Google DeepMind

Turbocharging Solution Concepts: Solving NEs, CEs and CCEs with Neural Equilibrium Solvers

Luke Marris, Ian Gemp, Thomas Anthony, Andrea Tacchetti, Siqi Liu, Karl Tuyls

Problem and Niche

Value-based methods such as Nash Q-Learning and Correlated Q-Learning solve for **subgame-perfect equilibria** in terminating **Markov Games**. These approaches involve estimating **action values** (equivalent to a **normal-form games**, or **payoffs** $G_p(a)$) at each state. In these algorithms, equilibria have to be recomputed:

1. Each time the action-values are updated
2. For continuous or large state space, each time an action is taken

These solutions need to be solved frequently. However, not necessarily to high accuracy. Traditional iterative equilibrium solvers are accurate, but take a relatively **long and nondeterministic** amount of time to converge, and **may fail** on ill-conditioned games.

The niche: fast, deterministic, approximate solvers.

The Goal: Train a feedforward neural network to map payoffs directly to equilibrium solutions. $G_p(a) \rightarrow \sigma(a)$

Equilibrium Definitions

CE:

$$A_p^{\text{CE}}(a'_p, a''_p, a) = \begin{cases} G_p(a'_p, a_{-p}) - G_p(a''_p, a_{-p}) & a_p = a''_p \\ 0 & \text{otherwise} \end{cases}$$

$$\sum_{a \in \mathcal{A}} \sigma(a) A_p^{\text{CE}}(a'_p, a''_p, a) \leq \epsilon_p \qquad \forall p \in [1, N], a''_p \neq a'_p \in \mathcal{A}_p$$

CCE:

$$A_p^{\text{CCE}}(a'_p, a) = \sum_{a''_p \in \mathcal{A}_p} A_p^{\text{CE}}(a'_p, a''_p, a) = G_p(a'_p, a_{-p}) - G_p(a)$$

$$\sum_{a \in \mathcal{A}} \sigma(a) A_p^{\text{CCE}}(a'_p, a) \leq \epsilon_p \qquad \forall p \in [1, N], a'_p \in \mathcal{A}_p$$

Invariant Preprocessing and Sampling

Payoffs, $G_p(a) \subset (-\infty, +\infty)$, can be any finite real number. It is impossible to uniformly sample from this full space. And a non-uniform sample would bias a network. **Invariances** are transforms to payoffs that **do not change the space of equilibria**. Two such invariances are:

- **Offset** of each player's payoff
- **Positive scale** of each player's payoff

We can use these invariances (e.g. zero-mean offset, unit-norm scale) to map the space of payoffs to a smaller **invariant subspace**. Benefits:

- Now possible to uniformly sample over this subspace.
- Neural network does not need to learn redundancies in scale and offset.

Unique Solution

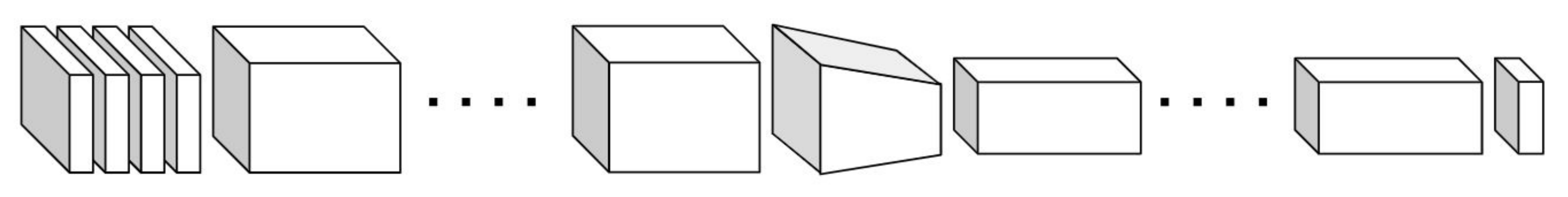
In general, there are **many possible equilibria** for games. Many solvers simply find **any equilibrium**, or any **from a set** according to some objective. Our method solves for a **unique equilibrium** by mixing between a number of convex **parameterisable equilibrium selection criterion**:

1. Linear welfare maximization.
2. Distance to an arbitrary target joint distribution.
3. Target equilibrium approximation parameter.

Benefits:

- Better optimization landscape
- Unique equilibrium selection

Architecture



Unsupervised Loss

Traditionally, neural networks are trained in a supervised fashion. For example with (input ($G_p(a)$), truth ($\sigma^*(a)$)) pairs. This is prohibitive because solving for the truth requires running expensive iterative solvers (discussed earlier). We **formulate an unsupervised loss function** that does require ground truth targets to be trained. Loss and gradients can be computed just from sampling inputs. Benefits:

- Infinite training regime (no pre-computed dataset)
- Training data can be sampled online and on-device
- Very fast training loop

Dual Space Optimization

CE primal problem:

- Primal variables: A^N
- Linear constraints: NA^2
- Nonnegative constraints: A^N
- Equality constraints: 1
- Objective: min-max

CE dual problem:

- Dual variables: NA^2
- Linear constraints: 0
- Nonnegative constraints: NA^2
- Equality constraints: 0
- Objective: loss

CCE primal problem:

- Primal variables: A^N
- Linear constraints: NA
- Nonnegative constraints: A^N
- Equality constraints: 1
- Objective: min-max

CCE dual problem:

- Dual variables: NA
- Linear constraints: 0
- Nonnegative constraints: NA
- Equality constraints: 0
- Objective: loss

Dual Loss Function

Loss:

$$\overset{\text{(C)CE}}{L} = \ln\left(\sum_{a \in \mathcal{A}} \hat{\sigma}(a) \exp\left(\overset{\text{(C)CE}}{l(a)}\right)\right) + \sum_p \epsilon_p^+ \left\{ \sum_{a'_p, a''_p} \overset{\text{CE}}{\alpha_p}(a'_p, a''_p), \sum_{a'_p} \overset{\text{CCE}}{\alpha_p}(a'_p) \right\} - \rho \sum_p \overset{\text{(C)CE}}{\epsilon_p}$$

Logits:

$$\overset{\text{(C)CE}}{l(a)} = \mu \sum_a W(a) - \sum_p \left\{ \sum_{a'_p, a_p} \overset{\text{CE}}{\alpha_p}(a'_p, a_p) \overset{\text{CE}}{A_p}(a'_p, a''_p, a), \sum_{a'_p} \overset{\text{CCE}}{\alpha_p}(a'_p) \overset{\text{CCE}}{A_p}(a'_p, a) \right\}$$

Primals:

$$\overset{\text{(C)CE}}{\sigma(a)} = \frac{\hat{\sigma}(a) \exp\left(\overset{\text{(C)CE}}{l(a)}\right)}{\sum_{a \in \mathcal{A}} \hat{\sigma}(a) \exp\left(\overset{\text{(C)CE}}{l(a)}\right)} \quad (9) \quad \overset{\text{(C)CE}}{\epsilon_p} = (\hat{\epsilon}_p - \epsilon_p^+) \exp\left(-\frac{1}{\rho}\left\{ \sum_{a'_p, a''_p} \overset{\text{CE}}{\alpha_p}(a'_p, a''_p), \sum_{a'_p} \overset{\text{CCE}}{\alpha_p}(a'_p) \right\}\right) + \epsilon_p^+$$

Equivariant Architecture

There are many **equivariances** in the representation of normal-form games. Equivariances are transforms to the payoffs that change the equilibrium in a predictable way. Two such equivariances:

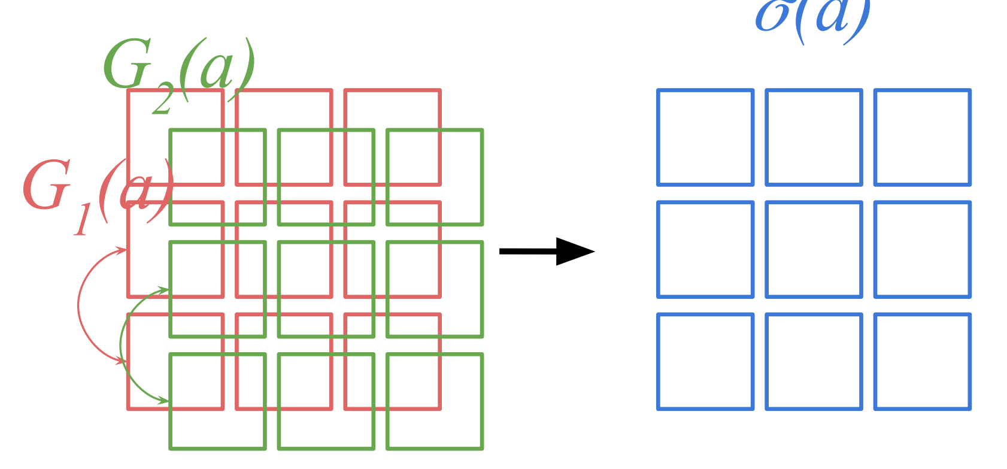
1. ***Permutation of actions*** in a payoff results in the same *permutation* in joint.
2. ***Permutation of players*** in a payoff results in the same *transposition* in joint.

We can exploit these equivariances by building them into the architecture of the neural network. We use a **channel dimension** and **pooling functions** to achieve this (see paper for details). This has three benefits:

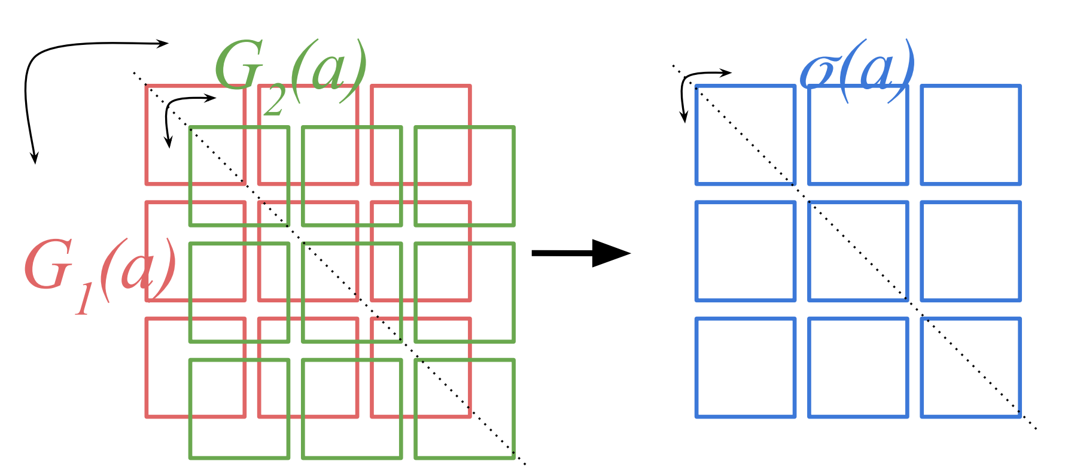
1. Reduces the number of parameters required the network requires
2. Equivariant games give consistent results
3. Each sample is equivalent to training over all permutations

As games get larger, the number of permutations grows rapidly: $N! \, (|A_p|!)^N$

Action permutation equivariance:



Player permutation equivariance:


$G_2(a)$
$G_1(a)$
$\sigma(a)$