# Notes on MATH 260 AA

Haosheng Zhou

Sept, 2022

# Contents

Introduction to Neural Network . . . 2

Different NN Structures . . . 2

Training . . . 3

Problems & Improvements . . . 4

Paper Summary: Universal Approximation Results . . . 5

Settings and Definitions . . . 5

Sketch of Proof for Feedforward NN . . . 6

Paper Summary: The Expressive Power of NNs: A View from the Width . . . 9

Other Results . . . 9

Deep Learning and Stochastic Control, Direct Approach . . . 10

Overview . . . 10

Direct Parametrization . . . 10

Method of *Han-E* . . . 10

NNContPI . . . 12

Hybrid-Now . . . 13

Delayed Control Problems . . . 13

Deep Learning and Stochastic Control, PDE Approach . . . 14

# 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 $64 \times 64$ 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 $\nabla C$ by computing $\nabla C_x$ 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 $\nabla C$ 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 $\sigma$ is not chosen carefully, $\sigma'$ 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 $\beta$ 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 $\rho$, **$\rho$-dense** is the same as that defined in topology, i.e. $S \subset X$ is $\rho$-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 $\mu$ on $(\mathbb{R}^r, \mathscr{B}^r)$ (Borel measurable space), the metric $\rho_\mu$ 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 $\mu$, 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 $\rho_K$.*

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 $\rho_K$-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 $\rho_\mu$-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 $\rho_\mu$-dense for any probability measure $\mu$ (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 $\rho_\mu$ 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 $\rho_\mu$-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 $\lambda$ 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 $\rho_\mu$-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 $\rho_\mu$-dense in $C^r$ and $C^r$ is $\rho_\mu$-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*** *$\mu$. In other words, an NN with simple structure can approximate a measurable function arbitrarily well (under the metric of convergence in measure $\mu$) 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 $\rho_\mu$ 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 $\varepsilon$.*

## 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, $W, \theta$ 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 $\theta_t$ 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 $\theta_{t_n}$ 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 $\theta_{t_n}$. To be clear with how the method works, we fix all parameters $\theta_{t_n}$ (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 $\theta_{t_n}$ 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 $\theta$ 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 $\sum ReLU(-X_{t_n})$ to the loss as penalty term. More generally, if we require $C(X_t, \alpha_t) = 0$, then add $\sum_n ||C(X_{t_n}, \alpha_{NN}(t_n, X_{t_n}; \theta))||^2$ to loss, if we require $C(X_t, \alpha_t) \geq 0$, then add $\sum_n ReLU(-C(X_{t_n}, \alpha_{NN}(t_n, X_{t_n}; \theta)))$ 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 $\alpha$ 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)Why?. This approach is **global in time** in that the optimal controls at all the time points are trained simultaneously.

## 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 $\hat{\theta}_{n+1}, ..., \hat{\theta}_{N_T-1}$ ($\theta_n$ 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. $\theta$. Note that when $\theta$ 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 $(t_{n+1}, T)$. So only works for Markovian case?*

*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 $\theta$ 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 $\alpha_{t_n}$ 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 $N_T$)
- 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)

## 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.

**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. What about NNContPI and hybrid-now? They use DPP right?*

## 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 $\delta$ is a fixed positive number (but possibly unknown). The dynamics allows the time for reaction formed as time delay $\delta$ 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 $\delta$ a priori when applying LSTM**.

## Deep Learning and Stochastic Control, PDE Approach

Deep Galerkin method: Use $NN(t,x;\theta)$ to approximate value function $u$ and all derivatives are computed by auto-differentiation