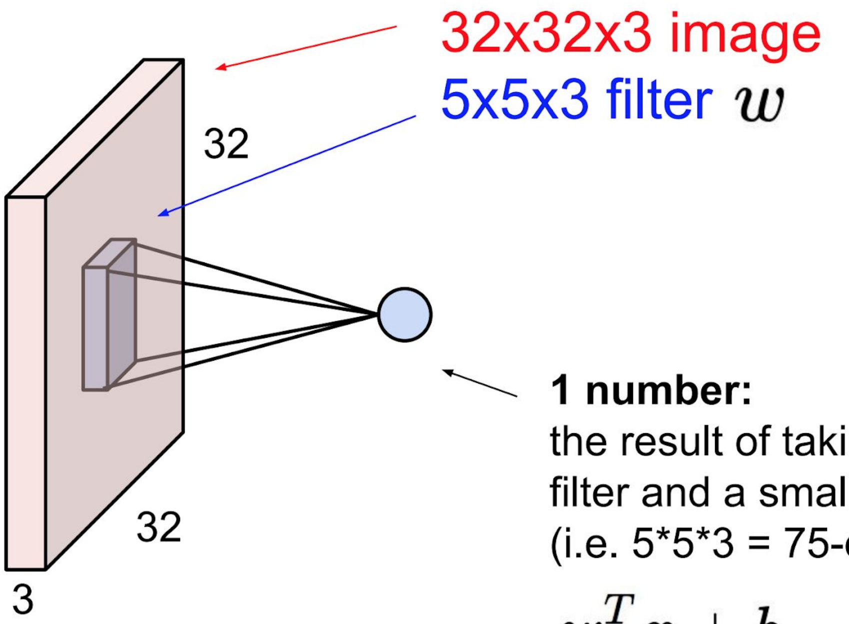


5x5x3 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer

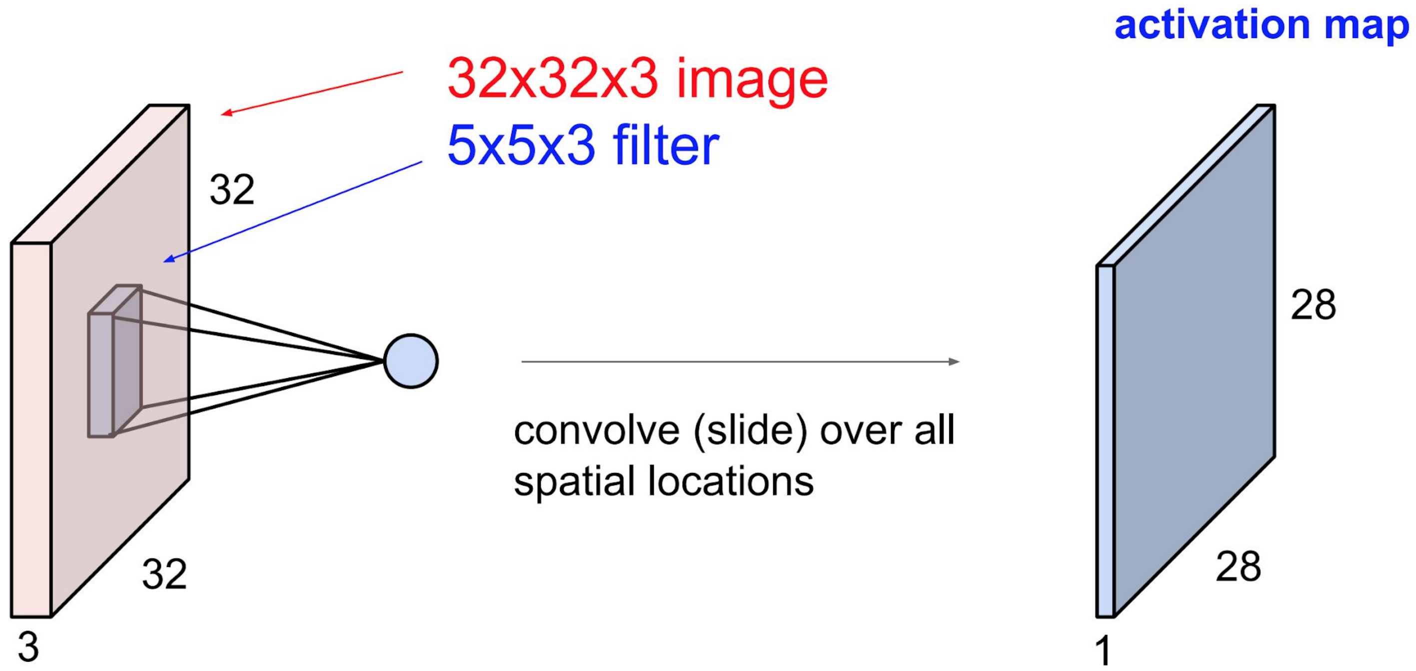


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture05.pdf

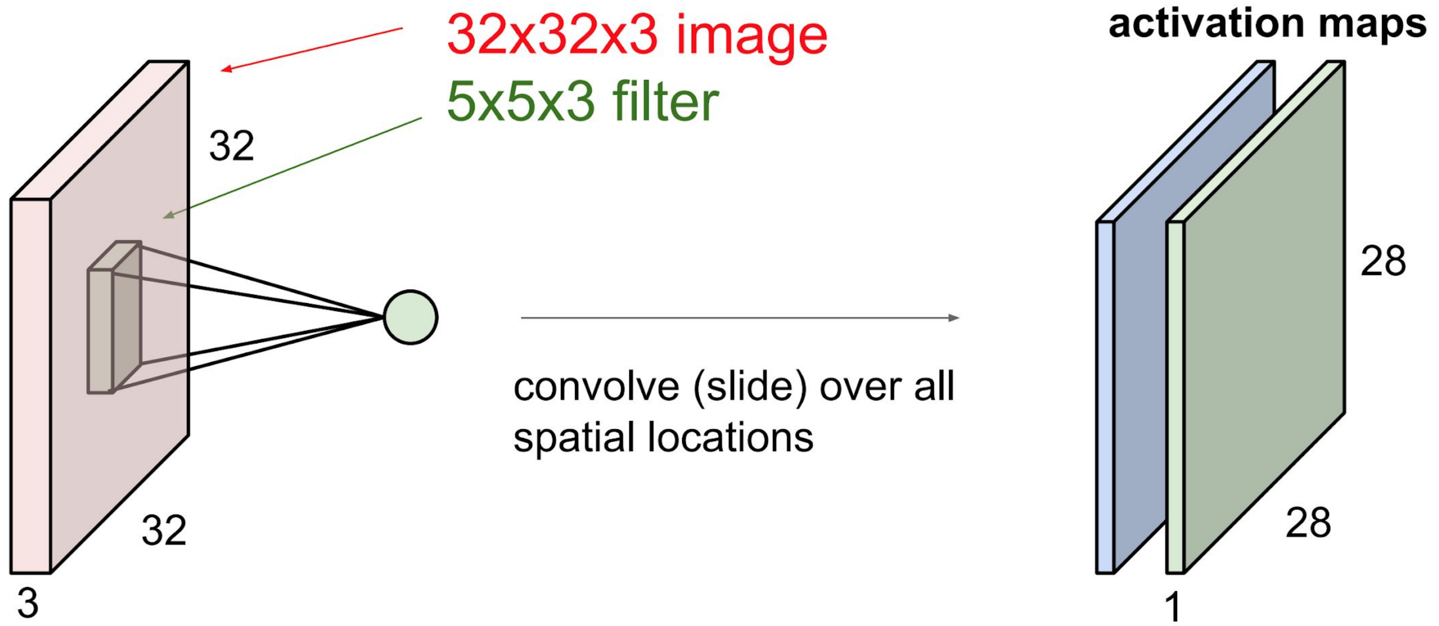
Convolution Layer



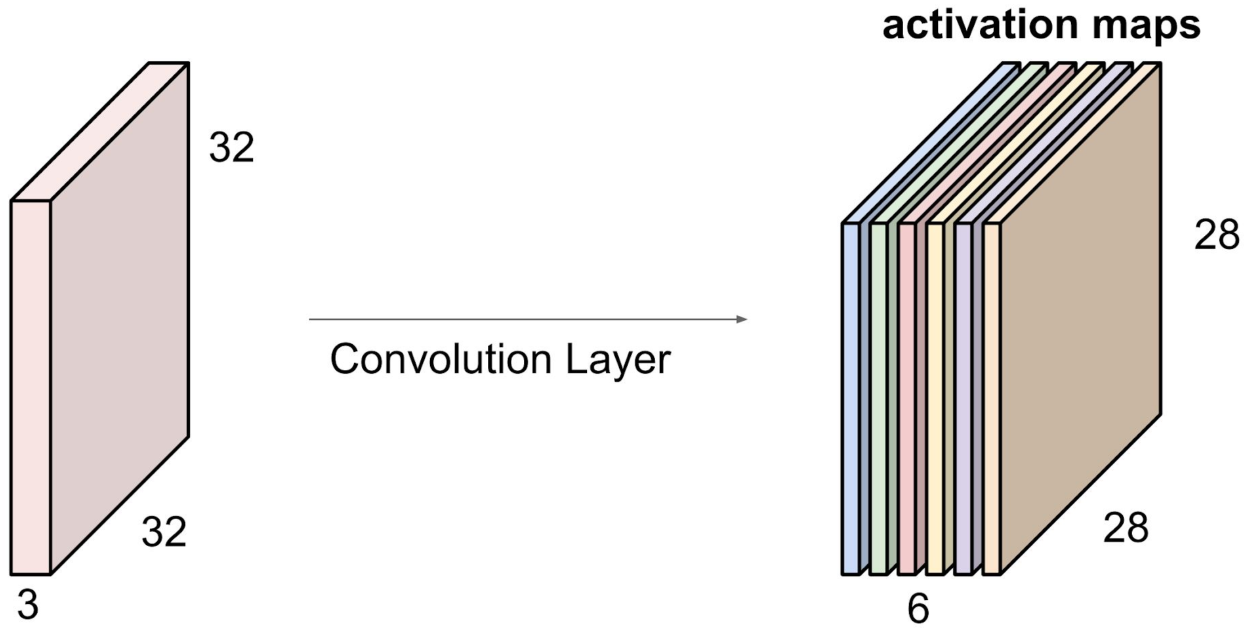
http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture05.pdf

Convolution Layer

consider a second, green filter

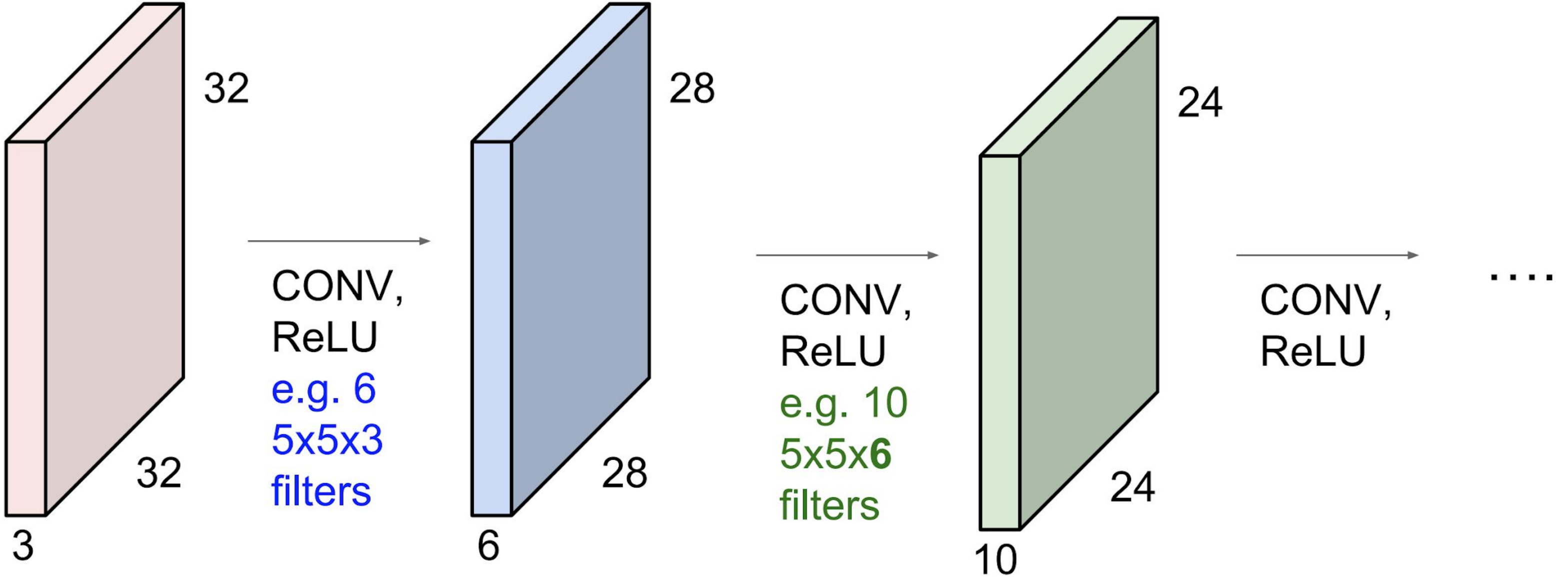


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



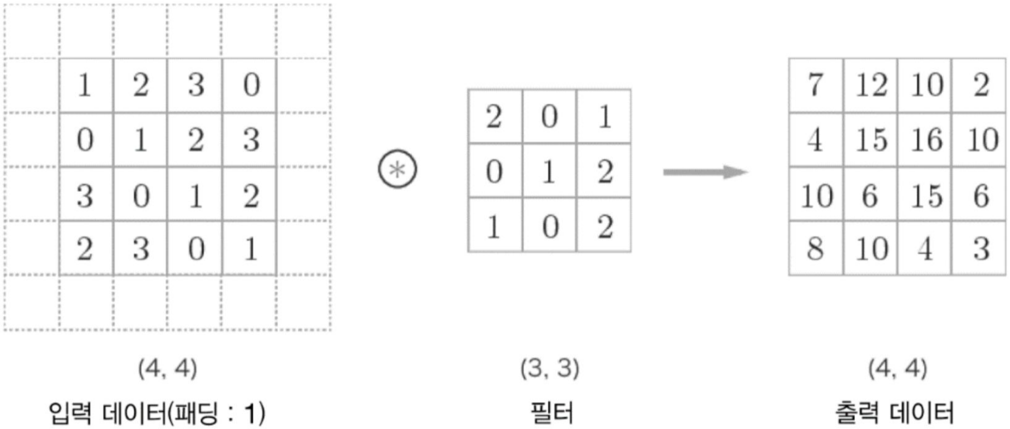
We stack these up to get a "new image" of size 28x28x6!

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



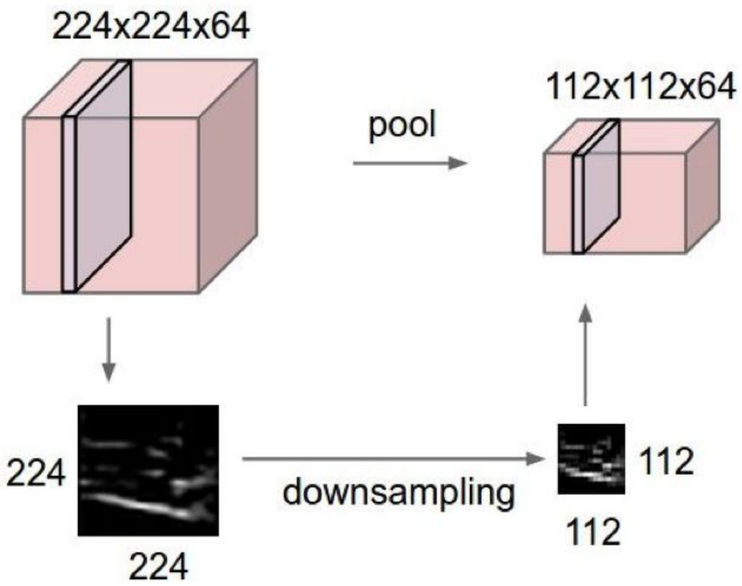
CNNs - Padding

- 입력 데이터 주변에 0으로 Padding 처리를 한다
- Padding은 출력의 크기를 유지하기 위해서 주로 사용한다

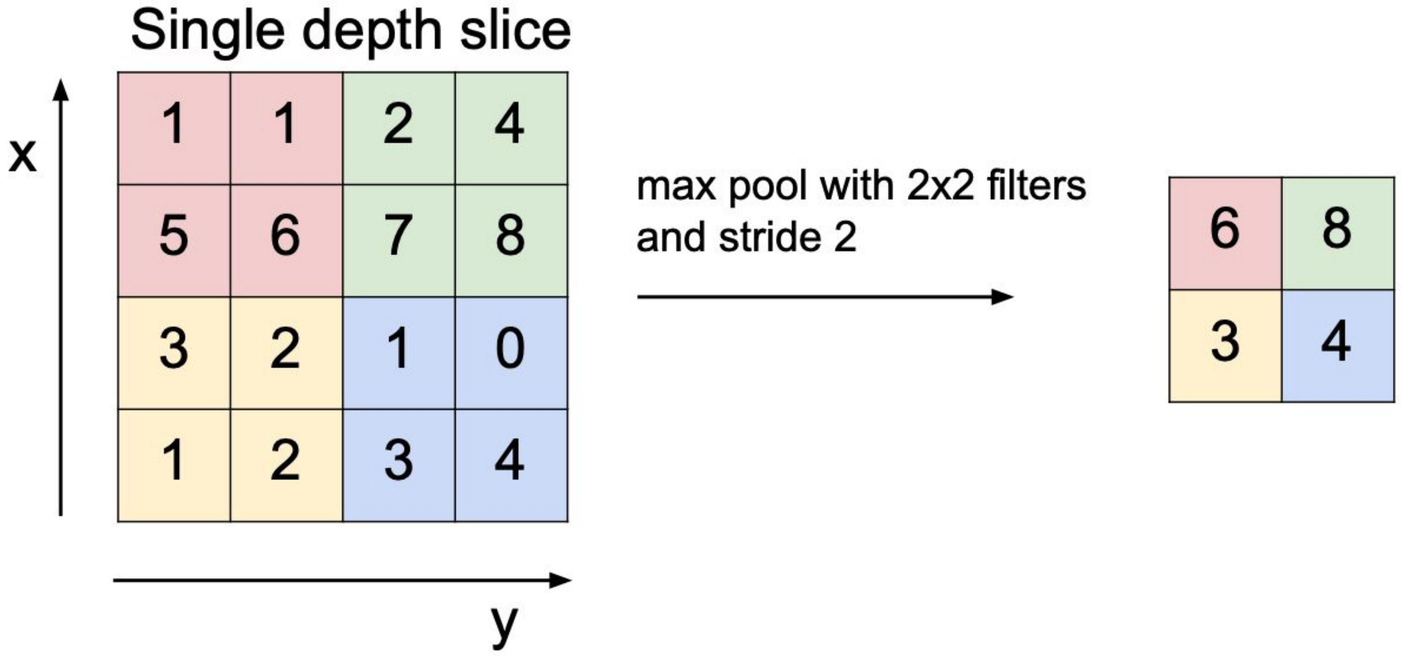


Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:

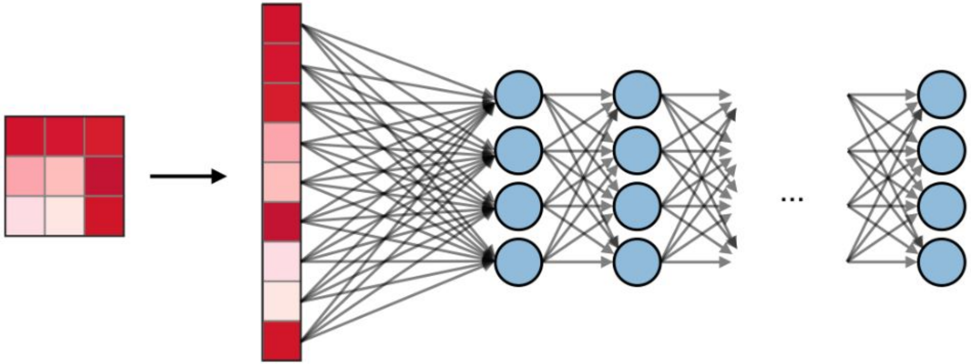


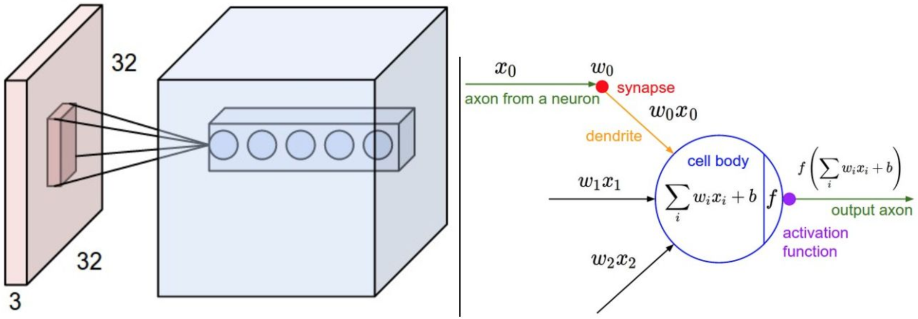
MAX POOLING



http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture05.pdf

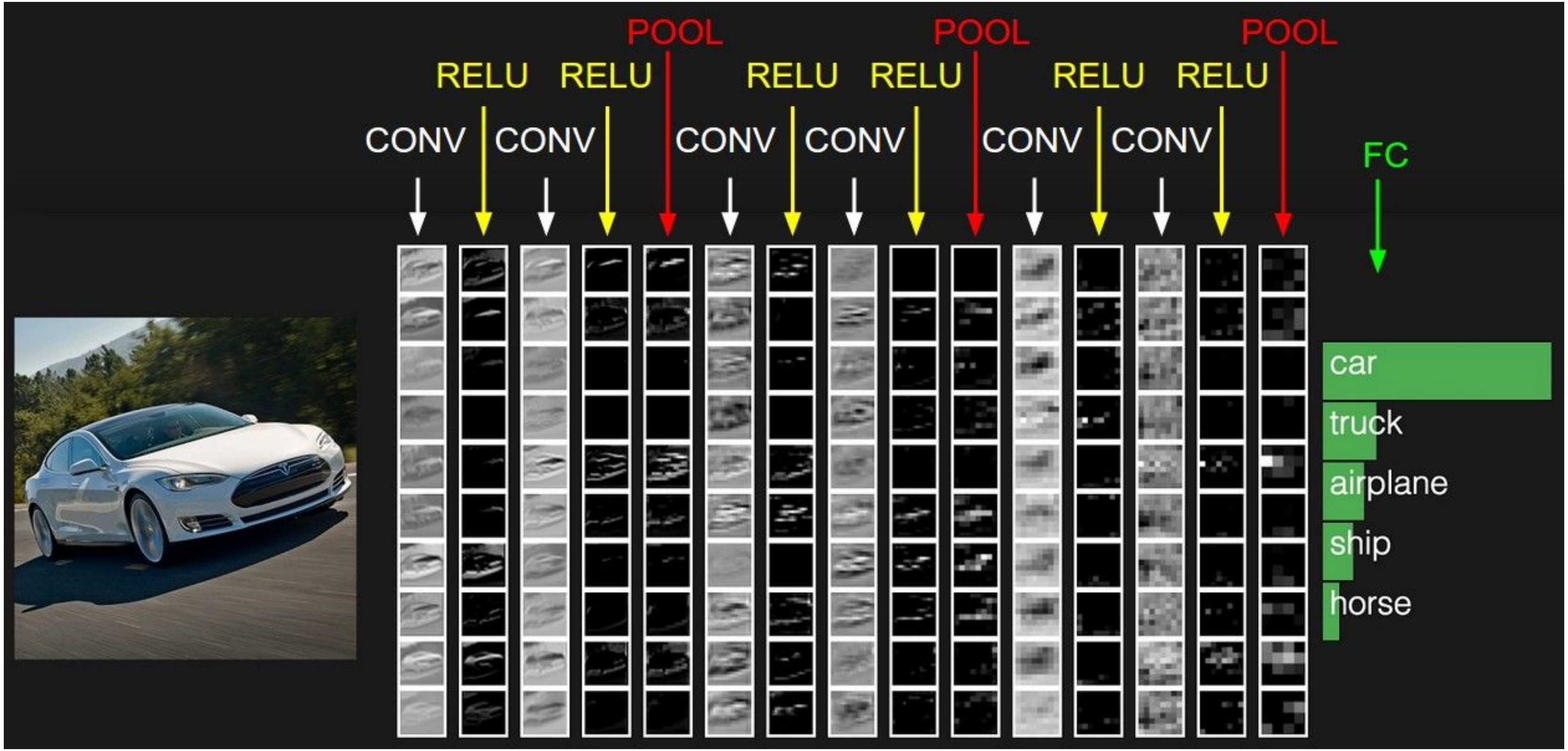
Convolutional Neural Networks





https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks
http://cs231n.github.io/convolutional-networks/

Convolutional Neural Networks

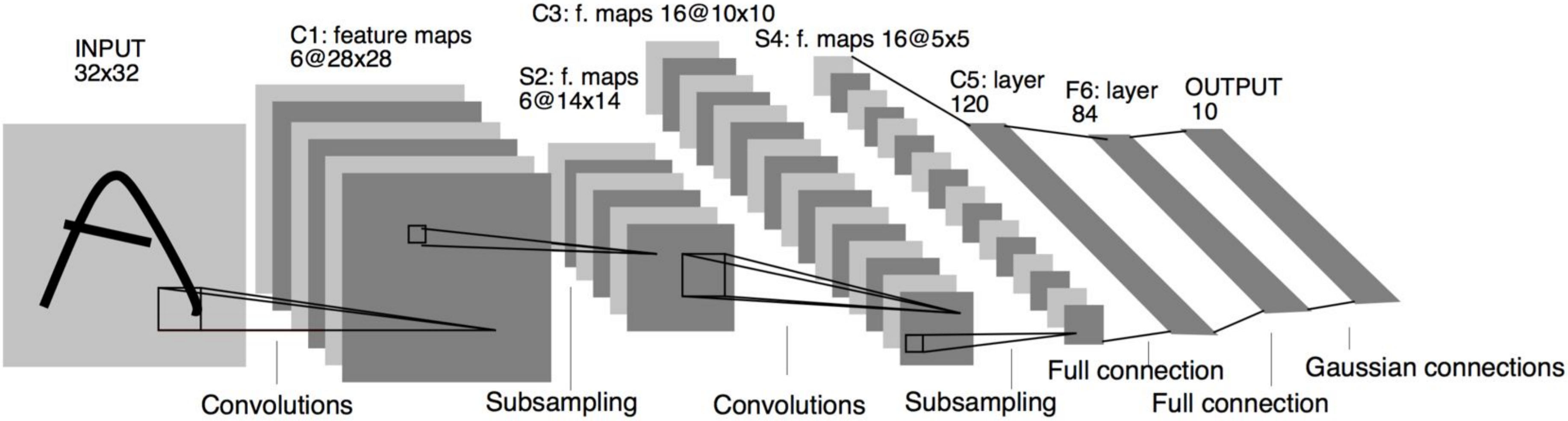


http://cs231n.github.io/convolutional-networks/

Convolutional Neural Networks

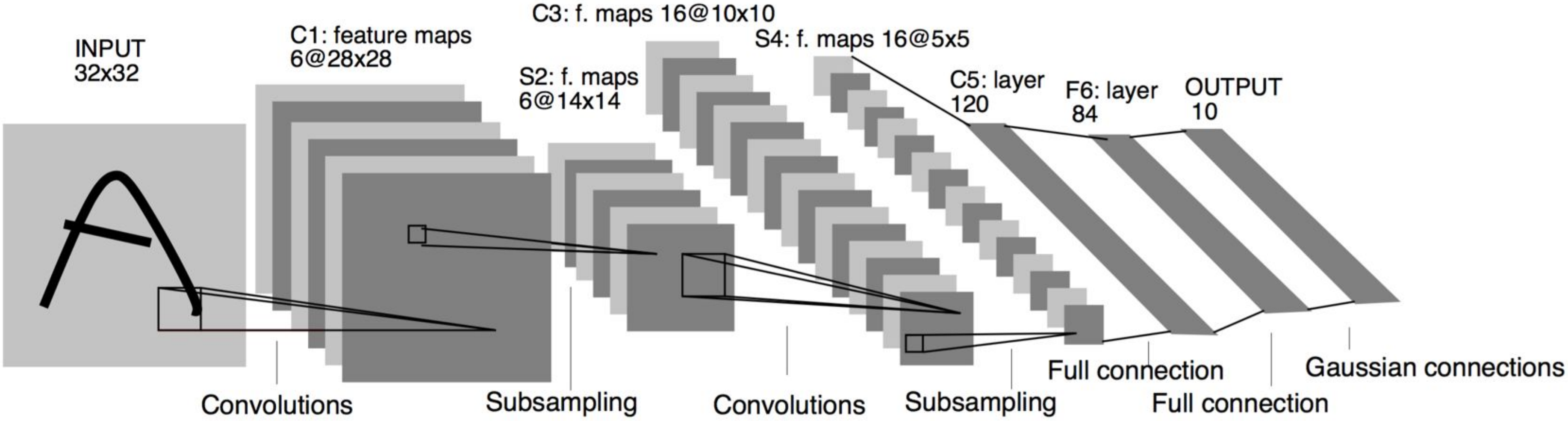
- 풀고자 하는 문제에 따라서 각각의 네트워크를 구성할 수 있다

Case Study - LeNet-5



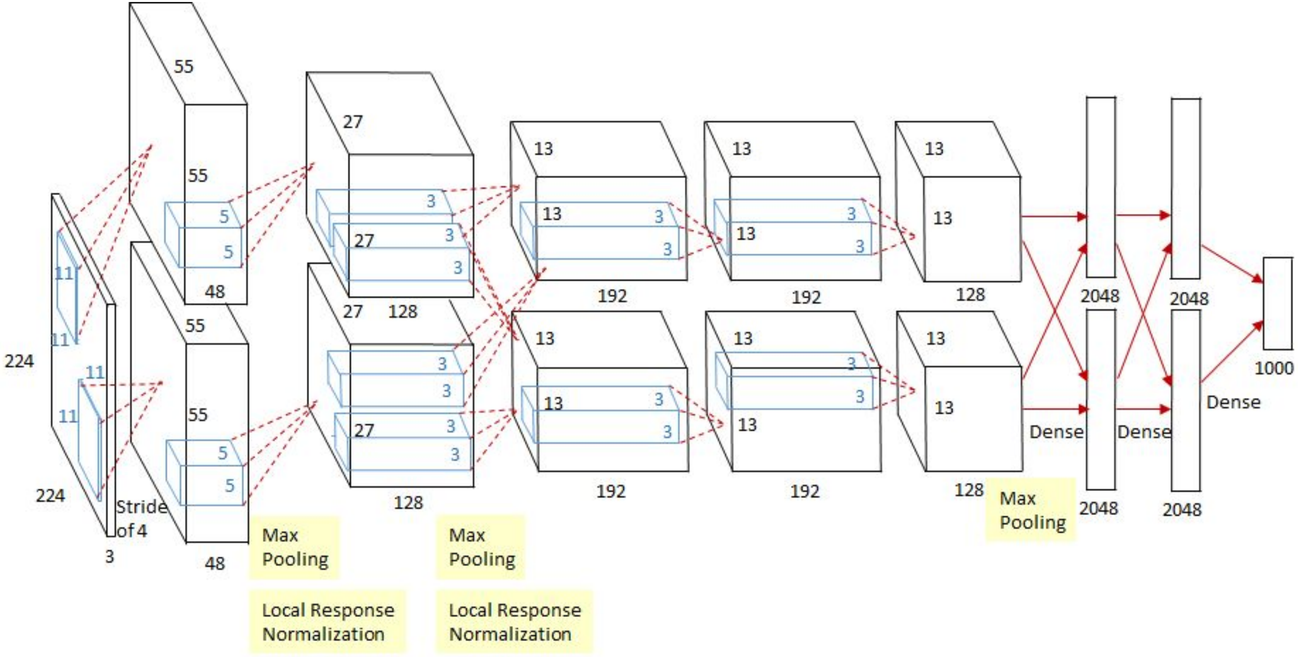
Gradient-based learning applied to document recognition
[LeCun, Bottou, Bengio, Haffner 1998]

Case Study - LeNet-5



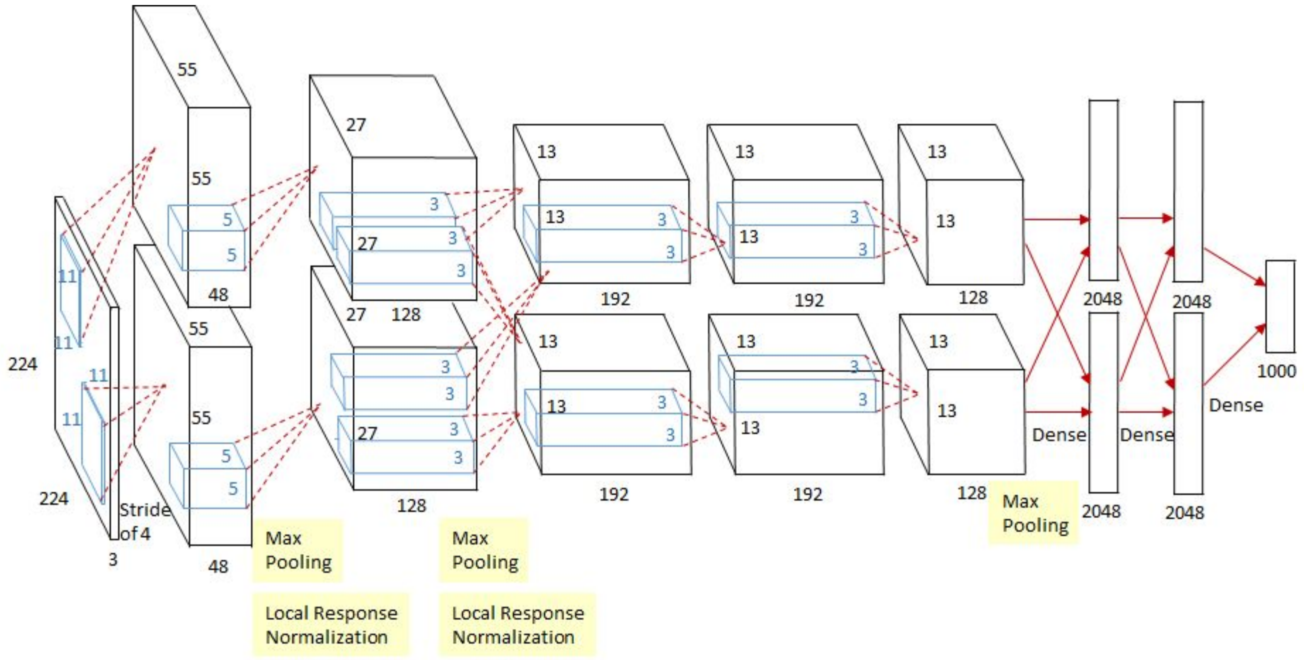
- CNN의 시초가 된 모델
- Activation으로 Sigmoid 사용

Case Study - AlexNet



ImageNet Classification with Deep Convolutional Neural Networks
[Krizhevsky, Sutskever, Hinton, 2012]

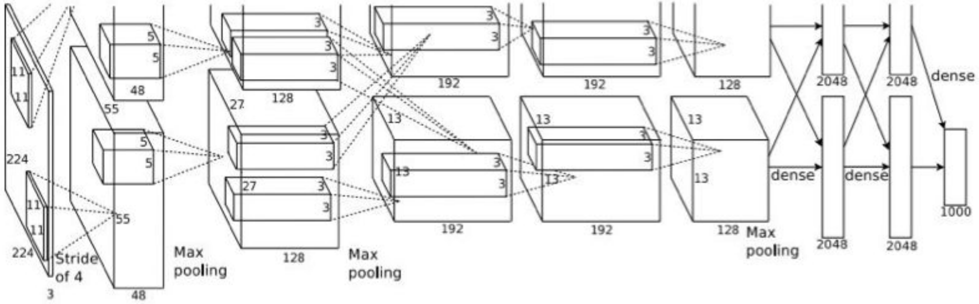
Case Study - AlexNet



- ReLU 함수 적용, Dropout 적용,
  GPU 계산 등 Deep Learning 발전의 시초가 됨

Case Study: AlexNet

[Krizhevsky et al. 2012]



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

ImageNet Classification with Deep Convolutional Neural Networks [Krizhevsky, Sutskever, Hinton, 2012]

Case Study: VGGNet

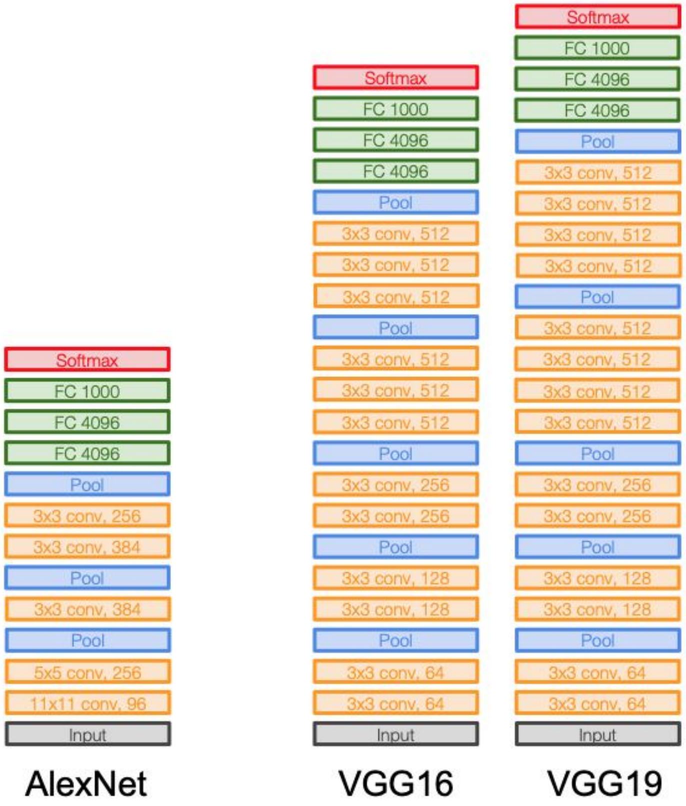
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13
(ZFNet)
-> 7.3% top 5 error in ILSVRC'14

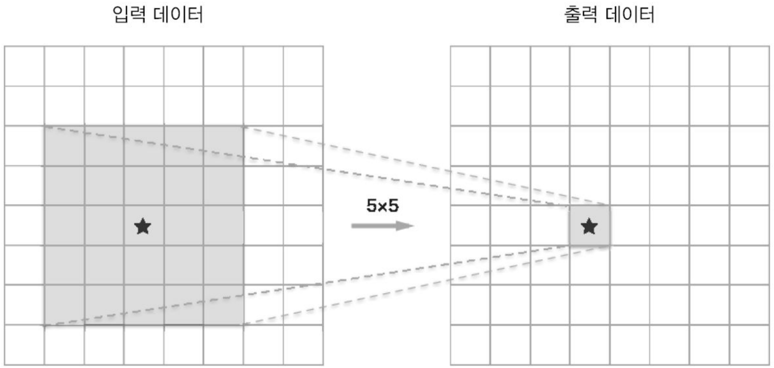


Very deep convolutional networks for large-scale image recognition [Simonyan and Zisserman, 2014]

Case study - VGG Net

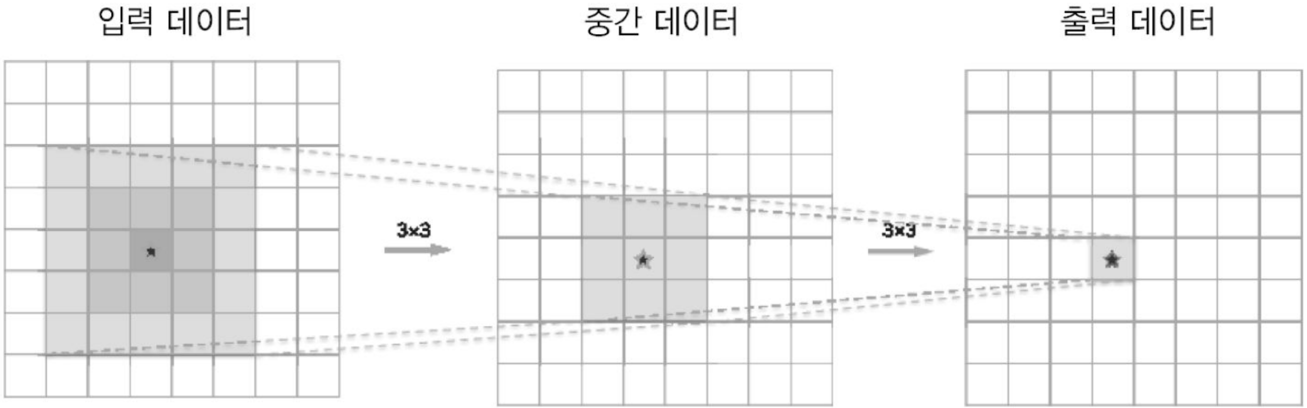
- AlexNet(8 layers)에 비해 두 배 이상 깊어짐 (16, 19 layers)
  - 단순한 Conv Layer (3 x 3)만 이용
  - 큰 필터보다 작은 필터를 여러번 사용하는 것이 더 좋음
- Conv-Pooling 구조 대신 Conv-Conv-Conv-Pooing 구조를 사용

Case study - Filter 5x5 vs (3x3)x2



- 5x5 = 25

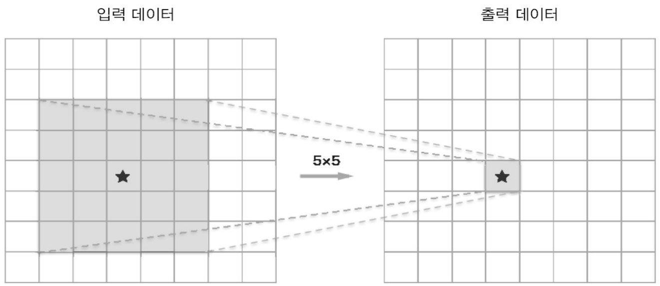
- (3x3) x 2 = 18

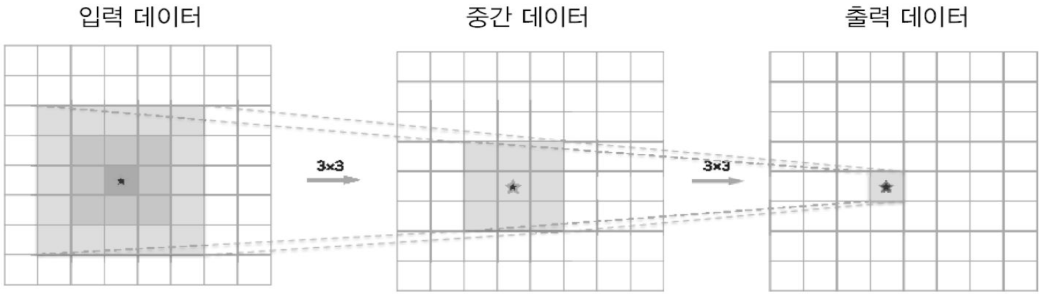


Case study - Filter 5x5 vs (3x3)x2

- 5x5 와 (3x3)x2 필터는 같은 영역을 처리
- 층이 깊어져 ReLU 같은 Activation function(비선형성)이 추가 된다

  => 비선형 함수가 겹쳐지면 더 복잡한 것도 표현 가능





INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

VGG16

Common names

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

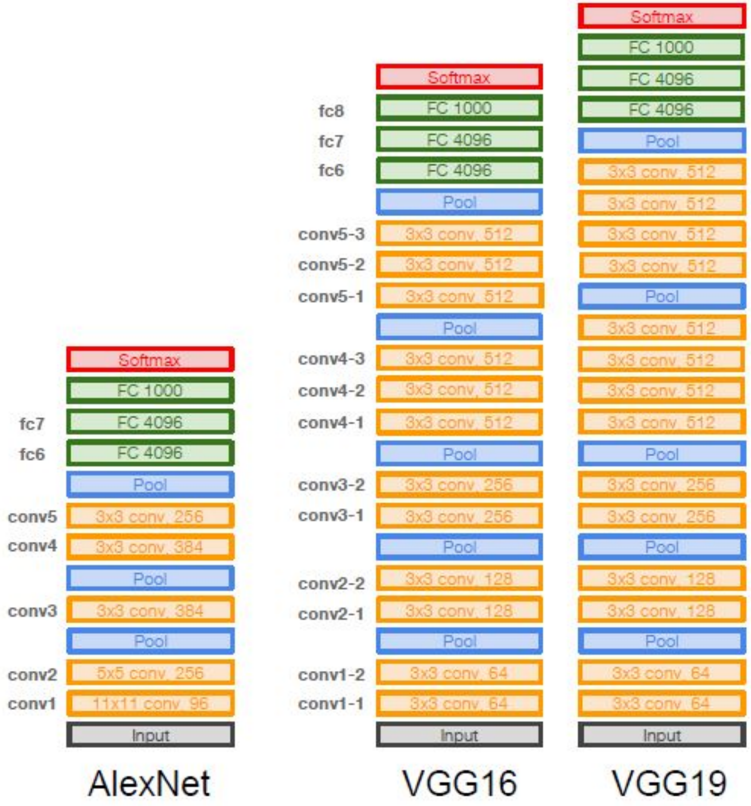
Very deep convolutional networks for large-scale image recognition [Simonyan and Zisserman, 2014]

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

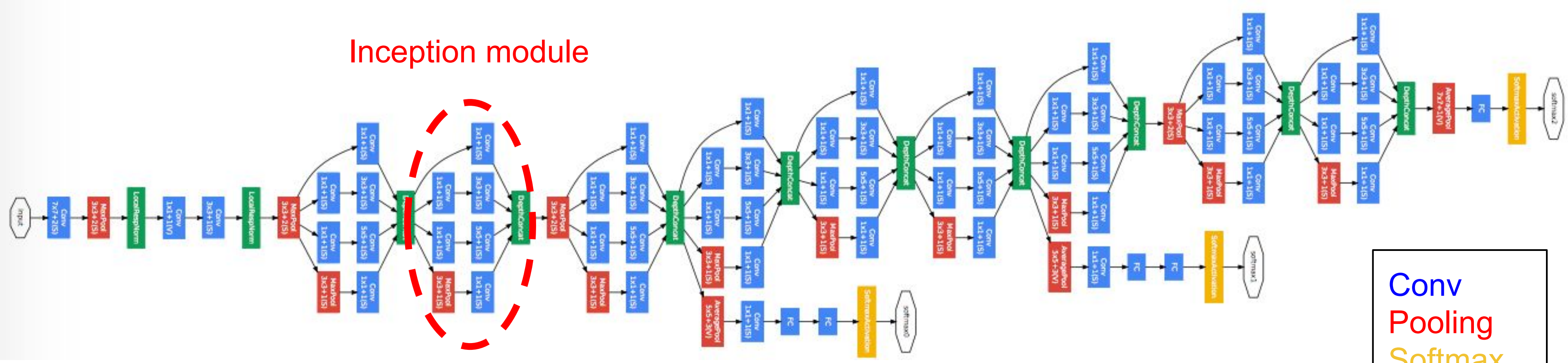
Details:

- ILSVRC'14 2nd in classification, 1st in localization
- Similar training procedure as Krizhevsky 2012
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks



Very deep convolutional networks for large-scale image recognition [Simonyan and Zisserman, 2014]

Case study - GoogLeNet



Conv
Pooling
Softmax
Concat

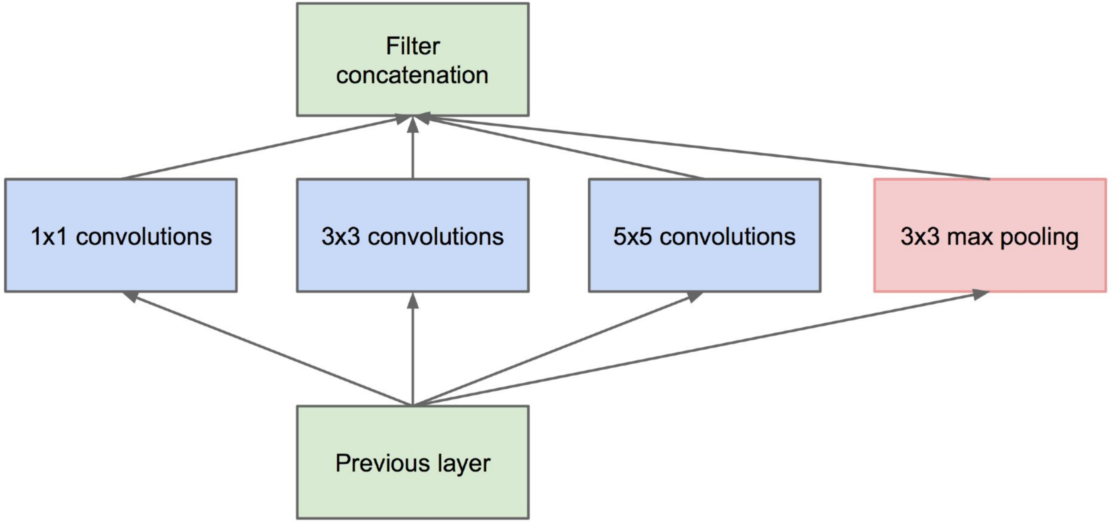
- Inception 모듈 사용
- 22개의 layers, 5M parameters(AlexNet의 1/12 수준)

Going deeper with convolutions [Szegedy et al., 2014]

Case study - Naive Inception Module

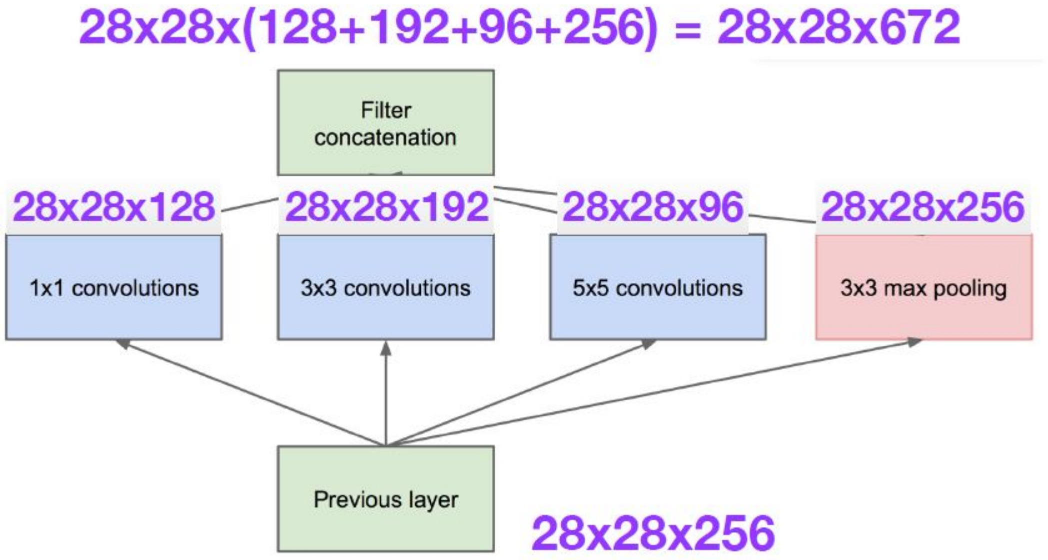
- 여러 크기의 필터를 동시에 적용
    - 다양한 receptive field를 학습

      성능을 개선



Going deeper with convolutions [Szegedy et al., 2014]

Case study - Naive Inception Module



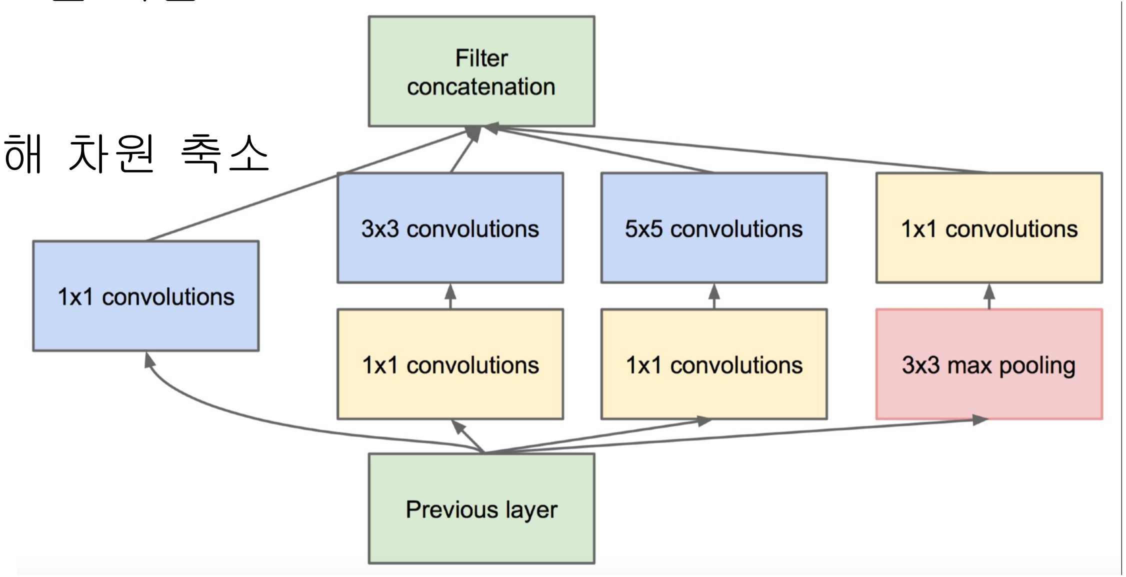
연산량이 많다

Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: 854M ops

Going deeper with convolutions [Szegedy et al., 2014]

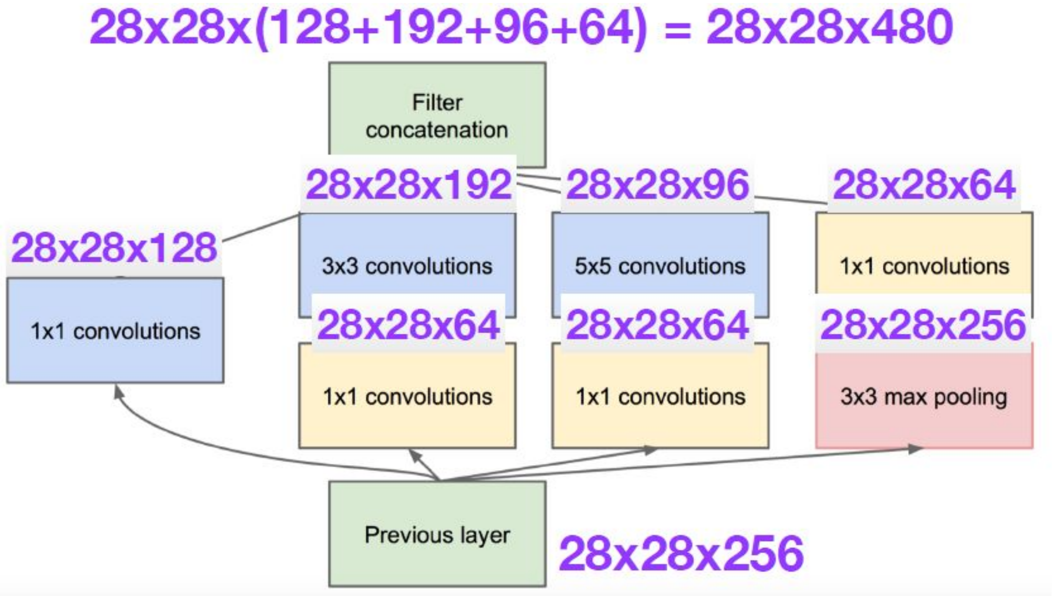
Case study - Naive Inception Module

- 여러 크기의 필터를 동시에 적용
  - 다양한 receptive field를 학습
    성능을 개선
- 1x1 Convolution을 이용해 차원 축소
  - 계산량 감소



Going deeper with convolutions [Szegedy et al., 2014]

Case study - Naive Inception Module

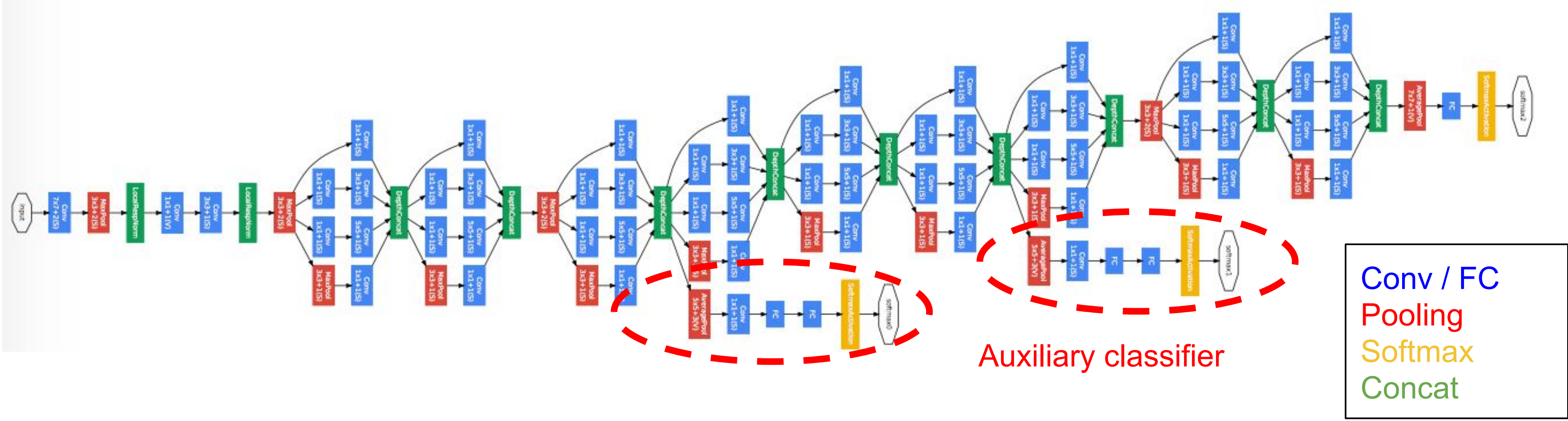


Conv Ops:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops

연산량 감소

Going deeper with convolutions [Szegedy et al., 2014]

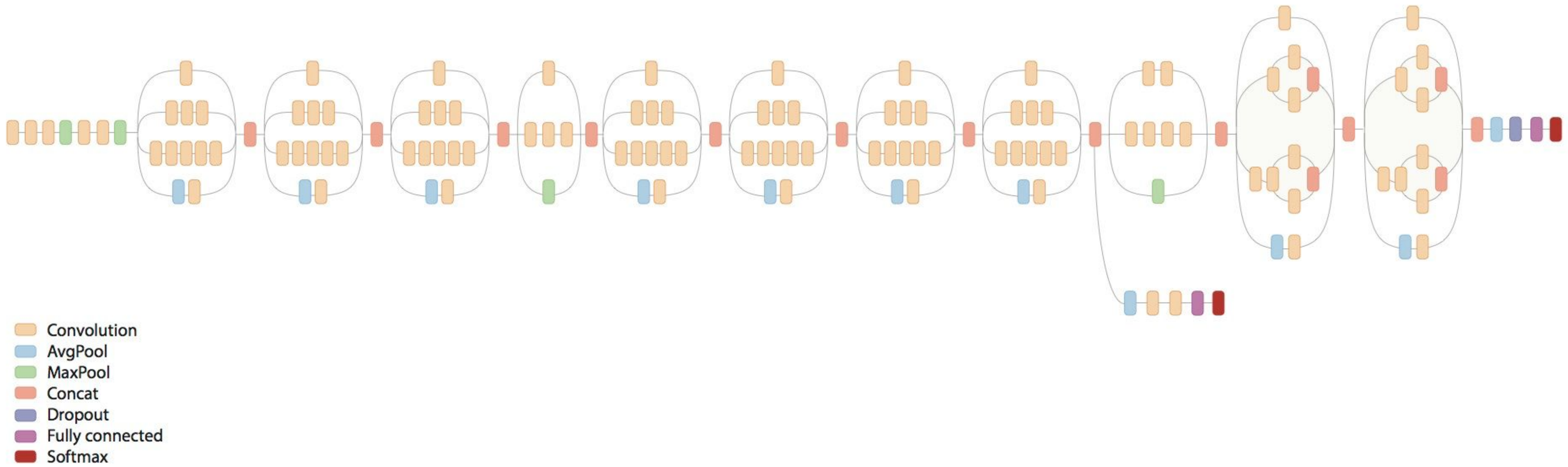
Case study - GoogLeNet



- 네트워크 중간 중간 classifier를 만들어 Vanishing gradient를 막는다.

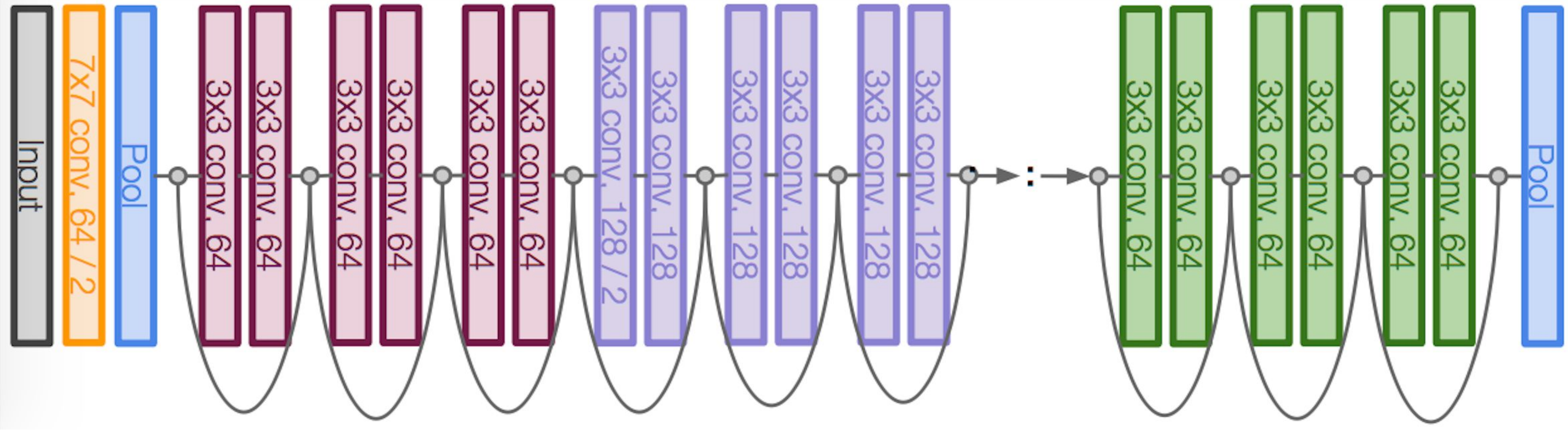
Going deeper with convolutions [Szegedy et al., 2014]

Case study - Inception-V3



Rethinking the Inception Architecture for Computer Vision [Szegedy et al., 2015]

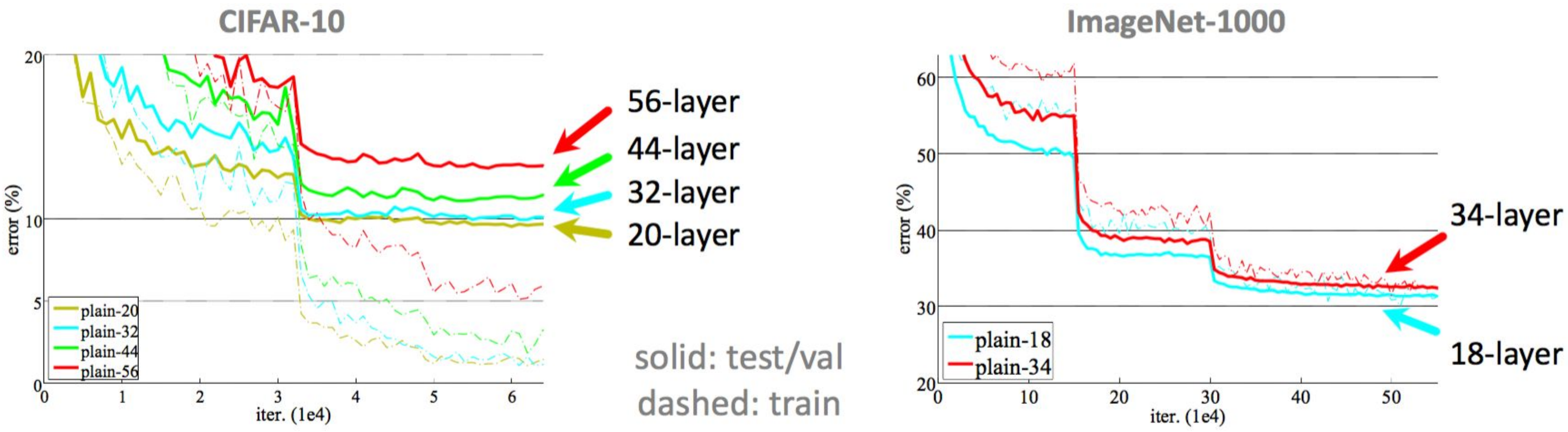
Case study - ResNet



- Residual Block 이라는 새로운 모듈 제안
- 152 Layers 이상 학습 가능

Deep Residual Learning for Image Recognition [He et. al. 2015]

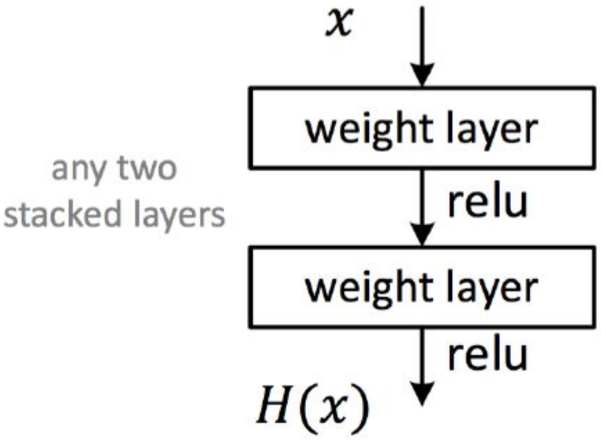
Case study - ResNet

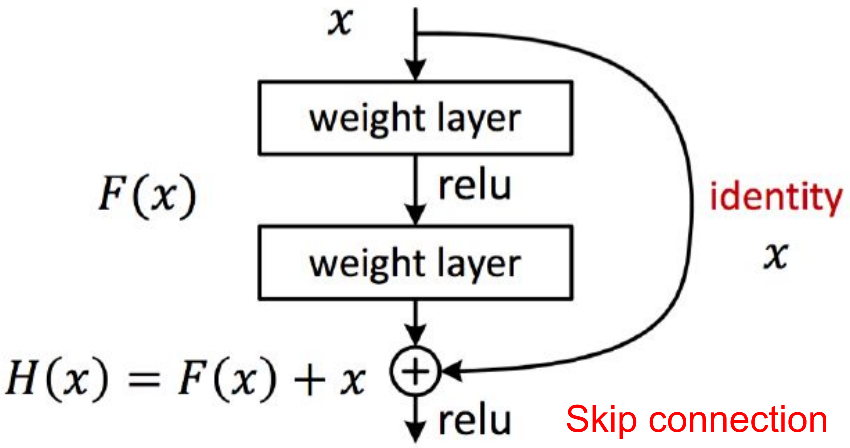


- 단순 Layer 갯수 증가로는 성능이 좋아지지 않는다.

Deep Residual Learning for Image Recognition [He et. al. 2015]

Case study - ResNet

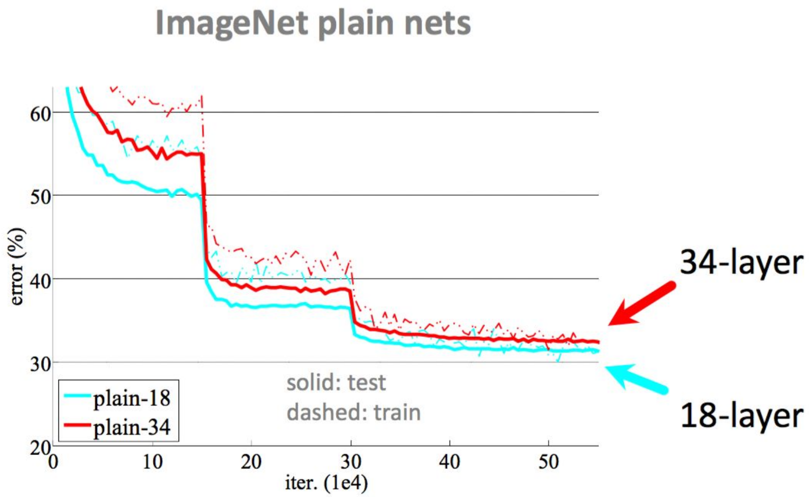


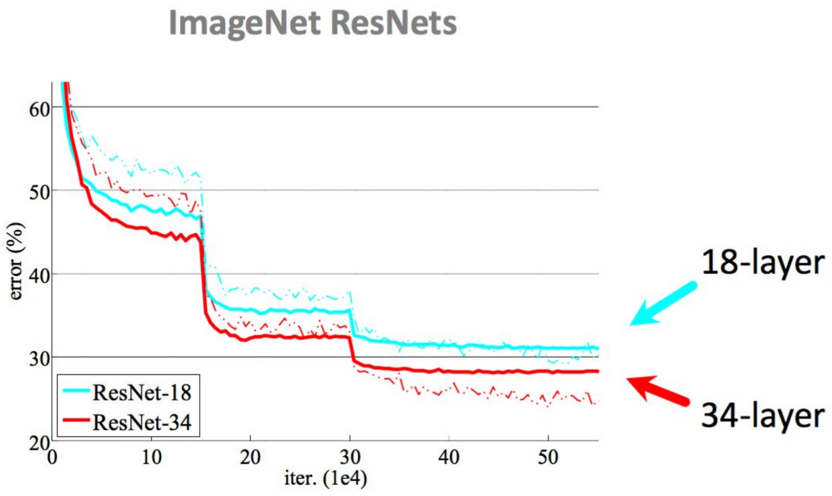


- 중간에 layer를 뛰어넘는 연결을 추가 -> 역전파시 Gradient가 그대로 전달
- Vanishing Gradient 를 줄여준다.

Deep Residual Learning for Image Recognition [He et. al. 2015]

Case study - ResNet

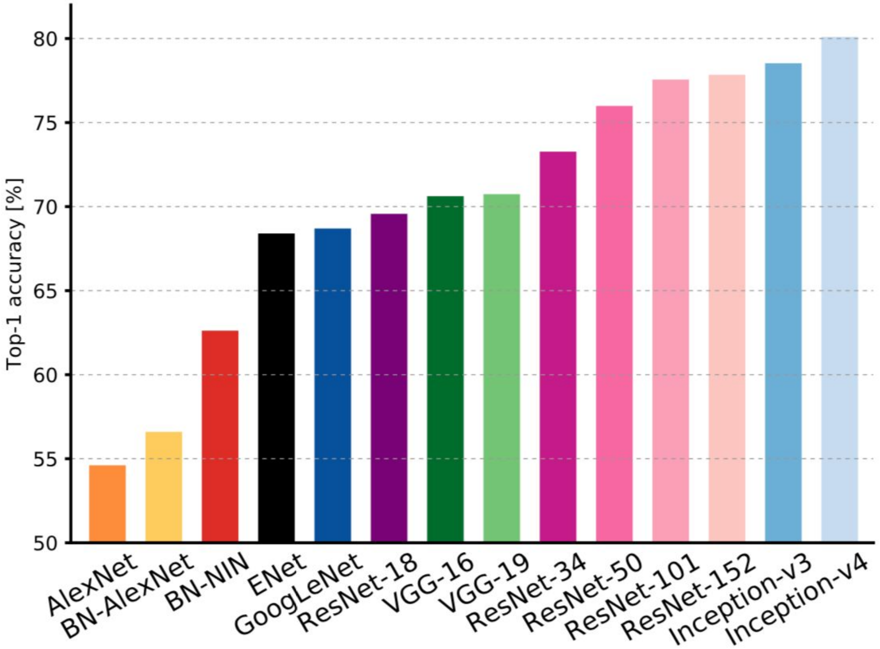


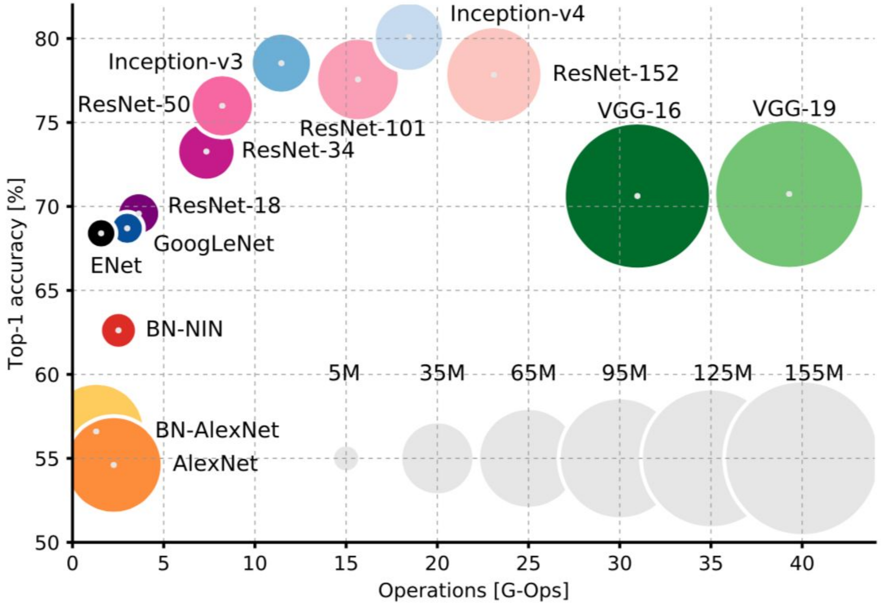


- 네트워크가 깊어져도 성능이 좋아진다.
- ILSVRC 2015 winners

Deep Residual Learning for Image Recognition [He et. al. 2015]

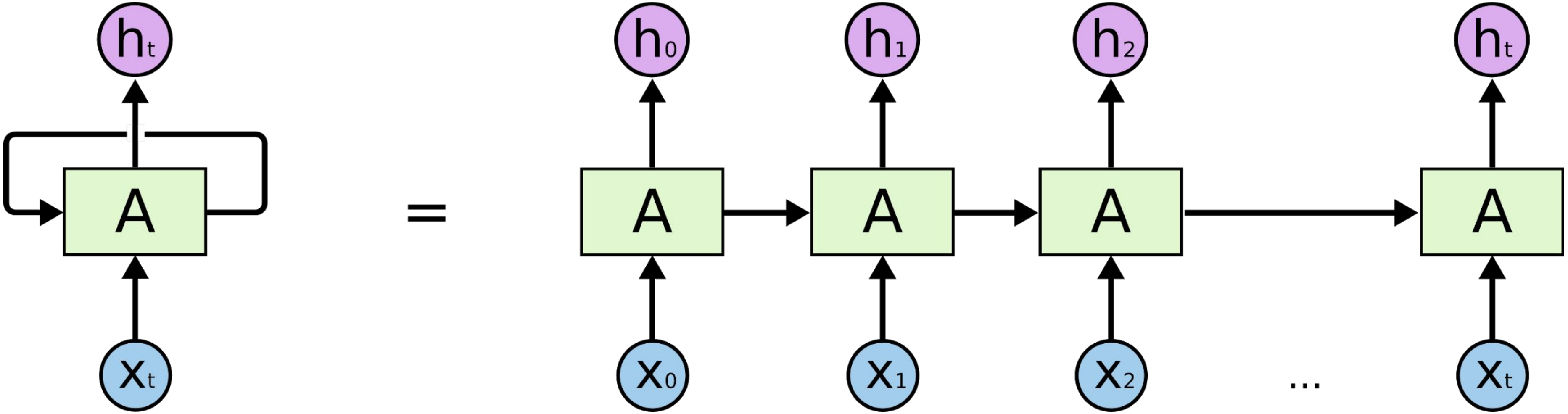
Case study - Comparing Complexity





Canziani et. al., An Analysis of Deep Neural Network Models for Practical Applications

Recurrent Neural networks



Speech Recognition

Speech Recognition

Google Home

Voice-activated speaker

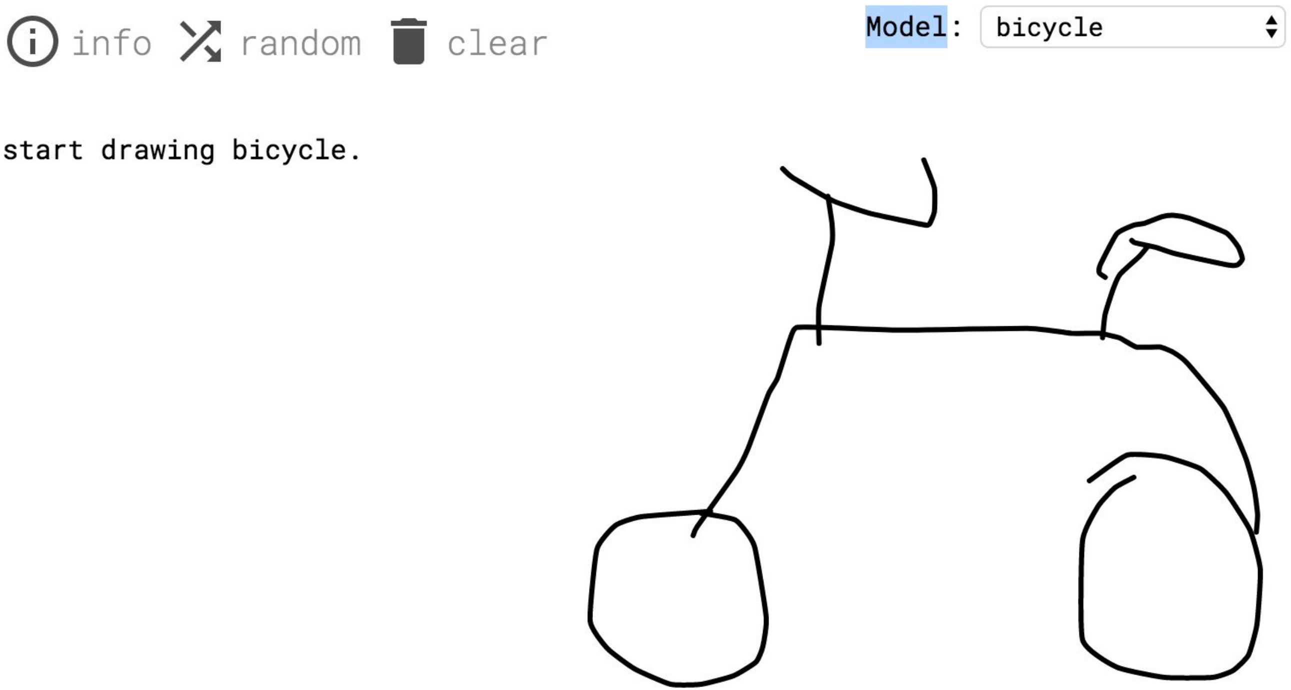
Machine Translation

http://blog.webcertain.com/machine-translation-technology-the-search-engine-takeover/18/02/2015/
https://www.youtube.com/watch?v=06olHmcJjS0

Deep Bach

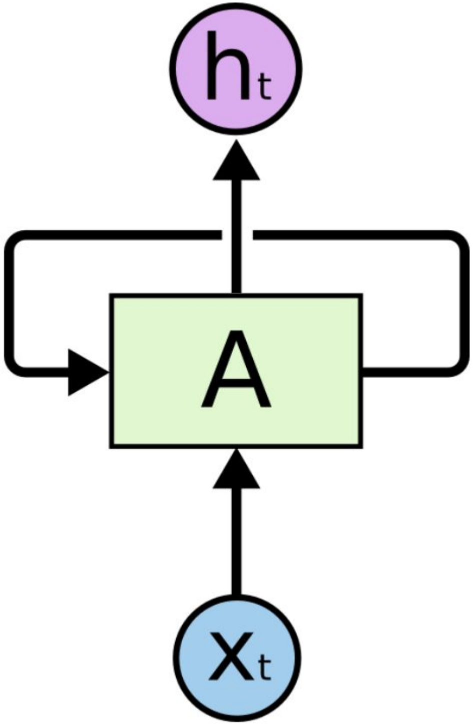
https://www.youtube.com/watch?v=QiBM7-5hA6o

Sketch RNN



https://magenta.tensorflow.org/assets/sketch_rnn_demo/index.html

Vanilla Neural Networks



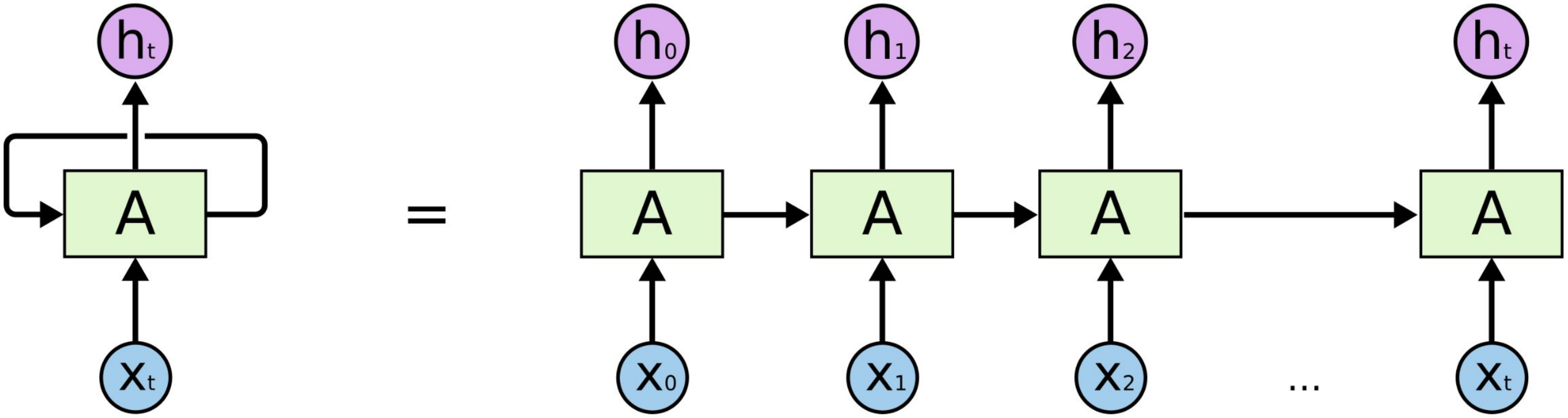
$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture10.pdf

RNNs - Fold / Unfold



http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture10.pdf

모델링을 해봅시다

여기에 ???

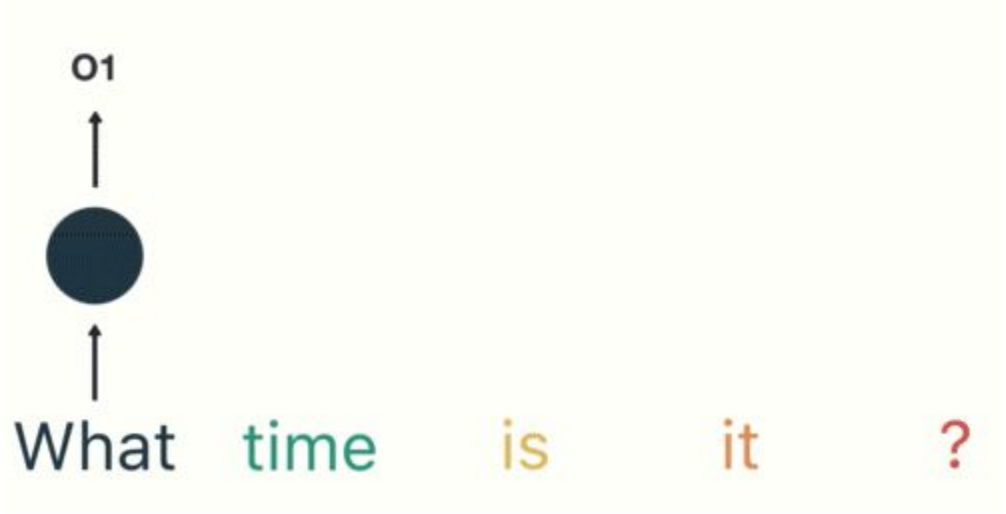
모델링을 해봅시다

내 물건을 찾으러 여기에 ???

모델링을 해봅시다

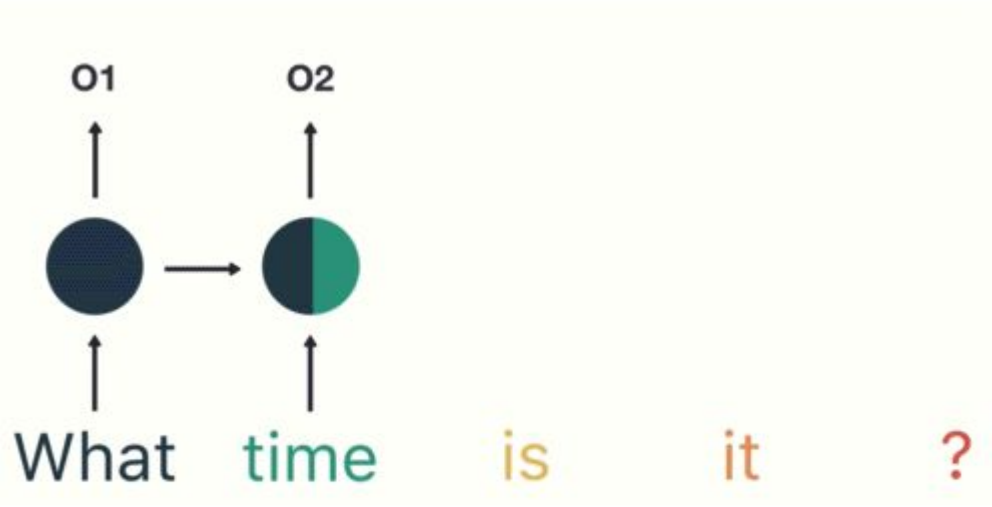
어제 내 물건을 찾으려 여기에 ???

RNNs



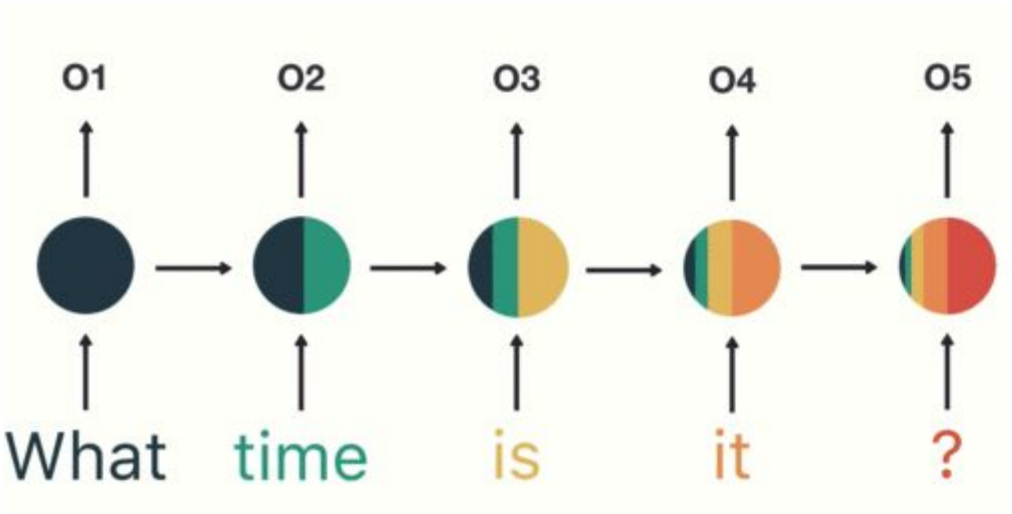
https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9

RNNs



https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9

RNNs

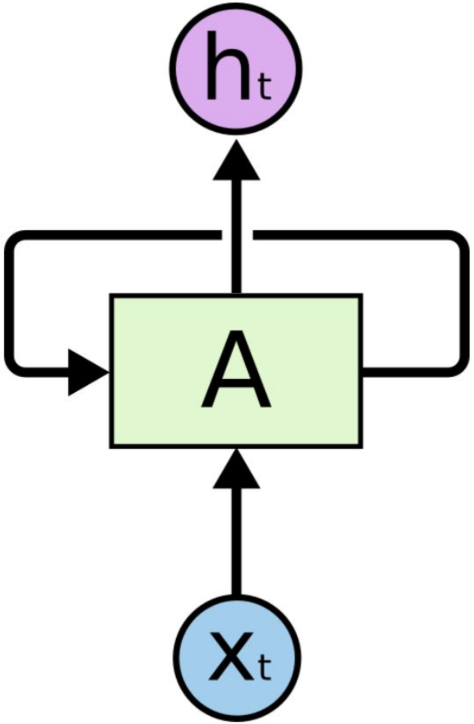


https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9

RNNs

http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture10.pdf

RNNs



$$\boxed{h_t} = \boxed{f}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input data at time t

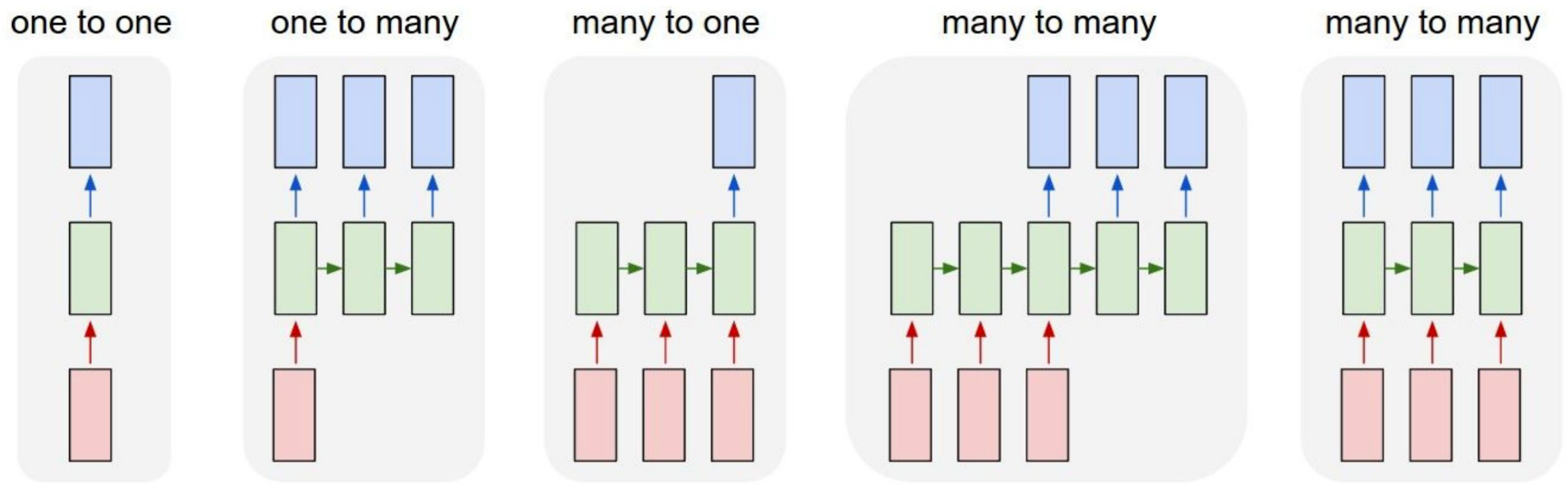
some function

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$y_t = W_{hy} h_t$$

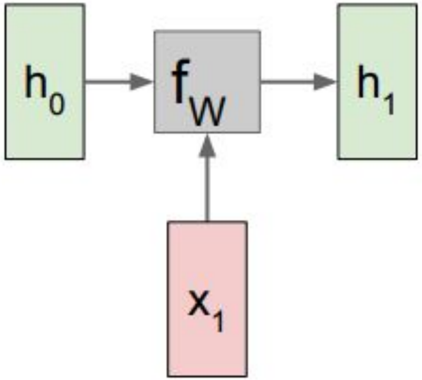
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNNs



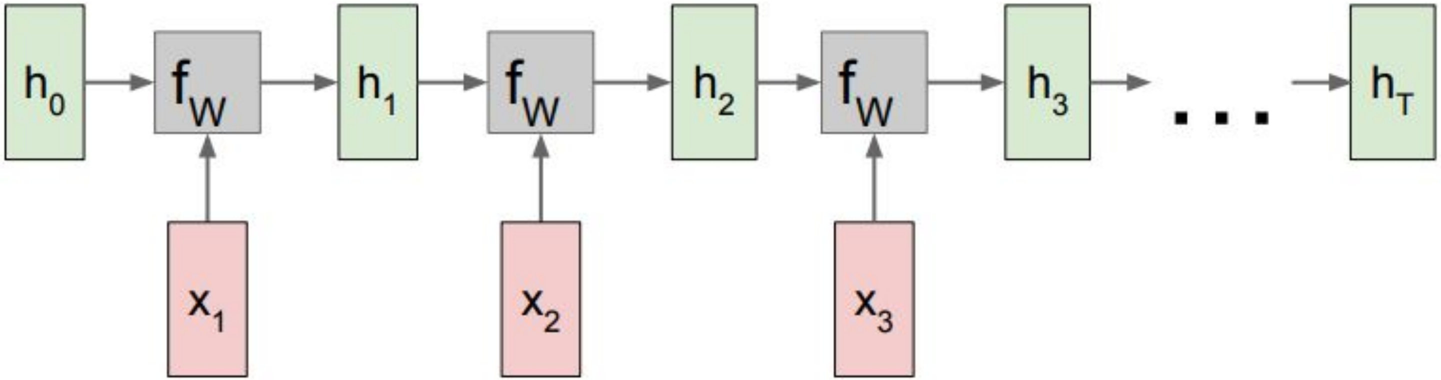
http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture10.pdf

RNN: Computational Graph



http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture10.pdf

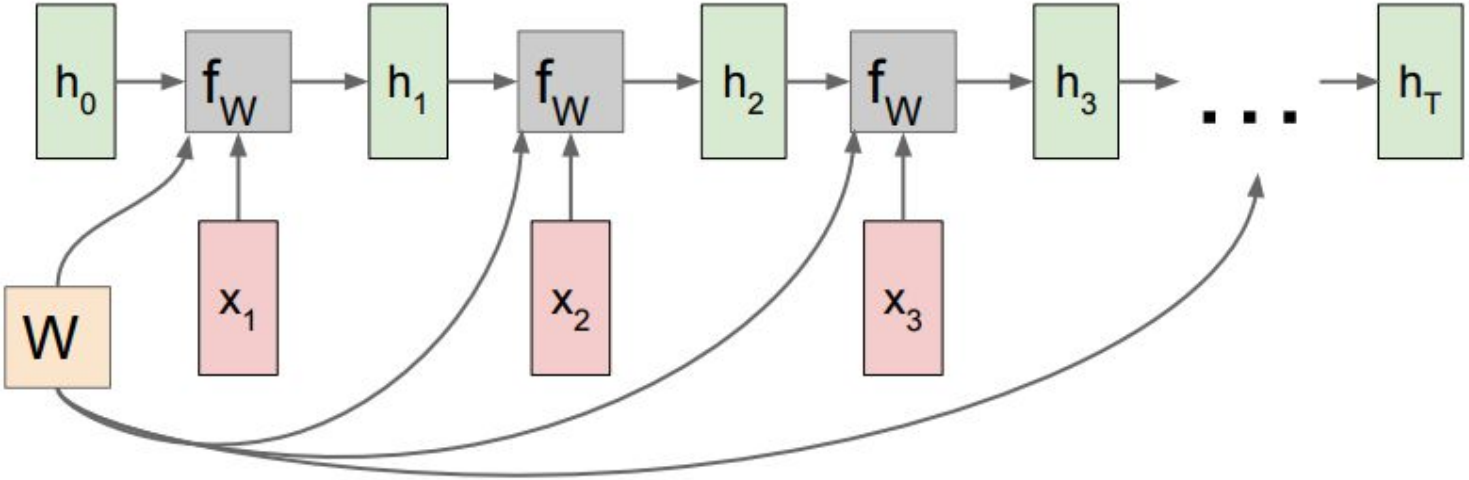
RNN: Computational Graph



http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture10.pdf

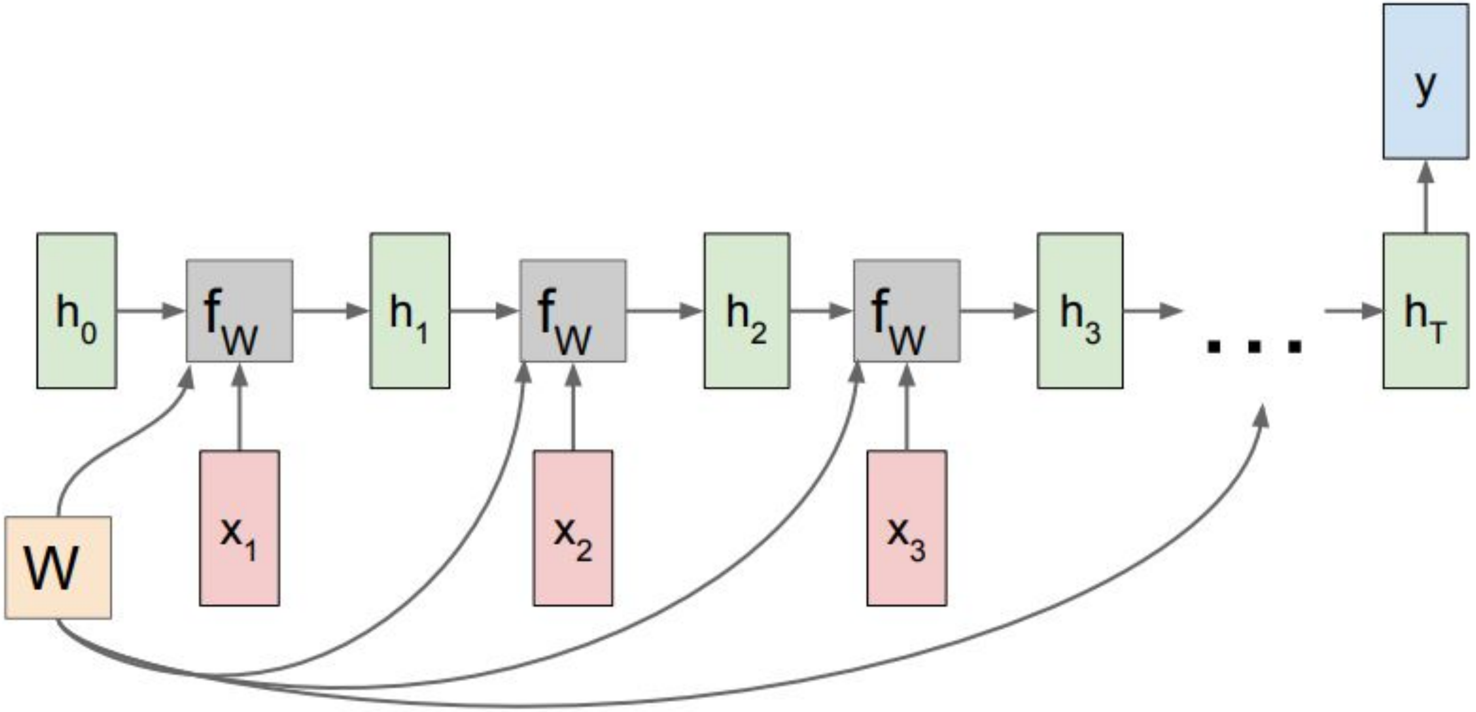
RNN: Computational Graph

Re-use the same weight matrix at every time-step



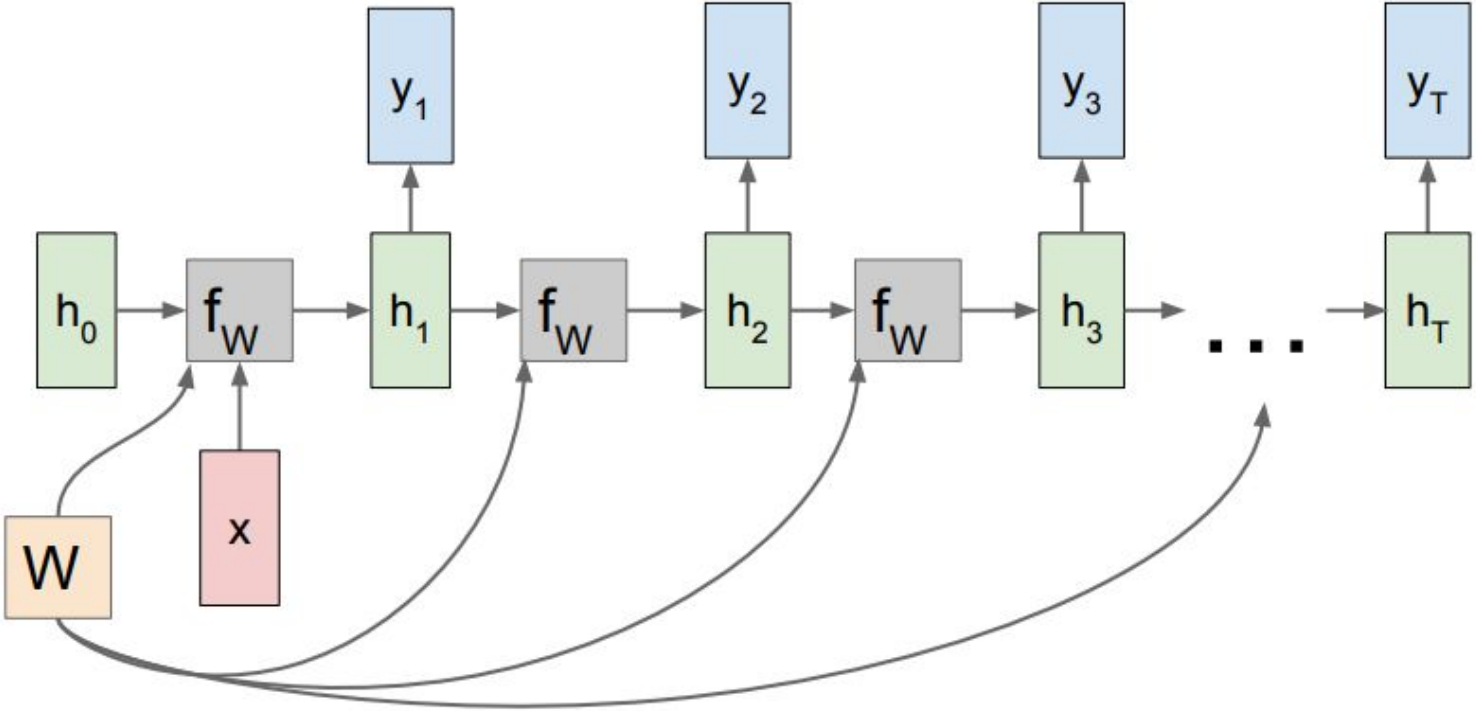
http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture10.pdf

RNN: Computational Graph: Many to One



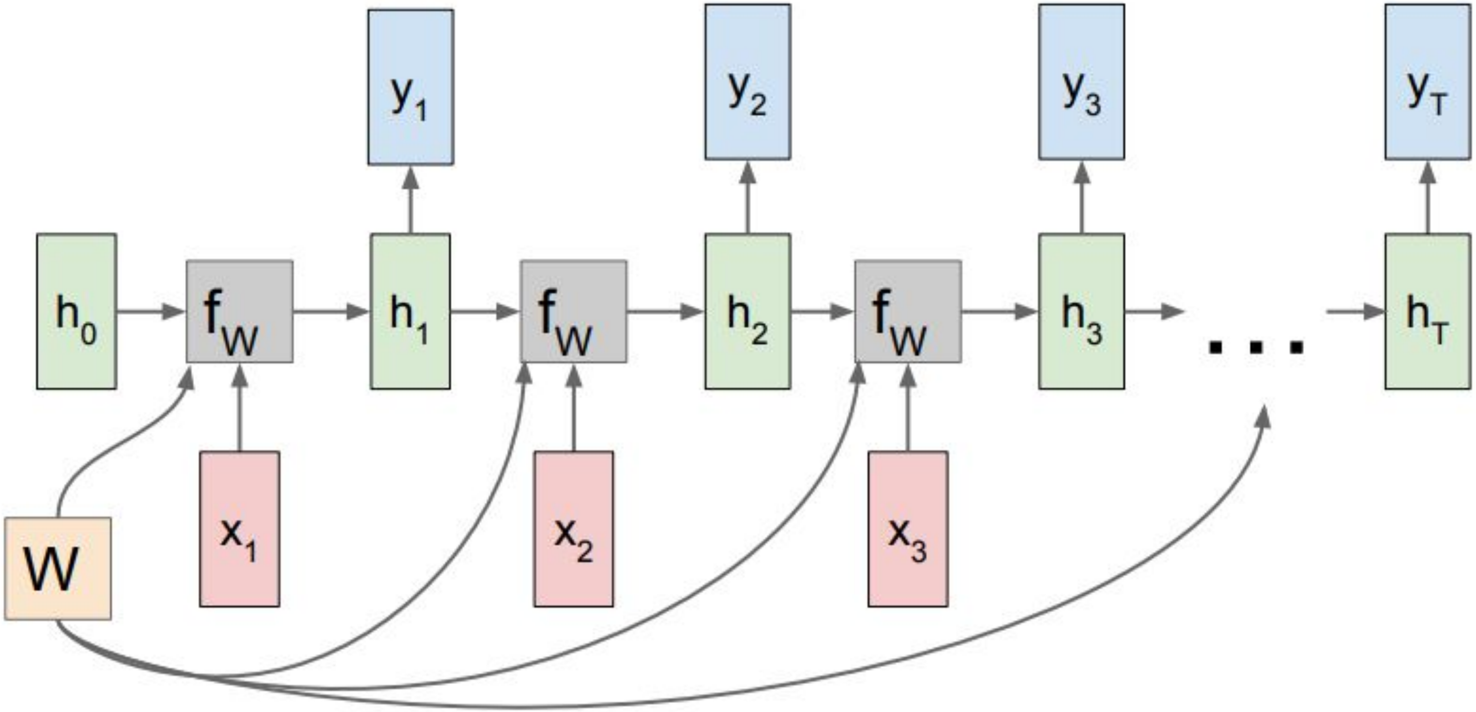
http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture10.pdf

RNN: Computational Graph: One to Many



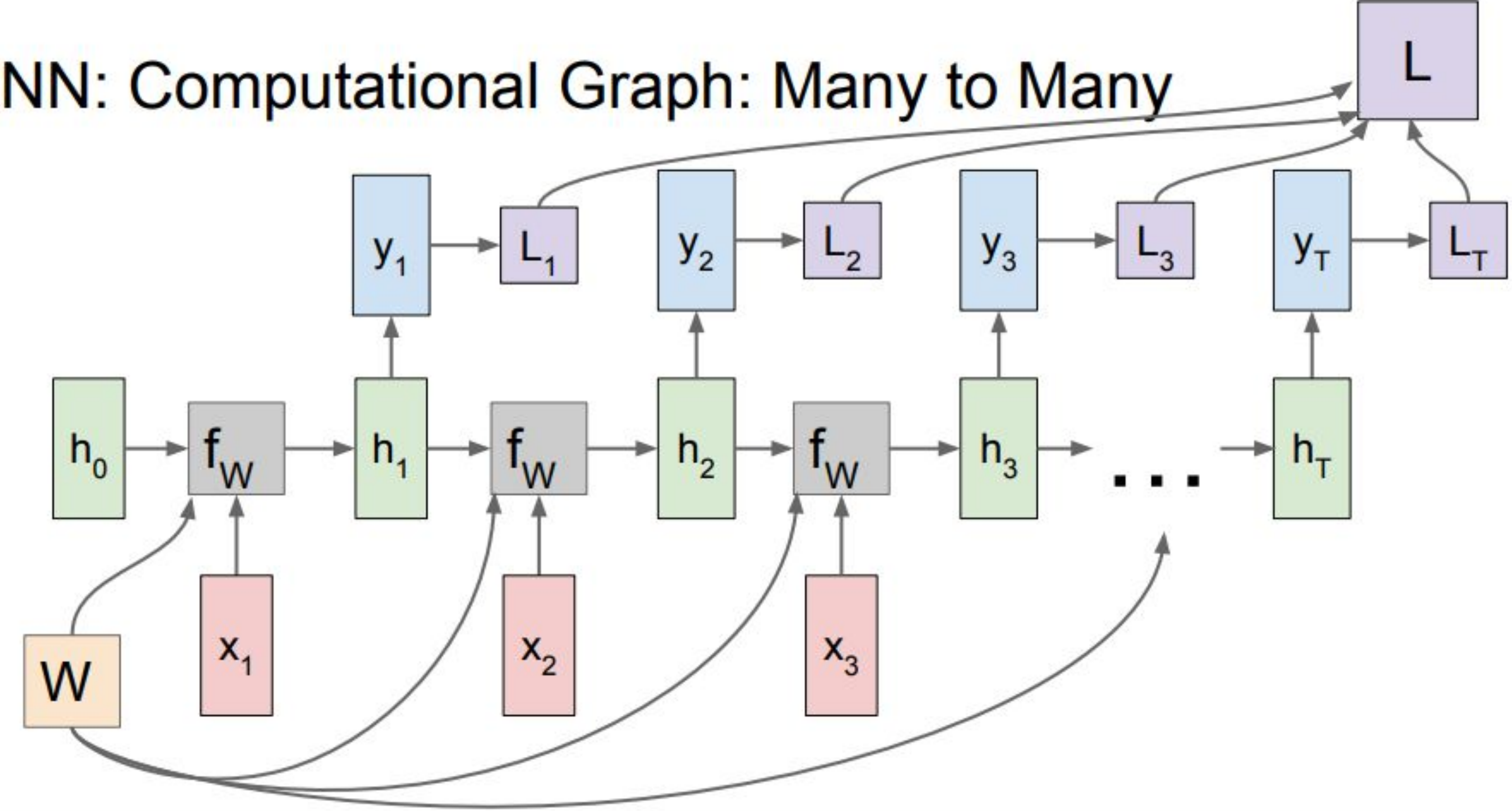
http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture10.pdf

RNN: Computational Graph: Many to Many



http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture10.pdf

RNN: Computational Graph: Many to Many

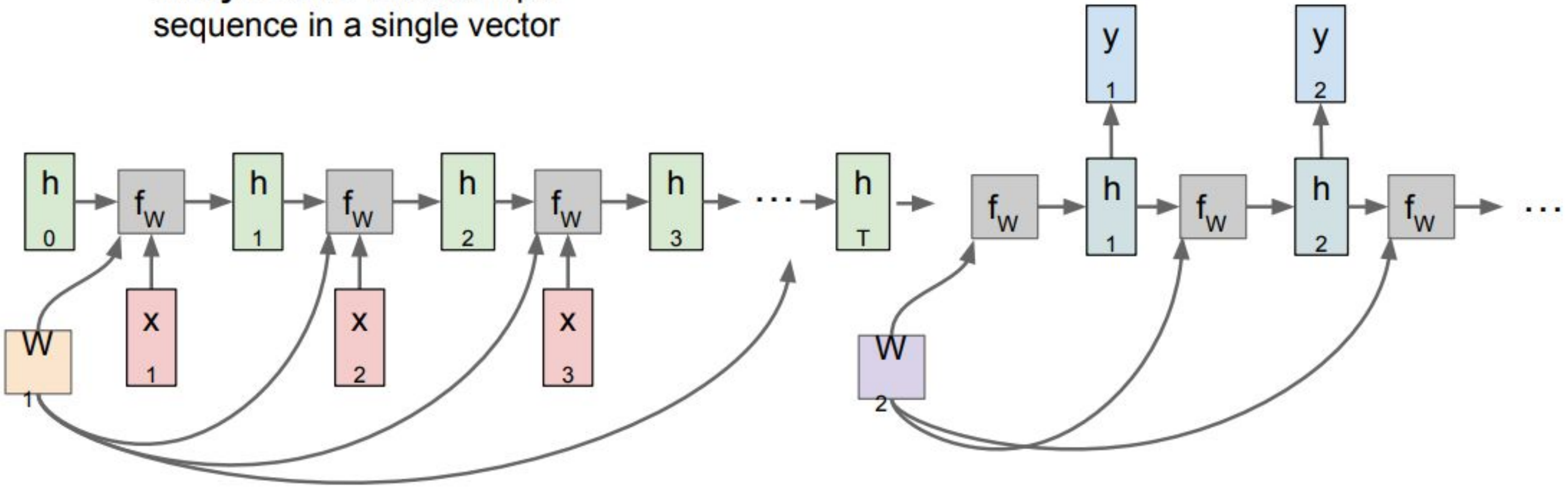


http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture10.pdf

Sequence to Sequence: Many-to-one + one-to-many

One to many: Produce output sequence from single input vector

Many to one: Encode input sequence in a single vector



Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

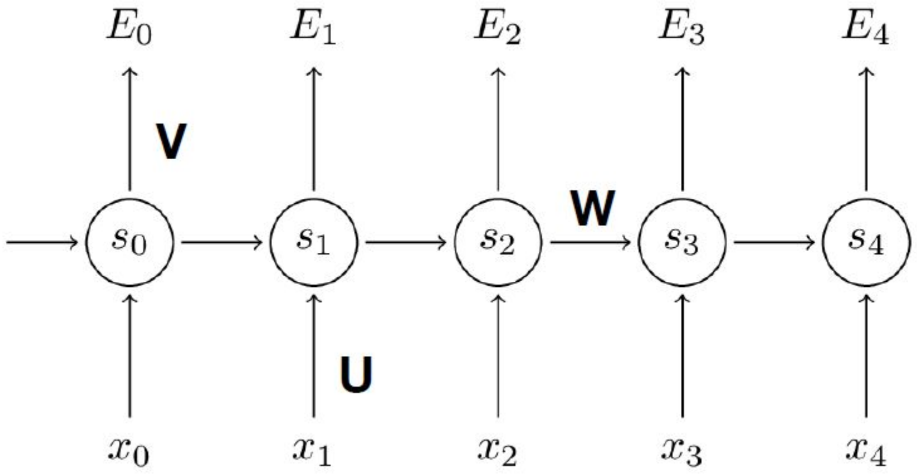
RNNs - Backpropagation Through Time

Cross entropy loss at time t
and total cross entropy

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t)$$

$$= -\sum_t y_t \log \hat{y}_t$$



RNN formula

$$h_t = \tanh(Ux_t + Wh_{t-1})$$

$$\hat{y}_t = \mathrm{softmax}(Vh_t)$$

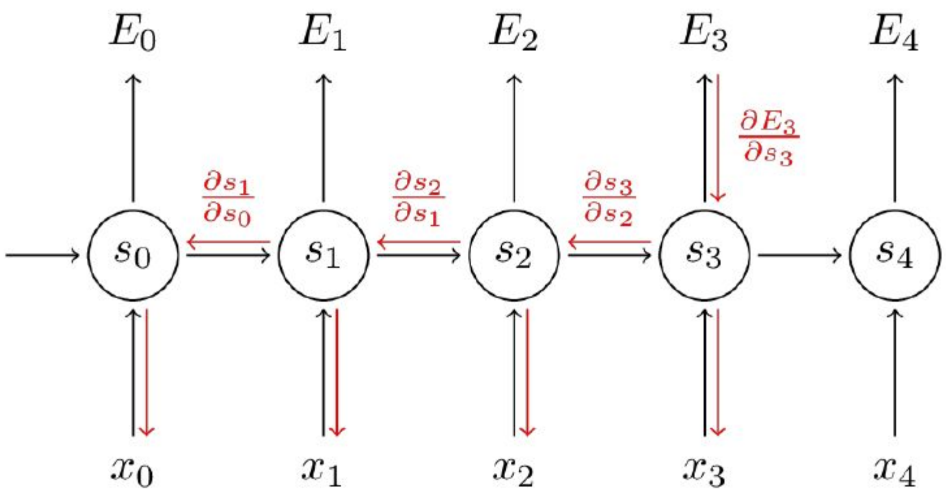
http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/

RNNs - Backpropagation Through Time

V에 대한 미분 at t=3

where $z_3 = Vh_3$

$$
\begin{aligned}
\frac{\partial E_3}{\partial V} &= \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial V} \\
&= \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3}\frac{\partial z_3}{\partial V} \\
&= (\hat{y}_3 - y_3) \otimes h_3
\end{aligned}
$$



RNN formula

$$
\begin{aligned}
h_t &= \tanh(Ux_t + Wh_{t-1}) \\
\hat{y}_t &= \mathrm{softmax}(Vh_t)
\end{aligned}
$$

http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/

RNNs - Backpropagation Through Time

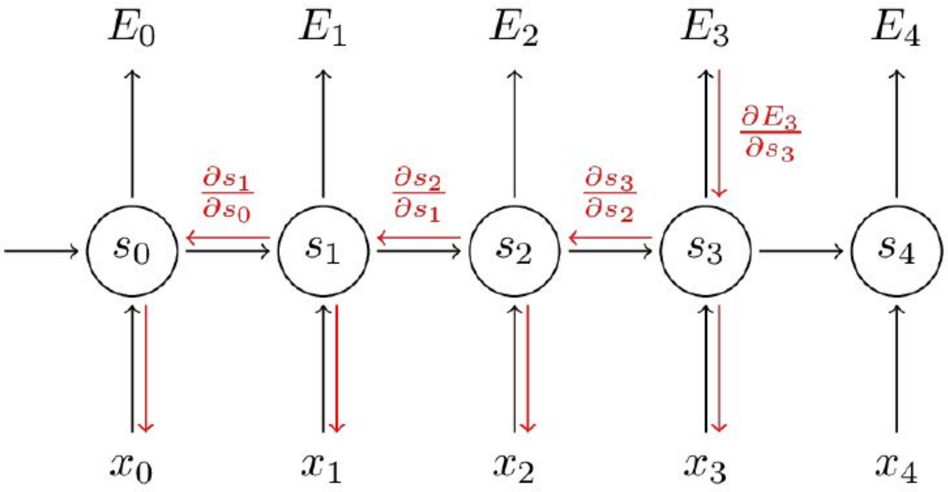
W에 대한 미분 at t=3

where $z_3 = Vh_3$

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial h_3}\frac{\partial h_3}{\partial W}$$

where $h_3 = \tanh(Ux_3 + Wh_2)$

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial h_3}\frac{\partial h_3}{\partial h_k}\frac{\partial h_k}{\partial W}$$

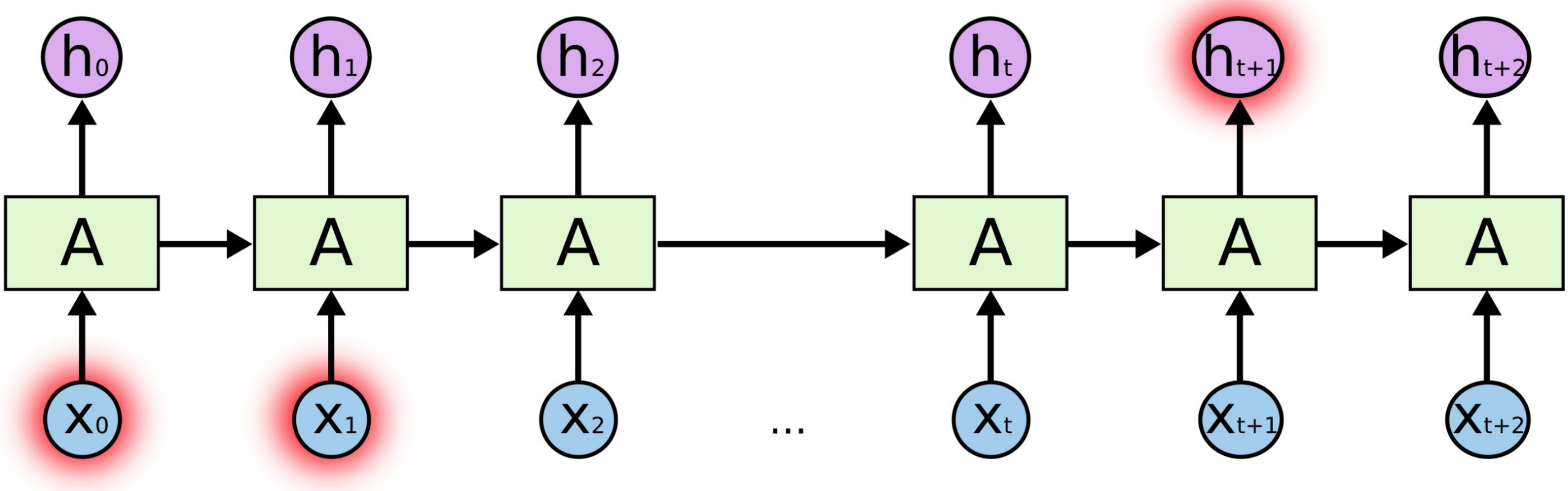


RNN formula

$$h_t = \tanh(Ux_t + Wh_{t-1})$$
$$\hat{y}_t = \mathrm{softmax}(Vh_t)$$

http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/

RNNs



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNNs

Final hidden state of the RNN

- Vanishing Gradient
- 시간적으로 멀리 있는 정보는

  Vanishing Gradient 현상이 일어나 학습이 힘들다.

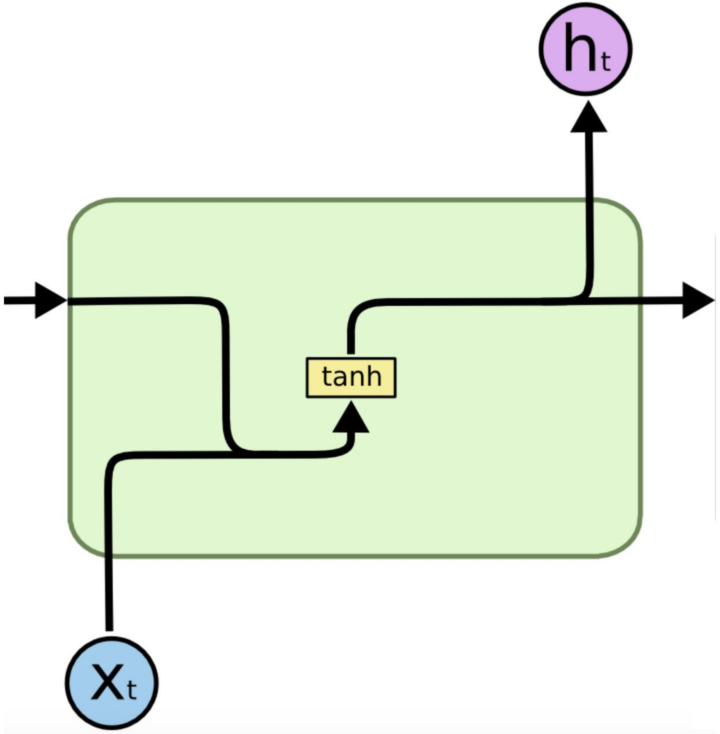
https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9

RNNs - Complex Models

- Gate의 도입으로 입력, 출력을 조절할 수 있게 됐다
- 메모리를 이용하여 Long-range correlation을 학습할 수 있게 됐다
- Error를 입력에 따라 다른 강도로 전달할 수 있다
- Vanishing Gradient 문제 해결

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Vanilla RNN

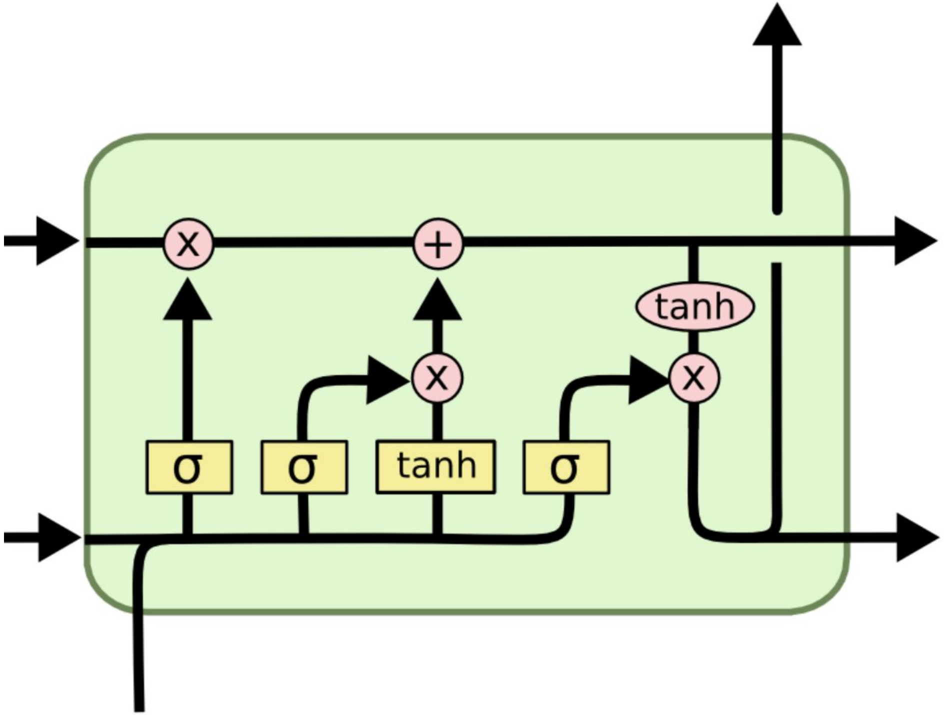


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short Term Memory(LSTM)



$$f_t = \sigma(W_{hf}h_{t-1} + W_{xh}x_t)$$
$$i_t = \sigma(W_{hi}h_{t-1} + W_{xi}x_t)$$
$$o_t = \sigma(W_{ho}h_{t-1} + W_{xo}x_t)$$
$$\tilde{C}_t = \tanh(W_{hC}h_{t-1} + W_{xC}x_t)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$h_t = o_t * \tanh(C_t)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short Term Memory(LSTM)

forget gate $f_t = \sigma(W_{hf} h_{t-1} + W_{xh} x_t)$

input gate $i_t = \sigma(W_{hi} h_{t-1} + W_{xi} x_t)$

output gate $o_t = \sigma(W_{ho} h_{t-1} + W_{xo} x_t)$

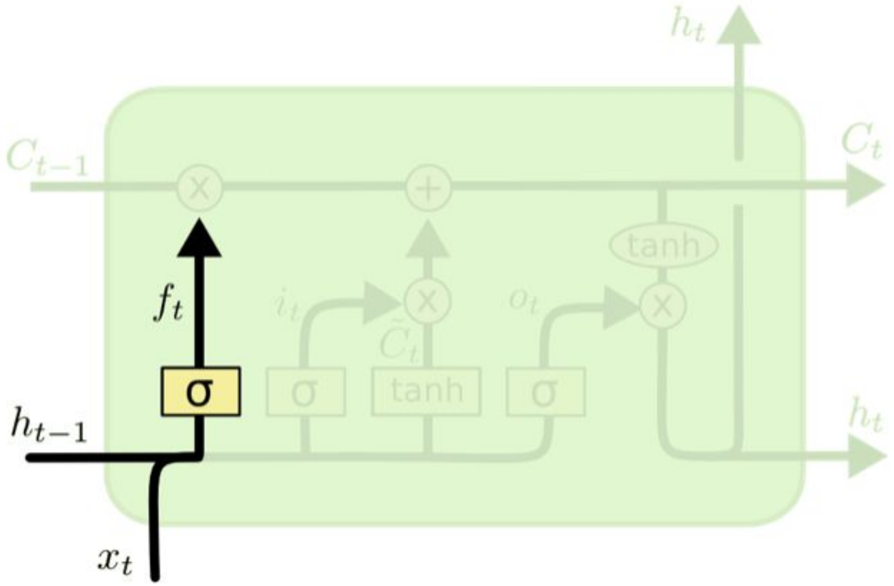
candidate memory $\tilde{C}_t = \tanh(W_{hC} h_{t-1} + W_{xC} x_t)$

memory cell $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$

hidden state $h_t = o_t * \tanh(C_t)$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short Term Memory(LSTM)

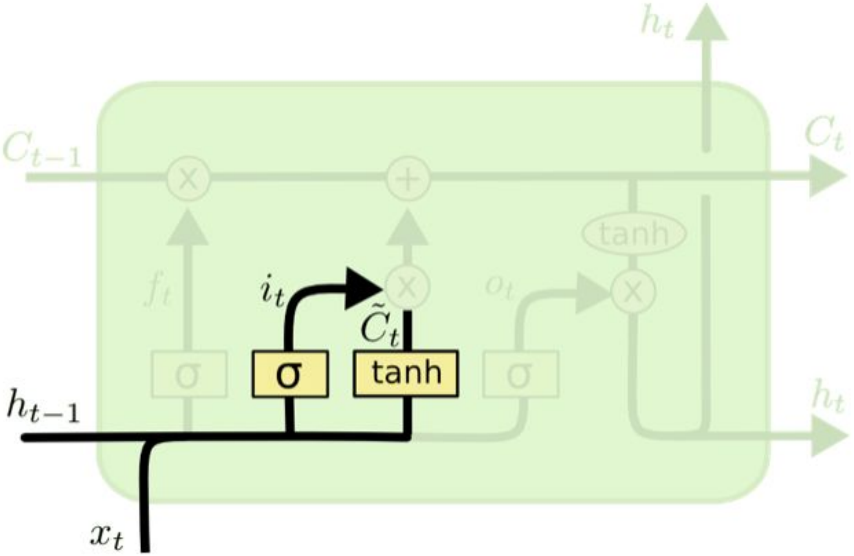


forget gate

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short Term Memory(LSTM)



input gate

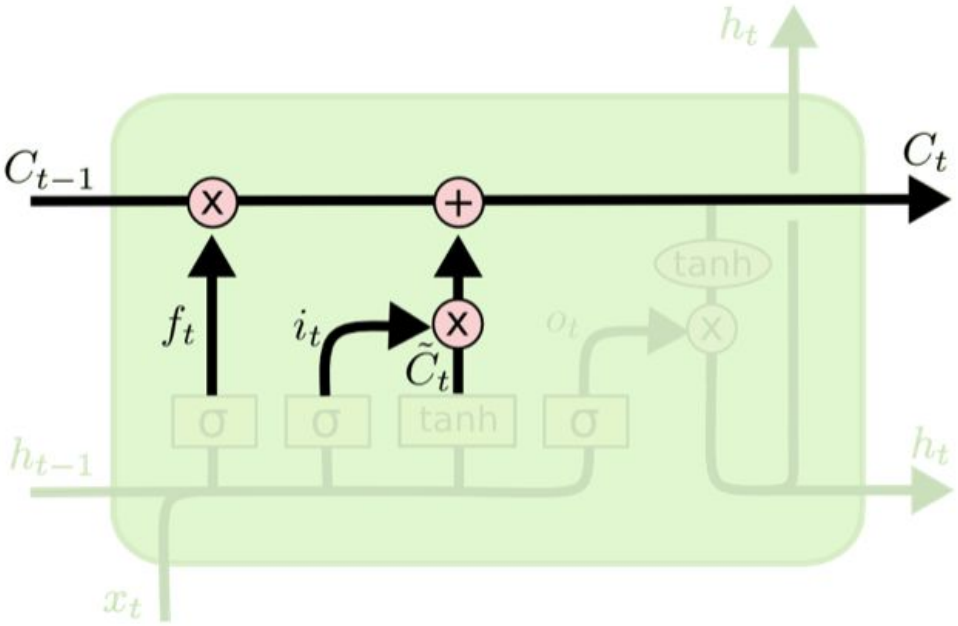
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

candidate memory

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short Term Memory(LSTM)

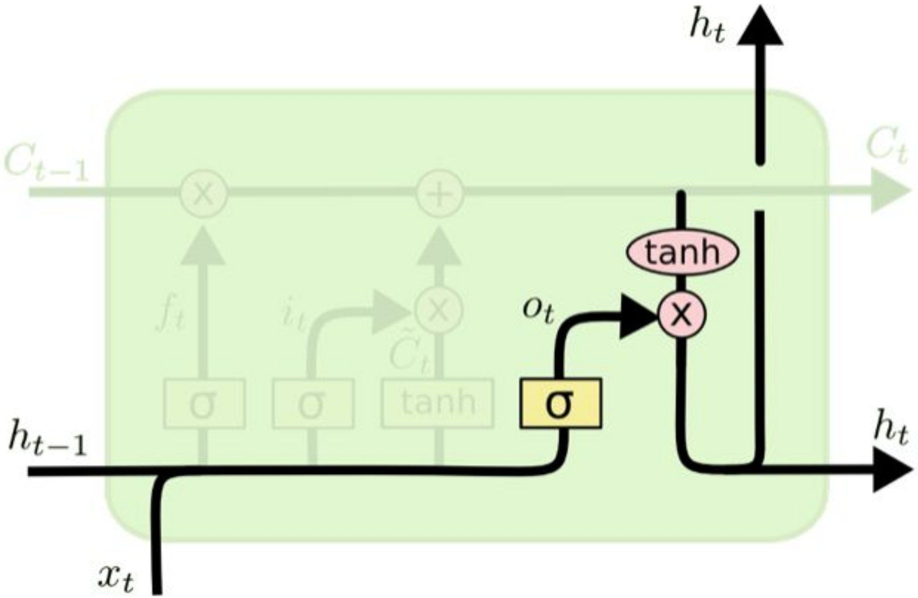


memory cell

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short Term Memory(LSTM)



output gate

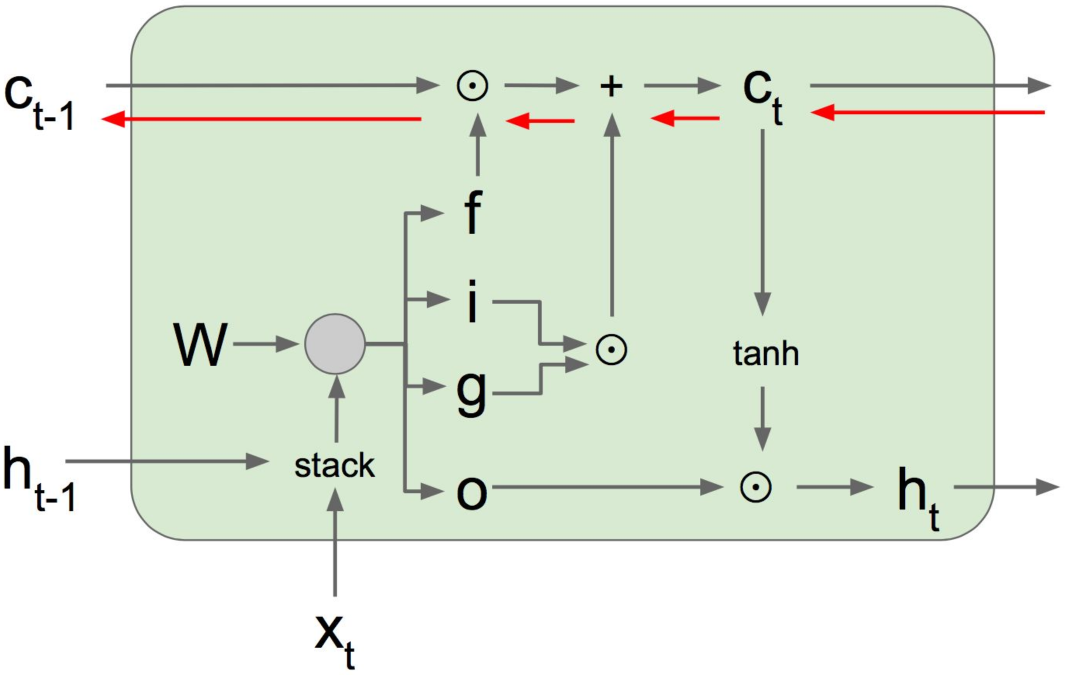
$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

hidden state

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Gradient Flow



Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

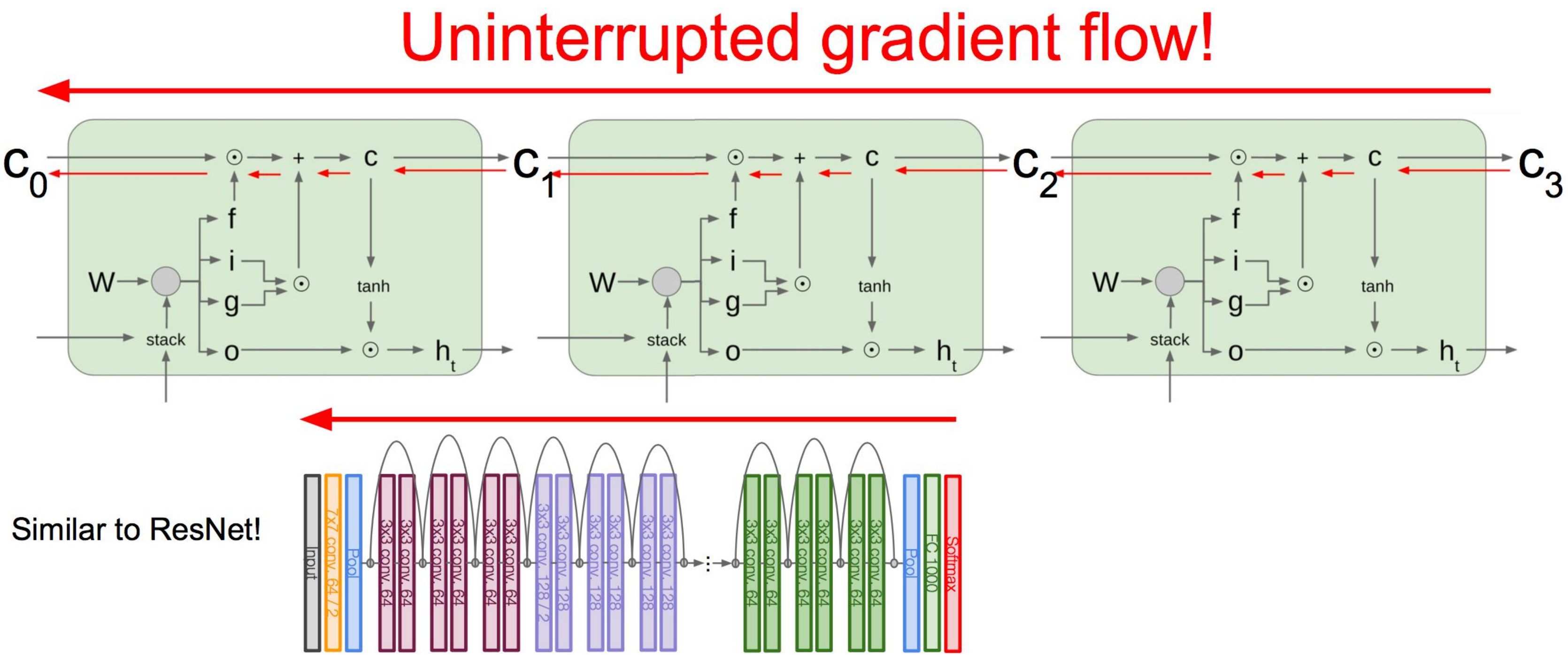
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

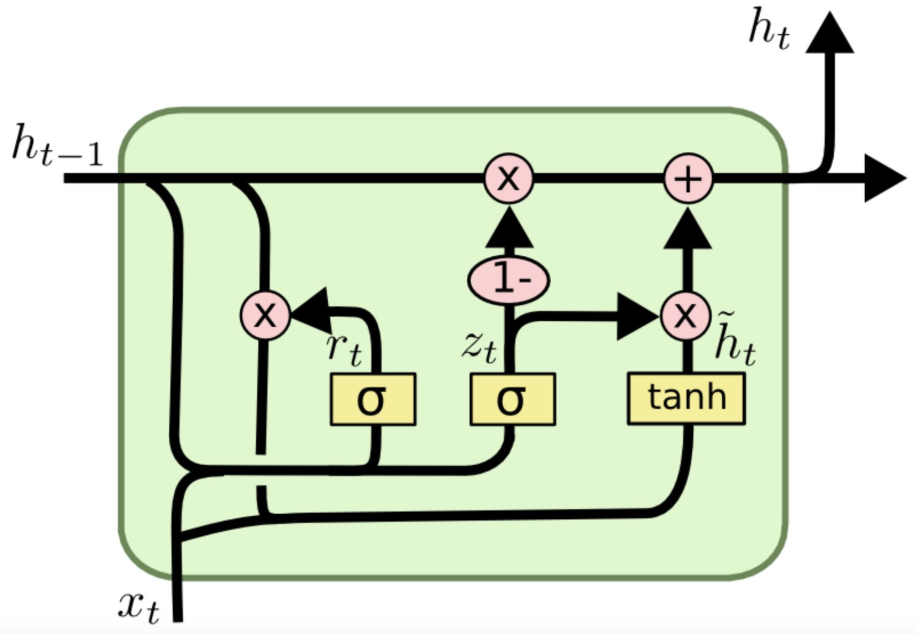
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

LSTM Gradient Flow



http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

Gated Recurrent Unit(GRU)



$$z_t = \sigma(W_{hz}h_{t-1} + W_{xz}x_t)$$

$$r_t = \sigma(W_{hr}h_{t-1} + W_{xr}x_t)$$

$$\tilde{h}_t = \tanh(W_{hh}(r_t * h_{t-1}) + W_{xh}x_t)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

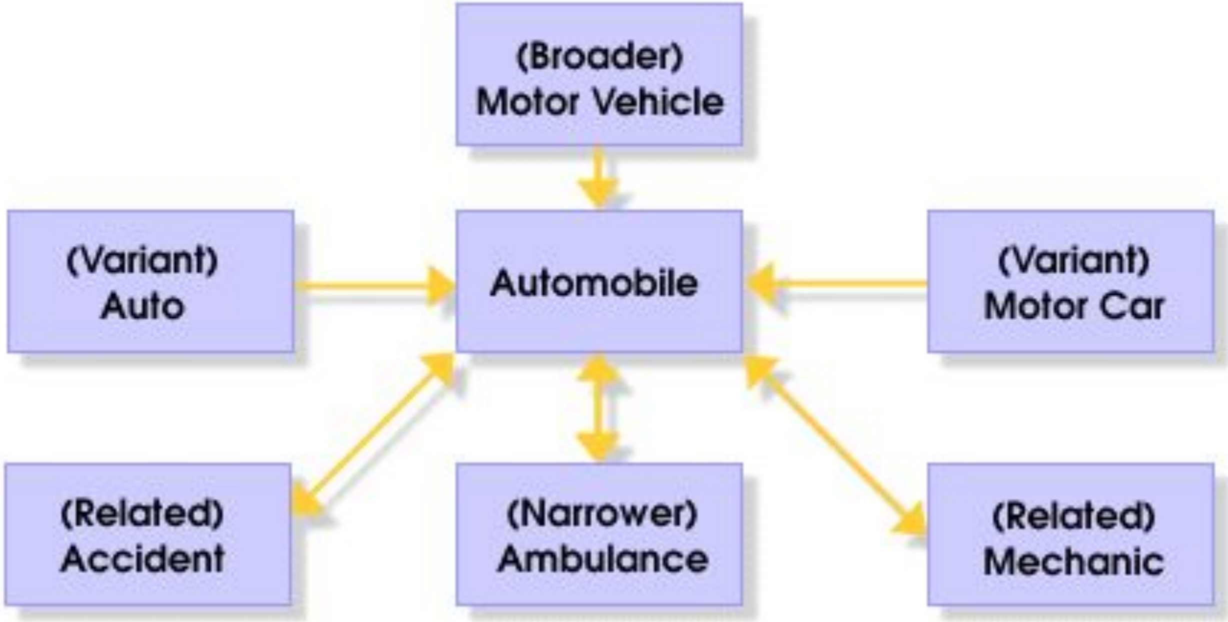
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gated Recurrent Unit(GRU)

update gate	$z_t = \sigma(W_{hz}h_{t-1} + W_{xz}x_t)$
reset gate	$r_t = \sigma(W_{hr}h_{t-1} + W_{xr}x_t)$
new memory	$\tilde{h}_t = \tanh(W_{hh}(r_t * h_{t-1}) + W_{xh}x_t)$
final memory state	$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

NLP - thesaurus



http://www.drdobbs.com/building-a-synonymous-search-index/184412944

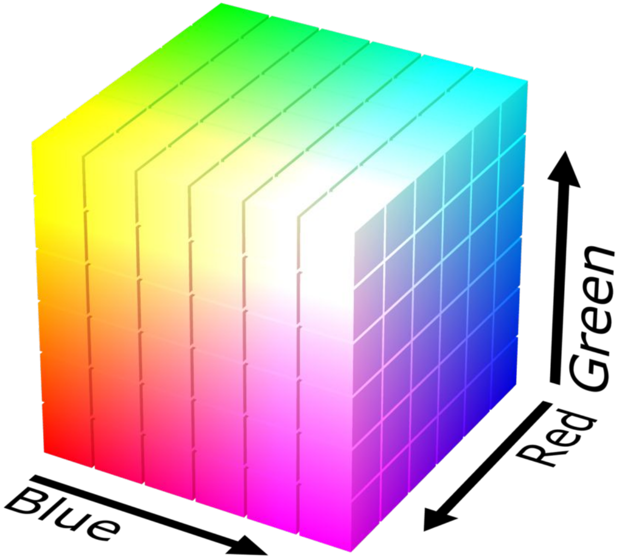
NLP - thesaurus

- 시소러스의 문제점
  - 시대 변화에 대응하기 힘들다 (시대에 따라 달라지는 뜻)
  - 사람이 만들어야한다
  - 단어의 미묘한 차이를 구분 할 수 없다.

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

NLP - distributional hypothesis

- Color => RGB 벡터로 표현
- 단어도 벡터로 표현해보자!



NLP - distributional hypothesis

- 단어의 의미는 주변 단어에 의해서 생성된다.
- 맥락 (Context) 사용

You say goodbye and I say hello.

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

NLP - distributional hypothesis

You say goodbye and I say hello.

You say goodbye and I say hello.

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

NLP - distributional hypothesis

You say goodbye and I say hello.

	you	say	goodbye	and	I	hello	.
say	0	1	0	0	0	0	0

You say goodbye and I say hello.

	you	say	goodbye	and	I	hello	.
say	1	0	1	0	1	1	0

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

NLP - co-occurrence matrix (동시발생행렬)

	you	say	goodbye	and	I	hello	.
you	0	1	0	0	0	0	0
say	1	0	1	0	1	1	0
goodbye	0	1	0	1	1	0	0
and	0	0	1	0	1	0	0
I	0	1	0	1	0	0	0
hello	0	1	0	0	0	0	1
.	0	0	0	0	0	1	0

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

NLP - co-occurrence matrix (동시발생행렬)

- 더 강력한 방법인 추론 기반 기법을 사용
- Word2Vec 를 사용

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

NLP - Word2Vec

- 추론 기반 기법

You $\boxed{\quad\quad}$ goodbye and I say hello.

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

NLP – Word2Vec

$$\begin{pmatrix} you \\ say \\ goodbye \\ and \\ I \\ Hello \\ . \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

http://jalammar.github.io/illustrated-word2vec/

NLP - Word2Vec

$$\begin{pmatrix} you \\ say \\ goodbye \\ and \\ I \\ Hello \\ . \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} you \\ say \\ goodbye \\ and \\ I \\ Hello \\ . \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

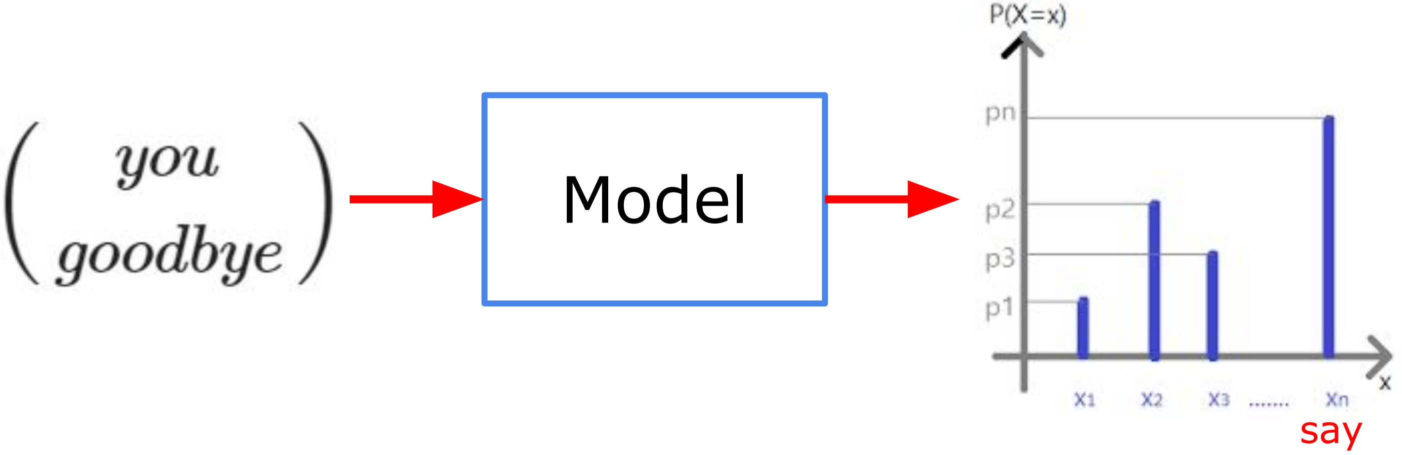
Model

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} you \\ say \\ goodbye \\ and \\ I \\ Hello \\ . \end{pmatrix}$$

http://jalammar.github.io/illustrated-word2vec/

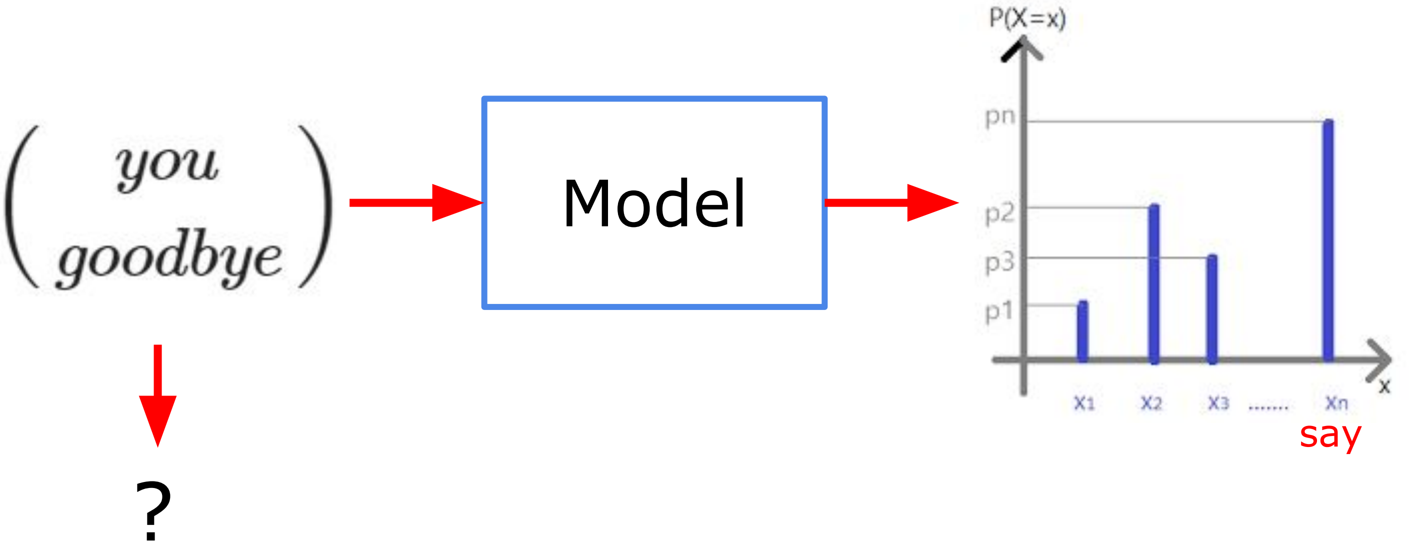
NLP - Word2Vec

- 추론 기반 기법

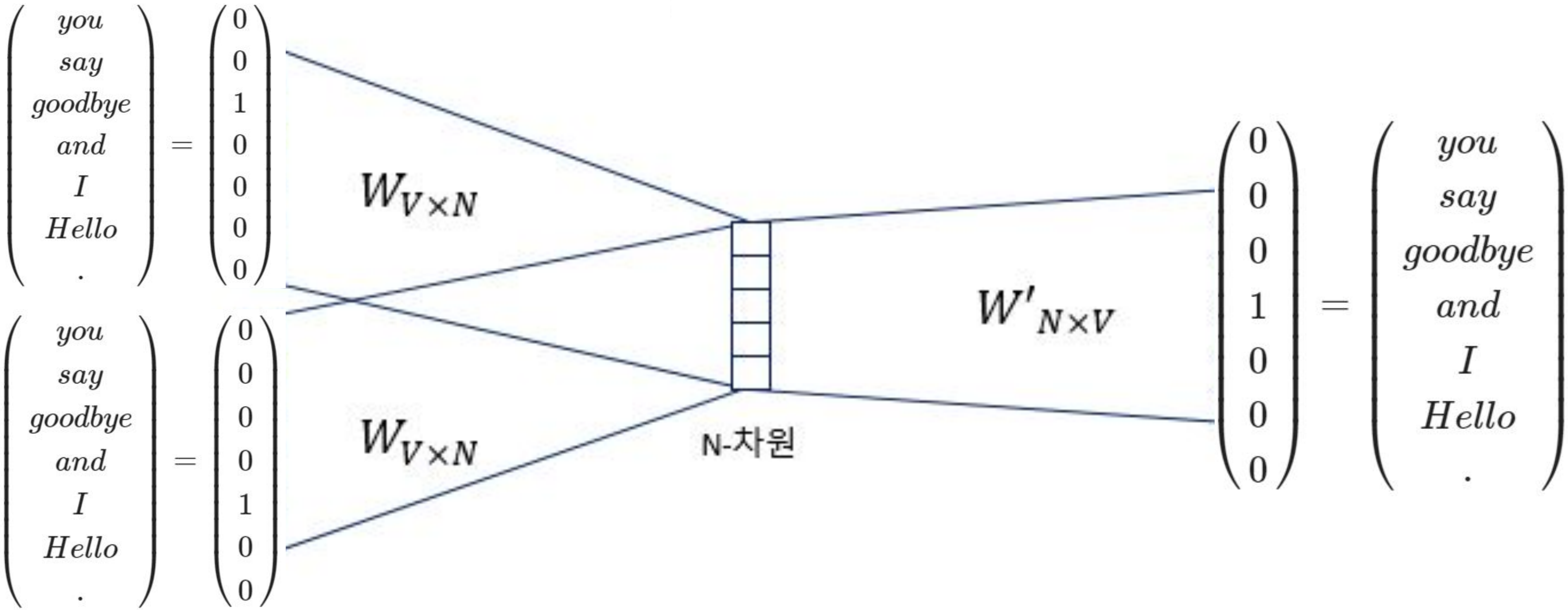


NLP - Word2Vec

- 추론 기반 기법



NLP – Word2Vec – CBOW



https://wikidocs.net/22660

NLP - Word Embedding

$$\begin{pmatrix} 0 & 1 & 0 & \dots & 0 & 0 \end{pmatrix} \; W \; = \; h$$

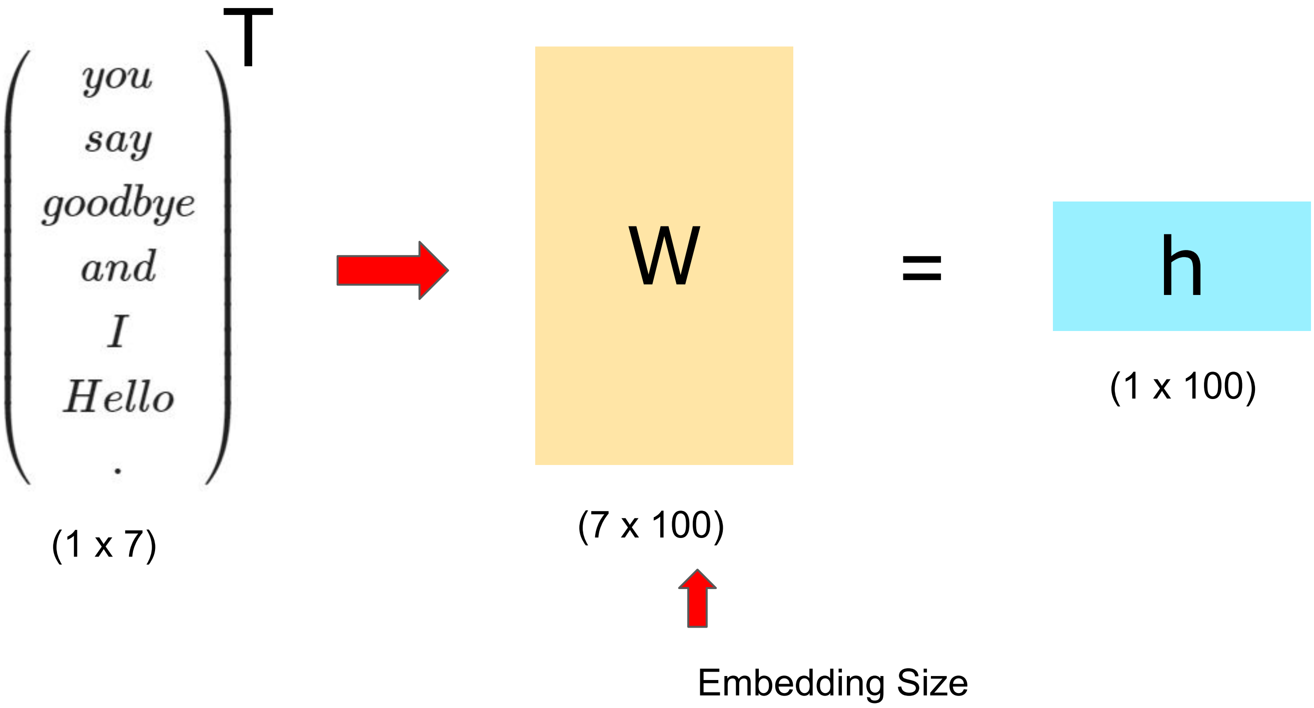
(1 x 1,000,000)

(1,000,000 x 100)

(1 x 100)

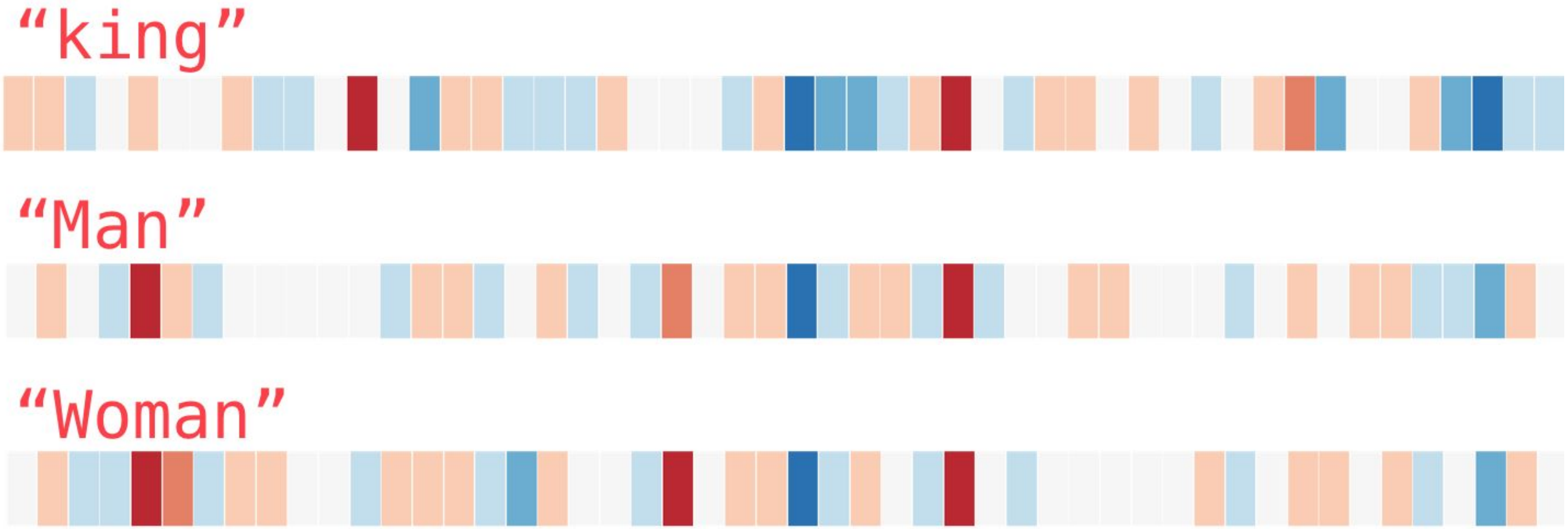
https://wikidocs.net/22660

NLP - Word Embedding



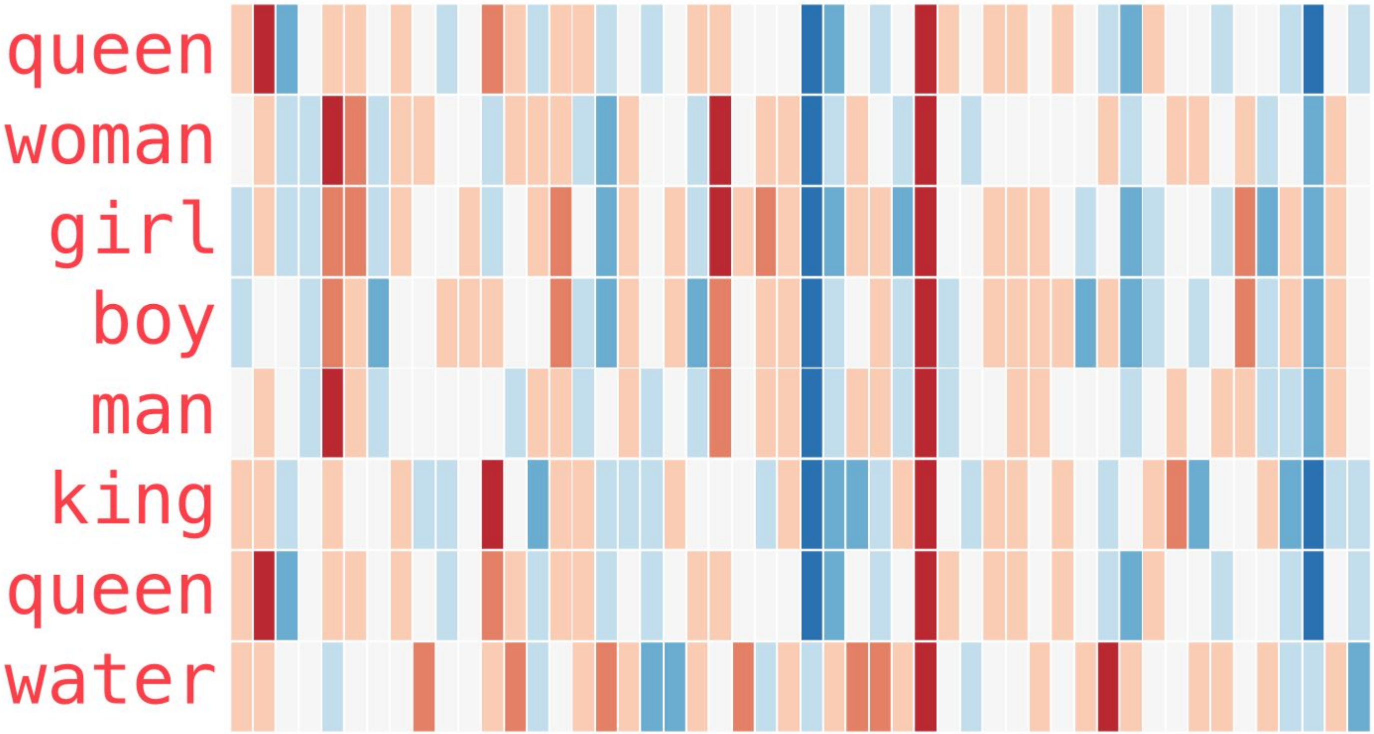
https://wikidocs.net/22660

NLP - Word Embedding



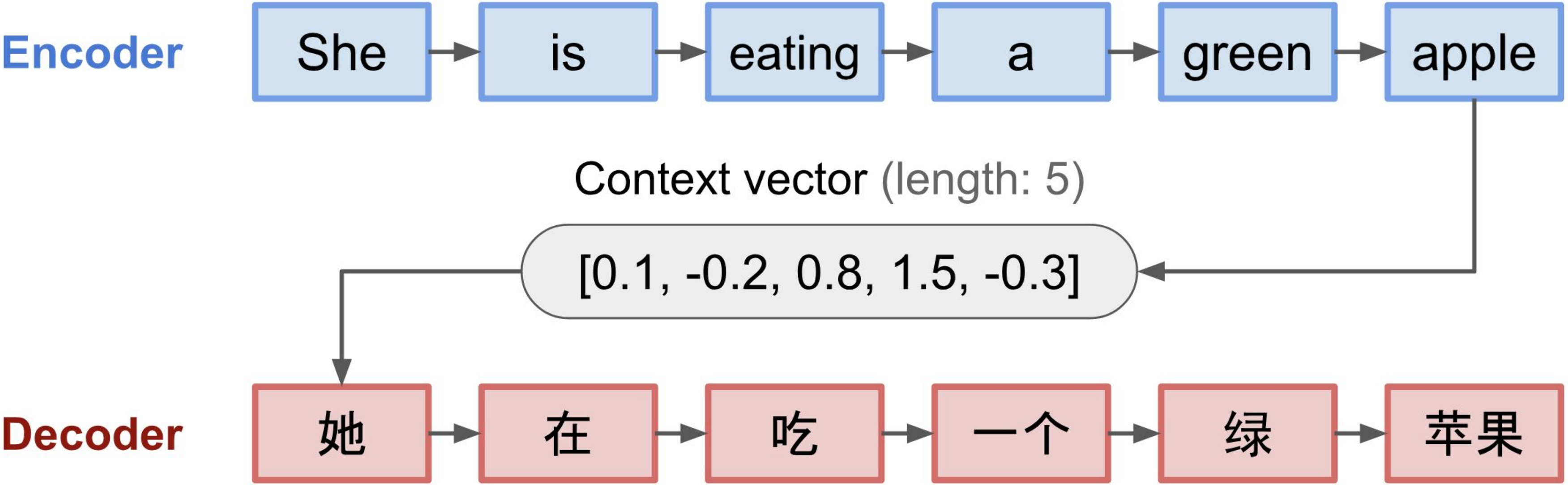
http://jalammar.github.io/illustrated-word2vec/

NLP - Word Embedding



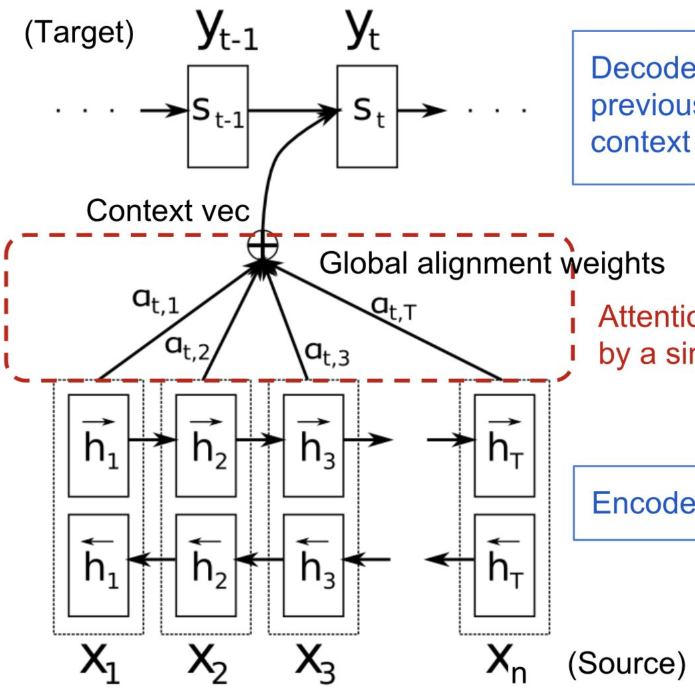
http://jalammar.github.io/illustrated-word2vec/

Seq2seq



https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

Attention



Decoder: RNN with input from previous state + dynamic context vector.

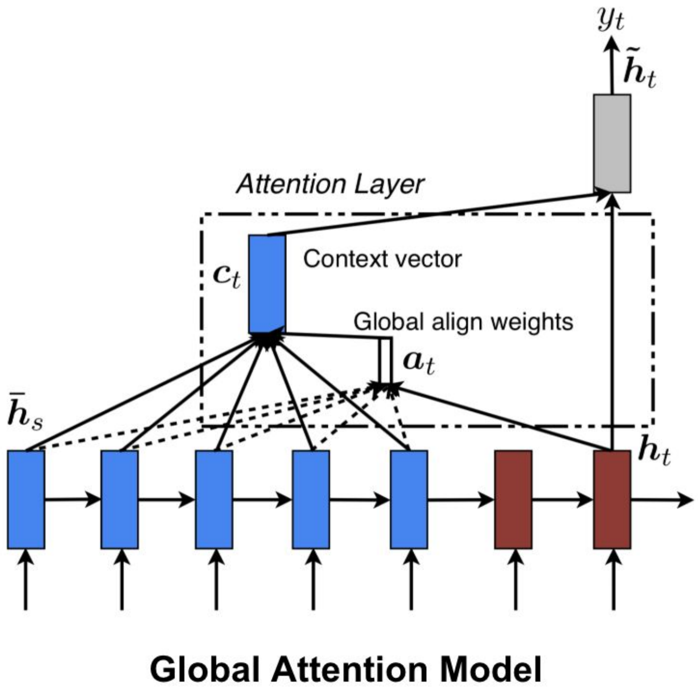
Attention layer: parameterized by a simple feed-forward network

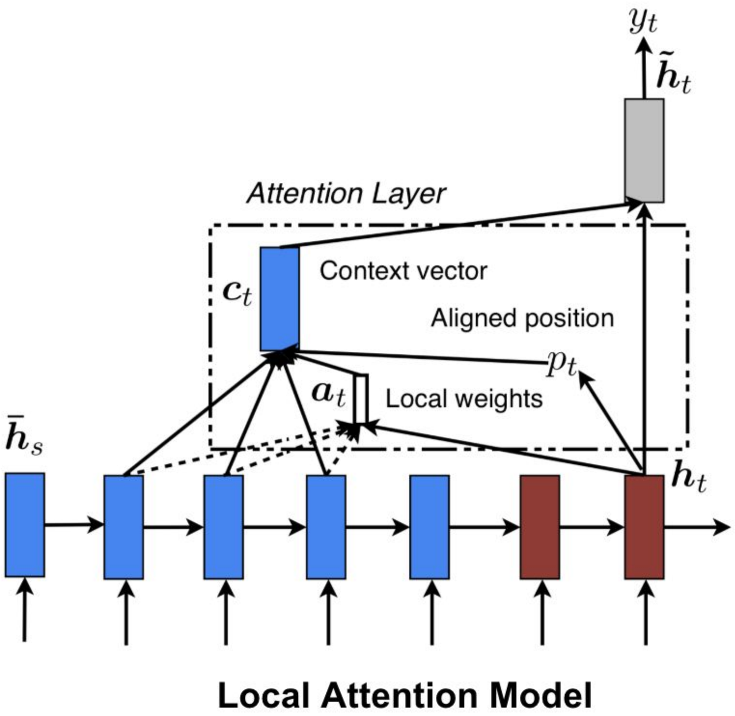
Additive Attention

Encoder: bidirectional RNN

https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

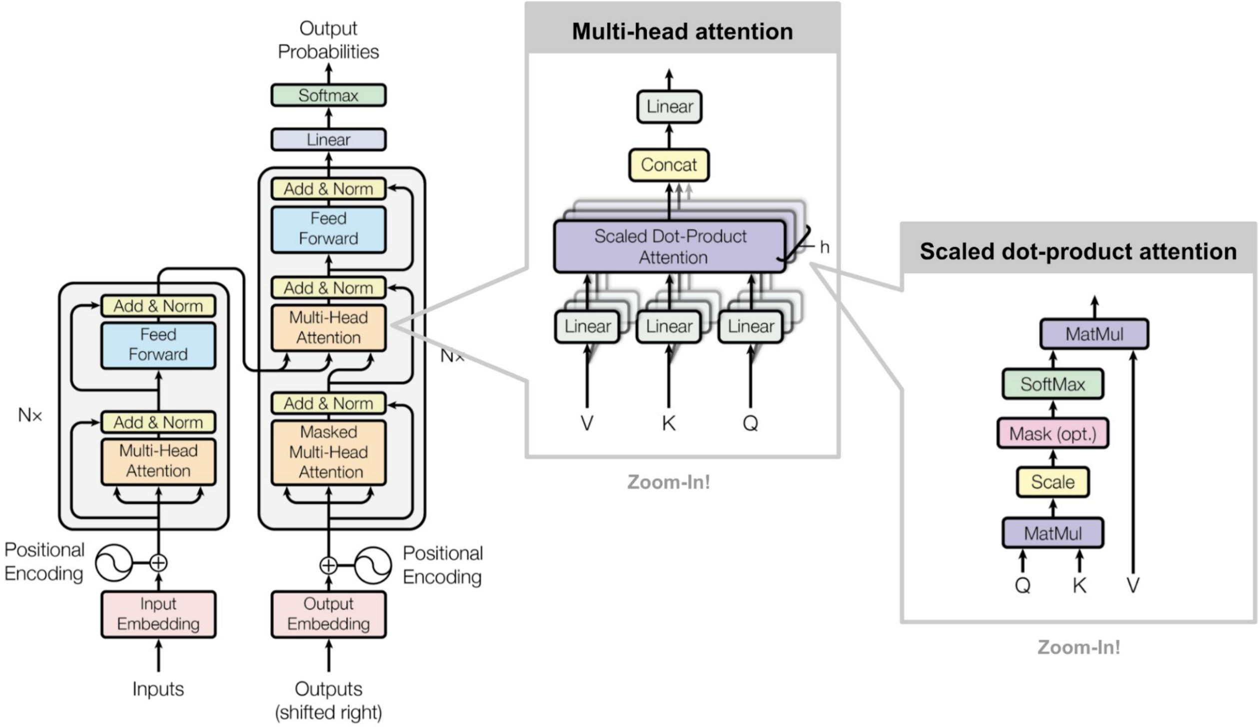
Attention





https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

Attention



https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

참고자료

- 밑바닥부터 시작하는 딥러닝 1, 2
  http://www.yes24.com/Product/Goods/34970929?Acode=101
  http://www.yes24.com/Product/Goods/72173703

- 모두를 위한 딥러닝 시즌2
  https://www.edwith.org/boostcourse-dl-tensorflow/joinLectures/22150

- 모두의연구소 이일구님 강의 자료
  https://github.com/ilguyi