DeepMind

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

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

NeurIPs 2022

Equilibrium Solving in Multiagent Learning Algorithms

Value-based methods such as Nash Q-Learning [Hu, 2003] and Correlated Q-Learning [Greenwald, 2004] 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.

Policies are the equilibrium solutions (eg NE or CE) at these states. In these algorithms, equilibria have to be recomputed:

1. Each time the action-values are updated
2. (For continuous or large state game) Each time an action is taken

These solutions need to be solved frequency. However, because the action values are approximations, often defined using a function approximator, high accurate solutions may not be important.

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.

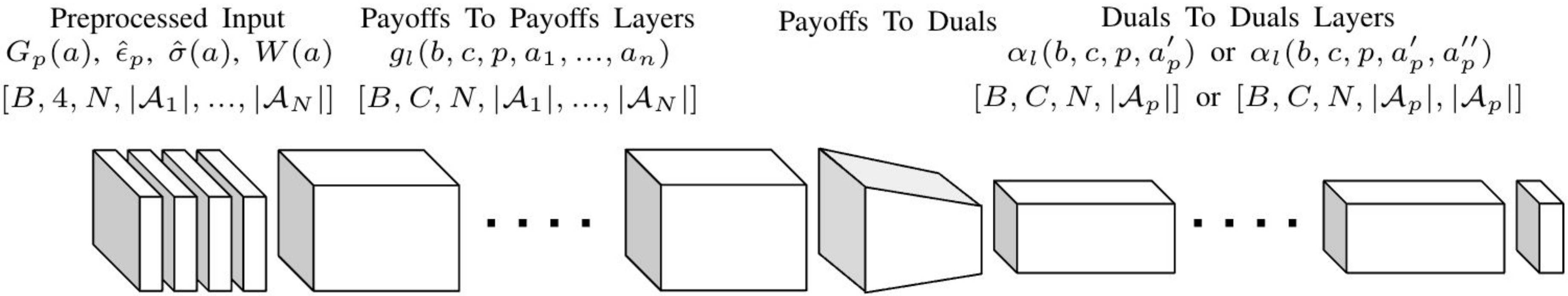
$$G_p(a) \rightarrow \sigma(a)$$

The Result: Neural Equilibrium Solver

Properties:

1. Finds a **unique equilibrium**
2. With flexible **equilibrium selection** objectives
3. Can be trained over **all games** of a specific shape
4. Can be trained **without supervised signal**
5. Is a **differentiable** model
6. Is super fast

How did we do this... ?



The Secret Sauce (1): 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.

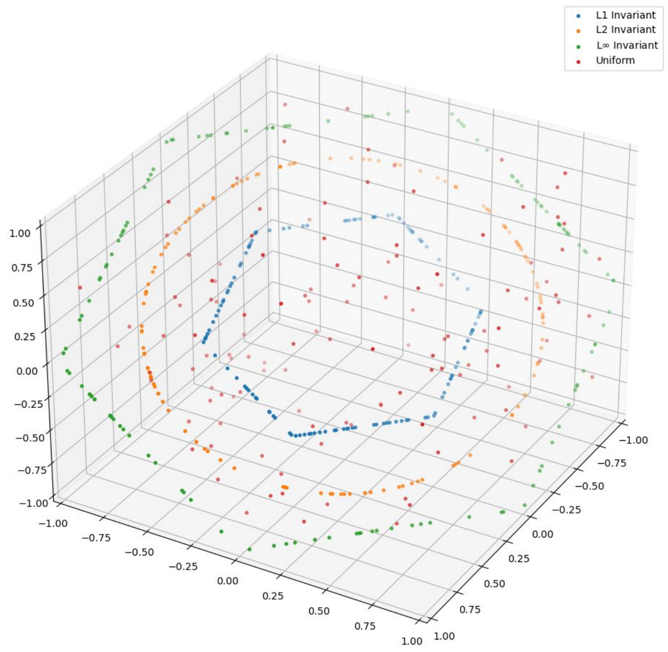
Invariances are transforms to payoffs that **do not change the space of equilibria**. Two such invariances are:

- **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.

The network can be trained for all games of a specific shape.



Different possible invariant subspaces with different scale normalizations.

The Secret Sauce (2): 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.

Mixing parameters: μ, ϱ

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

primal variable (output of NN)

target joint parameter

target linear parameter

target approximation parameter

$$L^{\sigma,\epsilon_p}_{\alpha_p,\beta,\lambda} = -\sum_a \sigma(a)\ln\left(\frac{\sigma(a)}{\hat{\sigma}(a)}\right) + \mu\sum_{a\in\mathcal{A}} W(a)\sigma(a) - \rho\sum_p \left(\epsilon_p^+ - \epsilon_p\right)\ln\left(\frac{1}{\exp(1)}\frac{\epsilon_p^+ - \epsilon_p}{\left(\epsilon_p^+ - \hat{\epsilon}_p\right)}\right)$$

Gains from payoffs (input of NN)

dual variable

dual variable

$$+\sum_a \beta(a)\sigma(a) - \lambda\left(\sum_a \sigma(a) - 1\right) - \sum_{p,a'_p,a''_p} \alpha_p(a'_p, a_p)\left(\sum_a \sigma(a) A_p(a'_p, a''_p, a) - \epsilon_p\right)$$

The Secret Sauce (3): 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 loss function:

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

Note there is no ground truth, $\sigma^*(a)$.

The Secret Sauce (4): Dual Space Optimization

CE primal problem:

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

CCE primal problem:

- Primal variables: A^N
- Linear constraints: NA
- 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 dual problem:

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

Benefits:

- Huge reduction in number of variables
- Huge reduction in number of constraints
- Nonnegative constraints are simply implemented
- Loss function much easier to to optimize over a min-max objective

N: Number of players

A: Number of actions per player

Notice that these reductions scale well to large games (large N and large A).

The Secret Sauce (5): 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 of actions* in the joint.
2. ***Permutation of players*** in a payoff results in the same *transposition of players* in the joint.

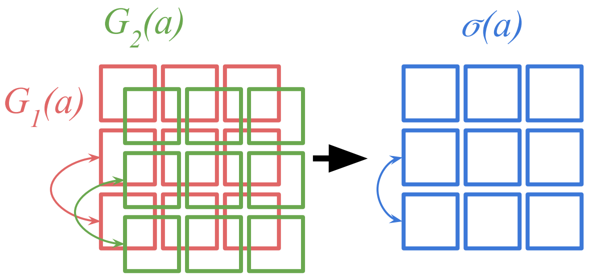
We can exploit these equivariances by building them into the architecture of the neural network. We use a **channel dimension** and **pooling functions** to achieve this (see paper for details).

This has three benefits:

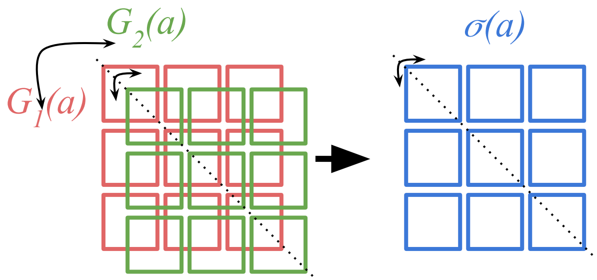
1. Reduces the number of parameters required the network requires
2. Equivariant games give consistent results
3. Each sample is equivalent to training over all permutations

As games get larger, the number of permutations grows rapidly: $N!\left(|\mathcal{A}_p|!\right)^N$

Action permutation equivariance:



Player permutation equivariance:



DeepMind

Thank you for Listening

See you at the Poster