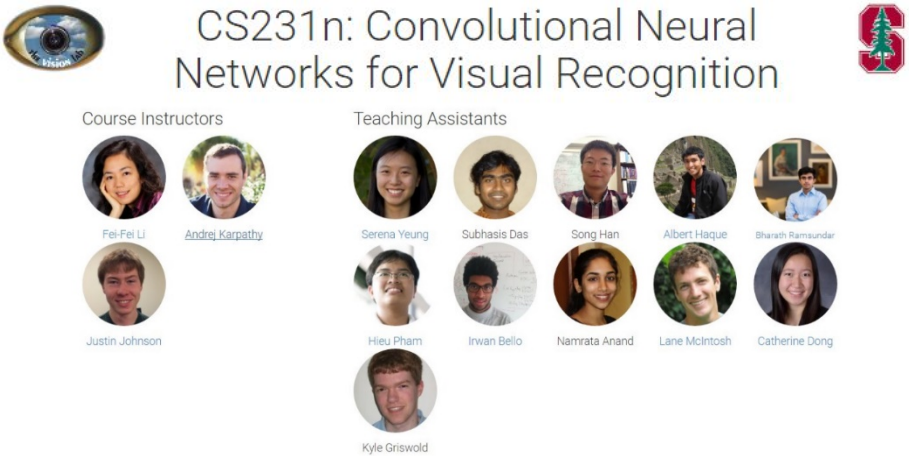
모두의연구소

- Most of the lecture materials have came from CS231n

I really appreciate that they give us the chance to study Convolutional Neural Networks and provide us with very good teaching materials

http://vision.stanford.edu/teaching/cs231n/index.html



모두의연구소

최종 정리

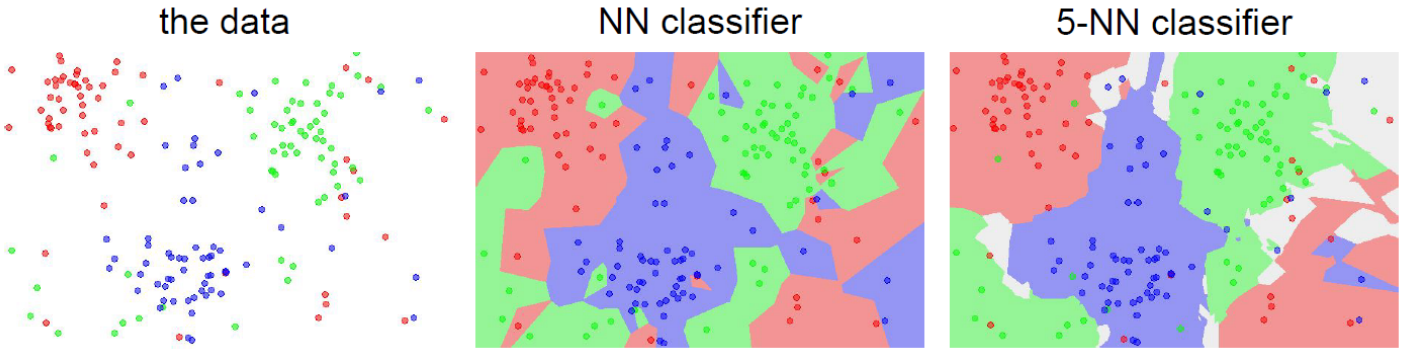
모두의연구소
박은수

Classifier와의 첫 만남 ...

모두의연구소

Nearest Neighbor은 알겠습니다. 제일 비슷한 것 찾아서 그 비슷한 것의 레이블을 반환하는 것이군요
k-Nearest Neighbor은 무엇인가?

k개의 Nearest Neighbor를 찾고, k개 중 많이 나온 레이블을 선택 (voting)



http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

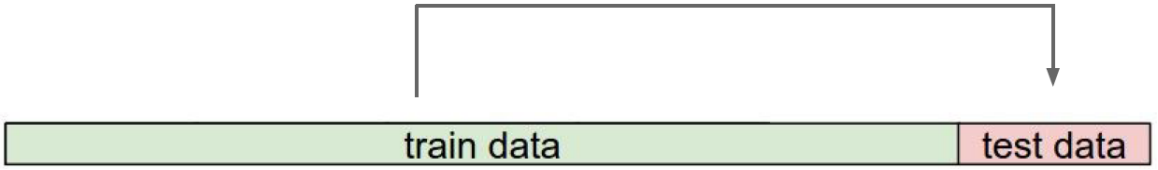
데이터 셋 관리

모두의연구소

하이퍼 파라미터를 선택 한 후 테스트 데이터를 통해 확인

안좋은 생각이다. 테스트 데이터는 실제 환경에서도 잘 작동됨을 보여야 할 최후의 보루

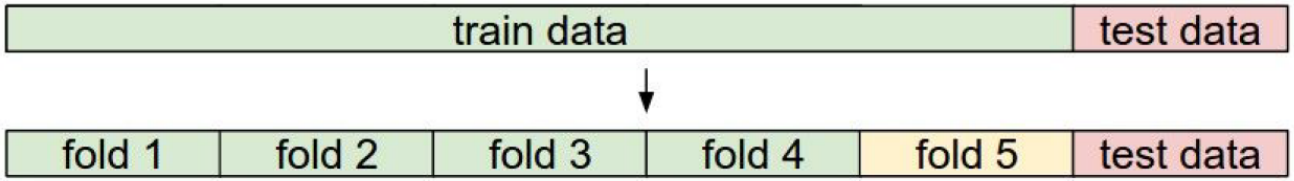
마지막을 위해 아껴라~!!



데이터 셋 관리

모두의연구소

Cross-Validation



Cross-validation

cycle through the choice of which fold is the validation fold, average results.

이제 테스트에 루프를 돌면서 트레이닝에 사용할 데이터와 테스트에 사용할 데이터 셋을 바꿔가면서 실험해 봅니다

모두의연구소

데이터를 소중하게 생각하고 잘 관리해야 하는 구나 …

픽셀끼리 distance 구해서 NN 로 분류하면 성능이 안좋겠지 …

이렇게 하지말고 Score 만들고 그랬던 것 같은데 …

모두의연구소

Classifier를 어떻게 만든다고 했지?

맞아 3가지 단계가 있다고 했어~!!

모두의연구소

- Classifier 생성의 3단계

1. **Score function 만들기**
2. **Loss function 만들기**
3. **Optimization 하기**

맞아 3가지 단계가 있다고 했어~!!

모두의연구소

- Classifier 생성의 3단계

1. **Score function 만들기**
2. **Loss function 만들기**
3. **Optimization 하기**

처음 만들어 본 Linear Score 함수

$$f(x, W) = Wx$$

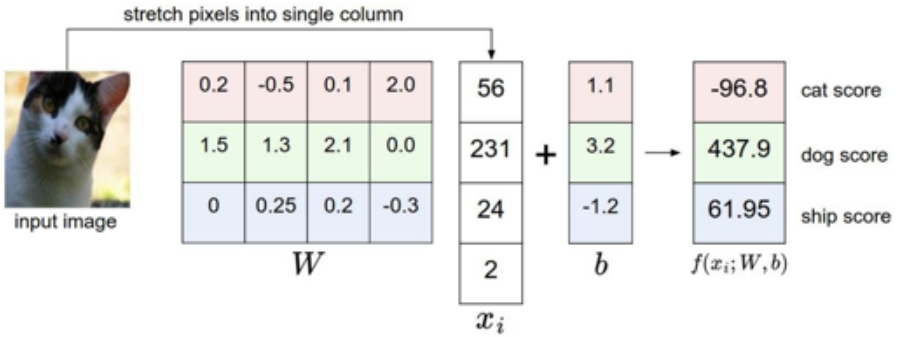
10x1 (output) = 10x3072 (W) × 3072x1 (x)

(+b) 10x1

[32x32x3]
array of numbers 0...1

10 numbers, indicating class scores

parameters, or "weights"



맞아 3가지 단계가 있다고 했어~!!

모두의연구소

- Classifier 생성의 3단계

1. **Score function 만들기**
2. **Loss function 만들기**
3. **Optimization 하기**

모두의연구소

Loss는 얼마나 Score가 얼마나 안좋은지를 측정하는 함수라고 했던 것 같아...

1. **Score function 만들기**
2. **Loss function 만들기**
3. **Optimization 하기**

Loss는 얼마나 Score가 얼마나 안좋은지를 측정하는 함수라고 했던 것 같아...

이게 완성되면 이 Loss를 최소로 하는 W를 찾는게 Optimization이라고 했었지 ... 맞아 맞아~

1. **Score function 만들기**
2. **Loss function 만들기**
3. **Optimization 하기**

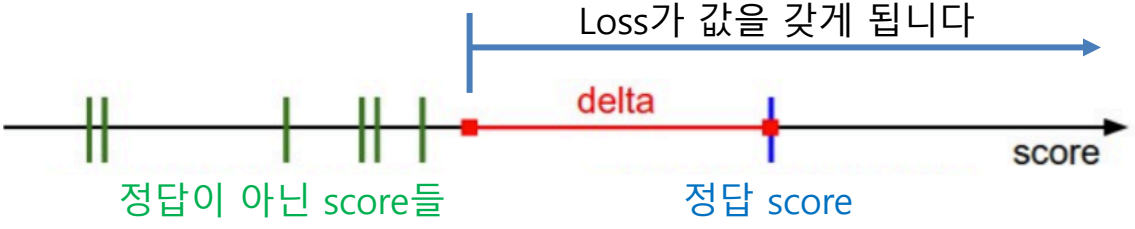
SVM Loss는 뭔 소리였을까 …

$$L_i = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta)$$

i 번째 이미지의 Loss

정답이 아닌 것들만 합한다

정답이 아닌 score와 정답 score와의 차이



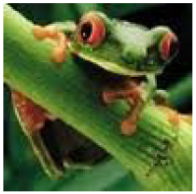
- 최소한 Loss가 0이 아닌 값을 가지려면 정답이 아닌 녀석들이 저 붉은 delta위치에는 오거나 정답 Score보다 크면 되겠네요

SVM Loss : 예제를 보니 감이 올 듯 …

$$L_i = \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta)$$

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:





cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9		

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

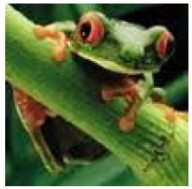
= max(0, 5.1 - 3.2 + 1)
+max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9

SVM Loss : 예제를 보니 감이 올 듯 …

모두의연구소

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:





cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

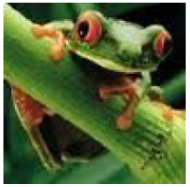
= max(0, 1.3 - 4.9 + 1)
+max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0

SVM Loss : 예제를 보니 감이 올 듯 …

모두의연구소

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:





cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

= max(0, 2.2 - (-3.1) + 1)
+max(0, 2.5 - (-3.1) + 1)

= max(0, 6.3) + max(0, 6.6)
= 6.3 + 6.6
= 12.9

SVM Loss : 예제를 보니 감이 올 듯

모두의연구소

...

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	12.9

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

and the full training loss is the mean over all examples in the training data:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

L = (2.9 +0+12.9) /3
= 5.26

모두의연구소

Loss function 뒤에 뭐가 붙어 있던 것 같았는데 ....

Loss function 뒤에 뭐가 붙어 있던 것 같았는데 ....

맞아~ Regularization ~!!

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

In common use:

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

이거 효과가 아마 아래와 같을 꺼야

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

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1^T x = w_2^T x = 1$$

Score가 같군~!

이거 효과가 아마 아래와 같을 꺼야

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

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1^T x = w_2^T x = 1$$

Score가 같군~!

- 이 *W*로 *R*(*W*)를 계산해 볼까 ?

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

이거 효과가 아마 아래와 같을 꺼야

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

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1^T x = w_2^T x = 1$$

Score가 같군~!

- 이 *W*로 *R*(*W*)를 계산해 볼까 ? $R(W) = \sum_k \sum_l W_{k,l}^2$
  - w_1 : 1
  - w_2 : 0.25

이거 효과가 아마 아래와 같을 꺼야

모두의연구소

- w_1 일때 R(W)는 1, w_2 일땐 0.25

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

Loss를 최소화해야하는게 Optimization의 목적이니깐, w_1 보다는 w_2 쪽으로 최적화 되겠구나 …

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

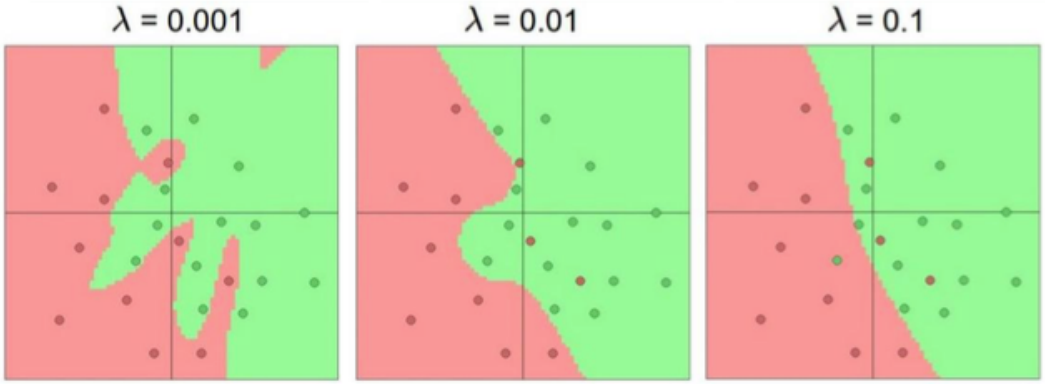
얘는 하나의 특징만 보겠구나…

얘는 좀 골로구 보네 …

이거 효과가 아마 아래와 같을 꺼야

또 이걸로 또 뭐했더라

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



- λ 가 커지면 W도 역시 작은 값을 갖을 수 밖에 없겠지 … Loss를 최소화해야 하니깐~
- 이걸로 모델의 표현력도 조절하고 overfitting도 방지할 수 있겠구나

모두의연구소

SVM Loss말고 더 많이 쓰는 Loss 있다고 했는데 …

모두의연구소

SVM Loss말고 더 많이 쓰는
Loss 있다고 했는데 …

맞아 Softmax Loss

Softmax 함수

모두의연구소

- Score를 구한다음 $s = f(x_i; W)$
- 요렇게 하는 거였어
  - 이렇게 하면 0~1범위의 확률로 표현뒤는 구나 …

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

  - 그리고 exponential로 각 스코어들의 차이를 좀 더 벌려 놓을 수도 있었지 …

Softmax 함수

모두의연구소

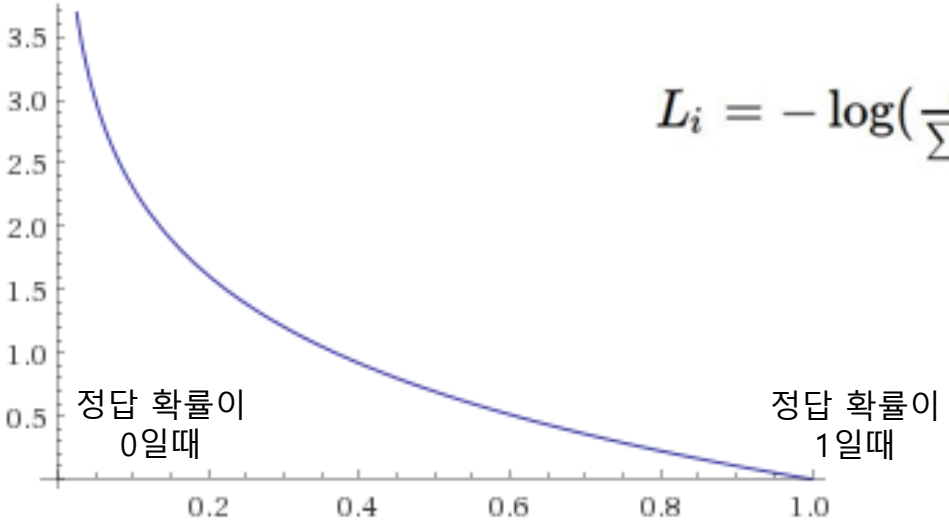
$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

- Softmax Loss는 정답 확률에 마이너스 로그를 붙이는 것이였어
- 정답은 높여야 하는거잖아.. 그런데 우린 로스를 정의하고 있으니깐 "–" 를 붙이면 되겠다~

Softmax 함수

모두의연구소

- 0~1 사이의 -log함수를 그려볼까~



$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

- 정답의 확률이 1이라면 Loss는 0이 되는구나~
- 정답임에도 불구하고 그 확률이 매우 낮으면 엄청 큰 Loss값을 갖게 되는군 ~~~ 아하~~~!!

Softmax 함수

모두의연구소

- Softmax한번 구해보자~

Softmax Classifier (Multinomial Logistic Regression)

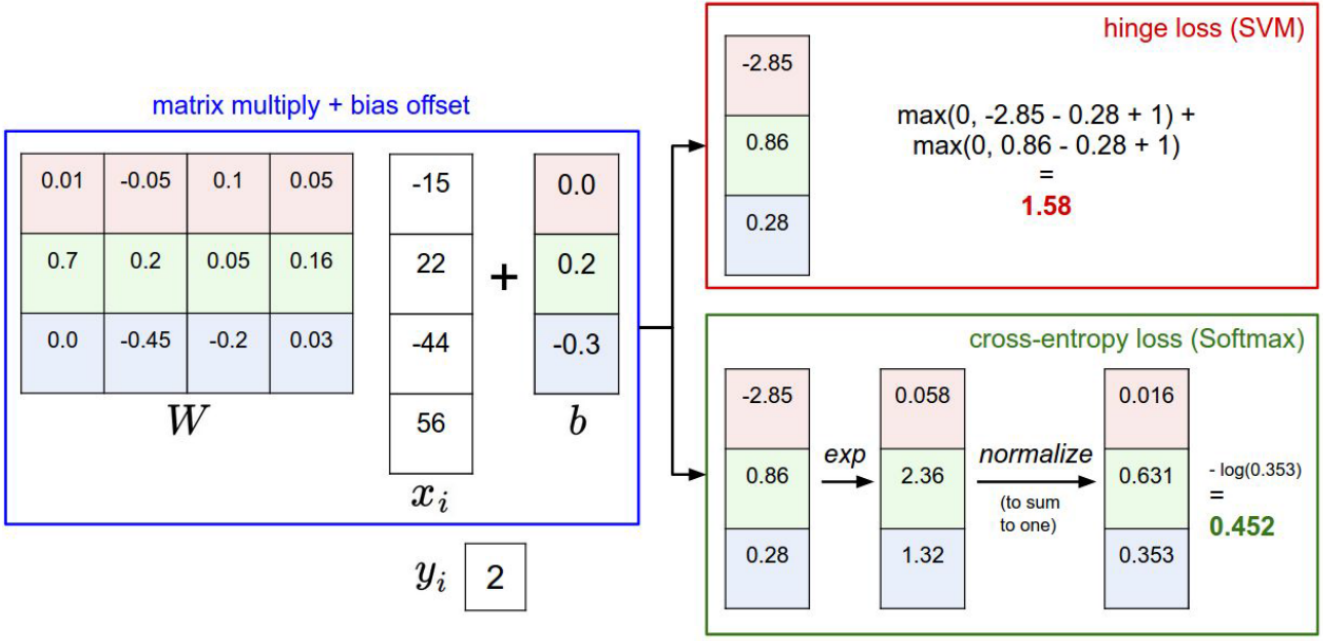
$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

	unnormalized log probabilities	exp	unnormalized probabilities	normalize	probabilities	
cat	**3.2**		**24.5**		**0.13**	L_i = -log(0.13) = **0.89**
car	5.1		164.0		0.87	
frog	-1.7		0.18		0.00	

둘 다 한번 같이 해보자

모두의연구소



맞아 3가지 단계가 있다고 했어~!!

모두의연구소

- 이제 마지막 단계군~

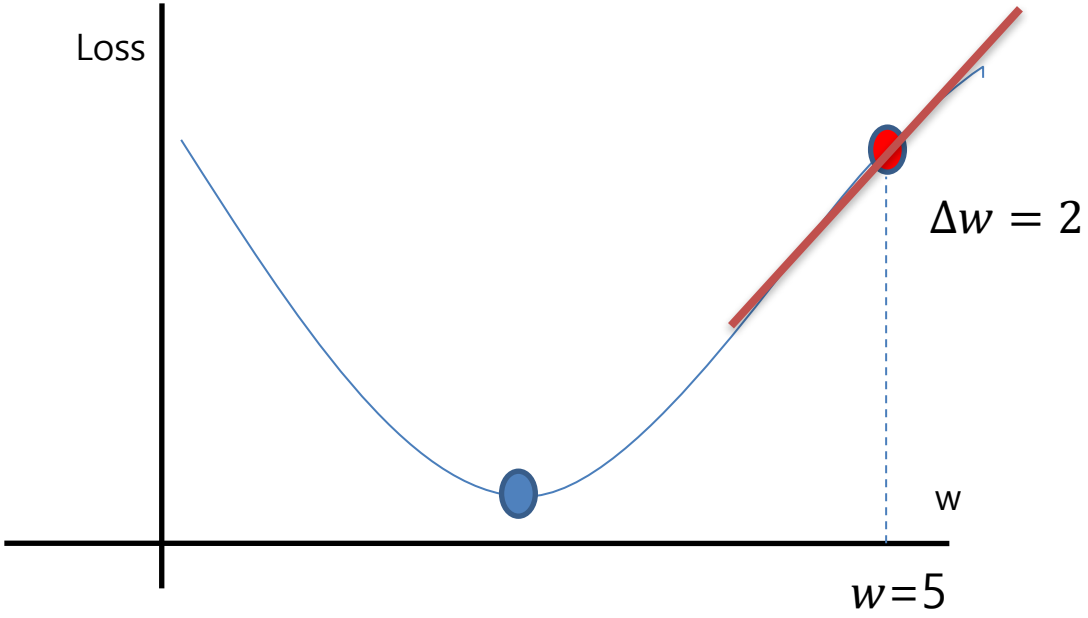
1. **Score function 만들기**
2. **Loss function 만들기**
3. **Optimization 하기**

모두의연구소

Gradient Descent를
이용한다고 했었지

아마 이런 개념이였었지...

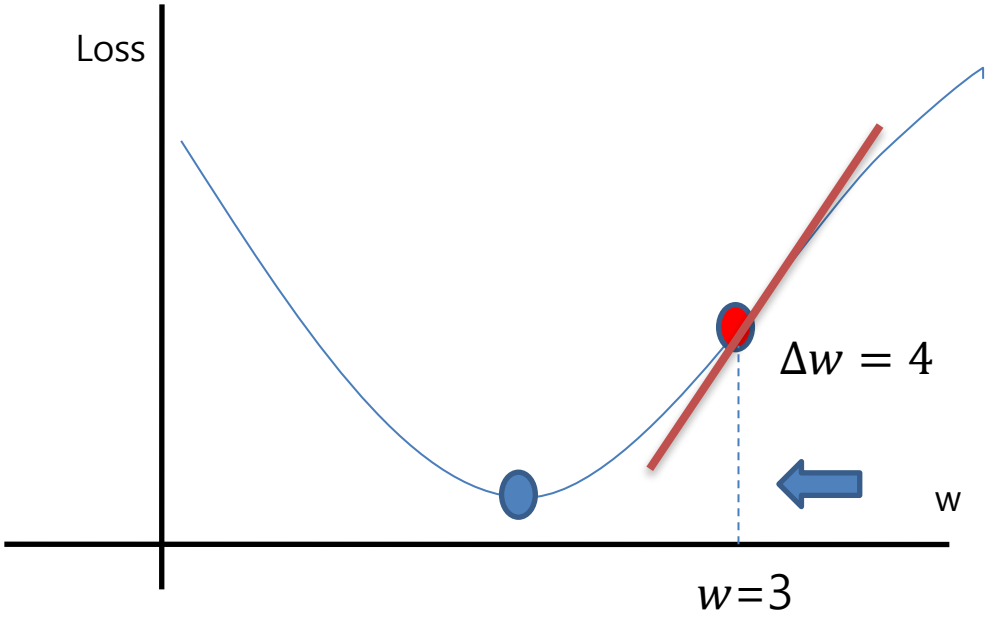
Gradient Descent



$$w = \mathrm{w} - \Delta w$$

$$3 = 5 - 2$$

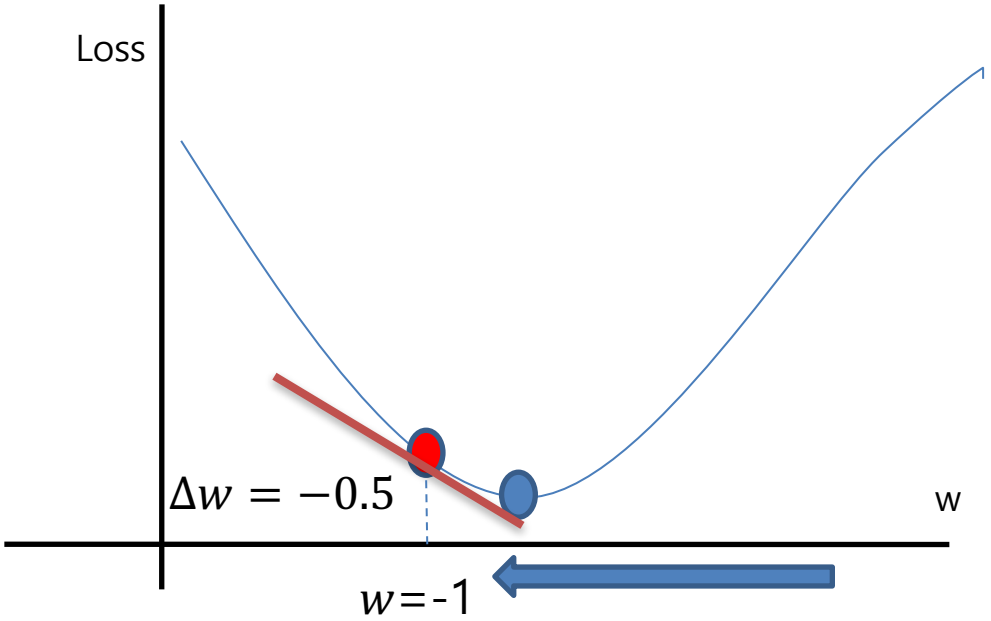
Gradient Descent



$$w = \mathrm{w} - \Delta w$$

$$-1 = 3 - 4$$

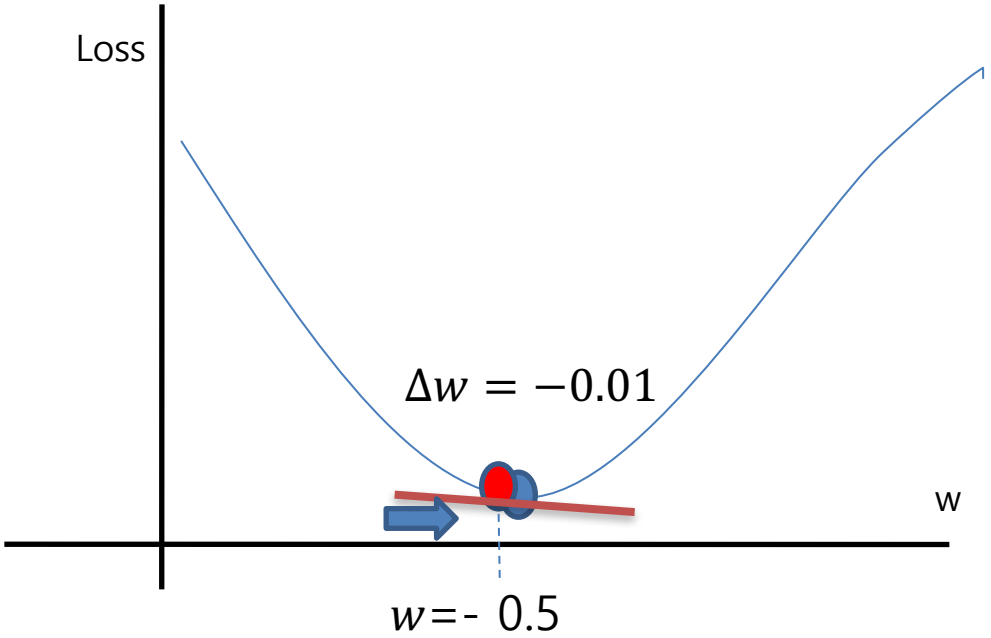
Gradient Descent



$$w = \mathrm{w} - \Delta w$$

$$-0.5 = -1 - (-0.5)$$

Gradient Descent



$$w = \mathrm{w} - \Delta w$$

$$-0.5 = -1 - (-0.5)$$

그레디언트는 어떻게 구했었지?

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

want $\nabla_W L$

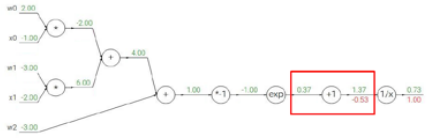
Calculus

배웠던 Score function,
Loss function이 저기 다 있군

BP (Back Propagation)을 했었지

모두의연구소

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

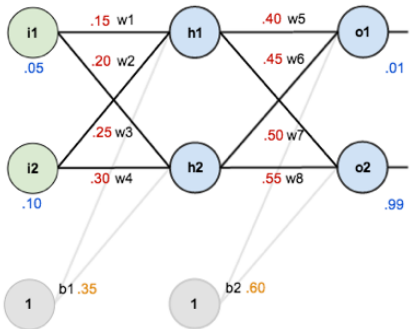
$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

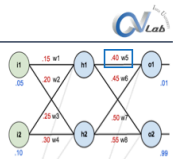
$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

괜찮은 2-layer 예제도 있었어. 직접 값의 변화를 볼 수 있었지 꼭 직접 들어가서 해봐야 해 : http://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

initial weights, the biases, and training inputs/outputs:

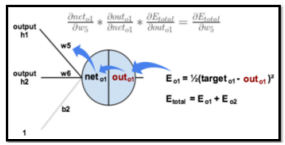


The Backwards Pass



- 일단 w5의 변화가 최종 에러에 어떤 영향을 줄지를 계산해 봅시다. 즉 $\frac{\partial E_{total}}{\partial w_5}$ 를 계산해 봅시다
  - 요걸 the gradient with respect to w5라고 합니다
- 이걸 계산하기위한 Chain rule은 다음과 같습니다

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$



W 업데이트 단위는 어떻게 되는거지?

모두의연구소

- only use a small portion of the training set to compute the gradient.

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Common mini-batch sizes are 32/64/128 examples
e.g. Krizhevsky ILSVRC ConvNet used 256 examples

- 전체 다 쓰지 않고 부분만을 취해서 업데이트 하는걸 : **mini-batch gradient descent**라고 했었지 (위의 경우, 256)
- 만약 위에 256이 1이면 ? **SGD** (Stochastic Gradient Descent). 이건 지금 처음 얘기해준 것 같아
- **그런데 대부분 그냥 mini-batch Gradient Descent를 SDG라고 불러**

모두의연구소

결국 Optimization을 통해서 최종적으로 좋은 Score function을 만드는 거구나. 이 Score function에 있는 w가 업데이트 되고...

실제 Test할땐 그냥 Score function만 필요하

Cifar-10의 경우 $f(x, W) = Wx$

10x1 10x3072 3072x1

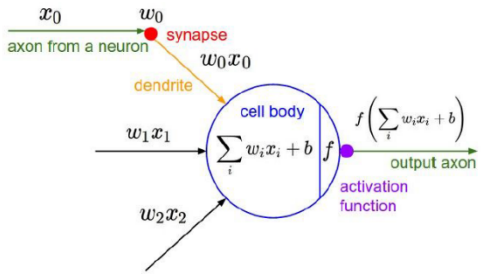
1. **Score function 만들기**
2. **Loss function 만들기**
3. **Optimization 하기**

모두의연구소

그다음에
Neural Network 를 배웠었어

모두의연구소

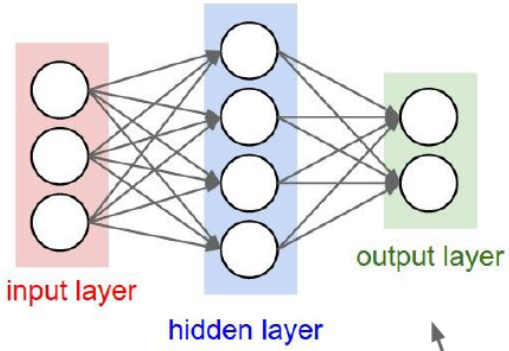
Score function하나가 Neuron하나와 비슷했던 것 같아

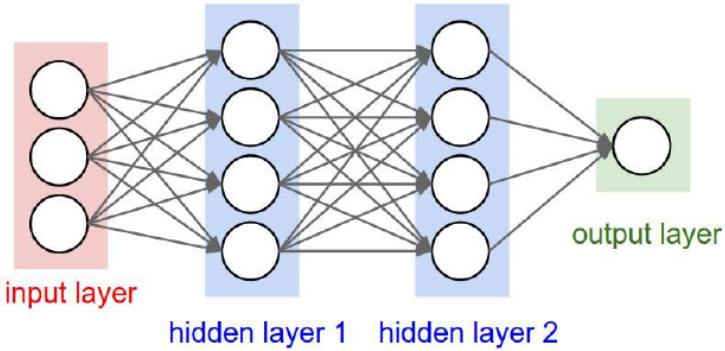


저기 뒤에 Loss만 붙이고 w 업데이트하면 Binary Classifier가 되겠군

모두의연구소

Neuron을 연결하면 Neural Network가 되겠군





"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"3-layer Neural Net", or
"2-hidden-layer Neural Net"

"Fully-connected" layers

뉴럴네트워크엔 Activation이 필요했지

모두의연구소

Activation (non-linearity)이 없으면 아무리 네트워크를 쌓아도 하나의 레이어 샌드위치와 같아질 꺼야

$$W_4(W_3(W_2(W_1x)))$$

$$Wx$$

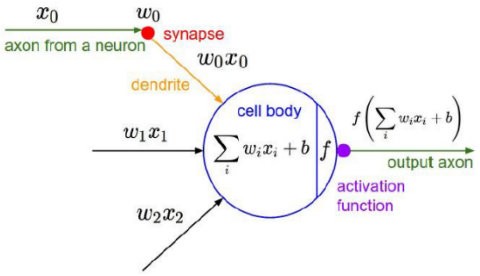
레이어를 통과한 후 Activation을 줘야하는거야

Activation function

$$f(W_4(f(W_3(f(W_2(f(W_1x)))))))$$

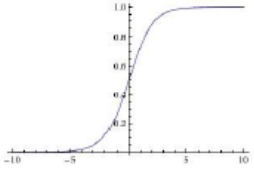
Activation이 여러종류가 있었지 ...

모두의연구소

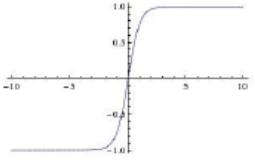


Sigmoid

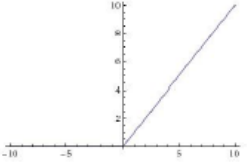
$$\sigma(x) = 1/(1+e^{-x})$$



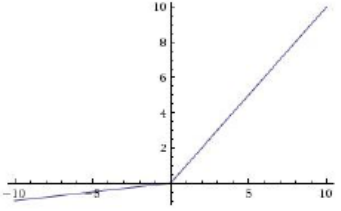
tanh tanh(x)



ReLU max(0,x)



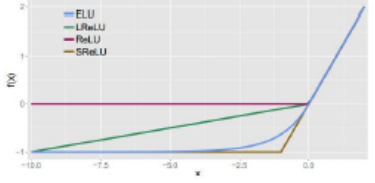
Leaky ReLU
max(0.1x, x)



Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

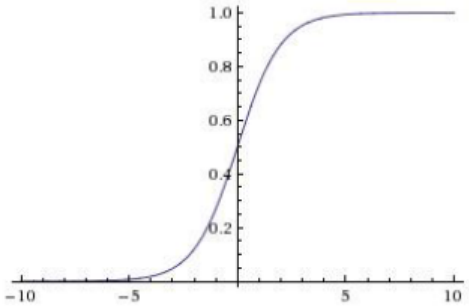
ELU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$



Gradient가 왜 Kill되는지도 배웠지

모두의연구소



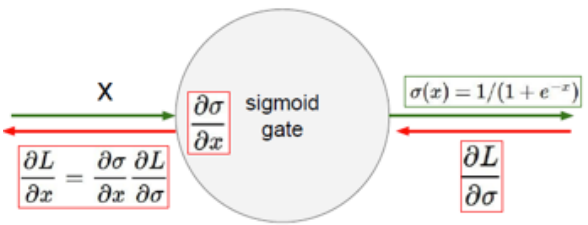
Sigmoid

$$\sigma(x) = 1/(1+e^{-x})$$

- **값을 [0,1] 사이로 조절해 준다**
- **과거 가장 널리 사용되었다**
  - 뉴런의 firing을 적절히 묘사하여
  - 미분가능

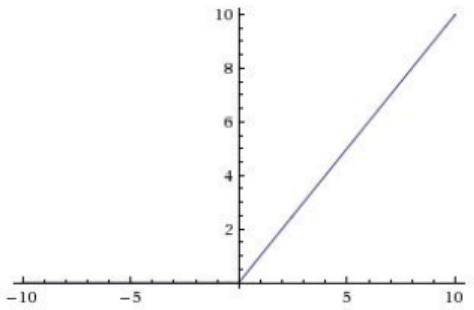
- **3가지 문제가 있음**

1. 양끝의 값을 갖는 뉴런들이 gradient를 kill 시킨다
2. 결과값의 중간값이 0 이 아님 (0~1로 양수만 출력함)
3. Exp함수는 느리고 계산량이 크다



ReLU가 좋다고 했었지

모두의연구소



ReLU
(Rectified Linear Unit)

- Computes **f(x) = max(0,x)**

- + 영역에서 값이 포화되지 않음
- 계산량이 적음
- Sigmoid/tanh보다 training 시 빠르게 수렴하는 경향이 있음 (약 6배정도?)

- 0 중심이 아님
- 불편한 점 :
  - 0 보다 작으면 gradient killed

이렇게 정리했었지

모두의연구소

TLDR (Too Long Didn't Read)

- **ReLU**를 사용하세요. Learning rate 주의하시구요
- **Leaky ReLU / Maxout / ELU** 등도 시도 해보세요
- **tanh** 도 해보세요. 기대는 마세요
- **Sigmoid**는 사용하지 마세요

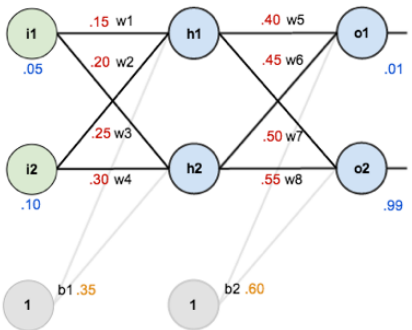
이제 다시하면 이해할 수 있을꺼야

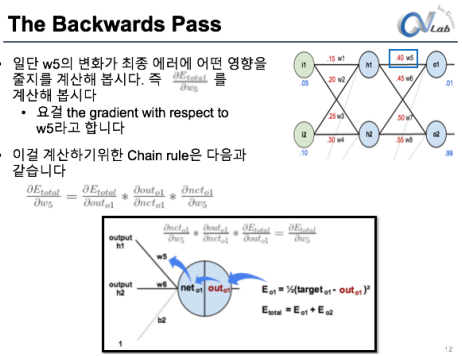
모두의연구소

아까 얘기했던 2 layer 예제야 :
http://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

원래 Backpropagation이 가장 어려워...

initial weights, the biases, and training inputs/outputs:



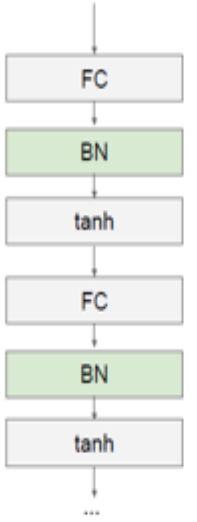


모두의연구소

W 초기화가 어려워서 Batch normalization 도 한다고 했었지...

Batch Normalization

[Ioffe and Szegedy, 2015]



Usually inserted after Fully Connected / (or Convolutional, as we'll see soon) layers, and before nonlinearity.

Problem: do we necessarily want a unit gaussian input to a tanh layer?

Normalize:

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

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

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

튜닝해야 할 많은 하이퍼 파라미터가 있었지

모두의연구소

Hyperparameters to play with:

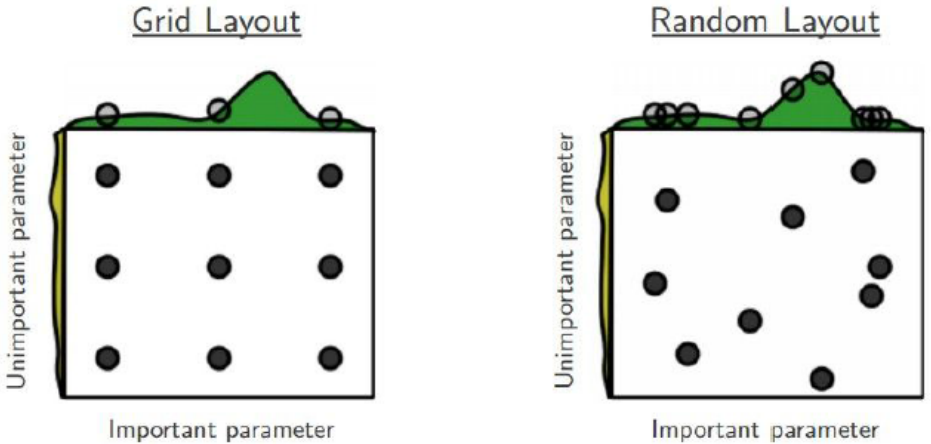
- network architecture
- learning rate, its decay schedule, update type
- regularization (L2/Dropout strength)

neural networks practitioner
music = loss function

랜덤하게 찾는게 좋다고 했어

모두의연구소

Random Search vs. Grid Search

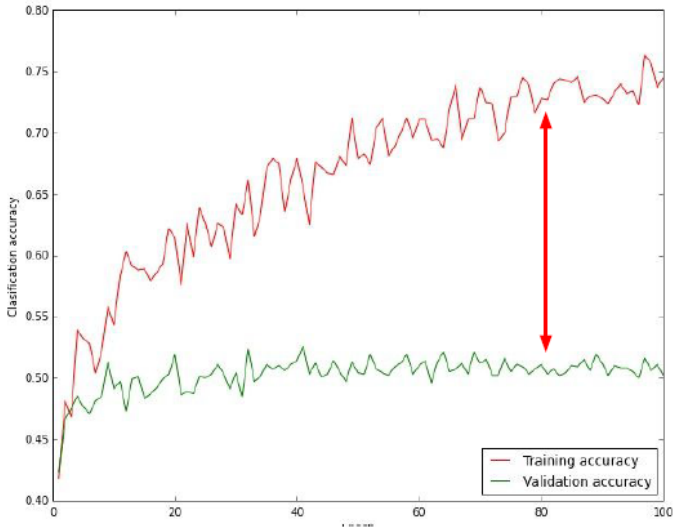


Random Search for Hyper-Parameter Optimization
Bergstra and Bengio, 2012

트레이닝(학습)이 잘되는걸 시각화 해봐야해

모두의연구소

Monitor and visualize the accuracy:



big gap = overfitting

=> increase regularization strength?

no gap

=> increase model capacity?

체크해야 할 중요한 내용이군

모두의연구소

Summary

TLDRs

We looked in detail at:

- Activation Functions (use ReLU)
- Data Preprocessing (images: subtract mean)
- Weight Initialization (use Xavier init)
- Batch Normalization (use)
- Babysitting the Learning process
- Hyperparameter Optimization
  (random sample hyperparams, in log space when appropriate)

업데이트 방법도 여러가지가 있었지

모두의연구소

```
while True:
  data_batch = dataset.sample_data_batch()
  loss = network.forward(data_batch)
  dx = network.backward()
  x += - learning_rate * dx
```

- Momentum

```
# Momentum update
v = mu * v - learning_rate * dx # integrate velocity
x += v # integrate position
```

- AdaGrad

```
# Adagrad update
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

- RMSProp

```
# RMSProp
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

- Adam

```
# Adam
m = beta1*m + (1-beta1)*dx # update first moment
v = beta2*v + (1-beta2)*(dx**2) # update second moment
x += - learning_rate * m / (np.sqrt(v) + 1e-7)
```

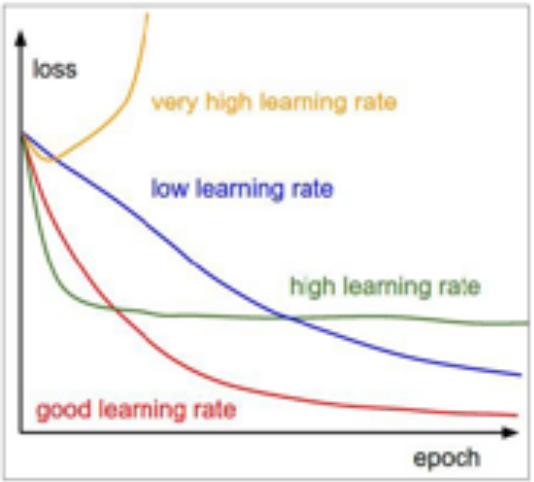
momentum

RMSProp-like

Learning rate도 스케쥴링 할 수 있어~

모두의연구소

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.



=> Learning rate decay over time!

step decay:
e.g. decay learning rate by half every few epochs.

exponential decay:

$$\alpha = \alpha_0 e^{-kt}$$

1/t decay:

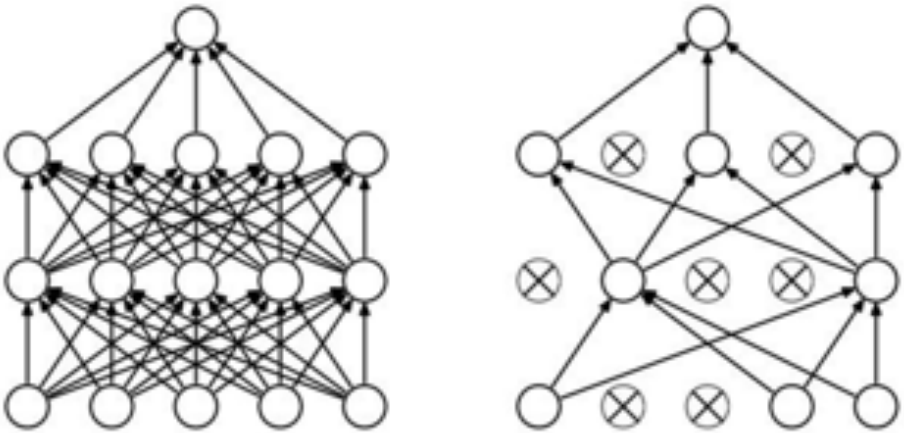
$$\alpha = \alpha_0 / (1 + kt)$$

Dropout 도 배웠었지

모두의연구소

Regularization: **Dropout**

"randomly set some neurons to zero in the forward pass"



(a) Standard Neural Net

(b) After applying dropout.

[Srivastava et al., 2014]

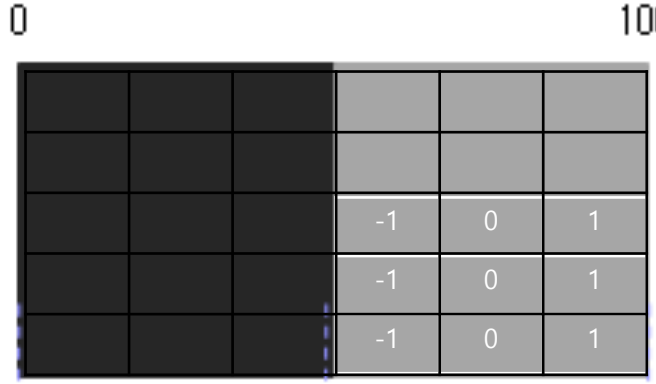
모두의연구소

CNN도 공부했었지

컨볼루션이 뭔지 배웠었어

모두의연구소

- 컨볼루셔널 뉴럴넷 (Convolutional Neural Network)
  - 컨볼루션 필터 : 이미지의 특징을 추출할 수 있다



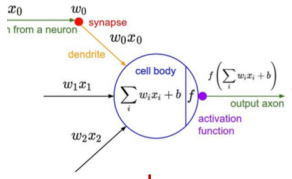
0	300	300	0
0	300	300	0
0	300	300	0

수직 에지 검출

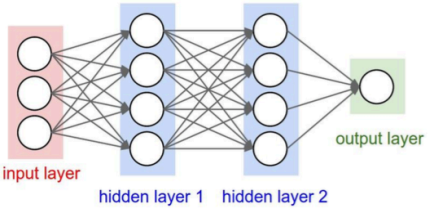
다양한 필터로 다양한 특징을 추출할 수 있다

컨볼루션을 제외하면 NN과 거의 같아

- 컨볼루셔널 뉴럴넷 (Convolutional Neural Network)
  - 필터를 학습하는 구조다

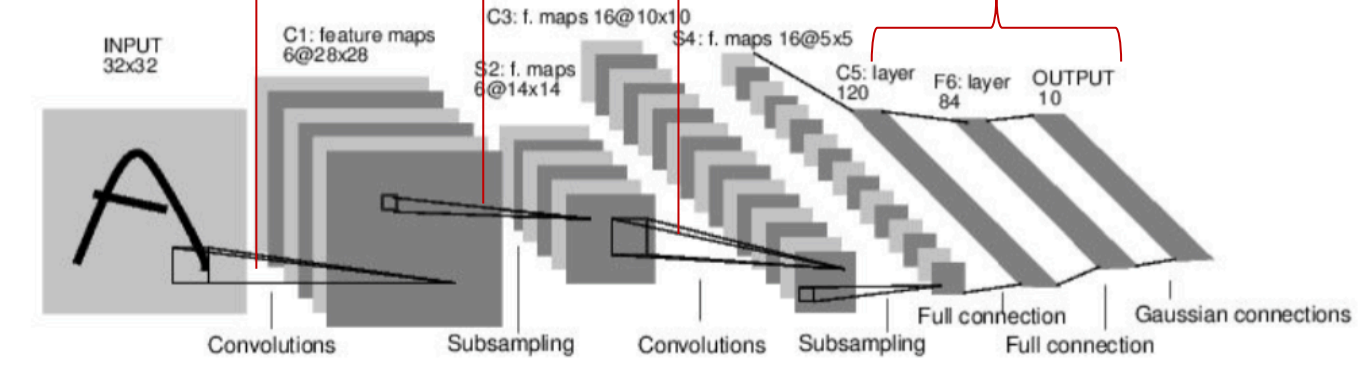


필터가 학습됨

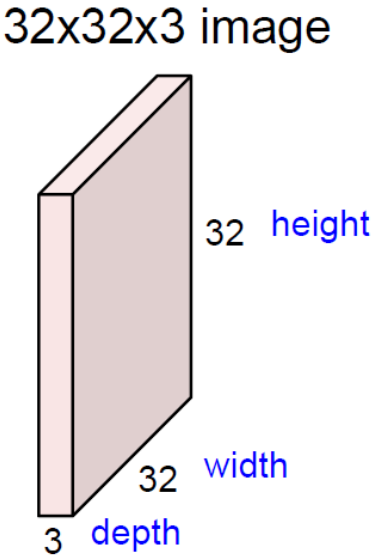


똑같은 구조

[LeCun et al., 1998]

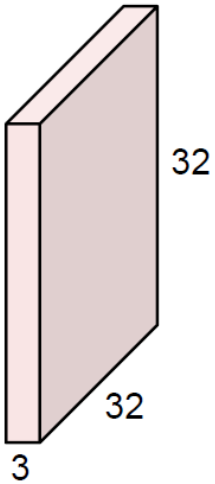


Convolution Layer

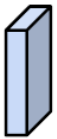


Convolution Layer

32x32x3 image



5x5x3 filter

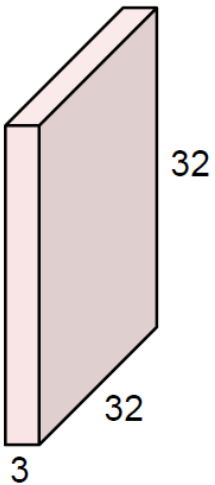


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

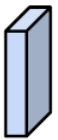
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

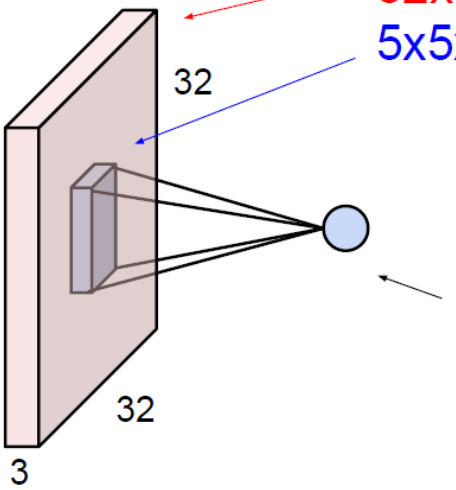


5x5x3 filter



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

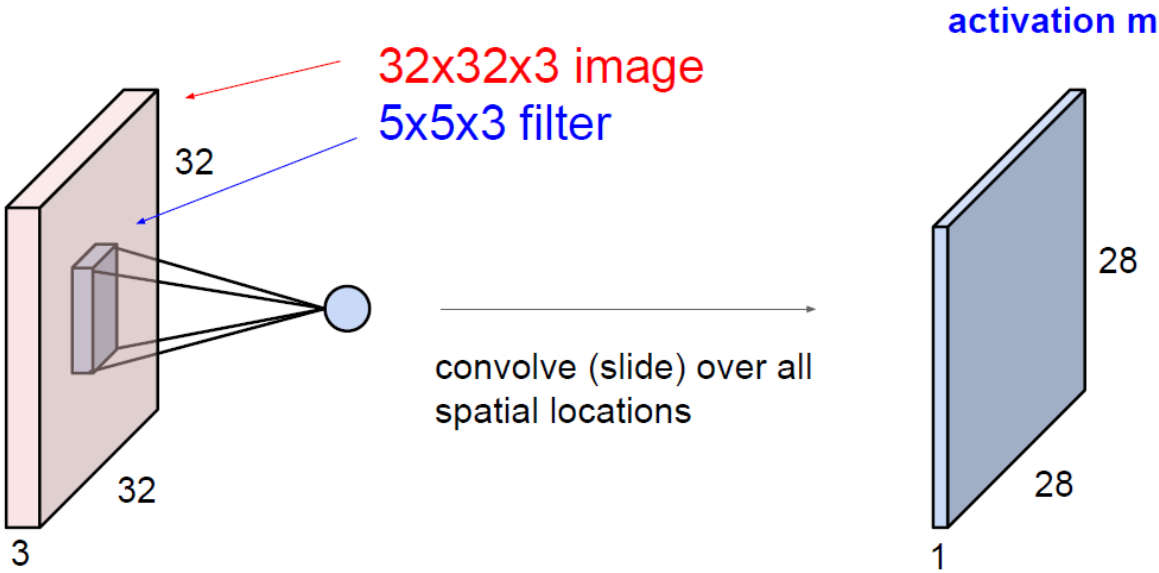
Convolution Layer



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)

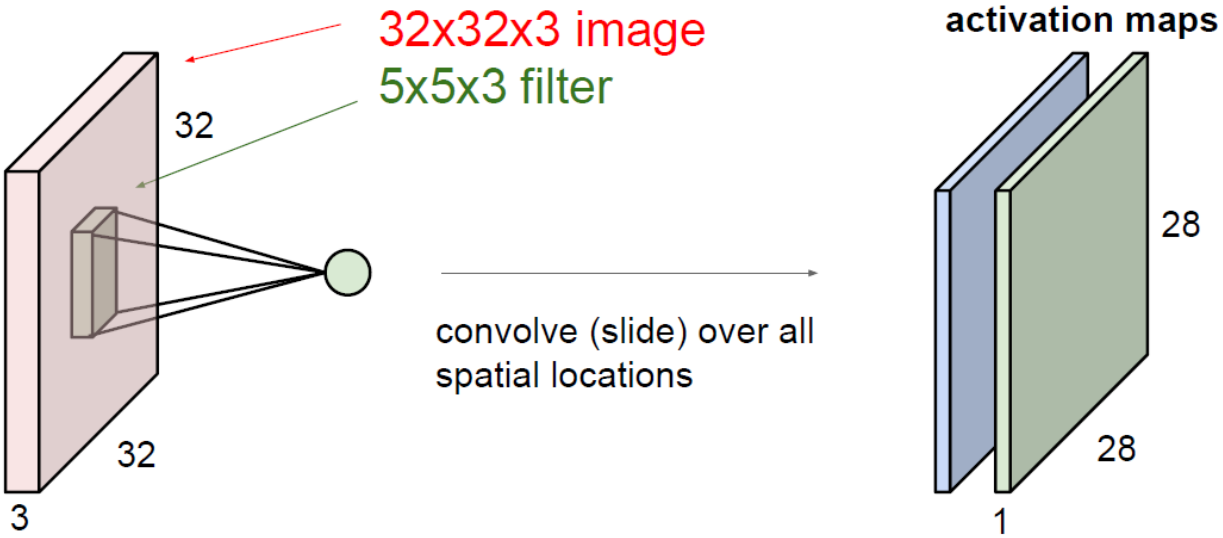
$$w^T x + b$$

Convolution Layer

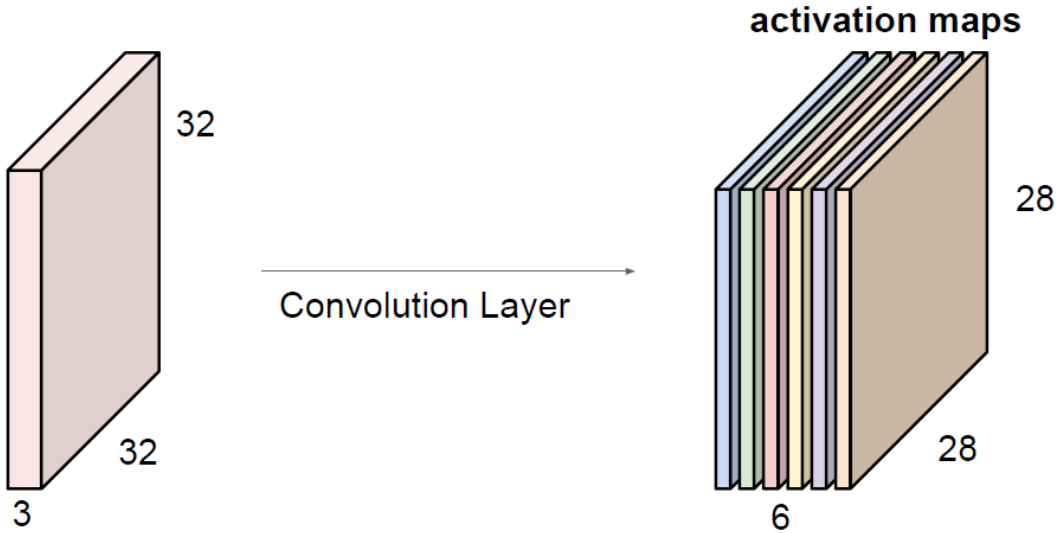


Convolution Layer

consider a second, green filter

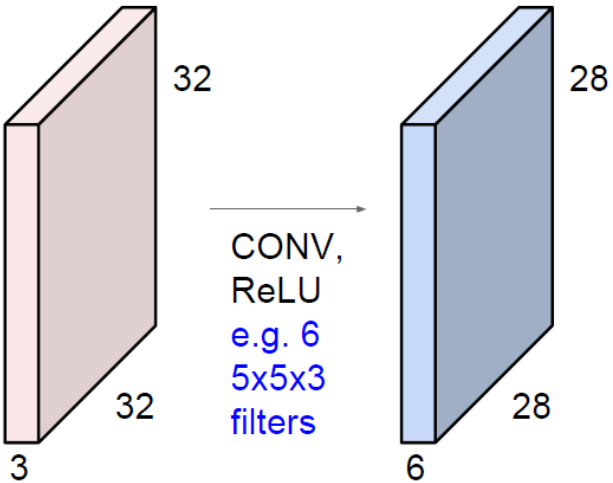


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

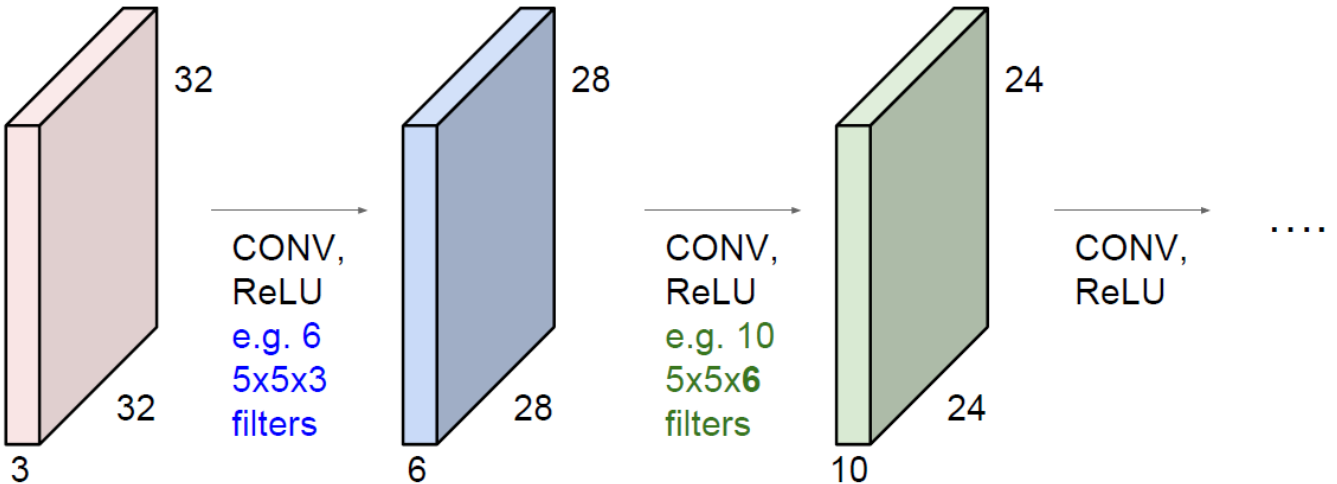


We stack these up to get a "new image" of size 28x28x6!

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



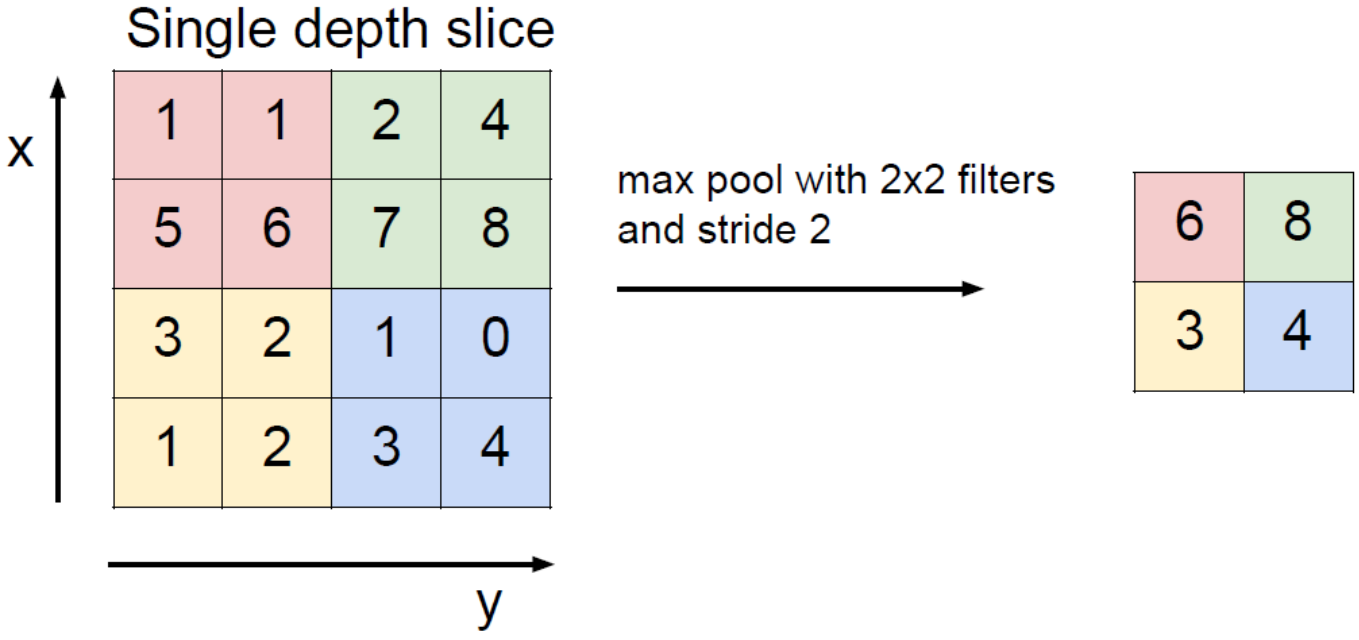
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



중간중간 Pooling도 했었지

모두의연구소

MAX POOLING

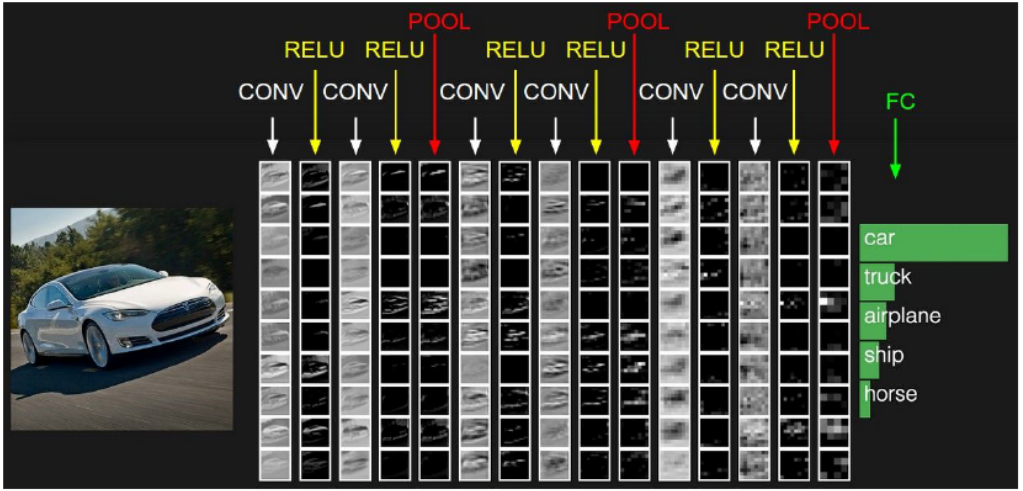


마지막엔 보통 FC layer가 위치해

모두의연구소

Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks

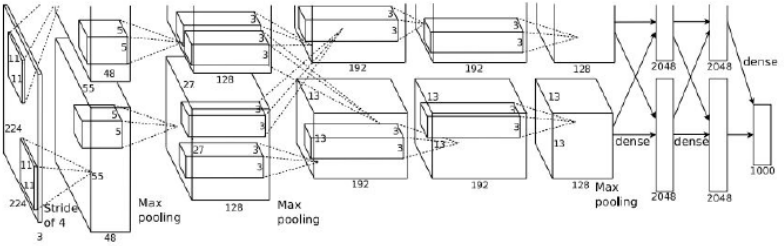


AlexNet을 살펴봤었지

모두의연구소

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%

VGGNet이 가장 많이 쓰인다고 했어

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

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

best model

11.2% top 5 error in ILSVRC 2013
->
7.3% top 5 error

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

네트워크의 특징을 알아보았지

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
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

Note:

Most memory is in early CONV

Most params are in late FC

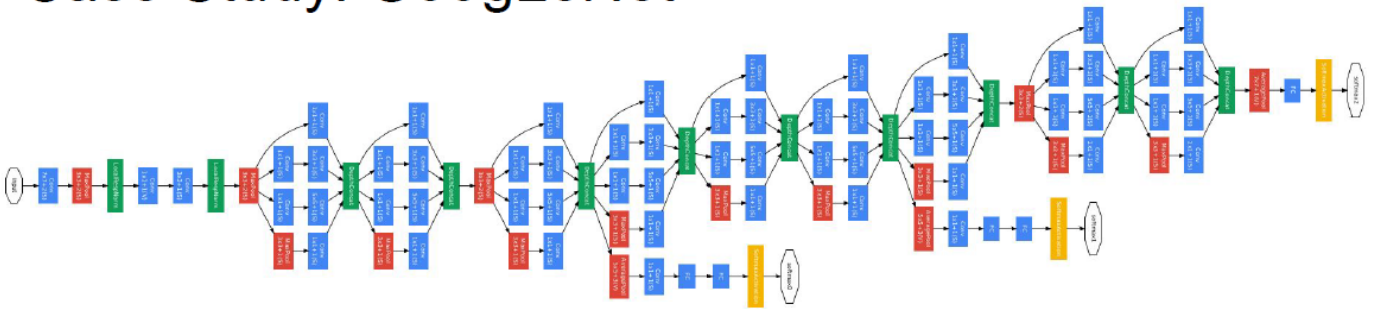
TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

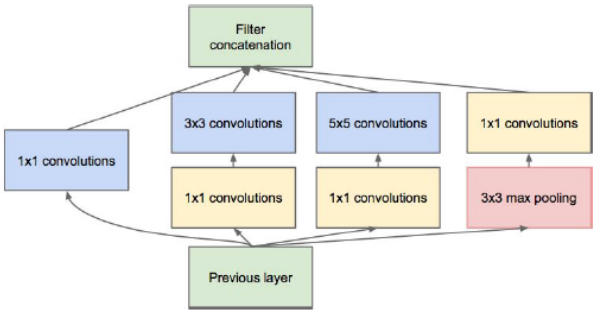
GoogLeNet도 알아봤었어

모두의연구소

Case Study: GoogLeNet

[Szegedy et al., 2014]





Inception module

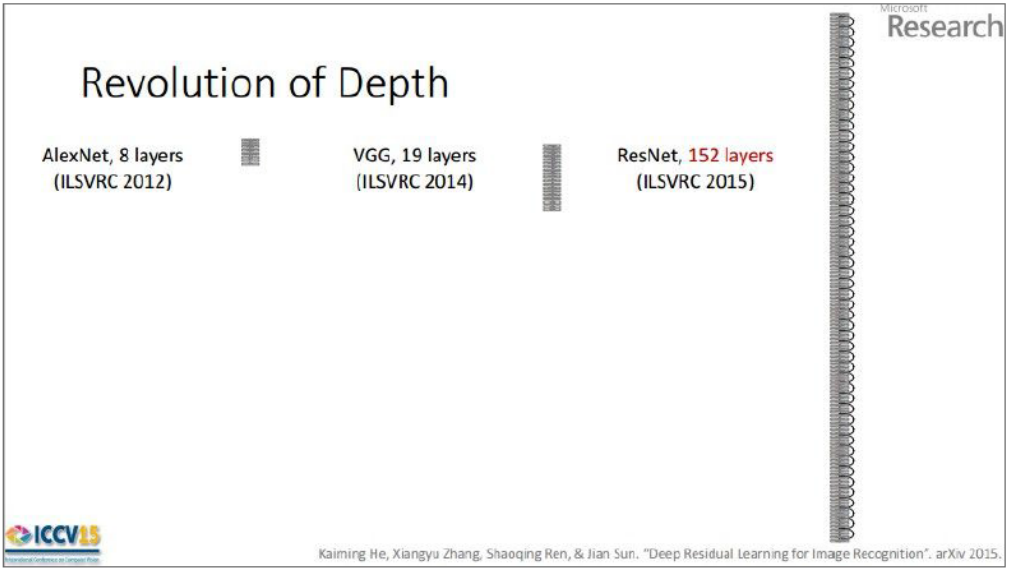
ILSVRC 2014 winner (6.7% top 5 error)

ResNet은 엄청 깊었지

모두의연구소

Case Study: ResNet *[He et al., 2015]*

ILSVRC 2015 winner (3.6% top 5 error)



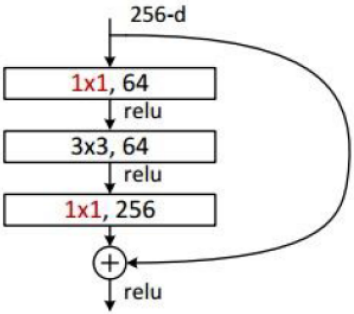
2-3 weeks of training on 8 GPU machine

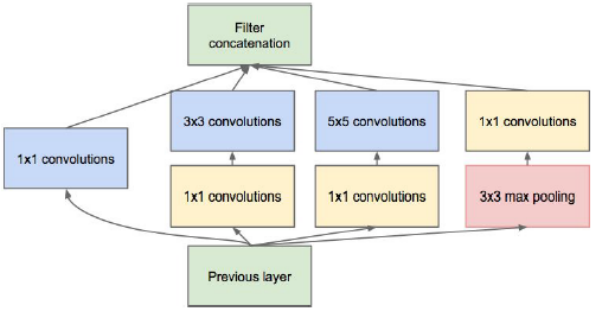
at runtime: faster than a VGGNet! (even though it has 8x more layers)

(slide from Kaiming He's recent presentation)

가장 큰 특징은 Identity mapping 이야

Case Study: ResNet [He et al., 2015]





(this trick is also used in GoogLeNet)

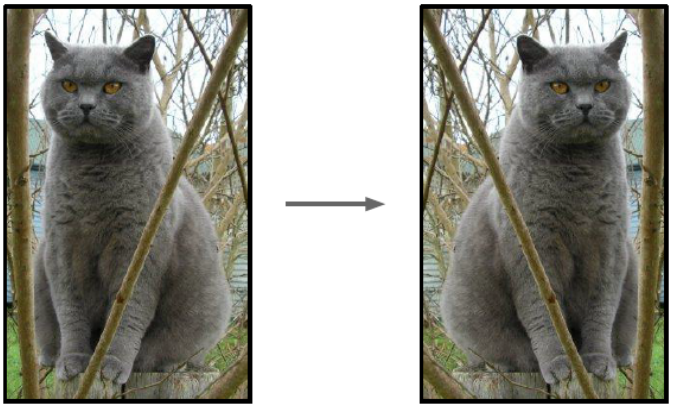
모두의연구소

학습을 위해서 Data augmentation도 했었어

Data Augmentation

모두의연구소

1. Horizontal flips



Data Augmentation

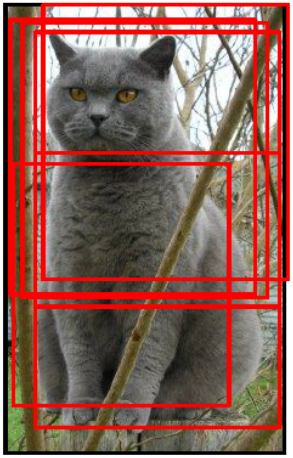
모두의연구소

2. Random crops/scales

Training: sample random crops / scales

ResNet:

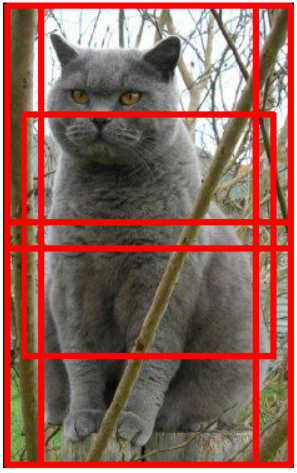
1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch



Data Augmentation

모두의연구소

2. Random crops/scales



Training: sample random crops / scales

ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch

Testing: average a fixed set of crops

ResNet:

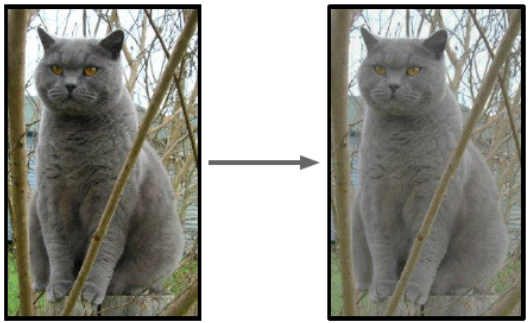
1. Resize image at 5 scales: {224, 256, 384, 480, 640}
2. For each size, use 10 224 x 224 crops: 4 corners + center, + flips

Data Augmentation

모두의연구소

3. Color jitter

Simple:
Randomly jitter contrast



Complex:

1. Apply PCA to all [R, G, B] pixels in training set
2. Sample a “color offset” along principal component directions
3. Add offset to all pixels of a training image

(As seen in *[Krizhevsky et al. 2012]*, ResNet, etc)

Data Augmentation

모두의연구소

4. Get creative!

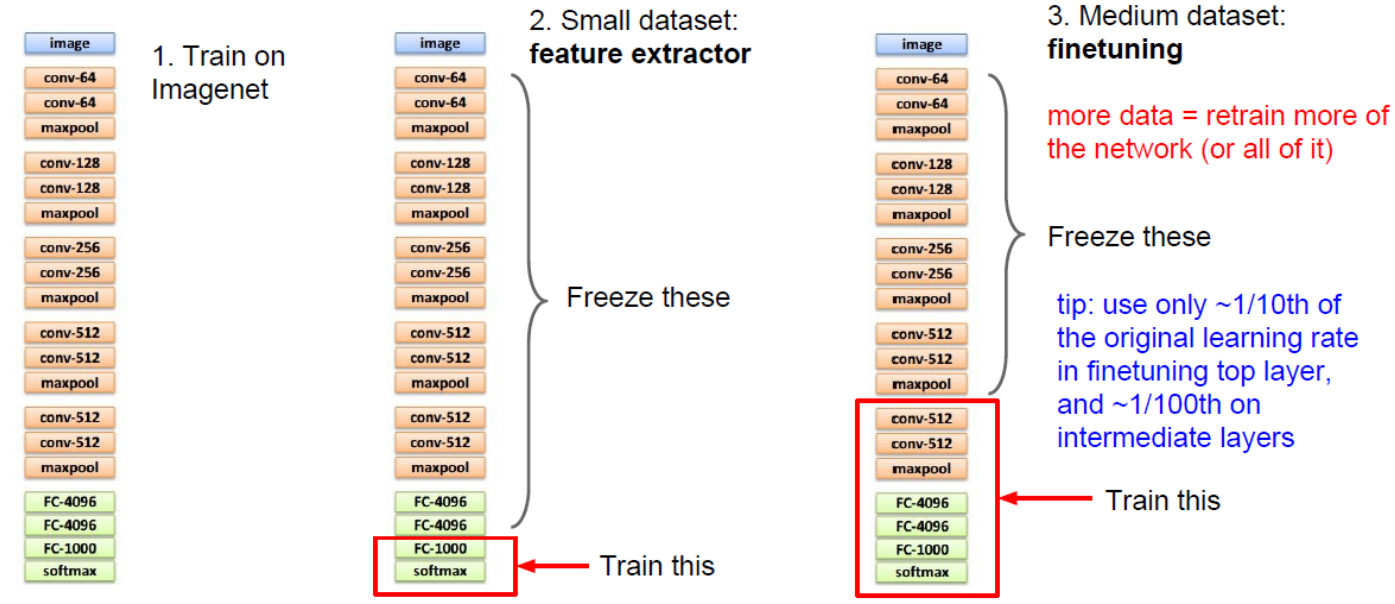
Random mix/combinations of :

- translation
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

Transfer learning도 배웠어

모두의연구소

Transfer Learning with CNNs



모두의연구소

Convolution을 주위깊게 봤었지

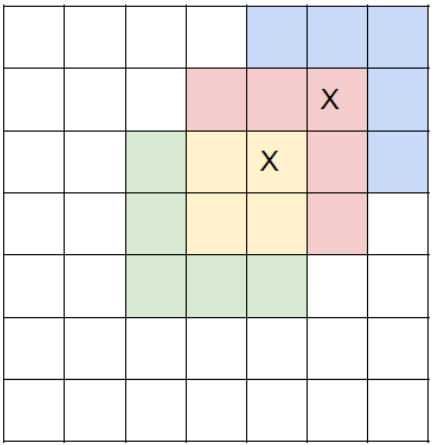
The power of small filters

Question: If we stack **three** 3x3 conv layers, how big of an input region does a neuron in the third layer see?

The power of small filters

Question: If we stack **three** 3x3 conv layers, how big of an input region does a neuron in the third layer see?

Answer: 7 x 7



Three 3 x 3 conv gives similar representational power as a single 7 x 7 convolution

The power of small filters

Suppose input is H x W x C and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W)

one CONV with 7 x 7 filters

Number of weights:
= C x (7 x 7 x C) = **49 C^2**

three CONV with 3 x 3 filters

Number of weights:
= 3 x C x (3 x 3 x C) = **27 C^2**

Fewer parameters, more nonlinearity = GOOD

The power of small filters

Suppose input is H x W x C and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W)

one CONV with 7 x 7 filters

Number of weights:
= C x (7 x 7 x C) = 49 C^2

Number of multiply-adds:
= (H x W x C) x (7 x 7 x C)
= **49 HWC^2**

three CONV with 3 x 3 filters

Number of weights:
= 3 x C x (3 x 3 x C) = 27 C^2

Number of multiply-adds:
= 3 x (H x W x C) x (3 x 3 x C)
= **27 HWC^2**

The power of small filters

Suppose input is H x W x C and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W)

one CONV with 7 x 7 filters

Number of weights:
= C x (7 x 7 x C) = 49 C^2

Number of multiply-adds:
= **49 HWC^2**

three CONV with 3 x 3 filters

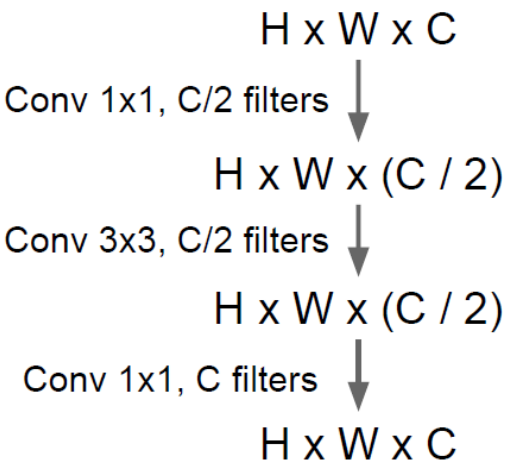
Number of weights:
= 3 x C x (3 x 3 x C) = 27 C^2

Number of multiply-adds:
= **27 HWC^2**

Less compute, more nonlinearity = GOOD

The power of small filters

Why stop at 3 x 3 filters? Why not try 1 x 1?

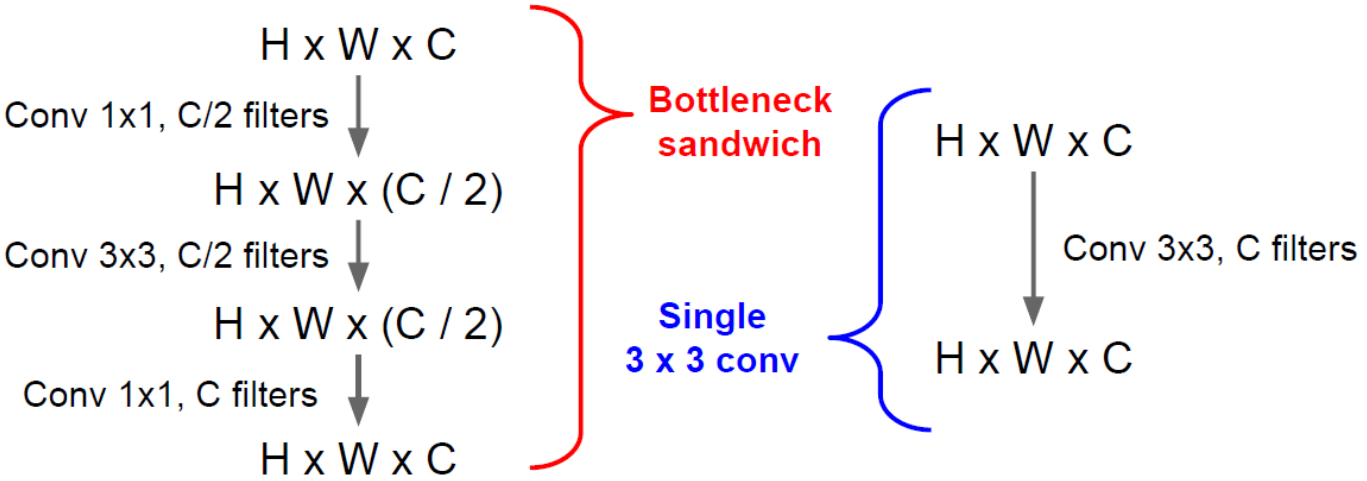


1. "bottleneck" 1 x 1 conv to reduce dimension
2. 3 x 3 conv at reduced dimension
3. Restore dimension with another 1 x 1 conv

[Seen in Lin et al, "Network in Network", GoogLeNet, ResNet]

The power of small filters

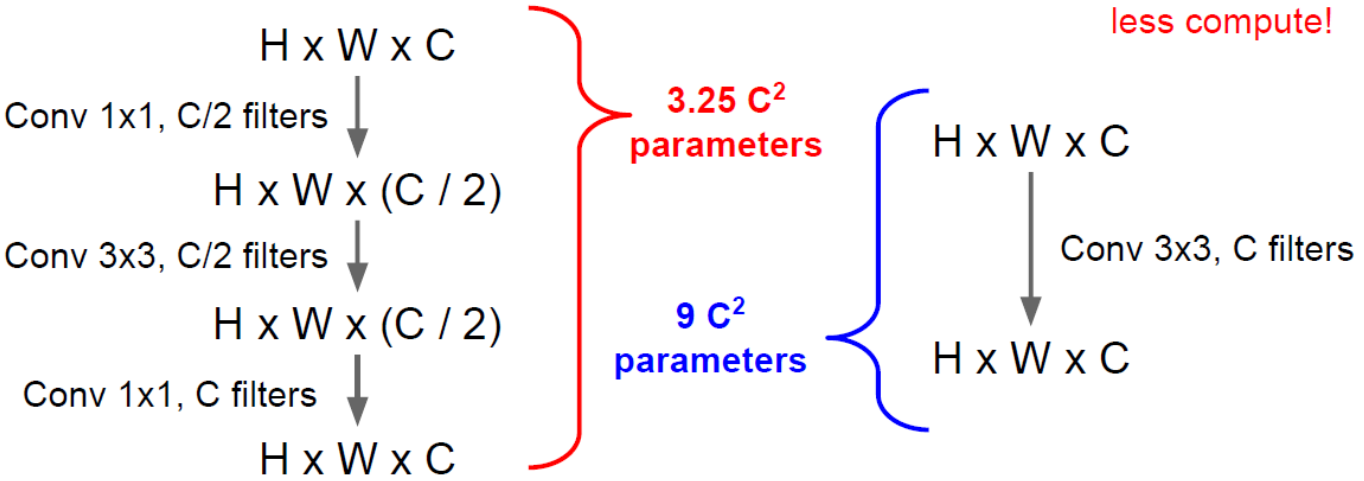
Why stop at 3 x 3 filters? Why not try 1 x 1?



The power of small filters

Why stop at 3 x 3 filters? Why not try 1 x 1?

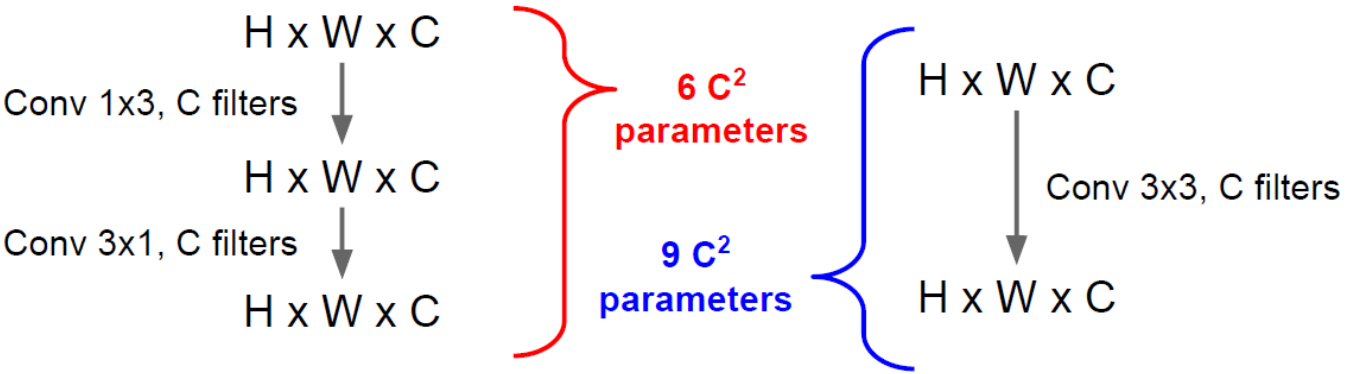
More nonlinearity, fewer params, less compute!



The power of small filters

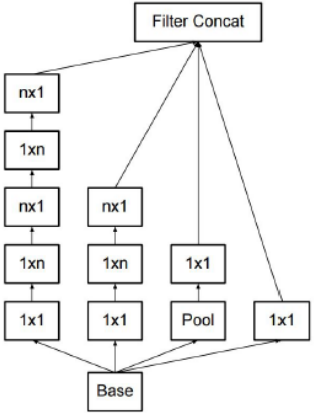
Still using 3 x 3 filters … can we break it up?

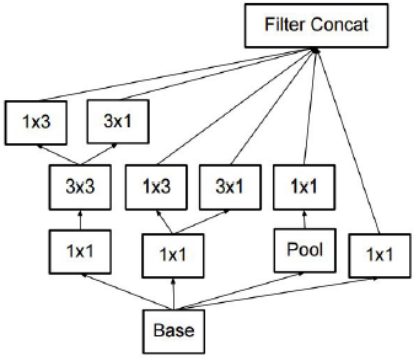
More nonlinearity, fewer params, less compute!

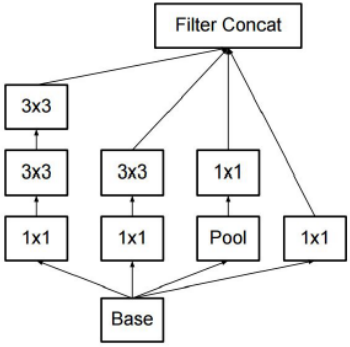


The power of small filters

Latest version of GoogLeNet incorporates all these ideas

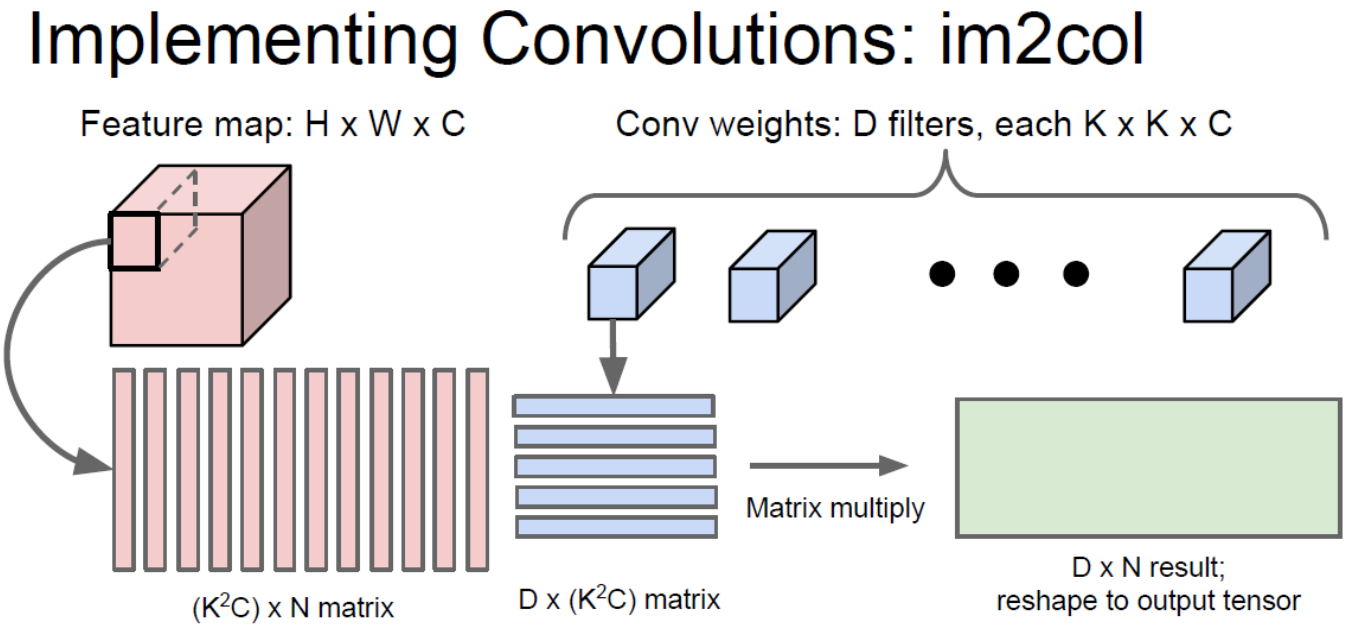






Szegedy et al, “Rethinking the Inception Architecture for Computer Vision”

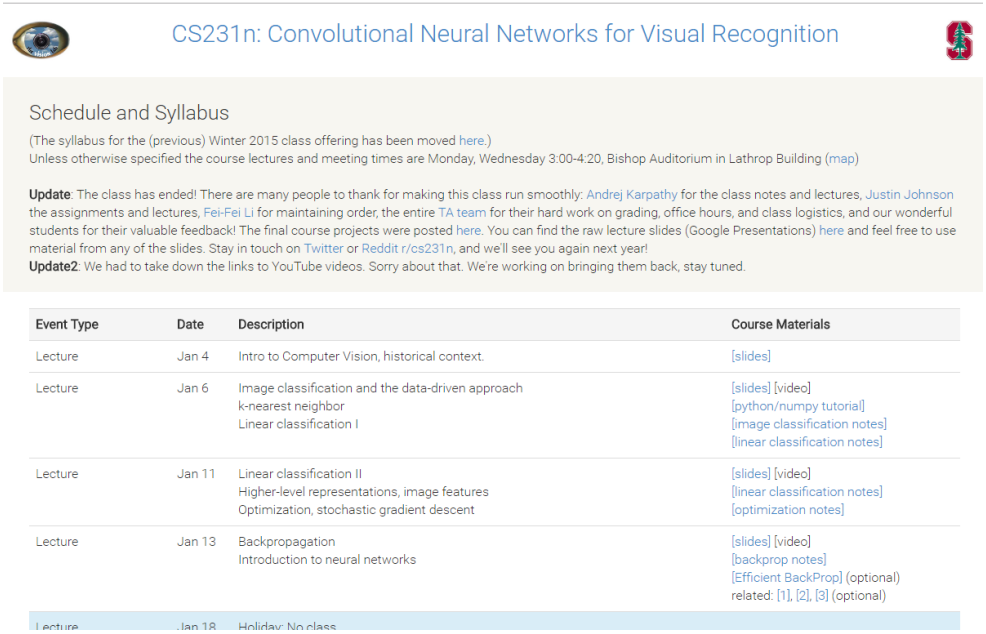
Convolution 연산은 im2col로 최적화 됐었어



잘 모르는 부분은 실제 강의노트 및 설명을 봐야 겠어

모두의연구소

http://vision.stanford.edu/teaching/cs231n/syllabus.html



CS231n: Convolutional Neural Networks for Visual Recognition

Schedule and Syllabus

(The syllabus for the (previous) Winter 2015 class offering has been moved here.)
Unless otherwise specified the course lectures and meeting times are Monday, Wednesday 3:00-4:20, Bishop Auditorium in Lathrop Building (map)

Update: The class has ended! There are many people to thank for making this class run smoothly: Andrej Karpathy for the class notes and lectures, Justin Johnson the assignments and lectures, Fei-Fei Li for maintaining order, the entire TA team for their hard work on grading, office hours, and class logistics, and our wonderful students for their valuable feedback! The final course projects were posted here. You can find the raw lecture slides (Google Presentations) here and feel free to use material from any of the slides. Stay in touch on Twitter or Reddit r/cs231n, and we'll see you again next year!
Update2: We had to take down the links to YouTube videos. Sorry about that. We're working on bringing them back, stay tuned.

Event Type	Date	Description	Course Materials
Lecture	Jan 4	Intro to Computer Vision, historical context.	[slides]
Lecture	Jan 6	Image classification and the data-driven approach k-nearest neighbor Linear classification I	[slides] [video] [python/numpy tutorial] [image classification notes] [linear classification notes]
Lecture	Jan 11	Linear classification II Higher-level representations, image features Optimization, stochastic gradient descent	[slides] [video] [linear classification notes] [optimization notes]
Lecture	Jan 13	Backpropagation Introduction to neural networks	[slides] [video] [backprop notes] [Efficient BackProp] (optional) related: [1], [2], [3] (optional)
Lecture	Jan 18	Holiday: No class	