nedeljainformatike v2.0

AlphaGo

Superljudska veštačka inteligencija u eksponencijalno rastućim prostorima

Petar Veličković (i Andrej Ivašković)

Nedelja informatike v2.5

14. april 2016.

Uvod

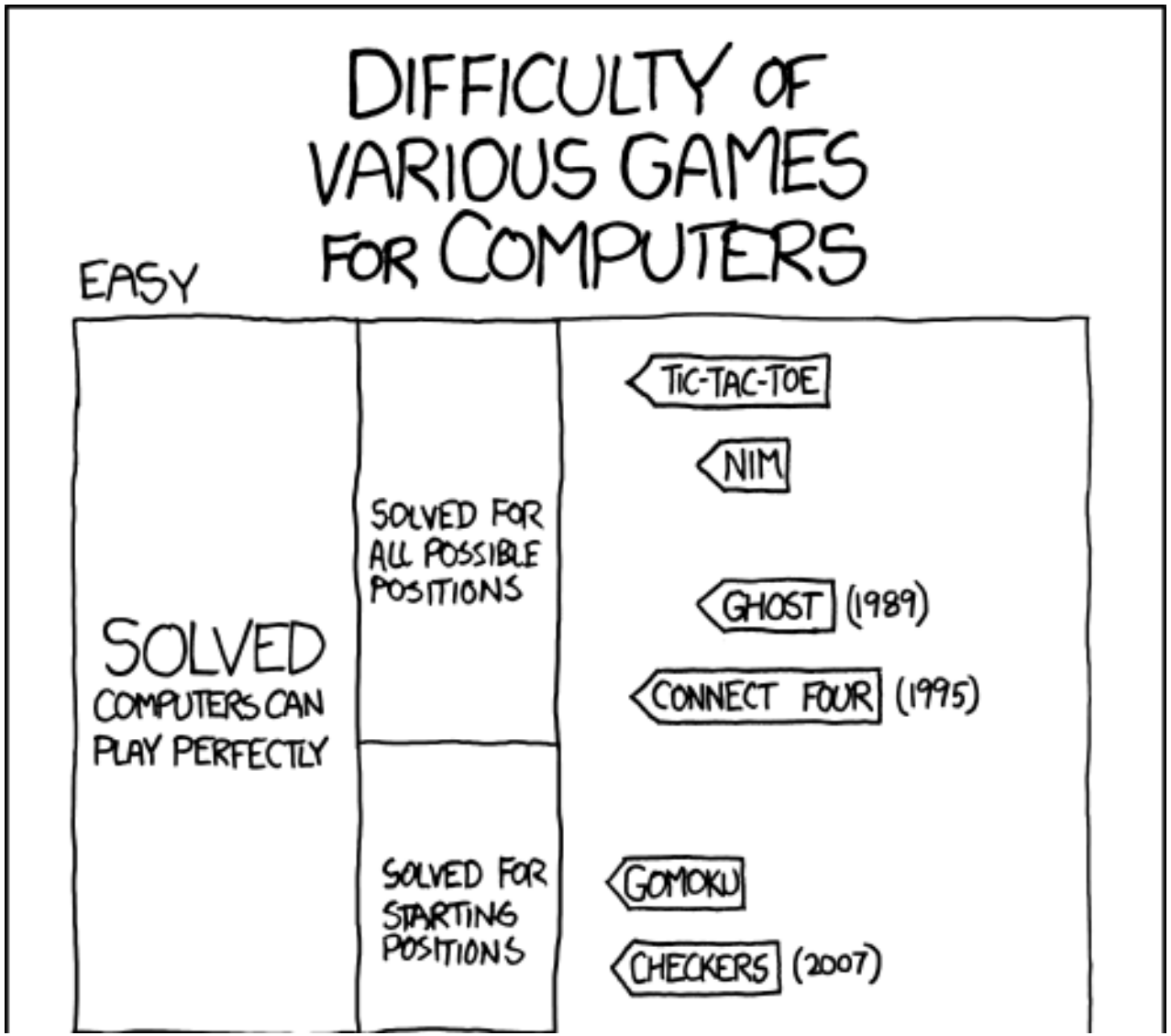
- *AlphaGo* je sistem za igranje igre Go koji je razvio *Google DeepMind*—nakon pobede nad Fan Hui-em (5:0), evropskim šampionom, nedavno je pobeđen i jedan od najboljih igrača svih vremena, **Lee Se-dol** (4:1).
- Ovo je, bez imalo prenaglašavanja, *jedan od najvećih napredaka veštačke inteligencije do sada*—mnogi eksperti su predviđali da ovakav rezultat neće biti moguć bar u narednih 10 godina.
- *Stream* svih pet partija (uz detaljne ekspertske komentare) možete naći na *DeepMind*-ovom *YouTube* kanalu.
- Ovo predavanje će biti sažeta verzija originalnog DeepMind naučnog rada: *"Mastering the game of Go with deep neural networks and tree search"*.

AlphaGo partija

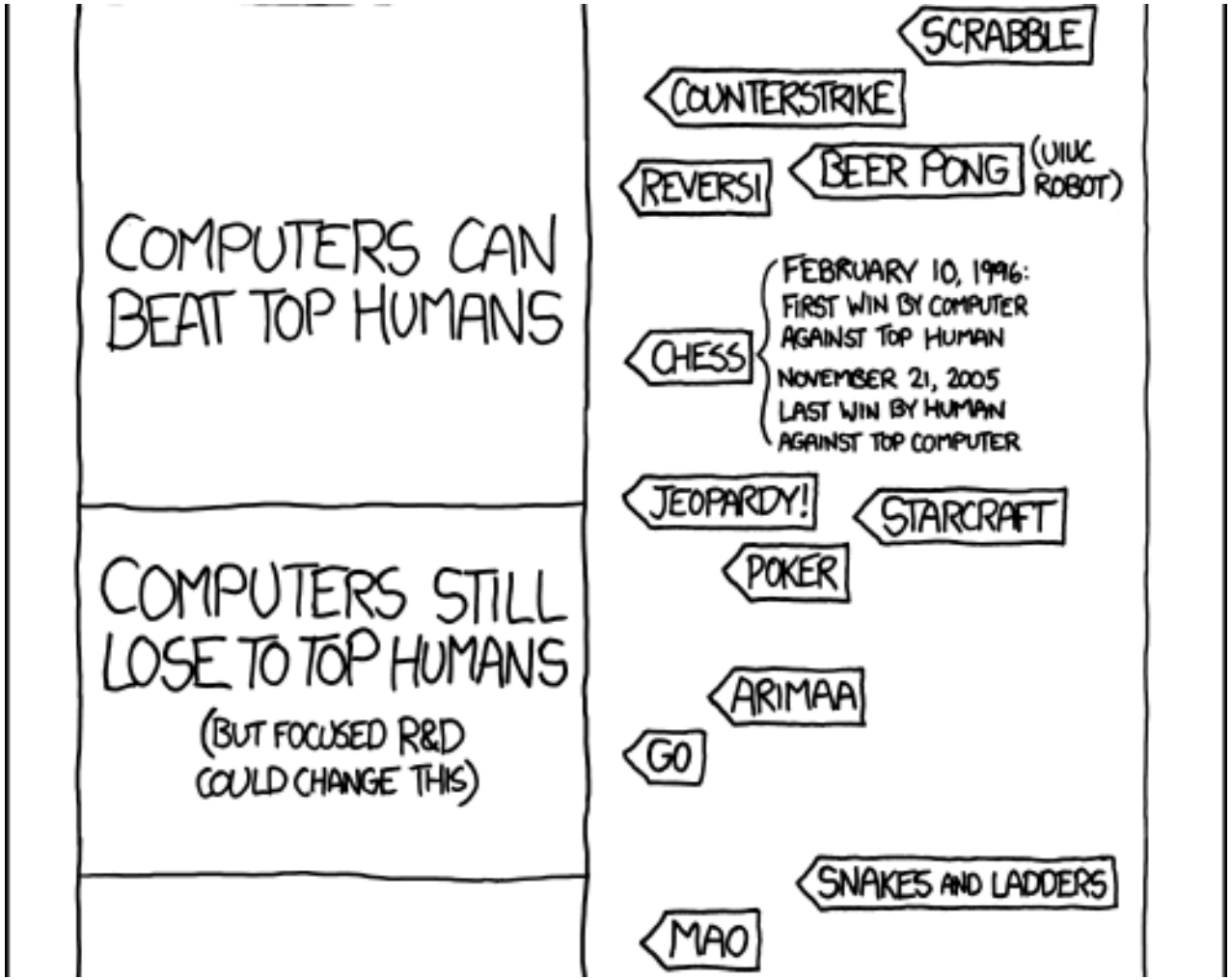
Stani malo...

- *Stvaran citat* od jednog mog rođaka:
  "Zašto se vi AI ljudi ponosite samo kad rešavate neke igrice, zašto ne rešite nešto korisno za promenu..."
- Ovo jeste dobro pitanje! Hajde da damo dobar odgovor.
- Igre su odlične za razvoj algoritama veštačke inteligencije, zato što imaju *jasno definisana pravila!* Stvarni svet je mnogo gori (a neretko i subjektivan...)
- Jedno od najvažnijih otkrića *AlphaGo* sistema je da je u eksponencijalno složenim prostorima (poput Go-a) moguće nadmašiti ljudske performanse *bez da se istražuje ikakav značajan deo tog prostora*!

Hijerarhija igara (xkcd.com/1002/)



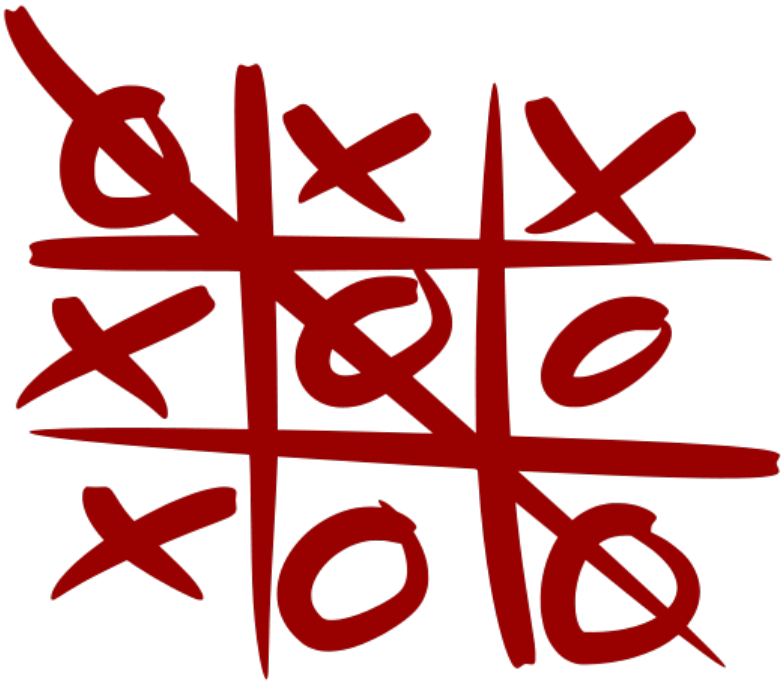
Hijerarhija igara (xkcd.com/1002/)



Hijerarhija igara (xkcd.com/1002/)

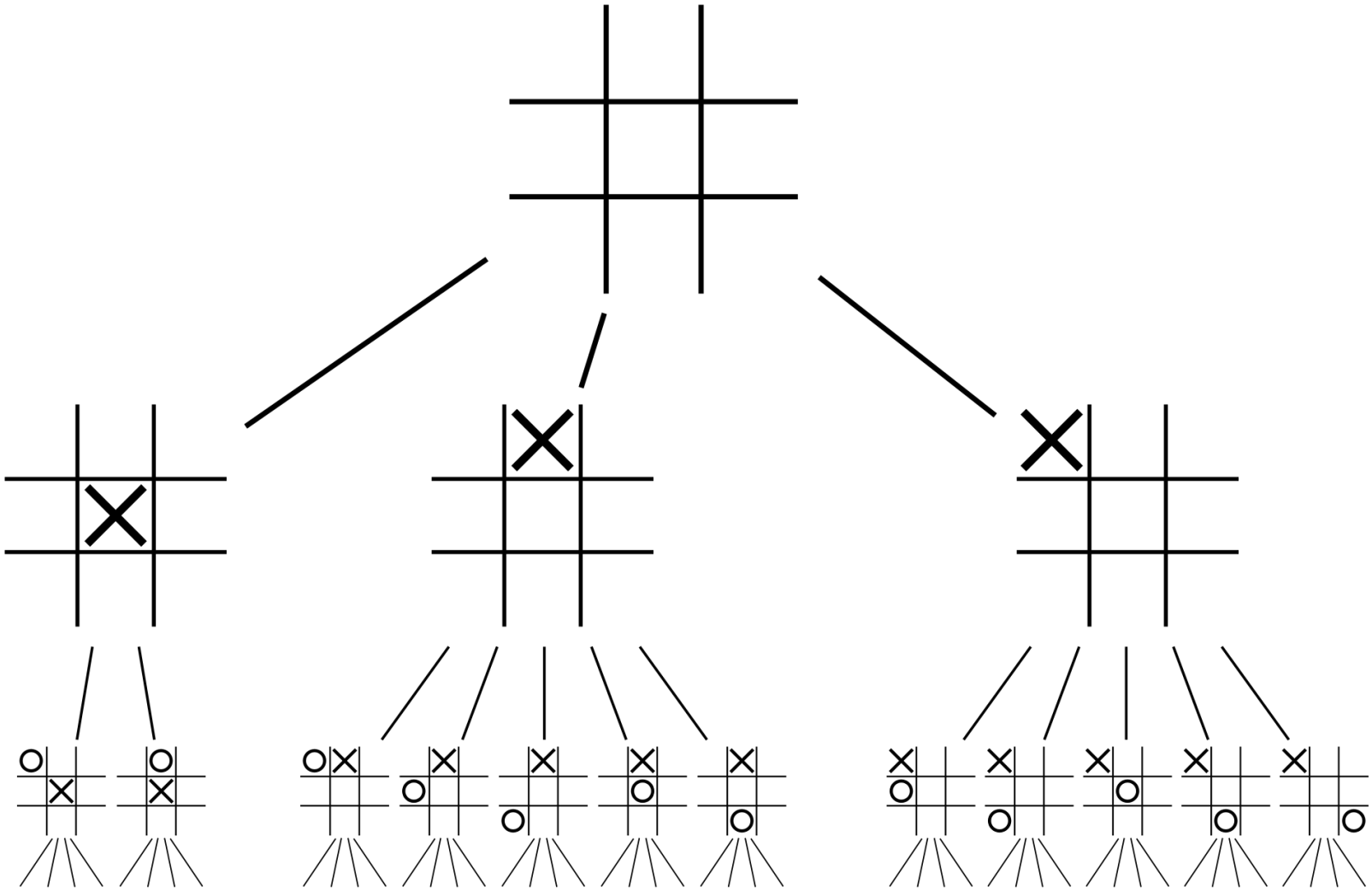
Iks-oks

- Svi ste (nadam se) upoznati sa tim kako ova igra funkcioniše.



- Postoji mali konačan broj **stanja** igre (ne više od $9! = 362880$) koja možemo da predstavimo čvorovima acikličnog usmerenog grafa, a grane ovog grafa su potezi.

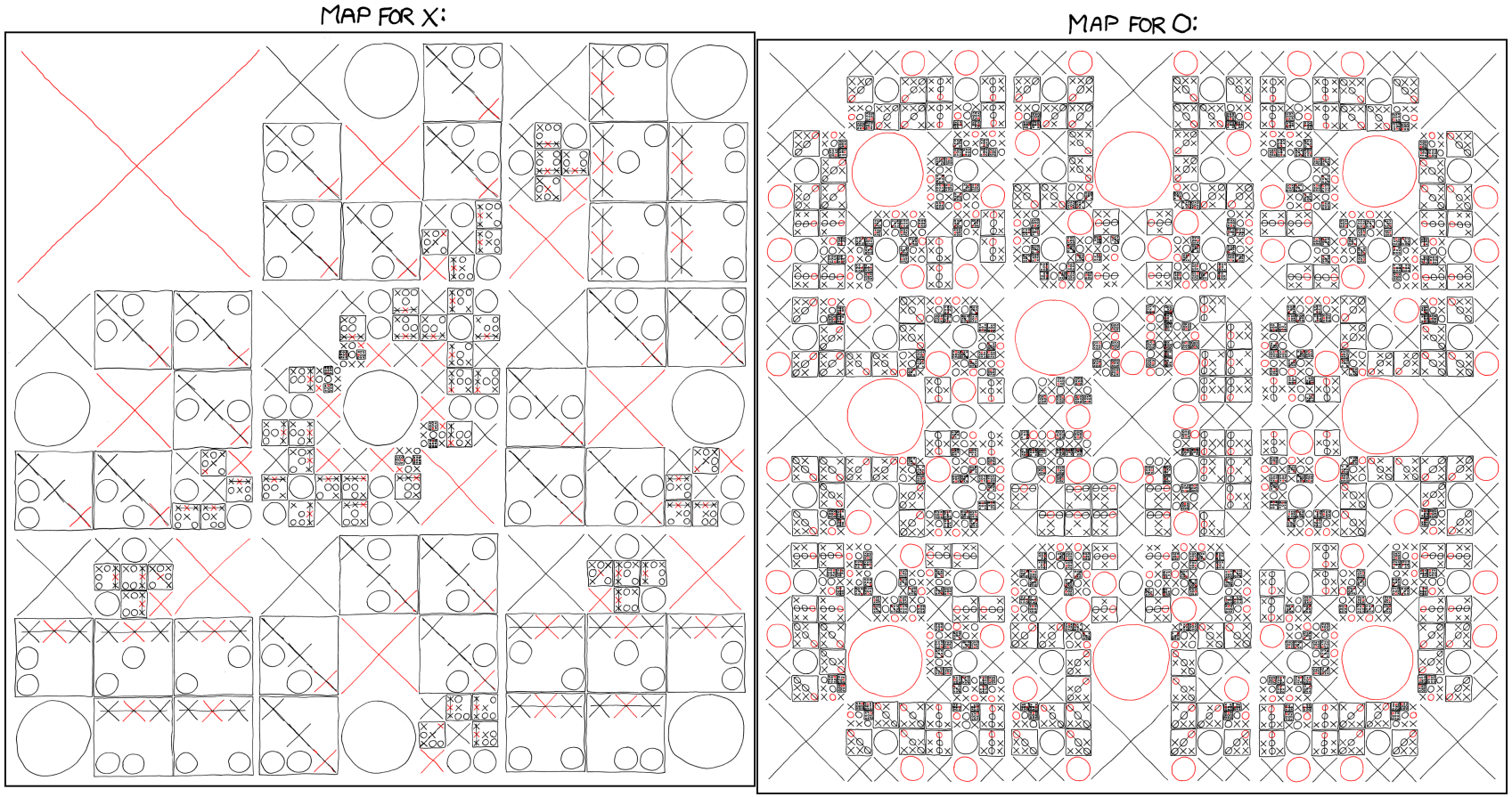
Prvih nekoliko mogućih poteza



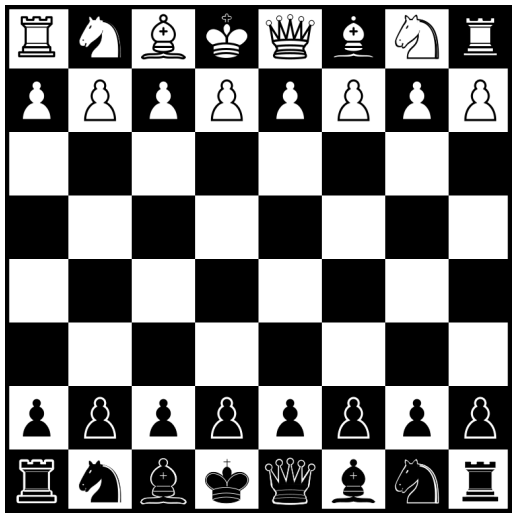
Potpuna pretraga

- Pošto mogućih stanja ima jako malo, možemo ih sve istražiti, i bektrekom odrediti koja stanja su pobednička za X, koja su gubitnička za X, a koja završavaju nerešenim rezultatom, kao i koje poteze treba povući iz svake pozicije.

Optimalna strategija (xkcd.com/832)



Malo o šahu



- Igra je stara ≈ 1500 godina.
- Istraživanja početka četrdesetih godina, prvi programi razvijani krajem pedesetih.
- Već sedamdesetih počeli da pobeđuju vrhunske igrače.
- 1996. godine *Deep Blue* igra šest partija protija Garija Kasparova sa rezultatom 4–2 (u korist Kasparova). Naredne godine Deep Blue pobeđuje rezultatom $3\frac{1}{2}$–$2\frac{1}{2}$.

Deep Blue

- Razvijan sedam godina od strane IBM-a.
- 256 procesora od po 120 MHz, rezultat je da se u jednoj sekundi obavi $1.138 \cdot 10^{10}$ floating point operacija.

Šta kaže IBM, kako ovo radi?

- Definišemo **funkciju procene** (*evaluation function*) eval koja daje "vrednost" svakom položaju i zavisi od sledećih parametara:
  - **materijalna vrednost**;
  - **poziciona vrednost**;
  - **vrednost bezbednosti kralja**;
  - **vrednost tempa**...
- Funkcije procene su česte u igrama ovog tipa, ali je njihovo smišljanje u opštem slučaju težak problem.

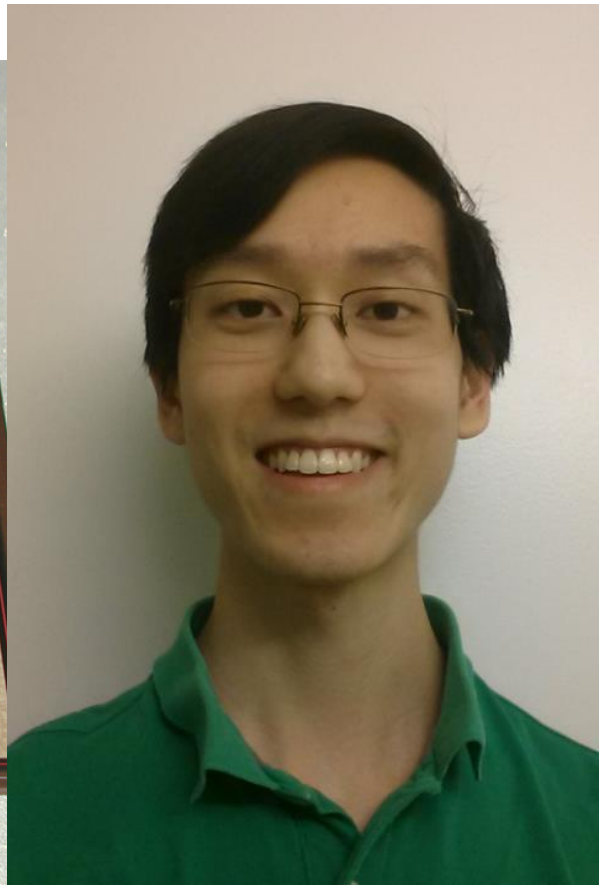
Šta sada sa eval?

- Primenimo sličan algoritam kao i za iks-oks, ali ne na celom stablu, već sa ograničenom dubinom. Na listovima ćemo izračunati funkciju procene, i onda na osnovu toga odrediti optimalne poteze iz svakog istraženog stanja.
- Međutim, čak i uz ovu aproksimaciju, na jednom savremenom računaru je neophodno $\approx 2\,\mathrm{min}$ za dubinu 3–4 poteza. Kako da pobedimo velemajstore?
- Problem je u tome što razmatramo dosta neoptimalnih stanja u algoritmu—ukoliko smo negde prethodno došli do rešenja koje trenutno razmatran potez u stablu ne može da nadmaši, možemo ignorisati celo njegovo podstablo.
- Ovo je algoritam α-β reza, i dovoljan je da se postigne velemajstorski nivo igre.

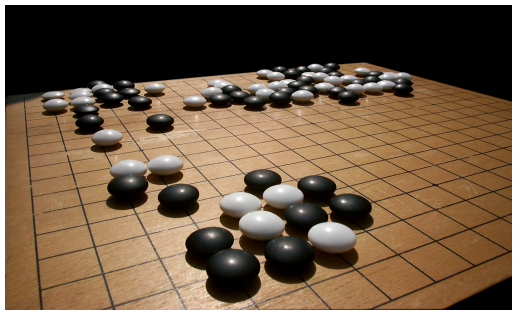
Arimaa

- Pomenućemo ukratko i *Arimaa* igru, za koju je prethodne godine prvi put napravljen AI koji je pobedio velemajstore.
- Program pod nazivom *Sharp* je dizajnirao *David Wu* (iz kompanije *Jane Street*, gde sam radio praksu).
- Igra je dizajnirana da može da se igra sa šahovskim figurama, da bude daleko sporija, i daleko vizuelnija (lakša za ljude da formiraju dugotrajniju strategiju, teža za računare da istražuju prostor).
- *Sharp*-ov algoritam koristi, efektivno, iste metode kao i za šah (α-β rez, sa jako pametnom evaluacijom trenutnog stanja).

Arimaa

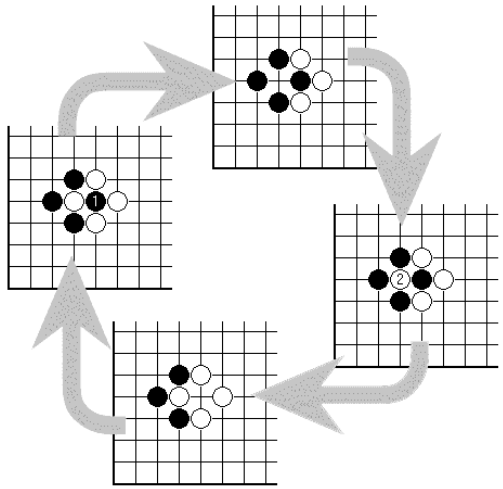


Pravila igre go



- Data je tabla sa 19 horizontalnih i 19 vertikalnih linija, na početku je prazna.
- Crni i Beli imaju žetone koje naizmenično stavljaju na tablu. Crni igra prvi.
- Žetoni se stavljaju na preseke linija.
- Igrač može da preskoči jedan potez bilo kada, ali mora da žrtvuje jedan svoj žeton.
- Igra se završava nakon dva uzastopna preskočena poteza, pri čemu je Beli taj koji završava igru.

Pravila igre go



- Teritorija nekog igrača podrazumeva broj preseka na kojima su žetoni tog igrača ili koji su žatvoreni" žetonima tog igrača.
- Pobeđuje igrač sa više teritorije.
- Ukoliko nakon poteza igrača A neka grupa žetona B postaje "potpuno okružena" žetonima A, ta grupa se sklanja sa table.
- **Superko pravilo**: ne sme da se ponovi nijedan položaj koji se javio ranije u toku igre.

Teškoće

- Tabla je mnogo veća nego u šahu, **eksponencijalno** više mogućnosti.
- Predstavljanje stanja je problematično, mora da se zadovolji superko pravilo.
- Vizuelna priroda igre: ljudima ovo ide bolje!
- Uprkos svemu ovome, *AlphaGo* je postigao naizgled nemoguće. **Kako?**

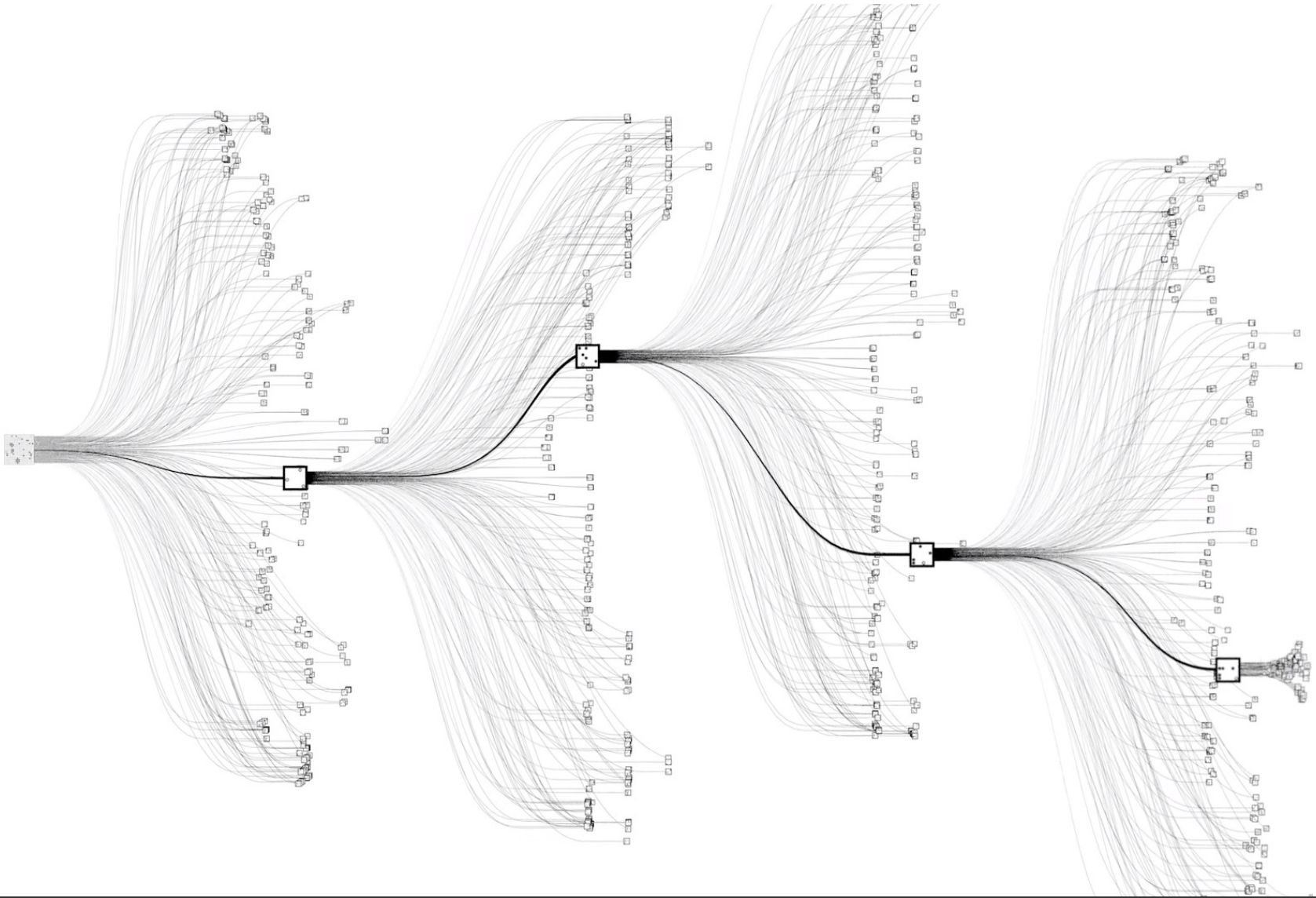
Monte karlo pretraga stabla

- Prethodno spomenut algoritam α-β reza nije dovoljno "agresivan" za potrebe igre poput Go. Neophodno je istražiti *daleko manje* potencijalnih poteza po dubini da bi se potez odigrao u potrebnim vremenskim ograničenjima!

- Tu na scenu stupaju **Monte karlo algoritmi**; na osnovu znanja koja smo sakupili do sada, odabiraćemo mali podskup poteza koji ćemo detaljnije istražiti, a sve ostale poteze ćemo *ignorisati*!

- Ovo, naravno, ne možemo raditi potpuno nasumično! Neophodno je imati neku vrstu *funkcije* koja će nam, za neko stanje table, reći koji potezi su (verovatno) bolji od drugih...

MCTS



Još neke pomoćne funkcije

- Kada odaberemo manji podskup poteza, onda možemo te poteze detaljnije (dublje) istraživati! Ukoliko su nam procene vrednosti poteza dobre, za očekivati je i da će ceo algoritam da se ponaša dobro!
- Go može da traje jako dugo—u većini slučajeva ne možemo sa velikom sigurnošću da simuliramo igru do kraja (mnogo potencijalnih poteza se ne razmatraju), tako da glavni deo pretrage uglavnom zaustavljamo na nekoj poziciji. Neophodna nam je, kao i za šah, funkcija koja daje vrednost ove pozicije.
- Međutim, često je korisno dobiti bar *grubu* procenu šta bi se desilo ako bismo igrali do kraja—za ovo možemo pripremiti *još jednu funkciju*, koja će brzo da odredi poteze za oba igrača.

TODO: funkcije

Da rezimiramo; potrebno je pripremiti *tri* funkcije koje će navoditi pretragu stabla. Za skup stanja table S i skup poteza A:

- **Brza polisa**: $p_\pi : S \to A$. "Za datu tablu s, **brzo** odaberi neki (otprilike) dobar potez".
- **(Standardna) polisa**: $p_\rho : S \to (A \to \mathbb{R})$. "Za datu tablu s, dodeli *relativne vrednosti* svakom potezu".
- **Funkcija procene**: $v_\theta : S \to \mathbb{R}$. "Za datu tablu s, odredi njenu vrednost (tj. verovatnoću da će igrač na potezu pobediti iz te pozicije)".

Odakle početi?

- Iako deluje kao da je problem sada jednostavniji, imamo još *mnogo* posla pre nego što izvedemo ove tri funkcije.

- Da bismo mogli da ovo da uradimo, neophodno je da upotrebimo metode **učenja sa pojačanjem** (*reinforcement learning*), koje su poslednjih godina jako popularne (*DeepMind Atari player*).

Učenje sa pojačanjem

- Pretpostavljamo vrlo generalne uslove: nalazimo se u svetu sa stanjem s iz nekog skupa stanja S (u ovom slučaju, S je skup svih mogućih stanja Go table), i možemo povući neki potez iz skupa poteza A.
- Takođe postoje i dve funkcije: $\mathcal{S} : S \times A \to S$ i $\mathcal{R} : S \times A \to \mathbb{R}$, kojima nemamo pristup.
- Nakon izvršenog poteza a u stanju s, prelazimo u novo stanje $\mathcal{S}(s, a)$, a dobijamo i *nagradu* $\mathcal{R}(s, a)$.
- Naša želja je da biramo poteze tako da *maksimizujemo* nagradu koju ćemo dobiti.

Kumulativna nagrada

- Osim toga što želimo da maksimizujemo nagradu, želimo i da nam nagrade stignu *što pre* (tj. dajemo veću važnost skorijim nagradama).
- Ovo radimo tako što definišemo faktor $\gamma \in [0, 1)$ (*faktor popusta*), kojim skaliramo nagrade u budućnosti:

$$V^p(s) = \sum_{k=1}^{+\infty} \gamma^{k-1} r_k = r_1 + \gamma r_2 + \gamma^2 r_3 + \dots$$

  gde je r_k nagrada dobijena posle k poteza, počevši iz stanja s, prateći polisu p.
- Ovako definisana nagrada je neophodna, između ostalog, i da bi algoritam *konvergirao* (ali nećemo ulaziti u te detalje danas).

$\mathcal{Q}$ funkcija

- Želimo da nađemo optimalnu polisu $p_{\text{opt}} : S \to A$, gde je $p_{\text{opt}}(s) = \arg\max_p \{V^p(s)\}$ za dato startno stanje s. Uvedimo i oznaku $V_{\text{opt}} = V^{p_{\text{opt}}}$.
- Definišimo funkciju $\mathcal{Q}$, gde $\forall s \in S, a \in A$:

$$\mathcal{Q}(s,a) = \mathcal{R}(s,a) + \gamma V_{\text{opt}}(\mathcal{S}(s,a))$$

- Značenje $\mathcal{Q}(s,a)$ odgovara nagradi koju dobijamo ako izvršimo potez a iz stanja s i nakon toga *igramo optimalno*.
- Naučiti $\mathcal{Q}$ je dovoljno za ustanovljavanje optimlne polise jer:

$$p_{\text{opt}}(s) = \arg\max_a \{\mathcal{Q}(s,a)\}$$

Nagoveštaj iterativnog postupka

- Primetimo $V_{\text{opt}}(s) = \max_\alpha \{\mathcal{Q}(s, \alpha)\}$.
- Stoga, kada ovo uvrstimo u definiciju $\mathcal{Q}$:

$$\mathcal{Q}(s, a) = \mathcal{R}(s, a) + \gamma \max_\alpha \{\mathcal{Q}(\mathcal{S}(s, a), \alpha)\}$$

- Ovo nagoveštava da, ako je $\mathcal{Q}'$ **trenutna procena** za $\mathcal{Q}$, da će

$$\mathcal{R}(s, a) + \gamma \max_\alpha \left\{\mathcal{Q}'(\mathcal{S}(s, a), \alpha)\right\}$$

biti bolja procena za $\mathcal{Q}(s, a)$ od $\mathcal{Q}'(s, a)$. Pri tome je $\mathcal{Q}'$ tabela aktuelnih procena za $\mathcal{Q}$ za sve elemente $S \times A$.

Algoritam $\mathcal{Q}$-učenja

Na početku je $\mathcal{Q}'$ tabela popunjena nasumično odabranim vrednostima.

1. Trenutno stanje je s, isprobaj neki potez a.
2. Uradi a, sleduje nagrada $\mathcal{R}(s, a)$.
3. Posmatraj novo stanje $\mathcal{S}(s, a)$.
4. Ažuriranje:
$$\mathcal{Q}'(s, a) \leftarrow \mathcal{R}(s, a) + \gamma \max_{\alpha} \left\{ \mathcal{Q}'(\mathcal{S}(s, a), \alpha) \right\}$$
5. Idi na 1.

Šta smo postigli?

- Ukoliko naučimo $\mathcal{Q}$ funkciju za sve parove stanja i poteza, imamo sve što nam treba da formiramo željene polise za pretragu stabla! (uz određene suptilnosti)
  - *Brza polisa* podrazumeva računanje $\mathcal{Q}$ funkcije koja se može brže izračunati, ali je manje precizna, pa birati potez koji je maksimizuje.
  - *Standardna polisa* direktno vraća sve $\mathcal{Q}$ vrednosti za neko stanje.
  - *Funkcija procene* vraća maksimalnu $\mathcal{Q}$ vrednost za neko stanje.

- **Veliki problem**: *ne možemo čuvati celu $\mathcal{Q}$ tabelu u memoriji!*

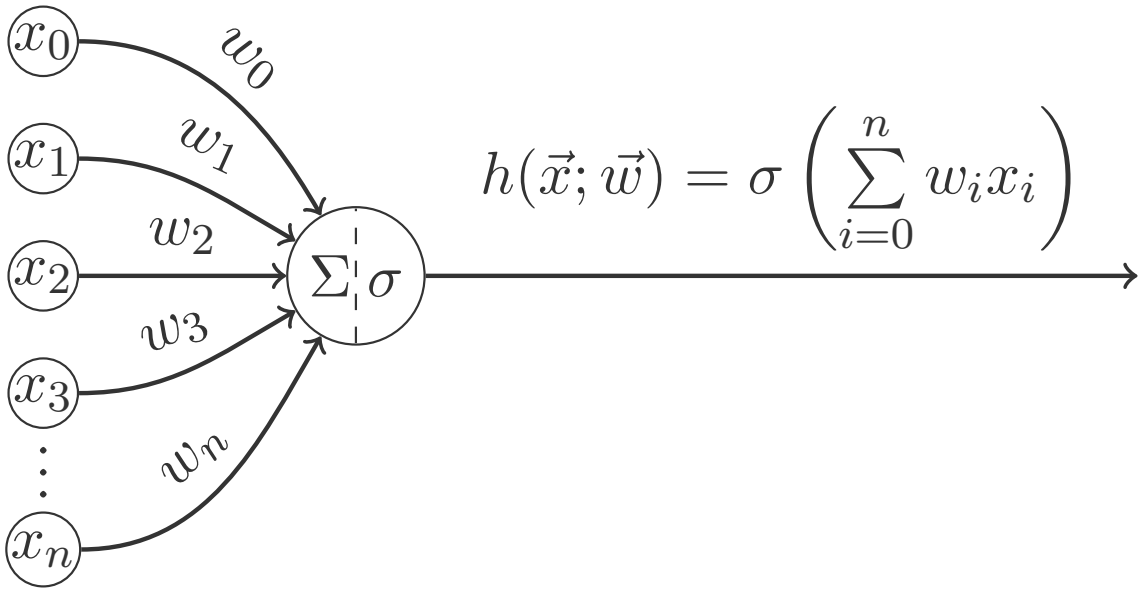
- Moramo nekako da je **aproksimiramo**...

Neuralne mreže

- Jedan vrlo popularan model za aproksimaciju Q funkcije su *neuralne mreže*—strukture povezanih procesirajućih jedinica (*neurona*) koje su sposobne da vrlo precizno nauče neku funkciju.
- Svaki neuron računa neku linearnu kombinaciju svojih *ulaza* i nad time primenjuje neku *aktivacionu funkciju* da bi dobio odgovarajući izlaz.
- Daleko dublji uvod u neuralne mreže će biti na *trećoj Nedelji informatike* :).

Jedan neuron

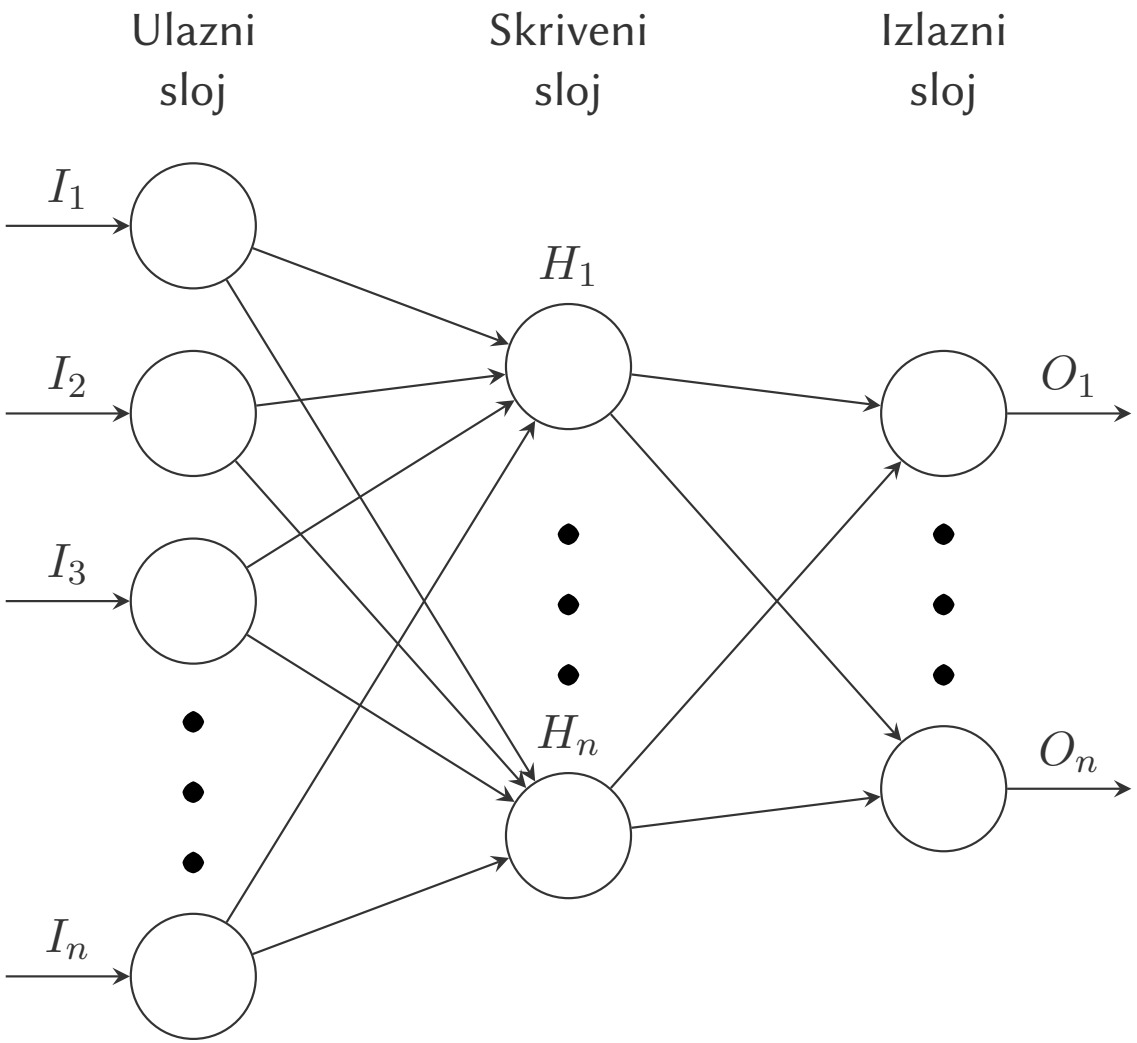
U ovom kontekstu često nazivan i *perceptronom* (...)



Česti izbori aktivacione funkcije σ:

- $\sigma(x) = x$ (*identitet*);
- $\sigma(x) = \max(0, x)$ (*ReLU*);
- $\sigma(x) = \frac{1}{1+\exp(-x)}$ (*logistička funkcija*).

Neuralna mreža



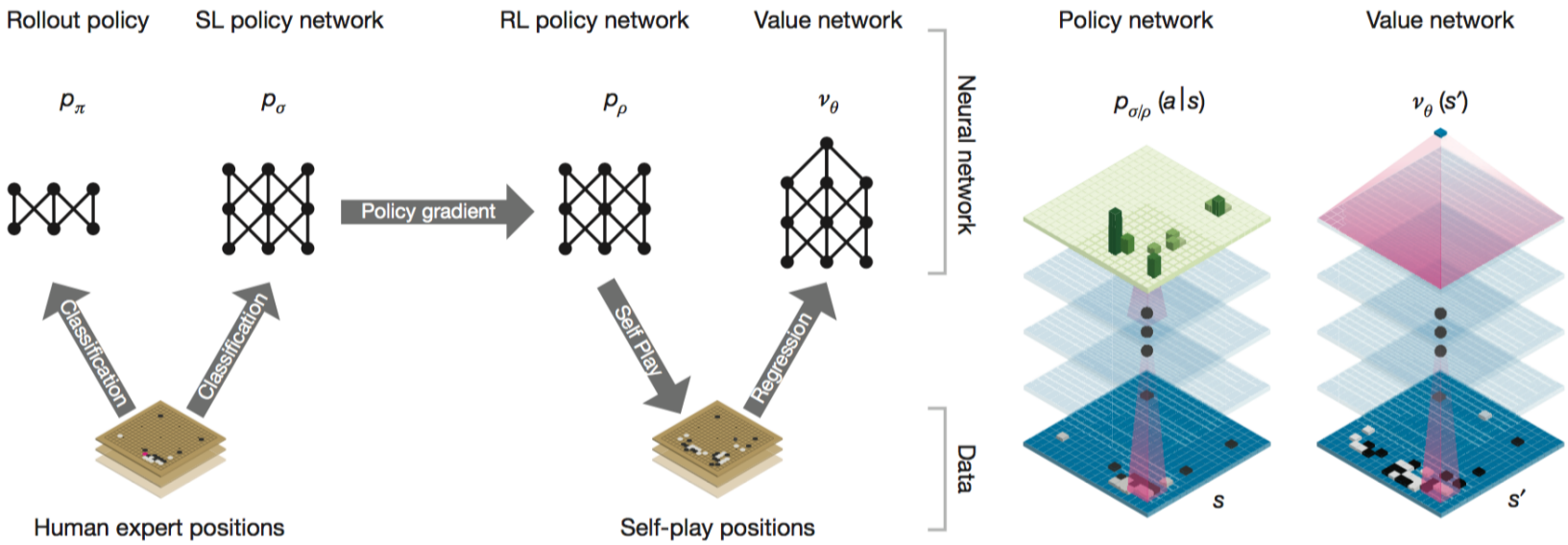
Par zanimljivosti

- Neuralne mreže se treniraju tako što im dajemo primere ulaza i izlaza, pomoću kojih ona uči ulazne težine za svaki neuron—*backpropagation* je često korišćen algoritam.
- Dokazano je (matematički) da ako koristimo funkciju poput logističke za svaki neuron i imamo samo jedan skriveni sloj, možemo aproksimirati **svaku funkciju** koliko god precizno hoćemo!
- Međutim, ovaj dokaz nije konstruktivan (ne daje metodu za trening), tako da se često pribegava dodavanju više od jednog sloja (*deep learning*).

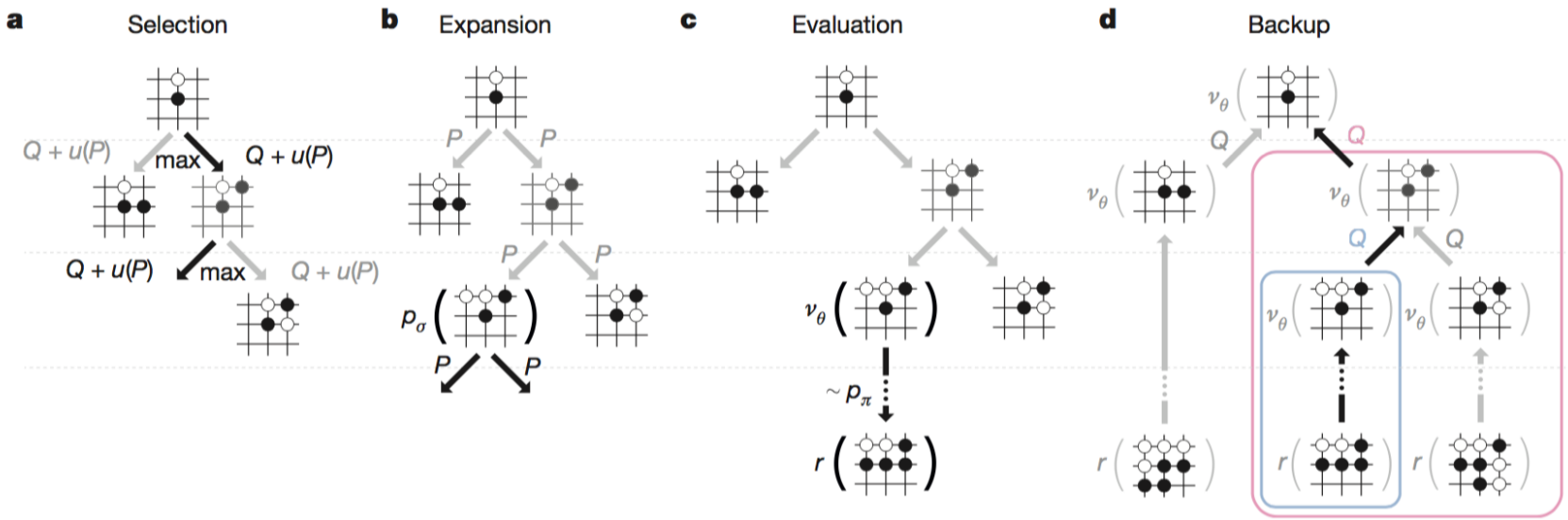
Sastavimo sve zajedno...

- Ulazi za neuralne mreže koje *AlphaGo* koristi su stanja Go table, a izlazi su Q vrednosti za sve moguće poteze.
- Brza polisa koristi *pliću* neuralnu mrežu (brže se računaju izlazi) od standardne.
- *AlphaGo* je prvo naučio neke početne vrednosti Q funkcije gledajući ekspertske partije, a nakon toga se dalje usavršavao *samo igrajući sam protiv sebe*!
- Sledi nekoliko slika iz originalnog rada koje opisuju ove procese...

Treniranje mreža



Monte karlo pretraga



Hvala na pažnji!