Reinforcement Learning with DNNs: *AlphaGo* to *AlphaZero*

Mark Craven and David Page

Computer Sciences 760

Spring 2019

Goals for the Lecture

- You should understand the following concepts:
  - Monte Carlo tree search (MCTS)
  - Self-play
  - Residual neural networks
  - *AlphaZero* algorithm

A Brief History of Game-Playing as a CS/AI Test of Progress

- 1944: Alan Turing and Donald Michie simulate by hand their chess algorithms during lunches at Bletchley Park
- 1959: Arthur Samuel's checkers algorithm (machine learning)
- 1961: Michie's Matchbox Educable Noughts And Crosses Engine (*MENACE*)
- 1991: Computer solves chess endgame thought draw: KRB beats KNN (223 moves)
- *1992: TD Gammon* trains for Backgammon by self-play reinforcement learning
- 1997: Computers best in world at Chess (*Deep Blue* beats Kasparov)
- 2007: Checkers "solved" by computer (guaranteed optimal play)
- 2016: Computers best at Go (*AlphaGo* beats Lee Sodol)
- 2017 (4 months ago): *AlphaZero* extends *AlphaGo* to best at chess, shogi

Only *Some* of these involved Learning

- 1944: Alan Turing and Donald Michie simulate by hand their chess algorithms during lunches at Bletchley Park
- 1959: Arthur Samuel's checkers algorithm (machine learning)
- 1961: Michie's Matchbox Educable Noughts And Crosses Engine (*MENACE*)
- 1991: Computer solves chess endgame thought draw: KRB beats KNN (223 moves)
- *1992: TD Gammon* trains for Backgammon by self-play reinforcement learning
- 1997: Computers best in world at Chess (*Deep Blue* beats Kasparov)
- 2007: Checkers "solved" by computer (guaranteed optimal play)
- 2016: Computers best at Go (*AlphaGo* beats Lee Sodol)
- 2017 (4 months ago): *AlphaZero* extends *AlphaGo* to best at chess, shogi

Only *Some* of these involved Learning

- 1944: Alan Turing and Donald Michie simulate by hand their chess algorithms during lunches at Bletchley Park
- 1959: Arthur Samuel's checkers algorithm (machine learning)
- 1961: Michie's Matchbox Educable Noughts And Crosses Engine (*MENACE*)
- 1991: Computer solves chess endgame thought draw: KRB beats KNN (223 moves)
- *1992: TD Gammon* trains for Backgammon by self-play reinforcement learning
- 1997: Computers best in world at Chess (*Deep Blue* beats Kasparov)
- 2007: Checkers "solved" by computer (guaranteed optimal play)
- 2016: Computers best at Go (*AlphaGo* beats Lee Sodol)
- 2017 (4 months ago): *AlphaZero* extends *AlphaGo* to best at chess, shogi

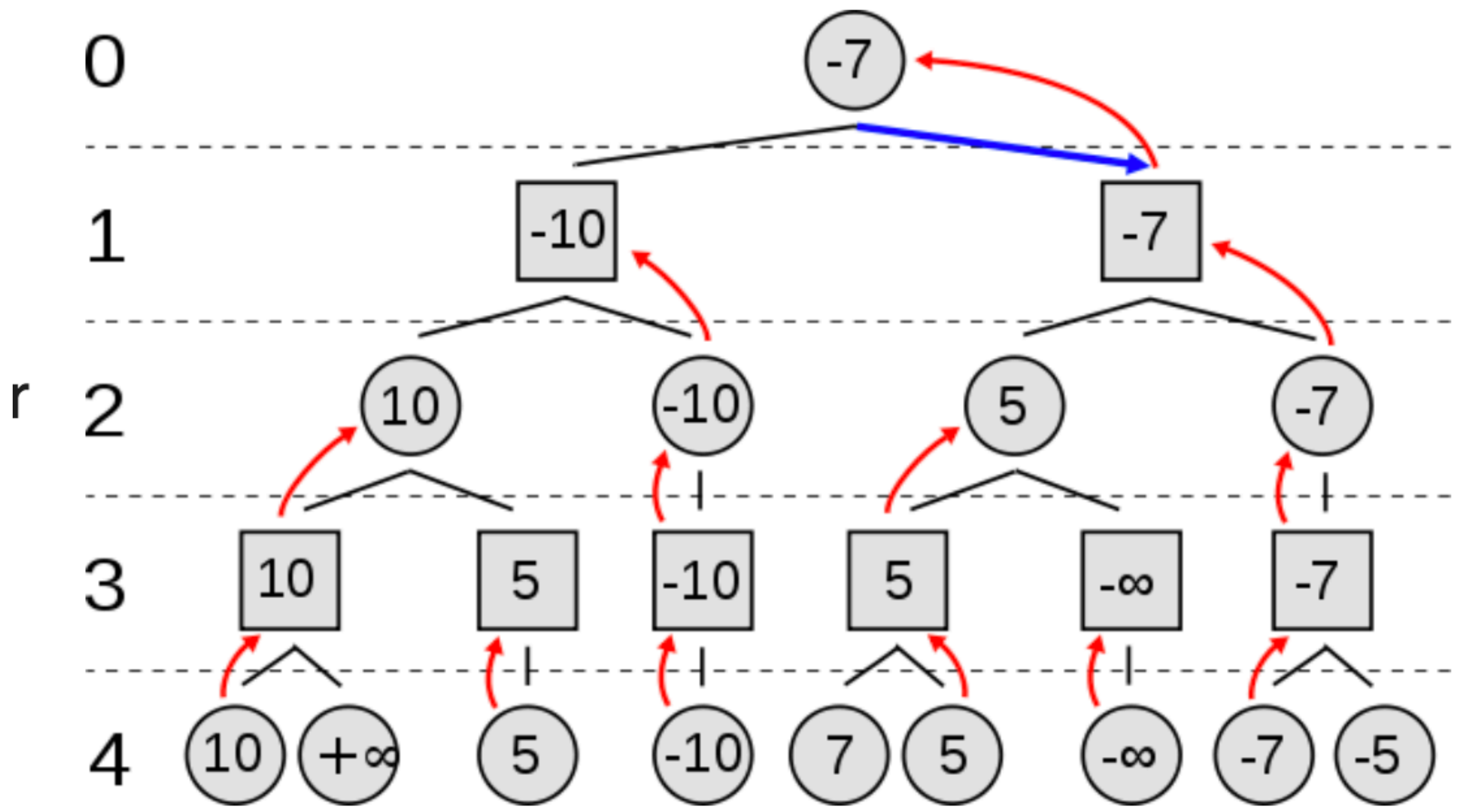
Background: Game Playing

- Until last year, state-of-the-art for many games including chess was *minimax* search with *alpha-beta pruning* (recall Intro to AI)

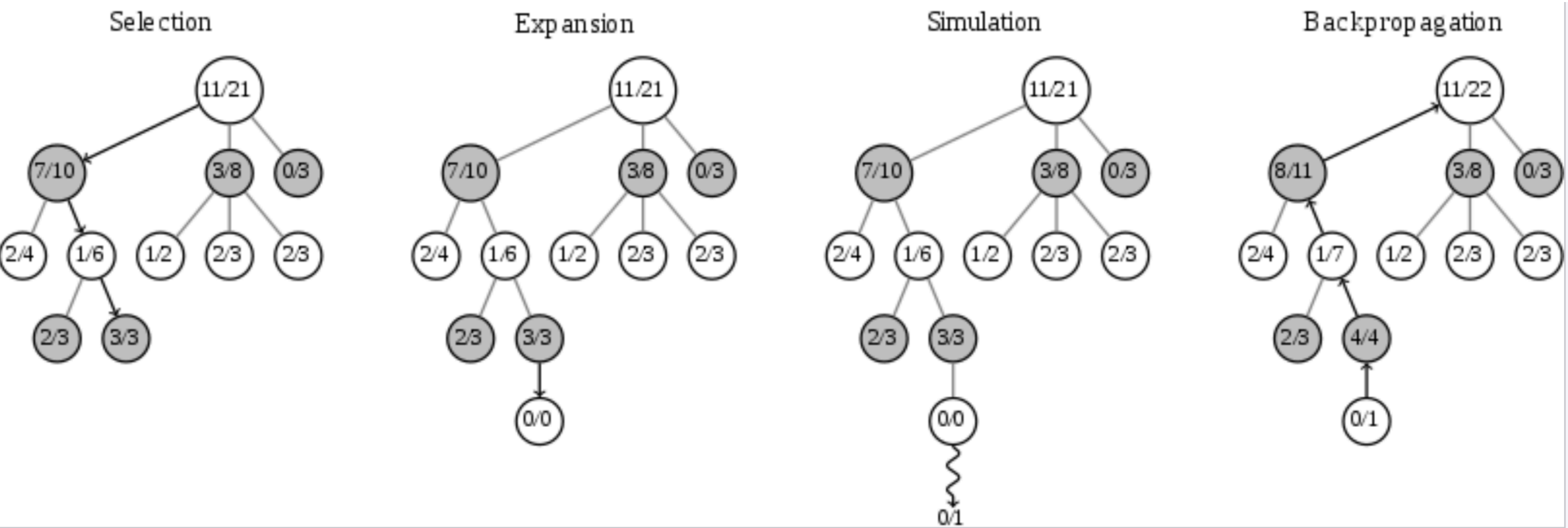
- Most top-performing game-playing programs didn't do learning

- Game of Go was one of the few games where humans still outperformed computers

Minimax in a Picture (thanks Wikipedia)



Monte Carlo Tree Search (MCTS) in a Picture (thanks Wikipedia)



Rollout
(Random Search)

Reinforcement Learning by *AlphaGo*, *AlphaGo Zero*, and *AlphaZero*: Key Insights

- MCTS with *Self-Play*
  - Don’t have to guess what opponent might do, so…
  - If no exploration, a big-branching game tree becomes *one path!*
  - You get an automatically improving, evenly-matched opponent who *is accurately learning your strategy*
  - “We have met the enemy, and he is us” (famous variant of Pogo, 1954)
  - No need for human expert scoring rules for boards from unfinished games

- Treat board as an image: use residual convolutional neural network

- AlphaGo Zero: One deep neural network learns both the value function and policy in parallel

- Alpha Zero: Remove rollout altogether from MCTS and just used current neural net estimates instead

AlphaZero (Dec 2017): Minimized Required Game Knowledge, Extended from Go to Chess and Shogi

Domain Knowledge

1. The input features describing the position, and the output features describing the move, are structured as a set of planes; i.e. the neural network architecture is matched to the grid-structure of the board.

2. *AlphaZero* is provided with perfect knowledge of the game rules. These are used during MCTS, to simulate the positions resulting from a sequence of moves, to determine game termination, and to score any simulations that reach a terminal state.

3. Knowledge of the rules is also used to encode the input planes (i.e. castling, repetition, no-progress) and output planes (how pieces move, promotions, and piece drops in shogi).

4. The typical number of legal moves is used to scale the exploration noise (see below).

5. Chess and shogi games exceeding a maximum number of steps (determined by typical game length) were terminated and assigned a drawn outcome; Go games were terminated and scored with Tromp-Taylor rules, similarly to previous work (*29*).

AlphaZero did not use any form of domain knowledge beyond the points listed above.

AlphaZero Uses:

- Reinforcement Learning
- Deep Neural Network learning

AlphaZero Uses:

- **Reinforcement Learning**

- Deep Neural Network learning

AlphaZero's version of Q-Learning

- No discount on future rewards
- Rewards of 0 until end of game; then reward of -1 or +1
- Therefore *Q*-value for an action *a* or policy ***p*** from a state *S* is exactly value function: $Q(S,\boldsymbol{p}) = V^{\boldsymbol{p}}(S)$
- *AlphaZero* uses one DNN (details in a bit) to model both ***p*** and *V*
- Updates to DNN are made (training examples provided) after game
- During game, need to balance exploitation and exploration

AlphaZero Algorithm

Initialize *DNN* $f_{\boldsymbol{\theta}}$

Repeat Forever

  Play Game

  Update $\boldsymbol{\theta}$

Play Game:

  Repeat Until Win or Lose:

    From current state *S*, perform MCTS

    Estimate move probabilities $\boldsymbol{\pi}$ by MCTS

    Record (*S*, $\boldsymbol{\pi}$) as an example

    Randomly draw next move from $\boldsymbol{\pi}$

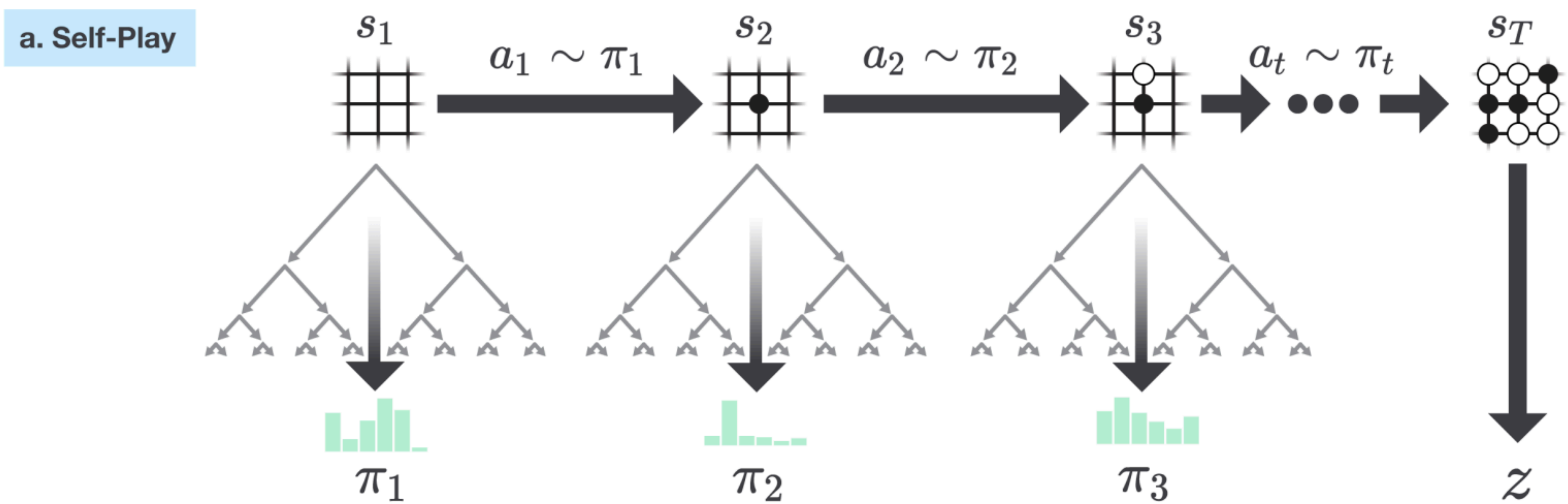
Update $\boldsymbol{\theta}$:

  Let *z* be previous game outcome (+1 or -1)

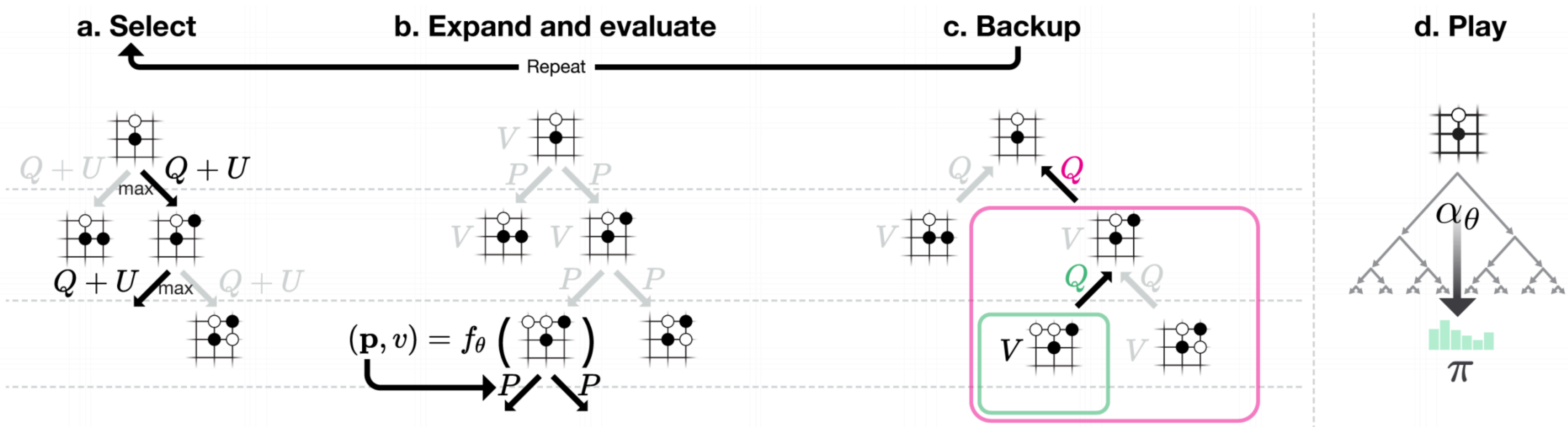
  Sample from last game's examples (*S*, $\boldsymbol{\pi}, z$)

  Train DNN $f_{\boldsymbol{\theta}}$ on sample to get new $\boldsymbol{\theta}$

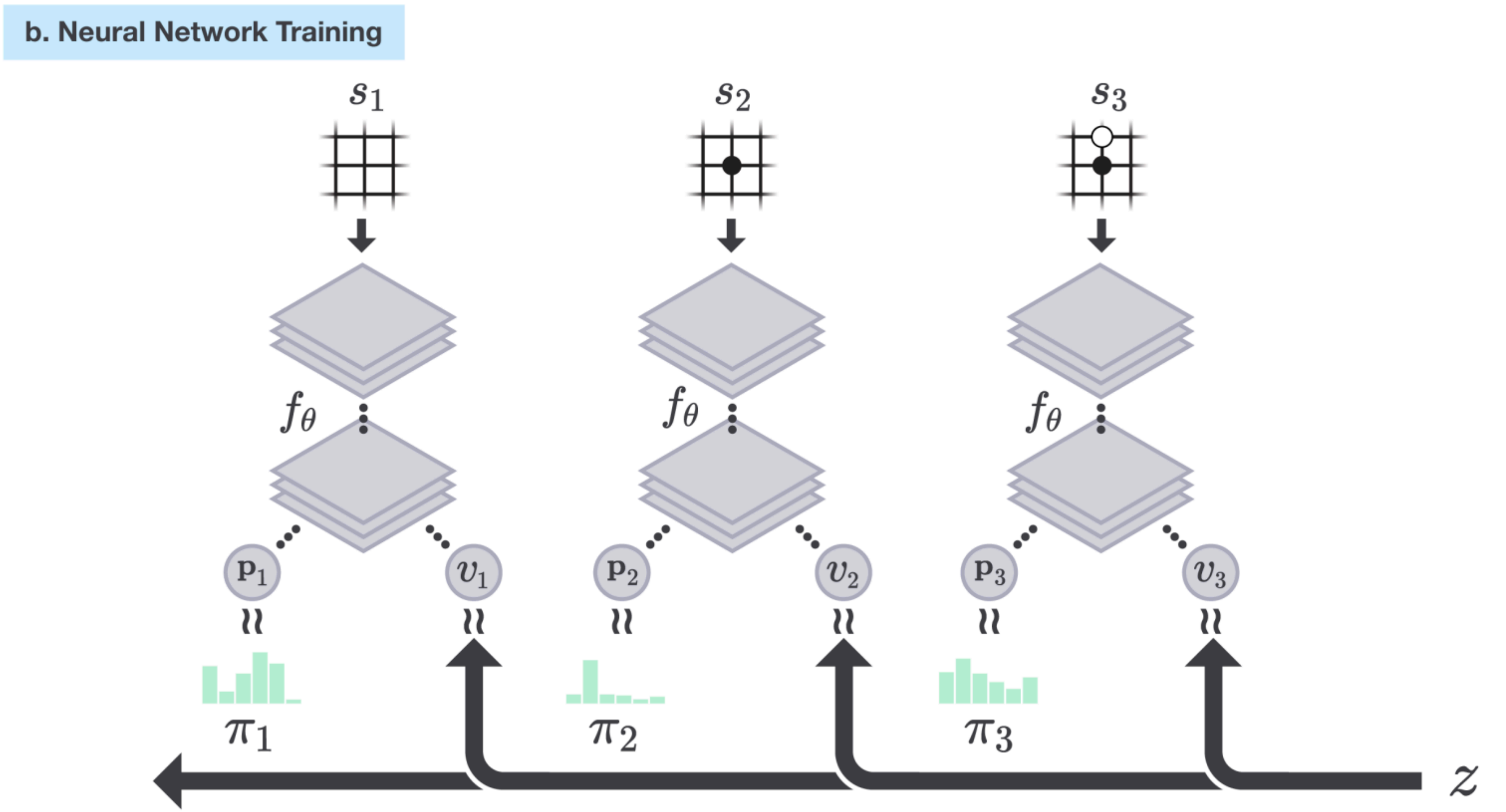
AlphaZero Play-Game



AlphaZero Monte Carlo Tree Search (MTCS)



AlphaZero Train DNN



Why Need MCTS At All?

- Could always make move DNN says has highest Q: no exploration
- Could just draw move from DNN's policy output
- Papers say MCTS output probability vector **p** selects stronger moves that just directly using the neural network's policy output itself (is there a possible lesson here for self-driving cars too?)
- Still need to decide how many times to repeat MCTS search (game-specific) and how to tradeoff exploration and exploitation in MCTS… AlphaZero paper just says choose move with "low count, high move probability, and high value"—AlphaGo paper more specific: maximize upper confidence bound
- Where $\boldsymbol{\tau}$ is temperature [1,2], and $N_{\tau}(s,b)$ is count of time action b has been taken from state s, raised to the power $1/\boldsymbol{\tau}$, choose:

$$a_t = \operatorname*{argmax}_a \left(Q(s_t, a) + u(s_t, a)\right), \text{ using a variant of the PUCT algorithm}$$

$$u(s,a) = c_{\text{puct}} P(s,a) \frac{\sqrt{\sum_b N_r(s,b)}}{1 + N_r(s,a)}$$

AlphaZero Uses:

- Reinforcement Learning
- **Deep Neural Network learning**

AlphaZero DNN Architecture: *Output Nodes* Represent Policy and Value Function

- A policy is a probability distribution over all possible moves from a state, so need units to represent all possible moves

- Chess is most complicated to describe moves (though Go and Shogi have higher numbers of moves to consider), so here is for Chess moves:
  - 8 x 8 = 64 possible starting positions for a move
  - 56 possible destinations for queen moves: 8 compass directions {N, NE, E, SE, S, SW, W, NW} times 7 possible move-lengths
  - Another 17 possible destinations for irregular moves such as knight
  - Some moves impossible, depending on the particular piece at a position (e.g., pawn can't make all queen moves) and location of other pieces (queen can't move through 2 other pieces to attack a third)
  - Weights for impossible moves are set to 0 and not allowed to change
  - Another layer to normalize results into probability distribution

- One deep neural network learns both the value function and policy in parallel: one additional output node for the *value* function, which estimates the expected outcome in the range [-1,1] for following the current *policy* from present (input) *state*

AlphaZero DNN Architecture: *Input Nodes* Represent Current Game State, Including any needed History

Go		Chess		Shogi	
Feature	Planes	Feature	Planes	Feature	Planes
P1 stone	1	P1 piece	6	P1 piece	14
P2 stone	1	P2 piece	6	P2 piece	14
		Repetitions	2	Repetitions	3
				P1 prisoner count	7
				P2 prisoner count	7
Colour	1	Colour	1	Colour	1
		Total move count	1	Total move count	1
		P1 castling	2		
		P2 castling	2		
		No-progress count	1		
Total	17	Total	119	Total	362

Table S1: Input features used by *AlphaZero* in Go, Chess and Shogi respectively. The first set of features are repeated for each position in a $T = 8$-step history. Counts are represented by a single real-valued input; other input features are represented by a one-hot encoding using the specified number of binary input planes. The current player is denoted by P1 and the opponent by P2.

Deep Neural Networks Trick #9: *ResNets* (*Residual Networks)*

- What if your neural network is *too deep*?

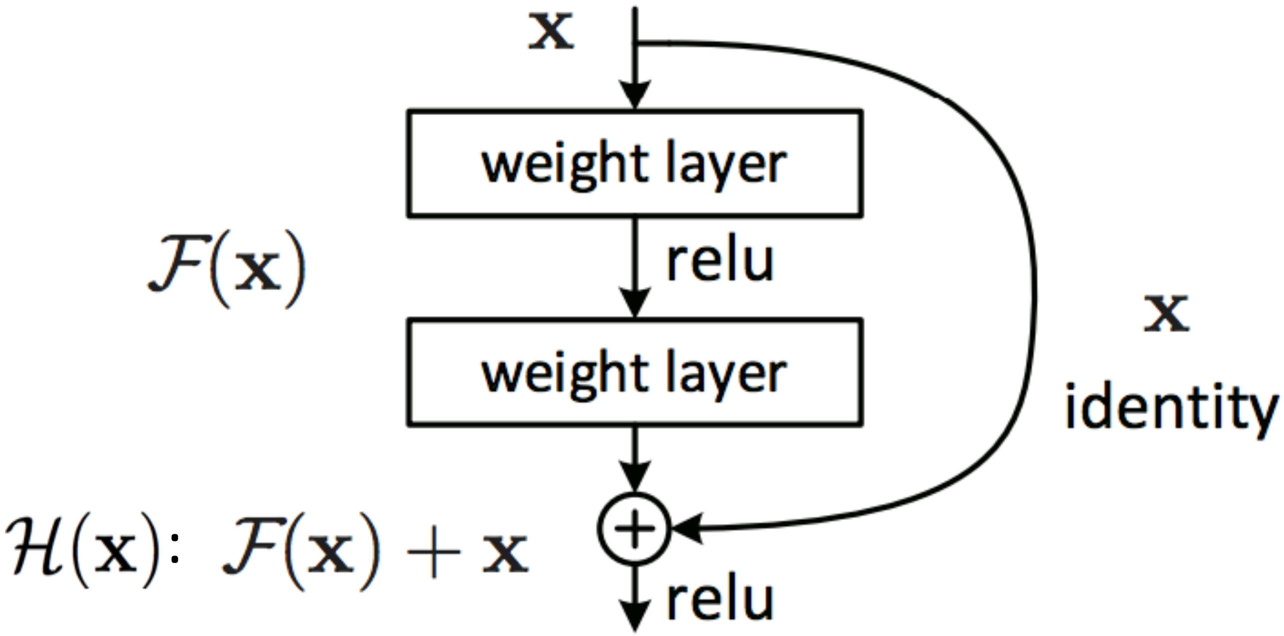
- In theory, that's no problem, given sufficient nodes and connectivity: early (or late) layers can just learn identity function (autoencoder)

- If already learned the right answer **x**, next layer should also output **x**

- In practice deep neural networks fail to learn identity when needed

- A solution: make identity *easy* or even the default; have to work hard to actually learn a non-zero addition to **x**, i.e., a non-zero *residual* and hence a non-identity function

Residual Network in a Picture (He, Zhang, Ren, Sun, 2015): Identity Skip Connection

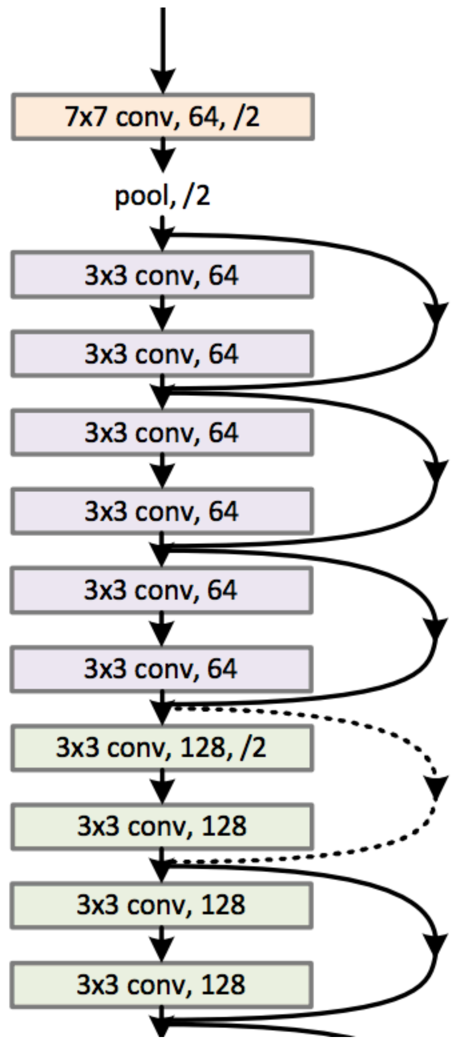


Note: output and input dimensionality need to be the same, componentwise sum right befor relu

$$\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$$

That's why called "residual"

Deep Residual Networks (ResNets): Start of a 35-layer ResNet (He, Zhang, Ren, Sun, 2015)



Dotted line denotes increase in Dimension (2 more such increases)

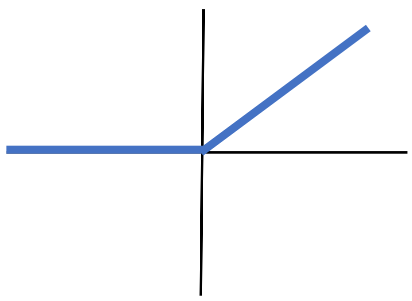
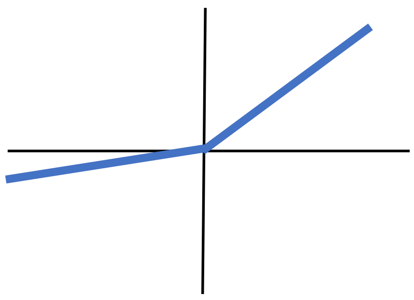
A Brief Aside: Leaky ReLUs

- Rectifiers used could be ReLU or ”Leaky ReLU”

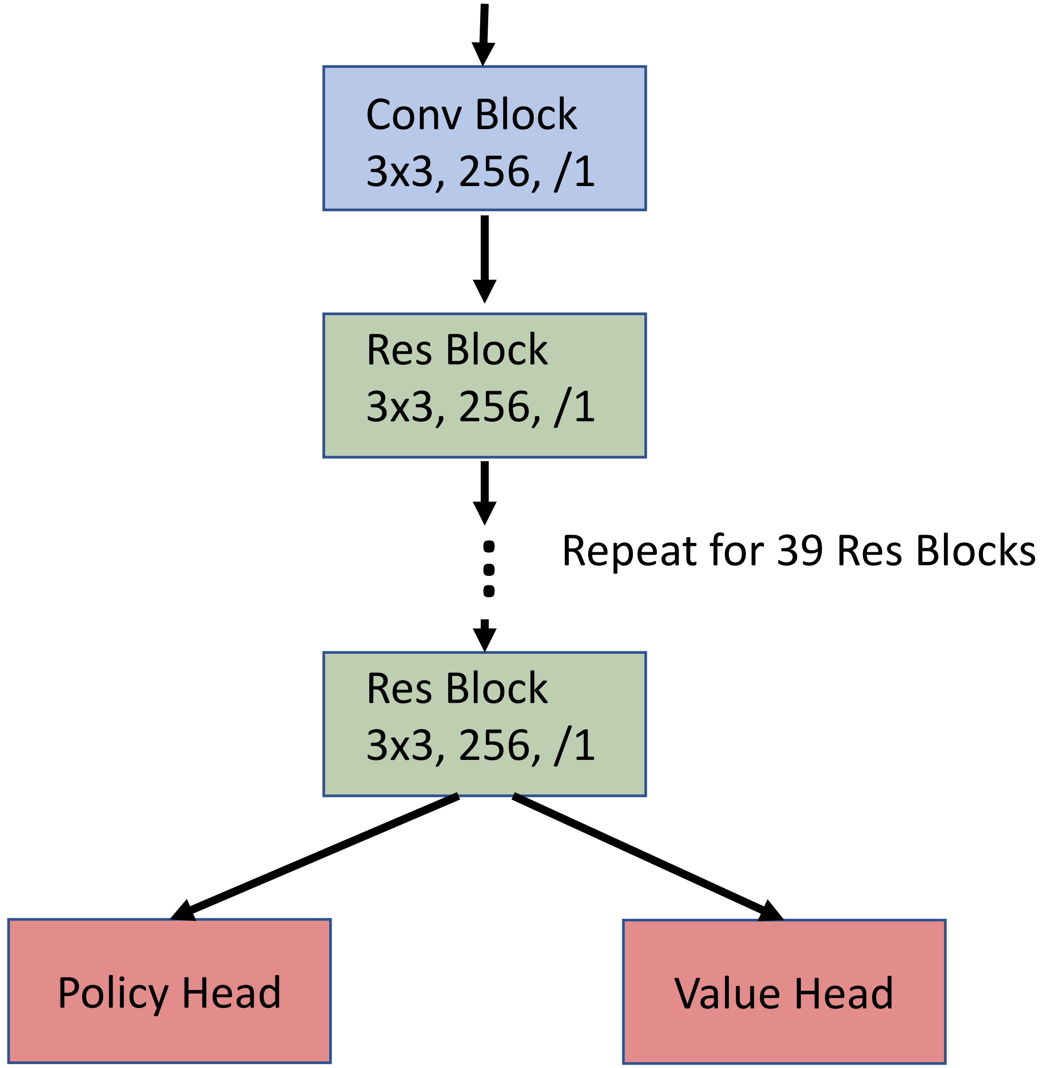
- Leaky ReLU addresses “dying ReLU” problem-–when input sum is below some value, output is 0, so no gradient for training

- ReLU: *f(x) = max(0,x)*

- Leaky ReLU: $f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$

- ReLU  Leaky ReLU 

AlphaZero DNN Architecture: Hidden Units Arranged in a Residual Network (a CNN with Residual Layers)



AlphaZero DNN Architecture: Convolution Block

The convolutional block applies the following modules:

1. A convolution of 256 filters of kernel size 3×3 with stride 1
2. Batch normalisation
3. A rectifier non-linearity

AlphaZero DNN Architecture: Residual Blocks

Each residual block applies the following modules sequentially to its input:

1. A convolution of 256 filters of kernel size 3×3 with stride 1
2. Batch normalisation
3. A rectifier non-linearity
4. A convolution of 256 filters of kernel size 3×3 with stride 1
5. Batch normalisation
6. A skip connection that adds the input to the block
7. A rectifier non-linearity

AlphaZero DNN Architecture: Policy Head (for Go)

The output of the residual tower is passed into two separate "heads" for computing the policy and value respectively. The policy head applies the following modules:

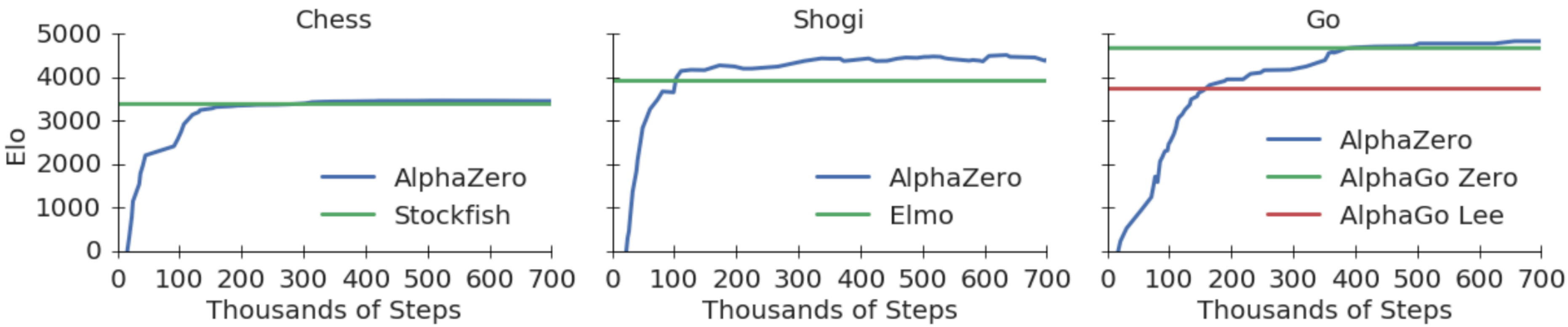
1. A convolution of 2 filters of kernel size 1×1 with stride 1
2. Batch normalisation
3. A rectifier non-linearity
4. A fully connected linear layer that outputs a vector of size $19^2 + 1 = 362$ corresponding to logit probabilities for all intersections and the pass move

AlphaZero DNN Architecture: Value Head

The value head applies the following modules:

1. A convolution of 1 filter of kernel size 1×1 with stride 1
2. Batch normalisation
3. A rectifier non-linearity
4. A fully connected linear layer to a hidden layer of size 256
5. A rectifier non-linearity
6. A fully connected linear layer to a scalar
7. A $\tanh$ non-linearity outputting a scalar in the range $[-1, 1]$

AlphaZero Compared to Recent World Champions



Program	Chess	Shogi	Go
AlphaZero	80k	40k	16k
Stockfish	70,000k		
Elmo		35,000k	

Table S4: Evaluation speed (positions/second) of *AlphaZero*, *Stockfish*, and *Elmo* in chess, shogi and Go.