Deep Hedging

Josef Teichmann

ETH Zürich

New York, May 2020

Introduction

Introduction

Goal of this talk is ...

- to present an abstract version of deep hedging and relate it to several problems in quantitative finance like pricing, hedging, or calibration.
- to relate this view to generative adversarial models.
- to present a result on representation of path space functionals with relations to simulations.

(joint works with Erdinc Akyildirim, Hans Bühler, Christa Cuchiero, Lukas Gonon, Lyudmila Grigoryeva, Jakob Heiss, Calypso Herrera, Wahid Khosrawi-Sardroudi, Jonathan Kochems, Martin Larsson, Thomas Krabichler, Florian Krach, Baranidharan Mohan, Juan-Pablo Ortega, Philipp Schmocker, Ben Wood, and Hanna Wutte)

Introduction

Goal of this talk is ...

- to present an abstract version of deep hedging and relate it to several problems in quantitative finance like pricing, hedging, or calibration.
- to relate this view to generative adversarial models.
- to present a result on representation of path space functionals with relations to simulations.

(joint works with Erdinc Akyildirim, Hans Bühler, Christa Cuchiero, Lukas Gonon, Lyudmila Grigoryeva, Jakob Heiss, Calypso Herrera, Wahid Khosrawi-Sardroudi, Jonathan Kochems, Martin Larsson, Thomas Krabichler, Florian Krach, Baranidharan Mohan, Juan-Pablo Ortega, Philipp Schmocker, Ben Wood, and Hanna Wutte)

Introduction

Goal of this talk is ...

- to present an abstract version of deep hedging and relate it to several problems in quantitative finance like pricing, hedging, or calibration.
- to relate this view to generative adversarial models.
- to present a result on representation of path space functionals with relations to simulations.

(joint works with Erdinc Akyildirim, Hans Bühler, Christa Cuchiero, Lukas Gonon, Lyudmila Grigoryeva, Jakob Heiss, Calypso Herrera, Wahid Khosrawi-Sardroudi, Jonathan Kochems, Martin Larsson, Thomas Krabichler, Florian Krach, Baranidharan Mohan, Juan-Pablo Ortega, Philipp Schmocker, Ben Wood, and Hanna Wutte)

... how it started

- Deep Hedging (learn trading strategies): joint projects with Hans Bühler, Lukas Gonon, Jonathan Kochems, Baranidharan MohanMartin and Ben Wood at JP Morgan (2017, 2019 in *arXiv* and *SSRN*).
- Deep Calibration (learn model parameters for local stochastic volatility models): joint project with Christa Cuchiero and Wahid Khosrawi-Sardroudi (2020 in *arXiv*).

... how it started

- Deep Hedging (learn trading strategies): joint projects with Hans Bühler, Lukas Gonon, Jonathan Kochems, Baranidharan MohanMartin and Ben Wood at JP Morgan (2017, 2019 in *arXiv* and *SSRN*).
- Deep Calibration (learn model parameters for local stochastic volatility models): joint project with Christa Cuchiero and Wahid Khosrawi-Sardroudi (2020 in *arXiv*).

Abstract generator

Consider a d-dimensional semi-martingale Y and (functional) stochastic differential equation

$$dX^\gamma(t) = \sum_{i=1}^{d} V_i^\gamma(X^\gamma, Y)_{t-}\, dY^i(t)\,,$$

where the vector fields $V_i^\gamma : \mathbb{D}^{N+n+d} \to \mathbb{D}^n$ map (càdlàg) paths (γ, X, Y) to paths in a functionally Lipschitz way. We consider X as state variables and γ as model parameters. t corresponds to time.

Abstract discriminator

Let $L^\delta : \mathrm{Def}(L) \subset L^0(\Omega) \to \mathbb{R}$ be a loss function depending on parameters δ. We are aiming for small values of $L^\delta(X^\gamma)$ for a fixed discriminating parameter δ, and for large values of $L^\delta(X^\gamma)$ for a fixed generating parameter process γ.

Symbolically we are trying to solve a game of inf-sup type: generate, by choosing γ, such that the loss L^δ is small, and discriminate, by choosing δ, when a generator X^γ is not good enough.

Models

- The processes X^γ are referred to as (generative) models, which generate certain structures.
- The loss function L^δ measures how well the generation of structure works.
- The process of choosing γ is called 'training'.
- In contrast to classical modeling the number of free parameters in models is very high (Occam's razor is not at all used!) and the loss function is adapted, again with a possibly high amount of free parameters, during the training process.
- Based on ideas of deep hedging we shall sometimes refer to this training problem as 'abstract hedging' since we hedge the possibly varying loss by choosing the strategy γ appropriately.

Models

- The processes X^γ are referred to as (generative) models, which generate certain structures.
- The loss function L^δ measures how well the generation of structure works.
- The process of choosing γ is called 'training'.
- In contrast to classical modeling the number of free parameters in models is very high (Occam's razor is not at all used!) and the loss function is adapted, again with a possibly high amount of free parameters, during the training process.
- Based on ideas of deep hedging we shall sometimes refer to this training problem as 'abstract hedging' since we hedge the possibly varying loss by choosing the strategy γ appropriately.

Models

- The processes X^γ are referred to as (generative) models, which generate certain structures.
- The loss function L^δ measures how well the generation of structure works.
- The process of choosing γ is called 'training'.
- In contrast to classical modeling the number of free parameters in models is very high (Occam's razor is not at all used!) and the loss function is adapted, again with a possibly high amount of free parameters, during the training process.
- Based on ideas of deep hedging we shall sometimes refer to this training problem as 'abstract hedging' since we hedge the possibly varying loss by choosing the strategy γ appropriately.

Models

- The processes X^γ are referred to as (generative) models, which generate certain structures.
- The loss function L^δ measures how well the generation of structure works.
- The process of choosing γ is called 'training'.
- In contrast to classical modeling the number of free parameters in models is very high (Occam's razor is not at all used!) and the loss function is adapted, again with a possibly high amount of free parameters, during the training process.
- Based on ideas of deep hedging we shall sometimes refer to this training problem as 'abstract hedging' since we hedge the possibly varying loss by choosing the strategy γ appropriately.

Models

- The processes X^γ are referred to as (generative) models, which generate certain structures.
- The loss function L^δ measures how well the generation of structure works.
- The process of choosing γ is called 'training'.
- In contrast to classical modeling the number of free parameters in models is very high (Occam's razor is not at all used!) and the loss function is adapted, again with a possibly high amount of free parameters, during the training process.
- Based on ideas of deep hedging we shall sometimes refer to this training problem as 'abstract hedging' since we hedge the possibly varying loss by choosing the strategy γ appropriately.

Neural vector fields

We shall always consider vector fields V^γ which are built from neural networks, i.e. linear combinations of compositions of simple functions and of non-linear functions of a simple one dimensional type. Neural networks satisfy remarkable properties.

Theorem

Let $(f_i)_{i \in I}$ be a sequence of real valued continuous functions on a compact space K (the 'simple' functions). We assume that the sequence is point separating and additively closed. Let $\varphi : \mathbb{R} \to \mathbb{R}$ be a sigmoid function (the simple 'non-linear function'), then

$$\left\langle x \mapsto \varphi(f_i(x) + c) \,\middle|\, i \in I,\, c \in \mathbb{R} \right\rangle$$

is dense in $C(K)$.

Models with vector fields of neural network type are called *neural models*.

Examples of abstract neural networks

- Classical shallow neural networks: $K = [0,1]^d$, f runs through all linear functions.
- Deep networks of depth k: $K = [0,1]^d$, f runs through all networks of depth $k-1$.
- Let X^* the dual of a Banach space and K its unit ball in the weak-$*$-topology: f runs through all evaluations at elements $x \in X$.
- Let X be a Banach space and K a compact subset: f runs through all continuous linear functionals.

Neural networks forget the natural grading of polynomial-type bases on space K.

Examples of abstract neural networks

- Classical shallow neural networks: $K = [0,1]^d$, f runs through all linear functions.
- Deep networks of depth k: $K = [0,1]^d$, f runs through all networks of depth $k-1$.
- Let X^* the dual of a Banach space and K its unit ball in the weak-$*$-topology: f runs through all evaluations at elements $x \in X$.
- Let X be a Banach space and K a compact subset: f runs through all continuous linear functionals.

Neural networks forget the natural grading of polynomial-type bases on space K.

Examples of abstract neural networks

- Classical shallow neural networks: $K = [0,1]^d$, f runs through all linear functions.
- Deep networks of depth k: $K = [0,1]^d$, f runs through all networks of depth $k-1$.
- Let X^* the dual of a Banach space and K its unit ball in the weak-$*$-topology: f runs through all evaluations at elements $x \in X$.
- Let X be a Banach space and K a compact subset: f runs through all continuous linear functionals.

Neural networks forget the natural grading of polynomial-type bases on space K.

Examples of abstract neural networks

- Classical shallow neural networks: $K = [0,1]^d$, f runs through all linear functions.
- Deep networks of depth k: $K = [0,1]^d$, f runs through all networks of depth $k-1$.
- Let X^* the dual of a Banach space and K its unit ball in the weak-$*$-topology: f runs through all evaluations at elements $x \in X$.
- Let X be a Banach space and K a compact subset: f runs through all continuous linear functionals.

Neural networks forget the natural grading of polynomial-type bases on space K.

Neural models

- Many algorithms in machine learning may be considered as training of neural models.
- Training is feasible when the dependence on state variables is sufficiently regular, for instance linear in the extreme case.
- Generalization of trained networks is successful when implicit or explicit regularizations appear.
- This means that state variables should contain as many features as possible, in particular redundant information might be helpful.

Neural models

- Many algorithms in machine learning may be considered as training of neural models.
- Training is feasible when the dependence on state variables is sufficiently regular, for instance linear in the extreme case.
- Generalization of trained networks is successful when implicit or explicit regularizations appear.
- This means that state variables should contain as many features as possible, in particular redundant information might be helpful.

Neural models

- Many algorithms in machine learning may be considered as training of neural models.
- Training is feasible when the dependence on state variables is sufficiently regular, for instance linear in the extreme case.
- Generalization of trained networks is successful when implicit or explicit regularizations appear.
- This means that state variables should contain as many features as possible, in particular redundant information might be helpful.

Neural models

- Many algorithms in machine learning may be considered as training of neural models.
- Training is feasible when the dependence on state variables is sufficiently regular, for instance linear in the extreme case.
- Generalization of trained networks is successful when implicit or explicit regularizations appear.
- This means that state variables should contain as many features as possible, in particular redundant information might be helpful.

Instances of the abstract GAN problem

Deep hedging

Let Y be an d-dimensional semi-martingale representing traded instruments. We assume an absence of arbitrage condition.

- Let $(\gamma, Y) \mapsto V^\gamma(Y)$ be a trading strategy depending on neural network parameters γ and on the price process Y in a functional way (deep hedge).
- X corresponds then to the profit and loss process of the trading strategy.
- Let F be an $\mathcal{F}_T$ measurable derivative and U a utility function.
- We choose the loss function L as squared difference of the expected utility of $X_T + \gamma_0 - F$ and the expected utility of the zero position ('indifference price of the seller of F').
- can be easily adapted for transaction costs, liquidity constraints, etc.
- adversarial training is not necessary.

Deep hedging

Let Y be an d-dimensional semi-martingale representing traded instruments. We assume an absence of arbitrage condition.

- Let $(\gamma, Y) \mapsto V^{\gamma}(Y)$ be a trading strategy depending on neural network parameters γ and on the price process Y in a functional way (deep hedge).
- X corresponds then to the profit and loss process of the trading strategy.
- Let F be an $\mathcal{F}_T$ measurable derivative and U a utility function.
- We choose the loss function L as squared difference of the expected utility of $X_T + \gamma_0 - F$ and the expected utility of the zero position ('indifference price of the seller of F').
- can be easily adapted for transaction costs, liquidity constraints, etc.
- adversarial training is not necessary.

Deep hedging

Let Y be an d-dimensional semi-martingale representing traded instruments. We assume an absence of arbitrage condition.

- Let $(\gamma, Y) \mapsto V^{\gamma}(Y)$ be a trading strategy depending on neural network parameters γ and on the price process Y in a functional way (deep hedge).
- X corresponds then to the profit and loss process of the trading strategy.
- Let F be an $\mathcal{F}_T$ measurable derivative and U a utility function.
- We choose the loss function L as squared difference of the expected utility of $X_T + \gamma_0 - F$ and the expected utility of the zero position ('indifference price of the seller of F').
- can be easily adapted for transaction costs, liquidity constraints, etc.
- adversarial training is not necessary.

Deep hedging

Let Y be an d-dimensional semi-martingale representing traded instruments. We assume an absence of arbitrage condition.

- Let $(\gamma, Y) \mapsto V^{\gamma}(Y)$ be a trading strategy depending on neural network parameters γ and on the price process Y in a functional way (deep hedge).
- X corresponds then to the profit and loss process of the trading strategy.
- Let F be an $\mathcal{F}_T$ measurable derivative and U a utility function.
- We choose the loss function L as squared difference of the expected utility of $X_T + \gamma_0 - F$ and the expected utility of the zero position ('indifference price of the seller of F').
- can be easily adapted for transaction costs, liquidity constraints, etc.
- adversarial training is not necessary.

Deep hedging

Let Y be an d-dimensional semi-martingale representing traded instruments. We assume an absence of arbitrage condition.

- Let $(\gamma, Y) \mapsto V^{\gamma}(Y)$ be a trading strategy depending on neural network parameters γ and on the price process Y in a functional way (deep hedge).
- X corresponds then to the profit and loss process of the trading strategy.
- Let F be an $\mathcal{F}_T$ measurable derivative and U a utility function.
- We choose the loss function L as squared difference of the expected utility of $X_T + \gamma_0 - F$ and the expected utility of the zero position ('indifference price of the seller of F').
- can be easily adapted for transaction costs, liquidity constraints, etc.
- adversarial training is not necessary.

Deep hedging

Let Y be an d-dimensional semi-martingale representing traded instruments. We assume an absence of arbitrage condition.

- Let $(\gamma, Y) \mapsto V^{\gamma}(Y)$ be a trading strategy depending on neural network parameters γ and on the price process Y in a functional way (deep hedge).
- X corresponds then to the profit and loss process of the trading strategy.
- Let F be an $\mathcal{F}_T$ measurable derivative and U a utility function.
- We choose the loss function L as squared difference of the expected utility of $X_T + \gamma_0 - F$ and the expected utility of the zero position ('indifference price of the seller of F').
- can be easily adapted for transaction costs, liquidity constraints, etc.
- adversarial training is not necessary.

Deep Calibration

Let W be a Brownian motion and α a stochastic volatility process: $dY_t = \alpha_t dW_t$:

- Let l^{γ_1} be a leverage function depending an neural network parameters γ_1:

$$dS_t = S_t \alpha_t l(\gamma_1(t), S_t) dW_t$$

  is a local stochastic volatility model with initial value S_0.

- Let C_j be finitely many derivatives with market price π_j, $j = 1, \dots, J$.
- Let h^{γ_2} be a trading strategy in the instrument S (for simplicity).
- Let the loss function L be the weighted sum of squared values of $E\big[C_j - \pi_j - (h \bullet S)_T\big]$ over J plus the $\sum_j E\big[(C_j - \pi_j - (h \bullet S)_T)^2\big]$ ('calibration of LSV model to finitely many market prices with variance reduction'). The weights will depend on discriminatory parameters δ.

Deep Calibration

Let W be a Brownian motion and α a stochastic volatility process: $dY_t = \alpha_t dW_t$:

- Let l^{γ_1} be a leverage function depending an neural network parameters γ_1:
$$dS_t = S_t \alpha_t l(\gamma_1(t), S_t) dW_t$$
is a local stochastic volatility model with initial value S_0.
- Let C_j be finitely many derivatives with market price π_j, $j = 1, \ldots, J$.
- Let h^{γ_2} be a trading strategy in the instrument S (for simplicity).
- Let the loss function L be the weighted sum of squared values of $E\left[C_j - \pi_j - (h \bullet S)_T\right]$ over J plus the $\sum_j E\left[(C_j - \pi_j - (h \bullet S)_T)^2\right]$ ('calibration of LSV model to finitely many market prices with variance reduction'). The weights will depend on discriminatory parameters δ.

Deep Calibration

Let W be a Brownian motion and α a stochastic volatility process: $dY_t = \alpha_t dW_t$:

- Let l^{γ_1} be a leverage function depending an neural network parameters γ_1:
$$dS_t = S_t \alpha_t l(\gamma_1(t), S_t) dW_t$$
is a local stochastic volatility model with initial value S_0.
- Let C_j be finitely many derivatives with market price π_j, $j = 1, \ldots, J$.
- Let h^{γ_2} be a trading strategy in the instrument S (for simplicity).
- Let the loss function L be the weighted sum of squared values of $E\left[C_j - \pi_j - (h \bullet S)_T\right]$ over J plus the $\sum_j E\left[(C_j - \pi_j - (h \bullet S)_T)^2\right]$ ('calibration of LSV model to finitely many market prices with variance reduction'). The weights will depend on discriminatory parameters δ.

Deep Calibration

Let W be a Brownian motion and α a stochastic volatility process: $dY_t = \alpha_t dW_t$:

- Let l^{γ_1} be a leverage function depending an neural network parameters γ_1:
$$dS_t = S_t \alpha_t l(\gamma_1(t), S_t) dW_t$$
is a local stochastic volatility model with initial value S_0.
- Let C_j be finitely many derivatives with market price π_j, $j = 1, \ldots, J$.
- Let h^{γ_2} be a trading strategy in the instrument S (for simplicity).
- Let the loss function L be the weighted sum of squared values of $E\big[C_j - \pi_j - (h \bullet S)_T\big]$ over J plus the $\sum_j E\big[(C_j - \pi_j - (h \bullet S)_T)^2\big]$ ('calibration of LSV model to finitely many market prices with variance reduction'). The weights will depend on discriminatory parameters δ.

Path functionals and Reservoir computing

Problem

In all previous instances it is desirable to have a flexible representation of adapted maps on path space:

- For (deep) hedging of path dependent options or in case of market frictions: hedging ratios will be path dependent.
- For (deep) calibration beyond plain vanilla prices: leverage functions will be path-dependent.

In the sequel we shall encounter a method to represent functionals on path space.

Problem

In all previous instances it is desirable to have a flexible representation of adapted maps on path space:

- For (deep) hedging of path dependent options or in case of market frictions: hedging ratios will be path dependent.
- For (deep) calibration beyond plain vanilla prices: leverage functions will be path-dependent.

In the sequel we shall encounter a method to represent functionals on path space.

Controlled ordinary differential equations (CODE)

The goal of this section is to develop methodology to *learn* efficiently represent functionals on path space $C^1([0,T],\mathbb{R}^d)$ (for simplicity). We consider differential equations of the form

$$dY_t = \sum_i V_i(Y_t) du^i_t \, , \; Y_0 = y \in E$$

to define evolutions in state space E depending on local characteristics, initial value $y \in E$ and the control u. We call this a controlled ordinary differential equation (CODE). CODE can be used as a model to explain expressiveness of deep neural networks, see joint work with Christa Cuchiero and Martin Larsson (2019 in *arXiv*).

Generic expansions for CODEs

Consider a controlled differential equation

$$dY_t = \sum_{i=1}^{d} V_i(Y_t) du_t^i \,, \; Y_0 = y \in E$$

for some smooth vector fields $V_i : E \to TE$, $i = 1, \ldots, d$ and d once continuously differentiable curves u^i, or finite variation continuous controls, or a rough path. This describes a controlled dynamics on E.

The goal is to understand $u \mapsto Y$ and to use this structure for representing general path space functionals.

We introduce some notation for this purpose:

Definition

Let $V : E \to E$ be a smooth vector field, and let $f : E \to \mathbb{R}$ be a smooth function, then we call

$$Vf(x) = df(x) \bullet V(x)$$

the transport operator associated to V, which maps smooth functions to smooth functions and determines V uniquely.

Theorem

Let Evol *be a smooth evolution operator on a convenient vector space* E *which satisfies (again the time derivative is taken with respect to the forward variable* t*) a controlled ordinary differential equation*

$$d\,\mathrm{Evol}_{s,t}(x) = \sum_{i=1}^{d} V_i(\mathrm{Evol}_{s,t}(x))du^i(t)$$

then for any smooth function $f : E \to \mathbb{R}$*, and every* $x \in E$

$$f\big(\mathrm{Evol}_{s,t}(x)\big) =$$
$$= \sum_{k=0}^{M} \sum_{i_1,\dots,u_k=1}^{d} V_{i_1} \cdots V_{i_k} f(x) \int_{s \le t_1 \le \cdots \le t_k \le t} du^{i_1}(t_1) \cdots du^{i_k}(t_k) +$$
$$+ R_M(s,t,f)$$

with remainder term

$$R_M(s,t,f) =$$
$$= \sum_{i_0,\dots,u_M=1}^{d} \int_{s\le t_1\le\cdots\le t_{M+1}\le t} V_{i_0}\cdots V_{i_k} f\big(\mathrm{Evol}_{s,t_0}(x)\big)\,du^{i_0}(t_0)\cdots du^{i_k}(t_M)$$

holds true for all times $s \le t$ and every natural number $M \ge 0$.

A lot of work has been done to understand the analysis, algebra and geometry of this expansion (Eckhard Platen, Kua-Tsai Chen, Gerard Ben-Arous, Terry Lyons). It is a starting point of *rough path analysis* (Terry Lyons, Peter Friz, etc) as well as of high-order numerical schemes (Kloeden-Platen).

An algebraic frame

Definition

Consider the free algebra $\mathbb{A}_d$ of formal series generated by d non-commutative indeterminates $e_1, \ldots, e_d$. A typical element $a \in \mathbb{A}_d$ is written as

$$a = \sum_{k=0}^{\infty} \sum_{i_1,\ldots,i_k=1}^{d} a_{i_1 \ldots i_k} e_{i_1} \cdots e_{i_k} \, ,$$

sums and products are defined in the natural way. We consider the complete locally convex topology making all projections $a \mapsto a_{i_1 \ldots i_k}$ continuous on $\mathbb{A}_d$, hence a convenient vector space.

Definition

We define on $\mathbb{A}_d$ smooth vector fields

$$a \mapsto ae_i$$

for $i = 1, \dots, d$.

Theorem

Let u be a smooth control, then the controlled differential equation

$$d\,\mathrm{Sig}_{s,t}(a) = \sum_{i=1}^{d} \mathrm{Sig}_{s,t}(a) e_i du^i(t)\,,\ \mathrm{Sig}_{s,s}(a) = a \qquad (1)$$

has a unique smooth evolution operator, called signature of u and denoted by Sig*, given by*

$$\mathrm{Sig}_{s,t}(a) = a \sum_{k=0}^{\infty} \sum_{i_1,\ldots,u_k=1}^{d} \int_{s\le t_1\le\cdots\le t_k\le t} du^{i_1}(t_1)\cdots du^{i_k}(t_k)\, e_{i_1}\cdots e_{i_k}\,. \qquad (2)$$

Theorem (Signature is a reservoir)

Let Evol *be a smooth evolution operator on a convenient vector space E which satisfies (again the time derivative is taken with respect to the forward variable t) a controlled ordinary differential equation*

$$d\,\mathrm{Evol}_{s,t}(x) = \sum_{i=1}^{d} V_i(\mathrm{Evol}_{s,t}(x))du^i(t)\,.$$

Then for any smooth (test) function $f : E \to \mathbb{R}$ and for every $M \geq 0$ there is a time-homogenous linear $W = W(V_1, \ldots, V_d, f, M, x)$ from $\mathbb{A}_d^M$ to the real numbers $\mathbb{R}$ such that

$$f\big(\mathrm{Evol}_{s,t}(x)\big) = W\big(\pi_M(\mathrm{Sig}_{s,t}(1))\big) + \mathcal{O}\big((t-s)^{M+1}\big)$$

for $s \leq t$.

Algebraic properties

- $\mathbb{A}_d$ is a Hopf Algebra and signature is group-like, whence polynomials of iterated integrals can be expressed as sums of iterated integrals.
- As a consequence the linear span of iterated integrals (where we add $u^0(t) = t$ as zeroth component) form a point separating algebra of functions on path space $C^1([0,T], \mathbb{R}^d)$. Whence continuous, non-linear functionals on compact subsets of path space can be approximated by *linear* combinations of signature.
- Adapted non-linear functionals can also be expressed in this way.

Algebraic properties

- $\mathbb{A}_d$ is a Hopf Algebra and signature is group-like, whence polynomials of iterated integrals can be expressed as sums of iterated integrals.
- As a consequence the linear span of iterated integrals (where we add $u^0(t) = t$ as zeroth component) form a point separating algebra of functions on path space $C^1([0, T], \mathbb{R}^d)$. Whence continuous, non-linear functionals on compact subsets of path space can be approximated by *linear* combinations of signature.
- Adapted non-linear functionals can also be expressed in this way.

Algebraic properties

- $\mathbb{A}_d$ is a Hopf Algebra and signature is group-like, whence polynomials of iterated integrals can be expressed as sums of iterated integrals.
- As a consequence the linear span of iterated integrals (where we add $u^0(t) = t$ as zeroth component) form a point separating algebra of functions on path space $C^1([0, T], \mathbb{R}^d)$. Whence continuous, non-linear functionals on compact subsets of path space can be approximated by *linear* combinations of signature.
- Adapted non-linear functionals can also be expressed in this way.

Signature as reservoir

- This explains that any solution can be represented – up to a linear readout – by universal reservoir, namely signature.
- This is used in many instances of provable machine learning by, e.g., groups in Oxford (Harald Oberhauser, Terry Lyons, etc), and also ...
- ... at JP Morgan, in particular great recent work on 'Nonparametric pricing and hedging of exotic derivatives' by Terry Lyons, Sina Nejad and Imanol Perez Arribas.
- in contrast to reservoir computing: signature is high dimensional (i.e. infinite dimensional) and a precisely defined, non-random object.
- Can we approximate signature by a lower dimensional random object with similar properties?

Signature as reservoir

- This explains that any solution can be represented – up to a linear readout – by universal reservoir, namely signature.
- This is used in many instances of provable machine learning by, e.g., groups in Oxford (Harald Oberhauser, Terry Lyons, etc), and also ...
- ... at JP Morgan, in particular great recent work on 'Nonparametric pricing and hedging of exotic derivatives' by Terry Lyons, Sina Nejad and Imanol Perez Arribas.
- in contrast to reservoir computing: signature is high dimensional (i.e. infinite dimensional) and a precisely defined, non-random object.
- Can we approximate signature by a lower dimensional random object with similar properties?

Signature as reservoir

- This explains that any solution can be represented – up to a linear readout – by universal reservoir, namely signature.
- This is used in many instances of provable machine learning by, e.g., groups in Oxford (Harald Oberhauser, Terry Lyons, etc), and also ...
- ... at JP Morgan, in particular great recent work on 'Nonparametric pricing and hedging of exotic derivatives' by Terry Lyons, Sina Nejad and Imanol Perez Arribas.
- in contrast to reservoir computing: signature is high dimensional (i.e. infinite dimensional) and a precisely defined, non-random object.
- Can we approximate signature by a lower dimensional random object with similar properties?

Signature as reservoir

- This explains that any solution can be represented – up to a linear readout – by universal reservoir, namely signature.
- This is used in many instances of provable machine learning by, e.g., groups in Oxford (Harald Oberhauser, Terry Lyons, etc), and also ...
- ... at JP Morgan, in particular great recent work on 'Nonparametric pricing and hedging of exotic derivatives' by Terry Lyons, Sina Nejad and Imanol Perez Arribas.
- in contrast to reservoir computing: signature is high dimensional (i.e. infinite dimensional) and a precisely defined, non-random object.
- Can we approximate signature by a lower dimensional random object with similar properties?

Signature as reservoir

- This explains that any solution can be represented – up to a linear readout – by universal reservoir, namely signature.
- This is used in many instances of provable machine learning by, e.g., groups in Oxford (Harald Oberhauser, Terry Lyons, etc), and also ...
- ... at JP Morgan, in particular great recent work on 'Nonparametric pricing and hedging of exotic derivatives' by Terry Lyons, Sina Nejad and Imanol Perez Arribas.
- in contrast to reservoir computing: signature is high dimensional (i.e. infinite dimensional) and a precisely defined, non-random object.
- Can we approximate signature by a lower dimensional random object with similar properties?

Random localized signature

A random localized signature

- choose a dimension M and random matrices with independent entries $A_1, \ldots, A_d$ on $\mathbb{R}^M$ as well as shifts $\beta_1, \ldots, \beta_d$, such that the following vector fields do not satisfy non-trivial relations.
- define
$$dX_t = \sum_{i=1}^{d} \sigma(A_i X_t + \beta_i) du^i(t)\,,\; X_0 = x\,.$$
for some smooth activation function σ.

Since the vector fields $x \mapsto \sigma(A_i x + b_i)$ are free as first order differential operators in the algebra of differential operators, then $f(X_.)$, for smooth functions f constitutes a regression basis equivalent to signature.

This is joint work with Christa Cuchiero, Lukas Gonon, Lyudmila Grigoryeva and Juan-Pablo Ortega. A more quantitative proof applies the Johnson-Lindenstrauss theorem.

Deep Simulation

Let $W^1, \dots, W^d$ be Brownian motions and V_i^θ neural network vector fields:

- Consider for fixed θ the autonomous stochastic differential equation

$$dX_t = \sum_{i=1}^{d} V_i^\theta(X_t) dW_t^i$$

  with initial value X_0.
- Assume that $(\hat{X}_t)_{0 \le t \le T}$ is a given observed trajectory for a Brownian motion trajectory $(W_t)_{0 \le t \le T}$.
- Let L be a possibly weighted distance of paths.

Conclusion and Outlook

State space extension

- whenever path dependencies appear it makes sense to include random localized signature (looking back for a certain period of time) as additional state variables to make path dependencies as linear as possible.
- random localized signature is of moderate dimension, so state spaces do not explode by this procedure.
- Reinforcement learning on such state spaces is still feasible and strategies are trainable.

State space extension

- whenever path dependencies appear it makes sense to include random localized signature (looking back for a certain period of time) as additional state variables to make path dependencies as linear as possible.
- random localized signature is of moderate dimension, so state spaces do not explode by this procedure.
- Reinforcement learning on such state spaces is still feasible and strategies are trainable.

State space extension

- whenever path dependencies appear it makes sense to include random localized signature (looking back for a certain period of time) as additional state variables to make path dependencies as linear as possible.
- random localized signature is of moderate dimension, so state spaces do not explode by this procedure.
- Reinforcement learning on such state spaces is still feasible and strategies are trainable.