Deep Learning

13 - AlphaZero

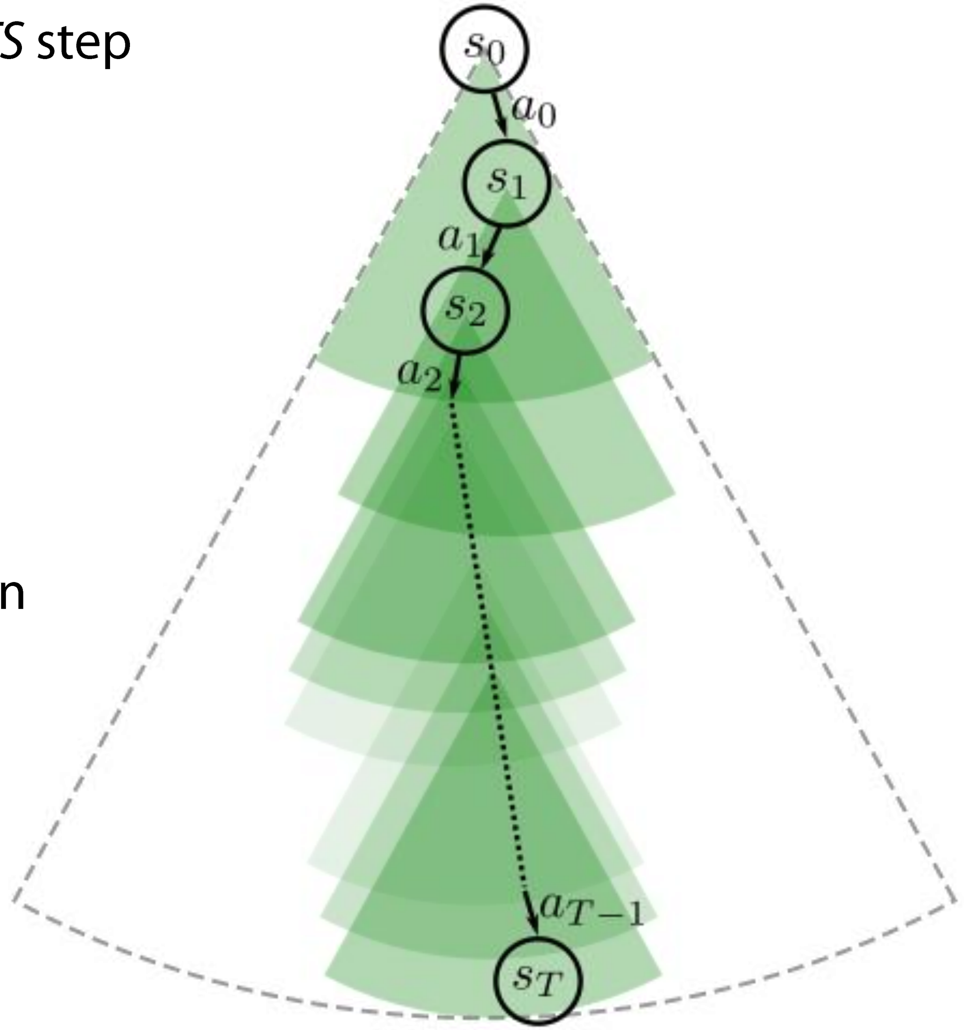
Marco Piastra

This presentation can be downloaded at:
http://vision.unipv.it/DL

AlphaZero =MCTS + DNN

Monte Carlo Tree Search (MCTS) method

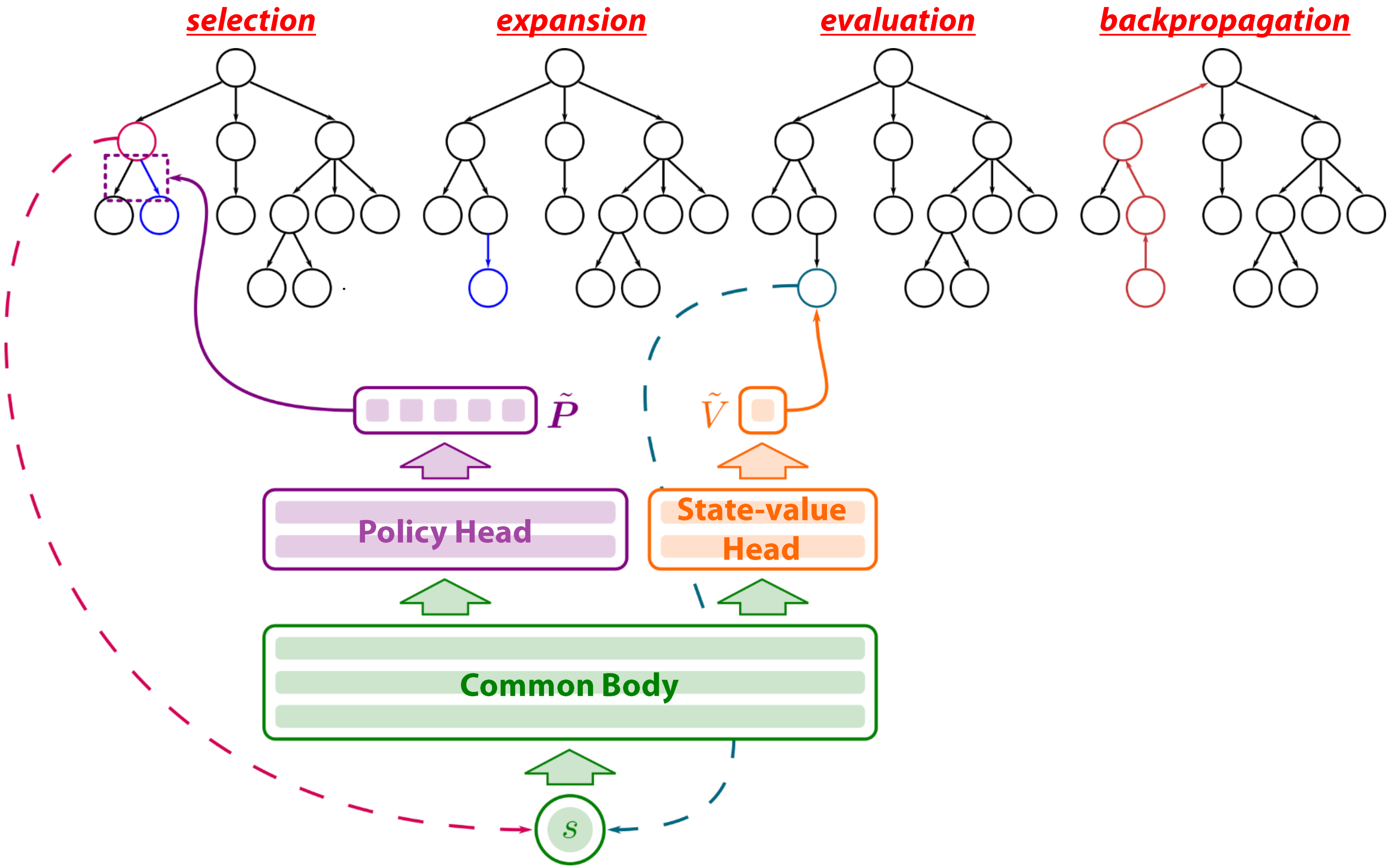
- ***MCTS*** *method:*
  - *memory* of past playouts in a single *MCTS* step
    (collected in the tree statistics)
  - *knowledge transfer* between *MCTS* steps
    (by reusing subtrees already explored)
  - optimal policy only *partially* defined
    (on actually computed states)
  - *intrinsically stochastic* policy optimization
    (the same initial state
    can give rise to different optimizations)
  - What about *knowledge transfer*
    between *MCTS episodes?*
    transferring the entire MCTS tree
    would rapidly cause its explosive growth…



Knowledge transfer between MCTS episodes

- ***AlphaZero*** [Silver et al. 2017]
  - *Monte Carlo Tree Search (MCTS):*
    improves the policy by focusing on the most promising actions
  - *Deep Neural Network (DNN):*
    learns the improved policy and transfers it between MCTS episodes

- ***AlphaZero = MCTS + DNN***



- **DNN in AlphaZero**
  - *input:* a *state* s
  - *output:* a *probability distribution* $\tilde{\boldsymbol{P}}(s) := [\tilde{P}(a \mid s)]_{a \in \mathcal{A}(S)}$

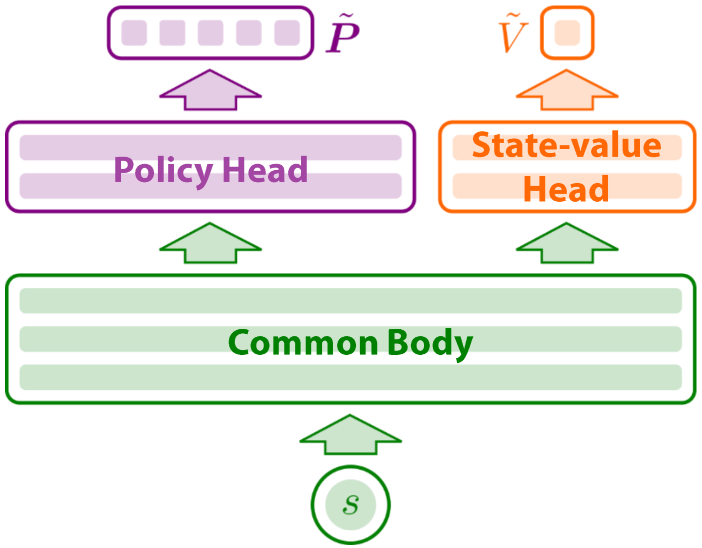
    stochastic policy (a vector of probabilities)

    and a *state-value* $\tilde{V}(s)$

    predicts the expected reward for state s

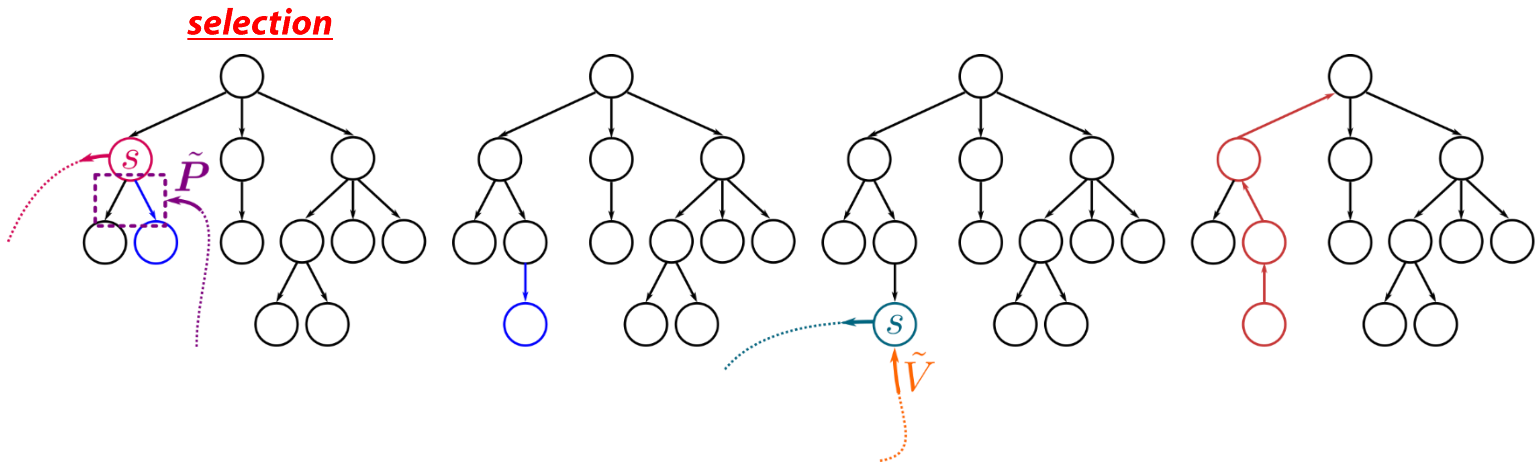
    acts as an ***actor-critic*** in the *training* of ***parameters*** ϑ of the net

    $\tilde{V}$ is compared with the *actual* reward r, which also impacts on training $\tilde{\boldsymbol{P}}$ by *backpropagating* through the *Common Body*



"Y" shape

- ***MCTS step*** **in** ***AlphaZero***



- *selection:* *UCT* policy is replaced with ***PUCT*** *("Predictor" + UCT)*

MCTS estimation of $Q(s,a)$ for *DNN* policy

DNN policy

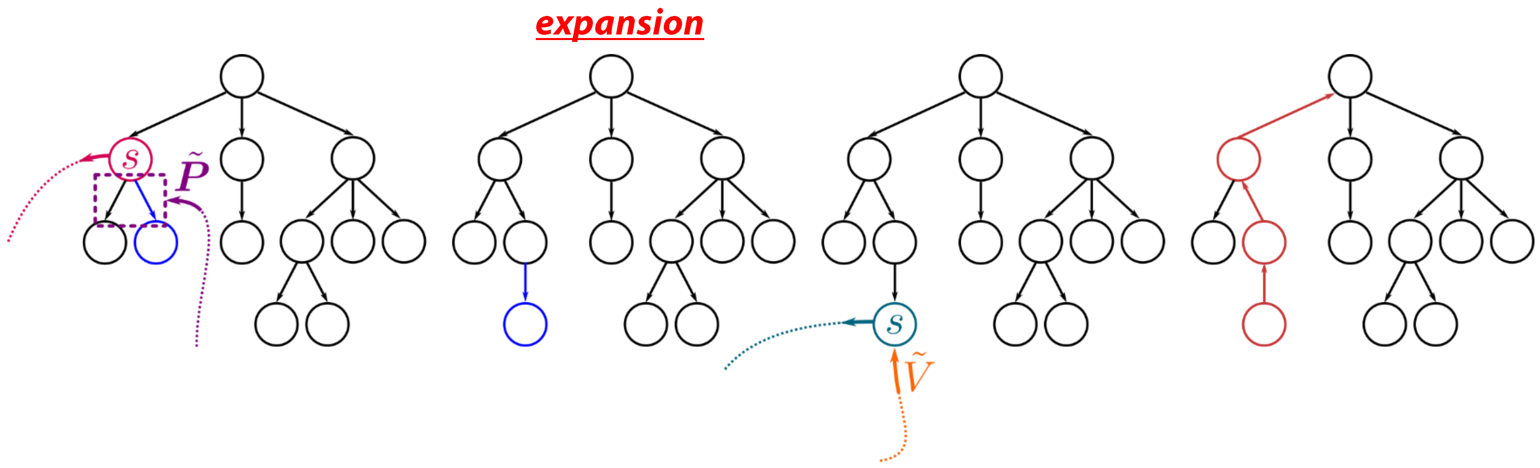
$$\pi^{\text{PUCT}}(s) := \operatorname*{argmax}_a \left\{ \hat{Q}(s,a) + c(s)\tilde{P}(a \mid s)\frac{\sqrt{N(s)}}{N(s,a)+1} \right\}$$

exploration rate $c(s) := \log \frac{1 + N(s) + c_{\text{base}}}{c_{\text{base}}} + c_{\text{init}}$

(slowly grows with search time)

avoids division by 0

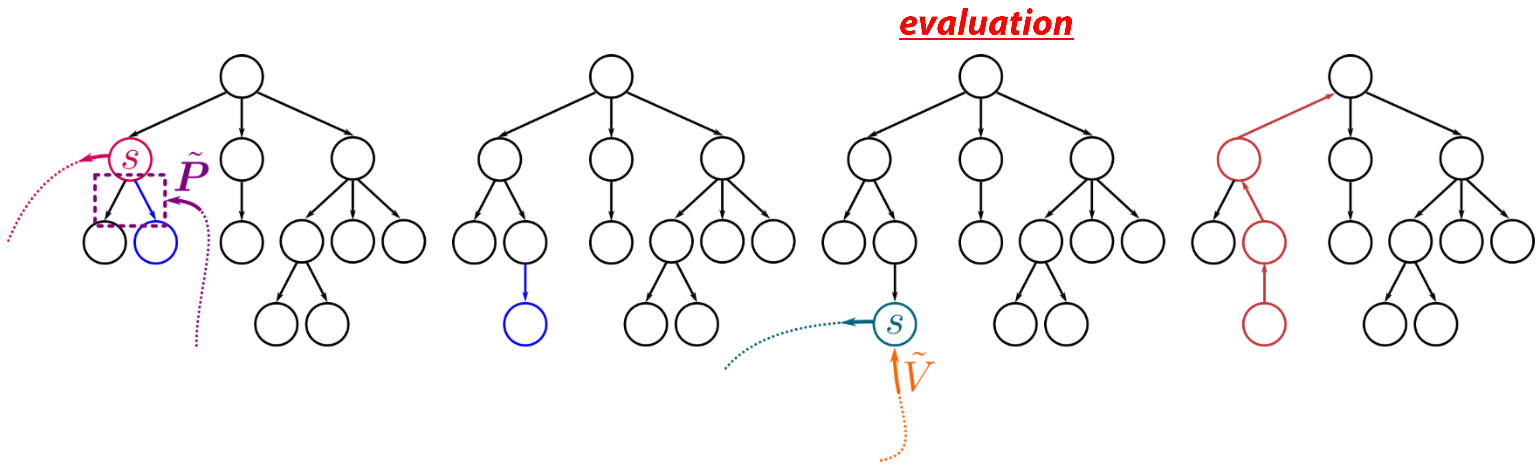
- ***MCTS step* in *AlphaZero***



- *expansion:* initialization of the leaf new node s_L:

$$N(s_L) := 0 \quad \text{and} \quad \forall\, a \in \mathcal{A}(s_L) \qquad N(s_L, a_L) := 0, \quad \hat{Q}(s_L, a_L) := -\infty$$

- ***MCTS step*** **in** ***AlphaZero***

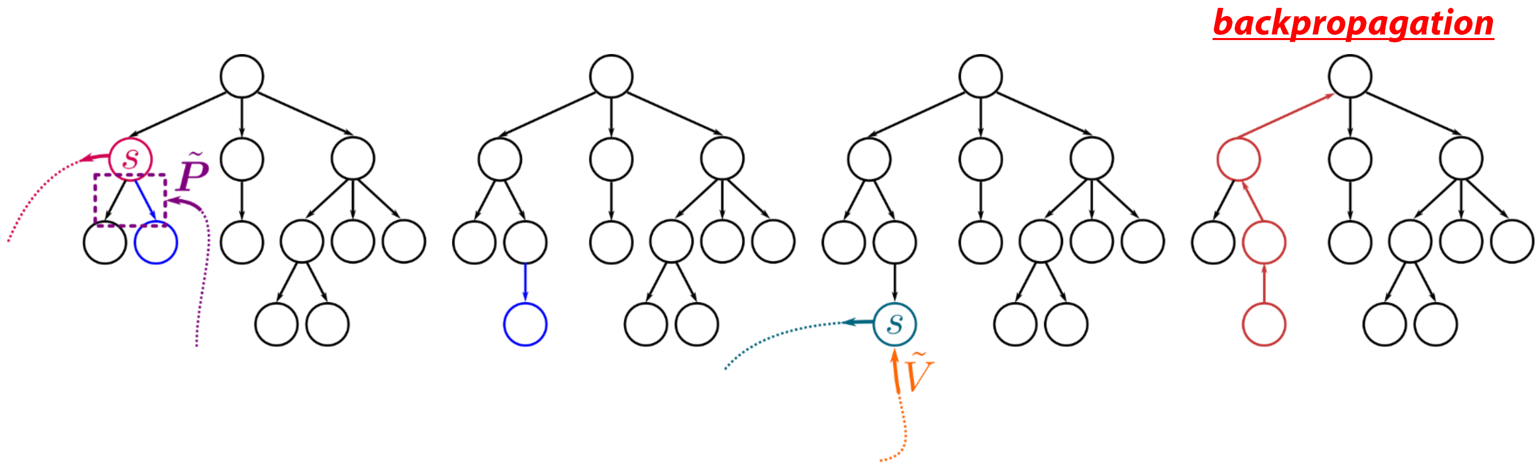


- *expansion:* initialization of the leaf new node s_L:

$$N(s_L) := 0 \quad \text{and} \quad \forall\, a \in \mathcal{A}(s_L) \qquad N(s_L, a_L) := 0, \quad \hat{Q}(s_L, a_L) := -\infty$$

- *evaluation* (in place of *simulation*): expected reward is $\tilde{V}(s_L)$

- **_MCTS step_ in _AlphaZero_**



- ***expansion:*** initialization of the leaf new node s_L:

$$N(s_L) := 0 \quad \text{and} \quad \forall\, a \in \mathcal{A}(s_L) \qquad N(s_L, a_L) := 0, \quad \hat{Q}(s_L, a_L) := -\infty$$

- ***evaluation*** (in place of ***simulation***): expected reward is $\tilde{V}(s_L)$
- ***backpropagation:*** for each state s and action a visited in selection/expansion:

$$\begin{aligned} N(s) &:= N(s) + 1, \\ N(s,a) &:= N(s,a) + 1 \end{aligned} \quad \text{and} \quad \hat{Q}(s,a) := \hat{Q}(s,a) + \frac{\boxed{\tilde{V}(s_L)} - \hat{Q}(s,a)}{N(s,a)}$$

MCTS step in AlphaZero: policies

- _Selection_ policy: ***PUCT***

$$\pi^{\text{sel}}(s) := \pi^{\text{PUCT}}(s) := \operatorname*{argmax}_a \left\{ \hat{Q}(s,a) + c(s)\tilde{P}(a \mid s)\frac{\sqrt{N(s)}}{N(s,a)+1} \right\}$$

- _Output_ policy:

$$\pi^{\text{out}}(s) \sim \left[\hat{P}(a \mid s) := \frac{N(s,a)}{N(s)} \right]_{a \in \mathcal{A}(s)}$$

taking frequencies as probabilities
(in place of their argmax as output action)
ensures ***exploration***

(the simulation policy does not exist anymore)

DNN training in AlphaZero

- ***Data items*** from a single *MCTS episode:*

After an *MCTS episode* $\mathcal{E} := \langle s_0, a_0, s_1, \ldots, a_{T-1}, s_T \rangle$
with actual reward $\hat{V}^{\mathcal{E}} := r(s_T)$:

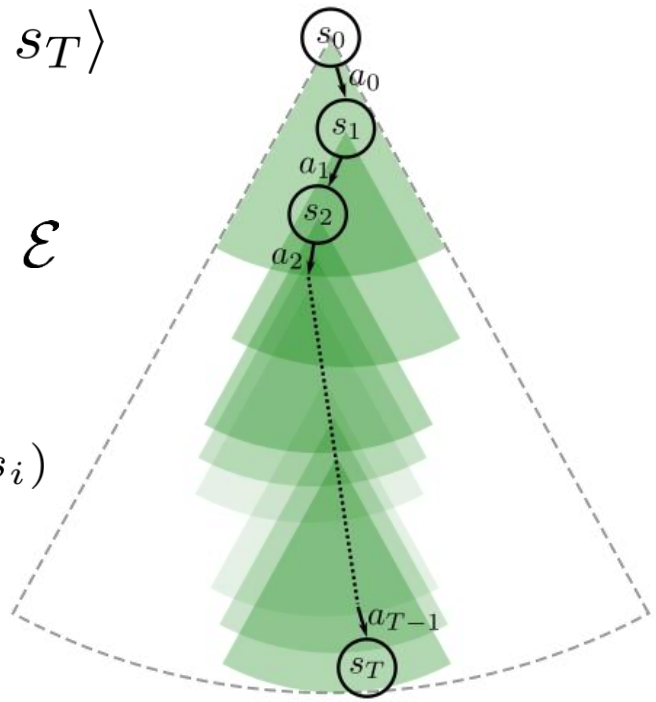
- for each *non-terminal* state $s_i \;\; (i = 0 \ldots T-1)$ in $\mathcal{E}$

$$\hat{\boldsymbol{P}}(s_i) := \left[\hat{P}(a \mid s_i) := \frac{N(s_i, a)}{N(s_i)} \right]_{a \in \mathcal{A}(s_i)}$$

vector of frequencies

- the ***output*** of $\mathcal{E}$ is

$$D^{\mathcal{E}} := \Big\{ \underbrace{\langle s_i, \hat{\boldsymbol{P}}(s_i), \hat{V}^{\mathcal{E}} \rangle}_{\text{data item}} \Big\}_{i=0\ldots T-1}$$



DNN training in AlphaZero

- ***Iteration:***

K times $\begin{cases} \text{1) play one } \textit{MCTS episode } \mathcal{E}_j \\ \text{2) collect data items } D^{\mathcal{E}_j} \end{cases}$

3) train the parameters of the *DNN* by using as ***dataset***

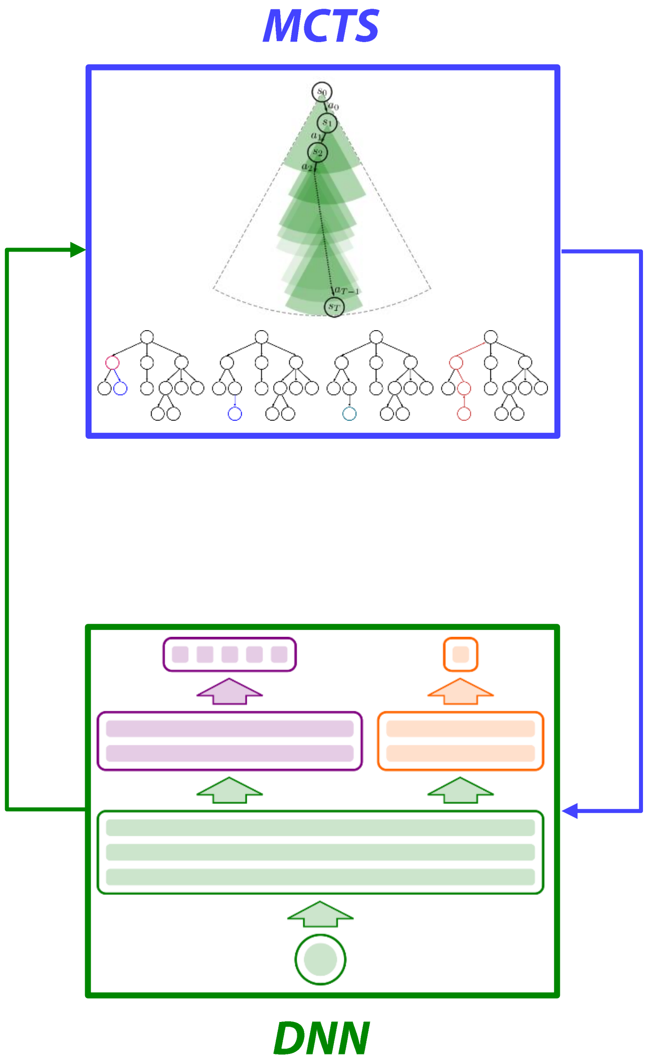
$$D := \bigcup_{j=1}^{K} D^{\mathcal{E}_j}$$

- In the limit of *infinite* iterations:

$$\pi^{\mathrm{DNN}}(s) := \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \, \tilde{P}(a \mid s) \to \pi^*(s) \quad \forall\, s$$

- ***AlphaZero:***
  - *memory* of past playouts in a single *MCTS* step
    (collected in the tree statistics)
  - *knowledge transfer* between *MCTS* steps
    (by reusing subtrees already explored)
  - *knowledge transfer* between *MCTS* episodes
    (provided by DNN)
  - *deterministic* policy optimization
    with policy defined for all states s :

$$\pi^{\mathrm{DNN}}(s) := \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}}\, \tilde{P}(a \mid s)$$



AlphaZero

in Continuous Spaces

Continuous Action Spaces

- *What happens when the space $\mathcal{A}(s)$ of admissible actions is continuous?*
  - How to compute the deterministic *policy optimization* in practice?

$$\pi^{\mathrm{DNN}}(s) = \boxed{\operatorname*{argmax}_{a \in \mathcal{A}(s)}} \tilde{P}(a \mid s)$$

it could be a high-dimensional space

continuous and analytic, but in general with a lot of (local) maxima

  - How to initialize (and deal with) a *new node* s in the MCTS *expansion* phase?

    Standard initialization requires:

$$\boxed{\forall\, a \in \mathcal{A}(s)} \qquad N(s,a) := 0, \quad \hat{Q}(s,a) := +\infty$$

each admissible action is initialized

each admissible action will be evaluated at least once

Cross-Entropy Maximization (CEM)

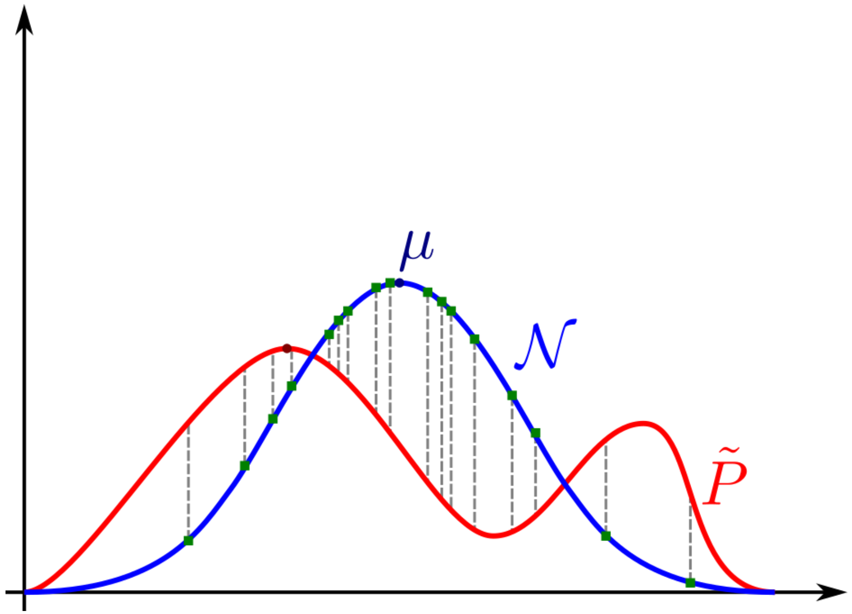
- ***CEM Method:***

1) choose *at random* initial values $\mu, \sigma \in \mathbb{R}^d$

2) sample m actions from

normal probability distribution $\mathcal{N}(\mu, \mathrm{diag}(\sigma))$ *mean* *variances (diagonal matrix)*

3) evaluate $\left\{\tilde{P}(a_i \mid s)\right\}_{i=1}^{m}$



- ***CEM Method:***

1) choose *at random* initial values $\mu, \sigma \in \mathbb{R}^d$

2) sample m actions from

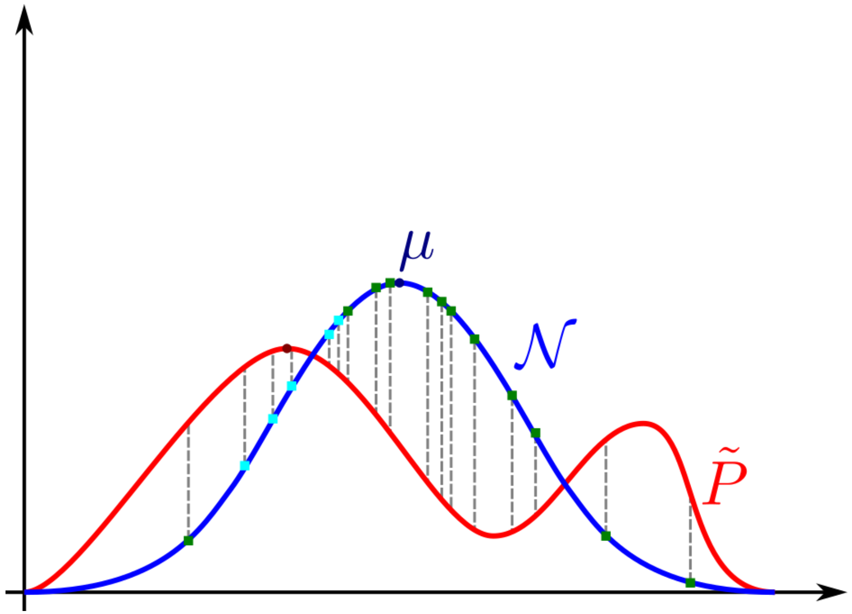
mean

variances (diagonal matrix)

normal probability distribution $\mathcal{N}(\mu, \mathrm{diag}(\sigma))$

3) evaluate $\left\{\tilde{P}(a_i \mid s)\right\}_{i=1}^{m}$

4) select $k < m$ actions with *highest probability*



Cross-Entropy Maximization (CEM)

- ***CEM Method:***

1) choose <u>*at random*</u> initial values $\mu, \sigma \in \mathbb{R}^d$

2) <u>sample</u> m actions from

mean

variances (diagonal matrix)

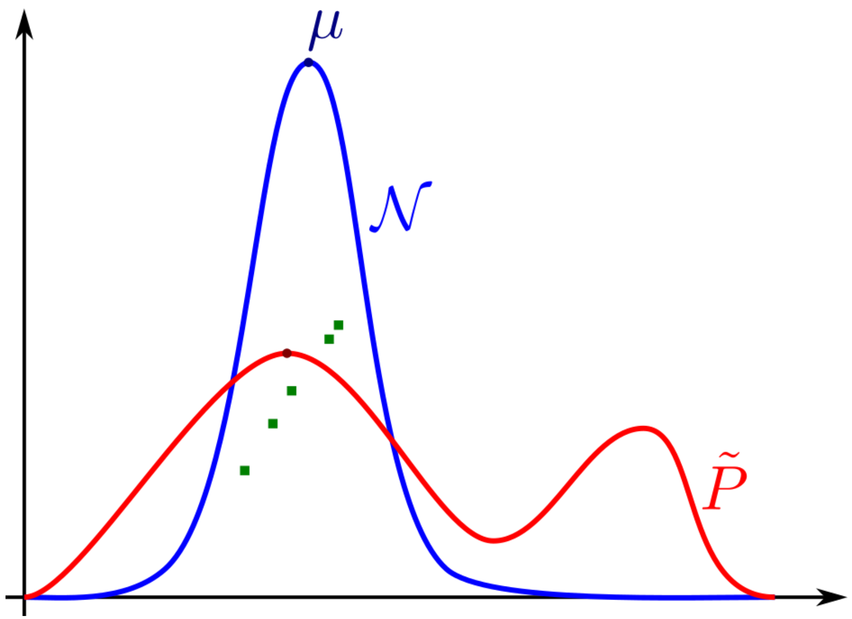
<u>normal</u> probability distribution $\mathcal{N}(\mu, \mathrm{diag}(\sigma))$

3) evaluate $\left\{\tilde{P}(a_i \mid s)\right\}_{i=1}^{m}$

4) select $k < m$ actions with *highest probability*

5) fit new μ, σ

6) if terminated, return μ otherwise go to 2)



Progressive Widening (PW)

- ***Progressive Widening (PW)*** *of action space* $\mathcal{A}(s)$ [Chaslot et al., 2007]:
  - For any *new node* s created in the MCTS *expansion* phase
    1. initialize $\mathcal{A}(s) := \{a_1, \ldots, a_k\}$ with k admissible actions by ***sampling*** the ***probability*** $\tilde{P}(a \mid s)$ (given by the DNN)
    2. initialize the statistics for each action $a \in \mathcal{A}(s)$ as usual:

$$N(s,a) := 0, \quad \hat{Q}(s,a) := +\infty$$

Progressive Widening (PW)

- ***Progressive Widening (PW)*** *of action space* $\mathcal{A}(s)$ [Chaslot et al., 2007]:
  - For any *new node* s created in the MCTS *expansion* phase
    1. initialize $\mathcal{A}(s) := \{a_1, \ldots, a_k\}$ with k admissible actions by ***sampling*** the ***probability*** $\tilde{P}(a \mid s)$ (given by the DNN)
    2. initialize the statistics for each action $a \in \mathcal{A}(s)$ as usual:

$$N(s,a) := 0, \quad \hat{Q}(s,a) := +\infty$$

  - Before any *selection* phase in state s, compare number of actions $|\mathcal{A}(s)|$ and number of visits $N(s)$:
    1. if $|\mathcal{A}(s)|^2 \leq N(s)$ add a *new action* a' by sampling the probability $\tilde{P}(a \mid s)$

not enough actions, a lot of visits

a' will be the next selected action

$$\mathcal{A}(s) := \mathcal{A}(s) \cup \{a'\} \quad \text{with} \quad N(s,a') := 0, \quad \hat{Q}(s,a') := +\infty$$

    2. proceed with the usual selection phase

Sampling DNN probability

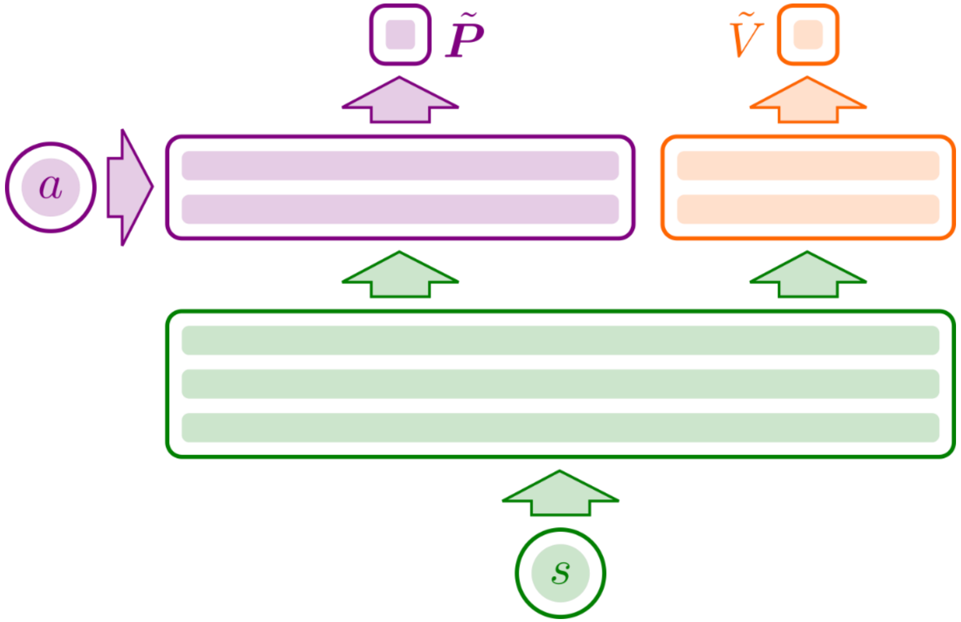
- ***How to sample the DNN probability $\tilde{P}(a \mid s)$?***
  - Probability $\tilde{P}(a \mid s)$ could be the *normalization* of a function such as

vector representing action a

$$p(a\,;s) = \boldsymbol{w} \cdot g(\boldsymbol{W}^{[\ell]} g(\cdots g(\underbrace{\boldsymbol{W}_s^{[1]}\boldsymbol{a} + \boldsymbol{b}_s^{[1]}}_{\text{depending on state } s}) + \cdots) + \boldsymbol{b}^{[\ell]}) + b$$

non-linear continuous function

  - Probability $\tilde{P}(a \mid s)$ is *computable* *<u>given</u>* the state s and the action a



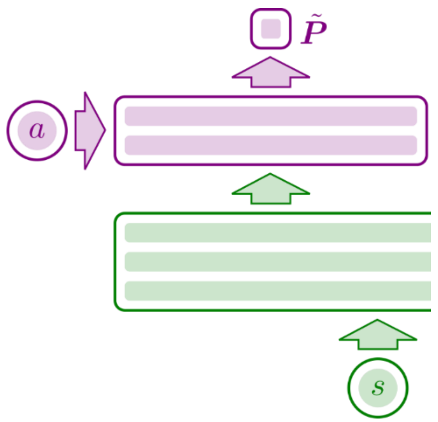
  - *What about sampling $\tilde{P}(a \mid s)$?*

Advanced methods:
Neural Importance Sampling

Neural Importance Sampling

- ***How to sample the DNN probability*** $\tilde{P}(a \mid s)$ ***?***

  we can use the Importance Sampling!



- ***Neural Importance Sampling***
  1) choose a suitable ***bijector*** $\mathcal{T}$
  2) sample $\boldsymbol{y} \in [0,1]^d$ with uniform probability distribution u
  3) apply $\mathcal{T}$ and compute the (vector representing the) action

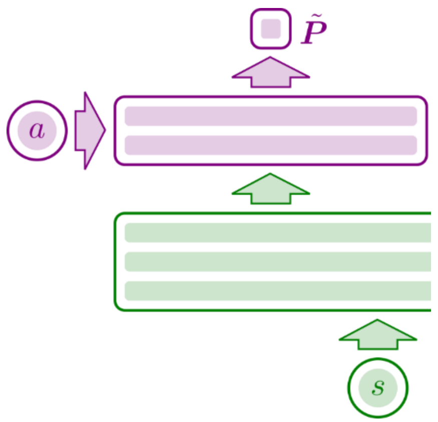
$$\boldsymbol{a} := \boldsymbol{\mathcal{T}}(\boldsymbol{y} \mid s)$$

Then

$$\tilde{P}(a \mid s) = \left| \det \left(\frac{\partial \boldsymbol{\mathcal{T}}(\boldsymbol{y})}{\partial \boldsymbol{y}} \bigg|_{\boldsymbol{y}=\boldsymbol{\mathcal{T}}^{-1}(\boldsymbol{a}|s)} \right) \right|^{-1} u(\boldsymbol{\mathcal{T}}^{-1}(\boldsymbol{a} \mid s))$$

Neural Importance Sampling

- *Training:*
  - minimize a suitable *loss*:



$$L_{\mathrm{KL}}(\hat{P}||\tilde{P}) := \mathbb{E}_{\hat{P}}[\log(\hat{P}(a \mid s)) - \log(\tilde{P}(a \mid s))]$$

$$= \int \hat{P}(a \mid s) \log\left(\frac{\hat{P}(a \mid s)}{\tilde{P}(a \mid s)}\right) \, \mathrm{d}a$$

e.g. ***Kullback-Leibler (KL) divergence***

it can be approximated by a *discrete sum*

  - over the *dataset*

$$D^f := \left\{\langle a_j, s_i, \hat{P}(a_j \mid s_i)\rangle\right\}$$