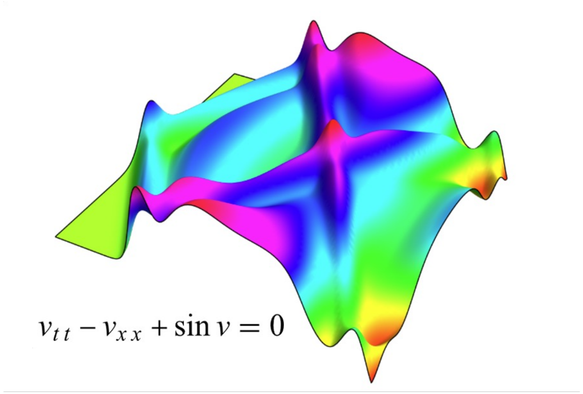
Solving high-dimensional partial differential equations using deep learning

Jiequn Han, Arnulf Jentzen and Weinan E

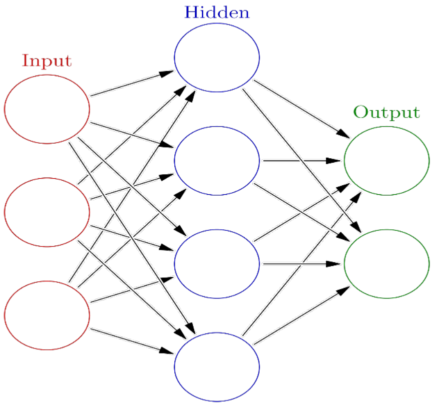
Presented by: Marcel Maciejczyk, Franz Schwinn

Intro 👋

PDE



Deep Learning



https://people.math.harvard.edu/~knill/teaching/math22b2022/handouts/lecture13.pdf
https://en.wikipedia.org/wiki/Neural_network_%28machine_learning%29

Positioning in literature 📍

Brother context:

font size indicates **amount of referenced papers by topic**

https://wordclouds.ethz.ch/

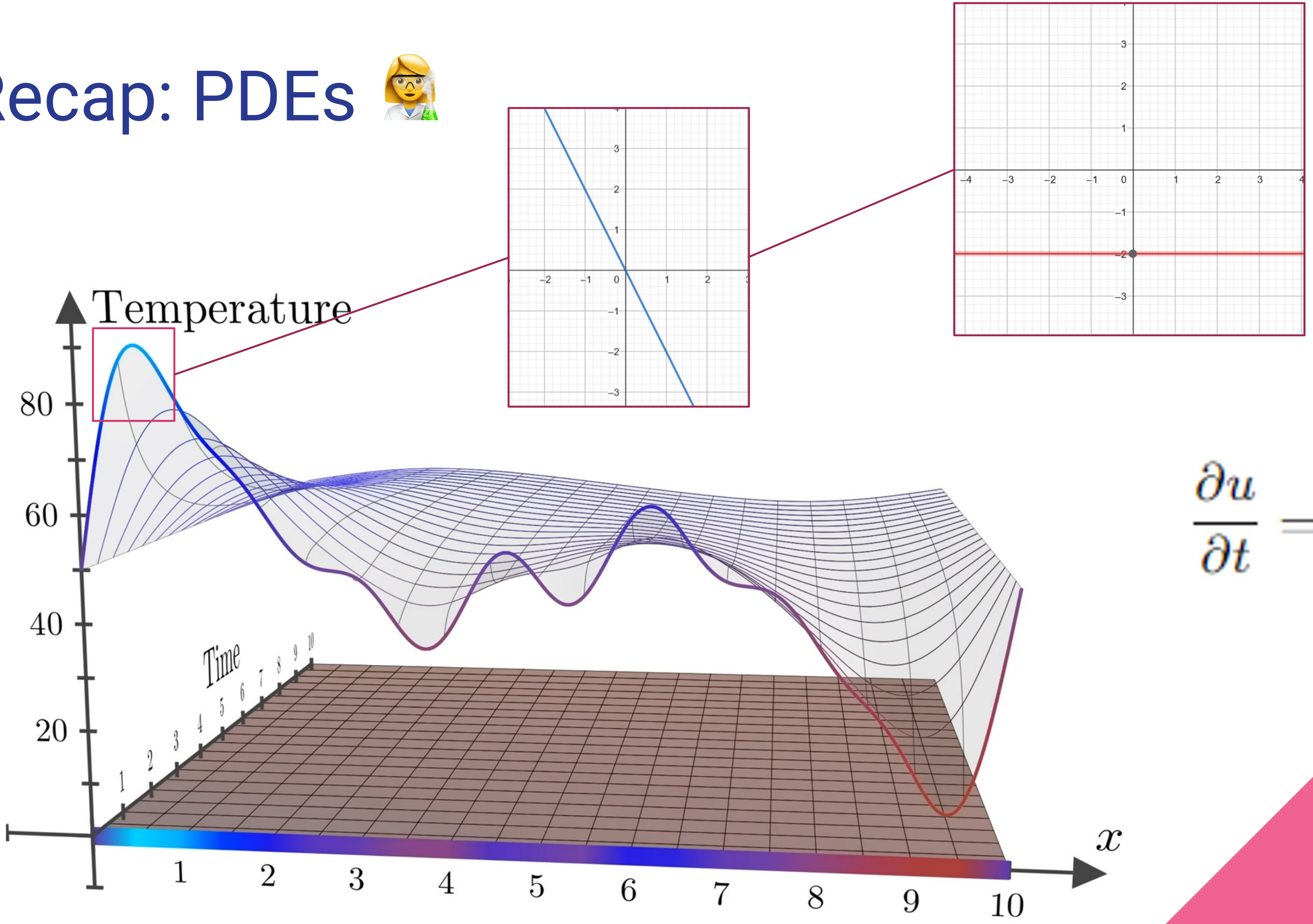
Recap: PDEs 👩‍🔬

- Mathematical **equations** that involve **functions of multiple variables** and their **partial derivatives**
- Example: 1-d heat function:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

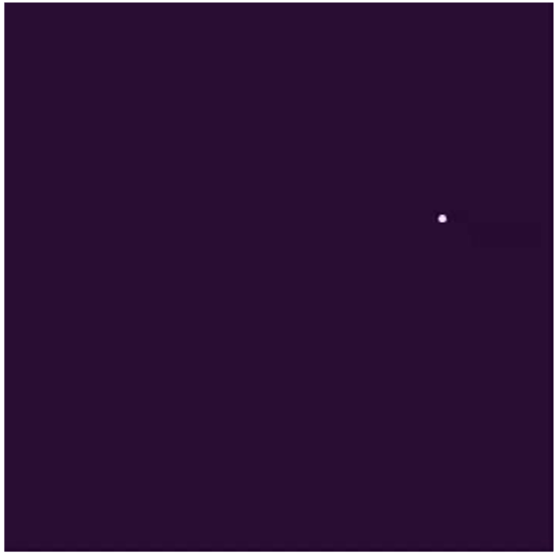
"How does temperature change over time?"

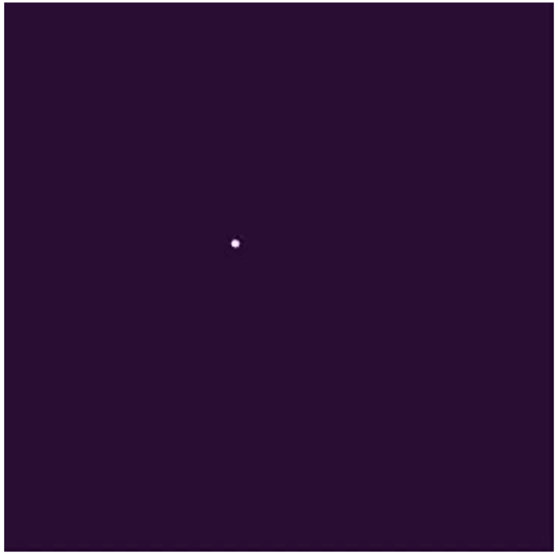
Recap: PDEs 👩‍🔬



$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

Recap: PDEs 👩‍🔬





https://visualpde.com/sim/?preset=heatEquation
https://visualpde.com/sim/?preset=waveEquation

Recap: PDEs

Linear	Semilinear	Parabolic
Operator that defines the pde is linear	L(u)+N(u)=0	Capture diffusion
Feynman-Kac formula (Monte Carlo method)		

Recap: BSDEs 👩‍🔬

BSDE

Backward Stochastic Differential Equations

random influence

SDEs are a type of differential equation that include **stochastic** terms

→ Finance, Physics, Biology, Engineering, Control theory…

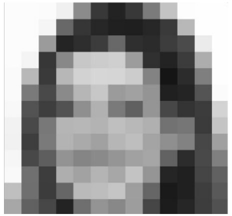
BSDE: backward in time

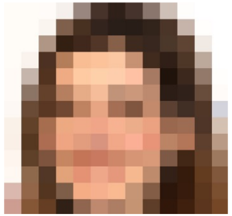
❗ Idea of the paper: Reformulate PDEs to BSDE and solve them using neural networks

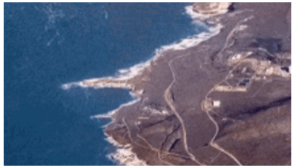
Curse of dimensionality problem 😰

- Solving PDEs is hard, as we often operate in **high-dimensional space**

$$\begin{pmatrix} \text{age} \\ \text{gender} \\ \text{blood type} \end{pmatrix}$$







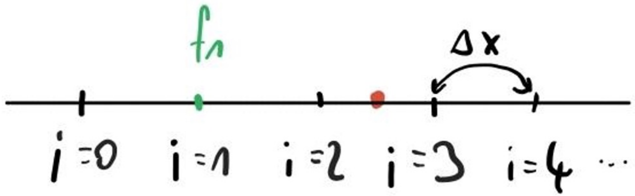
...

Dim:	3	169	507	24’883’200	89’579’520’000

Curse of dimensionality problem 😰

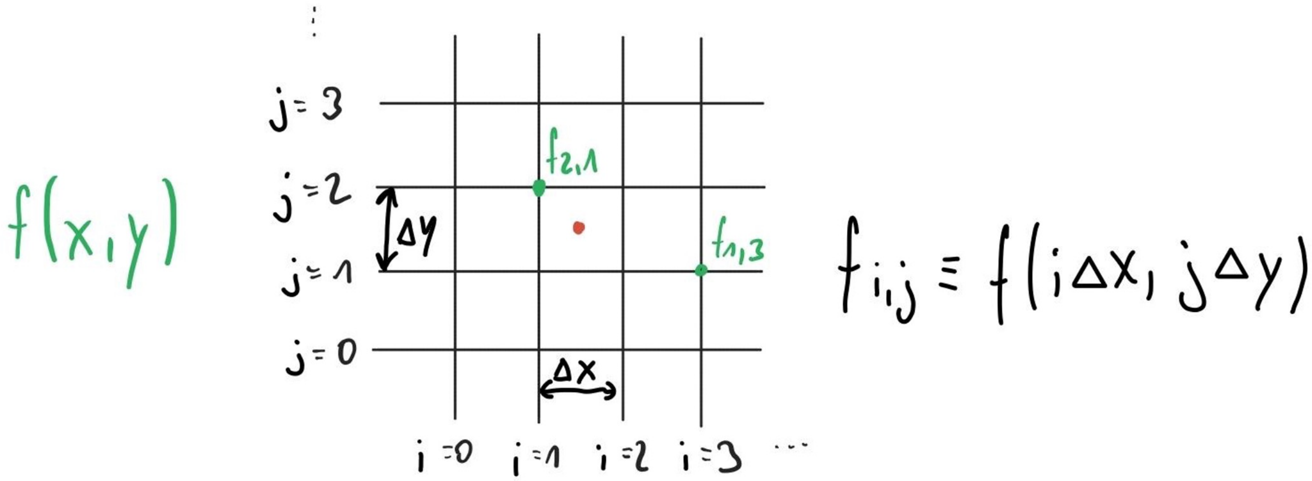
- PDEs depend on many variables
- Problem? → **Exponential increase** in computational resources!
- Finite difference method

$f(x)$



$f_i \equiv f(i\Delta x)$

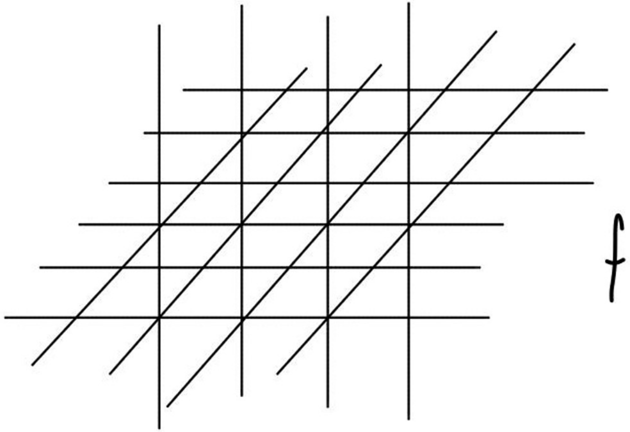
Curse of dimensionality problem 😰



Curse of dimensionality problem

Richard Bellman, 1961

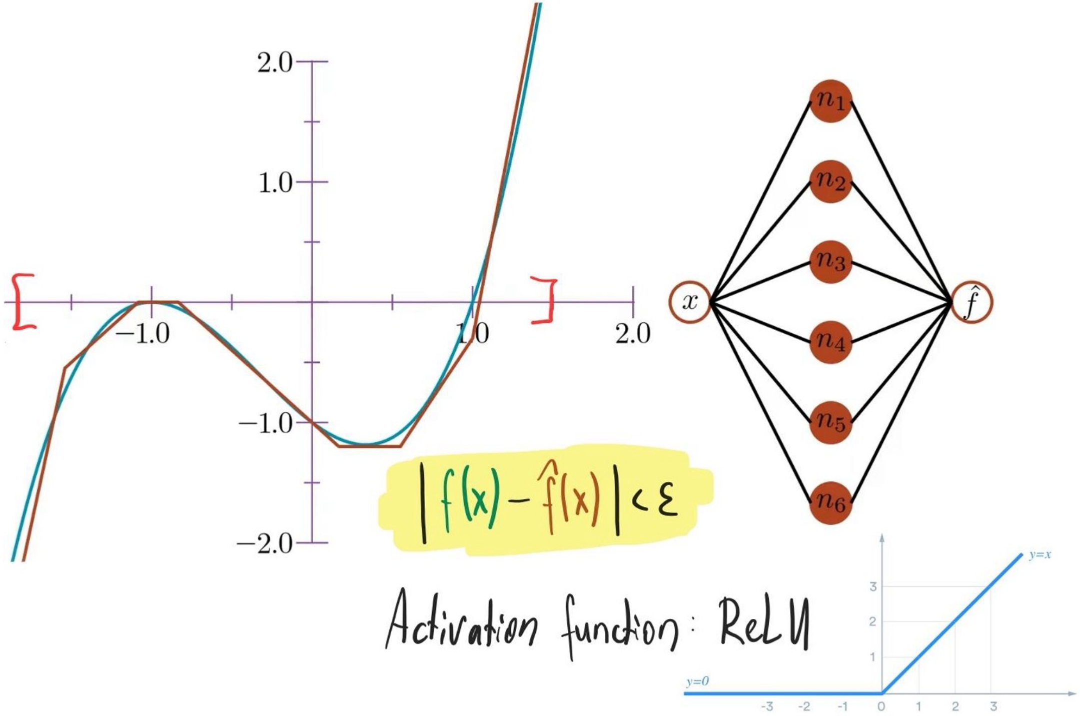
$f(x, y, z, \dots)$



$f_{i,j} \equiv f(i\Delta x, j\Delta y, k\Delta z, \dots)$

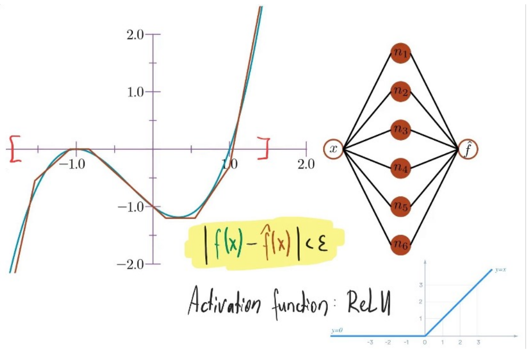
Universal approximation theorem

→ why can we even use neural nets?



“Neural network with a single hidden layer containing a finite number of neurons can approximate any continuous function on compact subsets of R^n”

Universal approximation theorem

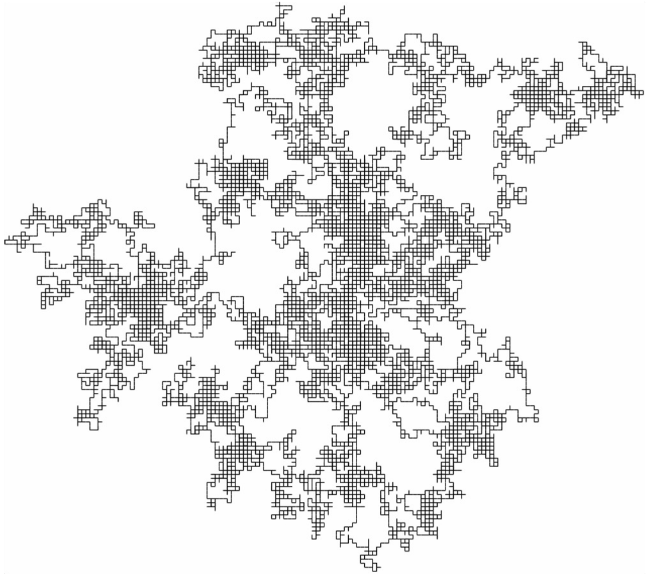


- Involved functions in parabolic PDEs are typically continuous

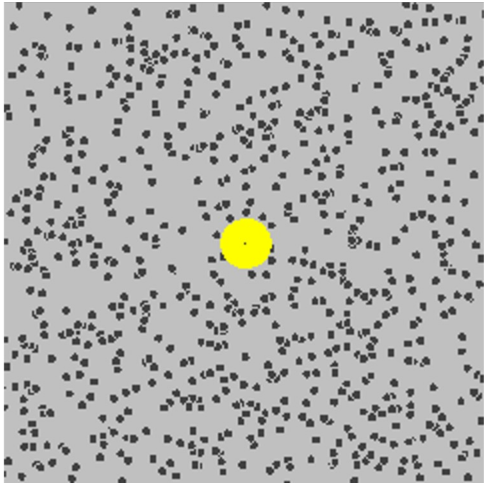
- The UAT provides **theoretical justification** for using neural networks to approximate solutions to PDEs

Brownian motion

Random walk



Brownian motion

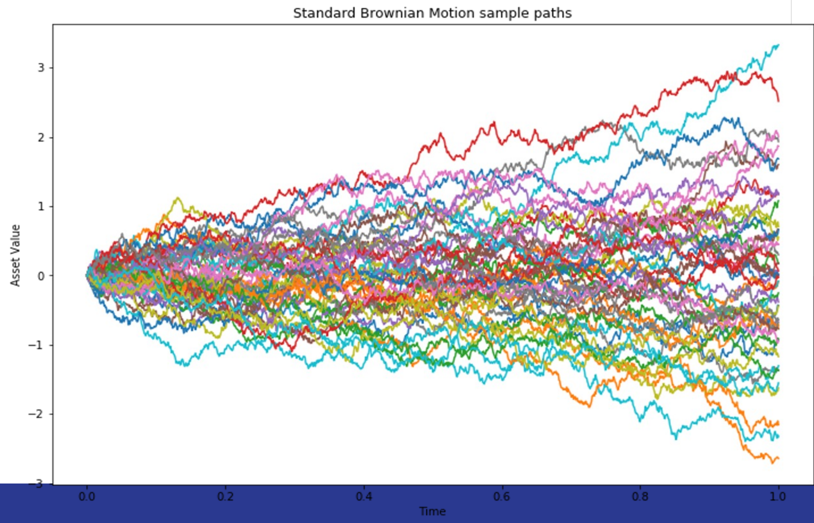


https://commons.wikimedia.org/wiki/File:Brownian_motion_large.gif.
https://commons.wikimedia.org/wiki/File:Random_walk_25000.gif.

Brownian motion

Brownian motion (also called Wiener process) has the following properties:

1. $W_0 = 0$ almost surely
2. W has indepdent increments: $W_{t+u} - W_t$ are independent of the past values $W_s,\ s < t$
3. W has Gaussian increments: $W_{t+u} - W_t \sim N(0, u)$
4. W has almost surely continuous paths



https://www.quantstart.com/articles/brownian-motion-simulation-with-python/

Objective of the paper

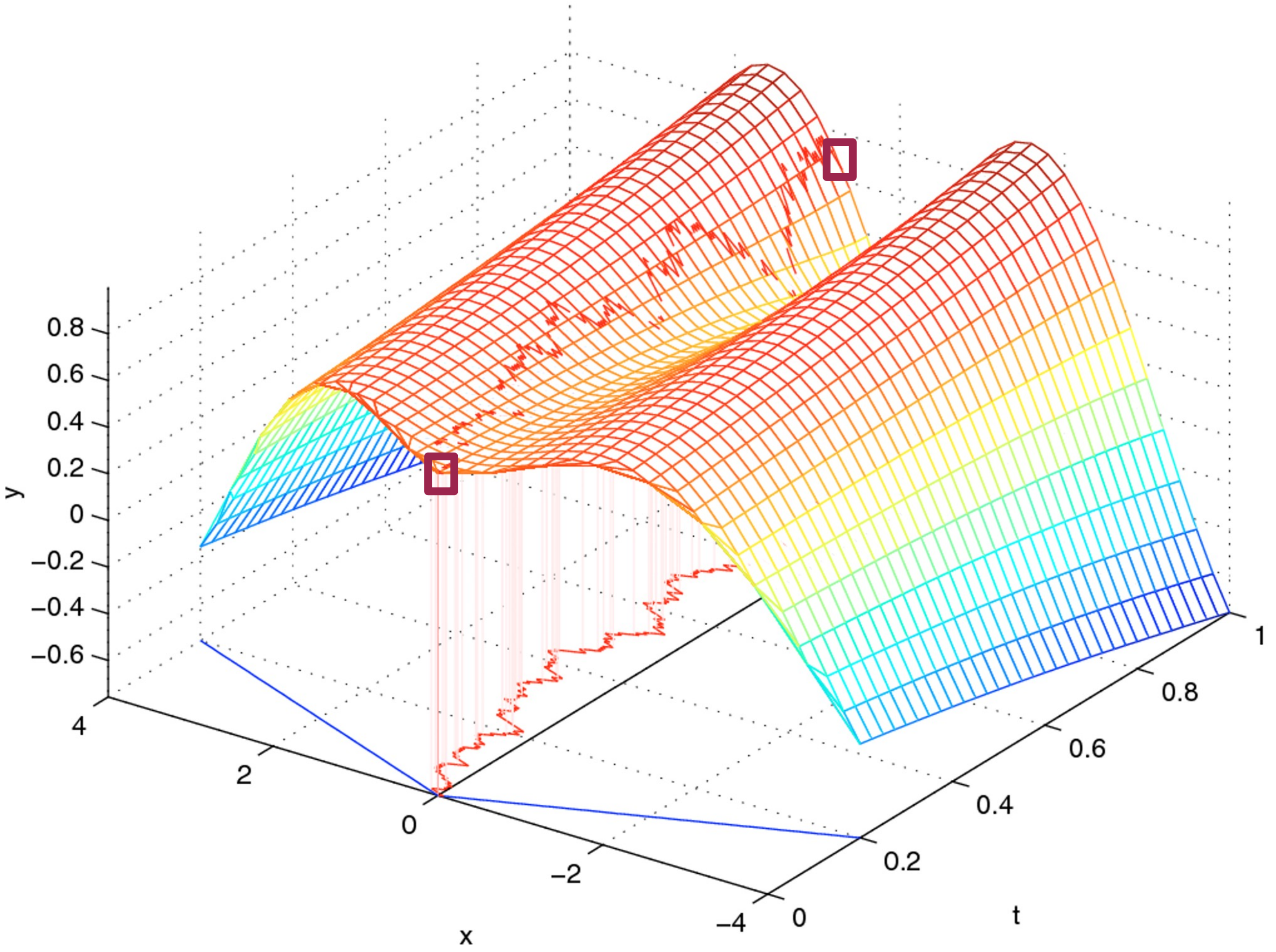
Solve *semilinear parabolic PDEs* with some specified terminal condition $u(T,x) = g(x)$

$$\frac{\partial u}{\partial t}(t,x) + \frac{1}{2}\mathrm{Tr}\Big(\sigma\sigma^{\mathrm{T}}(t,x)(\mathrm{Hess}_x u)(t,x)\Big) + \nabla u(t,x)\cdot\mu(t,x) + f\Big(t,x,u(t,x),\sigma^{\mathrm{T}}(t,x)\nabla u(t,x)\Big) = 0$$

t represents time
x represents a *d*-dimensional space variable
μ is a known vector-valued function
σ is a known matrix-valued function ($d \times d$)
$\boldsymbol{\sigma^T}$ denotes transpose associated to *σ*

∇u denotes the gradient of function ***u*** with respect to *x*
$\boldsymbol{Hess_x u}$ denotes the Hessian of function ***u*** with respect to *x*
Tr denotes the trace of a matrix
f is a known non-linear function

Goal: find the solution at $t=0$, $x=\xi$ for some vector $\xi \in R^d$



https://www.semanticscholar.org/paper/Numerical-algorithms-for-backward-stochastic-with-Peng-Xu/a3d03cb8b557ffa6b48a3c7c87cc8f0f22fd18b9

Maths behind the paper 🧮

Let $\{W_t\}_{t \in [0, T]}$ be a **d-dimensional Brownian motion**, and $\{X_t\}_{t \in [0, T]}$ be a **d-dimensional stochastic process** that satisfies:

$$X_t = \xi + \int_0^t \mu(s, X_s)\, ds + \underbrace{\int_0^t \sigma(s, X_s)\, dW_s}_{\text{Itô's integral}}.$$

Maths behind the paper 🧮

By Feynman-Kac formula, and Itô formula, the solution *u* to the PDE satisfies the following BSDE:

$$
\begin{aligned}
& u(t, X_t) - u(0, X_0) \\
= & - \int_0^t f\Big(s, X_s, u(s, X_s), \sigma^{\mathrm{T}}(s, X_s)\,\nabla u(s, X_s)\Big)\, ds \\
& + \int_0^t [\nabla u(s, X_s)]^{\mathrm{T}}\, \sigma(s, X_s)\, dW_s.
\end{aligned}
$$

**for details of the derivation refer to "Monte-Carlo Methods and Stochastic Processes" p. 206, 207*

Maths behind the paper 🧮

We apply temporal discretization to the two equations with partition $0=t_0 < t_1 < \ldots < t_N=T$:

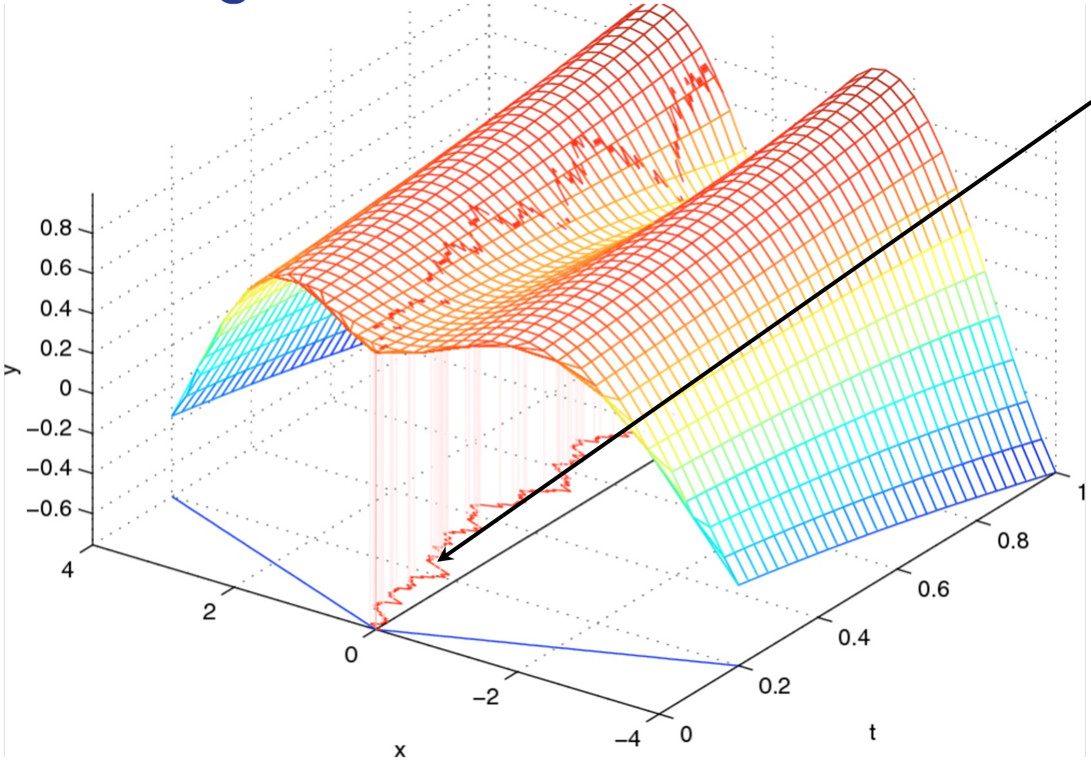
$$X_t = \xi + \int_0^t \mu(s, X_s)\, ds + \int_0^t \sigma(s, X_s)\, dW_s. \Rightarrow X_{t_{n+1}} - X_{t_n} \approx \mu(t_n, X_{t_n})\,\Delta t_n + \sigma(t_n, X_{t_n})\,\Delta W_n$$

$$\begin{aligned} & u(t, X_t) - u(0, X_0) \\ = & -\int_0^t f\Big(s, X_s, u(s, X_s), \sigma^{\mathrm{T}}(s, X_s)\,\nabla u(s, X_s)\Big)\, ds \\ & + \int_0^t [\nabla u(s, X_s)]^{\mathrm{T}}\, \sigma(s, X_s)\, dW_s. \end{aligned} \Rightarrow \begin{aligned} & u(t_{n+1}, X_{t_{n+1}}) - u(t_n, X_{t_n}) \\ \approx & -f\Big(t_n, X_{t_n}, u(t_n, X_{t_n}), \sigma^{\mathrm{T}}(t_n, X_{t_n})\,\nabla u(t_n, X_{t_n})\Big)\,\Delta t_n \\ & + [\nabla u(t_n, X_{t_n})]^{\mathrm{T}}\,\sigma(t_n, X_{t_n})\,\Delta W_n, \end{aligned}$$

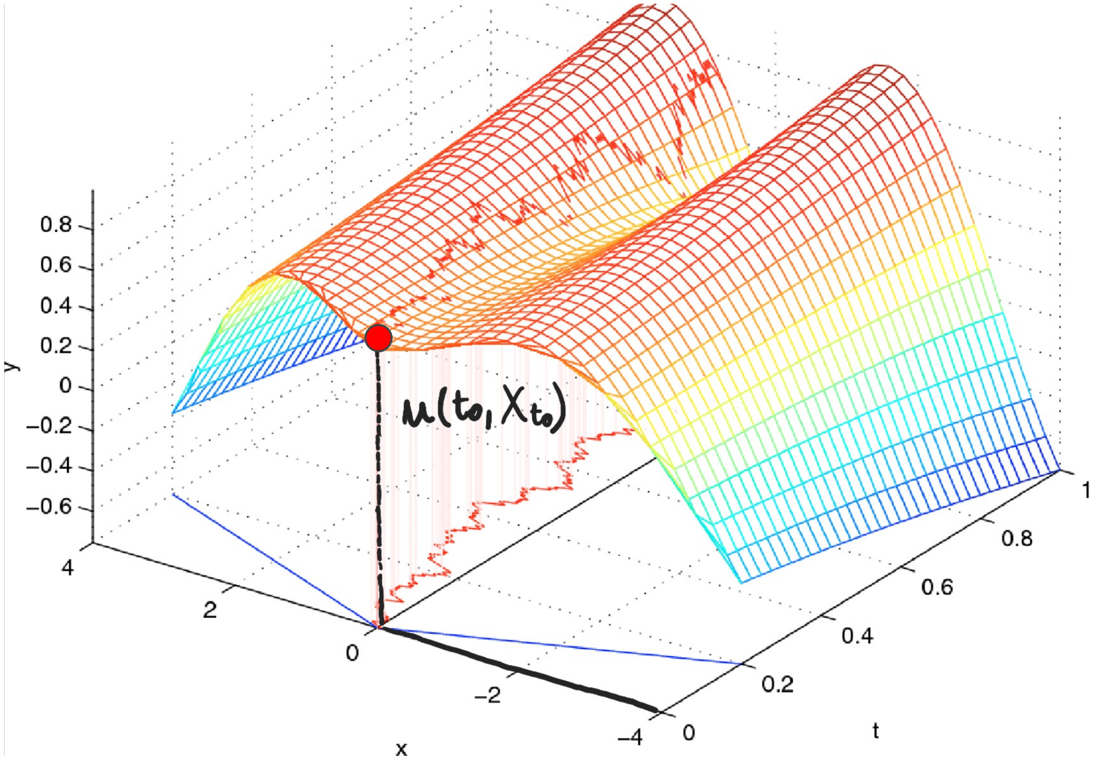
Euler scheme

where: $\Delta t_n = t_{n+1} - t_n, \quad \Delta W_n = W_{t_{n+1}} - W_{t_n}$

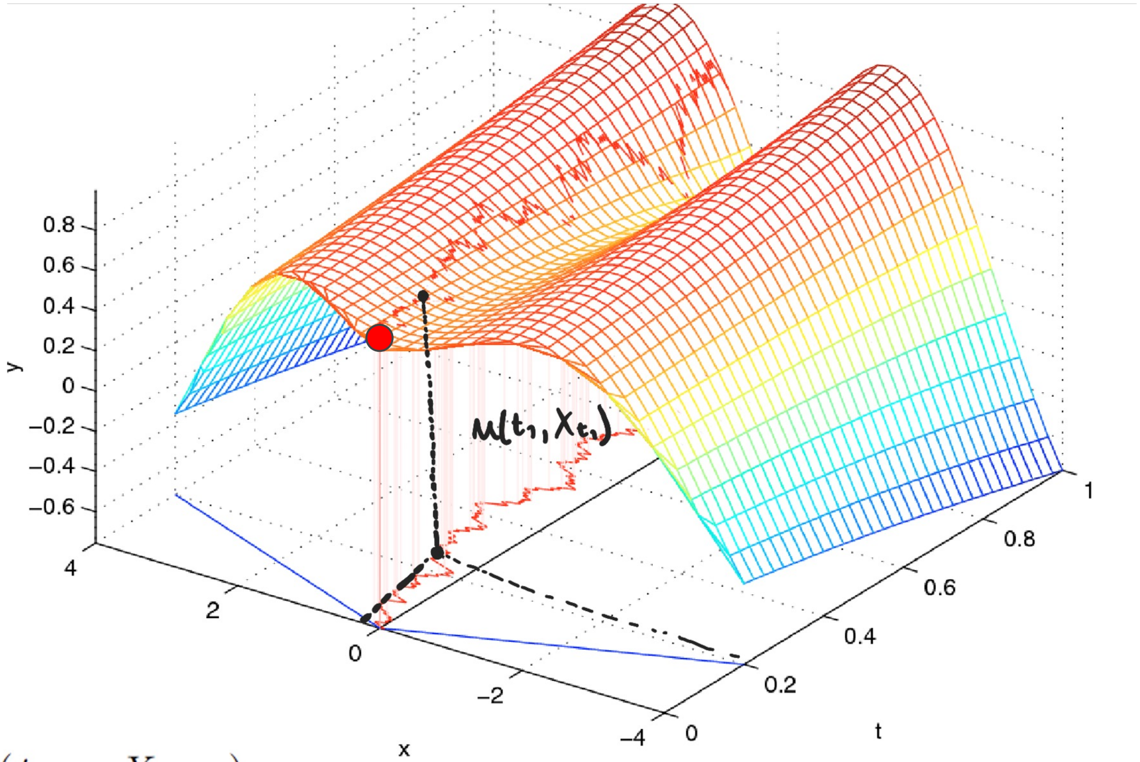
Solving the BSDE



stochastic process X

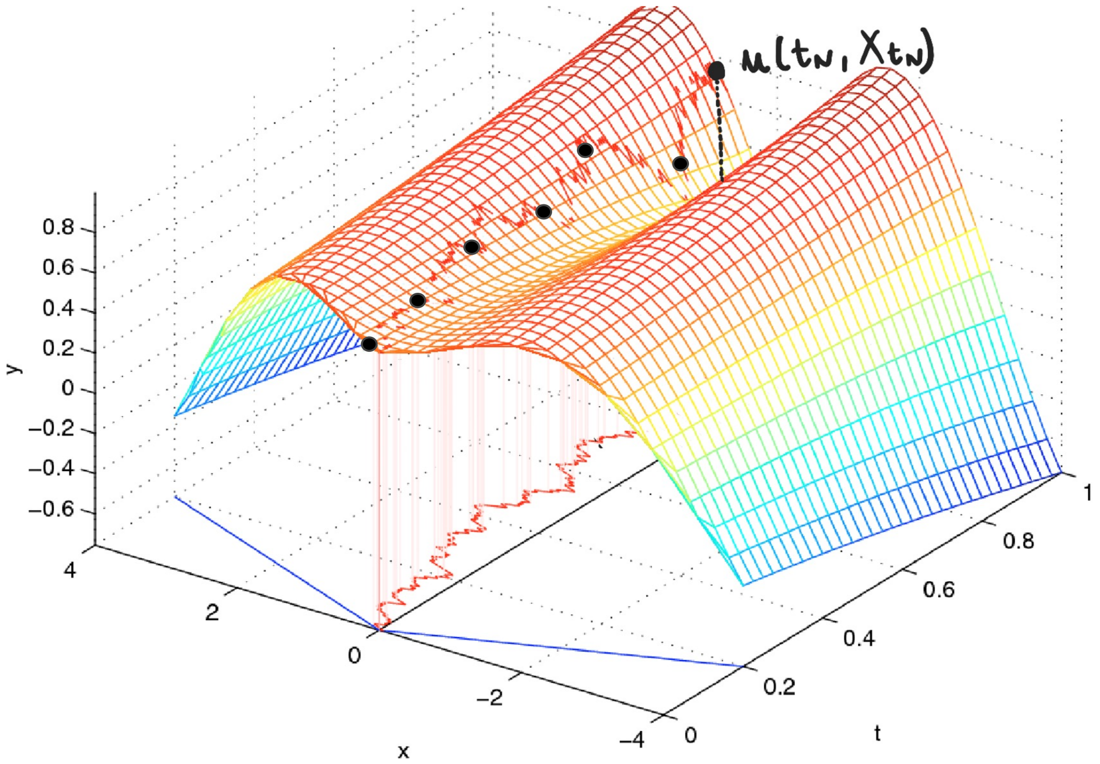


0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
y
4
2
0
−2
−4
x
0
0.2
0.4
0.6
0.8
1
t
$u(t_0, X_{t_0})$

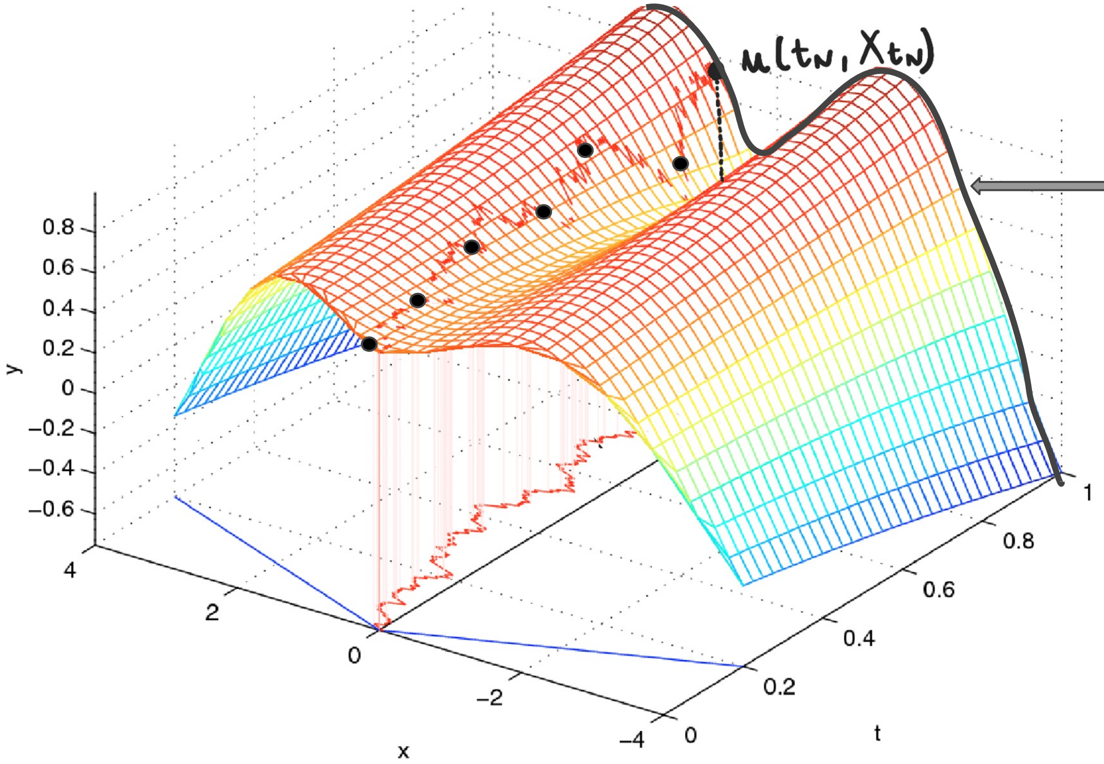


$$u(t_{n+1}, X_{t_{n+1}}) \approx$$

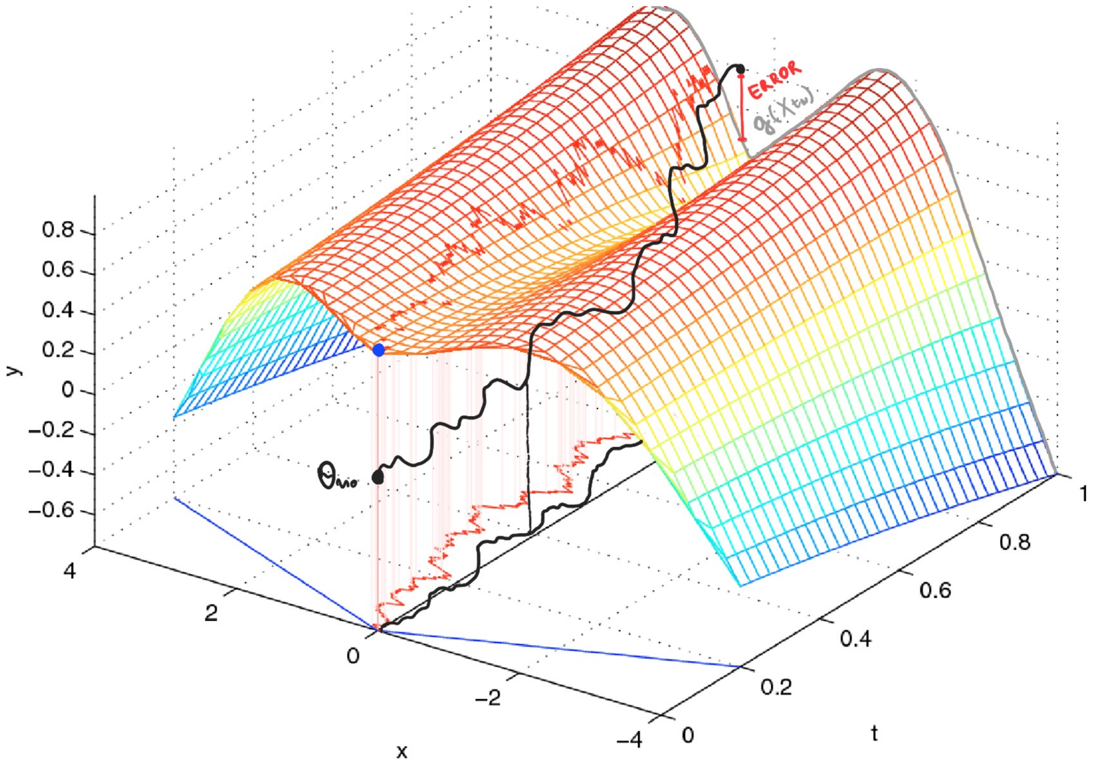
$$\underline{u(t_n, X_{t_n})} - f\Big(t_n, X_{t_n}, \underline{u(t_n, X_{t_n})}, \underbrace{\sigma^{\mathrm{T}}(t_n, X_{t_n}) \nabla u(t_n, X_{t_n})}_{?}\Big) \Delta t_n + [\underbrace{\nabla u(t_n, X_{t_n})]^{\mathrm{T}} \sigma(t_n, X_{t_n})}_{?} \Delta W_n$$



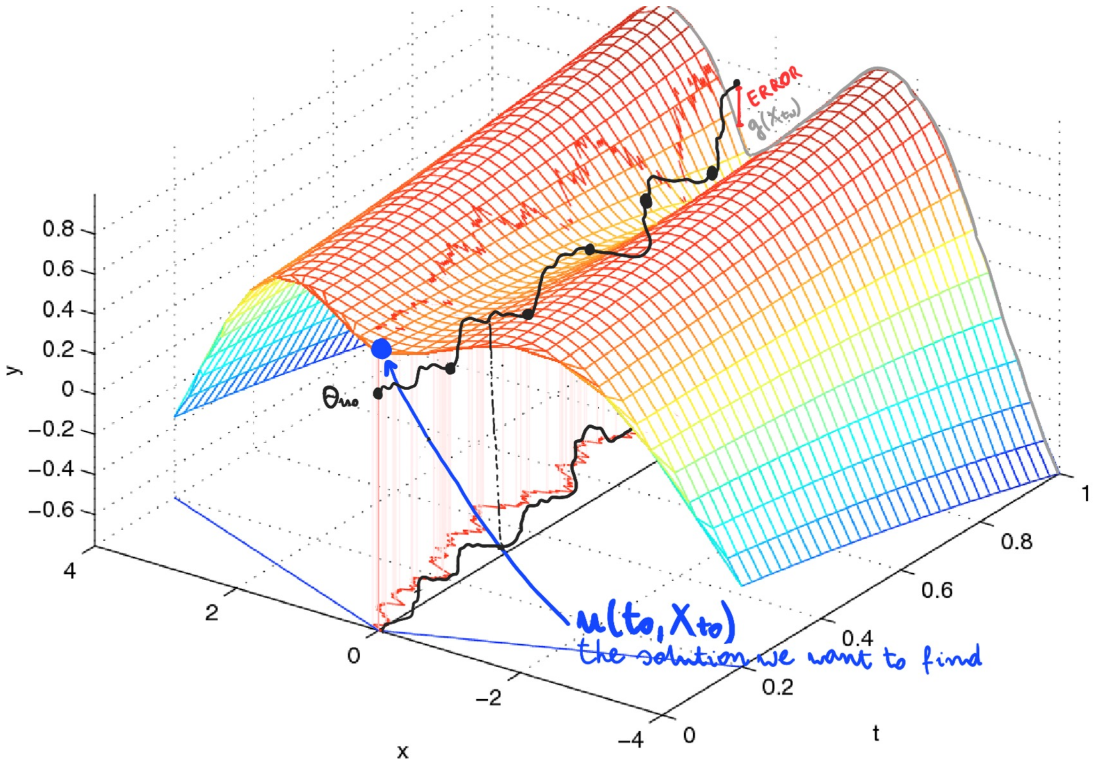
u(t_N, X_{t_N})
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
y
4
2
0
−2
−4
x
0
0.2
0.4
0.6
0.8
1
t



terminal condition

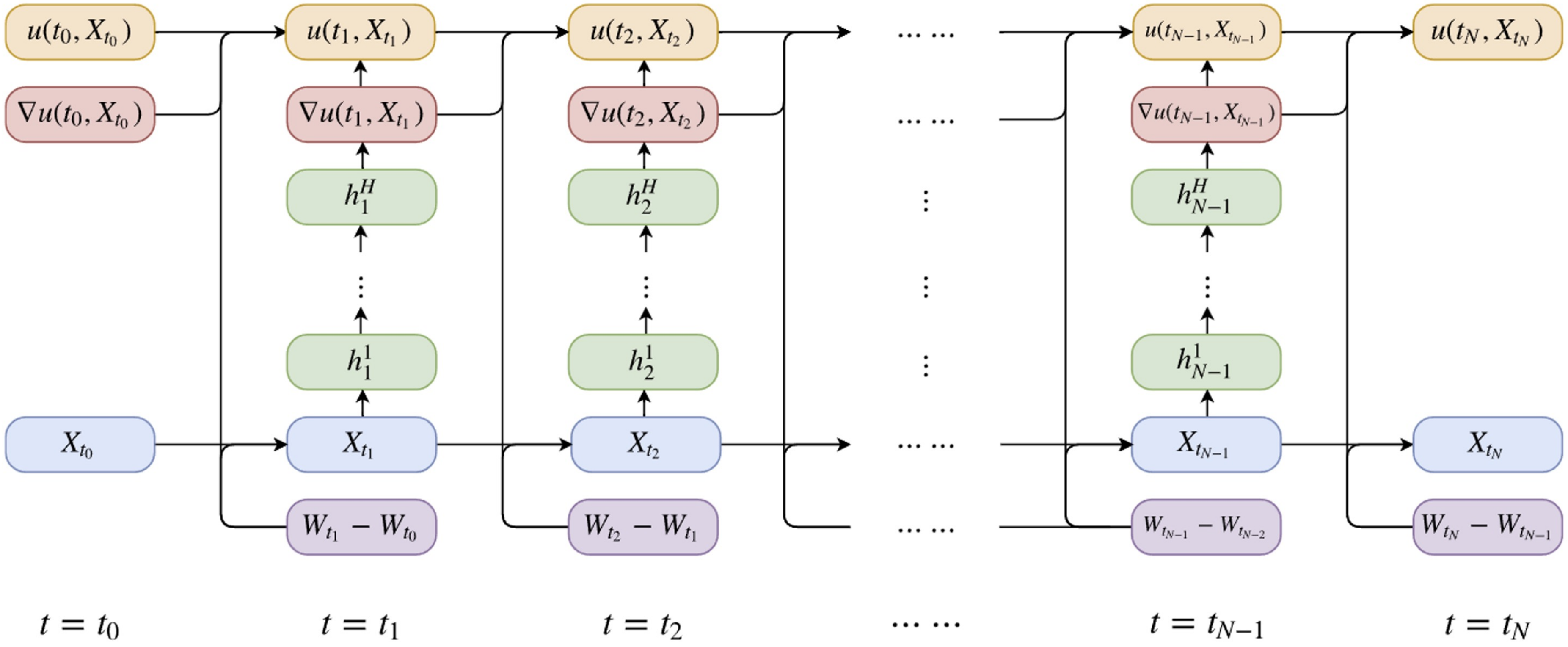


ERROR
y
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
x
4
2
0
−2
−4
t
0
0.2
0.4
0.6
0.8
1

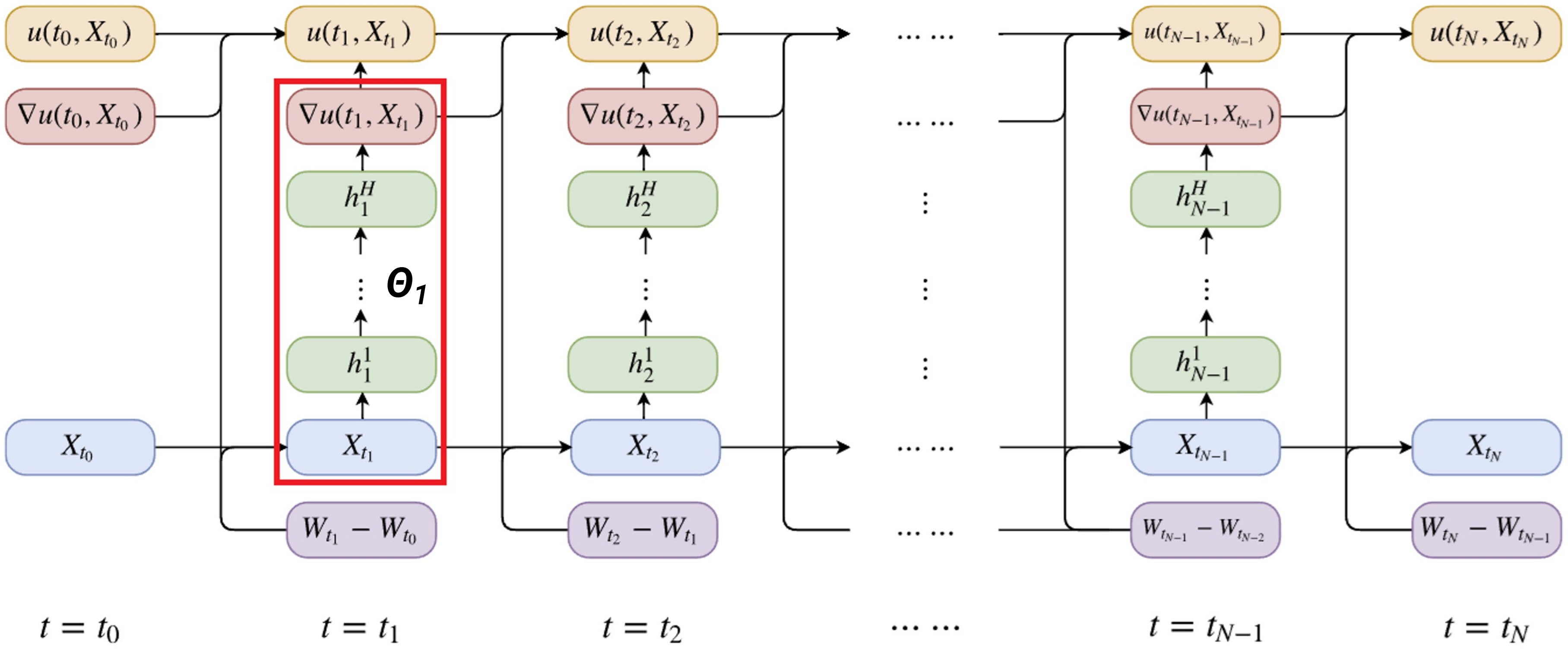


ERROR
$g(X_{t_N})$
Θ_{u0}
$u(t_0, X_{t_0})$
the solution we want to find
y
x
t
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
4
2
0
−2
−4
0
0.2
0.4
0.6
0.8
1

Structure of the neural network

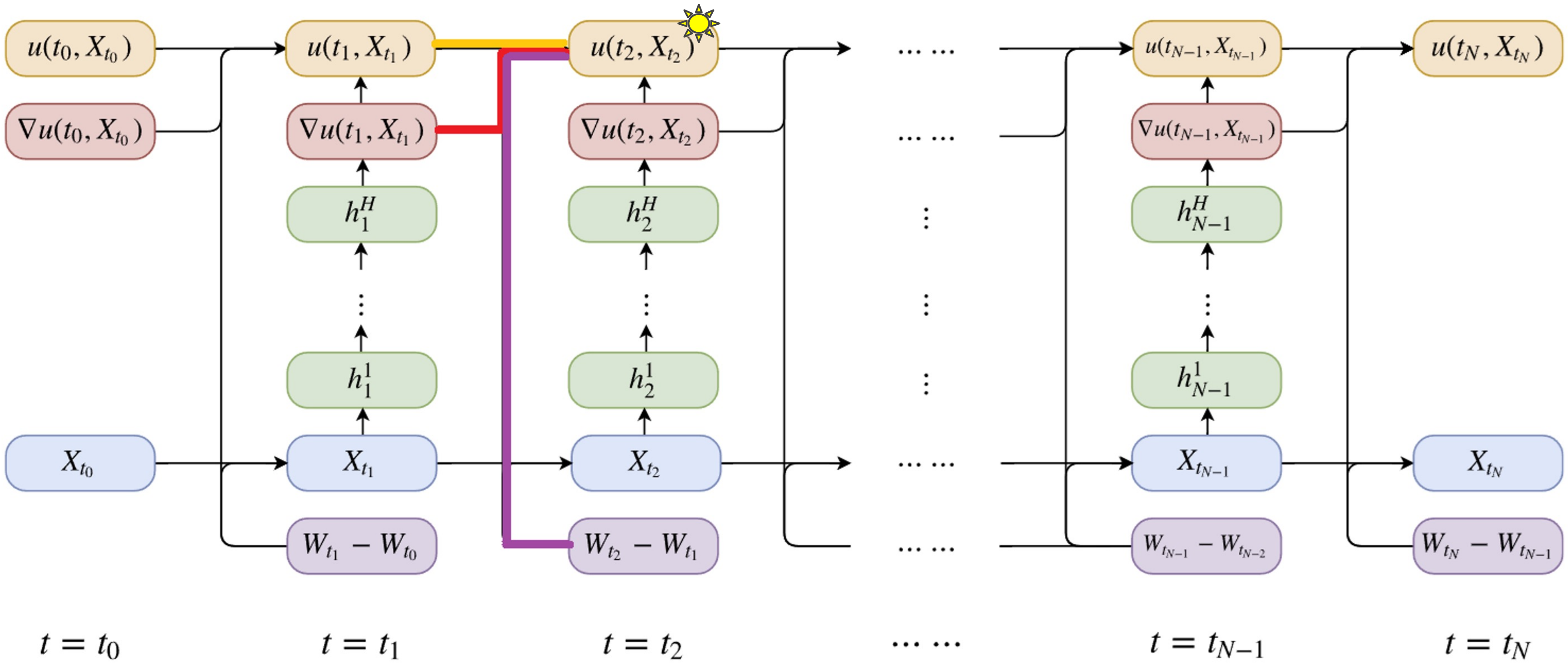


To simplify presentation σ satisfies: $\forall x \in R^d$: $\sigma(x) = Id$



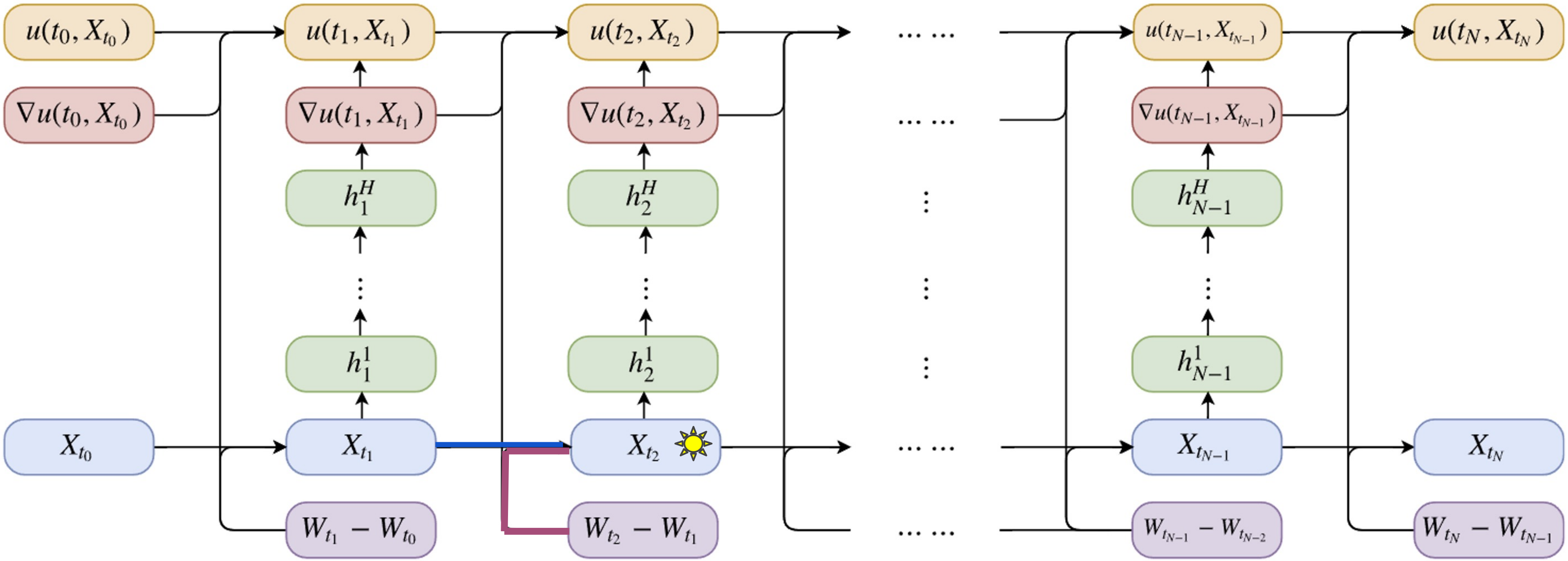
Feedforward neural network approximating the spatial gradients at time $t=t_N$

$$\sigma^{\mathrm{T}}(t_n, X_{t_n})\,\nabla u(t_n, X_{t_n}) = (\sigma^{\mathrm{T}}\nabla u)(t_n, X_{t_n}) \approx (\sigma^{\mathrm{T}}\nabla u)(t_n, X_{t_n}|\theta_n)$$

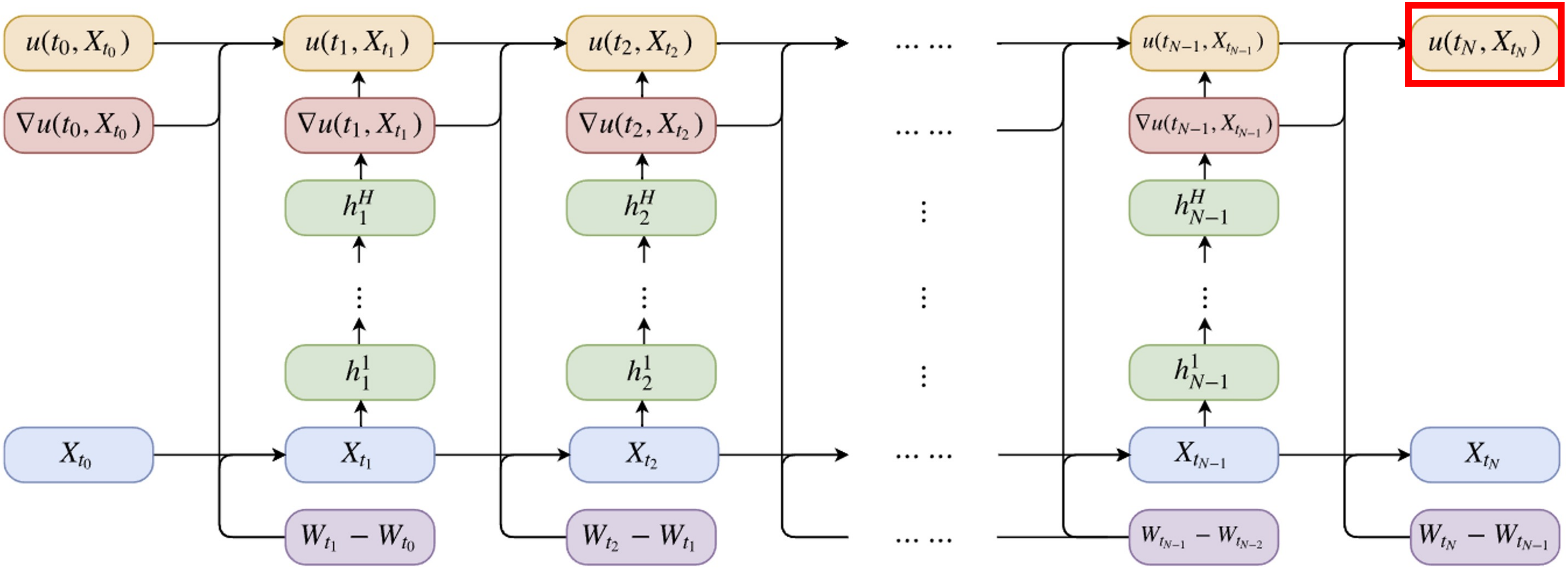


$$u(t_{n+1}, X_{t_{n+1}}) \approx$$

$$u(t_n, X_{t_n}) - f\left(t_n, X_{t_n}, u(t_n, X_{t_n}), \sigma^{\mathrm{T}}(t_n, X_{t_n})\, \nabla u(t_n, X_{t_n})\right) \Delta t_n + [\nabla u(t_n, X_{t_n})]^{\mathrm{T}} \sigma(t_n, X_{t_n})\, \Delta W_n$$



$$X_{t_{n+1}} \approx X_{t_n} + \mu(t_n, X_{t_n})\,\Delta t_n + \sigma(t_n, X_{t_n})\,\Delta W_n$$



$$l(\theta) = \mathbb{E}\left[\left| g(X_{t_N}) - \hat{u}\left(\{X_{t_n}\}_{0\le n\le N}, \{W_{t_n}\}_{0\le n\le N}\right)\right|^2\right]$$

Terminal condition

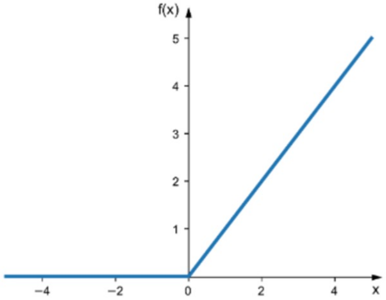
Structure of the neural network

- The total set of parameters is $\theta = \{\theta_{u_0}, \theta_{\nabla u_0}, \theta_1, \ldots, \theta_{N-1}\}$

- One can use a standard stochastic gradient descent algorithm to optimize Θ

Implementation 💻

- Each subnetwork has 4 layers
  - **d**-dimensional input layer
  - two **(d+10)**-dimensional hidden-layers
  - **d**-dimensional output layer
- Optimizer: **Adam**
- Activation function: **ReLU**



https://www.researchgate.net/figure/ReLU-function-graph_fig2_346250677
https://github.com/frankhan91/DeepBSDE/blob/master/solver.py

```python
class FeedForwardSubNet(tf.keras.Model):
    def __init__(self, config):
        super(FeedForwardSubNet, self).__init__()
        dim = config.eqn_config.dim
        num_hiddens = config.net_config.num_hiddens
        self.bn_layers = [
            tf.keras.layers.BatchNormalization(
                momentum=0.99,
                epsilon=1e-6,
                beta_initializer=tf.random_normal_initializer(0.0, stddev=0.1),
                gamma_initializer=tf.random_uniform_initializer(0.1, 0.5)
            )
            for _ in range(len(num_hiddens) + 2)]
        self.dense_layers = [tf.keras.layers.Dense(num_hiddens[i],
                                                   use_bias=False,
                                                   activation=None)
                             for i in range(len(num_hiddens))]
        # final output should be gradient of size dim
        self.dense_layers.append(tf.keras.layers.Dense(dim, activation=None))

    def call(self, x, training):
        """structure: bn -> (dense -> bn -> relu) * len(num_hiddens) -> dense -> bn"""
        x = self.bn_layers[0](x, training)
        for i in range(len(self.dense_layers) - 1):
            x = self.dense_layers[i](x)
            x = self.bn_layers[i+1](x, training)
            x = tf.nn.relu(x)
        x = self.dense_layers[-1](x)
        x = self.bn_layers[-1](x, training)
        return x
```

Examples in practice 1 (finance) 📈

Parabolic PDEs (**Black-Scholes** equations), allow to deduce the theoretical estimate of the price of European-style options.

Options are financial derivatives that give buyers the right, but not the obligation, to buy or sell an underlying asset at an agreed-upon price and date.

European-style options can only be exercised on the day of expiration.

Examples in practice 1 (finance) 📈

Traditional Black-Scholes model can be extended by some important factors in real markets, including defaultable securities, transaction costs etc.

Disregarded: **default risk**

https://www.imd.org/research-knowledge/economics/articles/the-european-crisis-business-threats-and-opportunities/

Examples in practice 1 (finance)

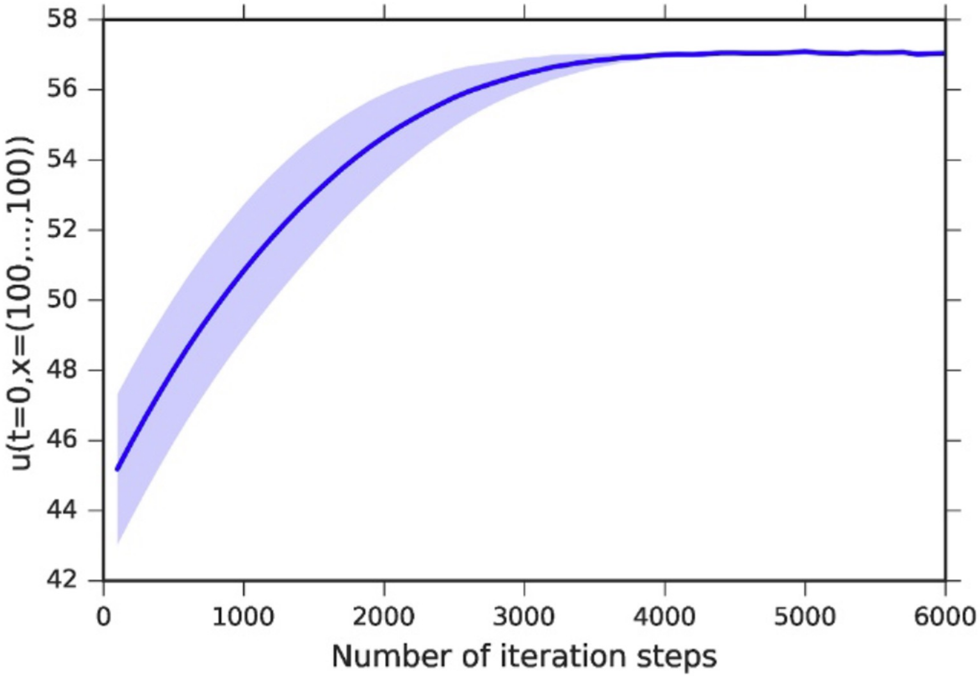
The associated Black-Scholes equation in [0, T] × R^{100}

$$\frac{\partial u}{\partial t}(t,x) + \bar{\mu} x \cdot \nabla u(t,x) + \frac{\bar{\sigma}^2}{2} \sum_{i=1}^{d} |x_i|^2 \frac{\partial^2 u}{\partial x_i^2}(t,x) - (1-\delta) Q(u(t,x))\, u(t,x) - R\, u(t,x) = 0$$

f

$$\frac{\partial u}{\partial t}(t,x) + \frac{1}{2} \mathrm{Tr}\left(\sigma\sigma^{\mathrm{T}}(t,x)(\mathrm{Hess}_x u)(t,x)\right) + \nabla u(t,x) \cdot \mu(t,x) + f$$

Examples in practice 1 (finance) 📈



The deep BSDE method achieves a relative error of size **0.46%**

Traditional approximative *Picard method*

≈57.300

training time 1607s

Examples in practice 2 (HJB) 💻

- **Hamilton-Jacobi-Bellman equation** is a concept of control theory
  - Deals with the control of dynamic systems
  - Typical questions are:
    - Is the system stable?
    - Is it possible to bring the system to a certain state of choice?
    - How should the input variable be chosen in order to achieve a target state in the shortest possible time and with the least amount of effort?

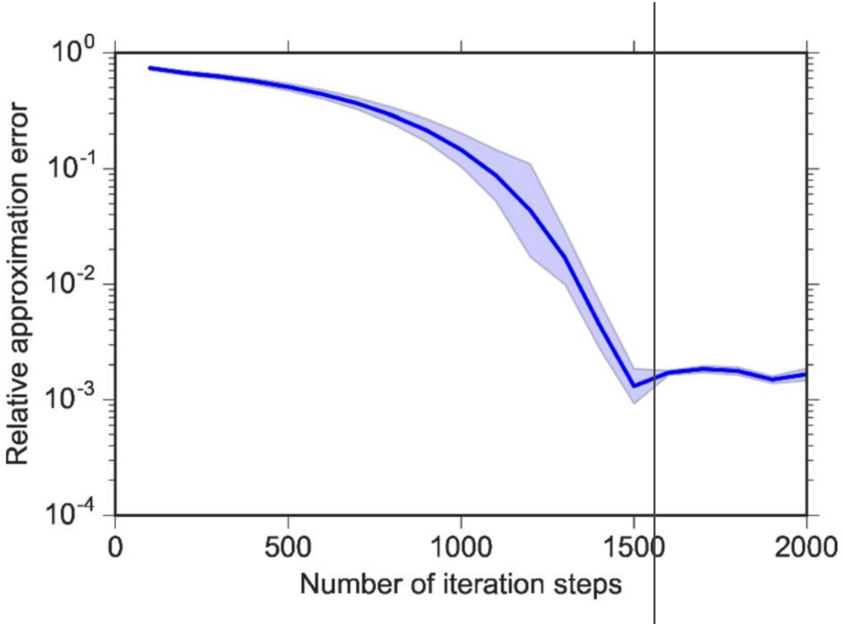
→ Highly relevant in practice

Examples in practice 2 (HJB) 💻

$$\frac{\partial V}{\partial t} + \min_{u} \{L(x,u,t) + \nabla V \cdot f(x,u,t)\} = 0$$

- $\frac{\partial V}{\partial t}$: value function with respect to time
- $\min_{u}$: minimum over all possible control inputs u
- $L(x,u,t)$: instantaneous cost incurred by applying control u at state x at time t
- ∇V: value function with respect to state x
- $f(x,u,t)$: system evolves over time in response to the control input u

Examples in practice 2 (HJB) 💻



→ relative error of size 0.17% in a run time of 303s (MacBook Pro)

Conclusion (pros) 💭

- The paper introduces an effective method for solving high-dimensional parabolic PDEs, overcoming the curse of dimensionality problem
  → relative error of size 0.46% (compared to benchmark solution for Black-Scholes equation)
  → training time 1607s (MacBook Pro with a 2.9GHz Intel Core i5 Processor and 16GB RAM)
- Opens up new possibilities in economics, finance, and operational research
- Similar methodology can be used to solve model based stochastic control problems, in which the optimal policies are approximated by neural nets

Conclusion (challenges) 💭

According to paper: Not able to deal with the **quantum many-body problem**

→ Behaviour of systems composed of many interacting quantum particles

✅ Classical physics: Predicting interactions is possible

❌ Quantum physics: Not applicable due to its laws

Conclusion (further questions) 💭

Payoff?

Hyperparameter optimization?

Theoretical guarantees?

Impact & follow-up work

- Beck et al. 2017 (deep 2BSDE method)
- Henry-Labordére 2017 (deep primal-dual for BSDEs)
- Fujii et al. 2017 (deep BSDE with asymptotic expansion)
- Becker et al. 2018 (deep optimal stopping)
- Raissi 2018, Beck et al. 2018, Chan-Wai-Nam et al. 2018, Huré et al. 2019
- European Journal of Applied Mathematics

Thank you for your attention!

Questions?

Sources

- J. Han, A. Jentzen, W. E, Solving high-dimensional partial differential equations using deep learning. Proc. Natl. Acad. Sci. U.S.A.115, 8505–8510 (2018)
- https://web.mit.edu/8.334/www/grades/projects/projects17/OscarMickelin/brownian.html
- https://en.wikipedia.org/wiki/Wiener_process#:~:text=In%20mathematics%2C%20the%20Wiener%20process,the%20one%2Ddimensional%20Brownian%20motion.
- https://visualpde.com/
- https://en.wikipedia.org/wiki/Black%E2%80%93Scholes_model
- https://web.math.princeton.edu/~weinan/control.pdf
- Emmanuel Gabet, Monte-Carlo Methods and Stochastic Processes: From Linear to Non-Linear