Notes on MATH 260 AA

Haosheng Zhou

Sept, 2022

Contents

Introduction to Neural Network

A handwriting recognition example: a dataset consisting of pictures of numbers to train the model. Human's recognition of number 9 is that it's a loop on the top with a vertical stroke in bottom right. The machine recognizes number using neural network, receive pictures as training samples and automatically infer the rules for recognition.

NN has neurons that can be activated or deactivated. It has input variables and for each input variable $x_i \in \mathbb{R}$ there is a weight parameter w_i and the linear combination $w \cdot x$, compared with a certain threshold, produces the output. For example, for sigmoid neurons, the output of a neuron is $\sigma(w \cdot x + b)$, with b as the bias working as the intercept in the linear combination. The sigmoid function is given by $\sigma(z) = \frac{1}{1+e^{-z}}$.

Different NN Structures

The simplest NN is given by the feedforward NN ('feedforward' refers to the fact that there's no edge going back, i.e. the values in hidden layer fed into input layer or hidden layer again), with 1 input layer, 1 hidden layer and 1 output layer, all neurons connected. The NN propogates forward, with the neurons in the hidden layer has the activated result of linear combinations of the real numbers in the input layer neurons. The number of neurons in a layer is called the width. In this specific example, the pictures are formed as 64×64 pixels with grey scales, so that will be the input of the NN.

On the other hand, recurrent NN (RNN) in NLP has loops and can deal with sequential data better and memorize previous inputs. The hidden layers can feed data to itself using a different activation function. Consider a 3-layer RNN and let h_i denote the number in the hidden neuron at time i, x_i as the input at time i into the hidden layer, y_i for the output at time i, then

$$\begin{cases} h_k = f(\beta^1 + w^{1,1}h_{k-1} + w^{1,2}x_k) \\ y_k = \tilde{f}(\beta^2 + w^2 h_k) \end{cases} \tag{1}$$

the point is that h_{k-1} is fed into h_k once more and the activation functions $f, \tilde{f}$ are different. RNN structure helps reduce the parameter space and has the cascade effect.

LSTM is a special case of RNN consisting of units. Each unit has a cell and three gates. The cell keeps track of the information received so far, the input gate captures to which extent new information will flow into the cell and the forget gate captures to which extent existing information remains in the cell, the output gate controls to which extent the information in the cell will be used to compute the output of the unit. The output of the unit can be understood as "given all words seen so far, what's the meaning of the sentence". The gates are formed as

$$\begin{cases} f_k = \rho_s(w_f x_k + u_f h_{k-1} + b_f) \\ i_k = \rho_s(w_i x_k + u_i h_{k-1} + b_i) \\ o_k = \rho_s(w_o x_k + u_o h_{k-1} + b_o) \end{cases} \tag{2}$$

and the cell is formed as $C_k = f_k \odot C_{k-1} + i_k \odot \rho(w_k x_k + u_k h_{k-1} + b_k)$ and the output of the k-th unit is denoted

$h_k = O_k \odot C_k$ where $\odot$ denotes the Hadamard product.

Training

Our objective is to let the NN learn the parameters to form estimates $\hat{w}, \hat{b}$ for the optimal w^*, b^*. One need a cost function to do this, the MSE cost function is formed as

$$C(w,b) = \frac{1}{2n}\sum_x ||Z - Z(x)||^2 \tag{3}$$

where Z is the output of the NN, $Z(x)$ is the true information given in the training set and n is the size of training inputs. Note that we will not choose things like $\frac{1}{2n}\sum_x \mathbb{I}_{Z(x)\neq Z}$ as the cost function since we hope to see that small changes in w, b will result in small changes in loss/cost.

In hand recognition, for example, $Z, Z(x) \in \mathbb{R}^{10}$ since there are altogether 10 possible predictions. $Z(x)$ is always a one-hot vector, having entry 1 at the location of the true label and 0 elsewhere. Entries of $Z(x)$ can stand for the probability of the label appearing given data x, but there are also other formations. Hand recognition is a supervised learning problem since true labels are given in the training set, while for unsupervised learning problems true labels are not known and one has to discover the pattern of the data without any prior information.

Now to find $\hat{w}, \hat{b}$ such that C is as small as possible, apply gradient descent for C. If $C(v)$ depends on parameter v, then $v' = v - \eta\nabla C(v)$ gives the update with learning rate $\eta > 0$. The gradient descent ensures that $C(v') \leq C(v)$ if the Armijo condition is satisfies. Note that the learning rate can't be too large since it needs to make Taylor expansion work. It can't be too small since that will result in slow convergence so in practice the learning rate changes with time.

For the example, we have updates

$$\begin{cases} w_k \to w_k' = w_k - \eta\frac{\partial C}{\partial w_k} \\ b_k \to b_k' = b_k - \eta\frac{\partial C}{\partial b_k} \\ C = \frac{1}{n}\sum_x C_x, C_x = ||Z - Z(x)||^2 \end{cases} \tag{4}$$

the problem is that computing C is too costly in a single update when we have a large training set. Using all information in the training set might not be a good choice. This leads to the appearance of stochastic gradient descent where one only just a part of the data to compute the gradients and to update parameters.

The thought of **stochastic gradient descent (SGD)** is to estimate ∇C by computing ∇C_x for a small set of samples $X_1, ..., X_M$ randomly chosen. M is the **mini-batch size**, typically taken as a power of 2 to speed up. So ∇C is estimated by $\frac{1}{M}\sum_{i=1}^M \nabla C(x_i)$. **Epoch** means the time we have exhausted the training inputs, if there are 60000 inputs and the batch size is 1000, then in each epoch the parameters are updated for 60 times (samples randomly chosen without replacement).

The **backpropagation** procedure is adopted to compute the gradients numerically (details ommitted, refer to Pytorch for realization in practice).

Problems & Improvements

Now under the settings described above, $C = \frac{1}{n}\sum_x ||y-a||^2$ where y is the truth, a is the output of NN where $a = \sigma(z), z = wx+b$. Then $\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x$, $\frac{\partial C}{\partial b} = (a-y)\sigma'(z)$. If σ is not chosen carefully, σ' can be pretty small, leading to **vanishing gradient issue**. That's why ResNet is appearing to fix this issue.

Cross entropy loss is always chosen especially in binary categorization problems.

$$C = -\frac{1}{n}\sum_x y\log(a) + (1-y)\log(1-a) \ (y, a \in [0,1]) \tag{5}$$

where $y = y(x)$ is the true data and $a = \sigma(z), z = wx+b$ is the output of NN. When C is close to 0 and $y = 0$, then a is close to 0, the same property holds when $y = 1$. So it's a good metric measuring the distance between y and a. Notice that

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x \frac{\sigma'(z)x_i}{\sigma(z)(1-\sigma(z))}(\sigma(z)-y) = \frac{1}{n}\sum_x x_i(\sigma(z)-y) \tag{6}$$

so the size of the gradient is actually proportional to the error $\sigma(z) - y$, and there's no vanishing gradient issue.

Paper Summary: Universal Approximation Results

We refer to *Multilayer Feedforward Networks are Universal Approximators by Kur Hornik, Maxwell Stinchcombe and Halber White* **and** *Recurrent Neural Networks Are Universal Approximators by Anton Schafer and Hans Zimmermann* **for the following materials**.

The universal approximation results of NN focus on the theoretical functionality of NN that for any error tolerance level $\forall\varepsilon > 0$, NN can approximate any Borel measurable function well enough regardless of the activation function and the input space environment. We first introduce the sketch of the proof that a 3-layer NN is a universal approximator and then use the result to prove that RNN is also a universal approximator.

Settings and Definitions

Denote $\mathscr{A}^r$ as the set of all affine functions $A(x) = w^T x + b$ from $\mathbb{R}^r$ to $\mathbb{R}$. For a given activation function G, a 3-layer NN first maps x to $A_j(x)$ for $A_j \in \mathscr{A}^r$ (w, b are the weight and bias between input layer and hidden layer) and then compose the activation function to get $G(A_j(x))$ as the input of the j-th neuron in the hidden layer. The output of such NN is then formed as $\sum_{j=1}^q \beta_j G(A_j(x))$, so β denotes the weight between hidden layer and output layer. As a result, we denote

$$\Sigma^r(G) = \left\{ g : \mathbb{R}^r \to \mathbb{R} : g(x) = \sum_{j=1}^q \beta_j G(A_j(x)) \right\} \tag{7}$$

as the collection of all mappings from x, the input of NN, to $g(x)$, the output of NN with activation function taken as G. For the activation function, we add some mild conditions that G is a **sigmoid function** if it's increasing and bounded, $G(\lambda) \to 1\ (\lambda \to +\infty), G(\lambda) \to 0\ (\lambda \to -\infty)$ (extracted from the property of the true sigmoid function).

Now to measure whether the approximation is good enough, let's define some density arguments in different sense. For metric ρ, **ρ-dense** is the same as that defined in topology, i.e. $S \subset X$ is ρ-dense in $T \subset X$ if and only if

$$\forall \varepsilon > 0, \forall t \in T, \exists s \in S, \rho(s,t) < \varepsilon \tag{8}$$

so all elements in T can be approximated arbitrarily well by some element in S. Set C^r to be the space of all continuous functions on $\mathbb{R}^r$ and M^r to be the space of all measurable functions on $\mathbb{R}^r$. Being **uniformly dense on compact sets in** C^r means that for any compact subset $K \subset C^r$, the set is dense under the metric $\rho_K(f,g) = \sup_{x\in K} |f(x) - g(x)|$ induced by the uniform norm on K. Moreover, if there is a probability measure μ on $(\mathbb{R}^r, \mathscr{B}^r)$ (Borel measurable space), the metric ρ_μ is defined as

$$\rho_\mu(f,g) = \inf \{\varepsilon > 0 : \mu(|f-g| > \varepsilon) < \varepsilon\} \tag{9}$$

one might notice that convergence under this metric is equivalent to convergence in measure μ, i.e. $\rho_\mu(f_n, f) \to 0\ (n \to \infty)$ iff $f_n \xrightarrow{\mu} f\ (n \to \infty)$.

Although we want to prove that $\sum^r(G)$ is generally a universal approximator, we do have to start the proof on a space with more mappings, i.e.

$$\Sigma\Pi^r(G) = \left\{ g : \mathbb{R}^r \to \mathbb{R} : g(x) = \sum_{j=1}^{q} \beta_j \prod_{k=1}^{l_j} G(A_{jk}(x)) \right\} \tag{10}$$

so the main thought is to first prove the universal approximation identity to continuous functions for good enough activation functions and a larger space $\Sigma\Pi^r(G)$, and then weaken the approximation to continuous functions into the approximation to measurable functions, weaken the condition of activation functions and shrink the space to $\Sigma^r(G)$.

Sketch of Proof for Feedforward NN

The main theorem used here is the Stone-Weierstrass theorem, telling us that when A, as a set of continuous functions, behaves well enough on a compact set K, its uniform closure includes all continuous functions on K.

Theorem 1. ***(Stone-Weierstrass Theorem)*** *A is an algebra (closed under addition, multiplication, scalar multiplication) of continuous functions on compact set K. If A separates points on K ($\forall x, y \in K, x \neq y, \exists f \in A, f(x) \neq f(y)$) and vanishes at no point of K ($\forall x \in K, \exists f \in A, f(x) \neq 0$), then the uniform closure of A contains all real continuous functions on K, i.e. A is dense under ρ_K.*

We start from the case where G is continuous and prove that it's enough for $\Sigma\Pi^r(G)$ to be uniformly dense on compact set in C^r.

Theorem 2. ***(Approx of Continuous Functions)*** *$\forall G : \mathbb{R} \to \mathbb{R}$ continuous nonconstant, $\Sigma\Pi^r(G)$ is uniformly dense on compact set in C^r.*

Proof. Take $K \subset C^r$ as any compact set, $\Sigma\Pi^r(G)$ is always an algebra on K regardless of G and it separates points on K. To see why it separates points, $\forall x, y \in K, x \neq y$, let's pick $a, b \in \mathbb{R}, a \neq b, G(a) \neq G(b)$ (G nonconstant) and pick $A \in \mathscr{A}^r$ such that it maps x to a, y to b (A affine, always possible), so $G(A(x)) = G(a) \neq G(b) = G(A(y))$.

Let's also verify that $\Sigma\Pi^r(G)$ vanishes at no point of K. Pick $b, G(b) \neq 0$ and set $w = 0$ so $A(x) = b, G(A(x)) = G(b) \neq 0$ for $\forall x \in K$.

By Stone-Weierstrass theorem, $\Sigma\Pi^r(G)$ is ρ_K-dense, so proved. □

Now let's keep G to be continuous but approximate measurable functions instead of continuous ones. Note that the definition of measurable function always depend on the sigma algerba specified (on $(\Omega, \mathscr{F}, \mathbb{P})$ a measurable function f is such that $\forall A \in \mathscr{F}, f^{-1}(A) \in \mathscr{B}_{\mathbb{R}}$). However, here we introduce a probability measure and prove that the approximation is universally good for every probability measure selected.

Theorem 3. ***(Approx of Measurable Functions, Continuous Activation Function)*** *$\forall G : \mathbb{R} \to \mathbb{R}$ continuous nonconstant, $\forall r$ (which is the dimension of input), $\forall \mu$ as probability measure on $(\mathbb{R}^r, \mathscr{B}^r)$, $\Sigma\Pi^r(G)$ is ρ_μ-dense in M^r.*

Proof. The proof is easy since it's just the application of facts in measure theory. By the theorem above, $\Sigma\Pi^r(G)$ is uniformly dense on compact set in C^r, so it must be ρ_μ-dense for any probability measure μ (uniform convergence on compact sets implies convergence in measure). To change the approximation to C^r into M^r, notice that when the measure is finite, C^r is always ρ_μ dense in M^r so we can always approximate C^r first and use continuous functions in C^r to approximate measurable functions in M^r, proved.

□

Let's then weaken the condition of the theorem by not requiring the activation function G to be continuous.

Theorem 4. ***(Approx of Measurable Functions, Sigmoid Activation Function)*** *$\forall G : \mathbb{R} \to \mathbb{R}$ sigmoid, $\forall r$ (which is the dimension of input), $\forall \mu$ as probability measure on $(\mathbb{R}^r, \mathscr{B}^r)$, $\Sigma\Pi^r(G)$ is ρ_μ-dense in M^r and uniformly dense on compact set in C^r.*

Proof. By the last theorem, we just have to prove that there exists F as continuous sigmoid activation function such that $\Sigma\Pi^r(G)$ is uniformly dense on compact set in $\Sigma\Pi^r(F)$. To prove this, we just have to prove that for any F as continuous sigmoid activation function, $\prod_{k=1}^l F(A_k(x))$ can always be uniformly approximated by functions in $\Sigma\Pi^r(G)$. This is because functions in $\Sigma\Pi^r(F)$ are just linear combinations of functions of the form $\prod_{k=1}^l F(A_k(x))$.

Here to approximate continuous sigmoid function by general sigmoid functions, we have to use a lemma that for F continuous sigmoid function fixed and any sigmoid function G, $\forall \varepsilon > 0, \exists H_\varepsilon \in \Sigma^1(G)$ such that $\sup_{\lambda \in \mathbb{R}} |F(\lambda) - H_\varepsilon(\lambda)| < \varepsilon$.

Now $\forall \varepsilon > 0$, we can find $H_\delta(x) = \sum_t \beta_t G(A_t^1(x))$ such that $\sup_{\lambda\in\mathbb{R}} |F(\lambda) - H_\delta(\lambda)| < \delta$ is small enough such that $\sup_{x\in\mathbb{R}^r} |\prod_{k=1}^l F(A_k(x)) - \prod_{k=1}^l H_\delta(A_k(x))| < \varepsilon$ (here we are using the uniform continuity of $\prod_{k=1}^l x_k$ on compact set, that's where the boundedness of sigmoid function comes into play and we plug in λ as affine function $A_k(x)$). Now observe that $\prod_{k=1}^l H_\delta(A_k(x)) \in \Sigma\Pi^r(G)$ concludes the proof (note that $H_\delta(A_k(x)) = \sum_t \beta_t G(A_t^1(A_k(x)))$, and the composition of two affine functions is still affine).

□

Finally, we can now go back from $\Sigma\Pi^r(G)$ to $\Sigma^r(G)$ and prove that a 3-layer feedforward NN is already a universal approximator.

Theorem 5. ***(Universal Approximation Theorem for 3-layer Feedforward NN)*** *$\forall G : \mathbb{R} \to \mathbb{R}$ sigmoid, $\forall r$ (which is the dimension of input), $\forall \mu$ as probability measure on $(\mathbb{R}^r, \mathscr{B}^r)$, $\Sigma^r(G)$ is ρ_μ-dense in M^r and uniformly dense on compact set in C^r.*

Proof. We only need to prove that $\Sigma^r(G)$ is uniformly dense on compact set in C^r. That's because this implies that $\Sigma^r(G)$ is ρ_μ-dense in C^r and C^r is ρ_μ-dense in M^r so the theorem is proved. (the same procedure as we have done above).

The proof of the first fact is based on the observation that by taking G as the cosine activation function in the theorem above. Since it's too technical, we omit the details here. □

Remark. *The universality comes from the fact that* ***the density argument holds for any sigmoid activation function, any dimension of input and any input space environment*** *μ. In other words, an NN with simple structure can approximate a measurable function arbitrarily well (under the metric of convergence in measure μ) on a compact set regardless of the selection of activation function, the dimension of NN and the probability measure selected on the input space.*

Remark. *Actually,* ***the same argument holds for the 3-layer feedforward NN that has multiple outputs****, i.e. the output is a vector in $\mathbb{R}^s$. Just replace ρ_μ with its multi-dimensional version $\rho_\mu^s(f,g) = \sum_{i=1}^s \rho_\mu(f_i, g_i)$ $(f, g : \mathbb{R}^r \to \mathbb{R}^s)$ and approximate each component respectively and reduce the error tolerance level such that the sum of error tolerance level adds up to ε.*

Result for RNN

RNN has a different structure from feedforward NN that each neuron will feed the state output back to itself as the state input in the next round. Despite the structural difference, RNN also has universal approximation property directly coming from the universal approximation property of feedforward NN.

Now we have a dynamical system in discrete time

$$\begin{cases} s_{t+1} = g(s_t, u_t) \\ y_t = h(s_t) \end{cases} \tag{11}$$

where $s_t \in \mathbb{R}^J$ is the state at time t, $u_t \in \mathbb{R}^I$ is the external input at time t and $y_t \in \mathbb{R}^n$ is the output at time t. So the system has some dynamics for state evolution that would be influenced by external inputs, and different outputs are produced for different current state. Here we propose the RNN as

$$\begin{cases} s_{t+1} = f(As_t + Bu_t - \theta) \\ y_t = Cs_t \end{cases} \tag{12}$$

with all actions to be linear except the activation function f. Now the input u_t has dimension I so we naturally consider $\Sigma^{I,n}(f)$ which is the set of functions that maps the input of feedforward NN to its output (the output has dimension n). We denote its element as $NN(x)$ such that

$$NN(x) = Vf(Wx - \theta) \tag{13}$$

where V is the weight matrix between hidden layer and output layer, W, θ are the weight matrix and bias between input layer and hidden layer, and the action of f is componentwise. We denote $RNN^{I,n}(f)$ as the set of functions

$$\begin{cases} s_{t+1} = f(As_t + Bu_t - \theta) \\ y_t = Cs_t \end{cases} \tag{14}$$

in RNN for fixed activation function $f : \mathbb{R}^J \to \mathbb{R}^J$ (which is a series of states and outputs as a dynamical system). Of course we hope that RNN can replicate any dynamical system

$$\begin{cases} s_{t+1} = g(s_t, u_t) \\ y_t = h(s_t) \end{cases} \tag{15}$$

well enough in the universal sense. The result is given below.

Theorem 6. ***(Universal Approximation Theorem for RNN)*** *Assume $g : \mathbb{R}^J \times \mathbb{R}^I \to \mathbb{R}^J$ is measurable and $h : \mathbb{R}^J \to \mathbb{R}^n$ is continuous in the real dynamical system with finite time horizon $t = 1, 2, ..., T$. Then any dynamical system of the form*

$$\begin{cases} s_{t+1} = g(s_t, u_t) \\ y_t = h(s_t) \end{cases} \tag{16}$$

can be approximated by an element of $RNN^{I,n}(f)$ arbitrarily well for some continuous sigmoid activation function f.

Proof. The proof is mainly repeated use of the universal approximation theorem for feedforward NN. First approximate the state dynamics $s_{t+1} = g(s_t, u_t)$ by an NN to get the estimation of state as $\overline{s}_t$ at time t. Next approximate $h(\overline{s}_t)$ by feedforward NN again and merge the results in two parts to conclude the proof.

□

Paper Summary: The Expressive Power of NNs: A View from the Width

Let n be input dimension and $f : \mathbb{R}^n \to \mathbb{R}$, there exists network with width $\leq n + 4$ which approximates f, and this cannot be done if width is $\leq n$. As a result, this shows a phase transition that when the width is less than the threshold, the approximation is weak while when the width is higher than the threshold the NN can always do well enough.

Let $\mathscr{A}$ be feedforward NN and $F_{\mathscr{A}}$ be its output, the activation functions are all taken as ReLU function $\max\{x, 0\}$ with h to be depth and w to be width of NN (the maximum width across all layers).

Theorem 7. ***(Width $n+4$ Enough for Approx)*** *$\forall f \in L^1(\mathbb{R}^n), \varepsilon > 0, \exists \mathscr{A}$ with width $w_{\mathscr{A}} \leq n + 4$ such that $\int_{\mathbb{R}^n} |\mathscr{F}_{\mathscr{A}} - f|\, dx < \varepsilon$.*

Theorem 8. ***(Width n Not Enough for Approx)*** *$\forall f \in L^1(\mathbb{R}^n), f \neq 0, \forall \mathscr{A}$ with width $w_{\mathscr{A}} \leq n$ such that $\int_{\mathbb{R}^n} |\mathscr{F}_{\mathscr{A}} - f|\, dx = \int_{\mathbb{R}^n} |f|\, dx$ or $+\infty$.*

Theorem 9. ***(Version on Bounded Set)*** *$\forall f \in C([-1, 1]^n), f$ nonconstant in any direction, $\exists \varepsilon > 0, \forall \mathscr{A}$ with width $w_{\mathscr{A}} \leq n - 1$, $\int_{[-1,1]^n} |\mathscr{F}_{\mathscr{A}} - f|\, dx \geq \varepsilon$.*

Other Results

Hornik also shows that a 3-layer feedforward NN is enough to do universal approximations to any measurable function and its derivative at the same time, which is quite important for solving stochastic control problems since in HJB equation we always have to deal with approximating derivatives. Most work on shallow networks are done.

Deep Learning and Stochastic Control, Direct Approach

Overview

For deep learning algorithms solving stochastic control problems, there are some classical algorithms. The algorithm from *Han-E* approximates the control directly, the one from *Hure-Pham-Bachouch-Langrene* uses DPP, the one from *Han-Hu* considers stochastic problem with delay. Those algorithms do not require reformulation of the problem.

On the PDE approach, *Sirignano-Spiliopoulous* has an algorithm called deep Galerkin method that solves HJBE directly, and the same authors have put up physical informed NN to deal with PDE. *Beck-Becker-Cheridito-Jentzen-Neufield* has the deep splitting method.

On the FBSDE approach, *E-Han-Jentzen* has deep BSDE solver, *Hure-Pham-Warin* has DBDP (deep backward dynamic programming).

Direct Parametrization

Consider minimization problem with state dynamics

$$dX_t^\alpha = b(t, X_t^\alpha, \alpha_t)\, dt + \sigma(t, X_t^\alpha, \alpha_t)\, dB_t \tag{17}$$

and the objective function to **minimize**

$$\mathbb{E}\left[\int_0^T f(t, X_t^\alpha, \alpha_t)\, dt + g(X_T^\alpha)\right] \tag{18}$$

we are always finding optimal Markovian control.

Method of ***Han-E***

One approach is from *Han-E* proposed on 2016 Neurips workshop. The idea is to **make use of the NN to approximate the function relationship between optimal control $\hat{\alpha}_t$ and state X_t at a given time** t. In the original paper, $\alpha_{NN}^t(x_t;\theta_t)$ denotes a feedforward NN at time t that approximates the optimal control $\hat{\alpha}_t$ with input $\check{X}_t = x_t$ as the state at time t (we denote $\check{X}_t$ as a simulation of X_t since we cannot know the true state value). Here θ_t denotes the parameter of NN and has to be trained. The loss function is simply discretized w.r.t. time partition $0 = t_0 < t_1 < ... < t_N = T$ as

$$\mathbb{E}\left[\sum_n f(t_n, \check{X}_{t_n}, \alpha_{NN}^{t_n}(\check{X}_{t_n};\theta_{t_n}))\Delta t + g(\check{X}_T)\right] \tag{19}$$

where the state dynamics is simulated in the Euler-Maruyama scheme to form $\check{X}$

$$\check{X}_{t_{n+1}} = \check{X}_{t_n} + b(t, \check{X}_{t_n}, \alpha_{NN}^{t_n}(\check{X}_{t_n}; \theta_{t_n}))\Delta t + \sigma(t, \check{X}_{t_n}, \alpha_{NN}^{t_n}(\check{X}_{t_n}; \theta_{t_n}))\Delta B_{t_n} \tag{20}$$

we update θ_{t_n} w.r.t. the loss function by SGD to complete the training.

Remark. *The expectation is the loss function is calculated by Monte Carlo and notice that the simulation of $\check{X}$ also depends on the selection of θ_{t_n}. To be clear with how the method works, we fix all parameters θ_{t_n} (according to prior knowledge or just randomly initialized). For fixed parameters, we simulate the trajectory of state process to get $\check{X}_{t_n}$ at each time point t_n. After that, at each time point t_n, send in $\check{X}_{t_n}, \theta_{t_n}$ as inputs of the NN to get the output $\alpha_{NN}^{t_n}$. By finishing all those operations, we are able to calculate a single realization of the discretized loss*

$$\sum_n f(t_n, \check{X}_{t_n}, \alpha_{NN}^{t_n}(\check{X}_{t_n}; \theta_{t_n}))\Delta t + g(\check{X}_T) \tag{21}$$

do this for a lot of times and use Monte-Carlo to get an approximation of the expectation

$$\mathbb{E}\left[\sum_n f(t_n, \check{X}_{t_n}, \alpha_{NN}^{t_n}(\check{X}_{t_n}; \theta_{t_n}))\Delta t + g(\check{X}_T)\right] \tag{22}$$

and then we can update all parameters θ_{t_n} by SGD for a single time.

As a result, for a single update of $\theta_{t_0}, ..., \theta_{t_N}$, we have to simulate the state process for many times and run the NN at each time point for many times. As a result, there's a natural update of the method that we can use $\alpha_{NN}(t, x_t; \theta)$ instead, which means that we ***only maintain a single NN with time t and simulated state process $\check{X}_t$ as inputs and the optimal control at time t as output****. The advantage is that now we do not have to maintain an NN at each time point, so we can discretize the time into finer parts and the parameter θ is uniform in time so the parameter space has lower dimension.*

The advantage of this method:

- No requirement on Hamiltonian
- Does not matter if the diffusion coefficient contains control variable or not
- Easy to accommodate constraints. For example, if we require $\alpha_t \geq 0$, then apply ReLU function on the output layer to guarantee non-negativity. If we require $X_t \geq 0$, then add $\sum ReLU(-\check{X}_{t_n})$ to the loss as penalty term. More generally, if we require $C(X_t, \alpha_t) = 0$, then add $\sum_n ||C(\check{X}_{t_n}, \alpha_{NN}^{t_n}(\check{X}_{t_n}; \theta_{t_n}))||^2$ to loss, if we require $C(X_t, \alpha_t) \geq 0$, then add $\sum_n ReLU(-C(\check{X}_{t_n}, \alpha_{NN}^{t_n}(\check{X}_{t_n}; \theta_{t_n})))$ to loss, so it can deal with all kinds of constraints. (when the algorithm ends, do a projection onto the space of admissible controls)

the disadvantage is that there's no proof on the behavior of this algorithm. We train all α at once and there's vanishing and exploding gradient problems for large number of time steps (when the number of time subintervals is too much since the NN is too deep). This approach is **global in time** in that the optimal controls at all the time points are trained simultaneously and the loss function is relevant to the behavior at all time points.

Remark. *In the context above, we only consider optimal Markovian control $\alpha_{t_n} = NN(X_{t_n};\theta)$. If we want to see open-loop control, we can set $\alpha_{t_n} = NN(X_0, B_0, B_{t_1}, ..., B_{t_n};\theta)$ and for closed loop control, we can set $\alpha_{t_n} = NN(X_0, X_{t_1}, ..., X_{t_n};\theta)$. This is the freedom of the direct parametrization method.*

NNContPI

The second approach is proposed by Bachouch-Hure-Langrene-Pham that is **local in time**, which means that the training of optimal controls happens backwardly, the optimal control at the last time point is trained first and fixed forever while the optimal controls at earlier time points are trained after that.

The NNContPI algorithm cuts the time horizon into N_T sub-intervals $0 = t_0 < t_1 < ... < t_{N_T} = T$ and assumes that the optimal control at time $t_{n+1}, ..., t_{N_T-1}$ is already learned with NN parameters $\hat{\theta}_{n+1}, ..., \hat{\theta}_{N_T-1}$ (θ_n is the parameter in the NN that describes the functional relationship between state and optimal control at time point t_n), then the optimal control at time t_n is approximated by $\alpha_{t_n}(\cdot, \hat{\theta}_n)$, where

$$\hat{\theta}_n = \arg\min_\theta \mathbb{E}\left[f(t_n, \check{X}_{t_n}, \alpha_{t_n}(\check{X}_{t_n};\theta))\Delta t + \sum_{n'=n+1}^{N_T-1} f(t_{n'}, \check{X}_{t_{n'}}, \alpha_{t_{n'}}(\check{X}_{t_{n'}};\hat{\theta}_{n'}))\Delta t + g(\check{X}_T)\right] \tag{23}$$

here the sum part follows **the already trained optimal controls** $\alpha_{t_{n+1}}(\cdot, \hat{\theta}_{t_{n+1}}), ..., \alpha_{t_{N_T-1}}(\cdot, \hat{\theta}_{t_{N_T-1}})$ and SGD is performed for the former part w.r.t. θ. Note that when θ is changed, it will affect the simulation of $\check{X}_{t_n}$ (the simulation uses Euler scheme and depends on $\alpha_{t_n}(\cdot, \theta), \alpha_{t_{n+1}}(\cdot, \hat{\theta}_{t_{n+1}}), ..., \alpha_{t_{N_T-1}}(\cdot, \hat{\theta}_{t_{N_T-1}})$).

Remark. *One might be able to find out that $\hat{\theta}_n$ is the best parameter we can find to approximate the optimal control $\hat{\alpha}_{t_n}$ using the state X_{t_n} at time point t_n. The update of $\hat{\theta}_n$ actually* ***makes use of DPP****, saying that if one always sticks to the best control after time t_{n+1}, the best control at time t_n shall minimize the sum of the cost in (t_n, t_{n+1}) and the cost of sticking to the best control in (t_{n+1}, T). Naturally, since DPP is made use of,* ***the method only works for finding optimal Markovian controls****.*

On the other hand, one might find that choosing the starting point when simulating $\check{X}_{t_n}$ is a big problem. Since we are only assuming that we have figured out the best control after time t_{n+1}, it's impossible for us to simulate $\check{X}$ from the beginning (the best control between time $(0, t_{n+1})$ is totally unknown). As a result, we have to simulate $\check{X}$ starting from time t_n for each fixed parameter θ so we are required to provide an estimate of $\check{X}_{t_n}$, which is totally unclear how to derive. In practice, one always assigns a random but reasonable value for $\check{X}_{t_n}$ to start the simulation.

- Learn α_{t_n} sequentially and backwardly so there's propagation of error
- Smaller but many NNs, can deal with large N_T (global in time approach may encounter vanishing and exploding gradient problem for large N_T) since the local in time approach has more shallow NN
- It's still unclear how to choose the starting point when simulating $\check{X}_{t_n}$
- There is theoretical analysis, for large N_T value function and the control will be approximated well (universal approximation sense)

Hybrid-Now

Another algorithm called Hybrid-Now has further approximation on value functions (the cost-to-go). At time t_{n+1} denote the value of value function as $V_{t_{n+1}}(\cdot, \tilde{\theta}_{n+1})$ where

$$\hat{\theta}_n \in \arg\min_{\theta} \mathbb{E}\left[f(t_n, \check{X}_{t_n}, \alpha(\check{X}_{t_n}; \theta))\Delta t + V_{t_{n+1}}(\check{X}_{t_{n+1}}; \tilde{\theta}_{t_{n+1}})\right] \tag{24}$$

so there's no need to simulate the state process in time (t_n, N_T) but it accumulates new error. $\tilde{\theta}_n$ is derived by

$$\tilde{\theta}_n \in \arg\min_{\theta} \mathbb{E}\left[\left|f(t_n, X_{t_n}, \alpha_{t_n}(X_{t_n}; \hat{\theta}_n))\Delta t + V_{t_{n+1}}(X_{t_{n+1}}; \tilde{\theta}_{t_{n+1}}) - V_{t_n}(X_{t_n}; \theta)\right|^2\right] \tag{25}$$

so $\hat{\theta}$ approximates the control and $\tilde{\theta}$ approximates the value function at time point t_n. The updates are still making use of **DPP**, the property of value function, but in an **alternating way**. First fix the parameter for the approximation of optimal control and optimize the parameter for the approximation of value function, then fix the parameter for the approximation of value function and optimize the parameter for the approximation of optimal control. It's obvious that this method also **only works for finding optimal Markovian controls**.

Delayed Control Problems

Han-Hu extends Han-E's method on delayed problems. The **delayed SDE** is formed as

$$\begin{cases} dX_t = b(t, \underline{X}_t, \alpha_t)\, dt + \sigma(t, \underline{X}_t, \alpha_t)\, dB_t & t \in [0, T] \\ X_t = \varphi_t & t \in [-\delta, 0] \end{cases} \tag{26}$$

where $\underline{X}_t$ denotes the trajectory of $X_{[t-\delta,t]}$ and δ is a fixed positive number (but possibly unknown). The dynamics allows the time for reaction formed as time delay δ so the evolution of X_t will be affected by $X_{[t-\delta,t]}$, the state in a small delay interval. The optimal control here shall minimize the cost w.r.t. the delayed process $\underline{X}_t$

$$J(\alpha) = \mathbb{E}\left[\int_0^T f(t, \underline{X}_t, \alpha_t)\, dt + g(\underline{X}_T)\right] \tag{27}$$

We can compare feedforward NN and LSTM used as approximation for optimal control. $\alpha_{t_n} = NN(X_{[t_n-\delta,t_n]}; \theta)$ is the FF scheme and $\alpha_{t_n} = Wh_n + b$ is the LSTM scheme, where h_n is the output of an LSTM at the n-th unit. In numerical experiments, **LSTM performs better** (RNN fits better into problems with time structure and path dependence) and another advantage is that we **don't need to know the δ a priori when applying LSTM**.

Deep Learning and Stochastic Control, PDE Approach

Deep Galerkin Method

The method is put up by *Sirignano and Spiliopoulos*. They use NN to approximate value function u and all derivatives are computed by auto-differentiation. They consider the PDE with initial value and boundary value conditions

$$\begin{cases} \partial_t u - Lu = 0 & (t,x) \in [0,T] \times Q \\ u(0,x) = u_0(x) & x \in Q \\ u(t,x) = \Gamma(t,x) & (t,x) \in [0,T] \times \partial Q \end{cases} \tag{28}$$

where L is any operator. Parametrize u by an NN $\tilde{u}(t,x;\theta)$ so the loss

$$J(\theta) = \eta||\partial_t \tilde{u}(t,x;\theta) - Lu(t,x;\theta)||^2 + \eta_I||\tilde{u}(0,x;\theta) - u_0(x)||^2 + \eta_{BC}||\tilde{u}(t,x;\theta) - \Gamma(t,x)||^2 \tag{29}$$

and we sample (t,x) using Latin hypercube sampling in high dimension. η, η_I, η_{BC} are hyperparameters adjusting the weights of three losses to make sure that each of them is taken into consideration.

For example in finance, u is the option price and n is the dimension of state process, i.e. the number of assets. As a result, approximating $\frac{\partial u}{\partial x}$ takes $O(n)$ time but approximating $\frac{\partial^2 u}{\partial x^2}$ takes $O(n^2)$ time. Second derivatives are time-costing to evaluate, it takes time $O(n^2 N_{Batch} k)$ where k is the dimension of parameter. As a result, a Monte Carlo method is proposed to do fast approximations of second derivatives. (section 3)

- This framework is applicable for any PDE
- There exists a proof for some PDE. For any error tolerance, there always exist $\tilde{u}(t,x;\theta)$ such that the loss can be controlled by the error tolerance. Moreover, there exists a sequence of NN $\tilde{u}^n(t,x;\theta)$ that converges to the true solution u in the almost surely sense as $n \to \infty$. (section 7 in the paper)
- Don't know how to sample on unbounded domain where there's no boundary value condition and don't know where is the area of interest. For example, in Merton problem with Black-Scholes model for stock price, x just have to be positive but no other information is provided.
- Don't know how to deal with unknown boundary conditions that appears in the optimal stopping problem. In American option pricing, ∂Q is the boundary on which one is indifferent between exercising the option and keeping the option, so the boundary itself is unknown and is the core of the problem.

Remark. *Closely related work: physics-informed NN from Rassi-Petikams-Karniadakis. Deep Ritz method from (E-Yu), solve variational problems using weak formulation.*

Semi-linear PDE & FBSDE

Notice that when **diffusion coefficient is free of control**, we can take trace terms out of the sup in HJBE to get (H denotes the Hessian of v at (t,x))

$$\partial_t v(t,x)+\frac{1}{2}Tr(\sigma(t,x)\sigma^T(t,x)H)+\sup_{\alpha}\{b(t,x,\alpha)\cdot\partial_x v(t,x)+f(t,x,\alpha)\}=0 \tag{30}$$

so the sup is taken when $\alpha^*=\alpha^*(t,x,\partial_x v)$ (assume such optimal control is unique and can be represented as a function) and plug back into HJBE to find

$$\partial_t v(t,x)+\frac{1}{2}Tr(\sigma(t,x)\sigma^T(t,x)H)+b(t,x,\alpha^*(t,x,\partial_x v))\cdot\partial_x v(t,x)+f(t,x,\alpha^*(t,x,\partial_x v))=0 \tag{31}$$

which is a **semi-linear PDE**.

To solve this semi-linear PDE, we would first notice that it's always possible to rewrite

$$b(t,x,\alpha^*(t,x,\partial_x v))\cdot\partial_x v(t,x)+f(t,x,\alpha^*(t,x,\partial_x v))=\mu(t,x)\cdot\partial_x v(t,x)+h(t,x,\sigma^T(t,x)\cdot\partial_x v) \tag{32}$$

notice that the trivial case is that $\mu(t,x)=0$, but **numerical experiments show that the more we put into $\mu(t,x)$, the better the method performs**.

Consider using non-linear Feynman-Kac formula to characterize it as decoupled FBSDE, note that $\mathcal{X}$ **is a newly created process to work as the FSDE, but it's not the state process X!** The **decoupled FBSDE** is given by

$$\begin{cases} d\mathcal{X}_t=\mu(t,\mathcal{X}_t)\,dt+\sigma(t,\mathcal{X}_t)\,dB_t\\ \mathcal{X}_0=x\\ dY_t=-h(t,\mathcal{X}_t,Z_t)\,dt+Z_t\,dB_t\\ Y_T=g(\mathcal{X}_T)\end{cases} \tag{33}$$

this FBSDE is decoupled since the dynamics of $\mathcal{X}_t$ has nothing to do with Y_t,Z_t.

Remark. *To see the* ***motivation*** *of the decoupled FBSDE, let's consider plugging in*

$$Y_t=v(t,\mathcal{X}_t) \tag{34}$$

to the BSDE (since the typical explanation for Y_t is the value of option and that for Z_t is the Delta-Hedging strategy, *that's why we assume that $Y_t = v(t, \mathcal{X}_t)$). We find that according to Ito formula*

$$\begin{aligned} dY_t &= dv(t, \mathcal{X}_t) && (35)\\ &= \partial_t v(t, \mathcal{X}_t)\, dt + \partial_x v(t, \mathcal{X}_t)\, d\mathcal{X}_t + \frac{1}{2}\partial_{xx} v(t, \mathcal{X}_t)\, d\left\langle \mathcal{X}, \mathcal{X}\right\rangle_t && (36)\\ &= \left[\partial_t v(t, \mathcal{X}_t) + \mu(t, \mathcal{X}_t)\cdot \partial_x v(t, \mathcal{X}_t) + \frac{1}{2} Tr(\sigma(t, \mathcal{X}_t)\sigma^T(t, \mathcal{X}_t) H)\right] dt + \sigma^T(t, \mathcal{X}_t)\cdot \partial_x v(t, \mathcal{X}_t)\, dB_t && (37) \end{aligned}$$

and compare coefficients with the BSDE

$$dY_t = -h(t, \mathcal{X}_t, Z_t)\, dt + Z_t\, dB_t \tag{38}$$

to find two conditions to satisfy

$$\begin{cases} \partial_t v(t, \mathcal{X}_t) + \mu(t, \mathcal{X}_t)\cdot \partial_x v(t, \mathcal{X}_t) + \frac{1}{2} Tr(\sigma(t, \mathcal{X}_t)\sigma^T(t, \mathcal{X}_t) H) = -h(t, \mathcal{X}_t, Z_t)\\ \sigma^T(t, \mathcal{X}_t)\cdot \partial_x v(t, \mathcal{X}_t) = Z_t \end{cases} \tag{39}$$

notice that

$$h + \mu\cdot \partial_x v = b\cdot \partial_x v + f = -\partial_t v - \frac{1}{2} Tr(\sigma\sigma^T H) \tag{40}$$

where the first equation follows from the definition of μ, h and the second equation follows from the original HJBE. So ***the first condition is making sure that the original HJBE holds****.*

After solving out Y_t, Z_t in the FBSDE, however, we have to return back to the solution of the PDE. A simple way to see ***the connection between the solution to the FBSDE and the solution to the HJBE*** *is from the second condition that*

$$\begin{cases} Y_t = v(t, \mathcal{X}_t)\\ Z_t = \sigma^T(t, \mathcal{X}_t)\cdot \partial_x v(t, \mathcal{X}_t) \end{cases} \tag{41}$$

this explains why the components of h are formed as $h(t, x, \sigma^T(t, x)\cdot \partial_x v)$ where the third component contains the diffusion coefficient*.*

To explain the initial condition of $\mathcal{X}_t$ and the terminal condition of Y_t, notice that $Y_T = v(T, \mathcal{X}_T) = g(\mathcal{X}_t)$ since function g gives the terminal cost. Since $Y_0 = v(0, \mathcal{X}_0)$ is the objective function we want to optimize and $X_0 = x$ gives the initial state, we actually want to optimize the control w.r.t. $v(0, x)$ so setting $\mathcal{X}_0 = x$ makes sense. ***Notice again that X_t and $\mathcal{X}_t$ have totally different dynamics but they start from the same initial value x at time*** *0.*

Typically we find solution in space $(\mathcal{X}_t, Y_t, Z_t) \in \mathbb{H}^2_T(\mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^d)$, where d is the dimension of BM and n is the dimension of $\mathcal{X}_t$.

Deep BSDE

Proposed by *E-Han-Jentzen*, the method makes use of the characterization of semi-linear PDE using FBSDE. *Han-Long* gives theoretical proof of Deep BSDE method. The method turns the problem of solving HJBE into solving the FBSDE, **a variational problem** that

$$\begin{cases} \inf_{Y_0,Z_t} \mathbb{E}||Y_T - g(\mathcal{X}_T)||^2 \\ s.t.\ \mathcal{X}_t = x_0 + \int_0^t \mu(s,\mathcal{X}_s)\,ds + \int_0^t \sigma(s,\mathcal{X}_s)\,dB_s \\ s.t.\ Y_t = Y_0 - \int_0^t h(s,\mathcal{X}_s,Z_s)\,ds + \int_0^t Z_s\,dB_s \end{cases} \tag{42}$$

notice that the basic idea here is to guess with NN where Y starts and what Z_t is. **For existing guess Y_0, Z_t, we simulate $\mathcal{X}, Y$ by FBSDE and try to find the best guess such that the terminal condition $Y_T = g(\mathcal{X}_T)$ of the BSDE holds**. The NN in the method is formed as $Y_0 \sim NN(X_0;\theta), Z_{t_i} \sim NN(X_{t_i};\tilde{\theta}_i)$ where **an NN guesses Y_0 with the information $\mathcal{X}_0$ and parameter θ and a family of NNs guess Z_{t_i} with the information $\mathcal{X}_{t_i}$ and parameter $\tilde{\theta}_{t_i}$ at each time point t_i**. Simulate $\check{\mathcal{X}}_t, \check{Y}_t$ using Euler scheme so the loss is actually discretized as

$$\frac{1}{M}\sum_{j=1}^M ||\check{Y}_T^j - g(\check{\mathcal{X}}_T^j)||^2 \tag{43}$$

and parameters $\theta, \tilde{\theta}_{t_n}$ are updated by SGD. This is **global in time** since the simulations at all time points happen simultaneously.

- There exists partial proof for the method. $\mathbb{E}||Y_T - g(\mathcal{X}_T)||^2$ can be bounded by the sum of time discretization error and $\inf \mathbb{E}||Y_0 - NN(\mathcal{X}_0;\theta)||^2$ (best approximation error of Y_0),$\sum_{i=1}^{N_T-1} \mathbb{E}||\tilde{Z}_{t_i} - NN(\mathcal{X}_{t_i};\tilde{\theta}_{t_i})||^2 \Delta t_i$ (approximation error of Z_t) where $\tilde{Z}_{t_i} = \frac{1}{\Delta t_i}\mathbb{E}[\int_{t_i}^{t_{i+1}} Z_s\,ds|\check{\mathcal{X}}_{t_i}]$.
- Don't need to worry about which domain to sample. As the method proceeds, the area with more trajectories of $\mathcal{X}$ is of more interest. Since $Y_t = v(t,\mathcal{X}_t)$, the method spontaneously investigates $v(t,x)$ $(x \in A)$ where $\mathcal{X}_t$ tend to take values near A so **the parts of interest automatically receives more attention and is more accurately approximated**.
- Only works with drift control problem, no control allowed in the diffusion coefficient

DBDP

Deep backward dynamic programming (DBDP) is a local in time algorithm stated by *Hure-Pham-Warin*. The idea is to solve Y_t backwardly and sequentially. The method is still based on the decomposition that

$$b(t,x,\alpha^*(t,x,\partial_x v))\cdot\partial_x v(t,x)+f(t,x,\alpha^*(t,x,\partial_x v))=\mu(t,x)\cdot\partial_x v(t,x)+h(t,x,\sigma^T(t,x)\cdot\partial_x v) \tag{44}$$

and the non-linear Feynman-Kac formula to characterize the solution of the HJBE using a BSDE

$$\begin{cases} d\mathcal{X}_t=\mu(t,\mathcal{X}_t)\,dt+\sigma(t,\mathcal{X}_t)\,dB_t \\ \mathcal{X}_0=x \\ dY_t=-h(t,\mathcal{X}_t,Z_t)\,dt+Z_t\,dB_t \\ Y_T=g(\mathcal{X}_T) \end{cases} \tag{45}$$

but now take the local in time approach to integrate the BSDE from t_i to t_{i+1} to get

$$v(t_{i+1},\mathcal{X}_{t_{i+1}})=v(t_i,\mathcal{X}_{t_i})-h(t_i,\mathcal{X}_{t_i},\sigma^T\partial_x v(t_i,\mathcal{X}_{t_i}))\Delta t+\sigma^T\partial_x v(t_i,\mathcal{X}_{t_i})\Delta B_{t_i} \tag{46}$$

where the correspondence is given by $Y_{t_i}=v(t_i,\mathcal{X}_{t_i}), Z_{t_i}=\sigma^T\partial_x v(t_i,\mathcal{X}_{t_i})$.

The method approximate Y_{t_i},Z_{t_i} using NN with the terminal value $Y_{t_{N_T}}$ initialized as g. At each time point t_i, assume that all parameters after the current time $\hat{\theta}_{i+1},...,\hat{\theta}_{N_T-1}$ has already been trained, the problem is formed as the variational problem

$$\inf_{Y_{t_i}(\cdot;\theta),Z_{t_i}(\cdot;\theta)}\mathbb{E}||Y_{t_{i+1}}(\cdot;\hat{\theta}_{i+1})-Y_{t_i}(\cdot;\theta)+h(t_i,\mathcal{X}_{t_i},Z_{t_i}(\cdot;\theta))\Delta t-Z_{t_i}(\cdot;\theta)\Delta B_{t_i}||^2 \tag{47}$$

so the optimal θ is just the best $\hat{\theta}_i$. BY doing that backwardly, we can train the NN at each time point. The advantage of local in time approach is that we can always divide time horizon into a lot small intervals, but such approach only applies for Markovian control problem with no control in diffusion coefficient.

Deep Splitting

The method is stated by *Beck-Becker-Cheridito-Jentzen-Newfield* and the idea is to apply the linear Feynman-Kac formula. Based on previous representations, HJBE is transformed into

$$\partial_t v(t,x) + \frac{1}{2}Tr(\sigma(t,x)\sigma^T(t,x)H) + \mu(t,x)\cdot\partial_x v(t,x) + h(t,x,\sigma^T(t,x)\cdot\partial_x v) = 0 \tag{48}$$

so the non-linearity comes from h, if there's no h term then v can be represented by Feynman-Kac formula. If $h = h(t,x)$, then by Feynman-Kac

$$v(t,x) = \mathbb{E}\left[\int_t^T h(s,\mathcal{X}_s)\,ds + v(T,\mathcal{X}_T)\middle|\mathcal{X}_t = x\right] \tag{49}$$

So we can apply Feynman-Kac from t_i to t_{i+1} and approximate the integral of the non-linear part to get

$$v(t_i,x) \sim \mathbb{E}\left[h(t,\mathcal{X}_{t_{i+1}},\partial_x v(t_{i+1},\mathcal{X}_{t_{i+1}}))\Delta t + v(t_{i+1},\mathcal{X}_{t_{i+1}}))\middle|\mathcal{X}_{t_i} = x\right] \tag{50}$$

and it's solved backwardly, also a local in time approach. The conditional expectation is formed in the L^2 minimization way to be calculated numerically.

Controlled Diffusion Coefficient

Recall the HJBE

$$\partial_t v(t,x)+\frac{1}{2}Tr(\sigma(t,x,\alpha)\sigma(t,x,\alpha)^T H)+\sup_\alpha\{b(t,x,\alpha)\cdot\partial_x v(t,x)+f(t,x,\alpha)\}=0 \tag{51}$$

and most of the discussion on solving PDE is based on the assumption that σ does not contain α, which is always not the case. To solve it, we rewrite it as

$$\partial_t v(t,x)+\frac{1}{2}Tr(\Sigma(t,x)\Sigma(t,x)^T H)+\mu(t,x)\cdot\partial_x v(t,x)+h(t,x,\partial_x v(t,x),H)=0 \tag{52}$$

so the HJBE solution corresponds to the **decoupled second order BSDE (2BSDE)**

$$\begin{cases} d\mathcal{X}_t=\mu(t,\mathcal{X}_t)\,dt+\Sigma(t,X_t)\,dB_t\\ dY_t=-h(t,\mathcal{X}_t,Y_t,Z_t)\,dt+Z_t^T\Sigma\,dB_t\\ Y_T=g(\mathcal{X}_T)\\ dZ_t=\mathscr{A}_t\,dt+\Gamma_t\Sigma\,dB_t\\ Z_T=\partial_x g(\mathcal{X}_T)\end{cases} \tag{53}$$

where the relationship with the HJBE is specified with

$$\begin{cases} Y_t=v(t,\mathcal{X}_t)\\ Z_t=\partial_x v(t,\mathcal{X}_t)\\ \Gamma_t=H(t,\mathcal{X}_t)\\ \mathscr{A}_t=\mathcal{L}\partial_x v(t,\mathcal{X}_t)\end{cases} \tag{54}$$

where $\mathcal{L}$ is the infinitesimal generator of $\mathcal{X}_t$ and (Y_t,Z_t,Γ_t) is the solution to the 2BSDE.

Beck-E-Jentzen extend Deep BSDE to solve 2BSDE to parametrize both Z_t,Γ_t and try to match terminal conditions. *Pham-Warin-Germain* extend DBDP to solve 2BSDE.

Deep Learning and Stochastic Control, BSDE Approach

Pontryagin Maximum Principle

Let's consider solving the maximization stochastic control problem and define the Hamiltonian

$$H(t,x,y,z,\alpha) = b(t,x,\alpha)\cdot y + \sigma(t,x,\alpha)\cdot z. + f(t,x,\alpha) \tag{55}$$

solve $\alpha^* = \arg\max_\alpha H(t,x,y,z,\alpha)$ where α^* is a function in t,x,y,z. Plug such optimal control $\hat{\alpha}_t = \alpha^*(t,X_t,Y_t,Z_t)$ back to get the **coupled FBSDE**

$$\begin{cases} dX_t = b(t,X_t,\hat{\alpha}_t)\,dt + \sigma(t,X_t,\hat{\alpha}_t)\,dB_t \\ dY_t = -\partial_x H(t,X_t,Y_t,Z_t,\hat{\alpha}_t)\,dt + Z_t\,dB_t \\ Y_T = \partial_x g(X_T) \end{cases} \tag{56}$$

solve the coupled FBSDE to get the optimal control. The verification condition requires the concavity of g and the concavity of H in (x,α) to ensure that the solution is actually an optimal control.

Fully Coupled FBSDE Solver

Shaolin-Shige-Ying-Xichuan has proposed in the paper *Three algorithms for solving high-dimensional fully-coupled FBSDEs through deep learning* three algorithms to numerically solve fully coupled FBSDE. The idea is to adopt the **optimal control criterion**, i.e. specify some of the processes as state process and some pf the processes as control process in a new stochastic control problem. Consider the following fully coupled FBSDE derived after applying Pontryagin maximum principle and plugging in the optimal control $\alpha_t^* = \alpha^*(t,X_t,Y_t,Z_t)$

$$\begin{cases} dX_t = b(t,X_t,Y_t,Z_t)\,dt + \sigma(t,X_t,Y_t,Z_t)\,dB_t \\ X_0 = x \\ dY_t = -f(t,X_t,Y_t,Z_t)\,dt + Z_t\,dB_t \\ Y_T = g(X_T) \end{cases} \tag{57}$$

the **first algorithm** is to specify Z_t as control and X_t, Y_t as state. So the control is the function of the state which is unknown for now, naturally, we specify this relationship with an NN as $Z_{t_i} = \phi(X_{t_i}, Y_{t_i}; \theta_i)$ (time horizon $[0,T]$ is discretized into $0 = t_0 < t_1 < ... < t_{N_T} = T$ with an NN maintained at each time point). In each iteration, approximate Z_{t_i} with X_{t_i}, Y_{t_i} by NN and then approximate $X_{t_{i+1}}, Y_{t_{i+1}}$ by Euler scheme. In order to deal with BSDE, we adopt the idea to guess the initial value Y_0 of BSDE and then match the terminal condition, similar to what we have done in Deep BSDE. So the loss function is formed as

$$\mathbb{E}||Y_T - g(X_T)||^2 \tag{58}$$

and besides the parameters of NNs, Y_0 is also formed and updated as the parameter. Refer to Fig. 1 for the procedure graph.

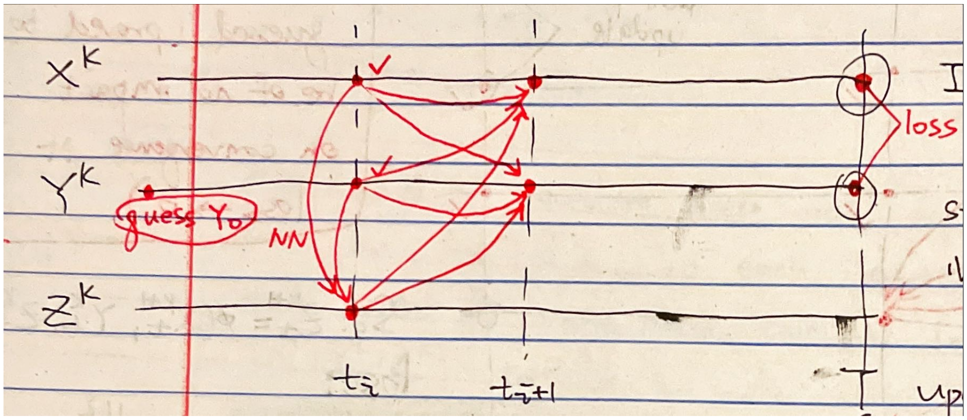

Figure 1: Procedure within Iteration for Algorithm 1

Check means that the information at this point is already known at the beginning of the iteration.
NN means the place NN approximation is used and the arrows show the dependency and the procedure in applying Euler scheme.

The **second algorithm** is to specify u_t, Z_t as control and X_t as state where u_t denotes the process Y_t in the FSDE. In other words, we pretend that the Y_t in the FSDE and the Y_t in the BSDE are different. So the control is the function of the state which is unknown for now, naturally, we specify this relationship with two NNs maintained at each time point $u_{t_i} = \phi^1(X_{t_i}; \theta_i^1), Z_{t_i} = \phi^2(X_{t_i}; \theta_i^2)$. In each iteration, approximate Z_{t_i}, u_{t_i} with X_{t_i}, Y_{t_i} by NN and then approximate $X_{t_{i+1}}, Y_{t_{i+1}}$ by Euler scheme. Note that we not only have to match the terminal condition of the BSDE but also have to match u_t, Y_t since those two processes are actually the same. The loss function is formed as

$$\mathbb{E}\left[||Y_T - g(X_T)||^2 + \int_0^T ||Y_t - u_t||^2 \, dt\right] \tag{59}$$

and when we guess Y_0, we can use the approximation $\phi^1(X_0; \theta_0^1)$ since u_0 should be equal to Y_0. Refer to Fig. 2 for the procedure within a single iteration.

The **third algorithm** is to use Picard iteration. We specify Z^{k+1} as control and X^{k+1}, Y^k, Z^k as state where the superscript k means the process X in the k-th iteration. The Picard iteration can be written in the SDE form as

$$\begin{cases} dX_t^{k+1} = b(t, X_t^{k+1}, Y_t^k, Z_t^k) \, dt + \sigma(t, X_t^{k+1}, Y_t^k, Z_t^k) \, dB_t \\ dY_t^{k+1} = -f(t, X_t^{k+1}, Y_t^{k+1}, Z_t^{k+1}) \, dt + Z_t^{k+1} \, dB_t \end{cases} \tag{60}$$

so now the simulation of trajectories is related across different iterations (while in the first two algorithm each iteration is independent of the other iteration). Since control is a function of the state, we set $Z_{t_i}^{k+1} = \phi(X_{t_i}^{k+1}, Y_t^k, Z_t^k; \theta_i)$. To

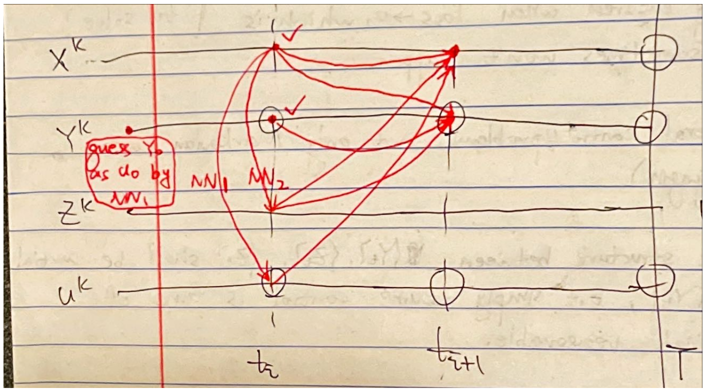

Figure 2: Procedure within Iteration for Algorithm 2

Check means that the information at this point is already known at the beginning of the iteration.
NN means the place NN approximation is used and the arrows show the dependency and the procedure in applying Euler scheme.

start the algorithm, now we have to have initial guess for two processes $\left\{Y_t^0\right\}_{t\in[0,T]}, \left\{Z_t^0\right\}_{t\in[0,T]}$ in the 0-th iteration (instead of simply guessing Y_0).

In each iteration, approximate $Z_{t_i}^{k+1}$ with $X_{t_i}^{k+1}, Y_{t_i}^k, Z_{t_i}^k$ by NN and then approximate $X_{t_{i+1}}^{k+1}, Y_{t_{i+1}}^{k+1}$ by Euler scheme. The loss function is formed as

$$\mathbb{E}||Y_T - g(X_T)||^2 \tag{61}$$

to match the terminal condition. Actually, the initial guess of $\left\{Y_t^0\right\}, \left\{Z_t^0\right\}$ does not matter if the loss converges to 0 since it will always converge to the true solution to the FBSDE. Refer to Fig. 3 for the procedure across different iterations.

- All three algorithms are **global in time**, so the time partition cannot be fine enough.
- The convergence of algorithms is only ensured when loss converges to 0, which is not necessarily the case in practice, and additional assumptions are needed (algorithm 1 & 2 requires the existence and uniqueness condition of the solution to the FBSDE, algorithm 3 requires the condition such that Picard iteration converges for the FBSDE).
- This approach can solve control problems other than Markovian control, but also open-loop control problems etc.
- The algorithms ignore the structure contained within X_t, Y_t, Z_t since we actually know from the meaning of BSDE that Z_t should be some kind of derivative of Y_t, but in those algorithms simple parametrization are used and no further structures are considered.
- In numerical experiments, algorithm 3 behaves well and algorithm 2 sometimes does not converge well. However, notice that algorithm 3 is different from the previous two since approximations of processes in different iterations

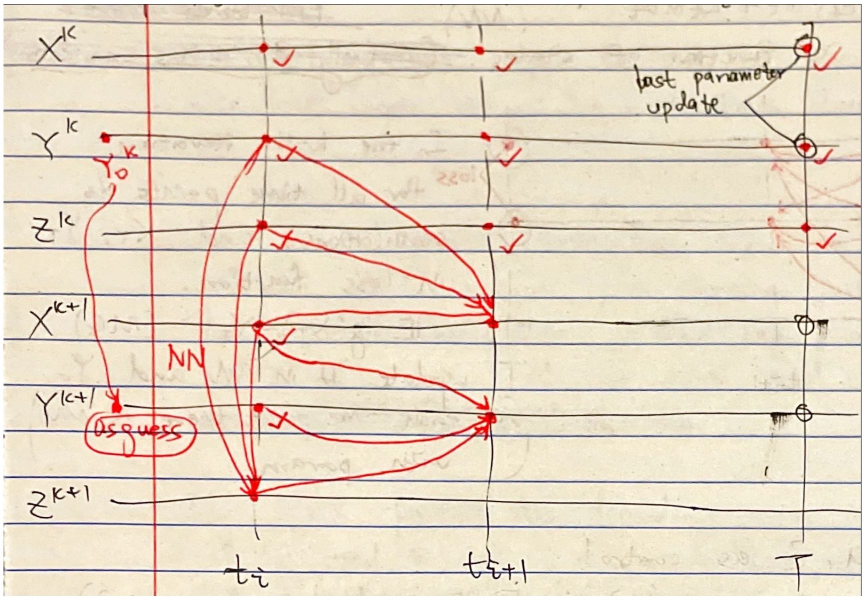

Figure 3: Procedure across Iteration for Algorithm 3

Check means that the information at this point is already known at the beginning of the iteration. NN means the place NN approximation is used and the arrows show the dependency and the procedure in applying Euler scheme.

are related. This might bring with the measurability problem since we will be using θ_i to approximate $Z_{t_i}^{k+1}$ but θ_i has been updated using the information contained in $\left\{Z_t^k\right\}_{t\in[0,T]}$.

Remark. *For the realization of algorithm 1 with an FBSDE with closed-form solution, refer to Python Notebook on Solving Fully Coupled FBSDE by Deep Learning for the one under Pytorch settings.*

Stochastic Differential Games

Formulation

Consider the game with N players, with state dynamics

$$dX_t = b(t, X_t, \alpha_t)\, dt + \sigma(t, X_t, \alpha_t)\, dB_t \tag{62}$$

where $X_t \in \mathbb{R}^d, B_t \in \mathbb{R}^m, b \in \mathbb{R}^d, \sigma \in \mathbb{R}^{d\times m}$. So X_t is the state vector containing the states of each player controlled by

$$\alpha = (\alpha^1, ..., \alpha^N) \in \mathscr{A} \tag{63}$$

the notation (α^{-i}, β^i) denotes the control that fixes $\alpha^1, ..., \alpha^{i-1}, \alpha^{i+1}, ..., \alpha^N$ and changes α^i to β^i. If the state X_t^i is only controlled by α_t^i, then it's called the private state of player i. B_t^i can be *i.i.d.* BM or can be correlated with a common noise B_t^0 included.

Remark. *When $N \to \infty$, if we are in private state setting with i.i.d. BM and b^i, σ^i identical for all players, we have* ***homogeneous mean field game (MFG) without common noise****. When b^i, σ^i are different for each player, it's the heterogeneous MFG.*

The whole system is still coupled since X_t appears in the dynamics of X_t^i. The cost functional for player i is formed as

$$J^i(\alpha) = \mathbb{E}\left[\int_0^T f^i(s, X_s, \alpha_s)\, ds + g^i(X_T)\right] \tag{64}$$

and shall be minimized.

Optimality Criterion

There are different notions of optimality in non-cooperative games. The **Pareto optimality** is defined such that $\alpha^* = (\alpha^{1,*}, ..., \alpha^{N,*})$ is Pareto optimal if there is no α such that

$$\begin{cases} \forall i, J^i(\alpha) \le J^i(\alpha^*) \\ \exists i_0, J^{i_0}(\alpha) < J^{i_0}(\alpha^*) \end{cases} \tag{65}$$

so α^* is Pareto optimal of there does not exist a better control α that improves at least one of the players' cost.

The **weakly Pareto optimality** is defined such that α^* is weakly Pareto optimal if there is no α such that

$$\forall i, J^i(\alpha) < J^i(\alpha^*) \tag{66}$$

weakly Pareto optimality just requires that there does not exists a better α that strictly improves all players' cost. (weaker requirement)

The **Nash equilibrium (NE)** is defined such that α^* is optimal if

$$\forall i, \forall \alpha^i, J^i(\alpha^*) \leq J^i(\alpha^{-i,*}, \alpha^i) \tag{67}$$

so if all other player's control is fixed, nobody has the incentive to deviate from NE.

Type of Information Sets

In order to make decisions, players have to use the available information in the game. By restricting the information set, one may get different kinds of NE. The **open-loop NE** allows

$$\alpha^i_t = \varphi^i(t, X_0, B_{[0,t]}) \tag{68}$$

where φ^i is a deterministic function. It's easy from math point of view and decisions are made based on background noises.

The **closed-loop NE** allows

$$\alpha^i_t = \varphi^i(t, X_{[0,t]}) \tag{69}$$

and the **Markovian NE** allows

$$\alpha^i_t = \varphi^i(t, X_t) \tag{70}$$

here we are always assuming the complete information setting, i.e. each player can see all other players' state realization if necessary. For open-loop NE, it's often solved by Pontryagin maximum principle and for Markovian NE, it's often solved by DPP and HJBE.

Open-Loop NE

The admissible set is set as

$$\mathbb{H}^2 = \left\{ \alpha \; prog, \mathbb{E}\int_0^T ||\alpha_t||^2 \, dt < \infty \right\} \tag{71}$$

with the filtration $\mathscr{F}_t$ generated by the m-dimensional BM.

Find the Hamiltonian for player i

$$H^i(t, x, y, z, \alpha) = b(t, x, \alpha) \cdot y + \sigma(t, x, \alpha) \cdot z + f^i(t, x, \alpha) \tag{72}$$

to get the adjoint BSDEs associated with player i that

$$\begin{cases} dY_t^{i,j} = -\partial_{x^j} H^i(t, X_t, Y_t^{i,j}, Z_t^{i,j}, \alpha_t) \, dt + Z_t^{i,j} \, dB_t \\ Y_T^{i,j} = \partial_{x^j} g^i(X_T) \; (j = 1, 2, ..., N) \end{cases} \tag{73}$$

where $Z_t^{i,j} \, dB_t = \sum_{k=1}^m Z_t^{i,j,k} B_t^k$.

The sufficient condition for open-loop NE $\hat{\alpha}$ is

$$H^i(t, \hat{X}_t, \hat{Y}_t, \hat{Z}_t, \hat{\alpha}_t) = \inf_{\alpha^i} H^i(t, \hat{X}_t, \hat{Y}_t, \hat{Z}_t, (\hat{\alpha}^{-i}, \alpha^i)) \tag{74}$$

where H^i is convex in (x, α^i) and g^i is convex and for player i, all other players' strategies are viewed as unknown variables (compared to the Markovian case).

So in order to find the open-loop NE, we define H^i for each player and find $\hat{\alpha}^i(t, x, (y^{i,1}, ..., y^{i,N}), (z^{i,1}, ..., z^{i,N}))$ as the minimizer of H^i, find a solution to the FBSDE that

$$\begin{cases} dX_t = b(t, X_t, \hat{\alpha}(t, X_t, (Y_t^1, ..., Y_t^N), (Z_t^1, ..., Z_t^N))) \, dt + \sigma(t, X_t, \hat{\alpha}(t, X_t, (Y_t^1, ..., Y_t^N), (Z_t^1, ..., Z_t^N))) \, dB_t \\ X_0 = x \\ dY_t^{i,j} = -\partial_{x^j} H^i(t, X_t, Y_t^{i,j}, Z_t^{i,j}, \hat{\alpha}(t, X_t, (Y_t^1, ..., Y_t^N), (Z_t^1, ..., Z_t^N))) \, dt + Z_t^{i,j} \, dB_t \\ Y_T^{i,j} = \partial_{x^j} g^i(X_T) \; (i, j = 1, 2, ..., N) \end{cases} \tag{75}$$

note that there's one state dynamics shared by all players and for player i there are N BSDEs for $(Y_t^{i,1}, Z_t^{i,1}), ..., (Y_t^{i,N}, Z_t^{i,N})$. If this FBSDE is solvable, we get the open-loop NE $\hat{\alpha}$ by plugging the solution.

Markovian NE

To apply the Pontryagin maximum principle to solve out the Markovian NE, the only change in the Hamiltonian of player i is to **view** $\alpha^1, ..., \alpha^{i-1}, \alpha^{i+1}, ..., \alpha^N$ **as a function of** t, X_t **since all other players will react according**

to the state feedback. We denote the Hamiltonian of player i as

$$H^i(t,x,y,z,\alpha^i,\alpha^1(t,x),...,\alpha^{i-1}(t,x),\alpha^{i+1}(t,x),...,\alpha^N(t,x)) \tag{76}$$

so when computing $\partial_{x^j} H^i$ as the driver in the adjoint BSDE, there are extra terms due to the dependence of other players' control on state variable x. This is the difference between open-loop and Markovian FBSDEs derived from Pontryagin maximum principle.

Remark. *To understand the difference between open-loop and Markovian control, note that for open-loop control, it's assuming that each player has its own "plan". When they see a trajectory of BM, they stick to their own plan to make decisions without noticing how other players behave. As a result, when the BM trajectory does not change and the controls of the players are deviating from the NE, the open-loop NE control does not change since the randomness has not changed and the players will still stick to their plans. However, the Markovian NE control will change since the feedback from other players has changed.*

The other way is to use the DPP and HJBE with the value function of player i given by

$$v^i(t,x) = \inf_{\alpha^i} \mathbb{E}\left[\int_t^T f^i(s,X_s,\alpha_s)\,ds + g^i(X_T)\Big| X_t = x\right] \tag{77}$$

so the HJBE is given by

$$\partial_t v^i + \inf_{\alpha^i}\left\{f^i(t,x,\alpha) + L^\alpha v\right\} = 0 \tag{78}$$

where L^α is the infinitesimal generator of the state dynamics. One can find the optimal $\hat{\alpha}^i$ that achieves the sup and the HJBE can be reduced to N coupled equations with the optimal control of all the players $\hat{\alpha}$ plugged in.

Fictitious Play

Although we have introduced algorithms solving fully coupled FBSDE and PDE systems, it's not realistic for us to apply those algorithms to numerically solve out NE. The largest problem is the dimension, since if there are N players in the game there will be N^2 processes $Y_t^{i,j}$ and N^2m processes $Z_t^{i,j,k}$ to solve numerically, which is a large burden even for $N = 50$. The other difficulty lies in the parallel formulation. Since the FBSDE and PDE are always coupled, the numerical simulation of all the players depends on each other, making it hard to parallel.

The idea of **fictitious play** solves both problems at once. The idea is to decouple the game into separate optimal control problems. One can solve them repeatedly and hope to see a convergent sequence of strategies with the limit as the NE.

For example, in the prisoner dilemma, denote C as "confess" and S as "remain silent". We start from any strategy (S,S) to do the fictitious play. In this round, if prisoner 1 sticks to S, then for prisoner 2, betraying and switching to C would be the best choice. Similarly, fixing the action of prisoner 2, prisoner 1 would choose to confess. So in the next round the control becomes (C,C) and it's obvious that it will always stay at (C,C) since it's already an NE. We see that by averaging the historical control taken by two prisoners and denote α^k to be the control tuple in the k-th round,

$$\alpha^0 = \delta_{(S,S)}, \alpha^1 = \frac{1}{2}\delta_{(S,S)} + \frac{1}{2}\delta_{(C,C)}, ..., \alpha^n = \frac{1}{n+1}\delta_{(S,S)} + \frac{n}{n+1}\delta_{(C,C)} \to \delta_{(C,C)} \ (n \to \infty) \tag{79}$$

so it converges to the true NE. (Those are called **mixed strategies** since it's a distribution on the space of all available strategies)

Remark. *Every non-cooperative game with finite players and finite many pure strategies has NE. However, such game does not necessarily have unique NE so it's possible for the sequence of strategy to swing between two NEs. To ensure the convergence of fictitious play, one needs some further conditions.*

Return to stochastic differential games, we start from initial strategy $\alpha^0 = (\alpha^{1,0}, ..., \alpha^{N,0})$ so player i takes strategy $\alpha^{i,k}$ at stage k. The optimal control problem faced by player i is formed as

$$dX_t^{i,k} = b(t, X_t^{i,k}, (\alpha^i, \alpha^{-i,k}))\, dt + \sigma(t, X_t^{i,k}, (\alpha^i, \alpha^{-i,k}))\, dB_t \tag{80}$$

$$J^i(\alpha^i, \alpha^{-i,k}) = \mathbb{E}\left[\int_0^T f^i(t, X_t^{i,k}, (\alpha^i, \alpha^{-i,k}))\, dt + g^i(X_T^{i,k})\right] \tag{81}$$

where $\alpha^{-i,k}$ is the strategy given in the previous stage. Player i optimizes $J^i(\alpha^i, \alpha^{-i,k})$ and the optimizer is denoted $\alpha^{i,k+1}$. This is now formed as a **decoupled problem** since all other players' strategies are fixed in the problem faced by player i. We expect to see that α^n converges when $n \to \infty$.

Remark. *We are not taking average over all historical strategies. It's too costly to memorize all historical controls since each control $\alpha^1, ..., \alpha^k$ is actually formed as an NN and there's no easy way to form the linear combination of the outputs of NN without consuming much storage space.*

There are algorithms based on fictitious play for multiple players but still high-dimensional and solved by deep learning algorithms introduced above. For open-loop NE, $\alpha_t = \phi(B_{[0,t]})$ by direct parametrization proposed by *Han-E*. For Markovian NE, $\alpha_t = \phi(t, X_t)$ by Deep BSDE solver proposed by *Han-E-Jentzen.*

Remark. *The convergence of fictitious play can be understood in the following way. Let's assume $\alpha^n \to \alpha^\infty$ $(n \to \infty)$ so given other players are using $\alpha^{-i,\infty}$ and player i solves the control problem with value $J^i(\alpha^i, \alpha^{-i,\infty})$ to get the best such strategy α^i as $\alpha^{i,\infty}$.*

*So if Φ maps α^k to α^{k+1} (describes how to go from the strategy in round k to round $k+1$), then α^* is NE if and only if $\alpha^* = \Phi(\alpha^*)$ when it's the **fixed point** of this mapping Φ by the definition of NE. The only question is whether α^∞ is a fixed point, which is to ask whether*

$$\lim_{k\to\infty} \Phi(\alpha^k) = \Phi(\lim_{k\to\infty} \alpha^k) \tag{82}$$

is true (limit of α^k is taken in the space of square integrable processes $L^2([0,T] \times \Omega)$). This is the place where we need assumptions to guarantee the convergence of fictitious play.

Deep Fictitious Play (DFP)

Deep BSDE solver can be generalized and combined with FP to solve NE for multiple player games. Now assume we are in the Markovian case with uncontrolled diffusion coefficient as we have stated before, the HJBE is

$$v_t^i + \inf_{\alpha^i} \left\{ b(t,x,\alpha^i) \cdot \partial_x v^i + f^i(t,x,\alpha^i) \right\} + \frac{1}{2} Tr(\sigma\sigma^T H_{v^i}) = 0 \tag{83}$$

with $v^i(T,x) = g^i(x)$ and $\alpha^{i,*}$ is the argmin of the inf in HJBE. Since HJBE is semi-linear, we can apply the same decomposition and apply non-linear Feynman-Kac formula to get

$$\begin{cases} d\mathcal{X}_t = \mu(t,\mathcal{X}_t)\,dt + \sigma(t,\mathcal{X}_t)\,dB_t \\ dY_t = -H(t,\mathcal{X}_t,Z_t)\,dt + Z_t\,dB_t \\ Y_T = g(X_T) \end{cases} \tag{84}$$

where the correspondence is given by $Y_t^i = v^i(t,\mathcal{X}_t), Z_t^i = \sigma^T \partial_x v^i(t,\mathcal{X}_t)$. Now let's apply the generalized version of this Deep BSDE solver to give the Deep fictitous play algorithm.

Apply FP to get decoupled HJBE

$$v_t^{i,k+1} + \inf_{\alpha^i} \left\{ b^i(t,x,\alpha^i;\alpha^{-i,k}) \cdot \partial_x v^{i,k+1} + f^i(t,x,\alpha^i;\alpha^{-i,k}) \right\} + \frac{1}{2} Tr(\sigma\sigma^T H_{v^{i,k+1}}) = 0 \tag{85}$$

with $v^{i,k+1}(T,x) = g^i(x)$ and we can also apply non-linear Feynman-Kac formula to get BSDE systems and DFP solves this BSDE system using Deep BSDE solver.

The convergence of FP is ensured by that $\exists q \in (0,1)$ such that

$$\sup_{t\in[0,T]} \mathbb{E}||Y_t^k - Y_t||^2 + \int_0^T \mathbb{E}||Z_t^k - Z_t||^2\,dt + \int_0^T \mathbb{E}||\alpha_t^k - \alpha_t^*||^2\,dt \le Cq^k \int_0^T \mathbb{E}||\alpha_t^0 - \alpha_t^*||^2\,dt \tag{86}$$

the RHS upper bound depends on how far the initial guess is from the true NE. When $k \to \infty$, RHS goes to 0 and this ensures good property of the mapping Φ. Here Y_t, Z_t, α^* are the processes and controls in the true NE.

The convergence of DFP is that let $Y^{\pi,k}$ be the numerical solution of Y_t^k using Deep BSDE solver, define $\pi(t) = t_i$ if $t \in [t_i, t_{i+1})$ for a partition $\pi : 0 = t_0 \le ... \le t_N = T$, then

$$\sup_{t\in[0,T]} \mathbb{E}||Y_t^k - Y_{\pi(t)}^{\pi,k}||^2 + \int_0^T \mathbb{E}||Z_t^k - Z_{\pi(t)}^{\pi,k}||^2\,dt + \int_0^T \mathbb{E}||\alpha_{\pi(t)}^{\pi,k} - \alpha_t^*||^2\,dt \tag{87}$$

$$\le C\left[||\pi|| + q^k \int_0^T \mathbb{E}||\alpha_{\pi(t)}^{\pi,0} - \alpha_t^*||^2\,dt + \sum_{j=1}^{k} q^{kj}\mathbb{E}||g(X_T^\pi) - Y_T^{\pi,j}||^2 \right] \tag{88}$$

and the last term on RHS is given by the Deep BSDE solver since we guess the initial value of BSDE and match the terminal condition as the loss. We can guarantee that the loss of Deep BSDE solver is small enough at each stage $j = 1,2,...,k$ to ensure convergence of DFP subject to universal approximation (the bound is actually combining those of FP and Deep BSDE solver). Actually for any $\varepsilon > 0$, we can always choose an appropriate k such that $\alpha^{\pi,k}$ is ε-NE.