Machine Learning in Finance

Josef Teichmann

ETH Zürich

Stockholm January 2019

Goal of this talk ...

- to present a standard perspective on machine learning.
- highlight one important instance of machine learning local stochastic volatility models (joint work with Christa Cuchiero and Wahid Khosrawi-Sardroudi)
- to present the paradigm of reservoir computing.
- to apply reservoir computing for scenario generation (based on joint work with Christa Cuchiero, Lukas Gonon, Lyudmila Grigoryeva and Juan-Pablo Ortega).

Goal of this talk ...

- to present a standard perspective on machine learning.
- highlight one important instance of machine learning local stochastic volatility models (joint work with Christa Cuchiero and Wahid Khosrawi-Sardroudi)
- to present the paradigm of reservoir computing.
- to apply reservoir computing for scenario generation (based on joint work with Christa Cuchiero, Lukas Gonon, Lyudmila Grigoryeva and Juan-Pablo Ortega).

Goal of this talk ...

- to present a standard perspective on machine learning.
- highlight one important instance of machine learning local stochastic volatility models (joint work with Christa Cuchiero and Wahid Khosrawi-Sardroudi)
- to present the paradigm of reservoir computing.
- to apply reservoir computing for scenario generation (based on joint work with Christa Cuchiero, Lukas Gonon, Lyudmila Grigoryeva and Juan-Pablo Ortega).

Neural Networks

Neural networks are nowadays frequently used to approximate functions due to ubiquitous universal approximation properties. A neural network, as for instance graphically represented in Figure 1,

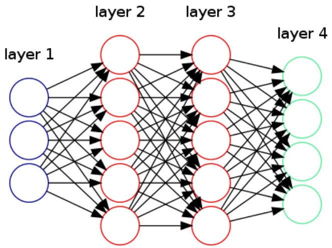


Figure: A 2 hidden layers neural network with 3 input and 4 output dimensions

just encodes a certain concatenation of affine and non-linear functions by composition in a well specified order. There are feed forward, recursive neural networks. Input as well as output dimension are fixed

Neural Networks and Universal Approximation

- Neural networks appear in the 1943 seminal work by Warren McCulloch and Walter Pitts inspired by certain functionalities of the human brain aiming for artificial intelligence (AI).
- Arnold-Kolmogorov Theorem represents functions on unit cube by sums and uni-variate functions (Hilbert's thirteenth problem), i.e.

$$F(x_1, \ldots, x_d) = \sum_{i=0}^{2d} \varphi_i \Big(\sum_{j=1}^{d} \psi_{ij}(x_j) \Big)$$

- Universal Approximation Theorems (George Cybenko, Kurt Hornik, et al.) show that *one hidden layer networks* can already *uniformly approximate* any continuous function on the unit cube.

Neural Networks and Universal Approximation

- Neural networks appear in the 1943 seminal work by Warren McCulloch and Walter Pitts inspired by certain functionalities of the human brain aiming for artificial intelligence (AI).
- Arnold-Kolmogorov Theorem represents functions on unit cube by sums and uni-variate functions (Hilbert's thirteenth problem), i.e.

$$F(x_1, \dots, x_d) = \sum_{i=0}^{2d} \varphi_i \Big(\sum_{j=1}^{d} \psi_{ij}(x_j) \Big)$$

- Universal Approximation Theorems (George Cybenko, Kurt Hornik, et al.) show that *one hidden layer networks* can already *uniformly approximate* any continuous function on the unit cube.

Neural Networks and Universal Approximation

- Neural networks appear in the 1943 seminal work by Warren McCulloch and Walter Pitts inspired by certain functionalities of the human brain aiming for artificial intelligence (AI).
- Arnold-Kolmogorov Theorem represents functions on unit cube by sums and uni-variate functions (Hilbert's thirteenth problem), i.e.

$$F(x_1, \dots, x_d) = \sum_{i=0}^{2d} \varphi_i \Big(\sum_{j=1}^{d} \psi_{ij}(x_j) \Big)$$

- Universal Approximation Theorems (George Cybenko, Kurt Hornik, et al.) show that *one hidden layer networks* can already *uniformly approximate* any continuous function on the unit cube.

Applications of neural networks

- *Learning* is the specification of a neural network which approximates a certain non-linear function on some input space.
- Modern learning technology performs training tasks in a highly accessible and very efficient way (Tensorflow, Theano, Torch).
- Deep neural networks (actually depth 4 is sufficient) easily approximate wavelet basis functions, whence the theory of all sorts of wavelet approximations of non-linear functions applies.
- "Unreasonable effectiveness" of learning (image and language recognition, classification tasks, etc).

Applications of neural networks

- *Learning* is the specification of a neural network which approximates a certain non-linear function on some input space.
- Modern learning technology performs training tasks in a highly accessible and very efficient way (Tensorflow, Theano, Torch).
- Deep neural networks (actually depth 4 is sufficient) easily approximate wavelet basis functions, whence the theory of all sorts of wavelet approximations of non-linear functions applies.
- "Unreasonable effectiveness" of learning (image and language recognition, classification tasks, etc).

Applications of neural networks

- *Learning* is the specification of a neural network which approximates a certain non-linear function on some input space.
- Modern learning technology performs training tasks in a highly accessible and very efficient way (Tensorflow, Theano, Torch).
- Deep neural networks (actually depth 4 is sufficient) easily approximate wavelet basis functions, whence the theory of all sorts of wavelet approximations of non-linear functions applies.
- "Unreasonable effectiveness" of learning (image and language recognition, classification tasks, etc).

Applications of neural networks

- *Learning* is the specification of a neural network which approximates a certain non-linear function on some input space.
- Modern learning technology performs training tasks in a highly accessible and very efficient way (Tensorflow, Theano, Torch).
- Deep neural networks (actually depth 4 is sufficient) easily approximate wavelet basis functions, whence the theory of all sorts of wavelet approximations of non-linear functions applies.
- "Unreasonable effectiveness" of learning (image and language recognition, classification tasks, etc).

Applications in Finance

- instead of using established numerical algorithms one *learns* solutions.
- Successfully implemented in several areas: deep hedging (Bühler-Gonon-Teichmann-Wood 2017), deep calibration (Cuchiero-Khousrawi-Teichmann et al. 2016-17-18),
- we see solutions of problems which we have never seen before and we are at the edge of fully realistic modelling: high dimensional risk management tasks with trading constraints and transaction costs can be treated, high dimensional calibration problems can be solved in real time. Networks with *tens of thousands of weights* are trained to perform this work.

Applications in Finance

- instead of using established numerical algorithms one *learns* solutions.
- Successfully implemented in several areas: deep hedging (Bühler-Gonon-Teichmann-Wood 2017), deep calibration (Cuchiero-Khousrawi-Teichmann et al. 2016-17-18),
- we see solutions of problems which we have never seen before and we are at the edge of fully realistic modelling: high dimensional risk management tasks with trading constraints and transaction costs can be treated, high dimensional calibration problems can be solved in real time. Networks with *tens of thousands of weights* are trained to perform this work.

Applications in Finance

- instead of using established numerical algorithms one *learns* solutions.
- Successfully implemented in several areas: deep hedging (Bühler-Gonon-Teichmann-Wood 2017), deep calibration (Cuchiero-Khousrawi-Teichmann et al. 2016-17-18),
- we see solutions of problems which we have never seen before and we are at the edge of fully realistic modelling: high dimensional risk management tasks with trading constraints and transaction costs can be treated, high dimensional calibration problems can be solved in real time. Networks with *tens of thousands of weights* are trained to perform this work.

Goals

Write a generic calibrator

- which does neither need model tractability (except Monte Carlo pricing) nor dimensional restrictions.
- which does allow for non European derivative data, dividends, incorporates market frictions.
- uses the primal method of pricing, e.g. super-replication.

Goals

Write a generic calibrator

- which does neither need model tractability (except Monte Carlo pricing) nor dimensional restrictions.
- which does allow for non European derivative data, dividends, incorporates market frictions.
- uses the primal method of pricing, e.g. super-replication.

Goals

Write a generic calibrator

- which does neither need model tractability (except Monte Carlo pricing) nor dimensional restrictions.
- which does allow for non European derivative data, dividends, incorporates market frictions.
- uses the primal method of pricing, e.g. super-replication.

Proof of concept

- We choose stochastic local volatility models in dimension 1 to prove feasibility.
- We are given finitely many European option data (i.e. no volatility surface) for different maturities and strikes.
- We follow the path of Saporito-Yang-Zubelli but we work in a stochastic setting and we calibrate directly to prices data.

Proof of concept

- We choose stochastic local volatility models in dimension 1 to prove feasibility.
- We are given finitely many European option data (i.e. no volatility surface) for different maturities and strikes.
- We follow the path of Saporito-Yang-Zubelli but we work in a stochastic setting and we calibrate directly to prices data.

Proof of concept

- We choose stochastic local volatility models in dimension 1 to prove feasibility.
- We are given finitely many European option data (i.e. no volatility surface) for different maturities and strikes.
- We follow the path of Saporito-Yang-Zubelli but we work in a stochastic setting and we calibrate directly to prices data.

Local stochastic volatility models

- In stochastic local volatility (SLV) models the price process $(S_t)_{t \geq 0}$ of one asset follows an SDE
$$dS_t = S_t L(t, S_t)\alpha_t dW_t,$$
where
  - α_t is a stochastic process,
  - $L(t, s)$ is the so called *leverage or local volatiliy function*,
  - W is the driving Brownian motion.
- The function L is the crucial part in this model. It should allow to perfectly calibrate the implied volatility surface seen in the market.
- Given an implied volatility the leverage function has to satisfy
$$L^2(t, s) = \frac{\sigma^2_{\text{Dupire}}(t, x)}{\mathbb{E}[\alpha_t^2 | S_t = s]}.$$
- This is an implicit equation for L since the leverage function is *needed* for the computation of $\mathbb{E}[\alpha_t^2 | S_t = s]$.

Existing methods

The implicit equation for L can be solved via McKean-Vlasov techniques. The existing calibration techniques can be roughly distinguished into:

- Monte Carlo techniques based on particle methdos for McKean-Vlasov equations:
  - P. Henry-Labordère (2009): "Calibration of Local Stochastic Volatility Models: A Monte-Carlo Approach"
  - J. Guyon and P. Henry-Labordère (2011): "The Smile Calibration Problem Solved"
- PDE techniques based on non-linear Fokker-Planck equations for McKean-Vlasov equations:
  - Y. Tian, Z. Zhu, G. Lee, F. Klebaner, and K. Hamza (2015): "Calibrating and Pricing with a Stochastic-Local Volatility Model".
- inverse problem techniques for PDEs:
  - Y Saporito, X. Yang, J. Zubelli (2017): "The Calibration of Stochastic-Local Volatility Models - An Inverse Problem Perspective"

Deep Calibration

- Choose the parameters of a usual stochastic volatility model.
- Define some grid $0 = t_0 < t_1 \cdots < t_n = T$. Parametrize the leverage function $L(t,s)$ via a family of neural networks, i.e.
$$L(t,s) = F^{i+1}(s) \quad t \in [t_i, t_{i+1}), \quad i \in \{0, \dots, n-1\},$$
where for every $i \in \{1, \dots, n\}$, $F^i \in \mathcal{NN}_{M,1,1}$.
- Parametrize hedging strategies δ (with respect to hedging instruments) by neural networks
- We denote the weights of all the L-networks by θ^L and of the hedging networks by θ^δ.
- Fix θ^L and learn to hedge by changing θ^δ, i.e. minimize a risk functional of **payoff minus market price minus hedge P&L**.
- Fix θ^δ and learn θ^L to price correctly, i.e. minimize the expectation of **payoff minus market price minus hedge**.

Data and Objectives

- Consider several call options on the underlying S parametrized by τ_j corresponding e.g. to different strikes and maturities.
- Fix θ^L and determine the weights θ^δ of all of the hedging networks

$$\operatorname{argmin}_{\theta^\delta} \sum_{j=1}^{J} w_j \mathbb{E}\left[u((S_{T_j} - K_j)_+ - (\delta(\theta^\delta, \tau_j) \bullet S)_{T_j} - \pi^{\mathrm{mkt}}(\tau_j))\right],$$

  where

  - S denote the prices of the underlying depending on θ^L,
  - π^{mkt} denote the market prices of the respective calls,
  - w_j are some product specific weights (e.g. vega weighting).
  - u is a function assessing risk of the position.
- the expectation operator is calculated along N^δ trajectories by MC.

Data and Objectives

- Fix next θ^δ and determine the weights θ^L of all of the L networks

$$\operatorname{argmin}_{\theta^L} \sum_{j=1}^{J} w_j \Big(\mathbb{E}\big[(S_{T_j} - K_j)_+ - (\delta(\theta^\delta, \tau_j) \bullet S)_{T_j} - \pi^{\text{mkt}}(\tau_j) \big] \Big)^2 ,$$

- The expectation operator is calculated along N^L trajectories, which can be chosen due to strong variance reducation, extraordinarily small.

Two important remarks

- standard BS hedges are in many cases excellent, hence learning θ^L is often not necessary and does not need to be very precise to do the variance reduction job.
- the variance reduction through hedging is tremendous, i.e. N^L only lies in the thousands.

Example: SABR stochastic local vol model

- In the SABR SLV model the price process $(S_t)_{t\geq 0}$ satisfies

$$\begin{aligned} dS_t &= S_t L(t, S_t)\alpha_t dW_t, \\ d\alpha_t &= \nu\alpha_t dB_t, \end{aligned}$$

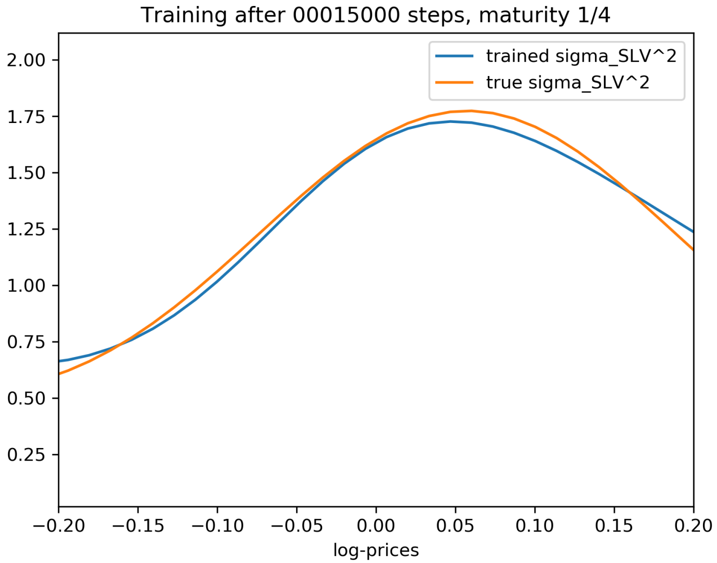
  where B and W are correlated Browian motions with $d\langle B_t, W_t\rangle_t = \rho dt$ and ν is the vol of vol.

- where ν, ρ, α_0 are some fixed parameters.
- For the "data" we use some 60 prices at maturities $T = 0.1, 0.25, 0.5, 1.0$ generated by some leverage function.

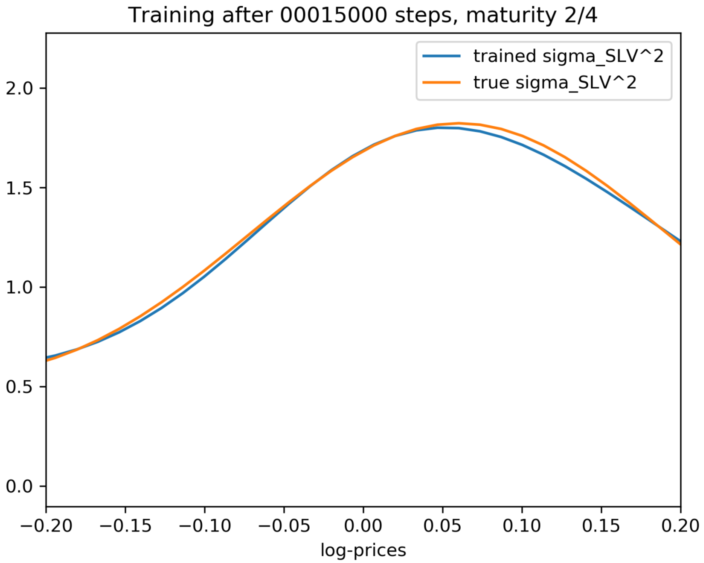
Implementation

- done in Tensorflow with ADAM optimizer.
- hedging strategies are initialized around the BS hedge.
- goal: re-construct the leverage function (stopping criterion: implied vol error less than 0.001)

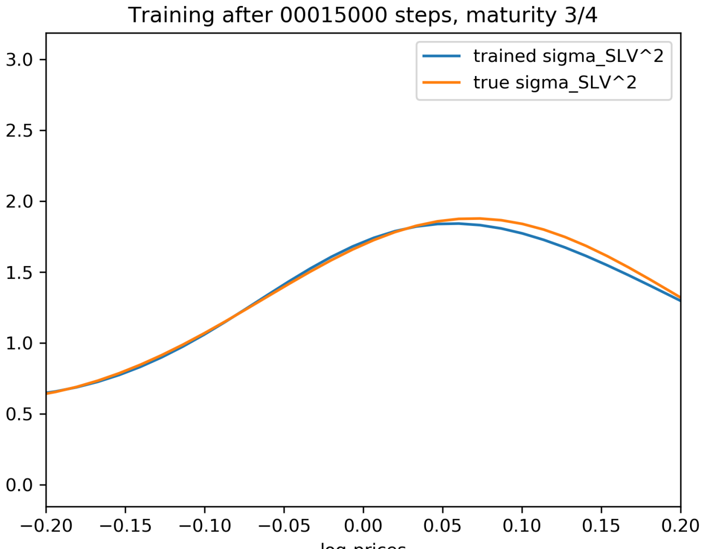
Learned Leverage functions versus true leverage function



Learned Leverage functions versus true leverage function



Learned Leverage functions versus true leverage function



Discrete-time market model with frictions

- Trading: at time points $t_0 = 0 < t_1 < \ldots < t_n = T$.
- Prices of hedging instruments: stochastic process $(S_{t_k})_{k=0,\ldots,n}$ in $\mathbb{R}^d$.
- Work on a (finite) probability space $(\Omega, \mathcal{G}, \mathbb{P})$ with filtration $\mathbb{G} = (\mathcal{G}_{t_k})_{k=0,\ldots,n}$, for simplicity $\mathcal{G}_{t_k} = \sigma(S_{t_0}, \ldots, S_{t_k})$.
- At $t = 0$ sell a contingent claim with (random) payoff Z at $T > 0$.
- Specify a (smaller) filtration $\mathbb{F} \subset \mathbb{G}$ for hedging.
- Charging price p_0 and hedging according to an $\mathbb{F}$-predictable strategy δ, terminal profit and loss is (with $\cdot$ discrete-time stochastic integration)

$$\mathrm{PL}_T(Z, p_0, \delta) := -Z + \underbrace{p_0}_{\text{price}} + \underbrace{(\delta \cdot S)_T}_{\text{trading gains}} - \underbrace{C_T(\delta)}_{\text{cum. transaction costs}} .$$

Setup and problem formulation in detail

$$\mathrm{PL}_T(Z, p_0, \delta) := -Z + \underbrace{p_0}_{\text{price}} + \underbrace{(\delta \cdot S)_T}_{\text{trading gains}} - \underbrace{C_T(\delta)}_{\text{cum. transaction costs}} . \qquad (1)$$

(1) in more detail:

- $(\delta \cdot S)_T = \sum_{k=1}^n \delta_{t_k} \cdot (S_{t_k} - S_{t_{k-1}})$.
- $C_T(\delta) = \sum_{k=0}^n c_k(\delta_{t_k} - \delta_{t_{k-1}}, S_{t_0}, \ldots, S_{t_k})$ with $\delta_{t_{-1}} := 0, \delta_{t_n} := 0$.
- Example: transaction costs proportional to transaction amount, i.e. $c_k(\delta_{t_k} - \delta_{t_{k-1}}, S_{t_0}, \ldots, S_{t_k}) = \sum_{i=1}^d \varepsilon_i |\delta^i_{t_k} - \delta^i_{t_{k-1}}| S^i_{t_k}$.
- Note: $\mathrm{PL}_T(Z, p_0, \delta) \geq 0$ represents a gain for seller.

Setup and problem formulation in detail

$$\mathrm{PL}_T(Z, p_0, \delta) := -Z + \underbrace{p_0}_{\text{price}} + \underbrace{(\delta \cdot S)_T}_{\text{trading gains}} - \underbrace{C_T(\delta)}_{\text{cum. transaction costs}} . \qquad (1)$$

(1) in more detail:

- $(\delta \cdot S)_T = \sum_{k=1}^{n} \delta_{t_k} \cdot (S_{t_k} - S_{t_{k-1}})$.
- $C_T(\delta) = \sum_{k=0}^{n} c_k(\delta_{t_k} - \delta_{t_{k-1}}, S_{t_0}, \ldots, S_{t_k})$ with $\delta_{t_{-1}} := 0, \delta_{t_n} := 0$.
- Example: transaction costs proportional to transaction amount, i.e. $c_k(\delta_{t_k} - \delta_{t_{k-1}}, S_{t_0}, \ldots, S_{t_k}) = \sum_{i=1}^{d} \varepsilon_i |\delta^i_{t_k} - \delta^i_{t_{k-1}}| S^i_{t_k}$.
- Note: $\mathrm{PL}_T(Z, p_0, \delta) \geq 0$ represents a gain for seller.

Indifference pricing and optimal hedging:

- Describe risk-preferences by a convex risk-measure ρ.
- Denote $\mathcal{H}$ the set of available hedging strategies.
- The indifference price is the (unique) solution $p(Z)$ to

$$\inf_{\delta \in \mathcal{H}} \rho\left(\mathrm{PL}_T(Z, p(Z), \delta)\right) = \inf_{\delta \in \mathcal{H}} \rho\left(\mathrm{PL}_T(0, 0, \delta)\right). \qquad (2)$$

- Optimal hedging strategy is minimizer δ^* (if it exists) in left-hand-side of (2).

Numerical calculation of $p(Z)$ and δ^*:

- *Highly challenging* by classical numerical techniques (very high-dimensional problem) in particular in the 2Filtration setting.
- $\rightarrow$ in practice more simplistic models are used (parametric, continuous-time, no transaction costs).

Indifference pricing and optimal hedging:

- Describe risk-preferences by a convex risk-measure ρ.
- Denote $\mathcal{H}$ the set of available hedging strategies.
- The indifference price is the (unique) solution $p(Z)$ to

$$\inf_{\delta \in \mathcal{H}} \rho\left(\mathrm{PL}_T(Z, p(Z), \delta)\right) = \inf_{\delta \in \mathcal{H}} \rho\left(\mathrm{PL}_T(0, 0, \delta)\right). \tag{2}$$

- Optimal hedging strategy is minimizer δ^* (if it exists) in left-hand-side of (2).

Numerical calculation of $p(Z)$ and δ^*:

- *Highly challenging* by classical numerical techniques (very high-dimensional problem) in particular in the 2Filtration setting.
- $\rightarrow$ in practice more simplistic models are used (parametric, continuous-time, no transaction costs).

- an approximate calculation *feasible* thanks to modern deep learning techniques.
- consider only hedging strategies $\delta = (\delta_{t_k})_{k=1,\dots,n}$ of the form

$$\delta_{t_k} = F^{\theta_k}(S_{t_{k-1}}, \delta_{t_{k-1}}), \quad k = 1, \dots, n$$

  where F^{θ_k} is a neural network with weights parametrized by θ_k.
- neural networks are surprisingly efficient at approximating multivariate functions.
- efficient machine learning optimization algorithms (stochastic gradient-type and backpropagation) and implementations (Tensorflow, Theano, Torch, ...) are available.

Approximate indifference price

- By cash-invariance, the indifference price $p(Z)$ is given as
$$\begin{aligned} p(Z) &= \pi(-Z) - \pi(0), \text{ where} \\ \pi(X) &:= \inf_{\delta \in \mathcal{H}} \rho(X + (\delta \cdot S)_T - C_T(\delta)) \, . \end{aligned}$$
- For suitable parameter set $\Theta_M \subset \mathbb{R}^r$ and $\delta^\theta_{t_k} = F^{\theta_k}(S_{t_{k-1}}, \delta^\theta_{t_{k-1}})$ as above, approximate $\pi(X)$ by
$$\pi_M(X) := \inf_{\theta \in \Theta_M} \rho\Big(X + (\delta^\theta \cdot S)_T - C_T(\delta^\theta)\Big) \, .$$
- Directly amenable to machine learning optimization algorithms (stochastic gradient-type and backpropagation) if ρ is entropic risk measure ($\rightarrow$ exponential utility indifference price) or more generally an *optimized certainty equivalent*, i.e.
$$\rho(X) := \inf_{w \in \mathbb{R}} \{w + E[\ell(-X - w)]\} \tag{3}$$
for $\ell \colon \mathbb{R} \to \mathbb{R}$ continuous, non-decreasing and convex.

Approximate indifference price

- By cash-invariance, the indifference price $p(Z)$ is given as

$$\begin{aligned} p(Z) &= \pi(-Z) - \pi(0), \text{ where} \\ \pi(X) &:= \inf_{\delta \in \mathcal{H}} \rho(X + (\delta \cdot S)_T - C_T(\delta)) . \end{aligned}$$

- For suitable parameter set $\Theta_M \subset \mathbb{R}^r$ and $\delta^\theta_{t_k} = F^{\theta_k}(S_{t_{k-1}}, \delta^\theta_{t_{k-1}})$ as above, approximate $\pi(X)$ by

$$\pi_M(X) := \inf_{\theta \in \Theta_M} \rho\Big(X + (\delta^\theta \cdot S)_T - C_T(\delta^\theta)\Big) .$$

- Directly amenable to machine learning optimization algorithms (stochastic gradient-type and backpropagation) if ρ is entropic risk measure ($\rightarrow$ exponential utility indifference price) or more generally an *optimized certainty equivalent*, i.e.

$$\rho(X) := \inf_{w \in \mathbb{R}} \{w + E[\ell(-X - w)]\} \tag{3}$$

for $\ell \colon \mathbb{R} \to \mathbb{R}$ continuous, non-decreasing and convex.

Results and Adapations

- with some right it is said that machine learning applications are nothing else than successful implementations of universal approximations. So far *Deep Hedging* is not much more.
- Deep Hedging becomes increasingly more interesting when *true* machine learning techniques come into play, for instance reservoir computing:
  - replace econometric model by a reservoir based model (deep simulation), then the whole model is directly data driven.
  - form reservoirs for hedging strategies by building a reservoir approach for BSDEs: build the analogue of signature maps and their Johnson-Lindenstrauss projection for hedging strategies.
  - considerably reduce the training effort by only learning last layers and make it *therefore* understandable (explainable AI).

Results and Adapations

- with some right it is said that machine learning applications are nothing else than successful implementations of universal approximations. So far *Deep Hedging* is not much more.
- Deep Hedging becomes increasingly more interesting when *true* machine learning techniques come into play, for instance reservoir computing:
  - replace econometric model by a reservoir based model (deep simulation), then the whole model is directly data driven.
  - form reservoirs for hedging strategies by building a reservoir approach for BSDEs: build the analogue of signature maps and their Johnson-Lindenstrauss projection for hedging strategies.
  - considerably reduce the training effort by only learning last layers and make it *therefore* understandable (explainable AI).

Results and Adapations

- with some right it is said that machine learning applications are nothing else than successful implementations of universal approximations. So far *Deep Hedging* is not much more.
- Deep Hedging becomes increasingly more interesting when *true* machine learning techniques come into play, for instance reservoir computing:
  - replace econometric model by a reservoir based model (deep simulation), then the whole model is directly data driven.
  - form reservoirs for hedging strategies by building a reservoir approach for BSDEs: build the analogue of signature maps and their Johnson-Lindenstrauss projection for hedging strategies.
  - considerably reduce the training effort by only learning last layers and make it *therefore* understandable (explainable AI).

Results and Adapations

- with some right it is said that machine learning applications are nothing else than successful implementations of universal approximations. So far *Deep Hedging* is not much more.
- Deep Hedging becomes increasingly more interesting when *true* machine learning techniques come into play, for instance reservoir computing:
  - replace econometric model by a reservoir based model (deep simulation), then the whole model is directly data driven.
  - form reservoirs for hedging strategies by building a reservoir approach for BSDEs: build the analogue of signature maps and their Johnson-Lindenstrauss projection for hedging strategies.
  - considerably reduce the training effort by only learning last layers and make it *therefore* understandable (explainable AI).

Results and Adapations

- with some right it is said that machine learning applications are nothing else than successful implementations of universal approximations. So far *Deep Hedging* is not much more.
- Deep Hedging becomes increasingly more interesting when *true* machine learning techniques come into play, for instance reservoir computing:
    - replace econometric model by a reservoir based model (deep simulation), then the whole model is directly data driven.
    - form reservoirs for hedging strategies by building a reservoir approach for BSDEs: build the analogue of signature maps and their Johnson-Lindenstrauss projection for hedging strategies.
    - considerably reduce the training effort by only learning last layers and make it *therefore* understandable (explainable AI).

Contempletation

- Implementation close to the development process is possible and – given some literacy in Tensorflow – not difficult.
- Such implementations are quite robust with respect to dimension: high dimensional markets as well as directly learning the map

  $$\text{market prices} \mapsto \text{model parameters}$$

  is now feasible. We see solutions of problems which we have never seen before.
- It remains mathematically open why it works so well, since we search in high dimensional spaces for optima by simple stochastic gradient descent.

Dynamic view on deep networks

Consider depth as time dimension and deep networks as Euler-discretized stochastic dynamical systems:

- **Input:** the noise
- **Output:** solution trajectory

Dynamic view on deep networks

Consider depth as time dimension and deep networks as Euler-discretized stochastic dynamical systems:

- **Input:** the noise
- **Output:** solution trajectory

Reservoir Computing

... We aim to learn an input-output map on a high- or infinite dimensional input state space.

Paradigm

Split the input-output map into a generic part (the *reservoir*), which is *not* trained and a readout part, which is trained.

A motivation from stochastic analysis

Consider a stochastic differential equation or any controlled system

$$dY_t = \sum_{i=1}^{d} V_i(Y_t) \circ dB_t^i\,,\ Y_0 = y \in \mathbb{R}^N$$

for some smooth vector fields $V_i : \mathbb{R}^N \to \mathbb{R}^N$, $i = 1, \dots, d$ and d independent Brownian motions B^i. This describes a stochastic dynamics on $\mathbb{R}^N$.

We want to learn the map

(input noise B) $\mapsto$ (solution Y).

Obviously a complicated, non-linear map, ...

Iterated integrals, signatures, enhanced models

Stochastic Taylor expansion (for rough input signals rough path theory, for more complicated noises the theory of regularity structures) gives a satisfying answer to this important question.

We can split the map $(B_t)_{0\le t\le T} \mapsto Y_T$ into two parts:

- A linear systems in the free algebra with d generators (the product is denoted by $\otimes$).

  $$dX_t = \sum_{i=1}^{d} X_t \otimes e_i \circ dB^i_t\,,\ X_0 = 1$$

  This is an infinite dimensional system (called, e.g., the signature process) whose solution is just the collection of all iterated Ito-Stratonovich integrals and which does *not* depend on the specific dynamics (the reservoir).
- A linear map $Y \mapsto WY$ which is trained (linear regression!) which is chosen to explain X_T as good as possible (the readout).

Iterated integrals, signatures, enhanced models

Stochastic Taylor expansion (for rough input signals rough path theory, for more complicated noises the theory of regularity structures) gives a satisfying answer to this important question.

We can split the map $(B_t)_{0\leq t\leq T} \mapsto Y_T$ into two parts:

- A linear systems in the free algebra with d generators (the product is denoted by $\otimes$).

  $$dX_t = \sum_{i=1}^{d} X_t \otimes e_i \circ dB^i_t\,,\ X_0 = 1$$

  This is an infinite dimensional system (called, e.g., the signature process) whose solution is just the collection of all iterated Ito-Stratonovich integrals and which does *not* depend on the specific dynamics (the reservoir).

- A linear map $Y \mapsto WY$ which is trained (linear regression!) which is chosen to explain X_T as good as possible (the readout).

A random localized signature

Instead of the previously (fixed) infinite dimensional system there might be better choices to construct a reservoir: fix an activation function σ.

A random localized signature

- there is a set of hyper-parameters $\theta \in \Theta$, and a dimension M.
- depending on θ choose randomly matrices $A_1, \dots, A_d$ on $\mathbb{R}^M$ as well as shifts $\beta_1, \dots, \beta_d$ such that maximal non-integrability holds on a starting point $x \in \mathbb{R}^M$.
- one can tune the hyper-parameters $\theta \in \Theta$ and dimension M such that

$$dX_t = \sum_{i=1}^{d} \sigma(A_i X_t + \beta_i) \circ dB_t^i \,, \; X_0 = x$$

locally (in time) approximates Y_t via a linear readout up to arbitrary precision.

Proof:

Apply the Johnson-Lindenstrauss Lemma to project the signature dynamics to a low dimensional space onto an almost random dynamics which has almost the geometry of signature and hence serves the same purpose.

For every n-point subset Q of a Hilbert space H and for every $0.5 > \epsilon > 0$ there exists a projection $f : H \to \mathbb{R}^k$ with

$$k = \frac{20 \log(n)}{\epsilon^2}$$

such that $(1-\epsilon)\|u-v\| \leq \|f(u)-f(v)\| \leq (1+\epsilon)\|u-v\|$ for all $u, v \in Q$ holds true. The map can be chosen randomly with high probability with respect to known laws!

Applications of Reservoir computing

- Often reservoirs can be realized physically, whence ultrafast evaluations are possible. Only the readout map W has to be trained.
- One can learn dynamic phenomena *without* knowing the specific vector fields V_i just from its results, even only along *one trajectory*.
- It works unreasonably well with generalization tasks, in particular state space constraints are often unreasonably well respected.

Applications of Reservoir computing

- Often reservoirs can be realized physically, whence ultrafast evaluations are possible. Only the readout map W has to be trained.
- One can learn dynamic phenomena *without* knowing the specific vector fields V_i just from its results, even only along *one trajectory*.
- It works unreasonably well with generalization tasks, in particular state space constraints are often unreasonably well respected.

Applications of Reservoir computing

- Often reservoirs can be realized physically, whence ultrafast evaluations are possible. Only the readout map W has to be trained.
- One can learn dynamic phenomena *without* knowing the specific vector fields V_i just from its results, even only along *one trajectory*.
- It works unreasonably well with generalization tasks, in particular state space constraints are often unreasonably well respected.

A view towards regularity structures

Rough paths and signatures can be seen from the point of view of regularity structures. One can formulate the problem of reservoir computing also from this perspective.

- Understand the structure of local expansions.
- generate them by dynamical systems.

A view towards regularity structures

Rough paths and signatures can be seen from the point of view of regularity structures. One can formulate the problem of reservoir computing also from this perspective.

- Understand the structure of local expansions.
- generate them by dynamical systems.

A view towards regularity structures

Rough paths and signatures can be seen from the point of view of regularity structures. One can formulate the problem of reservoir computing also from this perspective.

- Understand the structure of local expansions.
- generate them by dynamical systems.

Market scenario generation

- The market provides us with one trajectory of N prices Y of underlyings, derivatives, etc over some period of time $0 \dots T$.
- Construct a normalized noisy signal of dimension d, for instance take the most liquidly traded underlyings $S^1, \dots, S^K$ and estimate their instantanous covariance Σ along the trajectory, then normalize infinitesimally by its root, i.e. define
$$B_t := \int_0^t \sqrt{\Sigma_s^{-1}} dS_s \, .$$
By statistical tests one should check whether B has the properties of a Brownian path.
- Choose a reservoir (by tuning some hyperparameters) and learn the readout map W such that along B the output path Y is optimally explained.
- Generalize into the future by prolonging B as a Brownian motion.

Market scenario generation

- The market provides us with one trajectory of N prices Y of underlyings, derivatives, etc over some period of time $0 \dots T$.
- Construct a normalized noisy signal of dimension d, for instance take the most liquidly traded underlyings $S^1, \dots, S^K$ and estimate their instantanous covariance Σ along the trajectory, then normalize infinitesimally by its root, i.e. define

$$B_t := \int_0^t \sqrt{\Sigma_s^{-1}} dS_s \, .$$

  By statistical tests one should check whether B has the properties of a Brownian path.
- Choose a reservoir (by tuning some hyperparameters) and learn the readout map W such that along B the output path Y is optimally explained.
- Generalize into the future by prolonging B as a Brownian motion.

Market scenario generation

- The market provides us with one trajectory of N prices Y of underlyings, derivatives, etc over some period of time $0 \dots T$.
- Construct a normalized noisy signal of dimension d, for instance take the most liquidly traded underlyings $S^1, \dots, S^K$ and estimate their instantanous covariance Σ along the trajectory, then normalize infinitesimally by its root, i.e. define
$$B_t := \int_0^t \sqrt{\Sigma_s^{-1}} dS_s \,.$$
By statistical tests one should check whether B has the properties of a Brownian path.
- Choose a reservoir (by tuning some hyperparameters) and learn the readout map W such that along B the output path Y is optimally explained.
- Generalize into the future by prolonging B as a Brownian motion.

Market scenario generation

- The market provides us with one trajectory of N prices Y of underlyings, derivatives, etc over some period of time $0 \dots T$.
- Construct a normalized noisy signal of dimension d, for instance take the most liquidly traded underlyings $S^1, \dots, S^K$ and estimate their instantanous covariance Σ along the trajectory, then normalize infinitesimally by its root, i.e. define
$$B_t := \int_0^t \sqrt{\Sigma_s^{-1}} dS_s \, .$$
By statistical tests one should check whether B has the properties of a Brownian path.
- Choose a reservoir (by tuning some hyperparameters) and learn the readout map W such that along B the output path Y is optimally explained.
- Generalize into the future by prolonging B as a Brownian motion.

Remarks

- Notice that at no point of this procedure a parametric model (like, e.g., GARCH or Heston) was specified.
- Notice that we are actually dealing with a high dimensional situation with several constraints.
- Notice that the approach is completely data driven, however, we can always – through the choice of the reservoir – include additional features like extreme events into the picture.

Remarks

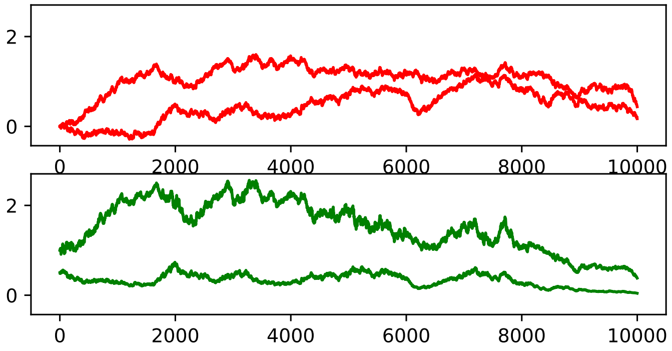
- Notice that at no point of this procedure a parametric model (like, e.g., GARCH or Heston) was specified.
- Notice that we are actually dealing with a high dimensional situation with several constraints.
- Notice that the approach is completely data driven, however, we can always – through the choice of the reservoir – include additional features like extreme events into the picture.

Remarks

- Notice that at no point of this procedure a parametric model (like, e.g., GARCH or Heston) was specified.
- Notice that we are actually dealing with a high dimensional situation with several constraints.
- Notice that the approach is completely data driven, however, we can always – through the choice of the reservoir – include additional features like extreme events into the picture.

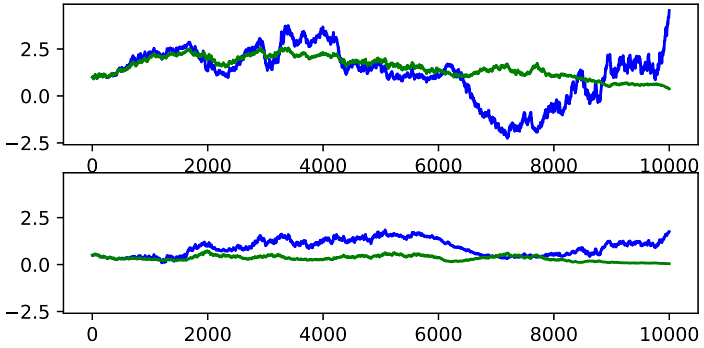
Learning a SABR market from one trajectory

We generate for one 2d Brownian trajectory (red) a SABR market trajectory (green), i.e. a price process together with a stochastic variance process for 10000 ticks.



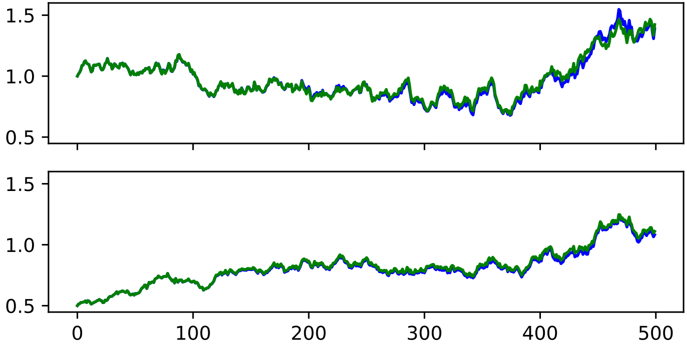
Learning a SABR market from one trajectory

We generate a 50 dimensional reservoir trajectory along the above Brownian path with random vector fields and perform a regression for 1000 ticks. The result is shown in blue. Already the visible generalization beyong 1000 ticks is amazing.



Generalization

We generalize for another Brownian trajectory and check whether the regression (blue) on the reservoir trajectory represents the SABR trajectory (green) accurately along 500 ticks.



Outlook

- test market generators in different environments.
- explore different explanations for the observed phenomena.
- explore the phenomena within the theory of regularity structures.

Outlook

- test market generators in different environments.
- explore different explanations for the observed phenomena.
- explore the phenomena within the theory of regularity structures.

Outlook

- test market generators in different environments.
- explore different explanations for the observed phenomena.
- explore the phenomena within the theory of regularity structures.

References

- H. Bühler, L. Gonon, J. Teichmann, and B. Wood: *Deep Hedging*, Arxiv, 2018.
- M. Hairer: *Theory of regularity structures, Inventiones mathematicae*, 2014.
- T. Lyons, *Rough paths, Signatures and the modelling of functions on streams Terry Lyons*, Arxiv, 2014.