

# AlphaGo 알고리즘 분석

추형석

2016.05.31

# 목 차

1. 인공지능 개요
2. 사람처럼 학습하는 원동력 – 딥러닝
3. 인공지능 연구 성공사례 – 알파고를 중심으로
4. 결 론

# 1. 인공지능 개요

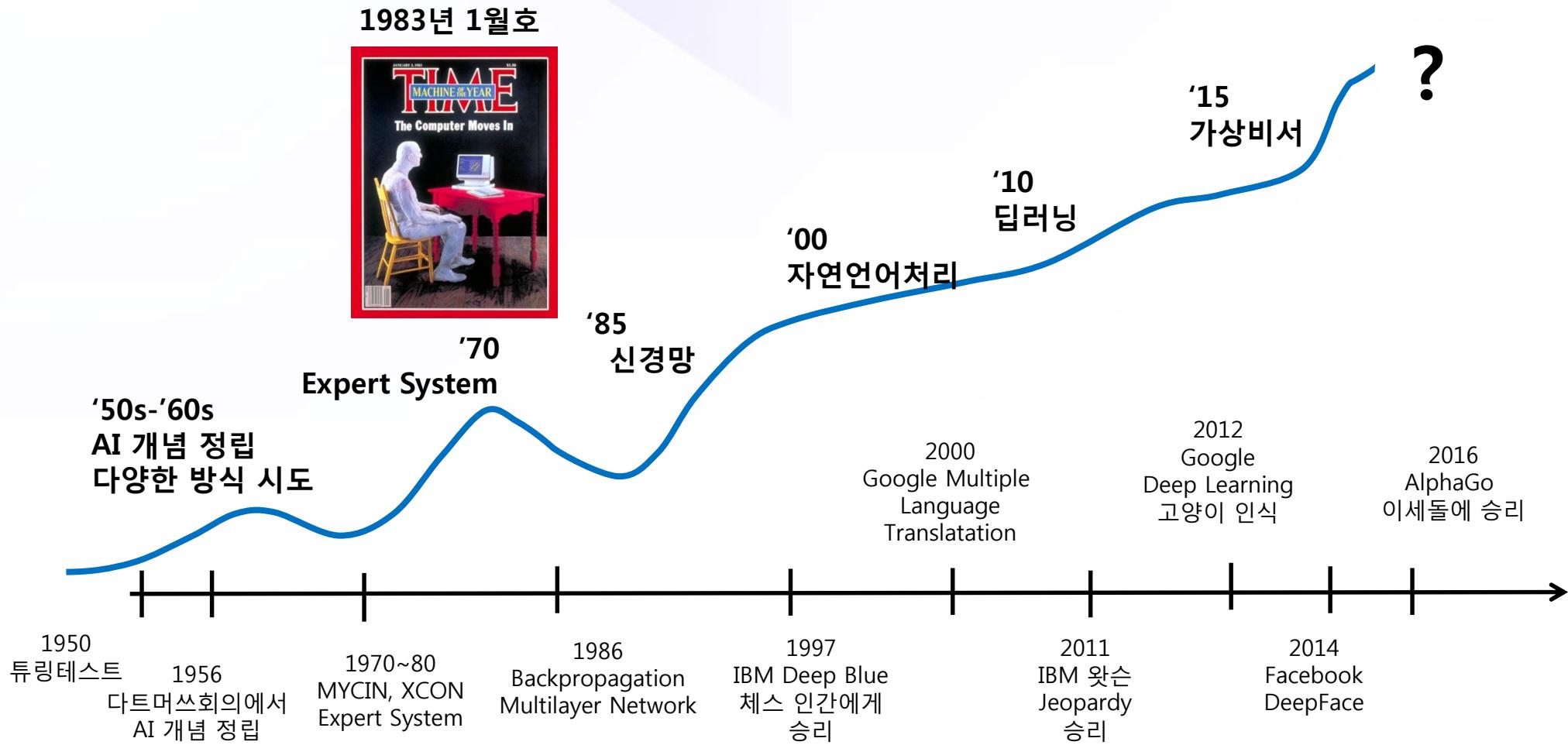
# 인공지능이란 ?

---

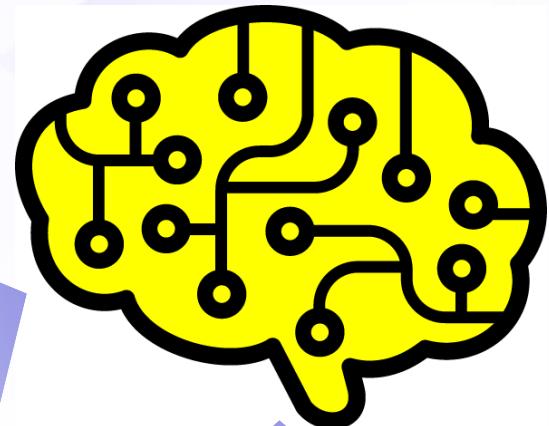
- 지능적 행동을 자동화 하기 위한 컴퓨터 과학의 한 분야  
(Luger & Stubblefield, 1993)
- 현재 사람이 더 잘 하는 일을 컴퓨터가하도록 하는 연구  
(Rich & Knight, 1991)
- 컴퓨터를 조금 더 스마트하게 만들기

현실적인 문제해결의 도구

# 인공지능의 역사는 컴퓨터 발명이래 70년간의 신기술 부침의 역사

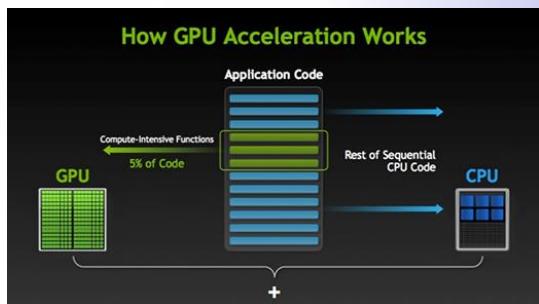


# 인공지능 성공의 원동력



## Computing Power

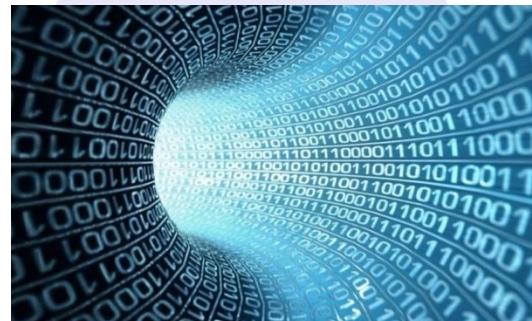
강력한 병렬 및 분산처리 능력



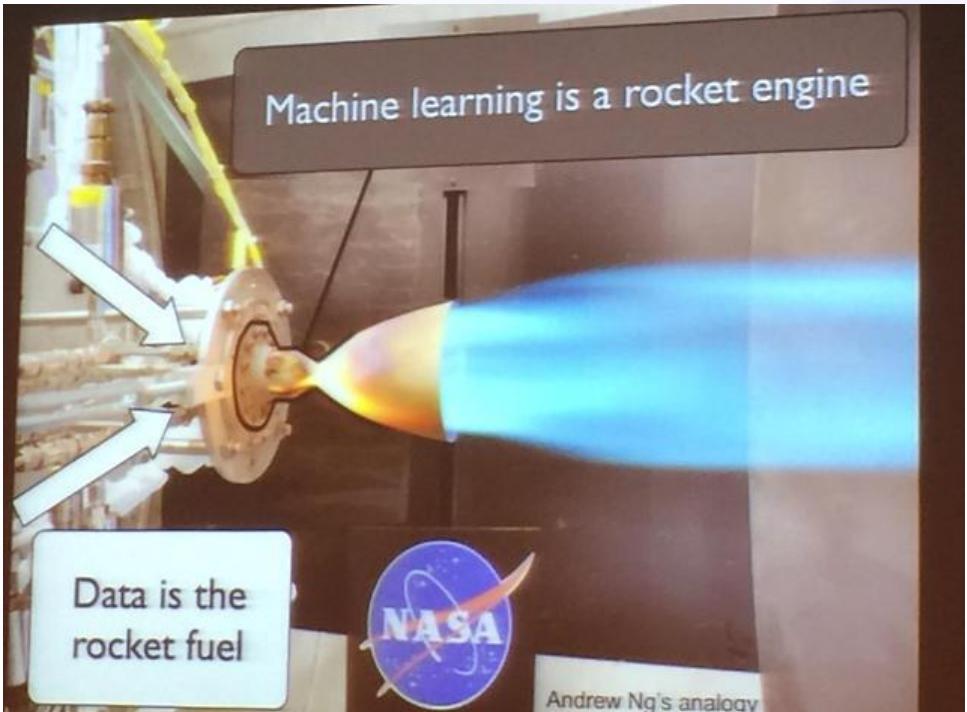
공개소프트웨어  
개방·공유·협업의 성과

## Big Data Power

인터넷, IOT, Sensor 기술을 통한 수집능력



# “차세대 핵심 기술은 머신러닝”



“기계학습은 로켓의 엔진과 같다. 로켓이 날아가려면 엔진에 넣을 연료가 필요한데 이것이 바로 데이터이고, 데이터는 IoT를 이용한 센서에서 얻어진다.”

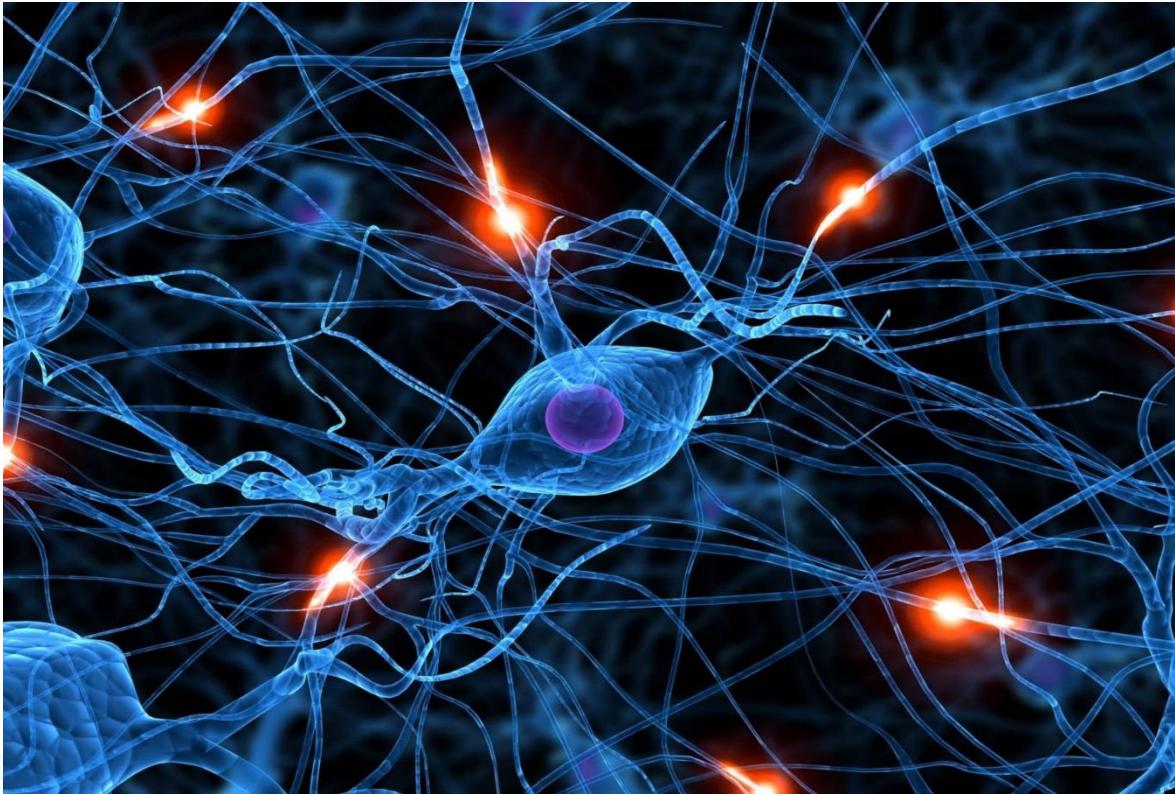
“5년 내에 모든 기업이  
머신러닝을 사용할 것이다”  
- Eric Schmidt

## Open Source Software for Deep Learning

- Google TensorFlow
- Microsoft CNTK, DMTK
- Skymind DL4j
- Baidu WARP-CTC
- Facebook Torch
- ...

## OPEN AI Community

## 2. 사람처럼 학습하는 원동력 - 딥러닝



# Perceptron : 신경세포의 수학적 모형

- 신경세포(뉴런)을 퍼셉트론으로 추상화 (1957, Rosenblatt)

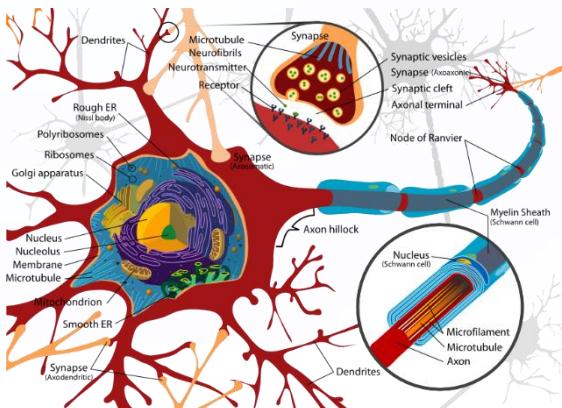
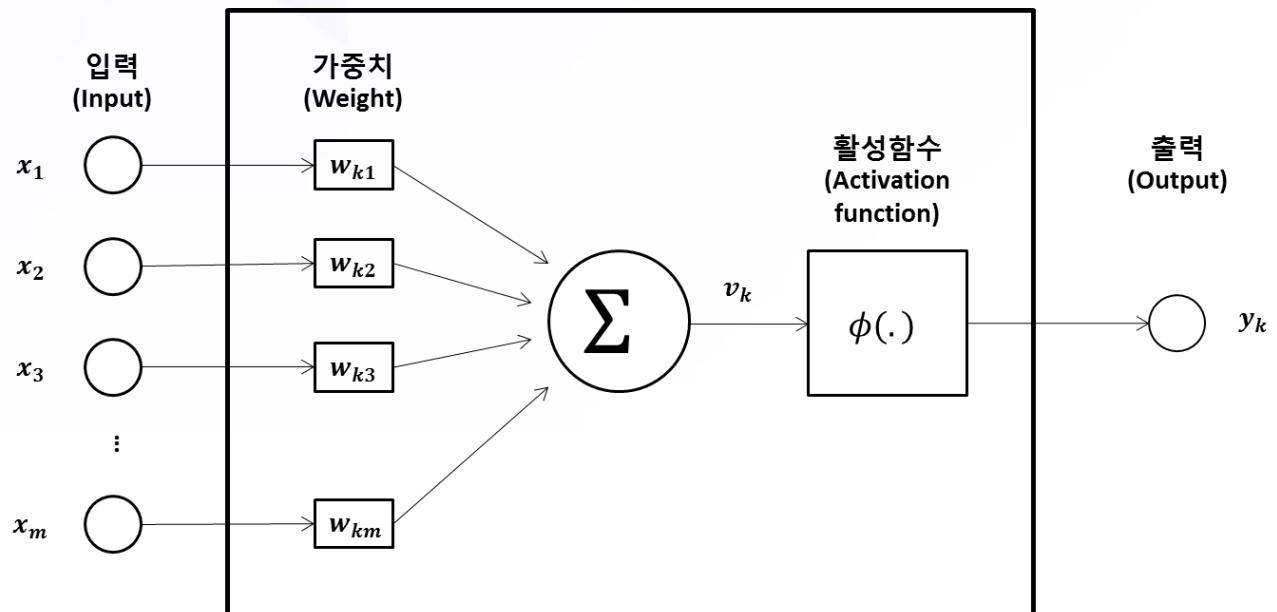


Diagram of Neuron  
<http://en.wikipedia.org/wiki/Neuron>

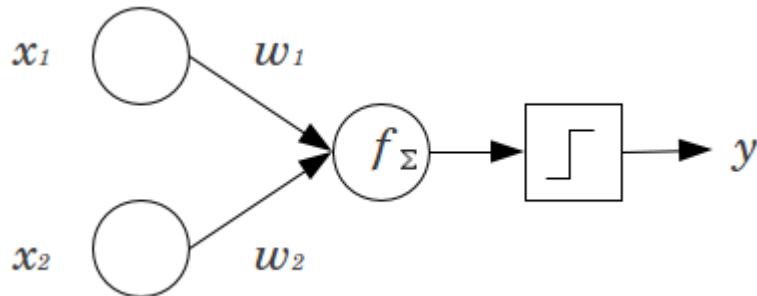


신경 세포의 모델링 : 자극(입력)의 합이 역치(threshold)를 넘어서면 정보를 전파

$$v_k = \sum_{j=1}^m w_{kj} x_j, \quad y_k = \phi(v_k), \quad \phi(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid 활성함수  
(역치를 미분 가능한 함수로 표현)

# Perceptron의 한계



- 간단한 가중치 Update Rule

$$\cdot w_i \leftarrow w_i + \Delta w_i$$

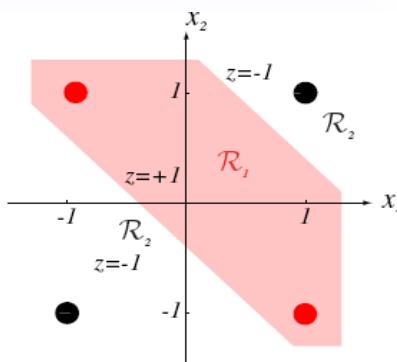
$$\Delta w_i = \eta(D - Y)x_i$$

Learning rate = 1

Desired output

Actual output

Input



XOR 문제

- 학습 데이터를 직선으로 구분할 수 있으면 항상 해에 수렴

- 선형 분류기의 한계

- XOR 문제는 해결 못함

# 복층구조신경망 (Multi-Layer Perceptron)

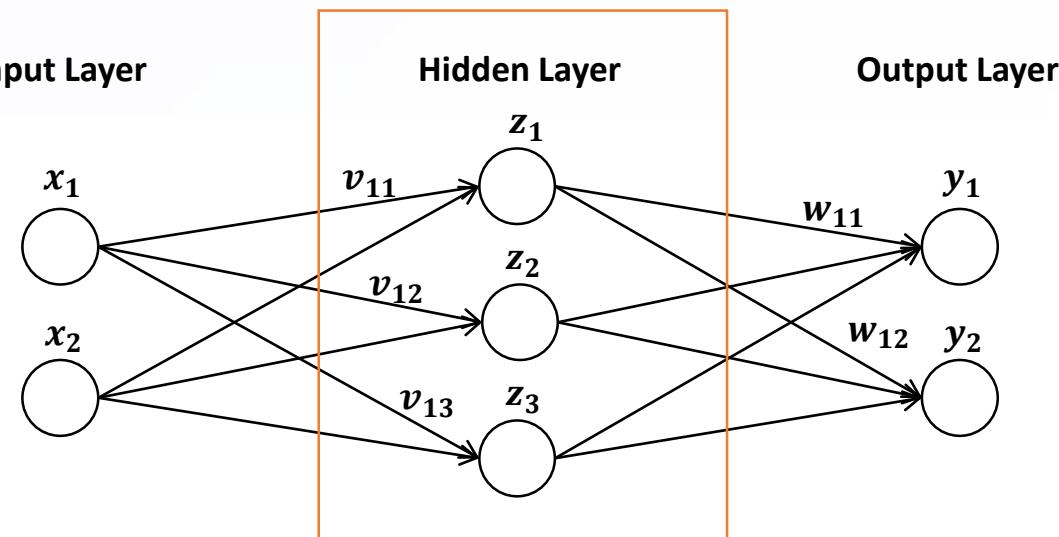
- 은닉노드의 층을 가진 구조 (1985, 인공신경망)

은닉노드(Hidden Node)란

- 입력단(저층)에서 오는 값을 처리하여 출력단(고층)으로 전파하는 역할

- 은닉노드 수를 늘려서 복잡한 함수 표현 가능

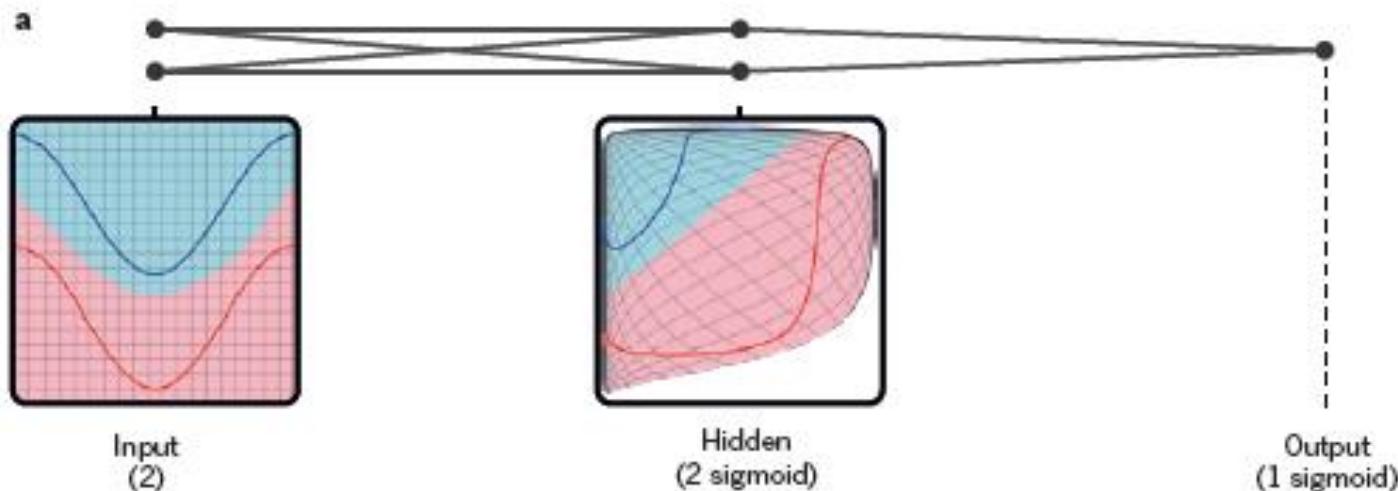
- XOR 문제 해결



$$z_j = \sigma \left( \sum_{i=1}^2 v_{ij} x_i + b_j \right),$$
$$y_k = \sigma \left( \sum_{j=1}^3 w_{jk} z_j + b_k \right)$$

# MLP의 표현력

- 비선형함수의 형태로 분류
  - Domain을 왜곡해서 선형분류가 가능



# Some Theorems

---

- **Universal Approximation Theorem**
  - MLP에서 은닉층 하나로도 모든 연속함수를 표현할 수 있다는 이론
  - 실제 구현상에서의 가능성과는 별개
- **은닉층의 제한 (+은닉층의 노드 수 제한)**
  - 학습 데이터의 개수가 미지수(가중치)의 개수보다 많아야 함

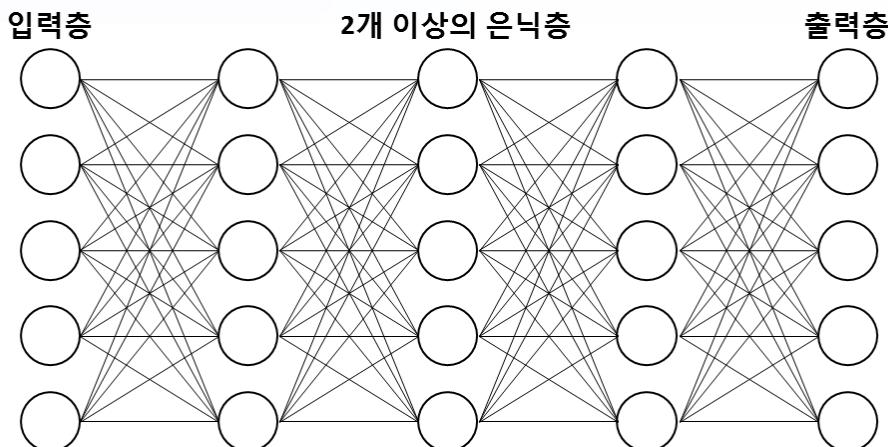
# MLP의 한계

---

- 미분 값이 0으로 수렴하는 현상
  - 오차역전파법(back propagation error method)에서 발생
  - 신경망의 오차를 가중치로 미분하는 과정에서 연쇄법칙(chain rule)에 따라 활성함수의 미분이 발생 → 활성함수의 양극에서 미분 값이 0으로 수렴
- 정답이 있는(labeled) 데이터의 부족
  - 사람이 직접 데이터를 수집해야 하는 어려움
  - 정답이 없는(un-labeled) 데이터가 대부분
- 과적합(over-fitting) 문제
  - 딥러닝의 경우, 데이터보다 가중치의 개수가 더 많을 수 있음
- 지역적인 최소값(local minima)에 갇히는 문제

# 딥러닝의 등장

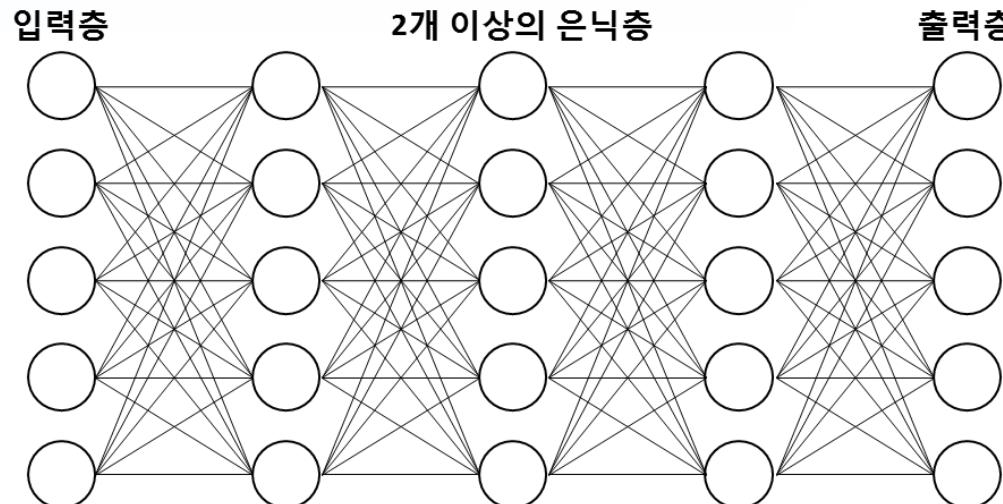
- 2층 이상의 은닉층 도입 → 딥러닝
  - 자율학습(unsupervised learning)으로 은닉층을 층 별로 학습
    - 오류역전파 알고리즘의 개선
  - 적은 데이터로도 과적합 문제 회피 가능
- 인공신경망의 한계점 해결
  1. 활성함수의 미분이 0으로 수렴
    - Rectified Linear Unit (ReLU) 활성함수 도입
  2. 데이터의 부족
    - 빅데이터의 출현, 자율학습
  3. 과적합 문제
    - Regularization 기법 개발 : dropout 등
  4. 극소값 문제
    - 자율학습(unsupervised learning) 활용



복잡한 문제해결의 도구로 진화

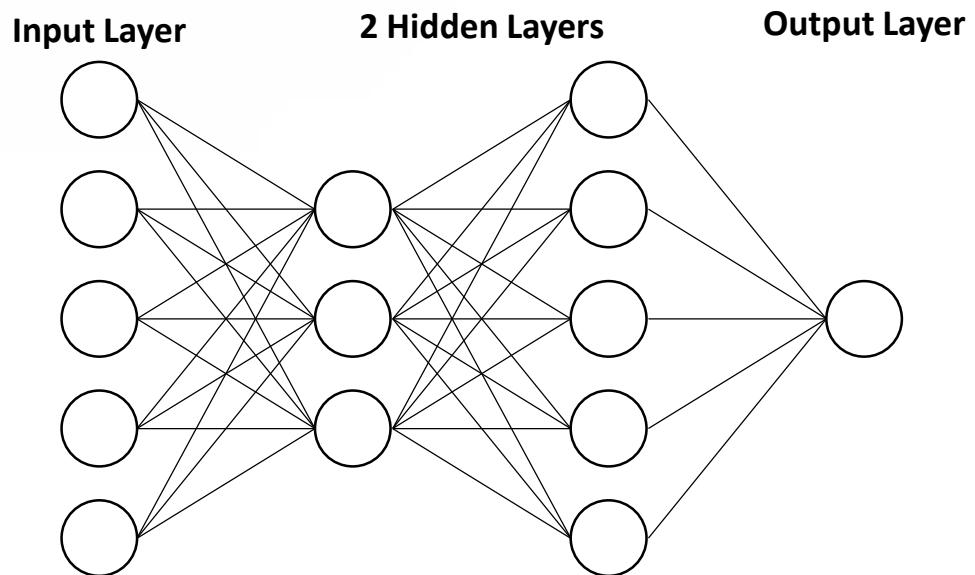
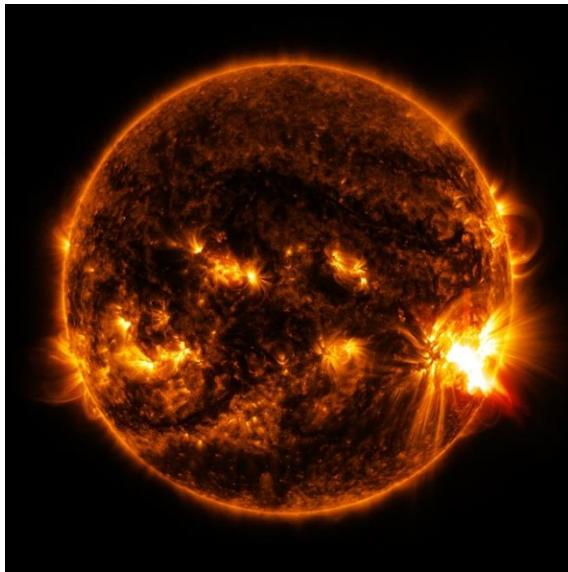
# 심층신경망 (Deep Neural Network, DNN)

- **심층신경망의 핵심은 자율학습 (unsupervised learning)**
  - MLP와 DNN은 신경망 형태는 유사하나, 학습방법에 차이가 있음
    - MLP는 가중치 초기값이 랜덤으로 주어짐 → 초기값에 따라 결과가 매우 다름
    - DNN은 가중치 초기값을 자율학습의 결과값으로 시작  
→ 이 후는 MLP와 같은 오류역전파법 활용
  - 따라서 상대적으로 적은 데이터로도 신경망을 학습시킬 수 있음
  - 층별로 학습하는 방법은 일반적으로 제한된 볼츠만 머신(Restricted Boltzmann machine) 사용



# 딥러닝 연구사례 – 태양 플레이어 예측

- 실시간 태양의 플레이어 예측을 위한 인공신경망 모델
- 데이터
  - 태양 흑점 분류, 흑점의 면적, 24h전의 태양 플레이어, 흑점의 자기장
  - 관측시기 : 1996년 ~ 2013년 (약 10만 건)
- 기존연구대비 약 20% 예측성능 향상

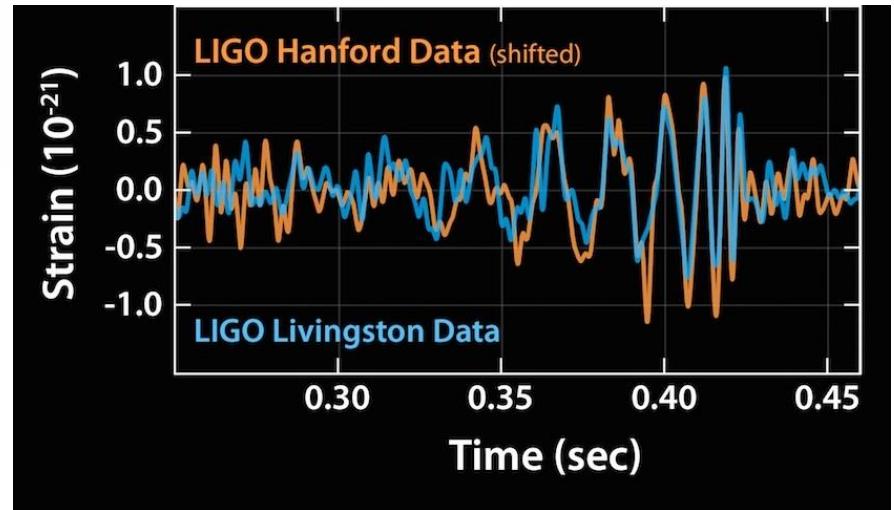


# 딥러닝 연구사례 – 중력파 신호 분류

- 중력파 관측을 위한 신호분석에서 잡음(Glitch)을 분류
- 중력파의 신호가 매우 미약한 반면 음파, 지진파 등 상대적으로 신호의 강도가 세기 때문에 이를 검출해 내는 것이 목적
- 64초 기준 약 100MB 신호데이터



LIGO Hanford Observatory



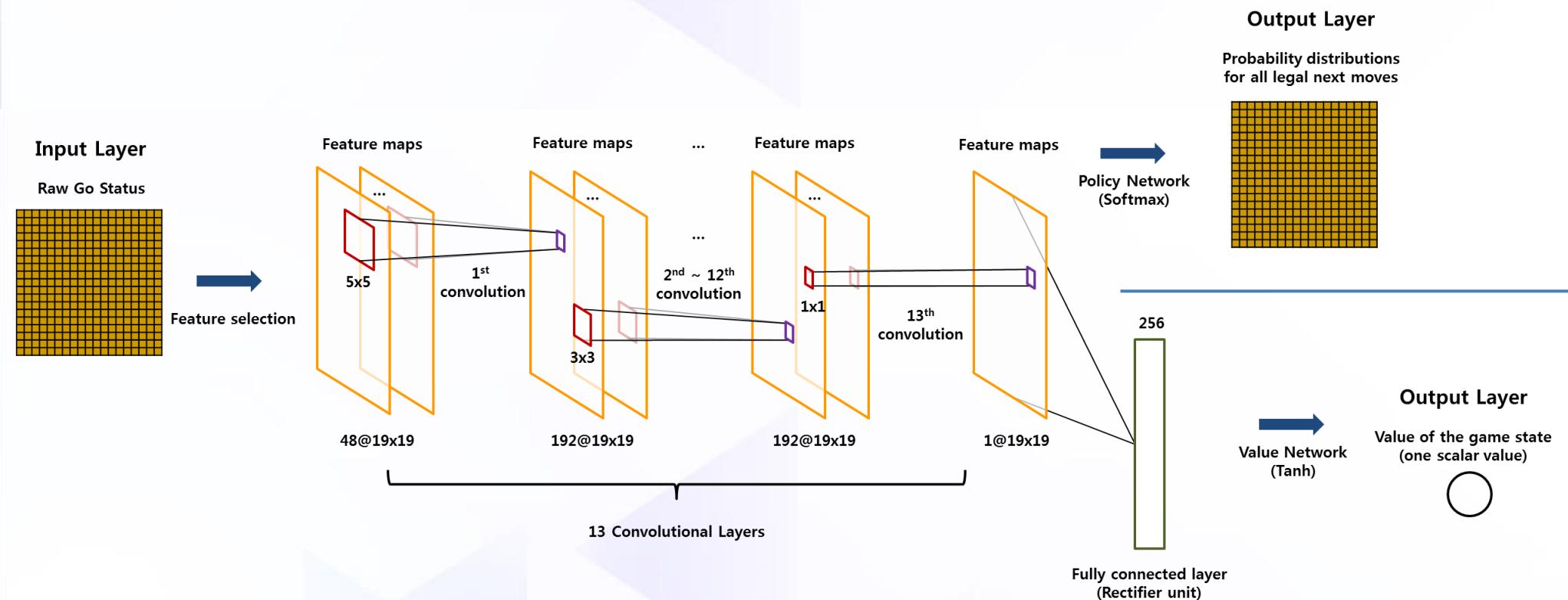
중력파 검출 성공 (2016.02.11)

# 딥러닝의 구현의 어려움

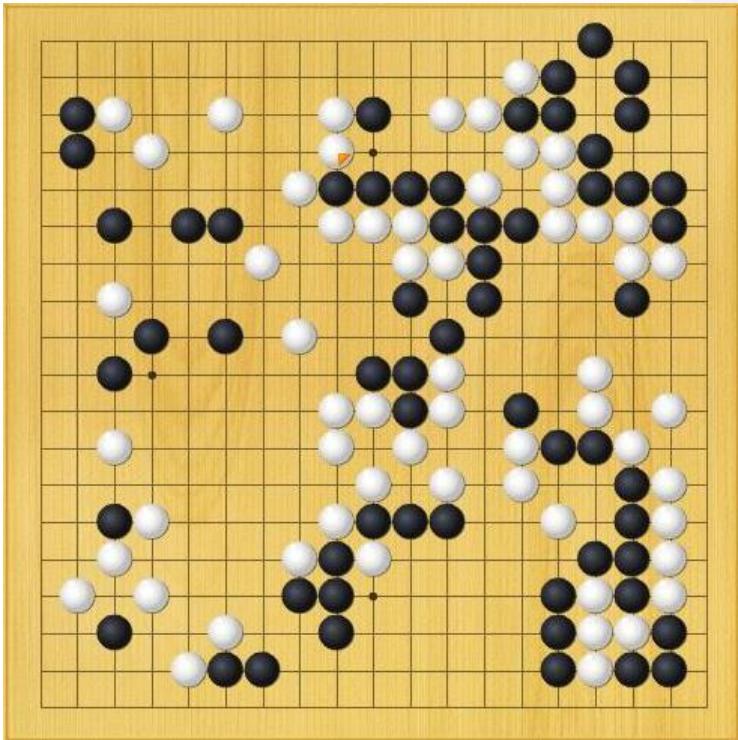
---

- 데이터 확보
  - 데이터 양이 많을 수록 학습효율이 증대됨
  - 데이터가 적으면 일반화 능력이 매우 떨어짐
- 신경망 구성에 대한 Knowhow가 요구됨
  - 은닉층의 개수? 은닉층의 노드 개수? → 이론적으로 정립된 것이 없음
    - 층이 많을 수록 더 정확한 결과를 기대할 수 있으나 많은 데이터와 컴퓨팅 파워가 필요
  - 입력값의 선택과 전처리, 활성함수의 선택
  - 경험적으로 먼저 해보고 결정 → 수많은 테스트가 필요
- 고층 신경망의 학습은 고성능컴퓨터 없이는 불가능
  - 2015 ImageNet 경진대회에서 1등을 차지한 Microsoft는 150층을 사용
  - 슈퍼컴퓨터급 계산자원이 확보되지 않으면 불가능

### 3. 인공지능 연구 성공사례 - 알파고를 중심으로



# 인공지능 관점에서의 바둑 문제



?

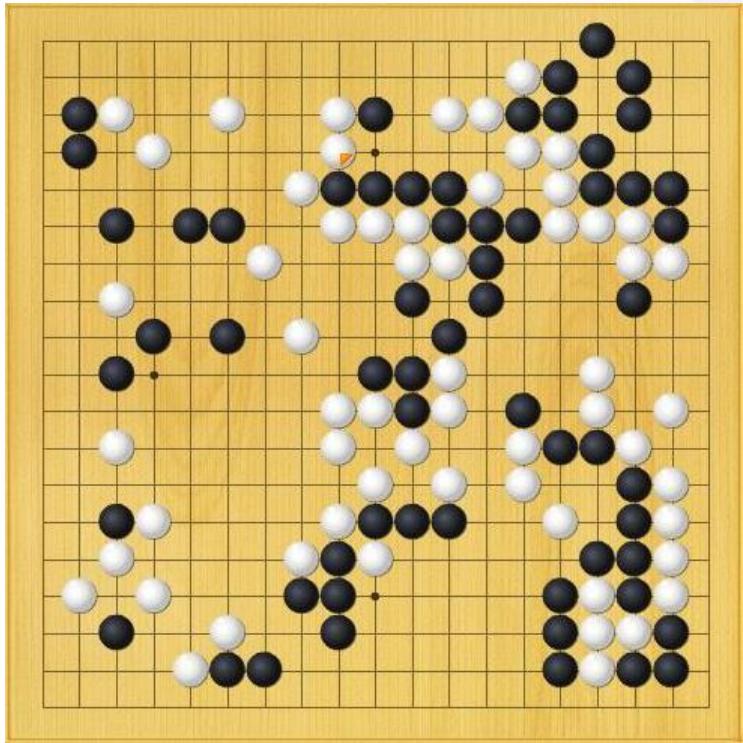
다음 수를 어디에 둘 것인가

정답이 있다면 어떻게 구할 수 있는가

박정환 9단 vs 이창호 9단  
제 7회 응씨배 준결승 제 2국  
2012.09.25

Source : <http://www.samsamstory.com/1056>

# 바둑이 어려운 이유 (1/3)



박정환 9단 vs 이창호 9단  
제 7회 응씨배 준결승 제 2국  
2012.09.25

Source : <http://www.samsamstory.com/1056>

흑과 백중 누가 유리한가?

흑의 승률은? 백의 승률은?

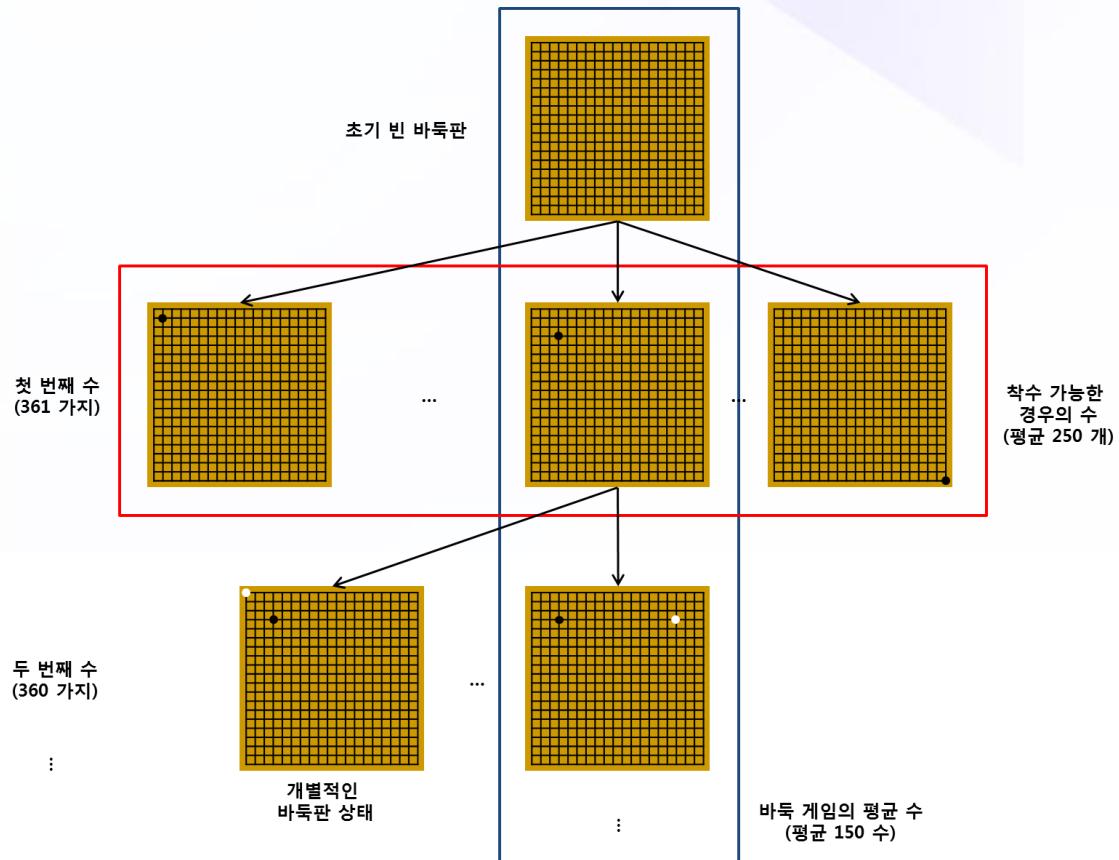
왼쪽 바둑판 상태에서 착수할 수 있는  
**모든 수(241수)를 다 계산하면 정확하게**  
흑과 백의 승률을 알 수 있음

산술적으로 **241!**의 대국이 존재  
 $241! \approx 9.803 \times 10^{471}$

1개의 대국을 계산하는데 1초가 걸린다면  
모두 계산하는데  **$3 \times 10^{464}$  년**이 소요

\* 우주의 나이 :  **$1.38 \times 10^{10}$  년!**

# 바둑이 어려운 이유 (2/3)



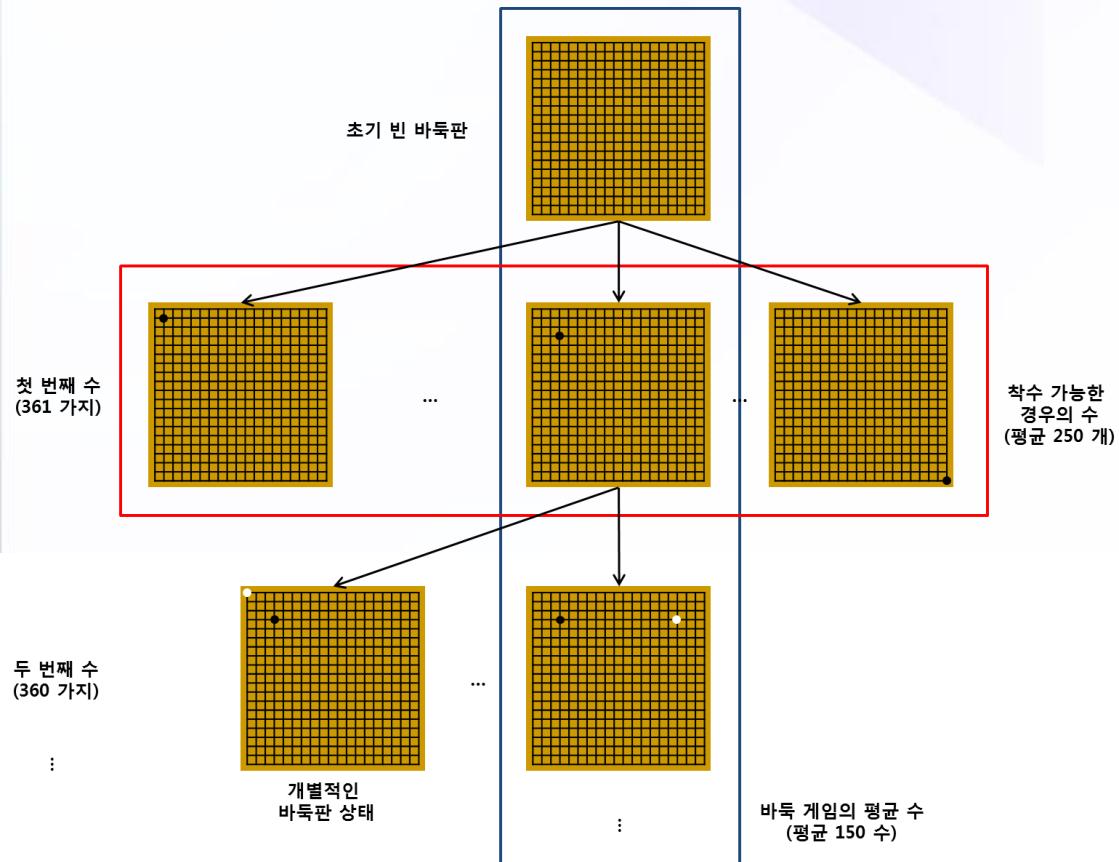
규칙을 무시하면 **361!** 의 대국이 존재  
( $361! \approx 1.438 \times 10^{768}$ )

바둑의 규칙을 적용하면  
착수 가능한 경우의 수 : 평균 250  
바둑 게임의 수 : 평균 150  
 $250^{150}$ 의 대국이 존재  
( $250^{150} \approx 10^{360}$ )

단순히 큰 것이 아니라  
여전히 엄청난 수

경로 1개를 계산하는데 연산이 1개라면  
세계에서 가장 빠른 슈퍼컴퓨터로도  
**10<sup>300</sup> 년** 소요

# 바둑이 어려운 이유 (3/3)



바둑 게임과정을 트리로 표현

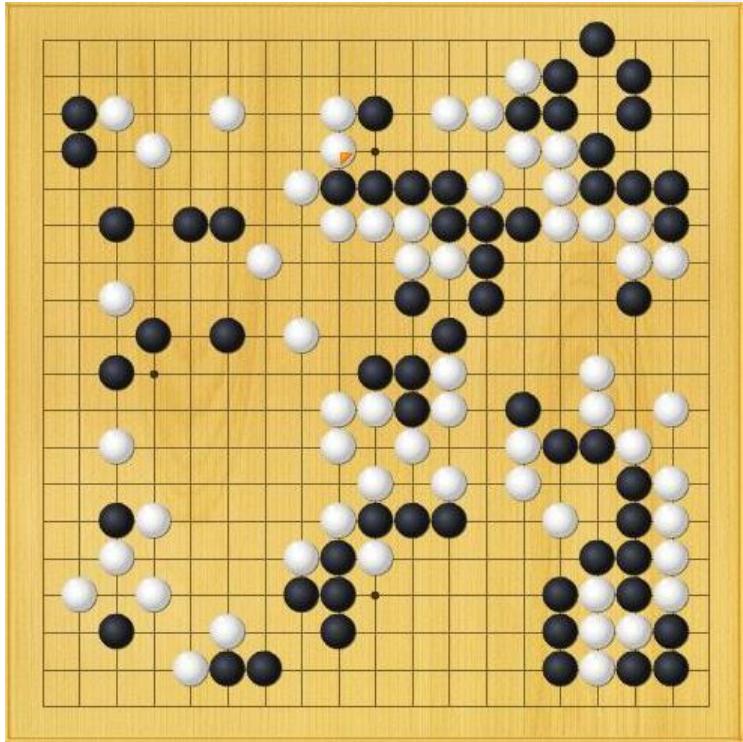
모든 트리를 탐색하고 각 노드(바둑판)  
의 승률을 계산할 수 있으면 바둑을 정복  
할 수 있음

$250^{150}$  상태를 다 저장한다면?  
바둑판 하나를 1bit이라고 가정해도

$8.796 \times 10^{346}$  테라바이트

\* 우주의 원자수 : 약  $10^8$  개!

# AlphaGo의 전략



박정환 9단 vs 이창호 9단  
제 7회 응씨배 준결승 제 2국  
2012.09.25

Source : <http://www.samsamstory.com/1056>

다음 수를 어디에 둘 것인가?

프로기사들이 가장 높은 확률로  
착수하는 지점을 학습

KGS(온라인 바둑서버)에서 6~9단 기보  
16만 개를 학습함  
(GPU 50개 사용, 3주 소요)

정확히 16만개 기보에서 추출한 2940만 개 위치에서  
2840만 개 학습, 100만 개 테스트  
(약 57.1%의 정확도)

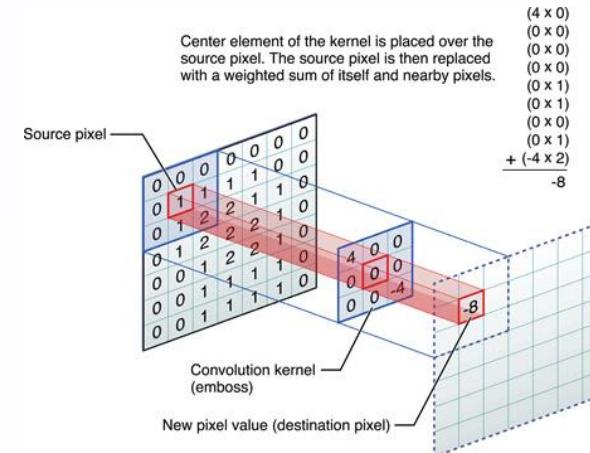
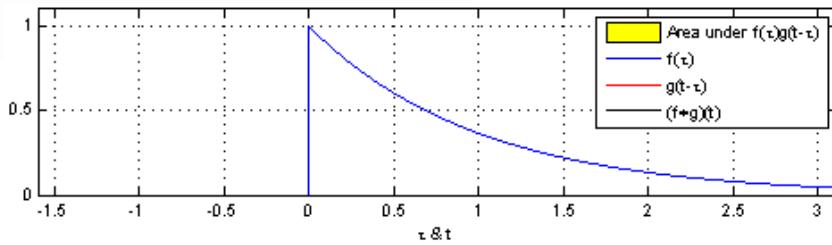
프로바둑기사가 1년에 둘 수 있는 대국을  
1,000회라고 한다면 160년이 소요

순식간에 바둑의 정수를 학습함!

# 컨볼루션 신경망 (Convolutional Neural Network, CNN)

- 이미지 분석에 특화된 딥러닝 기법
  - ‘컨볼루션 필터’를 학습
- 컨볼루션 이란?

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

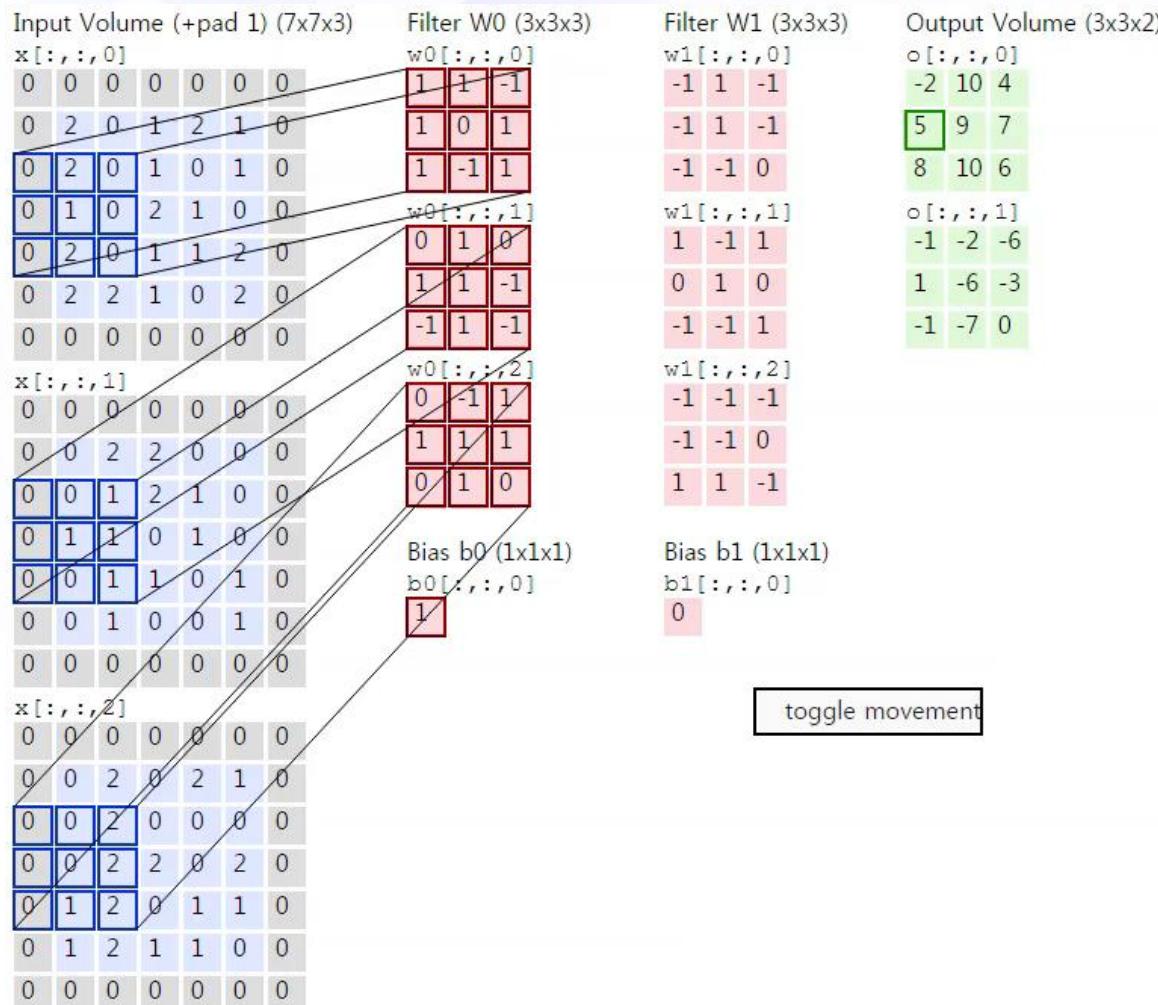


Source : iOS Developer Library – vImage Programming Guide  
<https://developer.apple.com/library/ios/documentation/Performance/Conceptual/vImage/ConvolutionOperations/ConvolutionOperations.html>

Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

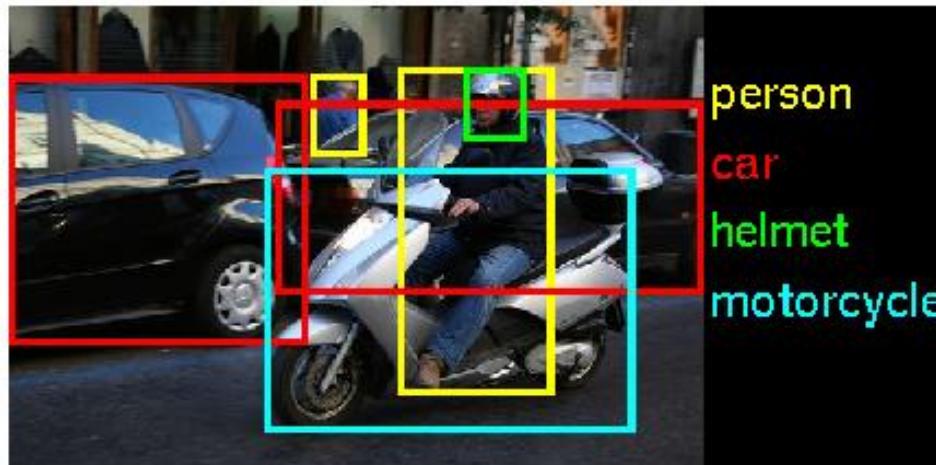
Source: [https://en.wikipedia.org/wiki/Kernel\\_\(image\\_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))

# 컨볼루션의 과정



# CNN 성공사례 - ImageNet 경진대회

- **ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)**
  - 연간 개최되는 이미지 인식 경진대회로, 이미지나 비디오에서 객체의 인식과 위치 정보를 판별하는 것이 문제
  - 객체는 약 200여 종, 위치는 약 1000여 종류
  - ILSVRC 2015 최종결과 (1위, 마이크로소프트, 150층 Deep Residual Learning)
    - 객체 인식 오류 : 9.02%
    - 객체 위치정보 오류 : 3.56%

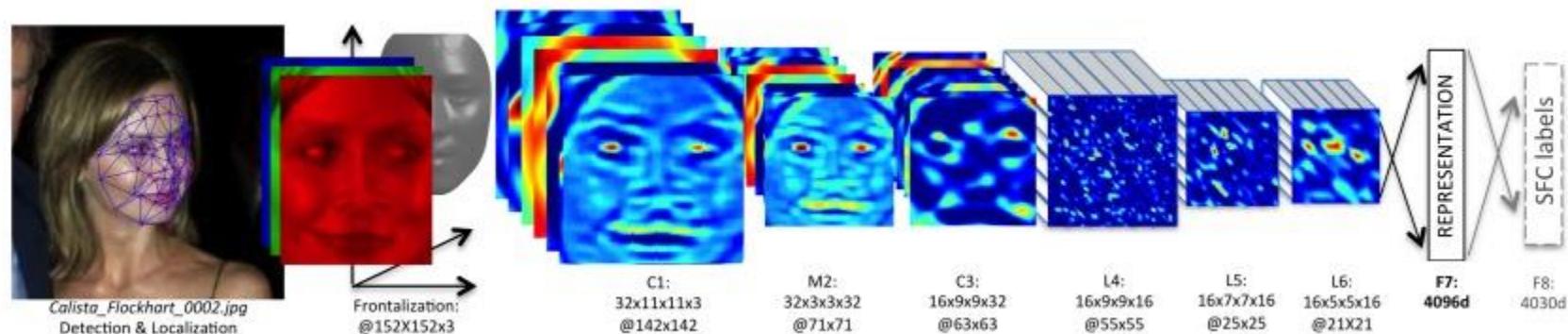


이미지 인식의 예시

# CNN 성공사례 – DeepFace

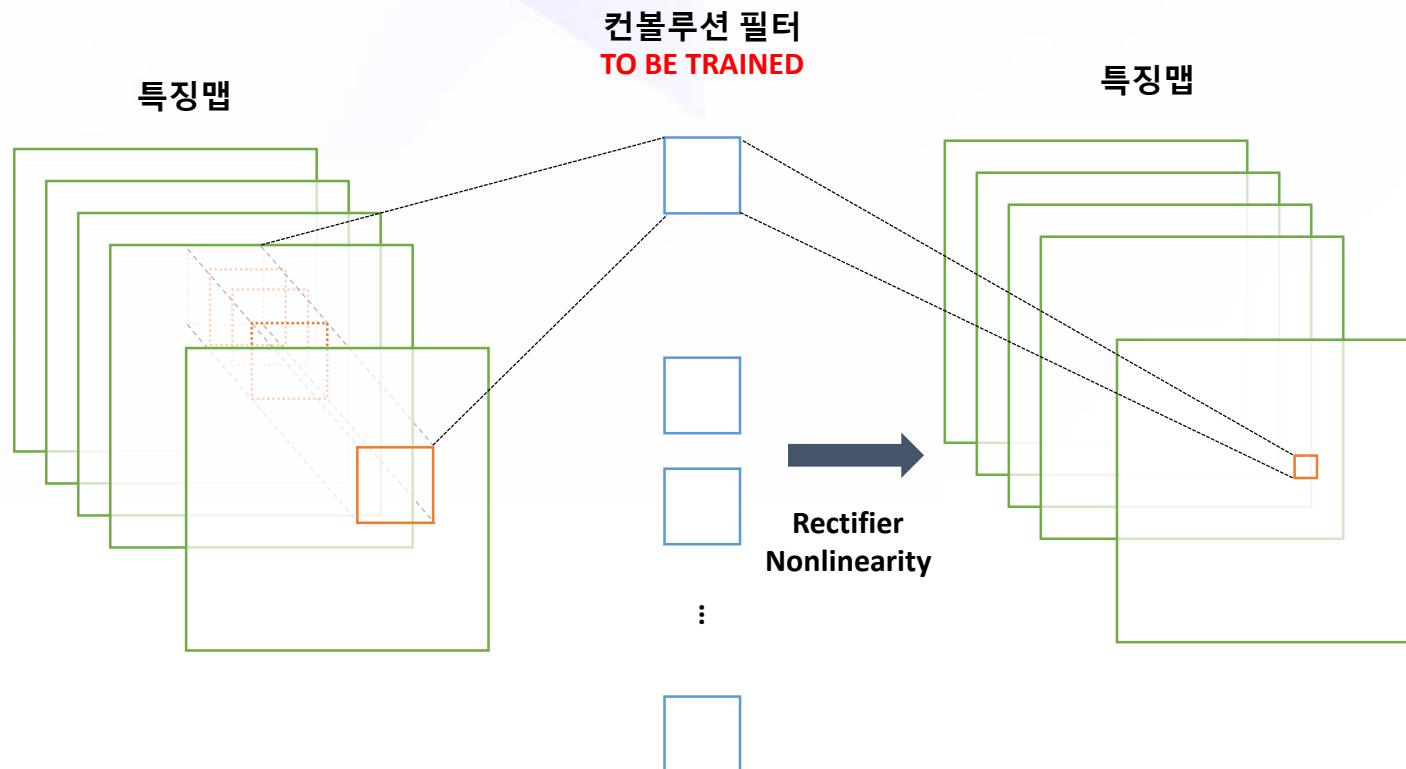
- 페이스북 – 딥페이스 (DeepFace)

- 딥러닝을 활용한 페이스북의 얼굴인식 시스템
- 약 400만 장의 이미지를 사용
- 9층으로 이루어진 심층신경망으로 Detect → Align → Represent → Classify 구현
  - 약 97%의 정확도로 얼굴 인식에 성공



DeepFace 인공신경망 구성

# 컨볼루션 신경망 – 컨볼루션 층



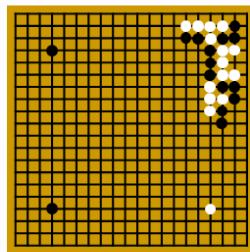
# AlphaGo의 특징맵 (48개)

- 특징맵에 따라 신경망 성능이 좌우됨

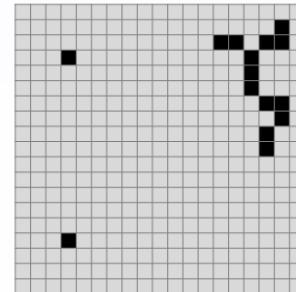
## 48 Feature maps

Feature maps는 이진수로 표현되며 검정색이 1이고 회색이 0을 나타냄

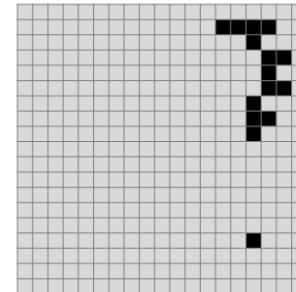
AlphaGo vs. Fan Hui  
두 번째 경기 30수



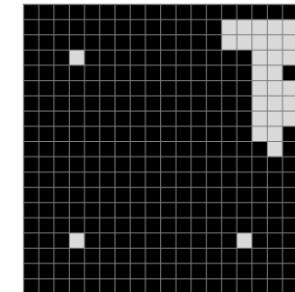
1. 흑 둘



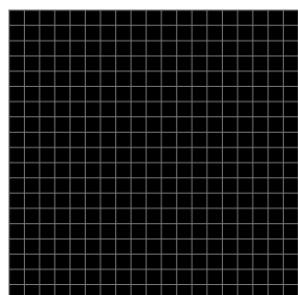
2. 백 둘



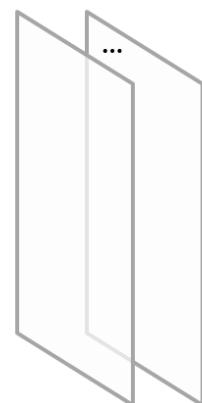
3. 빈 칸



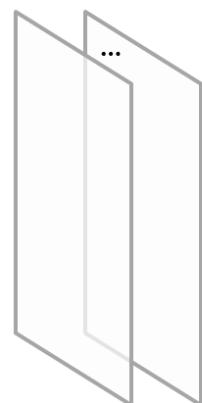
4. 상수 1



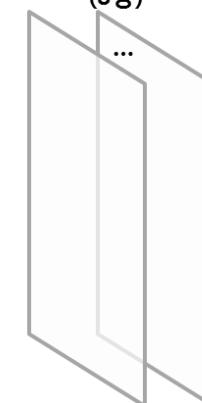
5 ~ 12. 꼬부림 (8장)



13 ~ 20. 활로 (8장)

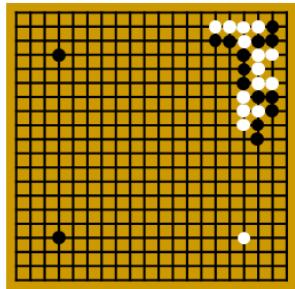


21 ~ 28. 상대방의 단수 (8장)

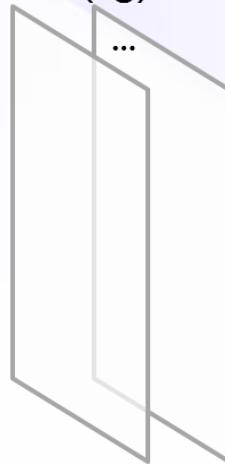


# AlphaGo의 특징맵 (48개)

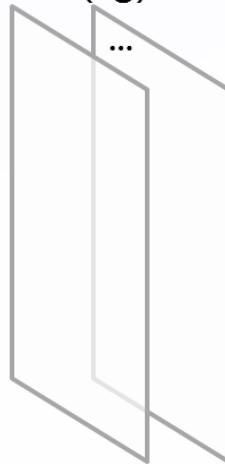
AlphaGo vs. Fan Hui  
두 번째 경기 30수



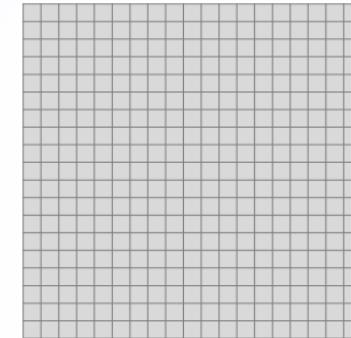
29 ~ 36. 자신의 단수  
(8장)



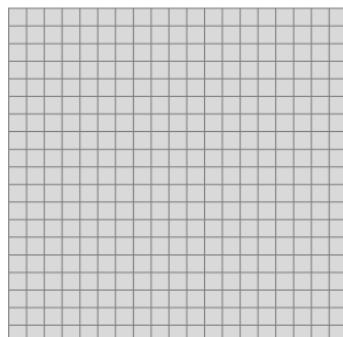
37 ~ 44. 한 수 뒤의 활로  
(8장)



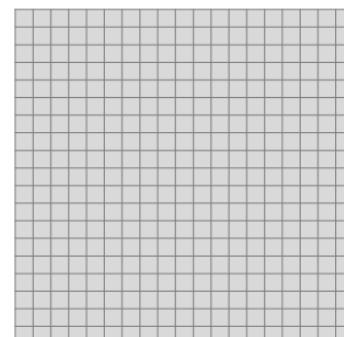
45. 축 감지



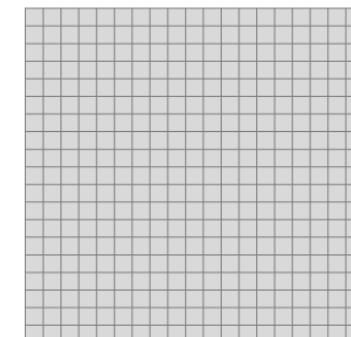
46. 축 탈출



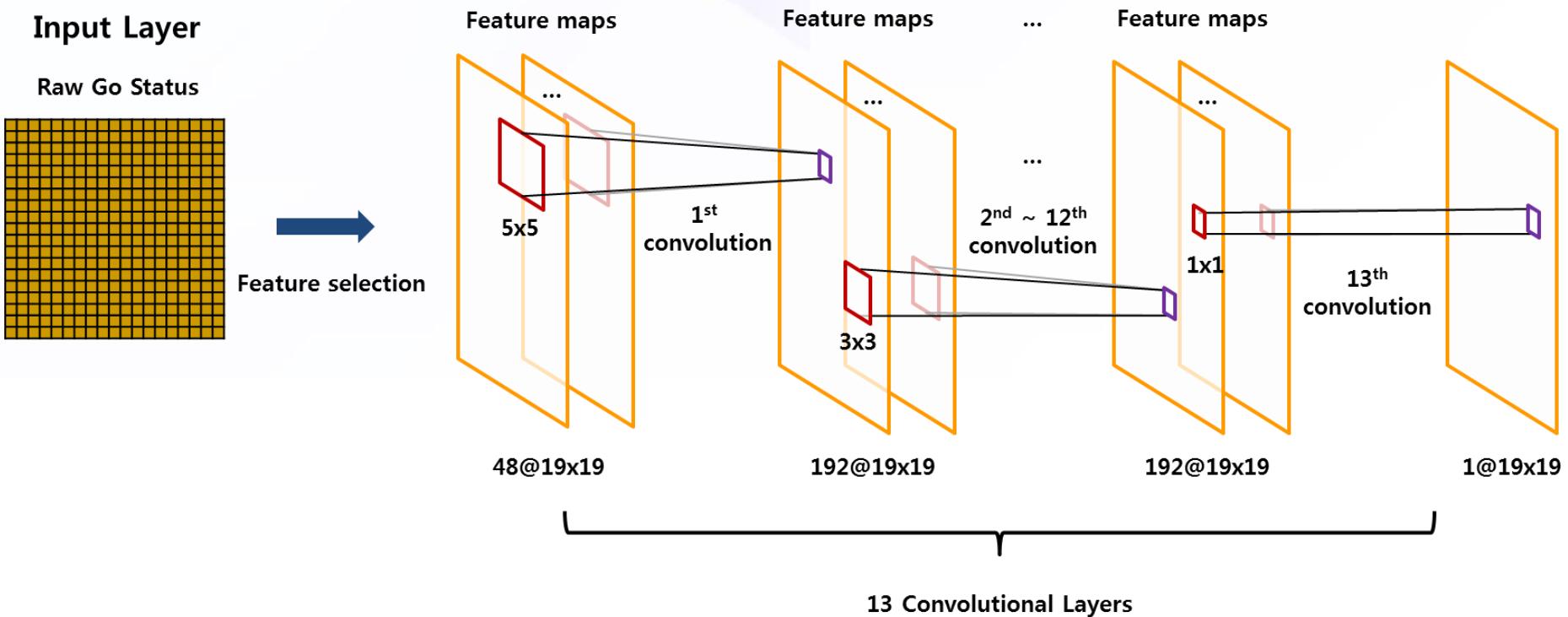
47. 자신의 눈을 채우는지 여부



48. 상수 0



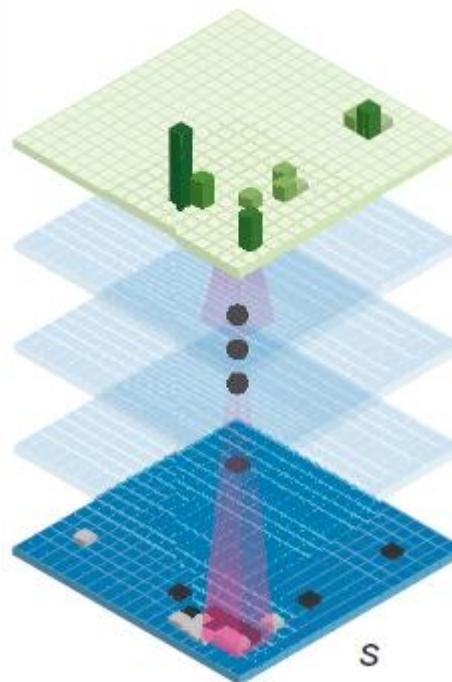
# AlphaGo의 컨볼루션 신경망 구조



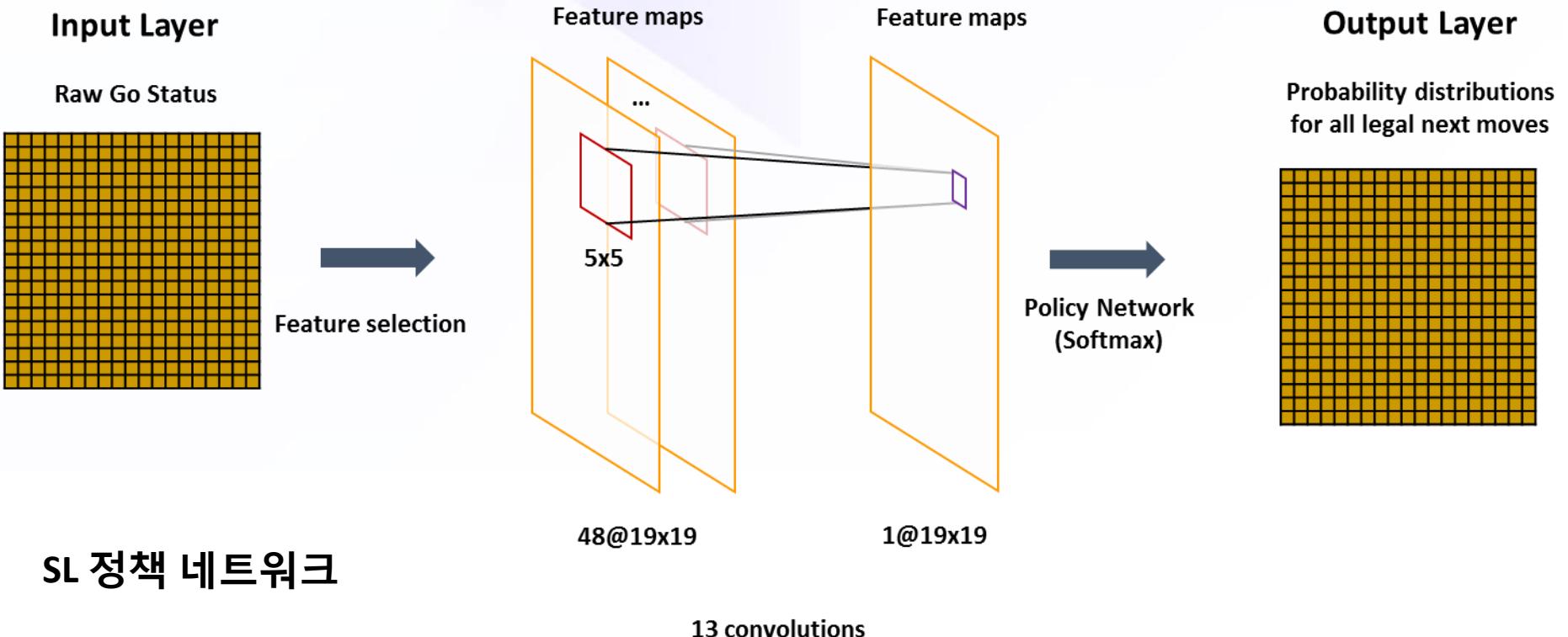
# 정책 네트워크

Policy network

$$p_{\sigma/\rho}(a|s)$$



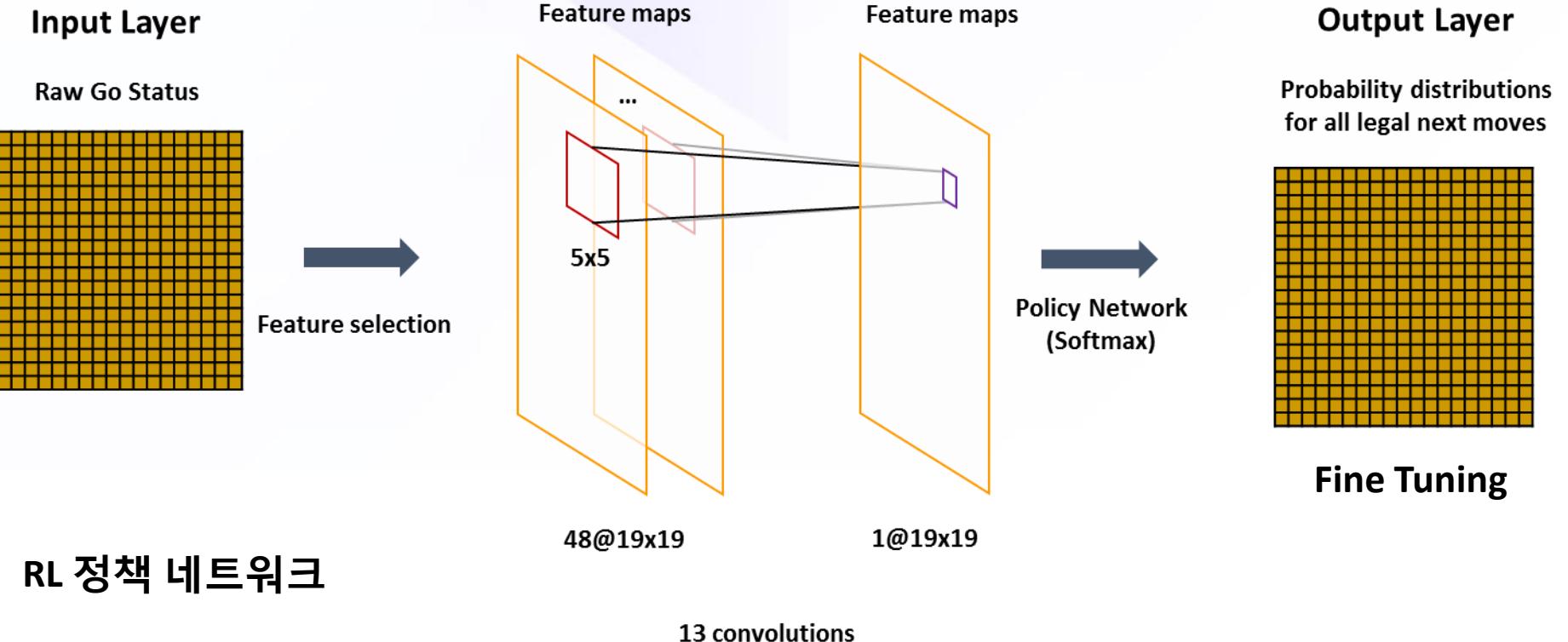
# 정책네트워크 – 프로바둑기사의 선호도 학습



데이터 : KGS 6~9단 기보 16만 개에서 바둑판 상태 약 3천만 개

계산비용 : 50 GPUs, 3 주

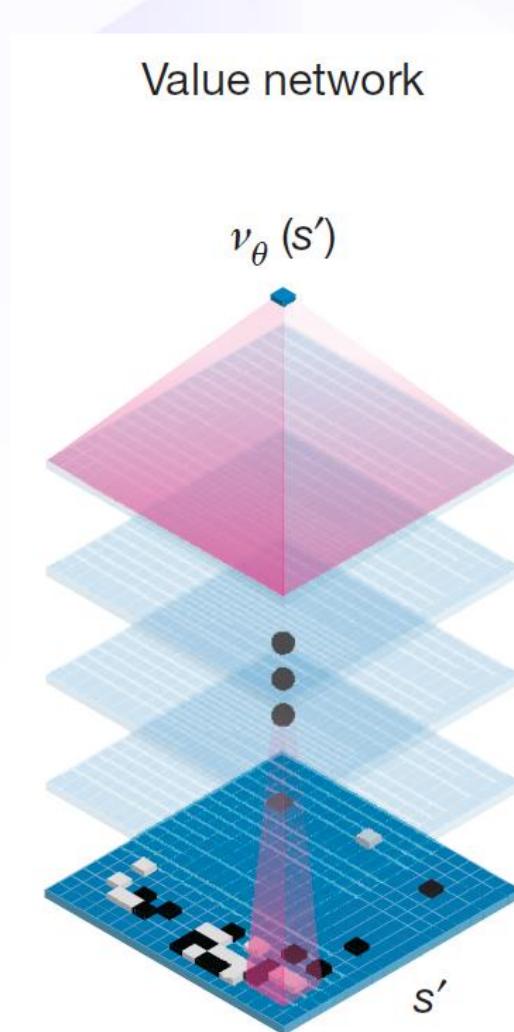
# 정책네트워크 – 스스로 경기하여 성능향상



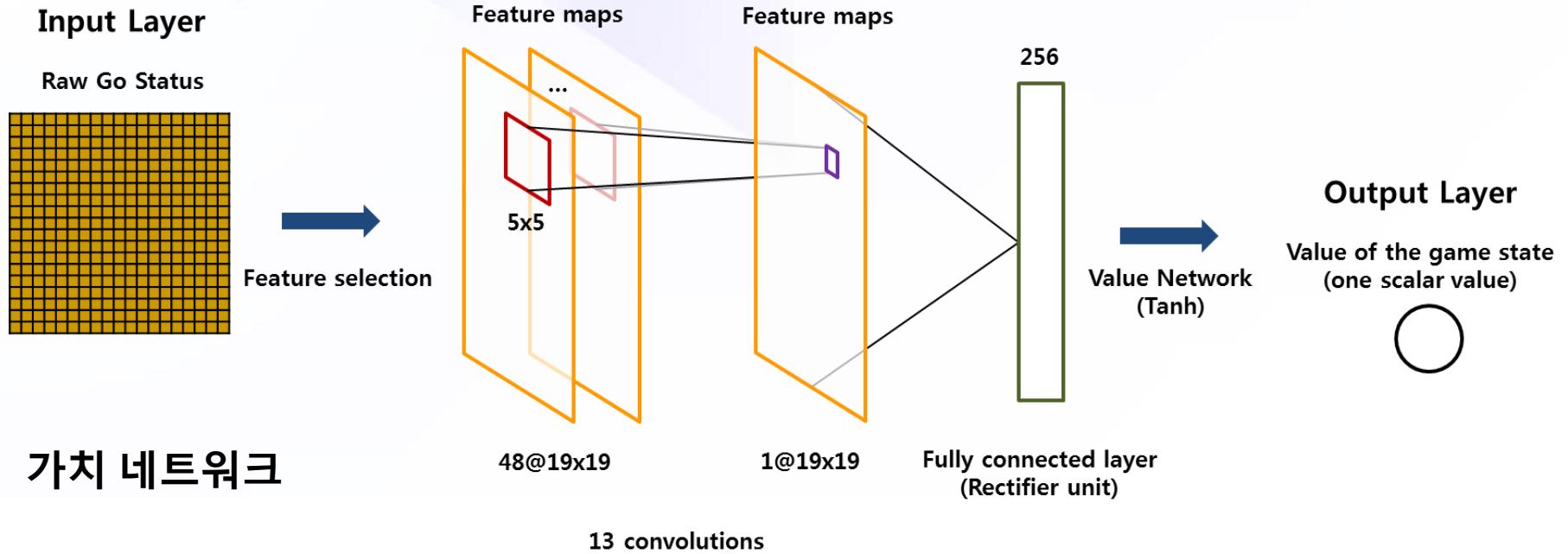
데이터 : 128만 번의 게임을 수행하면서 착수전략을 강화함

계산비용 : 50 GPUs, 1 일

# 가치 네트워크



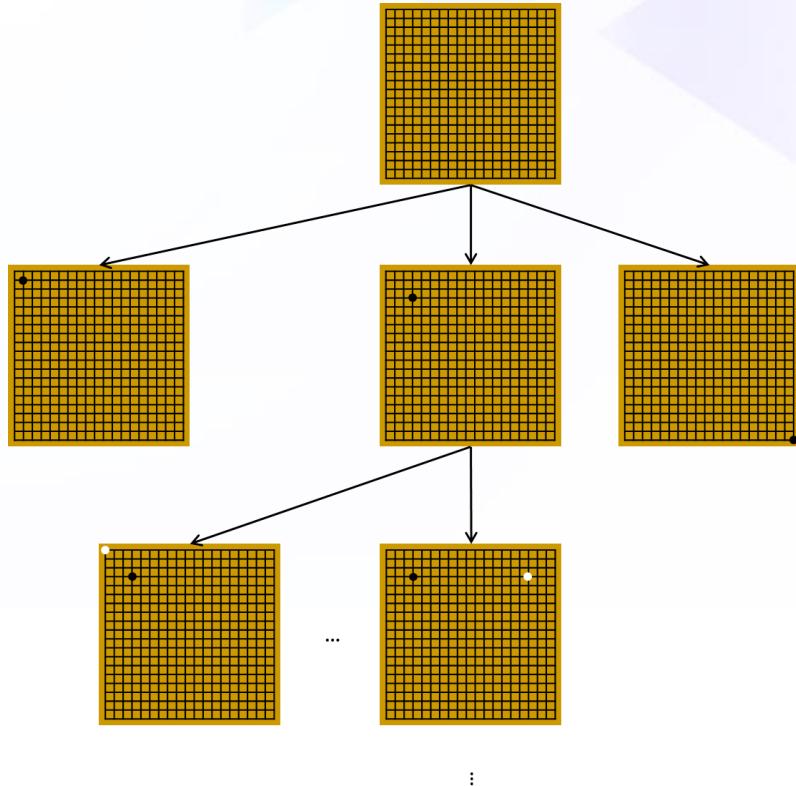
# 가치네트워크 – 바둑판 상태의 승률 계산



데이터 : 3천만 개의 모의전 (AlphaGo가 직접 생성)

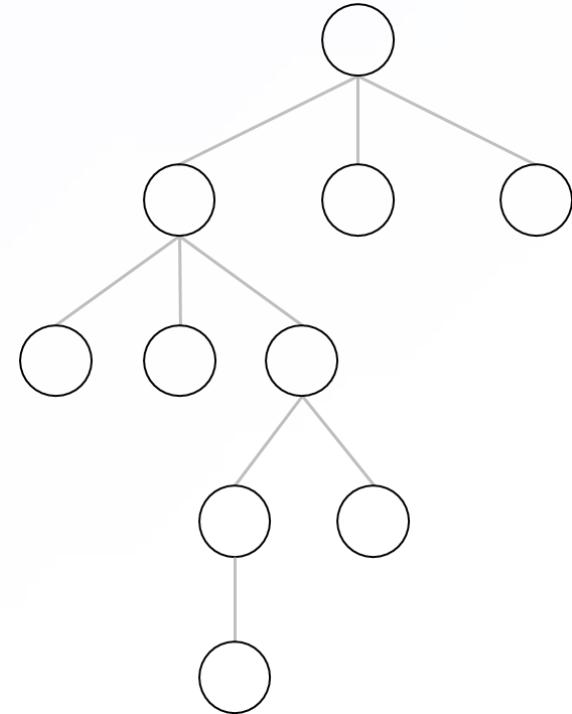
계산비용 : 50 GPUs, 1 주

# 바둑의 트리구조



모든 바둑판의 승률 계산 불가

한 개의 바둑판을 정확히 계산하는 것 조차  
비용이 매우 큼



노드 : 현재 바둑판 상태의 승률

엣지 : 다음 바둑판 상태의 승리 가능성

# AlphaGo의 착수 전략

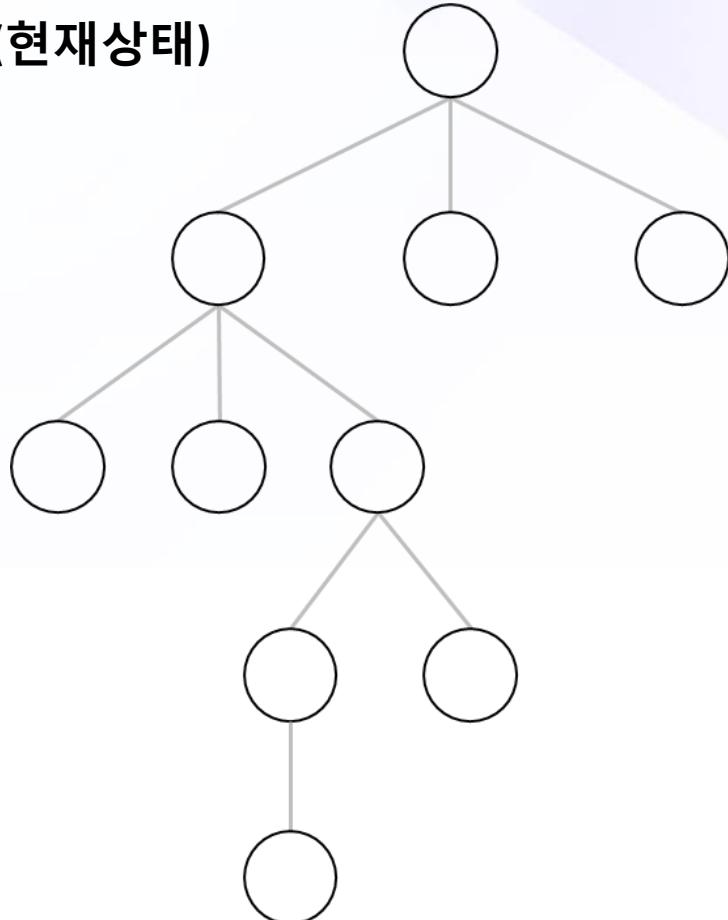
상대방(현재상태)

자신

상대방

자신

상대방



바둑 인공지능 프로그램을 구현하기 위해서  
자신 뿐만 아니라 상대방의 착수 전략을  
고려해야 함  
(상대방을 자신과 동일한 수준으로 둠)

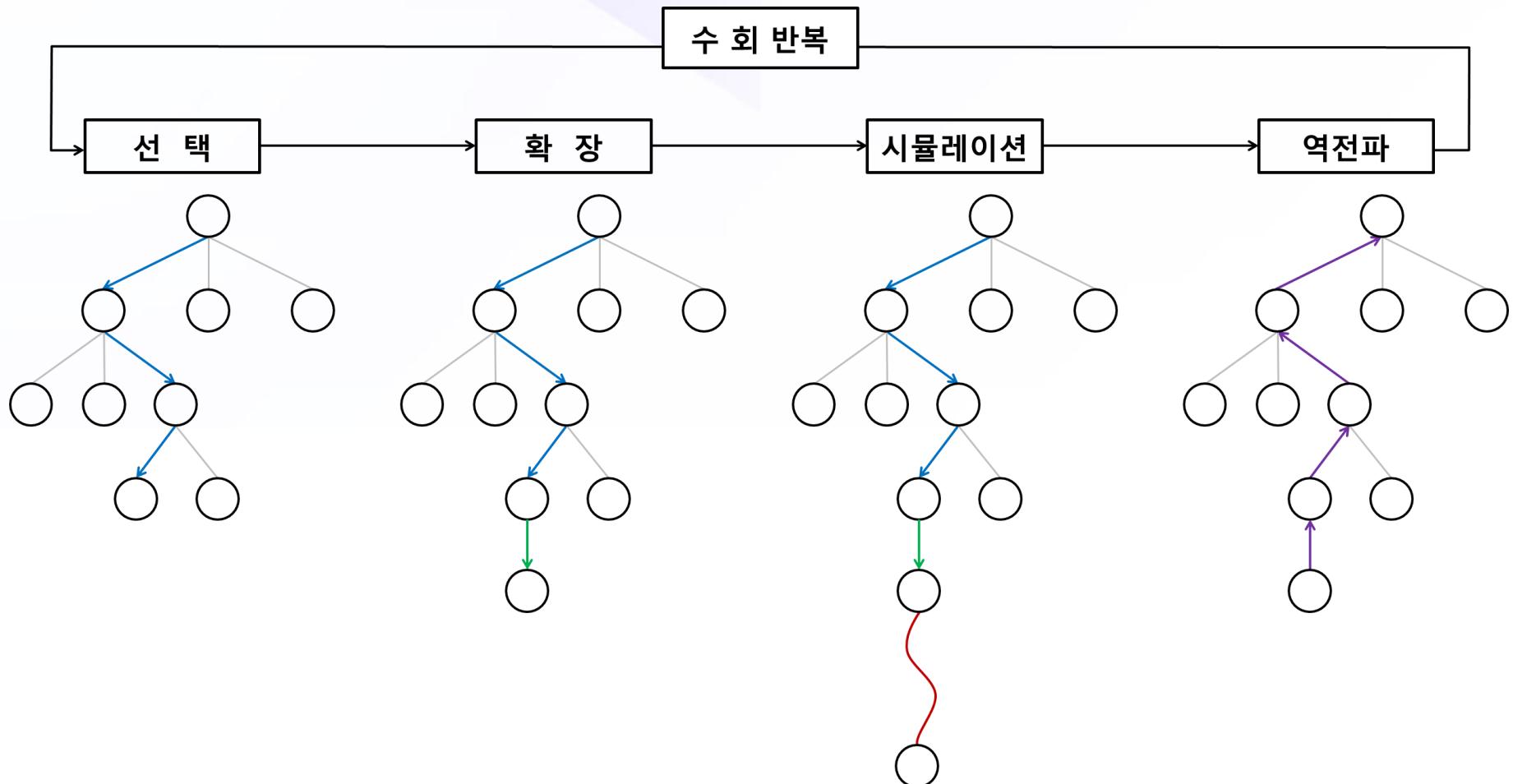


상대방을 이기려면 어느 경로를  
선택해야 하는가

무한대에 가까운 탐색의 폭과 깊이를  
어떻게 줄일 것인가?

# 몬테카를로 트리 탐색 알고리즘 (MCTS)

바둑에서 가장 널리 사용되는 탐색 알고리즘 (Coulom, 2006, Crazy Stone)

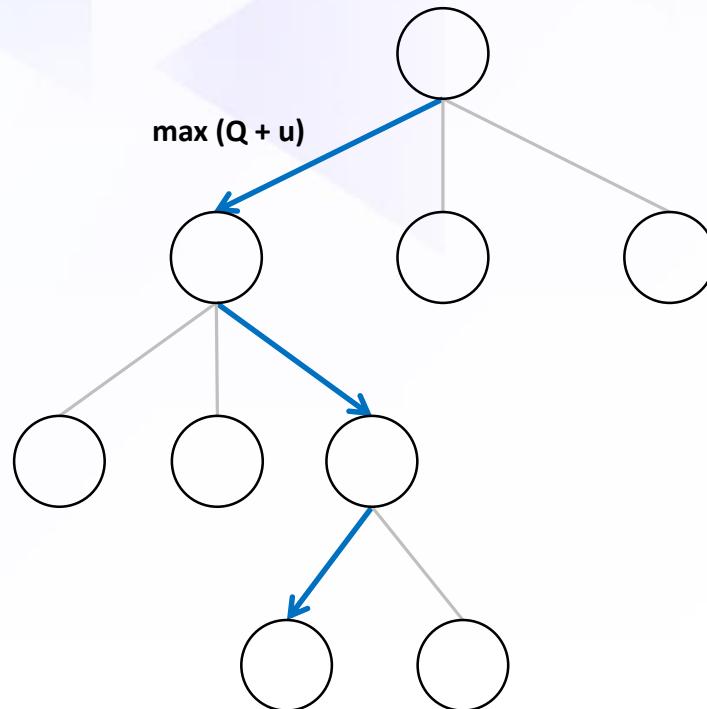


# MCTS의 핵심요소

정책	<p><b>트리의 폭을 제한하는 역할</b></p> <p>※ MCTS의 두 번째 단계인 확장에서 주로 사용되는 것으로, 특정 시점에서 가능한 모든 수 중 가장 승리가능성이 높 은 것을 제안</p>
가치	<p><b>트리의 깊이를 제한하는 역할</b></p> <p>※ 가치는 현재 대국상황의 승산을 나타낸 것으로, 승산이 정확할수록 많은 수(더 깊은 노드)를 볼 필요가 없음</p>

# MCTS step 1. 선 택

뿌리 위치 (t)



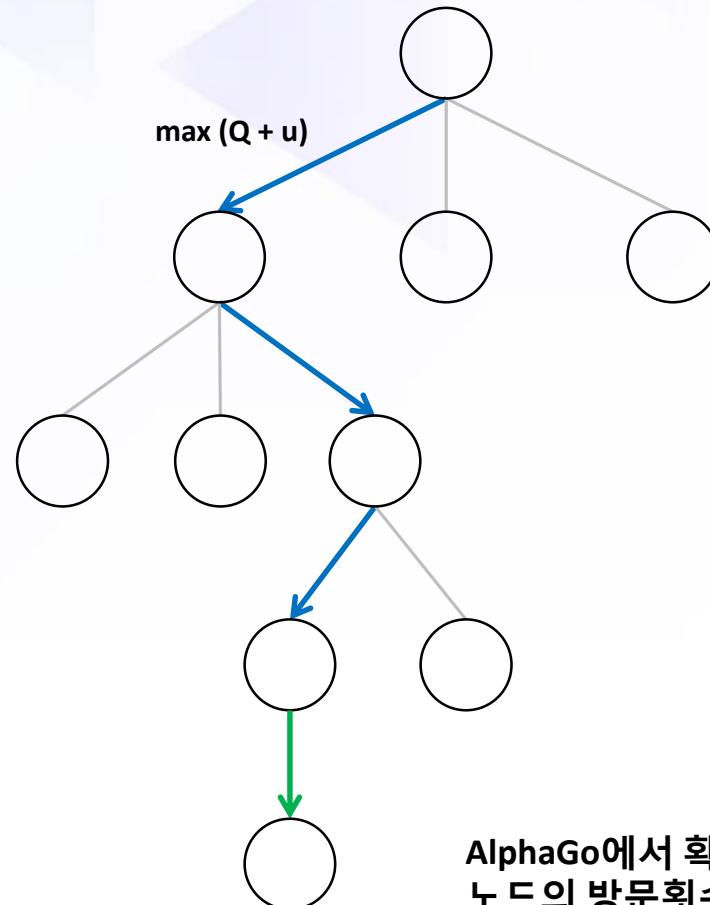
트리의 마지막까지  
승리가능성이 가장 높은 경로 선택

Q : 특정 바둑상태에서 다음 수를 둘 경우의 승리가능성  
(가치네트워크 + 시뮬레이션 결과)

u : 특정 다음 수를 두는 전문가의 확률(정책 네트워크)에 비례,  
노드 방문횟수에는 반비례 → 탐색의 폭을 넓히는 모수

# MCTS step 2. 확장

뿌리 위치 ( $t$ )

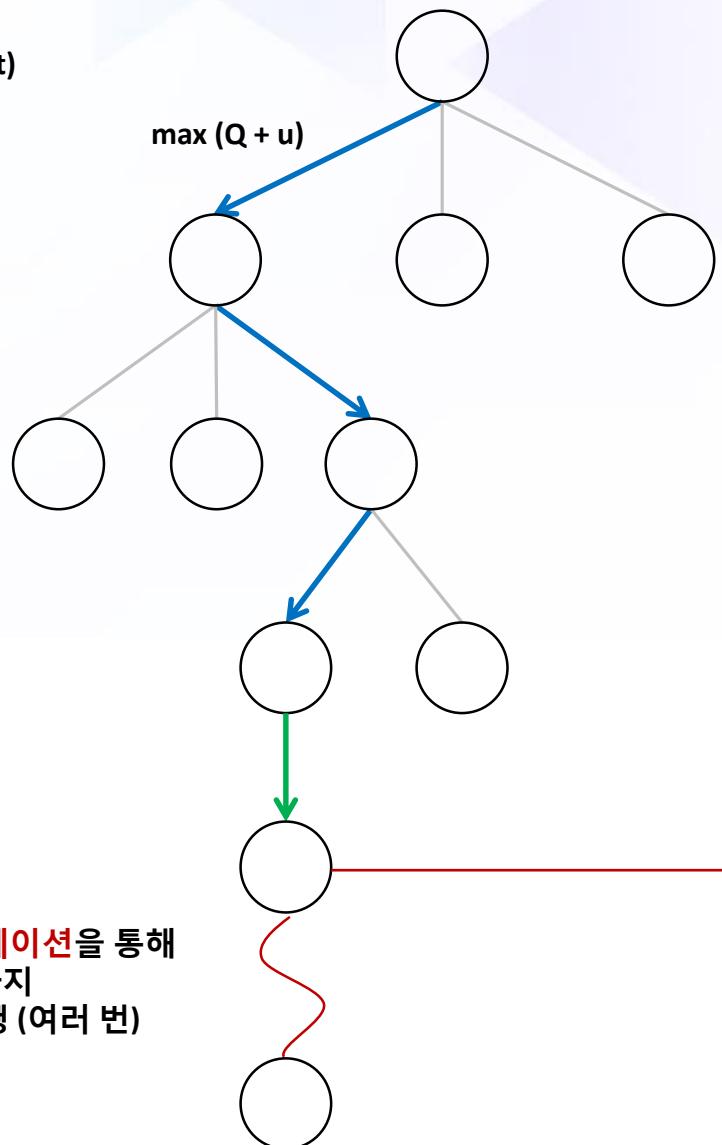


트리 마지막 노드의 방문횟수가  
일정 수 이상일 경우 노드를 확장

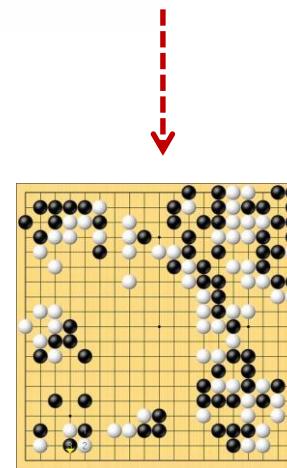
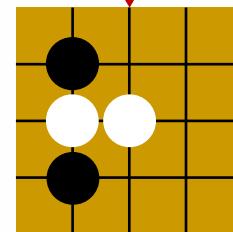
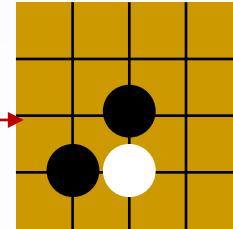
AlphaGo에서 확장하는 기준은  
노드의 방문횟수가 40회를 넘어설 때

# MCTS step 3. 시뮬레이션

뿌리 위치 (t)



고속 시뮬레이션을 통해  
종료시점까지  
게임을 진행 (여러 번)



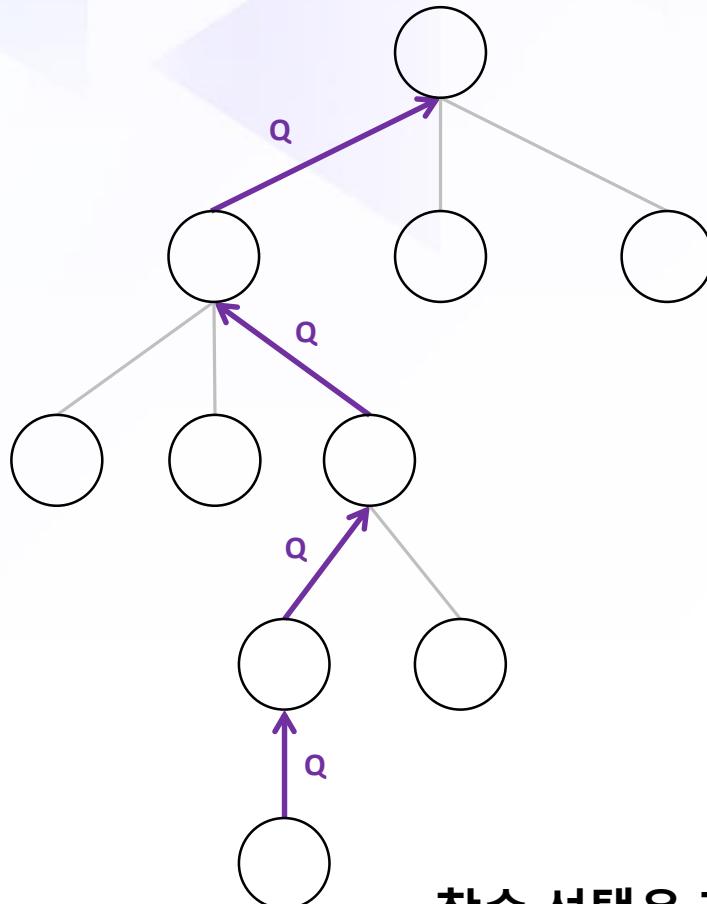
흑 돌, 백 돌, 빈 칸, 활로 정보를 통해서  
rollout 정책  $p_\pi$ 로 다음 수를 정함  
• 3x3 상황에서 가장 최적의 수로 착수  
• Rollout 정책은 Tygem 서버에서 약  
8백만개 데이터로 학습함

게임 종료시까지 시뮬레이션 수행

종료시점에서 승패 여부를 확인

# MCTS step 4. 역전파

뿌리 위치 (t)



시뮬레이션으로 갱신된 Q값 역전파

착수 선택은 가장 많이 방문한 노드로 결정

Q 값이 아닌 이유는 과적합 때문

# AlphaGo 정리

---

## ● 착수전략

- MCTS 탐색에서 가장 많이 방문한 노드(바둑판 상태)로 착수 결정
- 탐색의 범위는 딥러닝을 활용해서 전문가의 시점으로 좁힘
  - 정책네트워크 : 탐색의 폭을 줄임 (현재 상태에서 전문가들이 선호하는 다음 수)
  - 가치네트워크 : 탐색의 깊이를 줄임 (현재 상태의 승산을 근사 → 승산이 낮은 곳은 탐색x)

## ● 시뮬레이션 활용 (Fast Rollout)

- 가치네트워크는 입력값에 대한 결과가 정해져 있음 (deterministic)  
(i.e. 같은 바둑판 상태의 가치네트워크 값은 동일함)
- 하지만 가치네트워크는 여전히 근사값이므로 이를 보완하기 위해 바둑게임 진행시 추가적으로 시뮬레이션 수행
- AlphaGo는 기존에 있었던 연구결과들을 조합해서 최상의 성능을 이끌어냄 (알고리즘 차원에서 새로운 것은 없음)

# AlphaGo의 계산 성능

---

- AlphaGo는 딥러닝으로 구현된 정책과 가치네트워크를 활용하여 MCTS 탐색기법을 통해 착수를 결정
  - 분산 AlphaGo를 기준으로 착수(decision)는 초읽기 30초 동안 약 10만 번의 수읽기를 통해 결정
    - 체스 인공지능 프로그램인 딥블루의 경우 약 2억 번의 수읽기
    - 바둑계의 전설적인 인물인 이창호 9단의 경우 100수 정도의 수읽기
- AlphaGo의 MCTS 탐색에서 가장 시간이 많이 소요되는 부분은 가치, 정책네트워크의 값을 산출하는 과정
  - 13층의 컨볼루션 신경망의 값을 산출해내기 위해서는 약 300억 번의 연산수(30 GFLOP)가 필요함
  - 이러한 이유로 AlphaGo의 탐색 쓰레드는 비동기식(asynchronous)으로 진행됨

# 계산성능의 척도 - 부동소수점 연산수

---

## ● 부동소수점 연산수 (floating-point operations, flop)

- 알고리즘을 실제로 구현했을 때 필요한 연산수를 나타냄
- 한 개의 연산은 일반적으로 덧셈, 곱셉, 비교로 간주하나, 계산자원 구조에 따라 덧셈과 곱셈을 하나의 연산으로 보기도 함
  - GPU의 경우 FMAD(Fast Multiplication and Add)라는 장치가 내재되어 있음
- 유효숫자(Precision)에 따른 성능차이가 존재
  - 32-bit 부동소수점(float, 유효숫자 7자리)과 64-bit 부동소수점(double, 유효숫자 16자리)에 대한 연산성능이 다름 (일반적으로 float에 대한 성능이 높음)

## ● 초당 부동소수점 연산수 (floating-point operations per second, FLOPS or FLOP/s)

- 연산처리장치의 연산능력을 표현하는 지표로 슈퍼컴퓨터의 성능비교 등에 사용됨
- 2015년 11월 세계에서 가장 빠른 슈퍼컴퓨터의 성능은 33.86 PetaFLOP/s
  - (중국 광저우 슈퍼컴퓨팅 센터) 텐허2 : 16,000 nodes, 3,210,000 cores (including Xeon Phi)
  - ※ (한국 기상청, 29위) 누리 : 2.4PetaFLOPS, 69,600 cores

# AlphaGo의 (테스트) 계산자원

## ● CPU

- Intel Xeon CPU E5-2643 v2 @ 3.5 GHz



Source : <http://www.amazon.com/HP-712775-L21-E5-2643-3-5GHz-Processor/dp/B00PYTVVWI>

- 코어수/스레드수 : 6 cores / 12 threads
- 성능 : 66.61 GFLOP/s
- 최대 CPU 구성 : 2
- 가격 : \$ 1552
- 발매일 : Q3' 2013

## ● GPU

- GeForce GTX Titan Black



Source : <http://www.nvidia.co.kr/gtx-700-graphics-cards/gtx-titan-black/>

- 코어수 : 2880 cores
- 성능 : 5.1 Tera FLOP/s (single),  
1.7 Tera FLOP/s (double)
- 가격 : \$ 999
- 발매일 : March 25, 2014

Source : Maddison, Chris J., et al. "Move evaluation in go using deep convolutional neural networks." *arXiv preprint arXiv:1412.6564* (2014).

CPU Performance, [https://setiathome.berkeley.edu/cpu\\_list.php](https://setiathome.berkeley.edu/cpu_list.php)

List of NVIDIA Graphics Processing Units, [https://en.wikipedia.org/wiki/List\\_of\\_Nvidia\\_graphics\\_processing\\_units](https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units)

# AlphaGo의 계산자원 (추정)

## ● 싱글 머신

- CPU cores : 48 개
  - 12cores(with HTT) x 4 CPUs, or 8cores x 6 CPUs
- GPU 개수 : 8 개
- 노드 구성은 (4 CPU sockets + 8 PCIs)를 탑재한 고성능 계산서버로 추정
  - or (6 CPU sockets + 8 PCIs)
- 가격은 약 5만불 정도이고 시간당 소비전력은 2500 Watt 수준



Supermicro MB  
4CPUs + 4PCIe + (4PCIe)  
\$1,278

## ● 분산 머신

- CPU cores : 1202 개 (최대1920)
- GPU 개수 : 176 개 (최대280)
- 약 40대内外의 싱글머신으로 구성
  - 한화 약 22 ~ 25억 원



8 VGAs 예시

## 4. 결 론

# 인공지능 - 문제해결의 도구

---

- 딥러닝은 데이터를 학습하는 도구로써 각광받음
  - 문제를 딥러닝으로 해결하는 Knowhow가 매우 중요한 요소
    - HW, 오픈소스도구, 빅데이터가 기본적인 요소이지만 딥러닝을 어떻게 구축하고 어떻게 최적화시켜야 하는지가 더 중요
  - GPU 한 개라도 막강한 성능을 발휘
  - 아이디어 실현의 도구로 활용할 수 있음
  - 여전히 '딥러닝'에 대한 진입장벽은 존재하나 많이 낮아짐
- 인공지능에 대한 부정적인 걱정에 앞서 이해하는 노력이 필요