

Week 3

- Recurrent Neural Networks(RNNs)
& Long-Short Term Memory(LSTM) Networks

2019.05.25

Solaris

(<http://solarisailab.com>)

Week 2 복습 - Week2의 학습목표

▣ 2강의 학습목표 :

1. CNN의 기본 구성 요소-Convolution, Pooling(Subsampling)-의 개념을 이해한다.
2. 대표적인 CNN 모델-AlexNet, VGGNet, GoogLeNet-의 구조와 디자인철학을 이해한다.
3. TensorFlow를 이용한 CIFAR-10 분류기, Transfer Learning-Inception v3 Retraining-을 구현해본다.

Week 2 복습 – 영상인식의 어려운 점들

Challenges: Illumination



Challenges: Deformation



Challenges: Occlusion



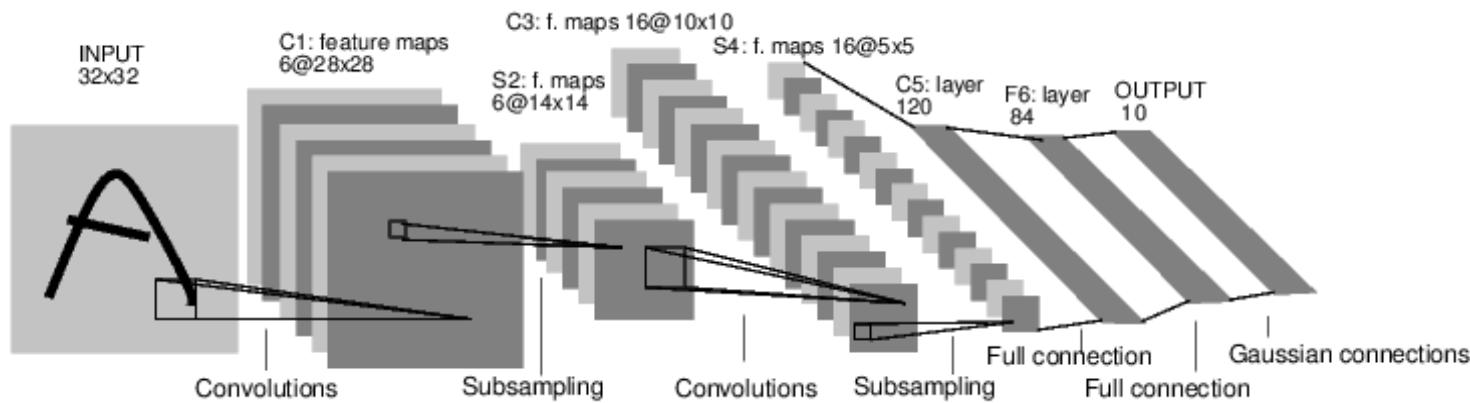
Challenges: Background Clutter



Challenges: Intraclass variation

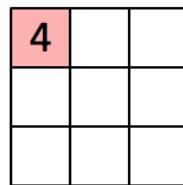


Week 2 복습 - Convolutional Neural Networks의 구성요소(Convolution, Pooling)

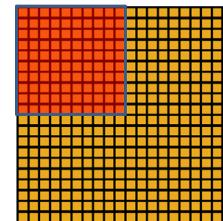


1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image



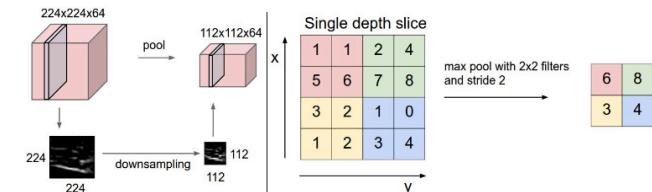
Convolved Feature



Convolved feature

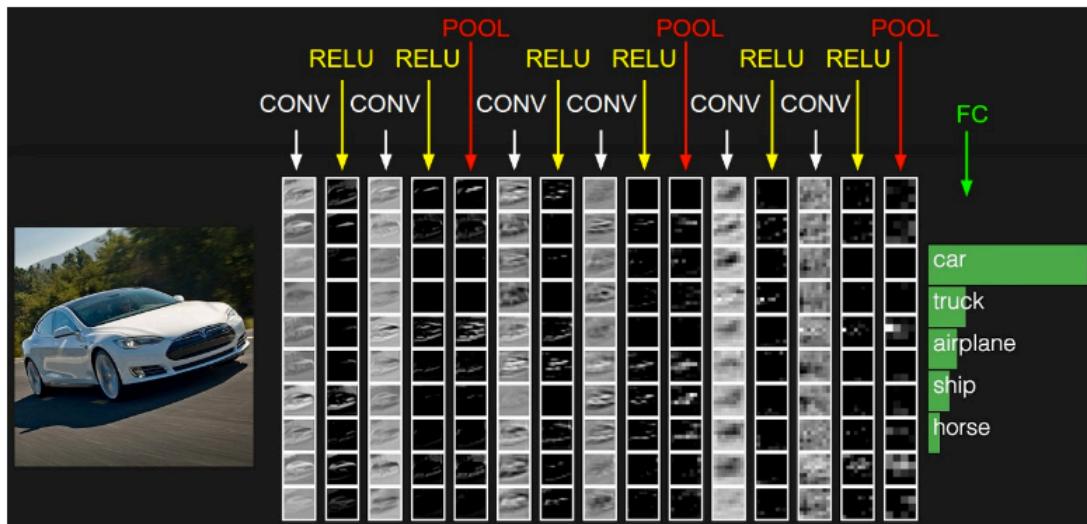


Pooled feature

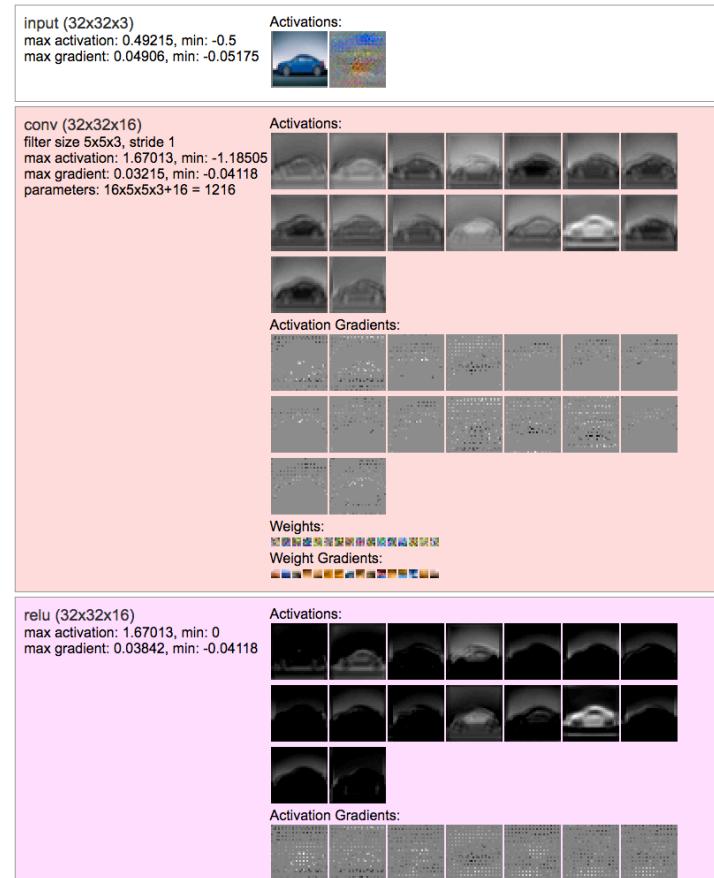


Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. Left: In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. Right: The most common downsampling operation is max, giving rise to max pooling, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

Week 2 복습 – Convolutional Neural Networks의 구성요소(Convolution, Pooling)



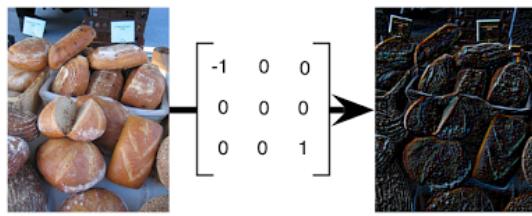
The activations of an example ConvNet architecture. The initial volume stores the raw image pixels (left) and the last volume stores the class scores (right). Each volume of activations along the processing path is shown as a column. Since it's difficult to visualize 3D volumes, we lay out each volume's slices in rows. The last layer volume holds the scores for each class, but here we only visualize the sorted top 5 scores, and print the labels of each one. The full [web-based demo](#) is shown in the header of our website. The architecture shown here is a tiny VGG Net, which we will discuss later.



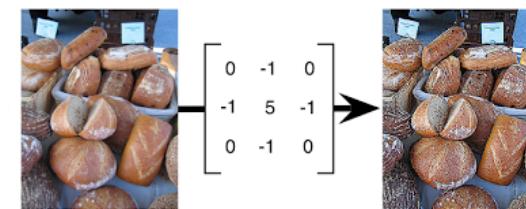
Week 2 복습 – Convolution Operation

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau) d\tau$$

Edge Detection Kernel을 이용한 경우



Sharpness Kernel을 이용한 경우



Some basic kernels



input



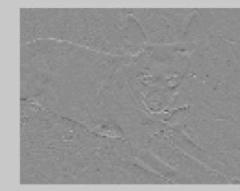
blur



sharpen



edge

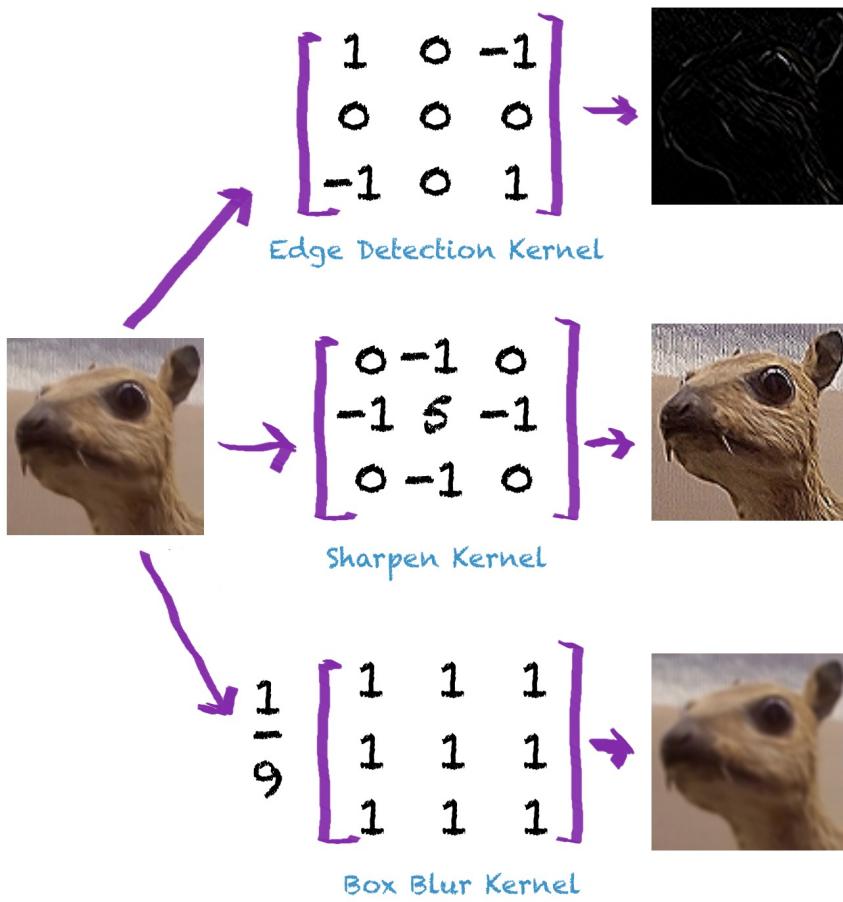


top sobel



emboss

Week 2 복습 – Convolution, Pooling



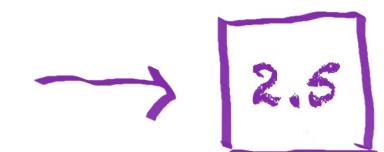
Max Pooling

4	2
1	3



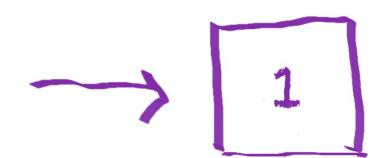
Average Pooling

4	2
1	3

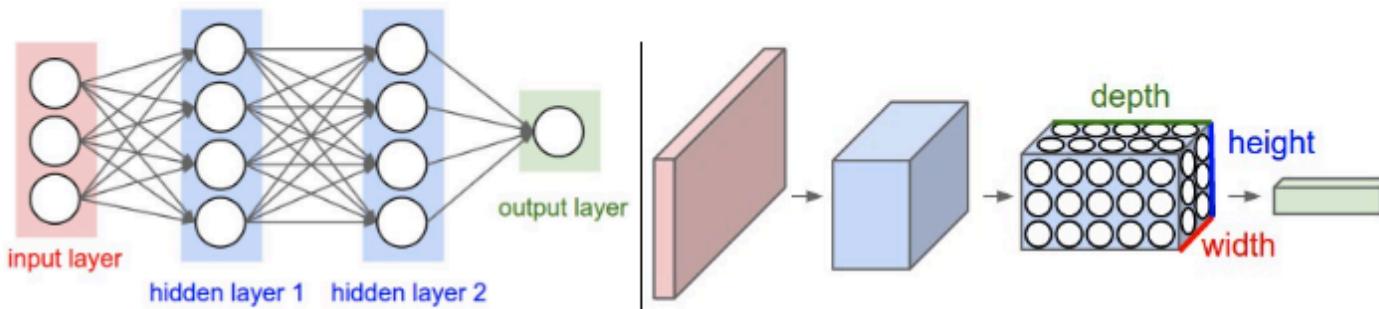


Min Pooling

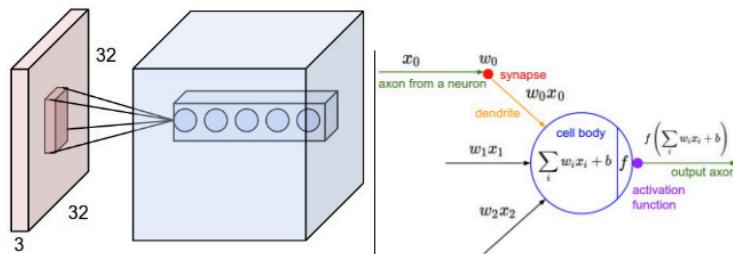
4	2
1	3



Week 2 복습 – CNNs – 3차원 레이터 표현

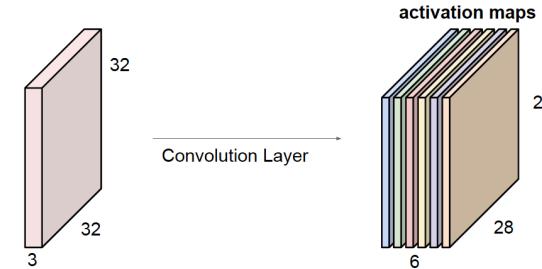


Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).



Left: An example input volume in red (e.g. a 32x32x3 CIFAR-10 image), and an example volume of neurons in the first Convolutional layer. Each neuron in the convolutional layer is connected only to a local region in the input volume spatially, but to the full depth (i.e. all color channels). Note, there are multiple neurons (5 in this example) along the depth, all looking at the same region in the input - see discussion of depth columns in text below. Right: The neurons from the Neural Network chapter remain unchanged: They still compute a dot product of their weights with the input followed by a non-linearity, but their connectivity is now restricted to be local spatially.

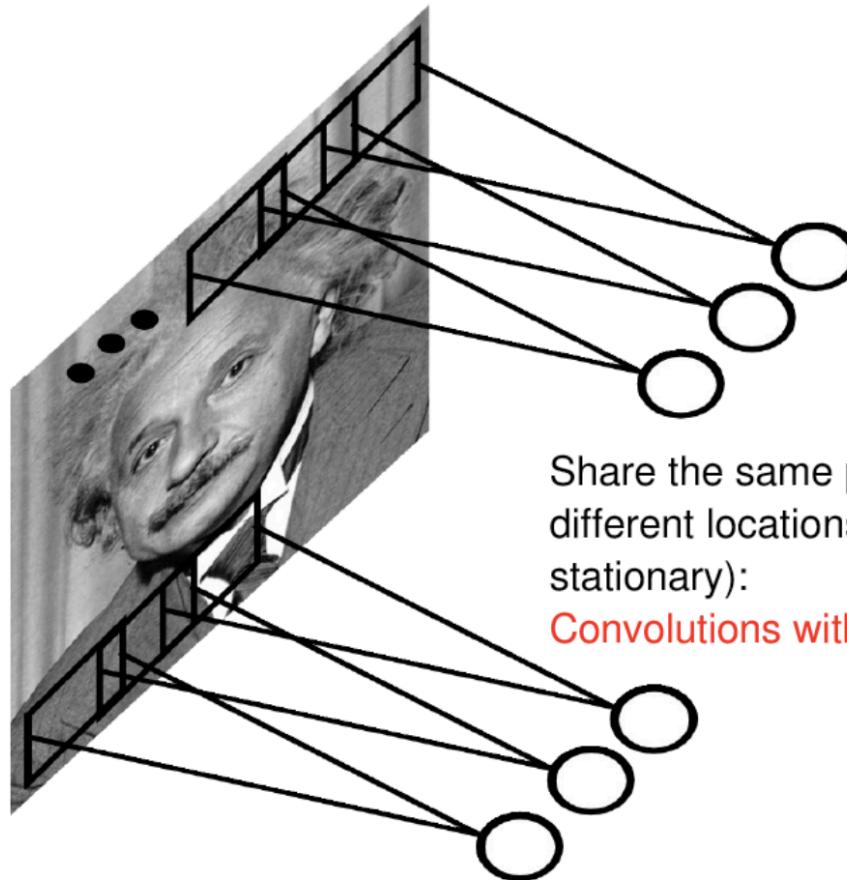
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

Week 2 복습 – 파라미터 공유(Parameter Sharing)

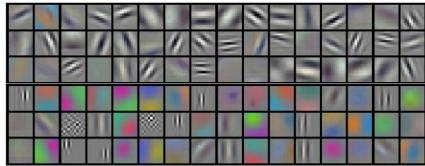
Weight sharing: Parameter saving



Share the same parameters across
different locations (assuming input is
stationary):
Convolutions with learned kernels

Week 2 복습 - CNNs의 볼륨 크기를 결정하는 요소 - 깊이, stride, 제로 패딩

출력 볼륨의 가로.세로 크기
 $(W-F+2P)/S + 1$



Example filters learned by Krizhevsky et al. Each of the 96 filters shown here is of size $[1 \times 1 \times 3]$, and each one is shared by the 55*55 neurons in one depth slice. Notice that the parameter sharing assumption is relatively reasonable: If detecting a horizontal edge is important at some location in the image, it should intuitively be useful at some other location as well due to the translationally-invariant structure of images. There is therefore no need to relearn to detect a horizontal edge at every one of the 55*55 distinct locations in the Conv layer output volume.

출력 볼륨의 깊이
 $K(\text{커널 개수})$

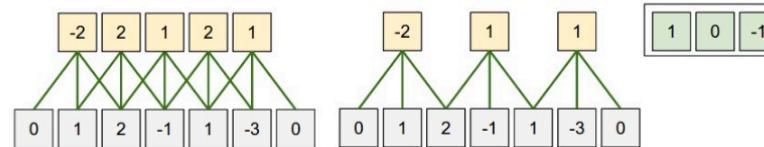
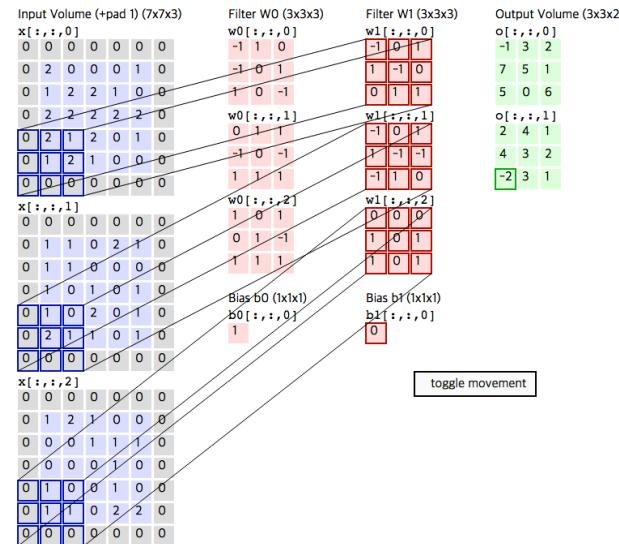


Illustration of spatial arrangement. In this example there is only one spatial dimension (x-axis), one neuron with a receptive field size of $F = 3$, the input size is $W = 5$, and there is zero padding of $P = 1$. Left: The neuron strided across the input in stride of $S = 1$, giving output of size $(5 - 3 + 2)/1 + 1 = 5$. Right: The neuron uses stride of $S = 2$, giving output of size $(5 - 3 + 2)/2 + 1 = 3$. Notice that stride $S = 3$ could not be used since it wouldn't fit neatly across the volume. In terms of the equation, this can be determined since $(5 - 3 + 2) = 4$ is not divisible by 3.

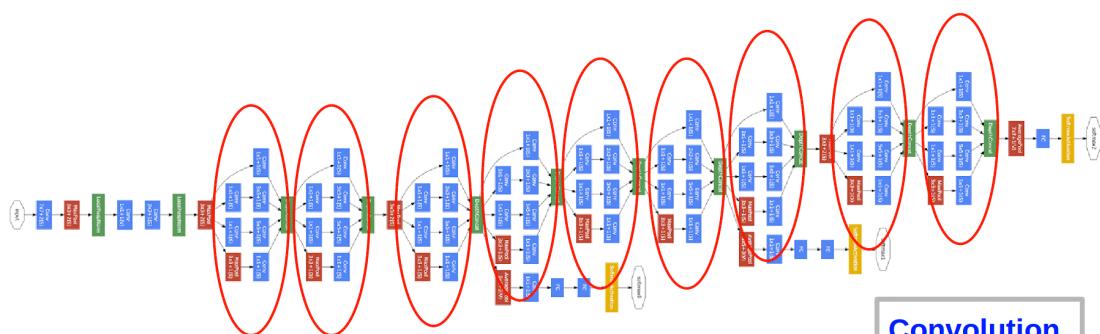
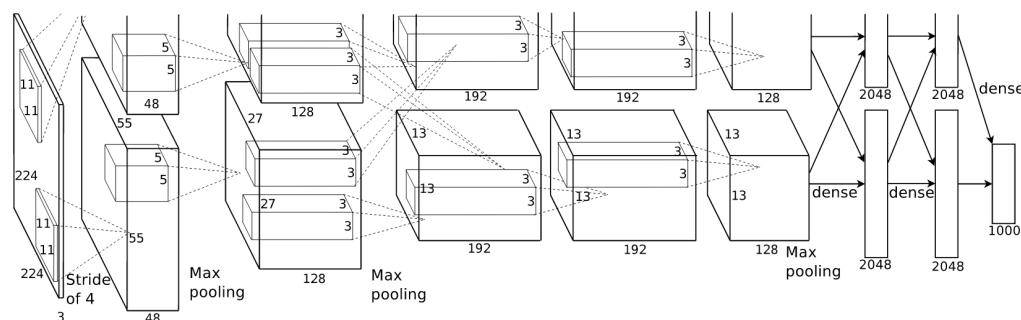
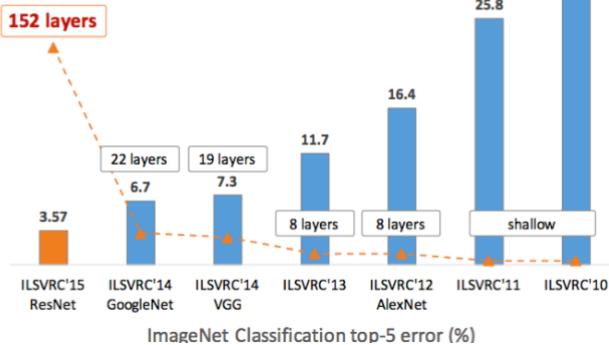
The neuron weights are in this example $[1, 0, 1]$ (shown on very right), and its bias is zero. These weights are shared across all yellow neurons (see parameter sharing below).



0	0	0	0	0	0	0	
0							
0							
0							
0							

Week 2 복습 - AlexNet, VGGNet, GoogLeNet

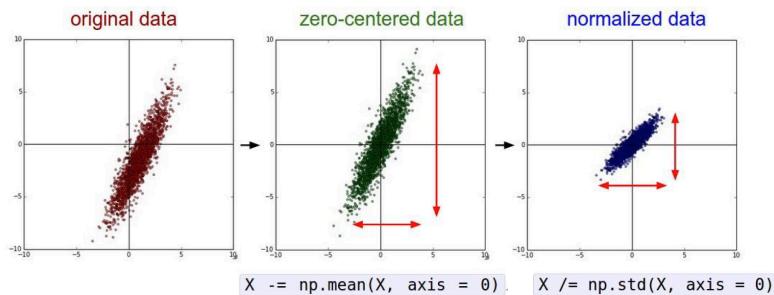
Revolution of Depth



**Convolution
Pooling
Softmax
Concat/Normalize**

Week 2 복습 – Data Preprocessing

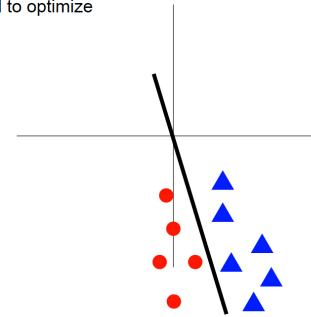
Step 1: Preprocess the data



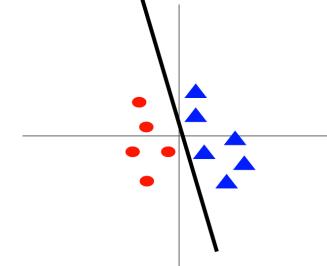
(Assume $X [NxD]$ is data matrix,
each example in a row)

Last time: Data Preprocessing

Before normalization: classification loss
very sensitive to changes in weight matrix;
hard to optimize



After normalization: less sensitive to small
changes in weights; easier to optimize



TLDR: In practice for Images: center only

e.g. consider CIFAR-10 example with $[32,32,3]$ images

- Subtract the mean image (e.g. AlexNet)
(mean image = $[32,32,3]$ array)
- Subtract per-channel mean (e.g. VGGNet)
(mean along each channel = 3 numbers)

Not common to normalize
variance, to do PCA or
whitening

Week 2 복습 – AlexNet

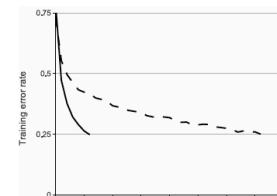
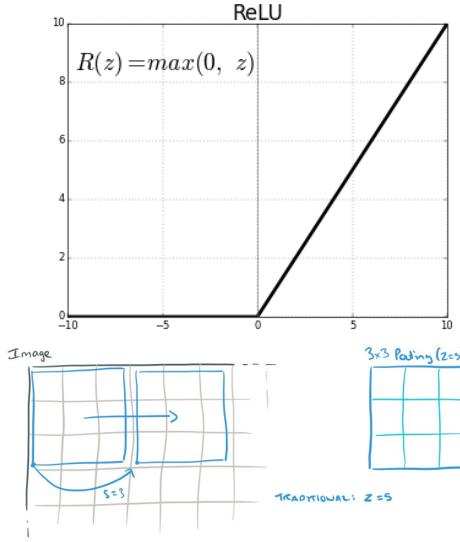
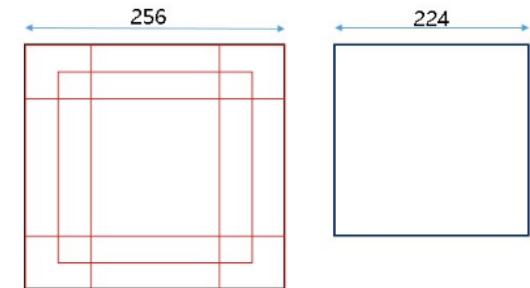
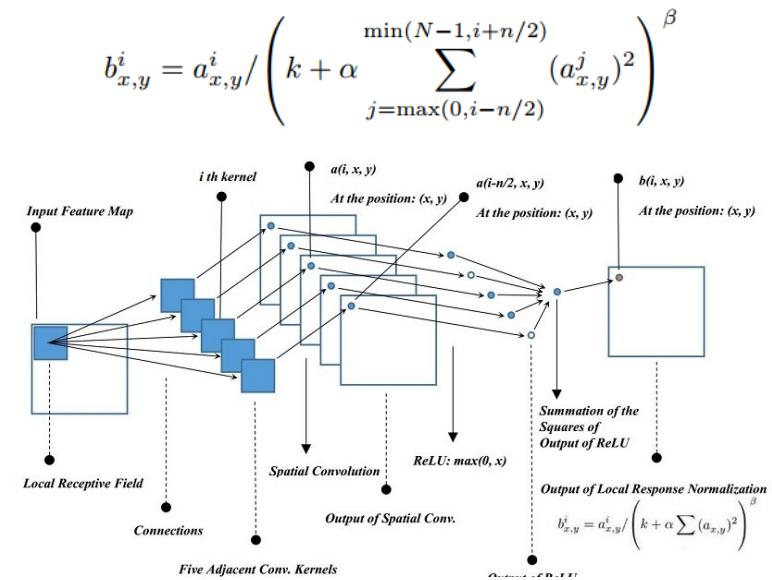
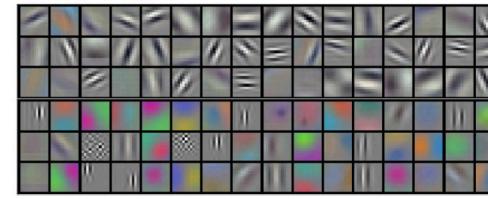
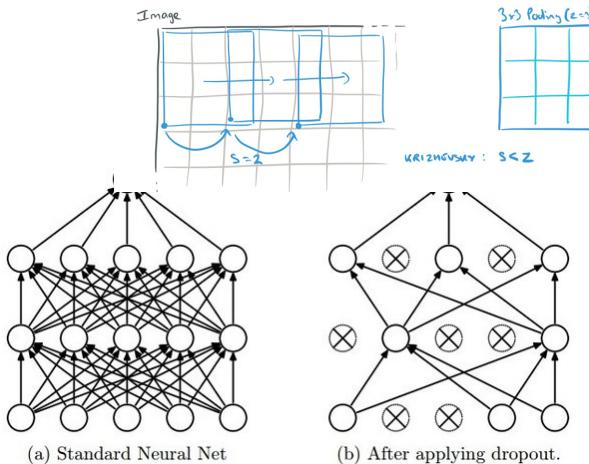


Figure 1: A four-layer convolutional neural network with ReLUs (solid line) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons (dashed line). The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.



(a) Standard Neural Net

(b) After applying dropout.

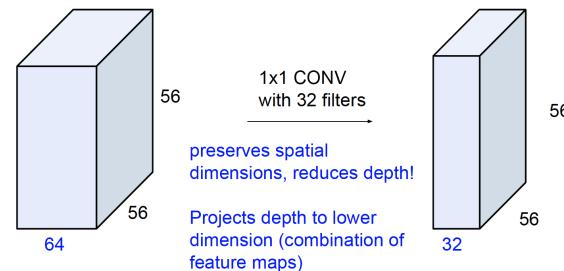
Week 2 복습 – VGGNet

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: Number of parameters (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Reminder: 1x1 convolutions



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: $(3*3*512)*512 = 2,359,296$

POOL2: [7x7x512] memory: $7*7*512=25K$ params: 0

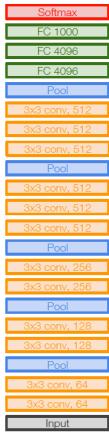
FC: [1x1x4096] memory: 4096 params: $7*7*512*4096 = 102,760,448$

FC: [1x1x4096] memory: 4096 params: $4096*4096 = 16,777,216$

FC: [1x1x1000] memory: 1000 params: $4096*1000 = 4,096,000$

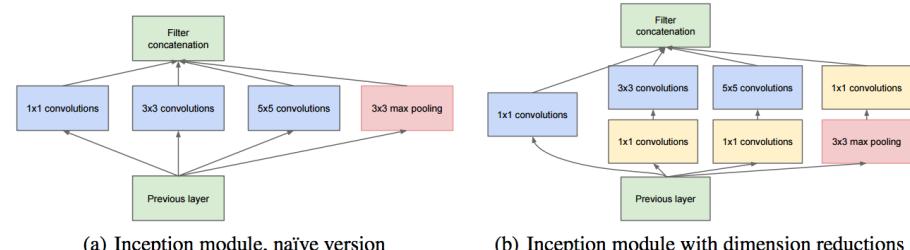
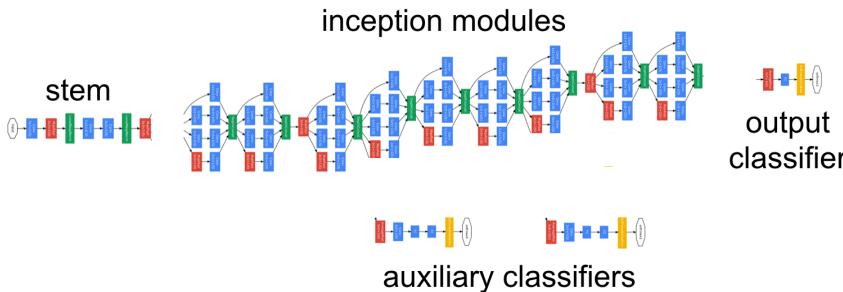
TOTAL memory: $24M * 4$ bytes $\sim= 96MB / \text{image}$ (only forward! $\sim*2$ for bwd)

TOTAL params: 138M parameters



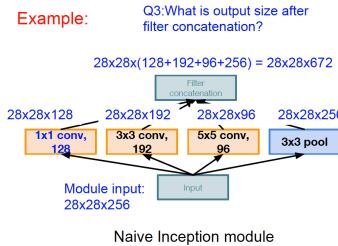
VGG16

Week 2 복습 – GoogLeNet



Case Study: GoogLeNet

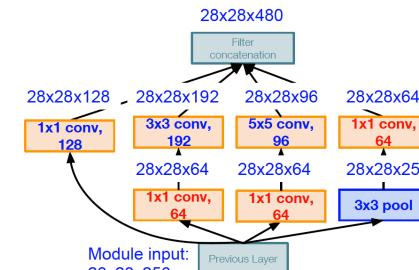
[Szegedy et al., 2014]



Q: What is the problem with this?
 [Hint: Computational complexity]

Case Study: GoogLeNet

[Szegedy et al., 2014]



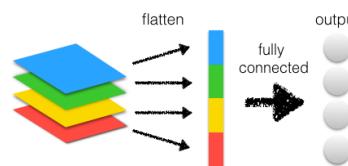
Using same parallel layers as naive example, and adding “1x1 conv, 64 filter” bottlenecks:

Conv Ops:

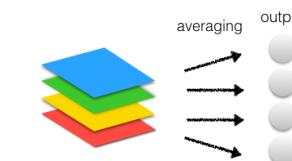
[1x1 conv, 64] $28 \times 28 \times 64 \times 1 \times 1 \times 256$
 [1x1 conv, 64] $28 \times 28 \times 64 \times 1 \times 1 \times 256$
 [1x1 conv, 128] $28 \times 28 \times 128 \times 1 \times 1 \times 256$
 [3x3 conv, 192] $28 \times 28 \times 192 \times 3 \times 3 \times 64$
 [5x5 conv, 96] $28 \times 28 \times 96 \times 5 \times 5 \times 64$
 [1x1 conv, 64] $28 \times 28 \times 64 \times 1 \times 1 \times 256$
Total: 358M ops

Compared to 854M ops for naive version
 Bottleneck can also reduce depth after pooling layer

Fully Connected Layer



Global Average Pooling



Week3의 학습목표

▣ 3강의 학습목표 :

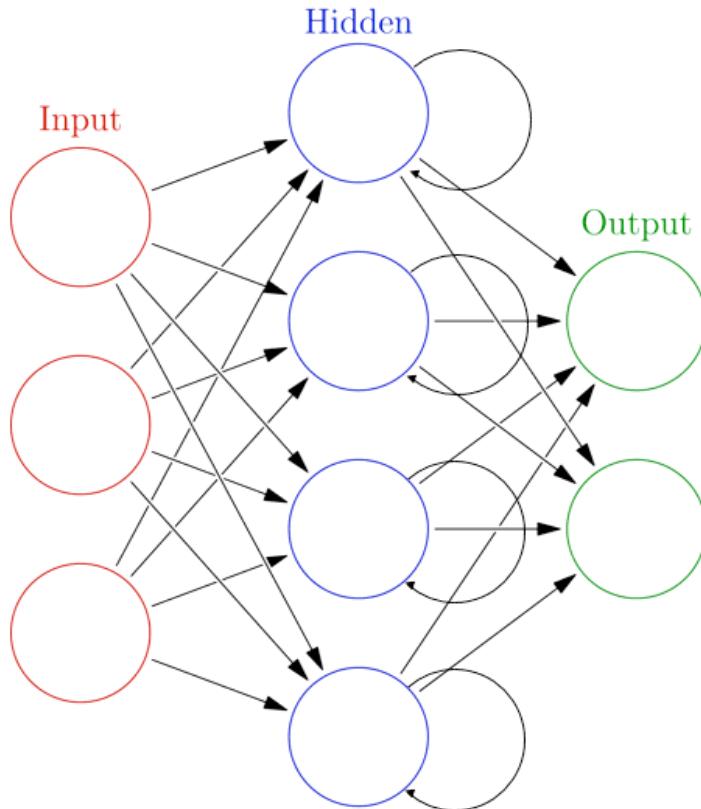
1. RNN, LSTM, GRU의 Architecture와 디자인 철학을 이해한다.
2. Regularization의 개념과 Regularization을 수행할수 있는 기법들(e.g. Regularization Term, Batch Normalization, ...)을 학습한다.
3. TensorFlow를 이용해서 Language Modelling을 위한 RNN을 구현해보자.

Outline

- ▣ Recurrent Neural Networks(RNNs)
- ▣ Bidirectional Recurrent Neural Networks(Bidirectional RNNs)
- ▣ Vanishing Gradient Problem 분석
- ▣ Long-Short Term Memory(LSTM) Networks
- ▣ Gate Recurrent Unit(GRU)
- ▣ Character-level Language Modeling을 위한 RNNs
- ▣ Ensemble Learning, Regularization, Batch Normalization
- ▣ TensorFlow를 이용한 Character-level Language Modelling 구현
- ▣ TensorFlow를 이용한 PTB Dataset Language Modelling 구현
- ▣ Word2Vec 소개 (Optional)

Recurrent Neural Networks(RNNs)

- Recurrent Neural Networks는 기본적인 Neural Networks 구조에 이전 시간 ($t-1$)의 Hidden Layer의 output을 다음 시간(t)에 Hidden Layer로 다시 집어넣는 경로가 추가된 형태이다.



Hidden Unit에서의 연산

$$a_h^t = \sum_{i=1}^I w_{ih} x_i^t + \sum_{h'=1}^H w_{h'h} b_{h'}^{t-1}$$
$$b_h^t = \sigma_h(a_h^t)$$

Output Unit에서의 연산

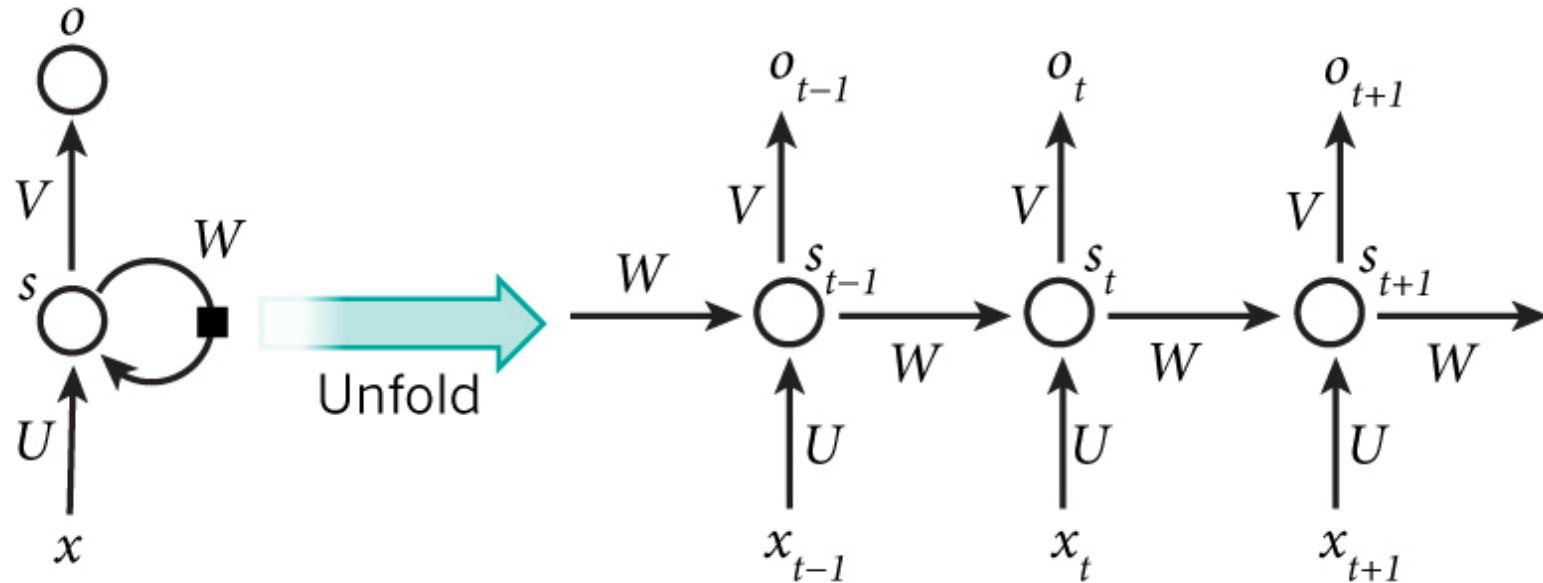
$$a_k^t = \sum_{h=1}^H w_{hk} x_h^t$$

Recurrent Neural Networks(RNNs)의 장점

- 이런 구조를 통해 얻을 수 있는 장점은 이전 상태에 대한 정보를 일종의 “**메모리(memory)**” 형태로 저장할 수 있다는 점이다. 이는 앞에서 얻은 정보가 뒤에서 얻은 데이터와 연관관계를 가지는 데이터를 다룰 때 매우 강력한 효과를 발휘한다.
- 실제적인 예로, 인간의 **언어(Natural Language)**는 앞뒤 **문맥(Context)**을 가지고 있기 때문에 RNNs을 적용하기에 매우 적합하다. 예를 들어 “푸른 하늘에 ○○이 떠있다.”라는 문장에서 ○○에 들어갈 단어를 예측하고자 한다면, 우리는 앞뒤 정보인 “푸른”, “하늘”, “떠있다.”라는 단어를 통해 ○○에 들어갈 단어가 “구름”이라는 것을 쉽게 예측할 수 있다.- 좀 더 명확히 말하면, 단순한 RNNs은 앞의 정보만을 저장 할 수 있다. 뒤에 정보도 저장하고 싶다면, 뒤에 살펴볼 Bidirectional RNNs 구조를 이용해야만 한다.-

Unfolded Representation

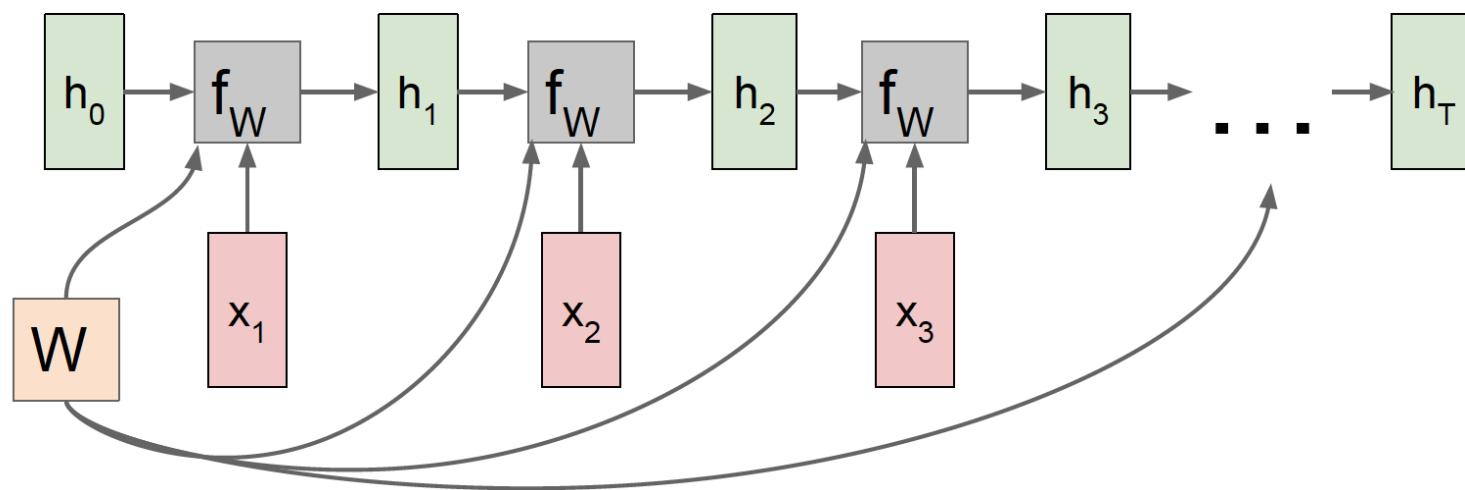
- ▣ RNNs을 다른 관점으로 바라보면, 일반적인 Neural Networks를 펼친(**Unfold**) 형태로 생각할 수도 있다.
- ▣ 예를 들어, 5개의 단어로 이루어진 문장을 RNNs의 Input으로 사용한다면, 아래와 같이 순환연결이 없는 Neural Networks를 5층(5-Layer) 쌓은 것으로 바라볼 수 있다.



Unfolded Representation

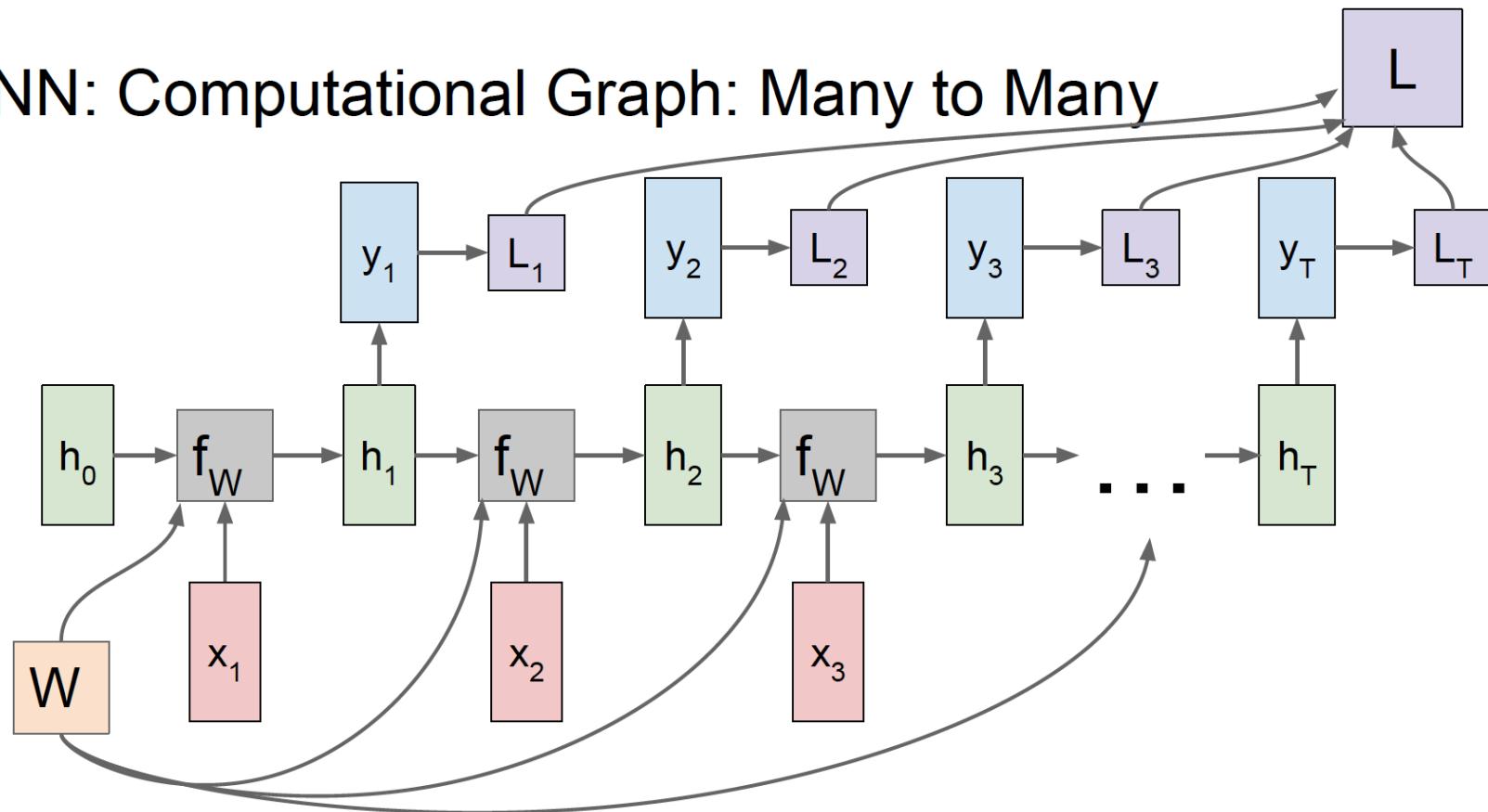
RNN: Computational Graph

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



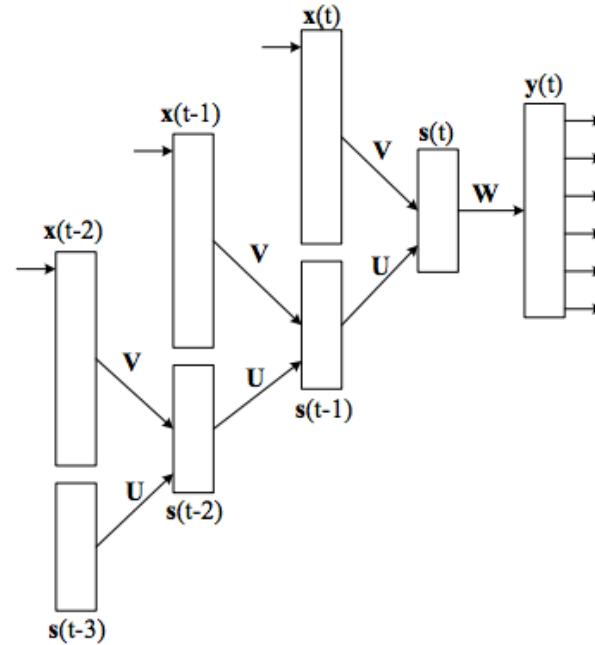
Unfolded Representation

RNN: Computational Graph: Many to Many

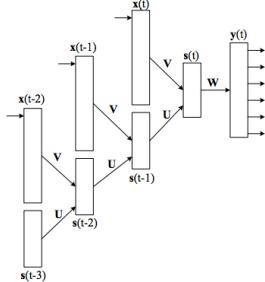


Backpropagation Through Time(BPTT)

- 위와 같은 특성들 때문에, RNNs를 학습시키기 위해서는 기존의 단순한 Neural Networks를 학습시키기 위해서 사용하던 Backpropagation 알고리즘과는 조금은 다른 방법이 필요하다. 이를 **Backpropagation Through Time(BPTT)** 알고리즘이라고 부른다.
- 다시, 위에 언급한 펼쳐진 형태(Unfolded)의 RNNs 구조에서 Weight들을 다음과 같이 U,V,W로 나타내보자.



Backpropagation in Unfolded RNNs



- 먼저, Backpropagation Through Time(BPTT)를 알아보기 전에, 먼저 기존의 Backpropagation(BP) 알고리즘 계산식을 Unfolded RNNs 구조에 적용해보자. Backpropagation(BP)에서 역전파할 값을 얻기 위해서 Output Layer에서의 에러인 예측값과 트루값과의 오차를 다음과 같이 계산하자.

$$e_o(t) = d(t) - y(t)$$

- 그럼 hidden layer에서 output 간의 가중치인 w 는 다음과 같이 업데이트 할 수 있다.

$$W(t+1) = W(t) + \eta s(t)e_o(t)^T$$

- 그런 다음 output layer의 에러의 기울기(gradient) hidden layer로 다음과 같이 역전파된다.

$$e_h(t) = d_h(e_o(t)^T V, t) \quad d_{hj}(x, t) = x f'(net_j)$$

- 이제, Input Layer에서 Hidden Layer의 가중치 V 는 다음과 같이 업데이트 된다.

$$V(t+1) = V(t) + \eta x(t)e_h(t)^T$$

Backpropagation을 이용한 gradient 계산

1. Set $\Delta W^{(l)} := 0$, $\Delta b^{(l)} := 0$ (matrix/vector of zeros) for all l .

2. For $i = 1$ to m ,

a. Use backpropagation to compute $\nabla_{W^{(l)}} J(W, b; x, y)$ and $\nabla_{b^{(l)}} J(W, b; x, y)$

b. Set $\Delta W^{(l)} := \Delta W^{(l)} + \nabla_{W^{(l)}} J(W, b; x, y)$

c. Set $\Delta b^{(l)} := \Delta b^{(l)} + \nabla_{b^{(l)}} J(W, b; x, y)$

3. Update the parameters:

$$W^{(l)} = W^{(l)} - \alpha \left[\left(\frac{1}{m} \Delta W^{(l)} \right) + \lambda W^{(l)} \right]$$

$$b^{(l)} = b^{(l)} - \alpha \left[\frac{1}{m} \Delta b^{(l)} \right]$$

- Hidden Layer에서 Hidden Layer로 순환되는 경로의 가중치인 U 는 다음과 같이 업데이트 된다.

$$U(t+1) = U(t) + \eta s(t-1)e_h(t)^T$$

Gradient Descent를 이용한 parameter update

Backpropagation Through Time(BPTT)

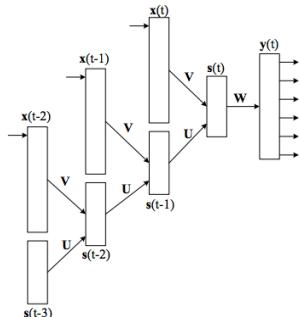
- 이제 RNNs에 더 적합한 **Backpropagation Through Time(BPTT)** 알고리즘을 살펴보자. BPTT에서 hidden layer에서 output간의 가중치인 W 는 BP와 똑같은 형태로 다음과 같이 업데이트 할 수 있다.

$$W(t+1) = W(t) + \eta s(t)e_o(t)^T$$

- 하지만, Input Layer에서 Hidden Layer의 가중치 V 와 Hidden Layer에서 Hidden Layer로 순환되는 경로의 가중치인 U 는 다음과 같은 일정 시간에서의 에러 값들을 계산하고

$$e_h(t-\tau-1) = d_h(e_h(t-\tau)U, t-\tau-1) \quad d_{hj}(x, t) = xf'(net_j)$$

- 일정 시간 동안 에러 값을 합한 값을 역전파함으로써 Input Layer에서 Hidden Layer의 가중치 V 와 Hidden Layer에서 Hidden Layer로 순환되는 경로의 가중치인 U 를 업데이트한다.

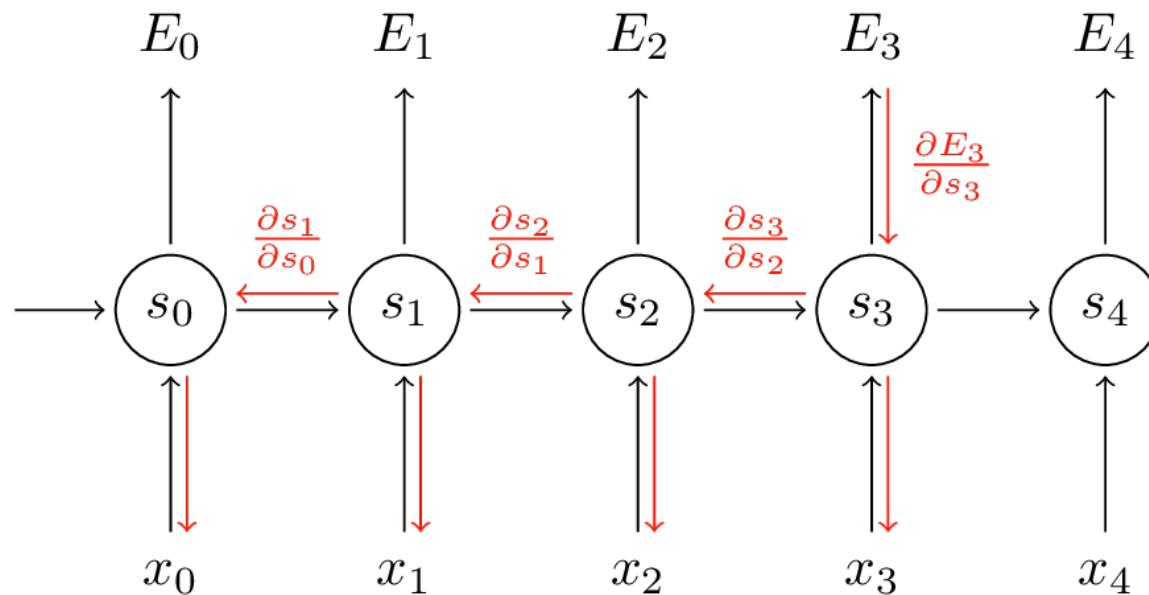


$$V(t+1) = V(t) + \eta \sum_{z=0}^T x(t-z)e_h(t-z)^T$$

$$U(t+1) = U(t) + \eta \sum_{z=0}^T s(t-z-1)e_h(t-z)^T$$

Backpropagation Through Time(BPTT)을 사용하는 이유

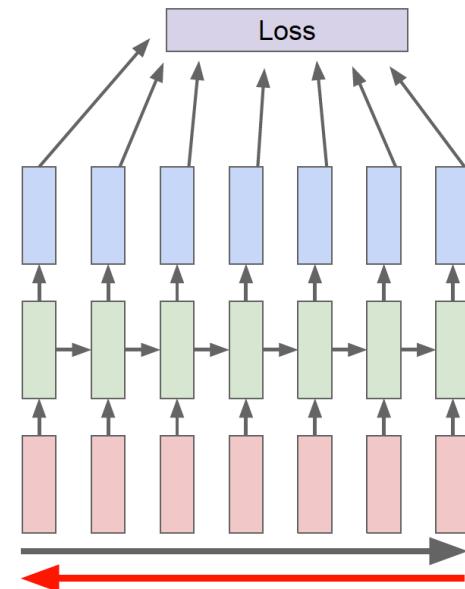
- 이렇게 업데이트를 진행하면, 현재 시간의 에러를 과거 시간의 상태까지 역전파할 수 있다. 이를 그림으로 나타내면 아래와 같다.



Truncated-BPTT

- ▣ 하지만 이런 식으로 BPTT를 진행하면, 시퀀스의 처음부터 끝까지를 모두 에러를 역전파해야 되기 때문에, 계산량을 줄이기 위해서 현재 time step에서 일정시간 이전까지만-보통 5 time step이전까지 본다.- 에러를 역전파하는 **Truncated-Backpropagation Through Time(생략된-BPTT)**를 많이 사용한다.

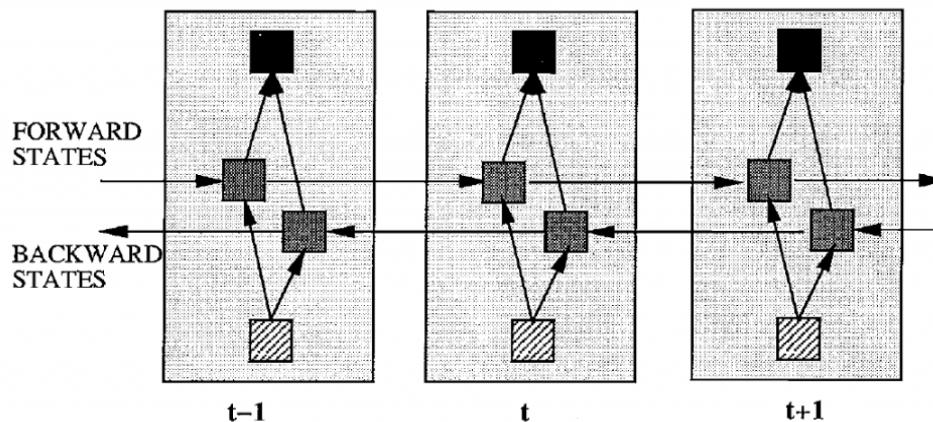
Truncated Backpropagation through time



Run forward and backward
through chunks of the
sequence instead of whole
sequence

Bidirectional Recurrent Neural Networks(RNNs)

- 이번 시간에는 과거의 상태뿐만 아니라, 미래의 상태까지 고려하는 확장된 Recurrent Neural Networks(RNNs) 형태인 Bidirectional Recurrent Neural Networks(BRNNs)에 대해 알아보자.



Unfolded Representation

Hidden Unit에서의 연산

$$\vec{h}_t = \sigma(W_{x \rightarrow h}^{\rightarrow} x_t + W_{h \rightarrow h}^{\rightarrow} \vec{h}_{t-1} + b_{h \rightarrow})$$

$$\overleftarrow{h}_t = \sigma(W_{x \rightarrow h}^{\leftarrow} x_t + W_{h \rightarrow h}^{\leftarrow} \overleftarrow{h}_{t+1} + b_{h \leftarrow})$$

Output Unit에서의 연산

$$y_t = W_{h \rightarrow y}^{\rightarrow} \vec{h}_t + W_{h \rightarrow y}^{\leftarrow} \overleftarrow{h}_t + b_y$$

Bidirectional Recurrent Neural Networks(RNNs)의 장점

- ▣ Bidirectional Recurrent Neural Networks(BRNNs)을 이용하면, 이렇게 이전 정보와 이후 정보를 모두 저장할 수 있다.
- ▣ 시계열 데이터의 현재 시간 이전 정보뿐만 아니라, 이후 정보까지 저장해서 활용할 수 있다면 더 좋은 성능을 기대할 수 있다. 예를 들어 “푸른 하늘에 ○○이 떠있다.”라는 문장에서 ○○에 들어갈 단어를 예측하고자 한다면, 우리는 앞의 정보인 “푸른”, “하늘”이라는 정보를 가지고 ○○에 들어갈 단어가 “구름”이라고 예측할 수도 있지만, 뒤의 정보인 “떠있다.”라는 단어까지 함께 고려하면 더 높은 확률로 정답이 “구름”이라는 것을 예측할 수 있을 것이다.

Algorithm 1 BRNNs Forward Pass

```
1: for t=1 to T do
   Do forward pass for the forward hidden layer, storing activations at
   each timestep
2: end for
3: for t=T to 1 do
   Do forward pass for the backward hidden layer, storing activations
   at each timestep
4: end for
5: for t=1 to T do
   Do forward pass for the output layer, using the stored activations
   from both hidden layers
6: end for
```

BPTT in Bidirectional RNNs

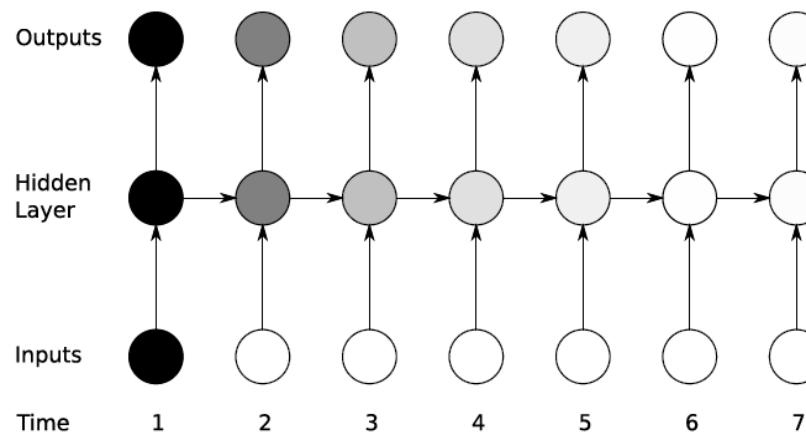
- BRNNs의 학습은 일반적 RNNs와 같은 Backpropagation Through Time(BPTT) 알고리즘을 이용한다.
- 단지 가중치를 업데이트 할 때, Forward hidden layer와 Backward hidden layer에 input 값을 반대 방향(opposite)으로 집어 넣고, Output layer 가중치는 두 방향의 Hidden layer에 모든 input이 적용된 후에 업데이트 한다는 점이 차이점이다.

Algorithm 1 BRNNs Backward Pass

```
1: for  $t=T$  to  $1$  do
   Do BPTT backward pass for the forward hidden layer, using the
   stored  $\delta$  terms from the output layer
2: end for
3: for  $t=T$  to  $1$  do
   Do BPTT backward pass for the forward hidden layer, using the
   stored  $\delta$  terms from the output layer
4: end for
5: for  $t=1$  to  $T$  do
   Do BPTT backward pass for the backward hidden layer, using the
   stored  $\delta$  terms from the output layer
6: end for
```

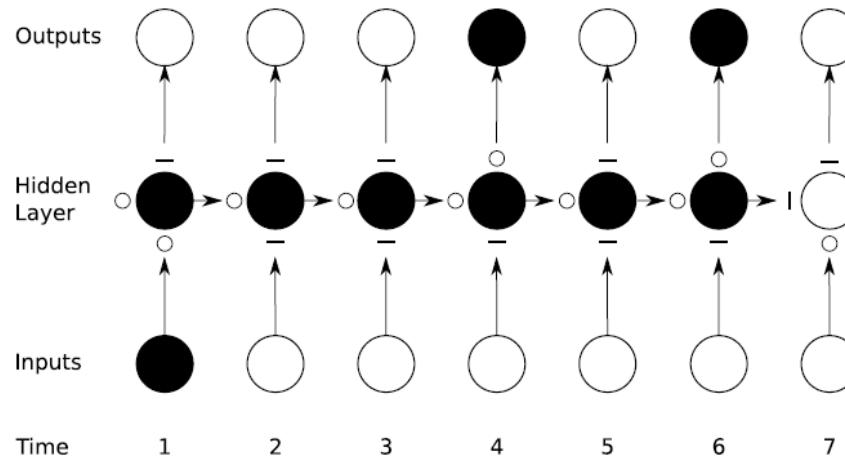
Vanishing Gradient Problem

- 아래 그림을 보면, 시간 1에서 들어온 인풋 데이터는 처음에는 네트워크 학습에 강한 영향력을 끼치다가, 점점 새로운 인풋이 들어오면서 시간 1에서 들어온 인풋의 영향력은 감소하고, 결국에는 시간 1에서 들어온 인풋은 네트워크 학습에 아무런 영향을 끼치지 못하게 된다. 이는 네트워크 학습에 사용하는 경사도(Gradient)가 사라지는(Vanishing) 현상이므로 이 문제를 “**Vanishing Gradient Problem**”이라고 부른다.



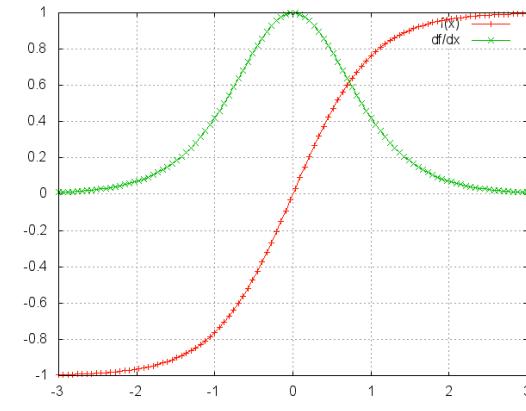
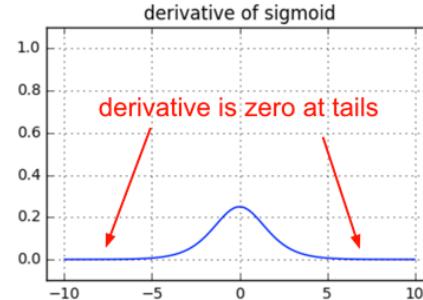
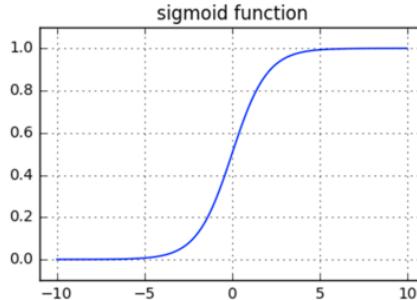
LSTM Networks을 이용한 Vanishing Gradient Problem의 해결

- 이 문제를 해결하기 위해서 좀더 복잡한 RNNs 구조인 Long Short-Term Memory(LSTM) Networks가 제안되었다. LSTM Networks는 RNNs의 **Hidden Layer**를 **Input Gate**, **Output Gate**, **Forget Gate**라는 세가지 게이트로 구성된 **Memory Block**으로 대체한 구조이다.
- 아래 그림에서 ○는 gate가 열려있음을, -는 게이트가 닫혀있음을 뜻한다. 아래 그림에서와 같이, 시간 1에서의 인풋데이터를 받은 이후에 **Input Gate**를 닫아버려서 새로운 인풋을 받지 않고, **Forget Gate**를 열어놔서 시간 1에서의 인풋데이터를 계속해서 전달받으면, 시간 1에서의 인풋의 영향력을 계속해서 가져갈 수 있다. 마지막으로, **Output Gate**를 열고 닫으면서, 시간 1에서의 인풋데이터의 영향력을 반영하거나 반영하지 않을 수 있다.



Vanishing Gradient Problem에 대한 자세한 분석

- Sigmoid, Tanh activation function의 문제점 -> 입력받는 값이 일정 범위보다 크거나 작으면 **gradient가 0이 된다.**
- 따라서, 행렬에 작은 값들이 들어있고 여러 ($t-k$ 번) 행렬곱이 이루어지면 gradient는 지수 함수로 감소하고, 시간 스텝 몇 번만 지나도 사라져 버린다 (vanish!). 시퀀스에서 여러 시간 스텝이 떨어진 곳에서는 gradient가 전달되지 못하고, 먼 과거의 상태(state)는 현재 스텝의 학습에 아무 도움이 되지 못하게 된다. 즉, **long-range dependency**를 제대로 배우지 못한다.
- Vanishing gradient 문제는 RNN에서만 나타나는 것이 아니다. Deep Feedforward Neural Network에서도 마찬가지로 발생하지만, RNN은 보통 시간 스텝 횟수만큼 매우 깊은 구조이기 때문에 이 문제가 훨씬 더 잘 나타난다.

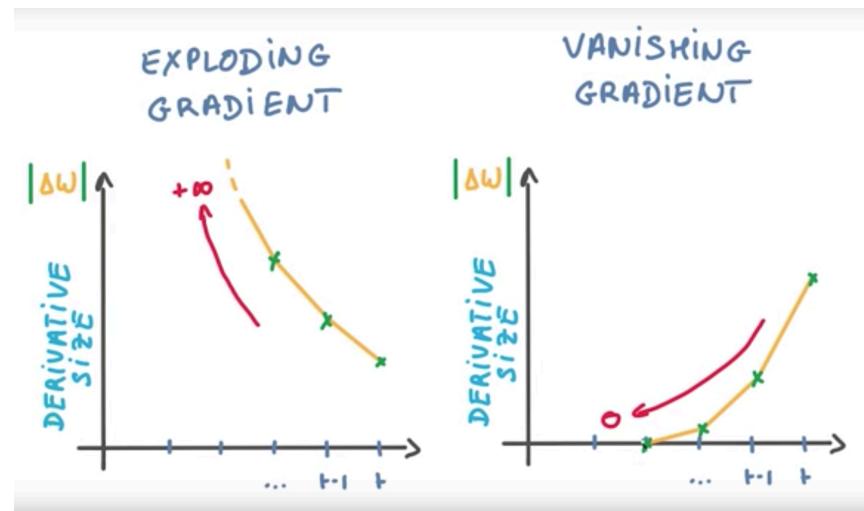


Sigmoid 함수와 미분값

Tanh 함수와 미분값

Exploding gradient & Vanishing gradient

- ▣ Gradient 계산을 보면, 자코비안 행렬 안의 값들이 크다면 activation 함수와 네트워크 파라미터 값에 따라 gradient가 사라지는게 아니라 지수 함수로 증가하는 경우도 충분히 상상해볼 수 있다. 이 문제 역시 **exploding gradient** 문제로 잘 알려져 있다.
- ▣ Vanishing gradient 문제가 더 많은 관심을 받은 이유는 두 가지이다.
 1. 하나는 exploding gradient 문제는 쉽게 알아차릴 수 있다는 점이다. **Gradient** 값들이 **NaN (not a number)**이 될 것이고 프로그램이 죽을 것이기 때문이다.
 2. 두 번째는, **gradient** 값이 너무 크다면 미리 정해준 적당한 값으로 잘라버리는 방법이 매우 쉽고 효율적으로 이 문제를 해결하기 때문이다. Vanishing gradient 문제는 언제 발생하는지 바로 확인하기가 힘들고 간단한 해결법이 없기 때문에 더 큰 문제였다.



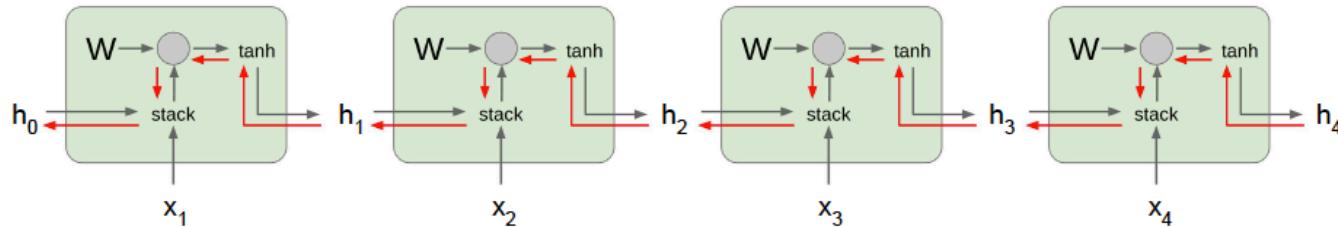
Exploding gradient Problem 해결책 – Gradient Clipping

Algorithm 1 Pseudo-code for norm clipping

```
 $\hat{g} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$ 
if  $\|\hat{g}\| \geq threshold$  then
     $\hat{g} \leftarrow \frac{threshold}{\|\hat{g}\|} \hat{g}$ 
end if
```

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1 :
Exploding gradients

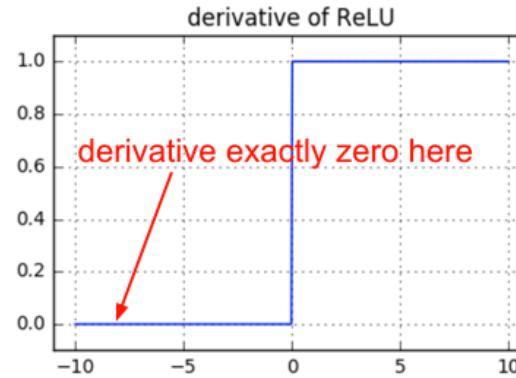
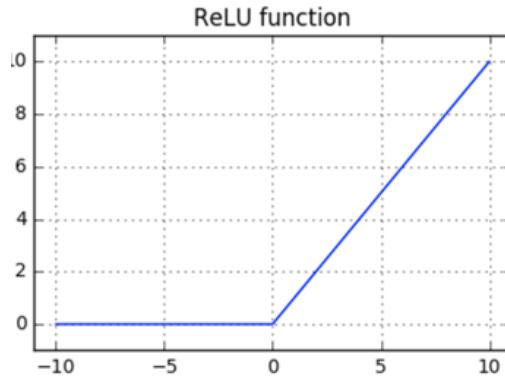
Largest singular value < 1 :
Vanishing gradients

Gradient clipping: Scale gradient if its norm is too big

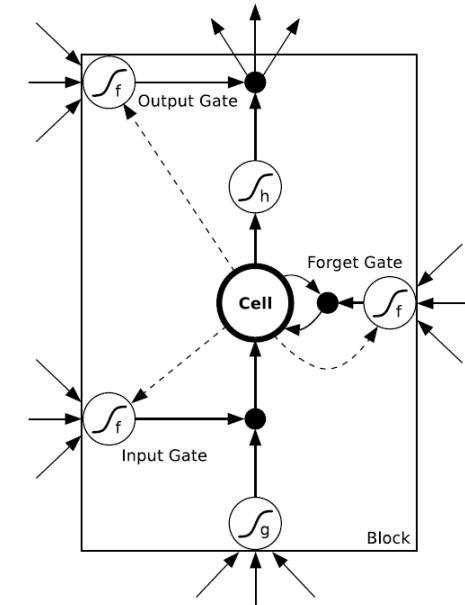
```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
    grad *= (threshold / grad_norm)
```

Vanishing Gradient Problem에 대한 해결책 – ReLU Activation, LSTM Networks

- ReLU activation function을 이용하면 큰 인풋값에 대해서도 gradient가 항상 존재하기 때문에 Vanishing Gradient Problem을 어느 정도 해결 할 수 있다.
- 하지만 더 좋은 해결책은 LSTM Networks를 이용하는 것이다.

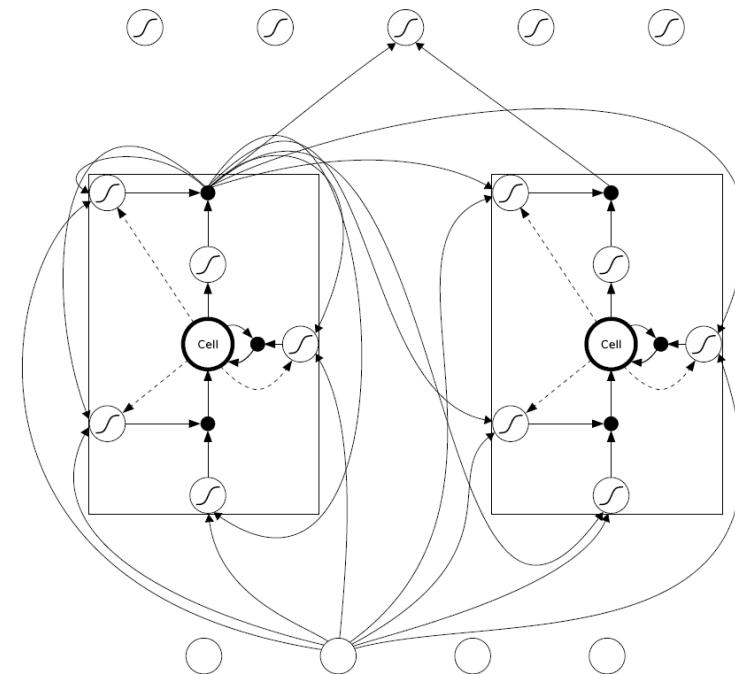
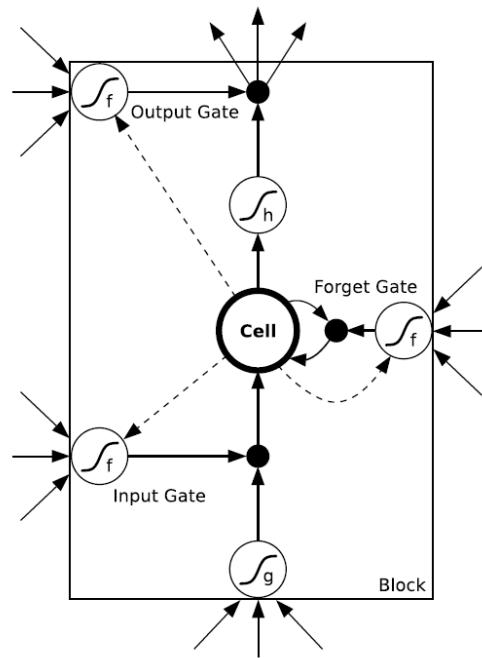


ReLU 함수와 미분값

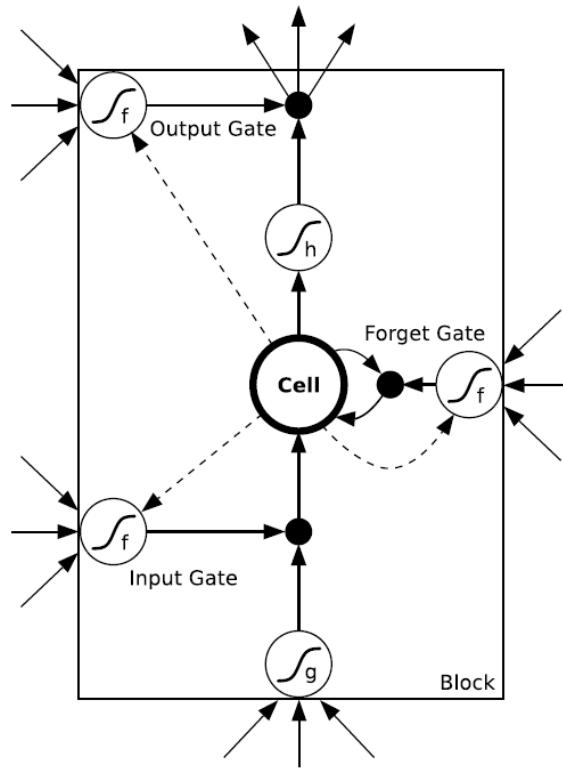


Long-Short Term Memory(LSTM) Networks

- ▣ 가장 발전된 형태의 Recurrent Neural Networks(RNNs) 구조
- ▣ 대부분의 복잡한 실제문제에서 RNNs을 적용할때 LSTM Networks 구조를 이용한다. (대부분 LSTM Networks 구조가 가장 좋은 성능을 보여준다.)



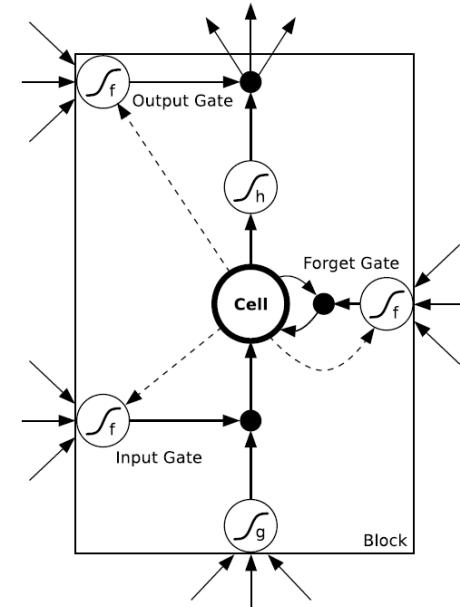
Memory Block 분석



1. 첫째로, Input Layer에서 들어오는 데이터 x 는 Input Gate, Output Gate, Forget Gate 세 Gate와 Memory Block의 입구로 전달된다.
2. 둘째로, Memory Block의 Output의 b_h 는 Input Gate, Output Gate, Forget Gate 세 Gate와 Memory Block의 입구로 전달된다.
3. 셋째로, Memory Block의 Cell의 Output s_c 는 Input Gate, Output Gate, Forget Gate 세 Gate로 전달된다.

Memory Block 분석

- 각각의 Gates에서 일어나는 연산은 아래와 같다. (f, g, h는 sigmoid와 같은 activation function이다.)



Input Gates에서의 연산

$$a_I^t = \sum_{i=1}^I w_{iI}x_i^t + \sum_{h=1}^H w_{hI}b_h^{t-1} + \sum_{c=1}^C w_{cI}s_c^{t-1}$$

$$b_I^t = f(a_I^t)$$

Forget Gates에서의 연산

$$a_F^t = \sum_{i=1}^I w_{iF}x_i^t + \sum_{h=1}^H w_{hF}b_h^{t-1} + \sum_{c=1}^C w_{cF}s_c^{t-1}$$

$$b_F^t = f(a_F^t)$$

f: Forget gate, Whether to erase cell

i: Input gate, whether to write to cell

g: Gate gate (?), How much to write to cell

o: Output gate, How much to reveal cell

Cells에서의 연산

$$a_c^t = \sum_{i=1}^I w_{ic}x_i^t + \sum_{h=1}^H w_{hc}b_h^{t-1}$$

$$s_c^t = b_F^t s_c^{t-1} + b_I^t g(a_I^t)$$

Output Gates에서의 연산

$$a_O^t = \sum_{i=1}^I w_{iO}x_i^t + \sum_{h=1}^H w_{hO}b_h^{t-1} + \sum_{c=1}^C w_{cO}s_c^t$$

$$b_O^t = f(a_O^t)$$

Cells Outputs(Memory Block의 Output에서의 연산)

$$b_c^t = b_O^t h(s_c^t)$$

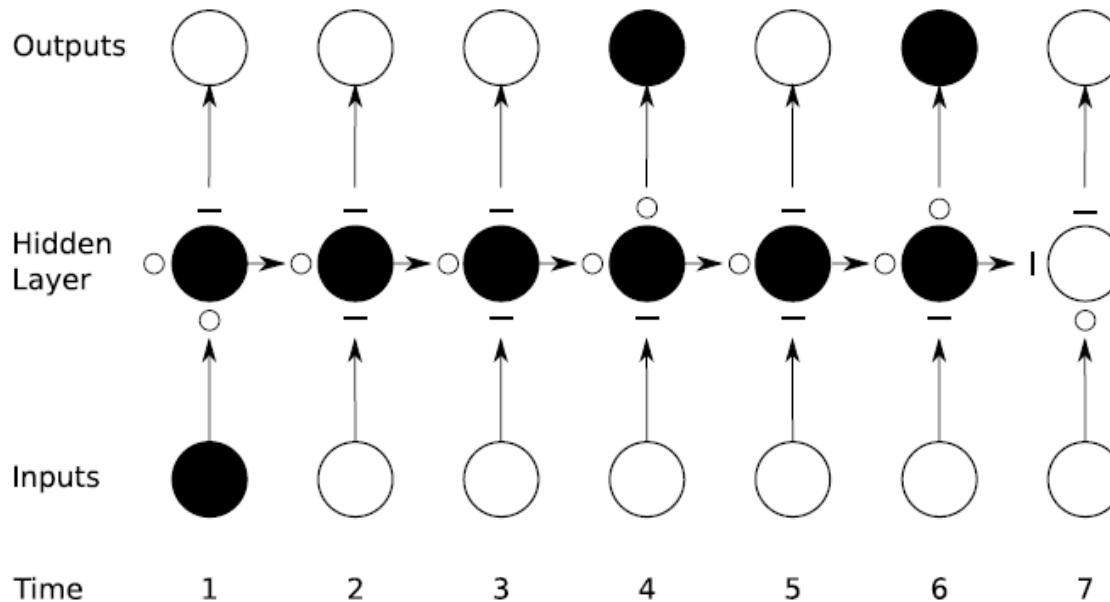
RNNs = LSTM의 특수 케이스

- 직관적으로 보면, 기본 RNN 모델은 LSTM의 특수 케이스로 생각할 수도 있습니다.
- 입력 게이트(Input Gate)를 전부 1로 두고, 까먹음 게이트(Forget Gate)를 전부 0으로 (이전 메모리는 무조건 까먹는 것으로) 하고, 출력 게이트(Output Gate)를 전부 1로 설정한다면 (메모리 값 전부를 보여줌) RNN 모델 기본형과 거의 같습니다.
- 출력값을 특정 범위 내로 압축시키는 \tanh 만 추가된 형태일 것입니다.

$$\begin{aligned}i &= \sigma(x_t U^i + s_{t-1} W^i) \\f &= \sigma(x_t U^f + s_{t-1} W^f) \\o &= \sigma(x_t U^o + s_{t-1} W^o) \\g &= \tanh(x_t U^g + s_{t-1} W^g) \\c_t &= c_{t-1} \circ f + g \circ i \\s_t &= \tanh(c_t) \circ o\end{aligned}$$

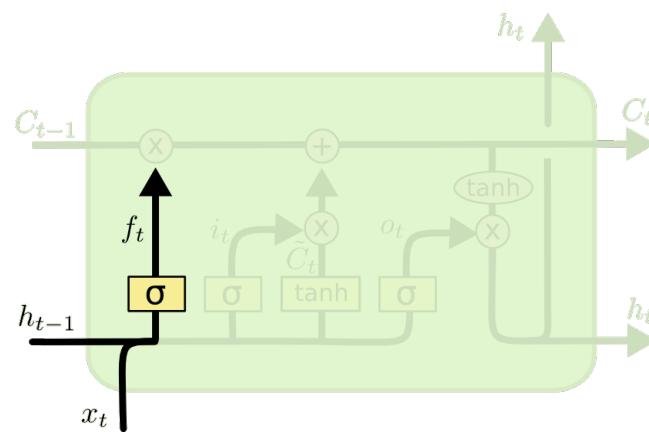
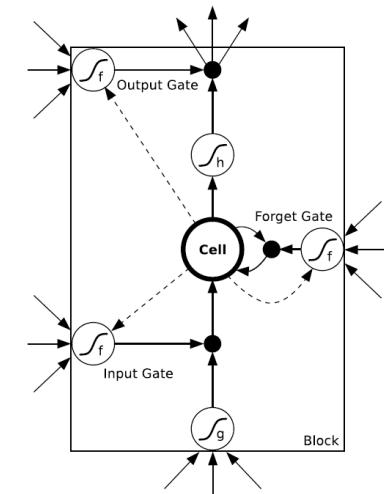
LSTM Networks을 이용한 Vanishing Gradient Problem의 해결

- 아래 그림에서 ○는 gate가 열려있음을, -는 게이트가 닫혀있음을 뜻한다. 아래 그림에서와 같이, 시간 1에서의 인풋데이터를 받은 이후에 **Input Gate**를 닫아버려서 새로운 인풋을 받지 않고, **Forget Gate**를 열어놔서 시간 1에서의 인풋데이터를 계속해서 전달받으면, 시간 1에서의 인풋의 영향력을 계속해서 가져갈 수 있다. 마지막으로, **Output Gate**를 열고 닫으면서, 시간 1에서의 인풋데이터의 영향력을 반영하거나 반영하지 않을 수 있다.



Forget Gate

- Forget Gate의 값이 1이면 "이 정보를 완전히 가져감"(열려 있음), 값이 0이면 "이 정보를 완전히 버림"(닫혀 있음)을 뜻한다.



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

f: Forget gate, Whether to erase cell

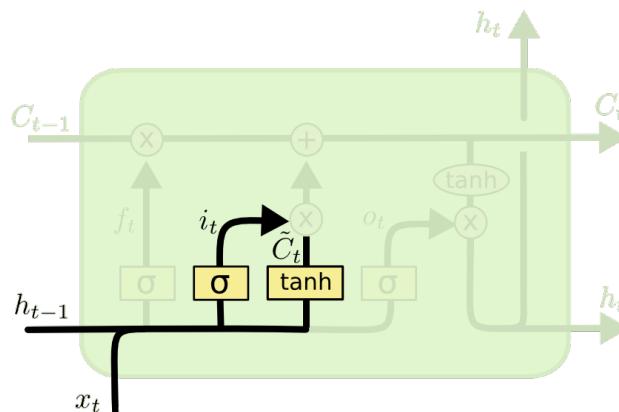
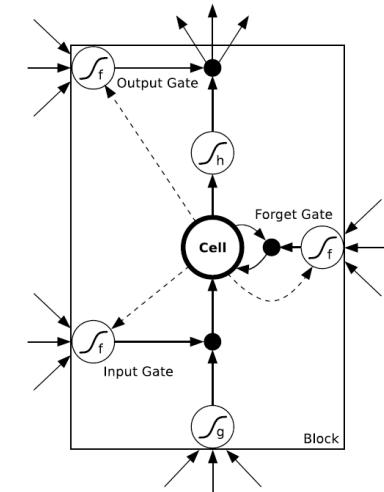
i: Input gate, whether to write to cell

g: Gate gate (?), How much to write to cell

o: Output gate, How much to reveal cell

Input Gate & New candidate value

- Input Gate는 “어떤 정보를 업데이트할지”를 결정한다.
- tanh layer는 state에 더해질 수 있는 새로운 후보값인(new candidate values) \tilde{C}_t 를 만든다.



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

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

f: Forget gate, Whether to erase cell

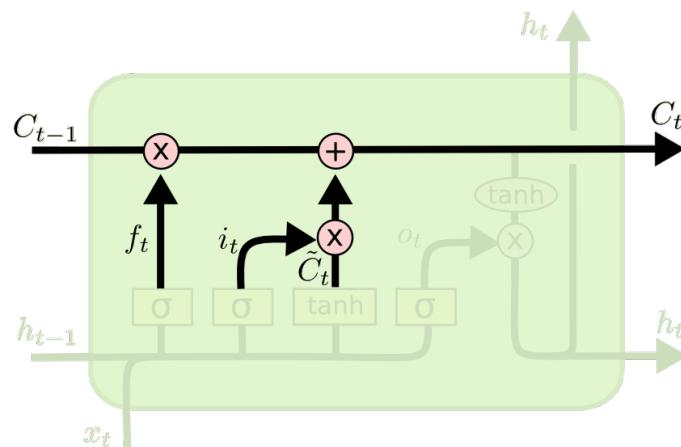
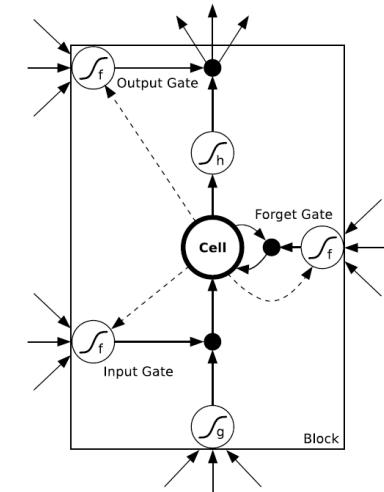
i: Input gate, whether to write to cell

g: Gate gate (?), How much to write to cell

o: Output gate, How much to reveal cell

Cell Value Update

- ▣ 이전 state값과 forget gate의 출력값을 곱하고, Input gate의 출력값과 새로운 후보값을 곱해서 더해서 state값을 업데이트한다.



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

f: Forget gate, Whether to erase cell

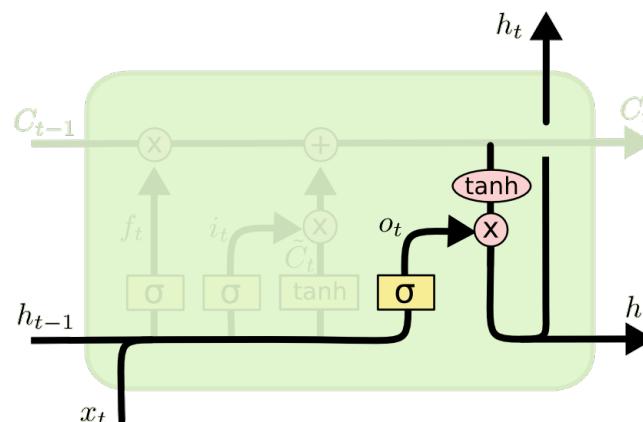
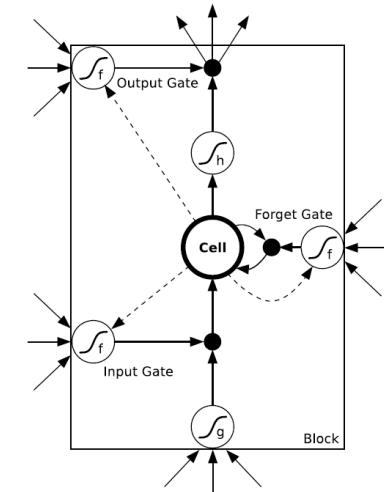
i: Input gate, whether to write to cell

g: Gate gate (?), How much to write to cell

o: Output gate, How much to reveal cell

Output Gate

- Cell state에 tanh을 써우고, sigmoid Layer의 값과 곱해서 Output Gate의 값을 구한다. 이는 값을 얼마나 출력할 것인지를 결정한다.



$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

f: Forget gate, Whether to erase cell

i: Input gate, whether to write to cell

g: Gate gate (?), How much to write to cell

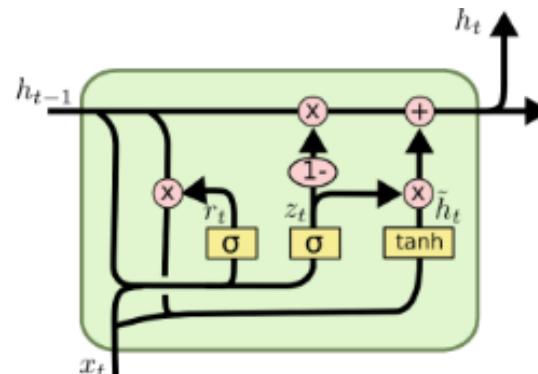
o: Output gate, How much to reveal cell

Gate Recurrent Unit(GRU)

- GRU는 리셋 게이트(Reset Gate) r 과 업데이트 게이트(Update Gate) z 로, 총 두 가지 게이트가 있습니다. 게이트 이름에서 알 수 있듯이, 리셋 게이트는 새로운 입력을 이전 메모리와 어떻게 합칠지를 정해주고, 업데이트 게이트는 이전 메모리를 얼마나 기억할지를 정해줍니다.
- GRU는 게이팅 메커니즘을 통해 긴 시퀀스를 잘 기억하도록 해준다는 점에서는 LSTM과 기본 아이디어가 같지만, 몇 가지 차이점이 있습니다.

- GRU는 게이트가 2개(Update Gate, Reset Gate)이고, LSTM은 3개(Input Gate, Forget Gate, Output Gate)입니다.
- GRU는 내부 메모리 값 (c_t)이 외부에서 보게되는 hidden state 값과 다르지 않습니다. LSTM에 있는 출력 게이트(Output Gate)가 없기 때문입니다.
- 입력 게이트(Input Gate)와 까먹음 게이트(Forget Gate)가 업데이트 게이트(Update Gate) z 로 합쳐졌고, 리셋 게이트(Reset Gate) r 은 이전 hidden state 값에 바로 적용됩니다. 따라서, LSTM의 까먹음 게이트(Forget Gate)의 역할이 r 과 z 둘 다에 나눠졌다고 생각할 수 있습니다.
- 출력값을 계산할 때 추가적인 비선형 함수를 적용하지 않습니다.

Slide credit: Christopher Olah
(<https://goo.gl/JNALXF>)



$$z_t = \sigma (W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma (W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh (W \cdot [r_t * h_{t-1}, x_t])$$

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

RNNs = GRU의 특수케이스

- 직관적으로 보면, 기본 RNN 모델은 GRU의 특수 케이스로 생각할 수도 있습니다.
- 리셋 게이트(Reset Gate) 값을 전부 1로 정해주고 업데이트 게이트(Update Gate)를 전부 0으로 정한다면, 기본 RNN 구조가 될 것입니다.

$$z = \sigma(x_t U^z + s_{t-1} W^z)$$

$$r = \sigma(x_t U^r + s_{t-1} W^r)$$

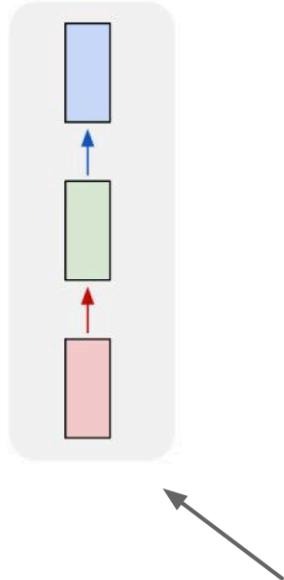
$$h = \tanh(x_t U^h + (s_{t-1} \circ r) W^h)$$

$$s_t = (1 - z) \circ h + z \circ s_{t-1}$$

RNNs 응용의 다양한 형태 – one to one

“Vanilla” Neural Network

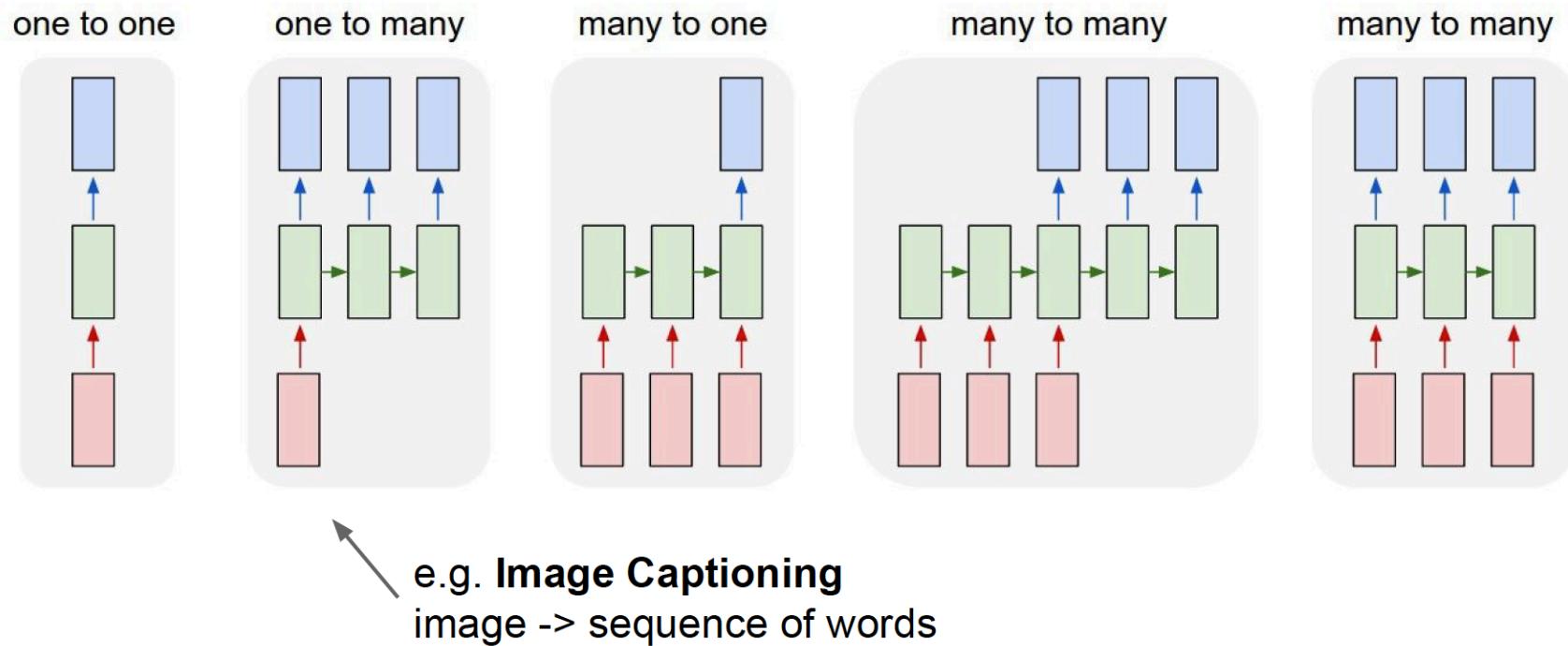
one to one



Vanilla Neural Networks

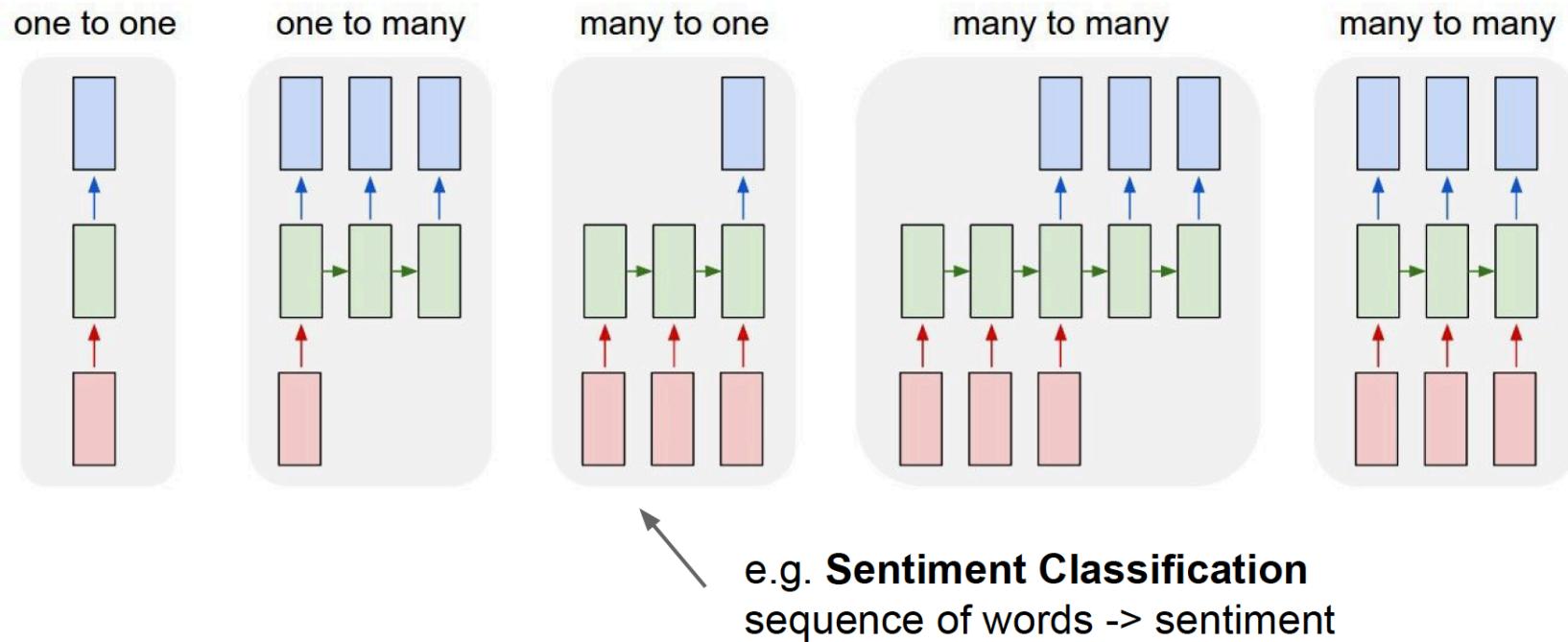
RNNs 응용의 다양한 형태 – one to many

Recurrent Neural Networks: Process Sequences



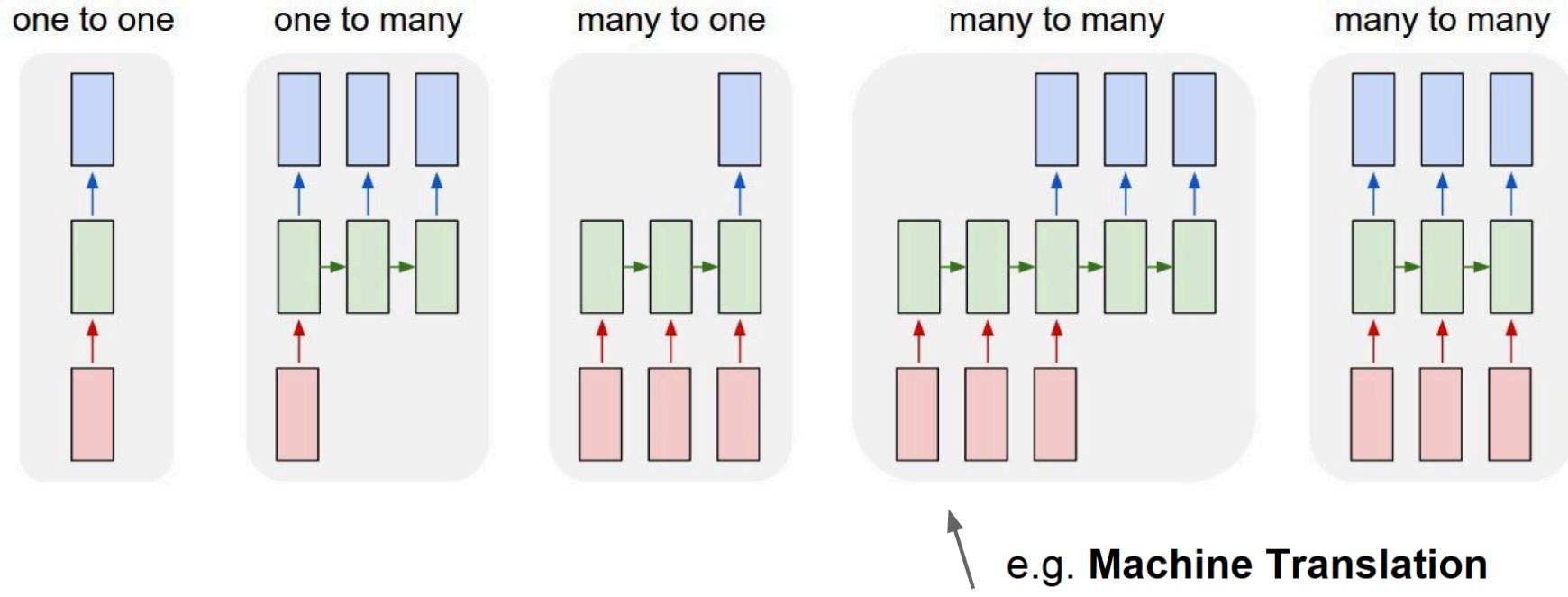
RNNs 응용의 다양한 형태 - many to one

Recurrent Neural Networks: Process Sequences



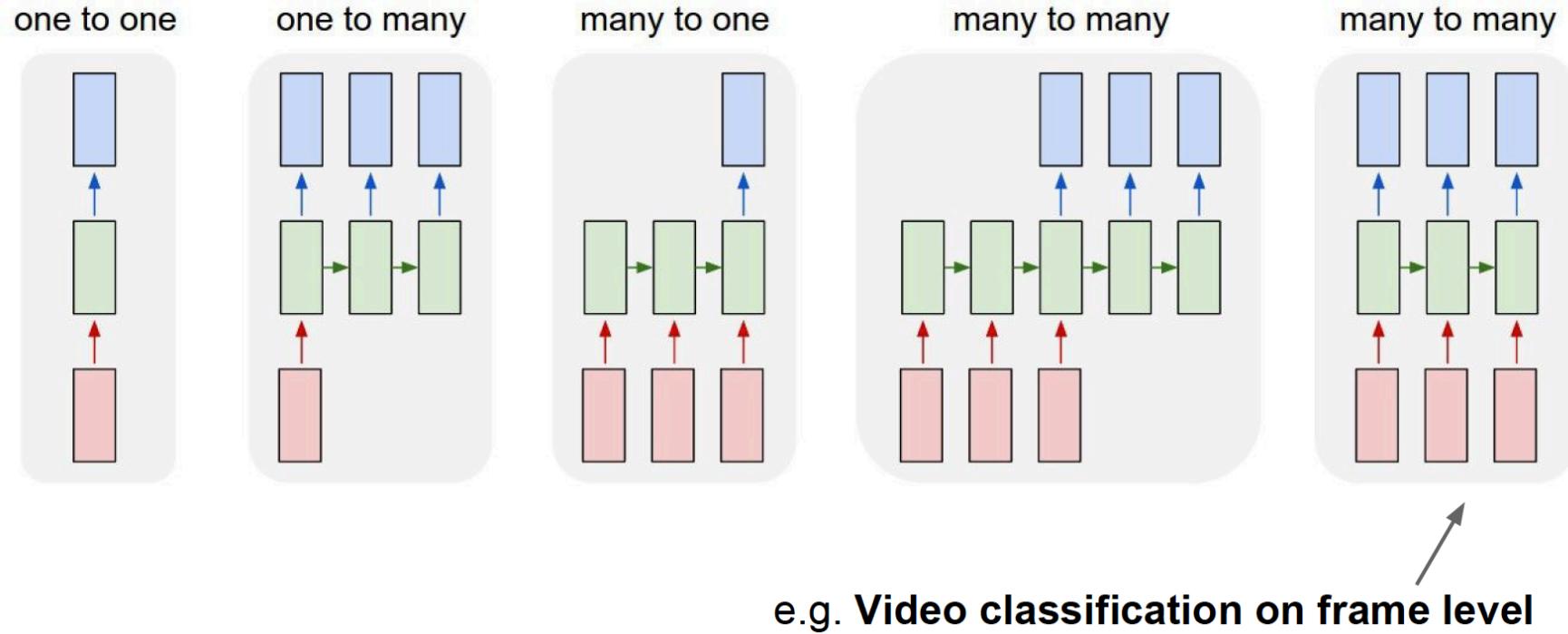
RNNs 응용의 다양한 형태 - many to many

Recurrent Neural Networks: Process Sequences



RNNs 응용의 다양한 형태 - many to many

Recurrent Neural Networks: Process Sequences



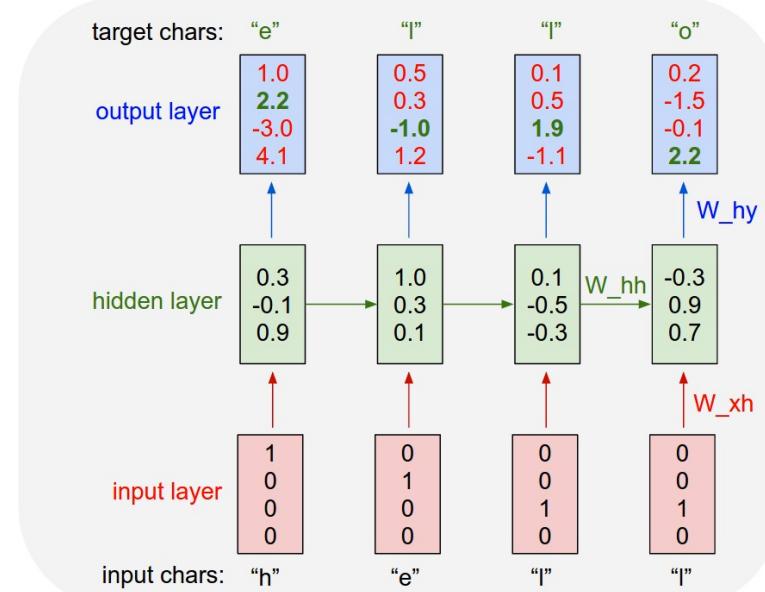
RNNs을 이용한 Character-level Language Modeling

- Preprocess : 텍스트 데이터셋을 글자(Character) 단위로 쪼개고 글자들의 사전(Dictionary)을 만들고 글자를 모두 할당된 숫자로 맵핑한다.

```
{': 0, '라': 1, '\n': 2, '다': 3, ':': 4, '.': 5, '에': 6, '이': 7, '르': 8, '디': 9, '미': 10, '스': 11, '그': 12, '블': 13, '트': 14, '공': 15, '자': 16, '(': 17, ')': 18, '가': 19, '는': 20, '?': 21, '고': 22, '아': 23, '하': 26, 'ㅂ': 27, '리': 28, '나': 29, '서': 30, '개': 31, '도': 32, '니': 33, '이': 34, '자': 35, '조': 36, '은': 37, '로': 38, '구': 39, '포': 40, '맞': 41, '거': 42, '기': 43, '시': 44, '한': 45, '까': 46, '외': 47, '무': 48, '아': 49, '내': 50, '민': 51, '음': 52, '오': 53, '보': 54, '소': 55, '으': 56, '모': 57, '사': 58, '복': 59, '의': 60, '이': 61, 'ჯ': 62, '길': 63, '하': 64, '워': 65, '데': 66, '린': 67, '여': 68, '별': 69, '두': 70, '데': 71, '레': 72, '데': 73, '안': 74, '지': 75, '우': 76, '나': 77, '생': 78, '일': 79, '대': 80, '주': 81, '각': 82, '여': 84, '링': 85, '부': 86, '있': 87, '체': 88, '수': 89, '와': 90, '간': 91, '것': 92, '와': 93, '건': 94, '목': 95, '파': 96, '길': 97, '신': 98, '린': 99, '남': 100, '진': 101, '위': 102, '인': 103, '점': 104, '본': 105, '분': 106, '마': 107, '당': 108, '정': 109, '할': 110, '립': 111, '필': 112, '네': 113, '녀': 114, '년': 115, '트': 116, '음': 117, '년': 118, '았': 119, '음': 120, '직': 121, '단': 122, '선': 123, '알': 124, '못': 125, '뭐': 126, '긴': 127, '憎': 128, '손': 129, '화': 130, '하': 131, '이': 132, '화': 133, '바': 134, '화': 135, '들': 136, '겠': 137, '군': 138, '발': 139, '예': 140, '날': 141, '부': 142, '큰': 143, '손': 144, '왕': 145, '속': 146, '의': 147, '동': 148, '태': 149, '화': 150, '말': 151, '침': 153, '몇': 154, '뭐': 155, '워': 156, '금': 157, '영': 158, '우': 159, '치': 160, '과': 161, '미': 162, '개': 163, '친': 164, '살': 165, '웃': 166, '웃': 167, '한': 168, '위': 169, '웃': 170, '또': 171, '던': 172, '한': 173, '정': 174, '번': 175, '상': 176, '프': 177, '세': 178, '늘': 179, '죽': 180, '화': 181, '은': 182, '은': 183, '영': 184, '장': 185, '문': 186, '문': 187, '명': 188, '체': 189, '분': 190, '분': 191, '자': 192, '키': 193, '진': 194, '월': 195, '월': 196, '화': 197, '길': 198, '느': 199, '심': 200, '찰': 201, '듯': 202, '화': 203, '찰': 204, '화': 205, '화': 206, '화': 207, '화': 208, '쓰': 209, '않': 210, '막': 211, '민': 212, '하': 213, '는': 214, '마': 215, '벗': 216, '풀': 217, '님': 218, '죽': 219, '죽': 220, '화': 221, '화': 222, '화': 223, '화': 224, '물': 225, '차': 226, '병': 227, '갈': 228, '뿐': 229, '설': 230, '크': 231, '복': 232, '화': 233, '화': 234, '운': 235, '화': 236, '화': 237, '화': 238, '근': 239, '증': 240, '피': 241, '화': 242, '빛': 243, '월': 244, '술': 245, '빛': 247, '란': 248, '방': 249, '방': 250, '방': 251, '방': 252, '방': 253, '방': 254, '방': 255, '방': 256, '방': 267, '물': 268, '파': 269, '기': 270, '화': 271, '화': 272, '화': 273, '화': 274, '화': 275, '행': 276, '적': 277, '날': 278, '맞': 279, '화': 280, '화': 281, '화': 282, '화': 283, '화': 284, '화': 285, '화': 286, '화': 287, '화': 288, '화': 289, '화': 290, '화': 291, '유': 292, '유': 293, '화': 294, '화': 295, '화': 296, '화': 297, '화': 298, '화': 299, '화': 300, '화': 301, '화': 302, '화': 303, '화': 304, '화': 305, '화': 306, '화': 307, '화': 308, '화': 309, '화': 310, '화': 311, '화': 312, '화': 313, '화': 314, '화': 315, '화': 316, '화': 317, '화': 318, '화': 319, '화': 320, '화': 321, '화': 322, '화': 323, '화': 324, '화': 325, '화': 326, '화': 327, '화': 328, '화': 329, '화': 330, '화': 331, '화': 332, '화': 333, '화': 334, '화': 335, '화': 336, '화': 337, '화': 338, '화': 339, '화': 340, '화': 341, '화': 342, '화': 343, '화': 344, '화': 345, '화': 346, '화': 347, '화': 348, '화': 349, '화': 350, '화': 351, '화': 352, '화': 353, '화': 354, '화': 355, '화': 356, '화': 357, '화': 358, '화': 359, '화': 360, '화': 361, '화': 362, '화': 363, '화': 364, '화': 365, '화': 366, '화': 367, '화': 368, '화': 369, '화': 370, '화': 371, '화': 372, '화': 373, '화': 374, '화': 375, '화': 376, '화': 377, '화': 378, '화': 379, '화': 380, '화': 381, '화': 382, '화': 383, '화': 384, '화': 385, '화': 386, '화': 387, '화': 388, '화': 389, '화': 390, '화': 391, '화': 392, '화': 393, '화': 394, '화': 395, '화': 396, '화': 397, '화': 398, '화': 399, '화': 400, '화': 401, '화': 402, '화': 403, '화': 404, '화': 405, '화': 406, '화': 407, '화': 408, '화': 409, '화': 410, '화': 411, '화': 412, '화': 413, '화': 414, '화': 415, '화': 416, '화': 417, '화': 418, '화': 419, '화': 420, '화': 421, '화': 422, '화': 423, '화': 424, '화': 425, '화': 426, '화': 427, '화': 428, '화': 429, '화': 430, '화': 431, '화': 432, '화': 433, '화': 434, '화': 435, '화': 436, '화': 437, '화': 438, '화': 439, '화': 440, '화': 441, '화': 442, '화': 443, '화': 444, '화': 445, '화': 446, '화': 447, '화': 448, '화': 449, '화': 450, '화': 451, '화': 452, '화': 453, '화': 454, '화': 455, '화': 456, '화': 457, '화': 458, '화': 459, '화': 460, '화': 461, '화': 462, '화': 463, '화': 464, '화': 465, '화': 466, '화': 467, '화': 468, '화': 469, '화': 470, '화': 471, '화': 472, '화': 473, '화': 474, '화': 475, '화': 476, '화': 477, '화': 478, '화': 479, '화': 480, '화': 481, '화': 482, '화': 483, '화': 484, '화': 485, '화': 486, '화': 487, '화': 488, '화': 489, '화': 490, '화': 491, '화': 492, '화': 493, '화': 494, '화': 495, '화': 496, '화': 497, '화': 498, '화': 499, '화': 500, '화': 501, '화': 502, '화': 503, '화': 504, '화': 505, '화': 506, '화': 507, '화': 508, '화': 509, '화': 510, '화': 511, '화': 512, '화': 513, '화': 514, '화': 515, '화': 516, '화': 517, '화': 518, '화': 519, '화': 520, '화': 521, '화': 522, '화': 523, '화': 524, '화': 525, '화': 526, '화': 527, '화': 528, '화': 529, '화': 530, '화': 531, '화': 532, '화': 533, '화': 534, '화': 535, '화': 536, '화': 537, '화': 538, '화': 539, '화': 540, '화': 541, '화': 542, '화': 543, '화': 544, '화': 545, '화': 546, '화': 547, '화': 548, '화': 549, '화': 550, '화': 551, '화': 552, '화': 553, '화': 554, '화': 555, '화': 556, '화': 557, '화': 558, '화': 559, '화': 560, '화': 561, '화': 562, '화': 563, '화': 564, '화': 565, '화': 566, '화': 567, '화': 568, '화': 569, '화': 570, '화': 571, '화': 572, '화': 573, '화': 574, '화': 575, '화': 576, '화': 577, '화': 578, '화': 579, '화': 580, '화': 581, '화': 582, '화': 583, '화': 584, '화': 585, '화': 586, '화': 587, '화': 588, '화': 589, '화': 590, '화': 591, '화': 592, '화': 593, '화': 594, '화': 595, '화': 596, '화': 597, '화': 598, '화': 599, '화': 600, '화': 601, '화': 602, '화': 603, '화': 604, '화': 605, '화': 606, '화': 607, '화': 608, '화': 609, '화': 610, '화': 611, '화': 612, '화': 613, '화': 614, '화': 615, '화': 616, '화': 617, '화': 618, '화': 619, '화': 620, '화': 621, '화': 622, '화': 623, '화': 624, '화': 625, '화': 626, '화': 627, '화': 628, '화': 629, '화': 630, '화': 631, '화': 632, '화': 633, '화': 634, '화': 635, '화': 636, '화': 637, '화': 638, '화': 639, '화': 640, '화': 641, '화': 642, '화': 643, '화': 644, '화': 645, '화': 646, '화': 647, '화': 648, '화': 649, '화': 650, '화': 651, '화': 652, '화': 653, '화': 654, '화': 655, '화': 656, '화': 657, '화': 658, '화': 659, '화': 660, '화': 661, '화': 662, '화': 663, '화': 664, '화': 665, '화': 666, '화': 667, '화': 668, '화': 669, '화': 670, '화': 671, '화': 672, '화': 673, '화': 674, '화': 675, '화': 676, '화': 677, '화': 678, '화': 679, '화': 680, '화': 681, '화': 682, '화': 683, '화': 684, '화': 685, '화': 686, '화': 687, '화': 688, '화': 689, '화': 690, '화': 691, '화': 692, '화': 693, '화': 694, '화': 695, '화': 696, '화': 697, '화': 698, '화': 699, '화': 700, '화': 701, '화': 702, '화': 703, '화': 704, '화': 705, '화': 706, '화': 707, '화': 708, '화': 709, '화': 710, '화': 711, '화': 712, '화': 713, '화': 714, '화': 715, '화': 716, '화': 717, '화': 718, '화': 719, '화': 720, '화': 721, '화': 722, '화': 723, '화': 724, '화': 725, '화': 726, '화': 727, '화': 728, '화': 729, '화': 730, '화': 731, '화': 732, '화': 733, '화': 734, '화': 735, '화': 736, '화': 737, '화': 738, '화': 739, '화': 740, '화': 741, '화': 742, '화': 743, '화': 744, '화': 745, '화': 746, '화': 747, '화': 748, '화': 749, '화': 750, '화': 751, '화': 752, '화': 753, '화': 754, '화': 755, '화': 756, '화': 757, '화': 758, '화': 759, '화': 760, '화': 761, '화': 762, '화': 763, '화': 764, '화': 765, '화': 766, '화': 767, '화': 768, '화': 769, '화': 770, '화': 771, '화': 772, '화': 773, '화': 774, '화': 775, '화': 776, '화': 777, '화': 778, '화': 779, '화': 780, '화': 781, '화': 782, '화': 783, '화': 784, '화': 785, '화': 786, '화': 787, '화': 788, '화': 789, '화': 790, '화': 791, '화': 792, '화': 793, '화': 794, '화': 795, '화': 796, '화': 797, '화': 798, '화': 799, '화': 800, '화': 801, '화': 802, '화': 803, '화': 804, '화': 805, '화': 806, '화': 807, '화': 808, '화': 809, '화': 810, '화': 811, '화': 812, '화': 813, '화': 814, '화': 815, '화': 816, '화': 817, '화': 818, '화': 819, '화': 820, '화': 821, '화': 822, '화': 823, '화': 824, '화': 825, '화': 826, '화': 827, '화': 828, '화': 829, '화': 830, '화': 831, '화': 832, '화': 833, '화': 834, '화': 835, '화': 836, '화': 837, '화': 838, '화': 839, '화': 840, '화': 841, '화': 842, '화': 843, '화': 844, '화': 845, '화': 846, '화': 847, '화': 848, '화': 849, '화': 850, '화': 851, '화': 852, '화': 853, '화': 854, '화': 855, '화': 856, '화': 857, '화': 858, '화': 859, '화': 860, '화': 861, '화': 862, '화': 863, '화': 864, '화': 865, '화': 866, '화': 867, '화': 868, '화': 869, '화': 870, '화': 871, '화': 872, '화': 873, '화': 874, '화': 875, '화': 876, '화': 877, '화': 878, '화': 879, '화': 880, '화': 881, '화': 882, '화': 883, '화': 884, '화': 885, '화': 886, '화': 887, '화': 888, '화': 889, '화': 890, '화': 891, '화': 892, '화': 893, '화': 894, '화': 895, '화': 896, '화': 897, '화': 898, '화': 899, '화': 900, '화': 901, '화': 902, '화': 903, '화': 904, '화': 905, '화': 906, '화': 907, '화': 908, '화': 909, '화': 910, '화': 911, '화': 912, '화': 913, '화': 914, '화': 915, '화': 916, '화': 917, '화': 918, '화': 919, '화': 920, '화': 921, '화': 922, '화': 923, '화': 924, '화': 925, '화': 926, '화': 927, '화': 928, '화': 929, '화': 930, '화': 931, '화': 932}
```

RNNs을 이용한 Character-level Language Modeling

- **Training** : 인풋 데이터(input data-x-)에서 글자(Character)를 하나 뒤로 미는 타겟 데이터(target data-y-)로 RNNs을 학습한다.
- **Input Data** : 전체 문장 중 일정 길이의 글자들의 배열 (e.g. hell(전체 문장 중 일정 길이의 글자들의 배열))
- **Target Data** : 전체 데이터중 Input Data를 한글자 뒤로 미는 배열 (e.g. ello(전체 데이터중 Input Data를 한글자 뒤로 미는 배열))



RNNs을 이용한 Character-level Language Modeling

- Sampling : 학습된 결과를 토대로 샘플링을 통해 임의의 글자를 생성한다.

에스트라공: (무대 옆에서 몸을 비꼰다.) 앞으로!

블라디미르: 나는 우릴 고맙다 (헐떡내 다친다) 이것 사실이오. (럭키에게) 하지만 생 각을 할까? 무슨 소리를 터터봐

블라디미르: 회연하는 성을 거일도 좋니?

에스트라공: 벌써 미칠대를 럭키의 주위를 지면줘요!

블라디미르: 빨리빨리! 이놈 좀 일으켜 세워요!

그는 발을 멈추고 블라디미르 모자를 놓고 럭키의 모자를 고우지 않을 거들어 줘!

블라디미르가 끈과 소년은 물러서며) 이번 한테전 것도 없었냐?

에스트라공: 안 그래요!

블라디미르: 조용히!

(침묵)

에스트라공: 왜 왜 누구?

블라디미르: 고도를 기다려야지.

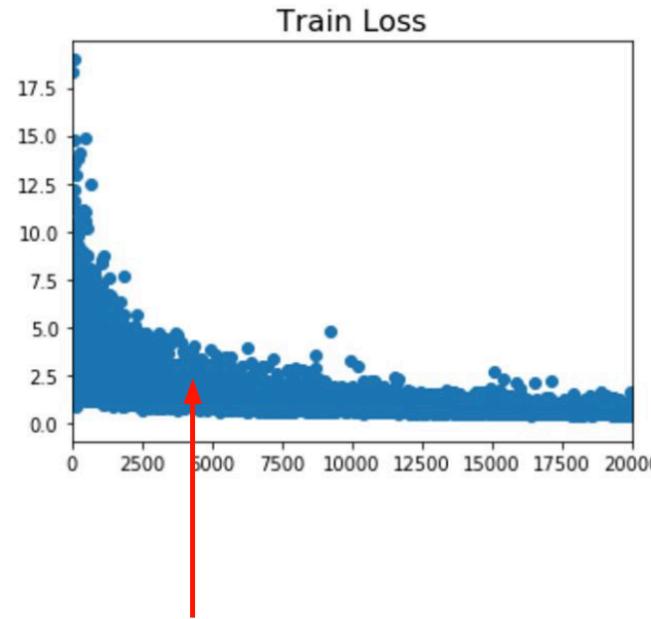
에스트라공: 참 그렇구 말야?

블라디미르: 고도가 싫다니까. 벌써 시간이 흐르는 게 있으면

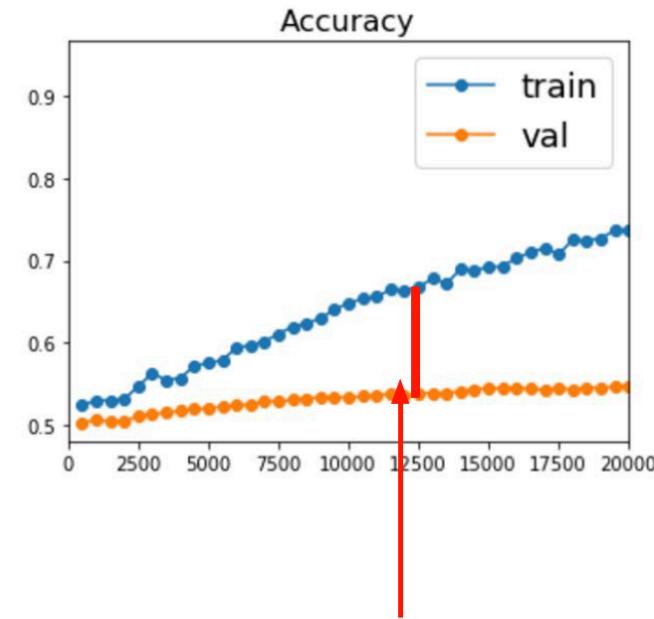
그림 4 – 사무엘 베케트의 희곡 “고도를 기다리며”를 데이터셋으로 학습을 진행한 이후에 샘플링한 결과

Training Error vs Test Error

Beyond Training Error



Better optimization algorithms help reduce training loss



But we really care about error on new data - how to reduce the gap?

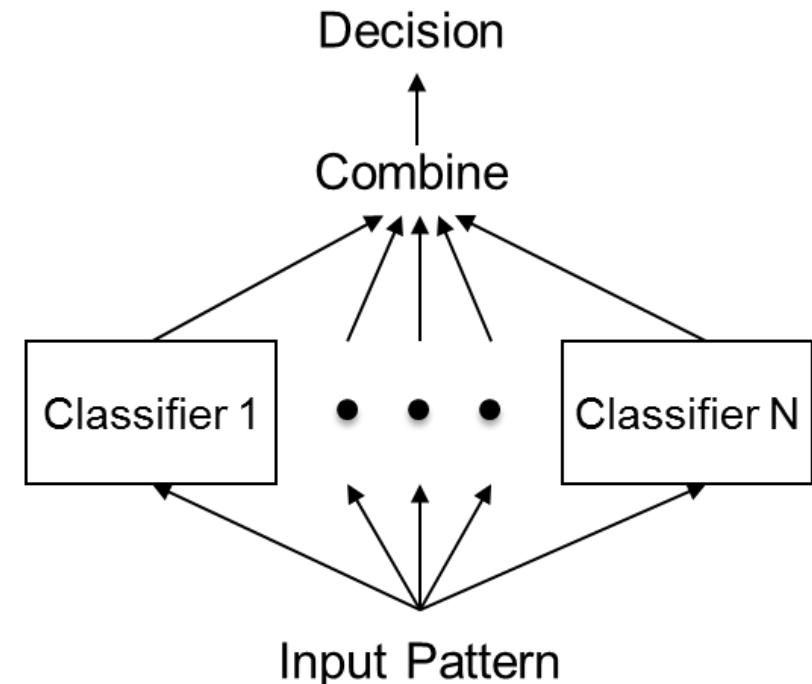
Ensemble Learning

- **Ensemble Learning(양상별 학습)** : 여러개의 분류기를 학습시킨 다음 Test시에 이들의 평균 혹은 다수결 투표를 이용한다.

Model Ensembles

1. Train multiple independent models
2. At test time average their results

Enjoy 2% extra performance

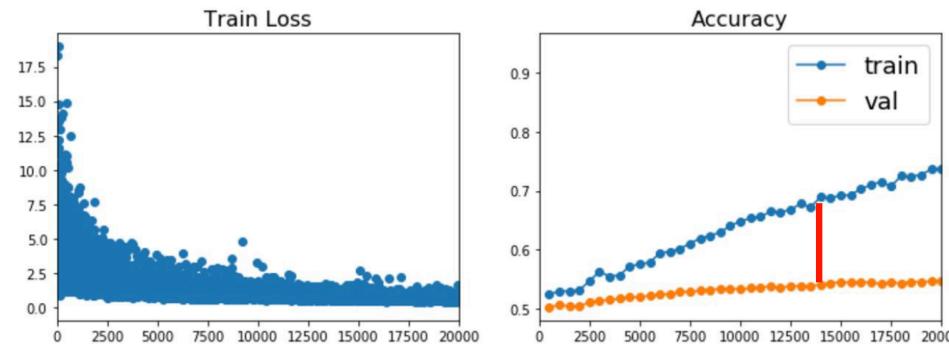


Regularization(일반화)

- **Regularization(일반화)** : 모델의 복잡도를 낮춰서 Test 데이터에 대한 정확도를 높인다.(=Overfitting을 방지한다.)

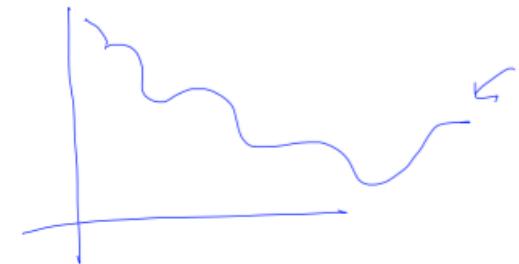
How to improve single-model performance?

Regularization



Regularization

- Let's not have too big numbers in the weight



Regularization 기법 1 - Regularization Term in Cost function

- Regularization Term을 Loss function에 추가해서 Weight가 너무 커지는 것을 방지한다.

Regularization: Add term to loss

$$L = \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay)

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

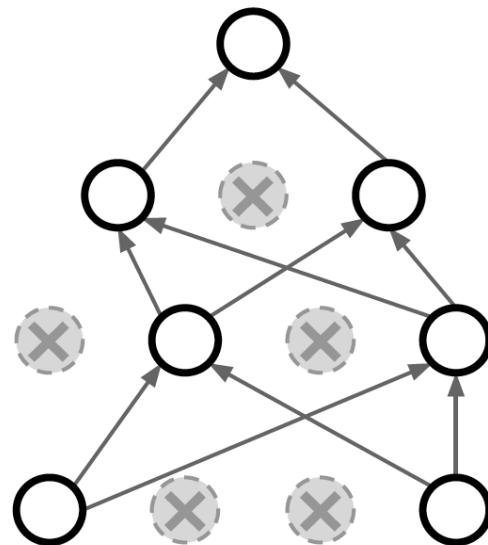
Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Regularization 기법 2 - Dropout

- Dropout = 일종의 Ensemble Learning 기법으로 바라 볼 수 있다.

Regularization: Dropout

How can this possibly be a good idea?



Another interpretation:

Dropout is training a large **ensemble** of models (that share parameters).

Each binary mask is one model

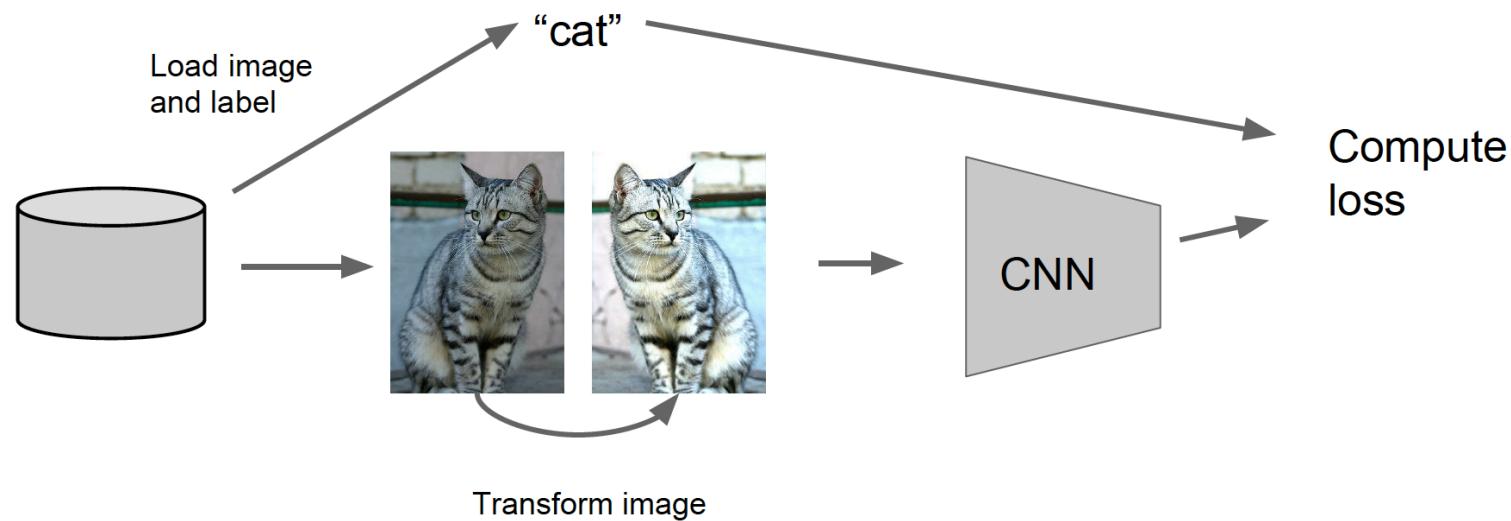
An FC layer with 4096 units has $2^{4096} \sim 10^{1233}$ possible masks!

Only $\sim 10^{82}$ atoms in the universe...

Regularization 기법 3 - Data Augmentation

- **Data Augmentation(데이터 증대)** : 트레이닝 데이터의 데이터의 다양성과 양을 늘려서 Overfitting을 방지한다.

Regularization: Data Augmentation



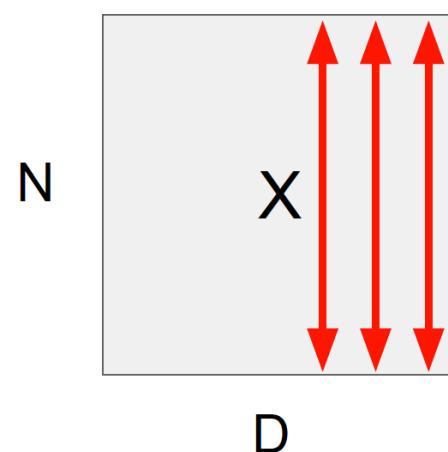
Regularization 기법 4 – Batch Normalization

- Batch Normalization(배치 정규화) : 배치 단위로 정규화(Normalization)을 해줌으로써 학습을 안정화시키는 기법

Batch Normalization

[Ioffe and Szegedy, 2015]

“you want unit gaussian activations?
just make them so.”



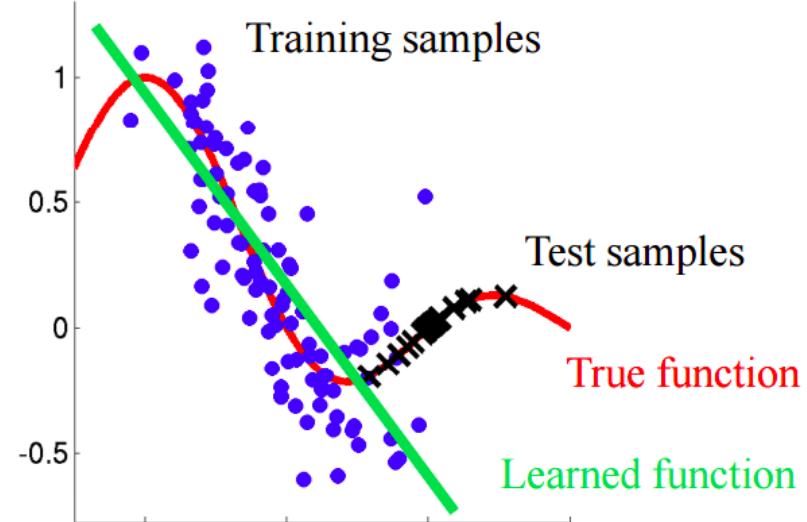
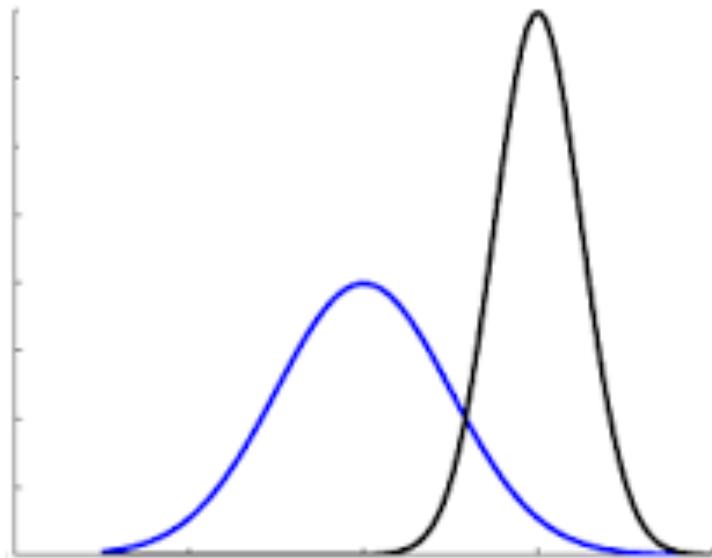
1. compute the empirical mean and variance independently for each dimension.

2. Normalize

$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

Vanishing Gradient Problem이 발생하는 이유 – Covariate Shift

- **Vanishing Gradient Problem이 발생하는 이유** : Internal Covariate Shift라는 Network의 각 층이나 Activation마다 input의 distribution이 달라지는 현상 때문

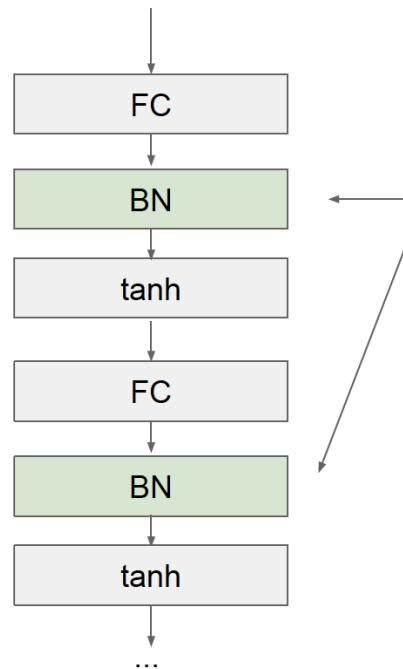


Batch Normalization 적용

- Activation Function 이전에 적용해준다.

Batch Normalization

[Ioffe and Szegedy, 2015]



Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

Batch Normalization 적용

- 모델의 표현력을 높여주기 위해 추가적으로 $\gamma^{(k)}, \beta^{(k)}$ 를 파라미터로 학습한다. (학습과정에서 Backpropagation을 이용해서 자동으로 최적의 값을 찾는다.)

Batch Normalization

[Ioffe and Szegedy, 2015]

Normalize:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\text{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \text{E}[x^{(k)}]$$

to recover the identity mapping.

Batch Normalization의 알고리즘과 장점

Batch Normalization의 장점

1. 학습 과정이 안정화된다.
2. Learning Rate를 크게 잡을 수 있어서 학습 속도를 높일 수 있다.

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

Test time에서 Batch Normalization

- Test Data에 대한 Inference를 수행할 때는 Batch Normalization을 조금 다르게 사용한다.
- Batch들의 Empirical Mean, Variance을 사용하는 대신에 지금까지 본 전체 데이터를 다 고려한다는 느낌으로, training 할 때 현재까지 본 input들의 이동평균 (moving average)을 계산하여 저장해놓은 뒤 이 값으로 normalize를 한다.

Batch Normalization

[Ioffe and Szegedy, 2015]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

Note: at test time BatchNorm layer functions differently:

The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.

(e.g. can be estimated during training with running averages)

TensorFlow에서 Gradient Clipping을 적용하는 법

Slide credit: CS 20SI
(<https://goo.gl/Ez8wRq>)

Exploding Gradients

Clip gradients with `tf.clip_by_global_norm`

```
gradients = tf.gradients(cost, tf.trainable_variables())
# take gradients of cost w.r.t. all trainable variables

clipped_gradients, _ = tf.clip_by_global_norm(gradients, max_grad_norm)
# clip the gradients by a pre-defined max norm

optimizer = tf.train.AdamOptimizer(learning_rate)
train_op = optimizer.apply_gradients(zip(gradients, trainables))
# add the clipped gradients to the optimizer
```

TensorFlow에서 Embedding Lookup을 적용하는 방법

Slide credit: CS 20SI
(<https://goo.gl/Ez8wRq>)

Embedding Lookup

$$\begin{bmatrix} 0 & 0 & 0 & \boxed{1} & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ \boxed{10} & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = \begin{bmatrix} 10 & 12 & 19 \end{bmatrix}$$

```
tf.nn.embedding_lookup(params, ids, partition_strategy='mod', name=None,  
validate_indices=True, max_norm=None)
```

TensorFlow를 이용한 Character-level Language Modeling을 위한 RNNs 구현

- ▣ https://github.com/solaris33/dl_cv_tensorflow_10weeks/tree/master/week3/char-rnn-tensorflow

에스트라공: (무대 옆에서 몸을 비꼰다.) 앞으로!
블라디미르: 나는 우릴 고맙다. (월역내 다친다) 이것 사실이오. (릭키에게) 하지만 생
각을 할까? 무슨 소리를 티더봐
블라디미르: 화연하는 성을 거기도 좋니?
에스트라공: 벌써 미침대를 럭키의 주위를 지면줘요!
블라디미르: 빌리밸리! 이놈 좀 일으켜 세워요!
그는 발을 땅놓고 블라디미르 모자를 놓고 럭키의 모자를 고우지 않을 거들어 줘!
블라디미르가 그과 소년은 물려서며 이번 한테전 것도 없었나?
에스트라공: 안 그래요!
블라디미르: 조용히!
침묵)
에스트라공: 왜 왜 누구?
블라디미르: 고도를 기다려야지.
에스트라공: 참 그렇구 말야?
블라디미르: 고도가 싫다니까. 벌써 시간이 흐르는 게 있으면

```
103     ret = prime
104     char = prime[-1]
105     for n in range(num):
106         x = np.zeros((1, 1))
107         x[0, 0] = vocab[char]
108         feed = {self.input_data: x, self.initial_state: state}
109         [probs, state] = sess.run([self.probs, self.final_state], feed)
110         p = probs[0]
111
112         if sampling_type == 0:
113             sample = np.argmax(p)
114         elif sampling_type == 2:
115             if char == ' ':
116                 sample = weighted_pick(p)
117             else:
118                 sample = np.argmax(p)
119         else: # sampling_type == 1 default:
120             sample = weighted_pick(p)
121
122         pred = chars[sample]
123         ret += pred
124         char = pred
125
126     return ret
```

그림 4 – 사무엘 베케트의 희곡 “고도를 기다리며”를 데이터셋으로 학습을 진행한 이후에 샘플링한 결과

다양한 char-rnn 예제들

Last Time: Recurrent Networks

For $\bigoplus_{n=1,\dots,m} \mathcal{L}_{m_n} = 0$, hence we can find a closed subset \mathcal{H} in \mathcal{H} and any sets \mathcal{F} on X , U is a closed immersion of S , then $U \rightarrow T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$\mathcal{S} = \text{Spec}(R) = U \times_X U \times_X U$$

and the comparicoly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \rightarrow V$. Consider the maps M along the set of points Sch_{fppf} and $U \rightarrow U$ is the fibre category of S in U in Section ?? and the fact that any U affine, see Morphisms, Lemma ???. Hence we obtain a scheme S and any open subset $W \subset U$ in $\text{Sh}(G)$ such that $\text{Spec}(R') \rightarrow S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S . We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \rightarrow \mathcal{O}_{X',x''}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\text{GL}_{S'}(x'/S')$ and we win. \square

To prove study we see that $\mathcal{F}|_U$ is a covering of \mathcal{X}' , and \mathcal{T}_i is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and \mathcal{F}_p exists and let \mathcal{F}_i be a presheaf of \mathcal{O}_X -modules on \mathcal{C} as a \mathcal{F} -module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\tilde{\mathcal{M}}^* = \mathcal{I}^* \otimes_{\text{Spec}(k)} \mathcal{O}_{S,*} - i_X^{-1} \mathcal{F}$$

is a unique morphism of algebraic stacks. Note that

$$\text{Arrows} = (\text{Sch}/S)^{\text{opp}}_{fppf}, (\text{Sch}/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longrightarrow (U, \text{Spec}(A))$$

is an open subset of X . Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S .

Proof. See discussion of sheaves of sets. \square

The result for prove any open covering follows from the less of Example ???. It may replace S by $X_{\text{spaces},\text{étale}}$ which gives an open subspace of X and T equal to S_{Zar} , see Descent, Lemma ???. Namely, by Lemma ?? we see that R is geometrically regular over S .

PANDARUS:

Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:

Come, sir, I will make did behold your worship.

VIOLA:

I'll drink it.

```
static void do_command(struct seq_file *m, void *v)
{
    int column = 32 << (cmd[2] & 0x80);
    if (state)
        cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
    else
        seq = 1;
    for (i = 0; i < 16; i++) {
        if (k & (1 << i))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000fffffff8) & 0x0000000f) << 8;
        if (count == 0)
            sub(pid, ppc_md.kexec_handle, 0x20000000);
        pipe_set_bytes(i, 0);
    }
    /* Free our user pages pointer to place camera if all dash */
    subsystem_info = &of_changes[PAGE_SIZE];
    rek_controls(offset, idx, &offset);
    /* Now we want to deliberately put it to device */
    control_check_polarity(&context, val, 0);
    for (i = 0; i < COUNTER; i++)
        seq_puts(s, "policy ");
}
```

대수 기하학 논문 생성

셰익스피어 희곡 생성

리눅스 C 코드 생성

TensorFlow를 이용한 Penn Tree Bank(PTB) dataset Language Modeling을 위한 RNNs 구현

- ▣ https://github.com/solaris33/dl_cv_tensorflow_10weeks/tree/master/week3/ptb

```
no it was n't black monday
but while the new york stock exchange did n't fall apart friday as the dow jones industrial average plunged N points most of it
in the final hour it barely managed to stay this side of chaos
some circuit breakers installed after the october N crash failed their first test traders say unable to cool the selling panic
in both stocks and futures
the N stock specialist firms on the big board floor the buyers and sellers of last resort who were criticized after the N crash
once again could n't handle the selling pressure
big investment banks refused to step up to the plate to support the beleaguered floor traders by buying big blocks of stock
traders say
heavy selling of standard & poor 's 500-stock index futures in chicago <unk> beat stocks downward
seven big board stocks val amr bankamerica walt disney capital cities\abc philip morris and pacific telesis group stopped
trading and never resumed
the <unk> has already begun
the equity market was <unk>
```

```
269     def run_epoch(session, model, eval_op=None, verbose=False):
270         """Runs the model on the given data."""
271         start_time = time.time()
272         costs = 0.0
273         iters = 0
274         state = session.run(model.initial_state)
275
276         fetches = {
277             "cost": model.cost,
278             "final_state": model.final_state,
279         }
280         if eval_op is not None:
281             fetches["eval_op"] = eval_op
282
283         for step in range(model.input.epoch_size):
284             feed_dict = {}
285             for i, (c, h) in enumerate(model.initial_state):
286                 feed_dict[c] = state[i].c
287                 feed_dict[h] = state[i].h
288
289             vals = session.run(fetches, feed_dict)
290             cost = vals["cost"]
291             state = vals["final_state"]
292
293             costs += cost
294             iters += model.input.num_steps
295
296             if verbose and step % (model.input.epoch_size // 10) == 10:
297                 print("%.3f perplexity: %.3f speed: %.0f wps" %
298                     (step * 1.0 / model.input.epoch_size, np.exp(costs / iters),
299                      iters * model.input.batch_size / (time.time() - start_time)))
300
301     return np.exp(costs / iters)
```

Penn Tree Bank(PTB) Dataset

- The Penn Treebank (PTB) project selected 2,499 stories from a three year Wall Street Journal (WSJ) collection of 98,732 stories for syntactic annotation.

no it was n't black monday
but while the new york stock exchange did n't fall apart friday as the dow jones industrial average plunged N points most of it
in the final hour it barely managed to stay this side of chaos
some circuit breakers installed after the october N crash failed their first test traders say unable to cool the selling panic
in both stocks and futures
the N stock specialist firms on the big board floor the buyers and sellers of last resort who were criticized after the N crash
once again could n't handle the selling pressure
big investment banks refused to step up to the plate to support the beleaguered floor traders by buying big blocks of stock
traders say
heavy selling of standard & poor 's 500-stock index futures in chicago <unk> beat stocks downward
seven big board stocks ual amr bankamerica walt disney capital cities\abc philip morris and pacific telesis group stopped
trading and never resumed
the <unk> has already begun
the equity market was <unk>

Perplexity and Loss Function

- Language Modeling이 얼마나 잘되었는지 측정하기 위해서 Perplexity라는 measure[2]를 사용한다. Perplexity는 정보 이론에서 확률 모델(Probability Model)이 샘플들을 얼마나 잘 예측(predict)하는지를 측정하는 지표이다. 확률 모델이 샘플들을 잘 예측할수록 perplexity가 낮아진다. 단어별(per-word) perplexity는 아래와 같은 수식으로 정의된다.

$$e^{-\frac{1}{N} \sum_{i=1}^N \ln p_{target_i}} = e^{loss}$$

- 우리가 만든 확률 모델이 정확할수록 다음에 올 단어 target_i를 정확히 예측할 확률 p_{target_i} 이 커질 것이다. 그렇다면 결과적으로 perplexity 값은 낮아질 것이다. 따라서 우리는 minimize할 손실 함수(Loss Function)의 값을 아래와 같이 정의한다.

$$loss = -\frac{1}{N} \sum_{i=1}^N \ln p_{target_i}$$

기존의 Text 데이터 표현의 문제점

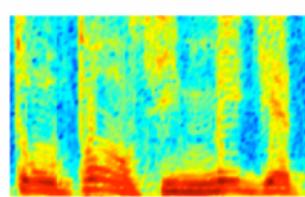
- 전통적인 방법에서는 하나의 단어를 discrete atomic symbol로 표현했다. 예를 들어,

cat = Id537

dog = Id143

- 이런식의 encoding 방법은 다음과 같은 문제점이 있다.

- 인코딩이 무작위적이고 데이터간의 관계를 보여주지 않는다. 따라서 모델이 “cats”라는 단어로부터 배운 특징을 “dogs”라는 단어를 처리할 때 적절하게 이용할 수 없다.
- 단어들을 discrete ID로 맵핑함으로써 데이터를 sparse하게 만든다. 따라서 통계적 모델을 성공적으로 트레이닝 하려면 많은 데이터가 있어야만한다.



Audio Spectrogram

DENSE

IMAGES

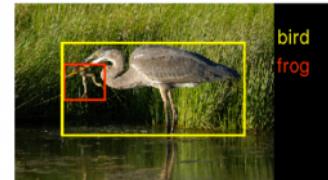


Image pixels

DENSE

TEXT

0 0 0 0.2 0 0.7 0 0 0

Word, context, or
document vectors

SPARSE

Word Embedding(Word2vec)

- vector representation을 통해 위의 두가지 문제점을 해결할 수 있다. (Vector Space Models(VSMs))

An embedding is a mapping from discrete objects, such as words, to vectors of real numbers. For example, a 300-dimensional embedding for English words could include:

```
blue: (0.01359, 0.00075997, 0.24608, ..., -0.2524, 1.0048, 0.06259)
blues: (0.01396, 0.11887, -0.48963, ..., 0.033483, -0.10007, 0.1158)
orange: (-0.24776, -0.12359, 0.20986, ..., 0.079717, 0.23865, -0.014213)
oranges: (-0.35609, 0.21854, 0.080944, ..., -0.35413, 0.38511, -0.070976)
```

- predictive methods는 small, dense **embedding vectors**로 표현 된 이웃 단어들을 이용해서 **직접적으로 단어를 예측한다.**
- word2vec은 raw text로부터 word embedding을 학습하는 계산 효율성이 좋은 predictive model이다. 이는 두가지 방법으로 구현될 수 있다.
(CBOW(Continuous Bag-Of-Words) model, Skip-Gram model)

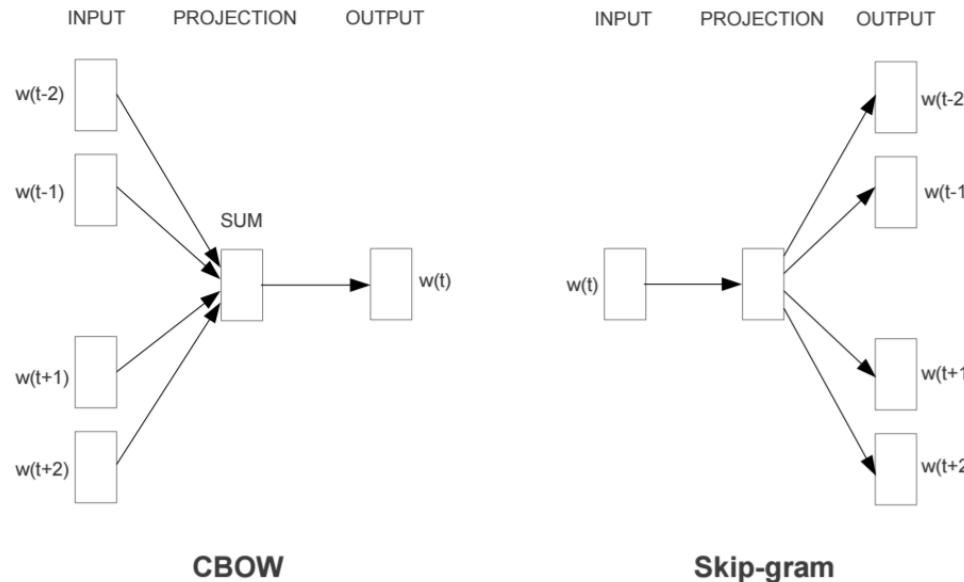
Appendix - CBOW(Continuous Bag-Of-Words) model vs Skip-Gram model

1. CBOW(Continuous Bag-Of-Words) model – 소스 컨텍스트에서 타겟 단어를 예측한다.

예를 들어, ‘the cat sits on the’라는 소스 컨텍스트로부터 ‘mat’이라는 타겟 단어를 예측한다. CBOW는 smaller 데이터셋에 적합하다.

2. Skip-Gram model – 타겟 단어로부터 소스 컨텍스트를 예측한다.

예를 들어, ‘mat’이라는 타겟 단어로부터 ‘the cat sits on the’라는 소스 컨텍스트를 예측한다. Skip-Gram model은 larger 데이터셋에 적합하다. 따라서, 앞으로 Skip-Gram model에 초점을 맞춰서 설명을 진행할 것이다.



Appendix - Noise-Contrastive 트레이닝을 통한 규모확장(Scaling Up)

- Neural probabilistic language 모델들은 전통적으로 maximum likelihood (ML)을 통해서 트레이닝을 진행하였다. 이전의 히스토리 h ("history")가 주어졌을 때 다음에 이어지는 목표 단어 w_t ("target")의 확률을 최대화하기 위해서 소프트맥스 함수를 사용한다.

$$P(w_t|h) = \text{softmax}(\text{score}(w_t, h)) = \frac{\exp\{\text{score}(w_t, h)\}}{\sum_{\text{Word } w' \text{ in Vocab}} \exp\{\text{score}(w', h)\}}$$

- 여기서 $\text{score}(w_t, h)$ 는 현재 콘텍스트 h 에서 단어 w_t 의 적합성을 나타낸다. (보통 내적을 이용한다.) 우리는 트레이닝 세트에서 log-likelihood를 최대화하는 방향으로 학습을 진행한다. 즉, 아래의 식을 최대화 한다.

$$J_{ML} = \log P(w_t|h) = \text{score}(w_t, h) - \log \left(\sum_{\text{Word } w' \text{ in Vocab}} \exp\{\text{score}(w', h)\} \right)$$

Appendix - Score = Dot products(내적)

$$P(w_t|h) = \text{softmax(score}(w_t, h)) = \frac{\exp\{\text{score}(w_t, h)\}}{\sum_{\text{Word } w' \text{ in Vocab}} \exp\{\text{score}(w', h)\}}$$

Dot products

Dot product

$$u^T v = u \cdot v = \sum_{i=1}^n u_i v_i$$

Bigger if u and v are more similar!

Iterate over $w=1 \dots W$: $u_w^T v$ means:

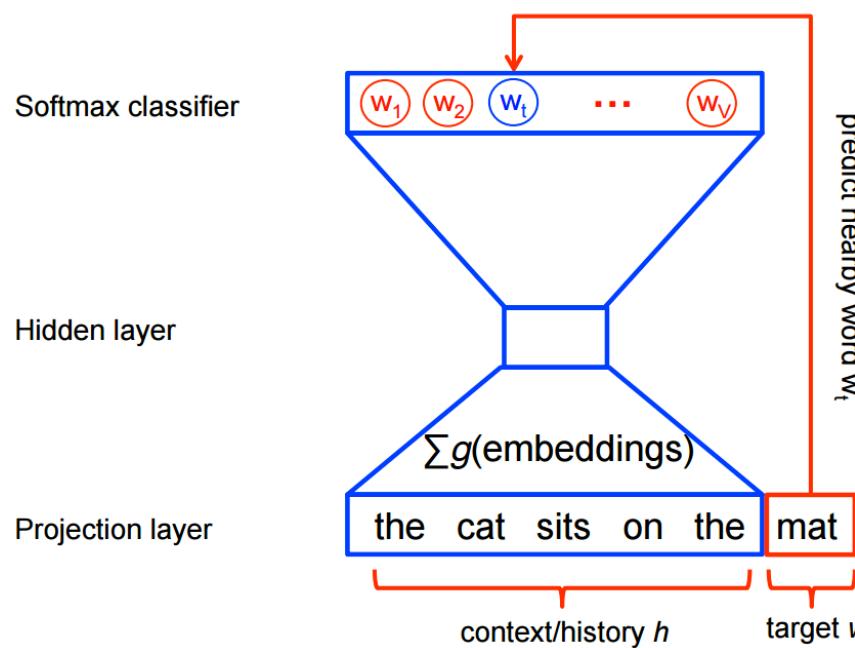
work out how similar each word is to v !

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^V \exp(u_w^T v_c)}$$

Noise-Contrastive 트레이닝을 통한 규모확장(Scaling Up)

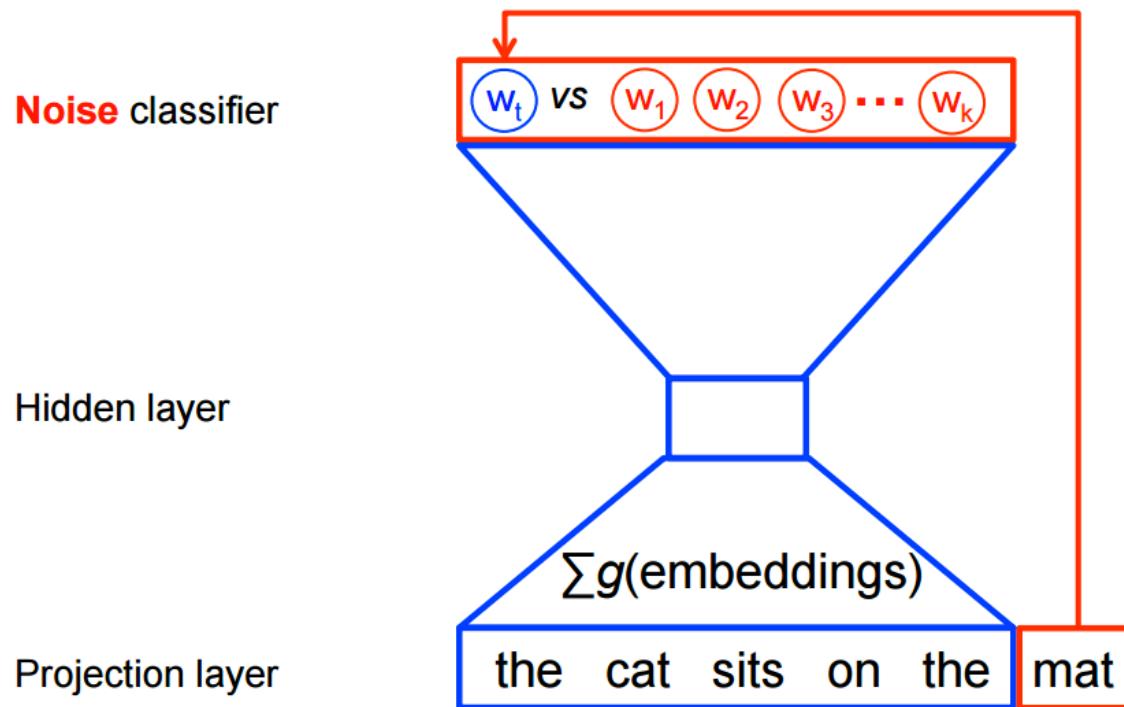
- 이런 모델은 언어 모델ing(language)을 위한 적당히 정규화된 확률 모델(normalized probabilistic model)을 산출해낸다. 하지만 이 모델은 계산 비용이 매우 크다. 왜냐하면 모든 트레이닝 스텝에서, 현재 콘텍스트 h 에서 어휘(Vocab)에 포함된 모든 단어 w' 에 대한 확률을 계산하고 정규화해야하기 때문이다.

$$J_{ML} = \log P(w_t|h) = \text{score}(w_t, h) - \log \left(\sum_{\text{Word } w' \text{ in Vocab}} \exp\{\text{score}(w', h)\} \right)$$



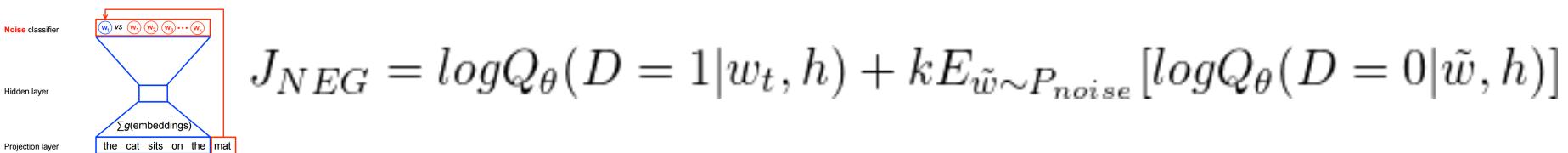
Appendix - Noise-Contrastive 트레이닝을 통한 규모확장(Scaling Up)

- 이에 반해, word2vec을 이용한 특징 학습(feature learning) 방법에서는 전체 확률 모델이 필요하지 않다. CBOW와 Skip-gram 모델은 이진 분류 (binary classification) 방법(로지스틱 회귀분석-logistic regression)을 이용한다. 같은 콘텍스트에서 **k개의 상상의 (noise) 단어** \tilde{w} 와 진짜 타겟 단어 w_t 를 식별한다. 아래의 그림은 CBOW 모델을 보여준다. Skip-gram 모델은 단순히 화살표의 방향이 반대로 바뀐 모양이다.



Appendix - Noise-Contrastive 트레이닝을 통한 규모확장(Scaling Up)

- 위 과정을 수학적으로 표현하면, 각각의 예제에 대한 목표는 아래의 목적 함수를 최대화하는 것이다.



- 여기서 $Q_\theta(D = 1|w_t, h)$ 은 데이터셋 D에서 콘텍스트 h에서 관찰한 단어가 w일 이진 로지스틱 회귀분석에서의 확률이다. 이는 학습된 embedding vectors θ 를 이용해서 계산된다.
- 이 목적함수는 모델이 실제 단어에는 높은 확률을 부여하고 노이즈 단어들에는 낮은 확률을 부여하면 최대화된다. 기술적으로, 이런 방법을 Negative Sampling이라고 한다. 이런 종류의 손실 함수(loss function)를 정의하는 것은 수학적으로 적절하다. 제한 상황 안에서 이 함수의 업데이트는 softmax 함수의 업데이트와 근사하다. 더욱이 계산 비용의 측면에서, 이런 방식은 매우 효율적이다. 왜냐하면 전체 어휘 V를 계산하는 것이 아니라 우리가 선택한 k개의 noise 단어들만 계산하면 되기 때문이다. 따라서 트레이닝 시간을 짧게 줄여준다. 우리가 실제로 구현할 때는 위의 손실 함수와 매우 유사한 noise-contrastive estimation (NCE) 손실 함수를 사용할 것이다. 이는 텐서 플로우에 미리 구현된 `tf.nn.nce_loss()` 함수를 이용하면 된다. 이제 예제를 통해 이 모델이 실제 상황에서 어떻게 동작하는지 직관적으로 이해해보자!

Appendix - Skip-Gram Model

- 예를 들어, 아래와 같은 데이터셋이 주어졌다고 가정해보자.

the quick brown fox jumped over the lazy dog

- 먼저 콘텍스트를 정의해야한다. 콘텍스트는 어떤 형태로든 정의할 수 있지만, 사람들은 보통 구문론적 콘텍스트를 정의한다. 이번 구현에서는 간단하게, 콘텍스트를 타겟 단어의 왼쪽과 오른쪽 단어들의 윈도우로 정의한다. 윈도우 사이즈를 1로하면 (context, target) 쌍으로 구성된 아래와 같은 데이터셋을 얻을 수 있다.

([the, brown], quick), ([quick fox], brown), ([brown, jumped], fox), ...

- skip-gram 모델은 타겟 단어로부터 콘텍스트를 예측한다는 점을 상기하라. 따라서 우리가 해야 할 일은 ‘quick’이라는 타겟단어로부터 콘텍스트 ‘the’와 ‘brown’을 예측하는 것이다. 따라서 우리 데이터셋은 아래와 같은 (input, output) 쌍으로 표현할 수 있다.

(quick, the), (quick, brown), (brown, quick), (brown, fox), ...

- 목적 함수는 전체 데이터셋에 대해 정의될 수 있다. 하지만, 우리는 보통 이를 stochastic gradient descent(SGD) 방식으로 한번에 하나의 예제에 대해서 최적화한다. 또는 ‘minibatch’라고 하는 일정 개수의 배치로 묶어서 최적화한다. 일반적으로 배치 사이즈는 $16 \leq \text{batch_size} \leq 512$)이다.

Appendix - Skip-Gram Model

- 트레이닝 스텝 t 에서 위의 첫번째 트레이닝 예제를 관찰했다고 가정해보자. 우리의 목표는 ‘quick’이라는 단어로부터 ‘the’라는 단어를 예측하는 것이다. noise distribution(보통은 unigram distribution $P(w)$)에서 num_noise 개수만큼 contrastive noisy example을 추출한다. 간단하게 생각해서 num_noise 개수를 1이라고 가정하고, noisy example로 ‘sheep’이라는 단어를 선택했다고 가정하자. 그럼 스텝 t 에서 목적함수는 아래와 같이 정의된다.

$$J_{NEG}^{(t)} = \log Q_{\theta}(D = 1 | the, quick) + k E_{\tilde{w} \sim P_{noise}} [\log Q_{\theta}(D = 0 | sheep, quick)]$$

- 우리의 목적은 위의 목적함수를 최대화하도록 embedding parameters θ 를 업데이트하는 것이다. 이는 embedding parameters θ 따른 손실 함수에 대한 gradient를 구함으로써 가능하다. 즉, $\frac{\partial}{\partial \theta} J_{NEG}$ 를 구하면 된다. (다행히 TensorFlow는 이를 위한 helper function을 지원한다.) 이런 과정은 전체 트레이닝셋에 대해 반복된다. 이는 모델이 real words와 noise words를 성공적으로 구분해낼때까지 embedding vectors를 이동시킨다.

Appendix - Skip-Gram Model

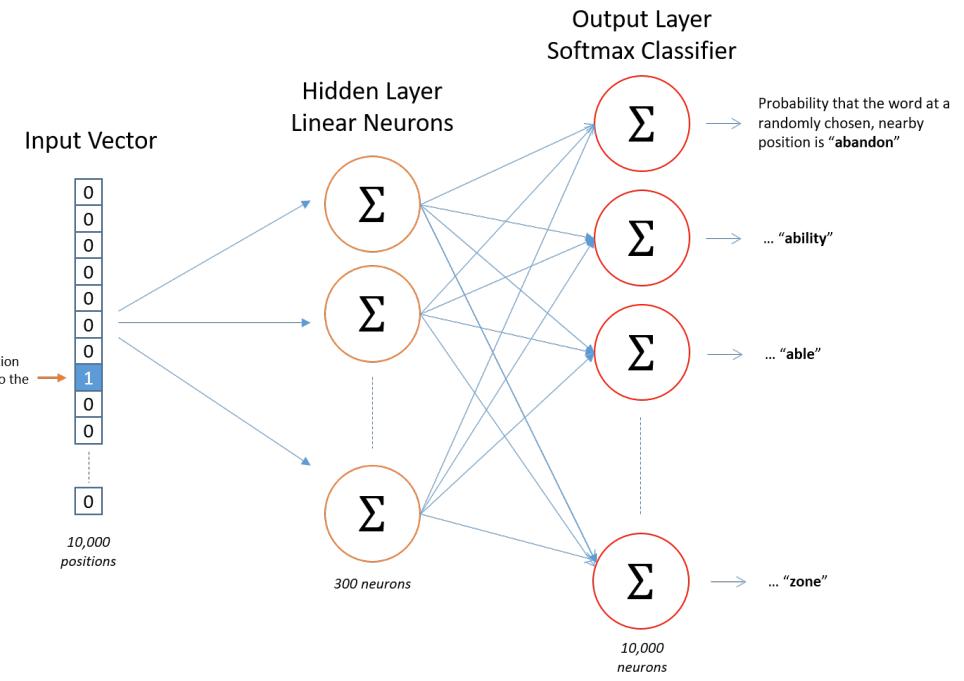
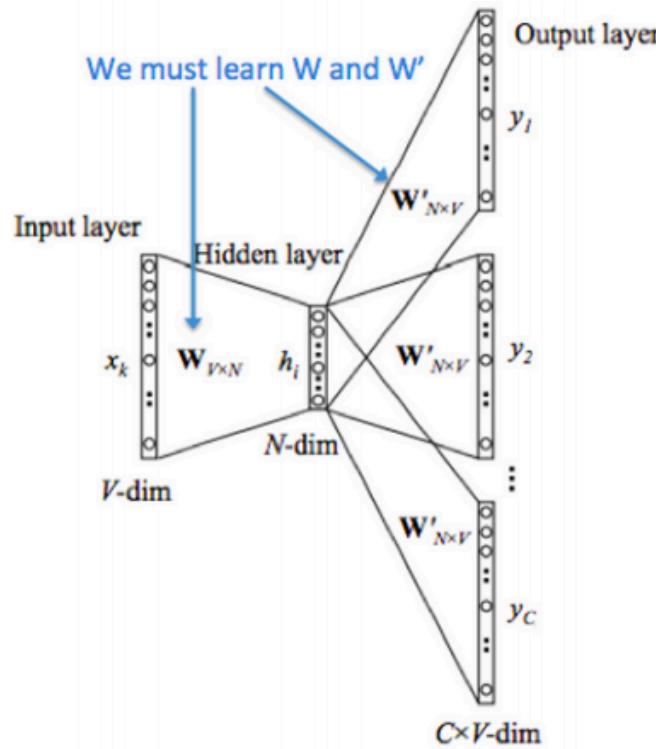
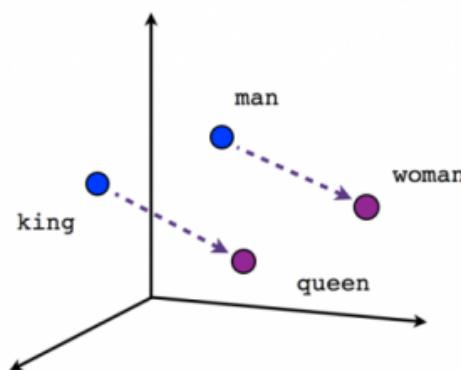


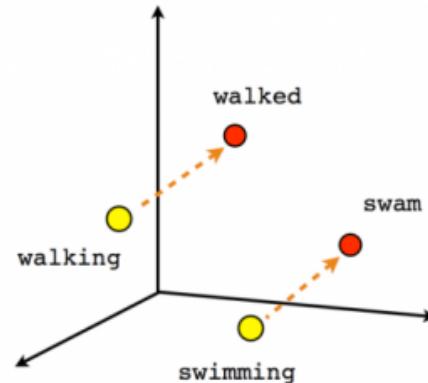
Figure 2: This image demonstrates how Skip-Gram works and how we must learn the transfer matrices

Appendix - Word2Vec 시각화

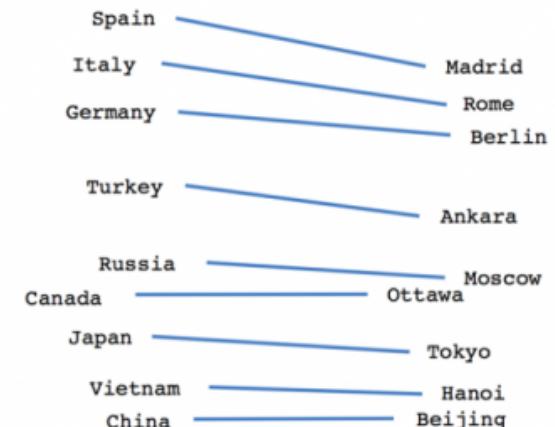
- 우리는 t-SNE dimensionality reduction technique 등을 이용해서 학습한 벡터들을 2차원으로 프로젝션해서 시각화할 수 있다. 시각화한 결과를 보면 학습한 벡터들이 유의미한 정보를 캡처하고 있다는 사실을 알 수 있다. 예를 들어, male-female, country-capital 같은 관계들을 포착하고 있다.



Male-Female



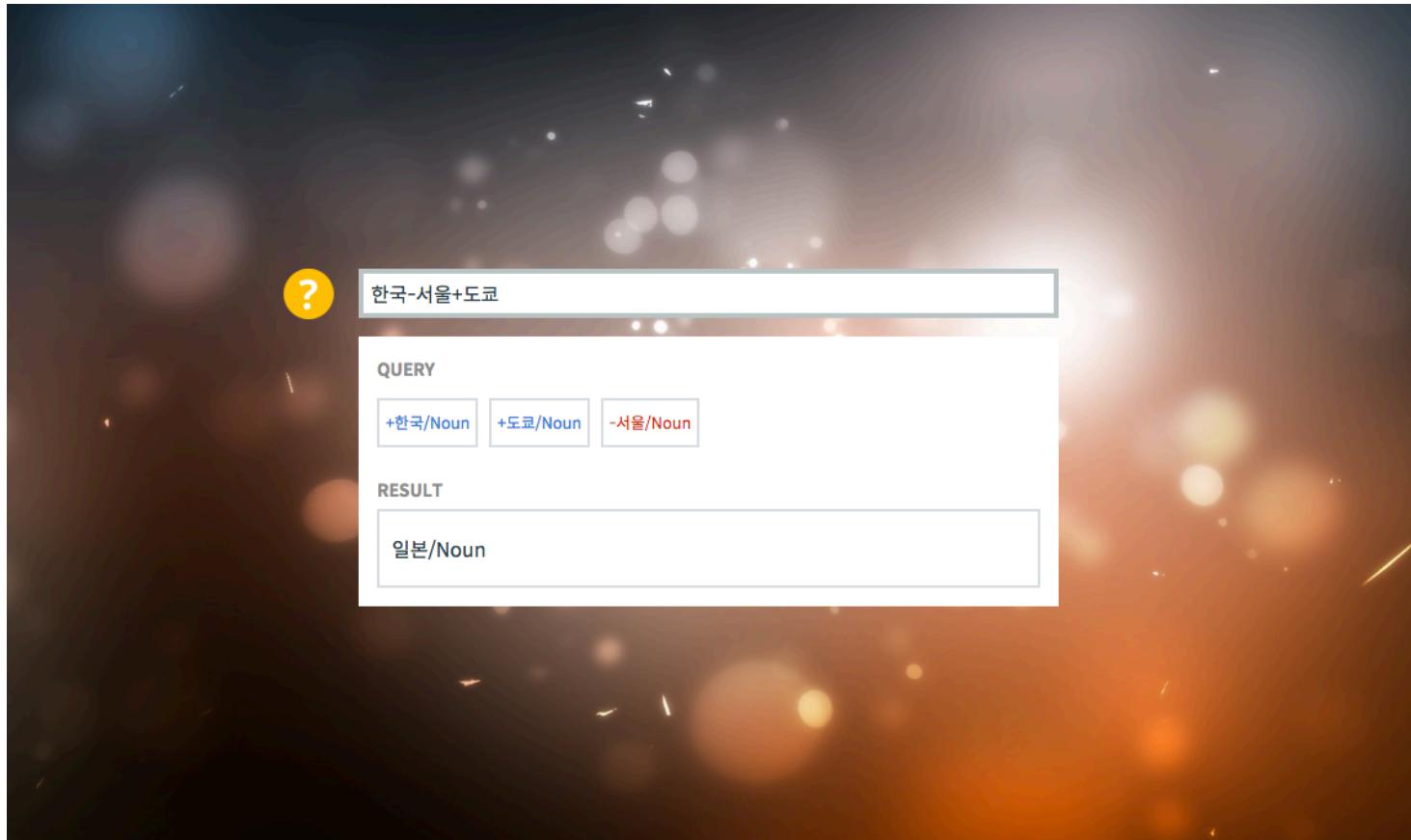
Verb tense



Country-Capital

Appendix - 한글 Word2Vec Demo

- <http://w.elnn.kr/search/>



과제 - Custom Dataset에 대한 Char-RNN 모델 구현

- TensorFlow를 이용해서 Custom Dataset에 대한 Char-RNN 모델을 구현해봅시다.

에스트라공: (무대 옆에서 몸을 비꼰다.) 앞으로!

블라디미르: 나는 우릴 고맙다 (헐떡내 다친다) 이것 사실이오. (럭키에게) 하지만 생각을 할까? 무슨 소리를 터터봐

블라디미르: 회연하는 성을 거일도 좋니?

에스트라공: 벌써 미칠태를 럭키의 주위를 지면줘요!

블라디미르: 빨리빨리! 이놈 좀 일으켜 세워요!

그는 발을 멈추고 블라디미르 모자를 놓고 럭키의 모자를 고우지 않을 거들어 줘!

블라디미르가 끈과 소년은 물러서며) 이번 한테전 것도 없었나?

에스트라공: 안 그래요!

블라디미르: 조용히!

침묵)

에스트라공: 왜 왜 누구?

블라디미르: 고도를 기다려야지.

에스트라공: 참 그렇구 말야?

블라디미르: 고도가 싫다니까. 벌써 시간이 흐르는 게 있으면

Questions & Answers

Thank You!