DEEP GALERKIN METHOD FOR MEAN FIELD CONTROL PROBLEMS

JINGRUO SUN

MENTORED BY: PROF. ASAF COHEN

Abstract. We consider an optimal control problem where the average welfare of weakly interacting agents is of interest. We examine the mean-field control problem as the fluid approximation of the N-agent control problem with the setup of finite-state space, continuous-time, and finite-horizon. The value function of the mean-field control problem is characterized as the unique viscosity solution of a Hamilton–Jacobi–Bellman equation in the simplex. We apply the DGM to estimate the value function and the evolution of the distribution.

1. Introduction

The N-agent optimal control problem considers a cooperative game of N weakly interacting agents traversing among multiple states in continuous time over a finite horizon and aiming at minimizing the average running cost. By weak interaction we mean that each player affects the dynamics of the others and their costs through the empirical distribution of the agents' states. Specifically, players select their transition rates and pay costs that depend on their private states and actions as well as on the empirical distribution of the states of all other players. When N is large, this problem is technically intractable. Hence, we approximate it by a mean field control problem (MFCP). Informally, this is a control problem with continuum of players, where the empirical distribution is replaced by a flow of measures. Rigorously, it is defined via an optimization problem of a McKean–Vlasov process.

In addition to the formulation of controlling a stochastic McKean–Vlasov process, the MFCP can be reformulated by controlling a (deterministic) Kolmogorov–Fokker–Planck equation, which permits to derive a dynamic programming principle and then a Hamilton-Jacobi-Bellman (HJB) equation stated in the d-dimensional simplex for the value function (optimal cost).

Using viscosity techniques, Cecchin [3] showed that the rate of convergence attained there for the value functions is of order $1/\sqrt{N}$. Under sufficient regularity, the rate of convergence is of order $1/N$. The latter rate was attained by Kolokoltsov assuming that the cost and transition rate are just Lipschitz-continuous with respect to the distribution and not in $\mathcal{C}^{1,1}$ and applying the convergence of the generator to the limiting dynamics[1].

In order to numerically solve the HJB equation and the Kolmogorov's equation, we construct a feed-forward neural network and apply the (deep Galerkin method) DGM. The DGM was introduced in section 4. It is a numerical method of solving PDE in high dimensions by training the neural network to satisfy the differential operator, initial condition, and boundary conditions using SGD at randomly sampled spatial points. Finally, we show the graphic results of value function in dimension 2 and distribution in dimension 5, and we provide a table of results with the running time of applying DGM on PDEs in dimension 10, 50, and 100 respectively.

We organize the paper as the followings. In Section 2, we introduce the N-agent control problem and the corresponding HJB equation. Next, we present the mean field control problem and its HJB equation. We end this section by stating the convergence of the N-agent control problem to the MFCP. In Section 3, we provide a short background for neural networks. In Section 4, we present the DGM to solve the HJB equation of MFCP and the Kolmogorov's equation and provide the results of value functions and distributions. In Section 5 we discuss our future plan to rigorously establish the convergence.

2. The N-agent control problem and the Mean Field Control Problem

The N-agent optimal control problem involves a decision process with multiple agents, where each agent is associated with an individual cost function and a strategy set. The players are targeting at minimizing the average cost. A dynamic programming principle yields the HJB equation. Before introducing the stochastic methods, we start with some notations.

2.1. **Notation.**

We denote by $[\![d]\!] = \{1, 2, ..., d\}$ the state space and let

$$S_d = \Big\{(\mu_1, ..., \mu_d) \in \mathbb{R}^d : \ \mu_i \geq 0 \ \ \forall i \in [\![d]\!], \ \ \sum_{i=1}^{d} \mu_i = 1\Big\}$$

be the simplex with dimension of $(d-1)$, endowed with the euclidean norm $|\cdot|$ in $\mathbb{R}^d$. We denote the elements of the simplex by m, while μ denotes processes with values in the simplex. The set of all admissible controls is denoted as $\mathcal{A}$. We introduce the exact term admissible controls later.

In the interior of the simplex, derivatives are allowed only along directions $(\delta_j - \delta_i)_{i,j\in[\![d]\!]}$, which are denoted as $\partial_{m_j - m_i} V(t, m)$. We define the vector

$$D^i V(t, m) \ := \ (\partial_{m_j - m_i} V(t, m))_{j\in[\![d]\!]}.$$

For $\mathbf{x} = (x_1, ..., x_N) \in [\![d]\!]^N$, denote the empirical measure

$$\mu^N_{\mathbf{x}} := \ \frac{1}{N} \ \sum_{k=1}^{N} \delta_{x_k}, \ \text{that is,} \ \ \mu^N_{\mathbf{x}}[i] = \frac{1}{N} \ \sum_{k=1}^{N} \mathbb{1}_{\{x_k = i\}}, \ i \in [\![d]\!]. \tag{2.1}$$

2.2. **Motivating the mean field control problem.**

We consider a symmetric N-agent control problem. We equip all the players with the same Markovian control with inputs: time, private state, and empirical distribution of all the players. Namely, $\alpha_N : [0, T] \times [\![d]\!] \times S^N_d \to A$. The state of player k at time t is denoted by X^k_t. Denote by μ^N_t the empirical distribution of the players at time t. Namely,

$$\mu^N_t \ = \ \frac{1}{N} \sum_{k=1}^{N} \delta_{X^k_t}. \tag{2.2}$$

The dynamics of the player k are given by

$$\mathbb{P}\Big(X^k_{t+h} = j | X^k_t = i, \mu^N_t = m\Big) = Q_{i,j}(t, \alpha_N(t, i, m), m)h + o(h) \quad \text{ as } h \to 0^+, \tag{2.3}$$

for some transition matrix Q, satisfying some conditions mentioned in the following subsection.

The common goal of the players is to choose α_N in order to minimize the cost:

$$\begin{aligned} J^N(\alpha_N) &:= \frac{1}{N} \ \sum_{k=1}^{N} \ \mathbb{E} \ \Big[\ \int_0^T \ f(t, X^k_t, \alpha_N(t, X^k_t, \mu^N_t), \mu^N_t)dt \ + \ g(X^k_T, \mu^k_T) \ \Big] \\ &= \mathbb{E} \ \Big[\ \int_0^T \langle f(t, \cdot\, , \alpha_N(t, \cdot\, , \mu^N_t), \mu^N_t \rangle dt + \langle g(\cdot\, , \mu^N_T), \mu^N_T \rangle \ \Big], \end{aligned} \tag{2.4}$$

where f and g are the running and terminal costs, and $\langle h, \nu\rangle$ stands for the integration of the function h with respect to the measure ν.

Therefore, the N-agent optimal control problem can be regarded as a single optimization problem for the empirical measure, which is a time-inhomogeneous Markov chain on S_d^N such that

$$\mathbb{P}\Big(\mu_{t+h}^N = m + \frac{1}{N}(\delta_j - \delta_i) | \mu_t^N = m\Big) = N m_i Q_{i,j}(t, \alpha_N(t,i,m), m)h + o(h) \quad \text{as } h \to 0^+, \tag{2.5}$$

for any $m \in S_d^N$ and $i \neq j \in [\![d]\!]$. We remark that the dynamics remains in S_d^N because $m + \frac{1}{N}(\delta_j - \delta_i)$ can be outside S_d^N only if $m_i = 0$, but in such a case the transition rate is zero.

We set

$$\mu_0 = \frac{1}{N}\sum_{k=1}^N \mathbb{1}_{\{X_0 = i\}}.$$

The generator of this Markov chain with transition probability (2.18) is hence given by:

$$\mathcal{L}_t^{N,\alpha_N} h = N \sum_{i,j \in [\![d]\!]} m_i Q_{i,j}(t, \alpha_N(t,i,m), m)\Big[h(t, m + \frac{1}{N}(\delta_j - \delta_i)) - h(t,m)\Big]. \tag{2.6}$$

Denoted by $V^N(t,m)$ the value function of the N-agent control problem:

$$V^N(t,m) = \inf_{\alpha_N} J^N(\alpha_N). \tag{2.7}$$

The HJB equation for the value function is then

$$-\frac{d}{dt}V^N(t,m) + \max_{\alpha \in A^d}\Big\{-\mathcal{L}^{N,\alpha}V^N(t,m) - \sum_{i \in [\![d]\!]} m_i f^i(t, \alpha^i, m)\Big\} = 0, \tag{2.8}$$

which may be rewritten as an ODE, indexed by $m \in S_d^N$ as follows

$$\begin{aligned} -\frac{d}{dt}V^N + \sum_{i \in [\![d]\!]} m_i H^i(t, m, D^{N,i}V^N(t,m)) &= 0, \\ V^N(T,m) &= \sum_{i \in [\![d]\!]} m_i g^i(m), \end{aligned} \tag{2.9}$$

where H^i is the Hamiltonian defined as:

$$H^i(t,m,z) := \max_{a \in \mathcal{A}} -\Big\langle Q_{i,\cdot}(t,a,m), z\Big\rangle - f(t,x,a,m). \tag{2.10}$$

2.3. **Assumptions and preliminary results for the N-agent game.** Let Q be the transition matrix with each entry $Q_{i,j} : [0,T] \times S_d \times A \to [0, +\infty)$ for $i \neq j$ to be the transition rate from state i to state j, and $Q_{i,i} = -\sum_{j \neq i} Q_{i,j}$. Let $F : [0,T] \times A^d \times S_d \to \mathbb{R}$ and $G : S_d \to \mathbb{R}$ be defined by

$$F(t, a^1, ..., a^d, m) := \sum_{i \in [\![d]\!]} m_i f(t,i,a,m), \;\; G(m) := \sum_{i \in [\![d]\!]} m_i g(i,m) \quad \text{as } h \to 0^+, \tag{2.11}$$

We provide three sets of assumptions. The first one yields existence of a solution to the HJB equation, as well as the existence of an optimal control. The second assumption implies uniqueness, while the last assumption implies regularity for the MFCP.

Assumption 2.1. *(1) The action space $\mathcal{A}$ is a compact metric space.*
(2) The transition rate $Q_{i,j}$ is continuous on $[0,T] \times A \times S_d$ and Lipschitz-continuous in (t,m),

uniformly continuous in a:

$$|Q_{i,j}(t,a,m) - Q_{i,j}(s,a,p)| \leq C(|t-s| + |m-p|). \tag{2.12}$$

(3) The functions F is continuous on $[0,T] \times A^d \times S_d$ and

$$|F(t,a,m) - F(s,a,p)| \leq C(|t-s| + |m-p|), \tag{2.13}$$
$$|G(m) - G(p)| \leq C|m-p|. \tag{2.14}$$

Assumption 2.2. *Assumption 2.1 holds, and in addition:*
(1) The range of control is $A = [0,M]^d$.
(2) The transition rate is $Q_{i,j}(t,m,a) = a_j$.
(3) The running cost f is continuously differentiable in A, $\nabla_a f$ is Lipschitz-continuous with respect to m, and f is uniformly convex in A, i.e., there exists $\lambda > 0$ such that

$$f(t,i,b,m) \geq f(t,i,a,m) + \langle \nabla_a f(t,\alpha,m),\ b-a\rangle + \lambda|b-a|^2. \tag{2.15}$$

It is shown in [6] that under this assumption, there exists a unique maximizer of H, which we denote by $\alpha^*(t,i,m,z)$, and further, α^* is Lipschitz continuous with respect to m and z, i.e.

$$|\alpha^*(t,i,m,z) - \alpha^*(t,i,p,w)| \leq C(|m-p| + |z-w|). \tag{2.16}$$

We consider feedback controls $\alpha : [0,T] \to A^d$ and denote $\alpha_{i,j}(t) := \alpha_j(t,i)$ when (B1) holds.

Assumption 2.3. *Assumption 2.2 holds, and in addition:*
(1) $F(\cdot,a,\cdot) \in \mathcal{C}^{1,1}([0,T] \times S_d)$ uniformly in a. $G \in \mathcal{C}^{1,1}(S_d)$.
(2) The function

$$[0,T] \times [0,+\infty)^{d\times d} \times Int(S_d) \ni (t,w,m) \to \sum_i m_i f(t,i,(\frac{w_{i,j}}{m_i})_{j\neq i}, m) \in \mathbb{R}$$

is convex in (w,m) and G is convex in m.
This assumption provides a sufficient condition for the value function of the MFCP to belong to $\mathcal{C}^{1,1}([0,T] \times S_d)$, see [3].

The main differences between the three assumptions are that in Assumption 2.1 there is no presumption of convexity, while in Assumption 2.2 we assume convexity in α and in Assumption 2.3 we assume convexity in (α, m).

We remark that we do not assume f is separable in the sense that it is not necessarily a function of (i,α) plus a function of (i,m), as it would be in the case of potential mean field games, where the cost functions f_0^i and g^i defining the MFCP do not depend on i, see[4].

Remark 2.1. Under Assumption 2.1, the HJB equation (2.9) admits a unique solution V^N which is $\mathcal{C}^1$ in time. V^N is the value function of the N-agent control problem, defined as the infimum over feedback controls $\alpha_N : [0,T] \times S_d^N \to A^d$, satisfying the Lipschitz property:

$$|V^N(t,m) - V^N(s,p)| \leq C(|t-s| + |m-p|) \tag{2.17}$$

for a constant C independent of N.

Remark 2.2. If Assumption 2.2 also holds, then the optimal control is unique, since the maximizer of the Hamiltonian in unique, see [5]. The control is the transition rate $\alpha_N(t,i,m) \in [0,M]^d$, which we denote as a transition matrix $\alpha_N = (\alpha_N^{i,j})_{i,j\in[\![d]\!]} \in [0,M]^{d\times d}$. In such a case, the transition probability given in (2.3) becomes simply

$$\mathbb{P}\Big(X^k_{t+h} = j | X^k_t = i, \mu^N_t = m\Big) = \alpha^{i,j}_N(t,m)h + o(h) \tag{2.18}$$

and the dynamics of μ^N in (2.5) are then given by

$$\mathbb{P}\left(\mu^N_{t+h} = m + \frac{1}{N}(\delta_j - \delta_i) \;\middle|\; \mu^N_t = m\right) = N m_i \alpha^{i,j}_N(t,m)h + o(h) \tag{2.19}$$

2.4. **Mean Field Control Problem.** As the number of players tends to infinity, the limit of the N-agent cooperative game is interpreted as MFCP. In this problem, the dynamics follow a McKean–Vlasov process. Namely, let $(X_t)_{t\in[0,T]}$ be a standard process, whose dynamics are given by

$$\mathbb{P}(X_{t+h} = j | X_t = i) = Q_{i,j}(t, \alpha^i(t), \mathrm{Law}(X_t))h + o(h), \quad \text{as } h \to 0^+, \tag{2.20}$$

where the control is a deterministic measurable function $\alpha : [0,T] \to A^d$. The goal is to minimize the cost:

$$\begin{aligned} J(\alpha) &:= \mathbb{E}\left[\int_0^T f(t, X_t, \alpha(t, X_t), \mathrm{Law}(X_t))dt + g(X_T, \mathrm{Law}(X_T))\right] \\ &= \int_0^T \Big\langle f(t, \cdot\,, \alpha(t), \mu_t), \mu_t\Big\rangle dt + \Big\langle g(\cdot\,, \mu_T), \mu_T\Big\rangle, \end{aligned} \tag{2.21}$$

where we use the notation $\mu(t) := \mathrm{Law}(X_t)$. Then its evolution is given by Kolmogorov's forward equation:

$$\begin{aligned} \frac{d}{dt}\mu^i_t &= \sum_{j\in[\![d]\!]} \Big[\mu^j_t Q_{j,i}(t, \alpha^j(t), \mu_t) - \mu^i_t Q_{i,j}(t, \alpha^i(t), \mu_t) \Big], \\ \mu_0 &= m_0. \end{aligned} \tag{2.22}$$

Then the value function of our average player is

$$V(t,m) = \inf_{\alpha\in A^d} J(\alpha). \tag{2.23}$$

which solves the PDE of

$$\begin{aligned} &-\partial_t V(t,m) + \sum_{i\in[\![d]\!]} m_i H^i(t, m, D^i V(t,m)) = 0, \\ &V(T,m) = \sum_{i\in[\![d]\!]} m_i g^i(m). \end{aligned} \tag{2.24}$$

in which we recall that $[D^i V(t,m)]_j := \partial_{m_j - m_i} V(t,m)$.

Remark 2.3. In general, the value function may not be differentiable. For this reason we consider a weaker notion of existence. Namely, viscosity solutions. Under Assumption 2.1, V is the unique viscosity solution on S_d and $\mathrm{Int}(S_d)$, and it is Lipschitz-continuous in (t,m), see [3]. If Assumption 2.2 holds, there exists an optimal control $\alpha \in A^d$. If Assumption 2.3 also holds, then $V \in \mathcal{C}^{1,1}([0,T] \times S_d)$ and is the unique classical solution of (2.24).

2.5. **Convergence of the value functions.** As discussed in previous sections, it is easy to notice the similarities between the value functions of N-agent control problem and the MFCP through (2.9) and (2.24). This makes sense since the MFCP is interpreted as a formal limit of N-agent control problem as N tends to infinity. Now we are going to have a deep look at this interpretation by giving an explicit mathematical verification of convergence theorem.

First, we state the result of convergence: as $N \to \infty$, the value function V^N of the N-agent optimal control problem tends to the value function V of the MFCP with a convergence rate of order $1/\sqrt{N}$. Recall that V^N is the classical solution to (2.9), while V is the viscosity solution to the PDE (2.24).

Theorem 2.1. Under Assumption 2.1,

$$\max_{t\in[0,T],m\in S_d} |V^N(t,m) - V(t,m)| \leq \frac{C}{\sqrt{N}} \tag{2.25}$$

The theorem is proved by exploiting the characterization of V as the viscosity solution to the PDE of (2.24), see [3]. In fact, the ODE (2.9) can be regarded as a finite difference scheme for the PDE (2.24). Indeed, the argument $D^{N,i}V$ of the Hamiltonian in (2.9) converges, at least formally, to D^iV appearing in (2.24) as

$$\lim_{N\to\infty} N\left[V\left(t, m + \frac{1}{N}(\delta_j - \delta_i)\right) - V(t,m)\right] = \partial_{m_j - m_i} V(t,m). \tag{2.26}$$

This result also permits to construct quasi-optimal controls for the N-agent optimization, starting from quasi-optimal controls for the MFCP, with an explicit rate of approximation.

Theorem 2.2. Under Assumption 2.1, fix $\epsilon > 0$ and $N \in \mathbb{N}$. Let $\alpha : [0,T] \to A^d$ be an $\epsilon-$optimal control for the MFCP. Then

$$J^N(\alpha) \leq \inf_{\alpha_N} J^N(\alpha_N) + \frac{C}{\sqrt{N}} + \epsilon \tag{2.27}$$

Here, $J^N(\alpha)$ is understood as applying the control $\alpha_N(t,m) = \alpha(t)$, which is independent of m. Recall that the infimum over controls α_N is the same as the infimum over controls $\boldsymbol{\beta}$.

Remark 2.4. Under Assumption 2.3, we get:

$$\max_{t\in[0,T],m\in S_d} |V^N(t,m) - V(t,m)| \leq \frac{C}{N} \tag{2.28}$$

In case the value function V is smooth, the optimal control of the MFCP is unique and then, if Assumption 2.2 holds, we are also able to establish a propagation of chaos result, that is, it is feasible to prove the convergence of the optimal trajectory of the N-agent optimization to the unique optimal trajectory of the MFCP with a suitable convergence rate.

3. Neural Network

A Neural Network (NN) is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. It consists of three main components: input layer, processing layer, and output layer, where each layer is an interconnected group of neurons, also known as perceptrons, using a mathematical model to learn representations of input data and to capture the salient characteristics of the distribution. In this way, the neural network performs as an adaptive system that changes its structure based on external or internal information flowing through the network.

3.1. **Feed-Forward Neural Network.** A Feed-Forward Neural Network (FNN) is an artificial neural network with information only flowing forward from input layer to hidden layer and finally to the output layer. Opposite to the Recurrent Neural Network (RNN) with directed unrolled cyclic graph, the feed-forward neural network has the following characteristics:
(1) Perceptrons are arranged in layers, with the first layer taking in inputs and the last layer producing outputs. The middle layers have no connection with the external world, and hence are called hidden layers. One FNN may have multiple hidden layers.
(2) Each perceptron in one layer is connected to every perceptron on the next layer. Therefore the information is constantly "fed forward" from one layer to the next.
(3) There is no connection among perceptrons in the same layer.

3.2. **Multilayer Perceptron.** A Multiplayer Perceptron (MLP) is a fully connected class of feed-forward neural network consisting of an input layer, hidden layers, and an output layer. The input layer collects input patterns. The output layer has classifications or output signals to which input patterns may map. Hidden layers fine-tune the input weightings until the margin of error in the neural network is minimized. It is hypothesized that hidden layers extrapolate salient features in the input data that have predictive power regarding the outputs, which accomplishes a utility similar to statistical techniques such as principle component analysis.

In particular, each layer is arranged by neurons to receive signals then transmit them to the neurons connected to it. The "signal" at the connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connection is called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Notice that neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform various transformations on their inputs. Signals travel from the input layer to the output layer, possibly after traversing multiple intermediate processing layers. Here we provide a graph of the structure of the multilayer perceptron to give a intuitive understanding.

As it is shown in Figure 1, layers are arranged in the order of an input layer, hidden layers, and an output layer. Neurons are organized into multiple layers. Each neuron has its own parameter of weight and bias. Specifically, let θ be the parameter of each neuron, then

$$\theta := (\text{weight, bias}) = (w, b). \tag{3.1}$$

To calculate the output of the neuron, we take the weighted sum of all the inputs, weighted by the weights of the connections from the inputs to the neuron, and add a bias term to this sum. This weighted sum, always defined as activation, is then passed through a non-linear activation function to produce the output. That is, let h be the activation function, w be the weight, b be the bias, z_i be the i-th neuron in the previous layer, and z_j be the j-th neuron in the current layer. Thus, $w_{j,i}$ denote the weight for the connection between z_i and z_j, and b_j denote the bias of z_j. Then the value of z_j is given by the function of z_i as the following:

$$z_j = \; h\,\Big(\sum_i \,[w_{j,i} z_i] + b_j\Big). \tag{3.2}$$

3.3. **Backpropagation.** After the organization of neural network, it comes to the step of training. Training is an adaptation of parameters for the neural network to improve the accuracy of the result. It occurs in the perceptron by adjusting connection weights after each piece of data is processed, based on the amount of error in the output compared to the expected result. Practically

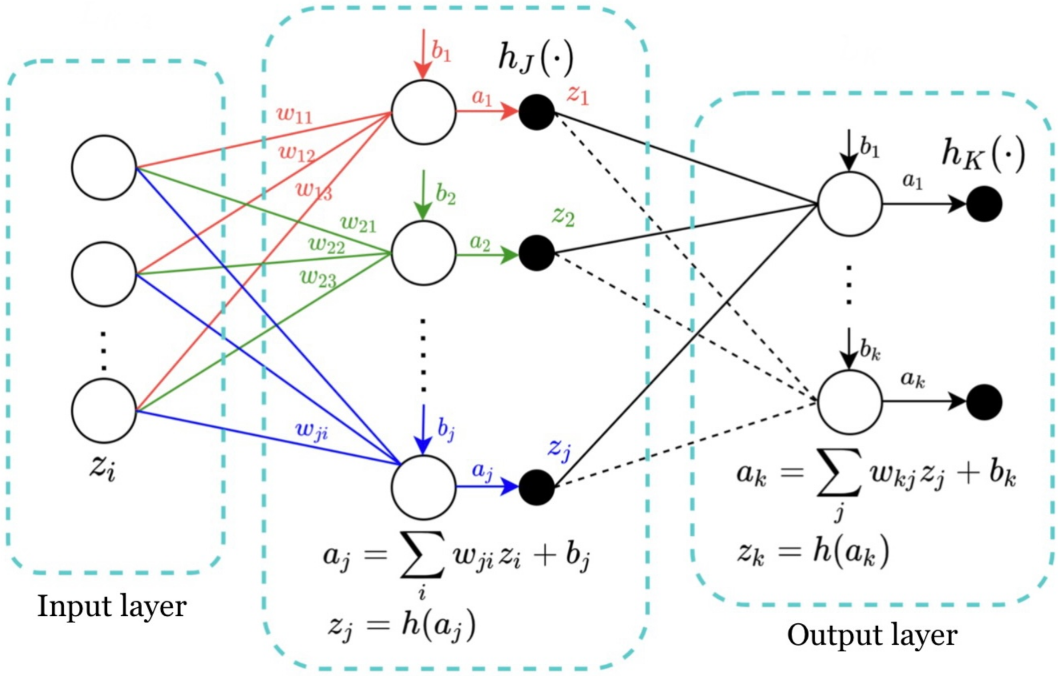

FIGURE 1. Constructure of MLP

this is done by defining a cost function that is evaluated periodically during training. Training continues until a predetermined error is sufficiently small.

Backpropagation is a widely used algorithm for training feed-forward neural networks, [2]. As the essence of training, it uses the method of fine-tuning the parameter of NN based on the error rate obtained in the previous epoch. To illustrate, the backpropagation computes the gradient of the loss function with respect to weights of the neural network for a single input-output example. Unlike a naive direct computation of the gradient individually, backpropagation leverages dynamic programming to prevent repeating calculations. The algorithm works by computing the gradient one layer at a time and iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule, making the computation scales linearly with the number of layers. This high efficiency makes it feasible to use gradient methods for training multilayer networks by updating weights to minimize loss. Such proper tuning of the weights allows us to reduce error rates and make the model reliable by increasing its generalization.

3.4. **Stochastic Gradient Descent.** Now we introduce an important method of deciding the value of parameters in the neural network. To begin with, Gradient Descent (GD) is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. The idea is to update the weights of the neuron in the in the opposite direction of the gradient calculated from all the data points of the loss function. Stochastic Gradient Descent (SGD), regarded as a stochastic approximation of gradient descent optimization, replaces the exact gradient calculated from the entire data set by an estimate thereof calculated from a randomly selected subset of the data. By this means, the SGD reduces the high computational burden and achieves faster iterations in trade for a lower convergence rate, especially in high-dimensional optimization problems.

The function where we are going to take the gradient is called loss function. The loss function is a measurement of the neural network that quantifies the difference between the expected output and the output produced by the machine learning model. Thus, if we denote the loss function

for our model as G with input variables of t and m, and recall that the parameter of the neural network is represented by $\theta = (w, b)$, then we could elucidate the three steps of the SGD algorithm as follows:
(1) Begin with an initial guess of θ
(2) Compute the gradient of the loss function G with respect to θ
(3) Update the value of θ according to SGD:

$$\theta_{n+1} = \theta_n - r\nabla_{\theta_n} G(t_n, m_n, \theta_n) \tag{3.3}$$

where t_n and m_n are observations that we take for the n-th calculation, and r is the learning rate to define the size of the corrective steps that the model takes to adjust for errors in each observation. As taking the descent direction, the loss function will decrease after each iteration, thus we get the updated parameter θ_{n+1} as a better estimate from the previous value θ_n. If we repeat this step, the parameter will finally converge to a critical point of the loss function.

Remark 3.1. In GD, we run through all the samples in the training set for a single update for the parameter in a particular iteration. In SGD, on the other hand, we use only one or a subset of the training sample from the training set to do the update for a parameter in a particular iteration. When using a subset, it is called Minibatch SGD. Specifically, in the method of GD, this step should be rewritten as:

$$\theta_{n+1} = \theta_n - r\,\frac{1}{n}\sum_{i=1}^{n} \nabla_{\theta_i} G(t_i, m_i, \theta_i)$$

where n is the total number of data. Thus, with this comparison, we get rid of the sum and the $1/n$ constant, where comes the time saving.

Remark 3.2. Note that changing the weight w_j will modify part of the input of its connected neuron, which is further modified in all subsequent layers. Hence, the adjustment of w_j will have a non-linear effect on the output of each subsequent layer. In mathematical term, this means the calculation will require the chain rule to evaluate the partial derivative, making the gradient become computationally intractable. The naive approach would be to calculate the gradient for each weight individually. In this case, however, the number of computations would scale exponentially with the number of layers. Therefore, to make the SGD a viable option, a more efficient calculation method is needed. This is where backpropagation comes into play.

4. DGM

In this section, we are interested in solving numerically the HJB equation in (2.24). Numerical methods that are based on grids can fail when the dimensionality of the problem becomes too large since the number of points in the mesh is going to grow exponentially. Even if we were to assume that the computational cost was manageable, ensuring that the grid is set up in a way to hold stability of the finite difference approach can be cumbersome. With this motivation, a mesh-free method that approximates the solution to the PDE of interest using deep neural network is proposed.

4.1. **DGM.**

Solving high-dimensional PDEs is a challenging problem. Mesh-grid methods become infeasible in high dimensions due to the explosion in the number of grid points and the demand for reduced time step size. That is to say, if there are d space dimensions and 1 time dimension, the mesh is of size $\mathcal{O}^{d+1}$, which will quickly become computationally intractable when the dimension d is even moderately large. We solve this equation by using the DGM method on which we detail.

The Galerkin method seeks a reduced-form solution to a PDE as a linear combination of basis functions. The Deep Galerkin Method (DGM) is a natural merger of Galerkin methods and machine learning, [7]. Instead of choosing linear combinations, it approximates the solution with a deep neural network which is trained to satisfy the differential operator, initial condition, and boundary conditions using SGD at randomly sampled spatial points. By randomly sampling spatial points, we avoid the need to form a mesh and instead convert the PDE problem into a machine learning problem.

The algorithm in principle is straightforward: with the parameterization of the neural network, a loss function is set up to penalize the fitted function's deviations from the desired differential operator and boundary conditions. For the training data, the networks uses points randomly sampled from the region where the objective function is defined and the optimization is performed using SGD. These are the key innovative parts of this method.

The main insight of this approach lies in the fact that by sampling mini-batches from different parts of the domain and processing these small batches sequentially, the neural network "learns" the function without the computational bottleneck present with grid-based methods. Thus, instead of a huge mesh of $\mathcal{O}^{d+1}$, DGM converts the computational cost of finite differences to a more convenient form and trains the neural network on batches of randomly generated time and space points. Although the total number of spatial points could be vast, the algorithm can process the sampling points sequentially without harming the convergence rate. More precisely, we raise a theorem to illustrate the convergence as following.

4.2. **Application of DGM to Mean Field Control Problem.**

Now we are going to apply the DGM to the mean field control problem. To begin with, recall that our PDE for the value function $V(t,m)$ in (2.24) is:

$$\begin{aligned} &-\partial_t V(t,m) + \sum_{i\in[\![d]\!]} m_i H^i(t,m,D^iV(t,m)) = 0, \quad \big((t,m)\in[0,T]\times S_d\big) \\ &V(T,m) = \sum_{i\in[\![d]\!]} m_i g^i(m). \end{aligned}$$

Then to solve this HJB equation, we apply the DGM neural network following three basic steps.

Step 1. Generate samples

We generate 100 sample points (t_n, m_n) randomly from $[0,T]\times S_d$ and 100 sample points (τ_n, z_n) from $\{T\}\times S_d$ according to respective probability densities ν_1 and ν_2, which we choose both as uniform distributions in our model. This way of generation corresponds to the background of the control problem well since the first equation represents the ongoing game process, whereas the second equation describes the situation for the end time of the game. Technically, when we take samples of $m_n, z_n \in S_d$, we generate each of its coordinate once at a time from the uniform distribution with range of $[0,1]$, and combine them together as a vector in $\mathbb{R}^d$ by taking their values in proportion to ensure the sum is 1.

Step 2. Calculate the loss function

We consider a parameterized class of functions $\big(f(t,m;\theta)\big)_\theta$ and the loss function for the MFCP:

$$J(h) = \underbrace{\Big\| -\frac{\partial h}{\partial t}(t,m;\theta) + \sum_{i\in[\![d]\!]} m_i H^i(t,m,D^ih(t,m;\theta))\Big\|^2_{S_1,\nu_1}}_{\text{differential operator}} + \underbrace{\Big\| h(T,m;\theta) - \sum_{i\in[\![d]\!]} m_i g(i,m)\Big\|^2_{S_2,\nu_2}}_{\text{terminal condition}}. \tag{4.1}$$

This L2 loss function, stands for Least Square Errors, calculates the sum of all the squared differences between the true value and predicted value at each sampled point.

Our goal is to find the parameter θ such that the function $h(t,m;\theta)$ minimizes the error $J(h)$, which measures how well h satisfies the PDE differential operator and terminal condition. If $J(h)=0$, then $h(t,m;\theta)$ is the true solution to the PDE.

Finally, with function (4.1), we obtain the loss function calculated at the randomly sampled points. Denote those points as $s_n=\{(t_n,m_n),(\tau_n,z_n)\}$ and the loss function as G, we have:

$$\begin{aligned} G(s_n,\theta_n) &= \Big(-\frac{\partial h}{\partial t}(t_n,m_n;\theta_n)+\sum_{i\in[\![d]\!]} m_{n,i}H^i(t_n,m_n,D^ih(t_n,m_n;\theta_n))\Big)^2 \\ &\quad + \Big(\ h(\tau_n,z_n;\theta_n)-\sum_{i\in[\![d]\!]} z_{n,i}g(i,z_n)\Big)^2 \end{aligned} \tag{4.2}$$

This function $G(s_n,\theta_n)$ is the target function that we are going to apply SGD to minimize the loss in the next step.

Step 3. Stochastic Gradient Descent

According to the optimization algorithm discussed in Section 3.4, we take descent steps at the random points s_n to improve our parameter of the deep neural network as:

$$\theta_{n+1} = \ \theta_n - \gamma\nabla_{\theta_n}G(s_n,\theta_n) \tag{4.3}$$

where γ is the learning rate. We repeat this procedure until the convergence criterion is satisfied, which means the value of loss function $J(f)$ is small enough. It is important to notice that the whole problem is strictly an optimization problem. Unlike typical machine learning applications where we are concerned with issues of underfitting and overfitting, here a smaller value of loss function represents a closer approximation of the solution to PDEs.

Remark 4.1. The steps $\nabla_{\theta_n}G(s_n,\theta_n)$ are unbiased estimates of $\nabla_{\theta_n}J(f(\cdot;\theta_n))$ such that

$$\mathbb{E}[\nabla_{\theta_n}G(s_n,\theta_n)|\theta_n] = \nabla_{\theta_n}J(f(\cdot;\theta_n)) \tag{4.4}$$

Thus, the SGD algorithm will on average take steps in a descent direction for the objective function J, which means the loss function will decrease after each iteration, that is,

$$J(f(\cdot;\theta_{n+1})) < J(f(\cdot;\theta_n)) \tag{4.5}$$

Thus, θ_{n+1} is a better parameter estimate than θ_n. Under technical conditions listed in [7], the algorithm θ_n will converge to a critical point of the objective function $J(f(\cdot;\theta_n))$ as $n\to\infty$:

$$\lim_{n\to\infty}\Big\|\nabla_{\theta_n}J(f(\cdot;\theta_n))\Big\| = 0. \tag{4.6}$$

In this way, we obtain the numerical result of the value function $V(t,m)$ from DGM neural network. Similarly, for the ODE stated as (2.9), we apply the DGM to find the approximate solution of $\mu(t)$. The question comes with the fact that the value function V has two input variables t and m, while the distribution μ is only affected by the single variable t. Therefore, the little difference between solving PDE for the value function and solving ODE for the distribution is the disparity of constructions of layers in the neural network, which is solved by embedding the Long Short-Term memory layer into Dense layers.

4.3. **Numerical Results of Mean Field Control Problem.** In the mean field control problem, the behavior of each agent is affected by congestion depending on cost function. Specifically, players would like to stay together at one particular state if the cost is low, while they do not want to be so crowded if this would lead to a high punishment.

The running cost represents the total energy of the system. In our model, we choose the running cost function f to be quadratic in the control defined as:

$$f^i(t,\alpha,m) = \frac{1}{2}\sum_{j\neq i} c_{i,j}\cdot\alpha_{i,j}^2 + f_0(t,m), \tag{4.7}$$

with the initial cost $f_0(t,m) = 20$ as a constant. Also define the terminal cost function g as a linear function such that:

$$g(i,m) = \sum_{i\in[\![d]\!]} m_i. \tag{4.8}$$

$\{c_{i,j}\}_{i,j\in[\![d]\!]}$ is a $d\times d$ matrix, where for $i\neq j, c_{i,j}$ represents the penalty rate of jumping from state i to state j.

Consider the case $d = 2$ with the coefficient matrix c:

$$c \;=\; \begin{bmatrix} 0 & 10 \\ 10 & 0 \end{bmatrix} \tag{4.9}$$

so that means the penalty rate of staying constant is 10 and jumping to another state is 1.

Let the range of control be $\mathcal{A} = [0,30]$. There is a 3D plot of the value function with x-axis as time t and y-axis as the distribution of the first state m_1 (note that $m_2 = 1 - m_1$).

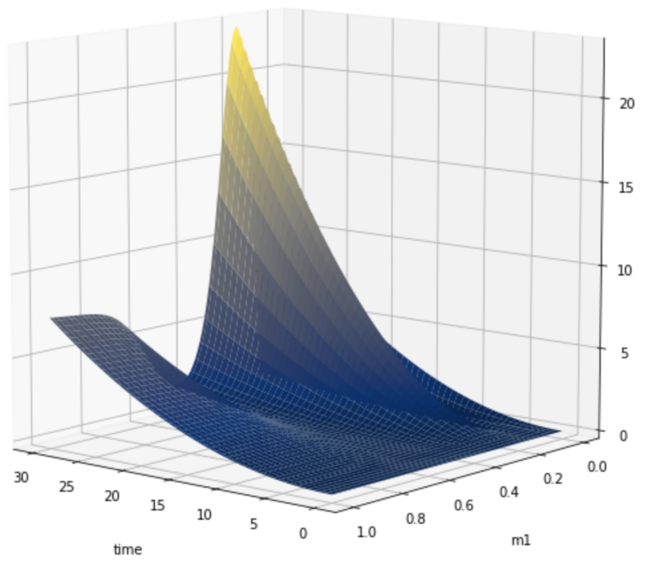

FIGURE 2. Value function of MFCP in dimension 2

Here we take the time range to be $[0,30]$ with the initial state as $(1,0)$. It is observed that the aggregate cost increases as time grows, and reaches its minimum at the distribution of $(0.5,0.5)$.

Now we extend the dimension of state space by setting $d = 5$. This time we have five choices of states in total, so we redefine the coefficient matrix c as:

$$c \ := \ \begin{bmatrix} 0 & 2 & 1 & 3 & 1 \\ 5 & 0 & 2 & 1 & 1 \\ 1 & 5 & 0 & 3 & 5 \\ 2 & 5 & 1 & 0 & 5 \\ 1 & 3 & 3 & 3 & 0 \end{bmatrix}. \tag{4.10}$$

To show the result, we plot the graph of the distribution $\mu(t)$ in Figure 3, which shows us the real-time state position of the agent in the game.

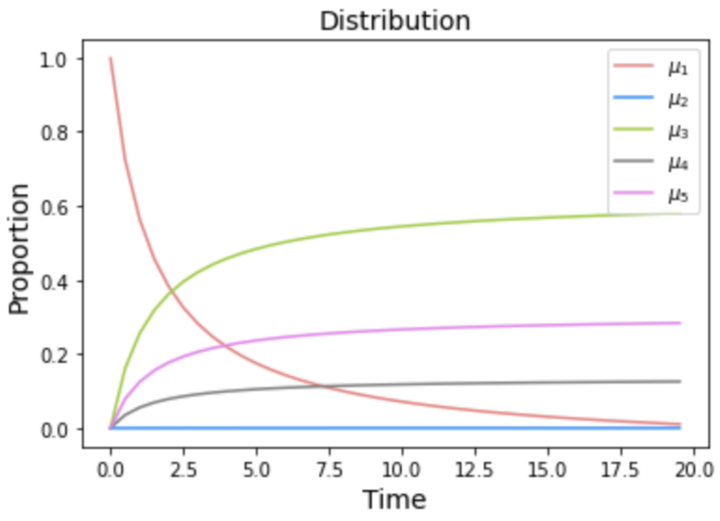

FIGURE 3. Distribution of MFCP in dimension 5

Moreover, setting the sampling times as 30, steps for SGD as 5, and the sample size as 100, we record the time that the DGM model takes to solve the PDE for value function V and distribution μ and their corresponding loss values when the dimensions are 2, 5, 10, 50, 100 in Table 1.

Dimension	Run time for V	Loss	Run time for μ	Loss
2	4m 39s	1.9629	7m 14s	0.2556
5	18m 38s	0.5093	27m 37s	0.0545
10	56m 24s	0.0171	1h 29m 23s	0.0227
20	3h 43m 36s	0.0542	4h 29m 19s	0.0037
50	8h 18m 15s	0.0216	9h 25m 31s	0.0023
100	TBD	TBD	TBD	TBD

TABLE 1. Run time

The whole TensorFlow code can be reached from this GitHub link: https://github.com/Jingruo/DGM-MFCP. Current codes are written for $d = 10$ with both sampling size and training stage set as 30.

5. Future Outlook

In this chapter, we are going to prove the convergence of two functions. First, we will show the error function converges to 0. Then we will prove the approximated result obtained from the numerical method converges to the true solution of the PDE.

To be specific, let V be the true solution to the PDE in (2.24). Let the L^2 error $J(f)$ measure how well the neural network f satisfies the differential operator and the terminal condition. Define ζ^n as the class of neural networks with n hidden units and f^n as a neural network with n hidden units which minimizes $J(f)$. We are going to prove:

$$\text{there exists } f^n \in \zeta^n \text{ such that } J(f^n) \to 0, \text{ as } n \to \infty, \text{and}$$
$$f^n \to \mu \text{ as } n \to \infty,$$

for a class of quasilinear parabolic PDEs with the principle term in divergence form under certain growth and smoothness assumptions on the nonlinear terms. The proof requires the joint analysis of the approximation power of neural networks as well as the continuity properties of partial differential equations.

The precise statement of the theorem and the presentation of the proof are going to be stated in the next two sections. Section 5.1 will prove that the neural network can satisfy the differential operator and terminal conditions well for sufficiently large n, that is, $J(f^n) \to 0$ as $n \to \infty$. Section 5.2 will contain convergence results of f^n to the solution V of the PDE as $n \to \infty$ in the appropriate space holds using compactness arguments.

References

[1] R. Basna, A. Hilbert, and V. N. Kolokoltsov. An epsilon-Nash equilibrium for non-linear Markov games of mean-field-type on finite spaces. *Commun. Stoch. Anal.*, 8(4):449–468, 2014.
[2] Y. Bengio, T. Mesnard, A. Fischer, S. Zhang, and Y. Wu. STDP-compatible approximation of backpropagation in an energy-based model. *Neural Comput.*, 29(3):555–577, 2017.
[3] A. Cecchin. Finite state N-agent and mean field control problems. *ESAIM Control Optim. Calc. Var.*, 27:Paper No. 31, 33, 2021.
[4] A. Cecchin and F. Delarue. Selection by vanishing common noise for potential finite state mean field games. *Comm. Partial Differential Equations*, 47(1):89–168, 2022.
[5] A. Cecchin and M. Fischer. Probabilistic approach to finite state mean field games. *Applied Mathematics & Optimization*, 2018.
[6] D. A. Gomes, J. Mohr, and R. R. Souza. Continuous time finite state mean field games. *Appl. Math. Optim.*, 68(1):99–143, 2013.
[7] J. Sirignano and K. Spiliopoulos. DGM: a deep learning algorithm for solving partial differential equations. *J. Comput. Phys.*, 375:1339–1364, 2018.