Deep Learning for Partial Differential Equations

William Wei

National Center for Supercomputing Applications
Center for Artificial Intelligence Innovation
Department of Physics, University of Illinois Urban-Champaign

Reference: http://www.dam.brown.edu/people/mraissi/research/1_physics_informed_neural_networks/

General formulation of partial differential equations (PDEs)

$$u_t + \mathcal{N}[u;\lambda] = 0,\ x \in \Omega,\ t \in [0,T]$$

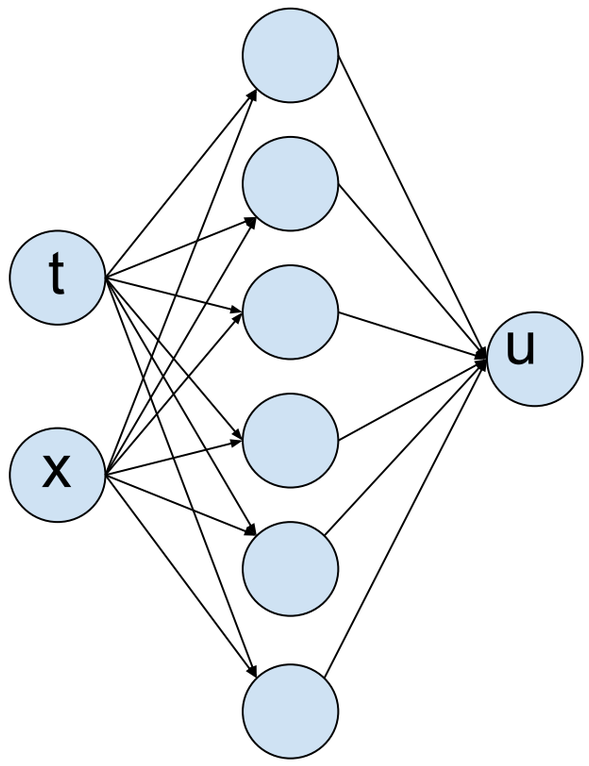
$u(t,x)$ **is the solution to the equation.**

$\mathcal{N}[u;\lambda]$ **is a non-linear function of** $u(t,x)$ **parameterized by** λ

- **Data-driven solutions for PDEs**
- **Data-driven discovery of PDEs**

Data-driven solutions for PDEs: Continuous Time Models

$$u(t,x) \quad =$$



$$u_t + \mathcal{N}[u;\lambda] = 0$$

Minimize the loss function: $f := u_t + \mathcal{N}[u;\lambda]$

```
def u(t, x):
    u = neural_net(tf.concat([t,x],1), weights,
biases)
    return u
```

```
def f(t, x):
    u = u(t, x)
    u_t = tf.gradients(u, t)[0]
    u_x = tf.gradients(u, x)[0]
    u_xx = tf.gradients(u_x, x)[0]
    f = u_t + u*u_x - (0.01/tf.pi)*u_xx
    return f
```

Example: Burgers

$$u_t + uu_x - (0.01/\pi)u_{xx} = 0,\ x \in [-1,1],\ t \in [0,1],$$

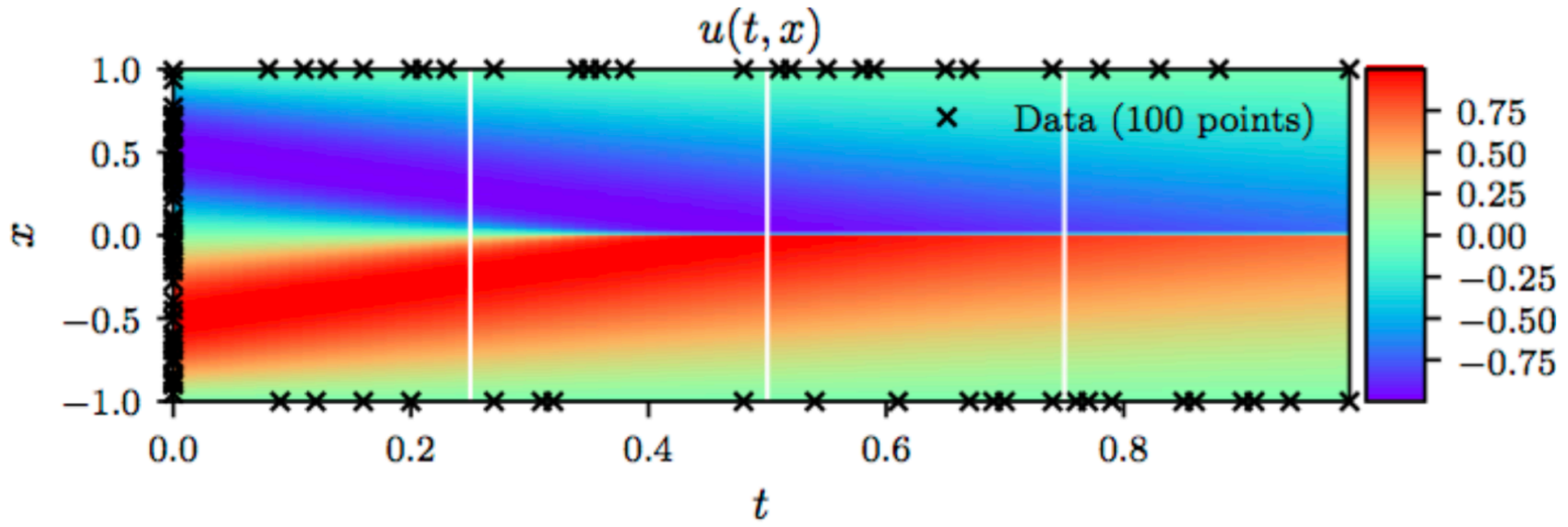
$$u(0,x) = -sin(\pi x)$$

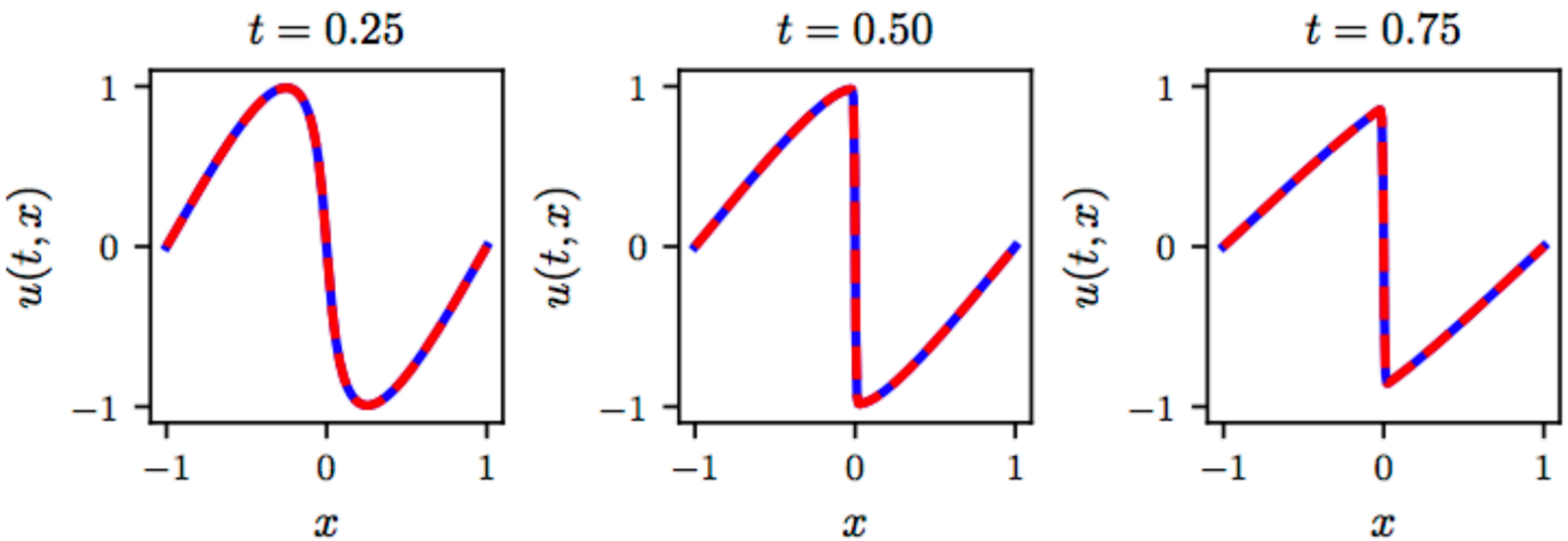
$$u(t,-1) = u(t,1) = 0$$

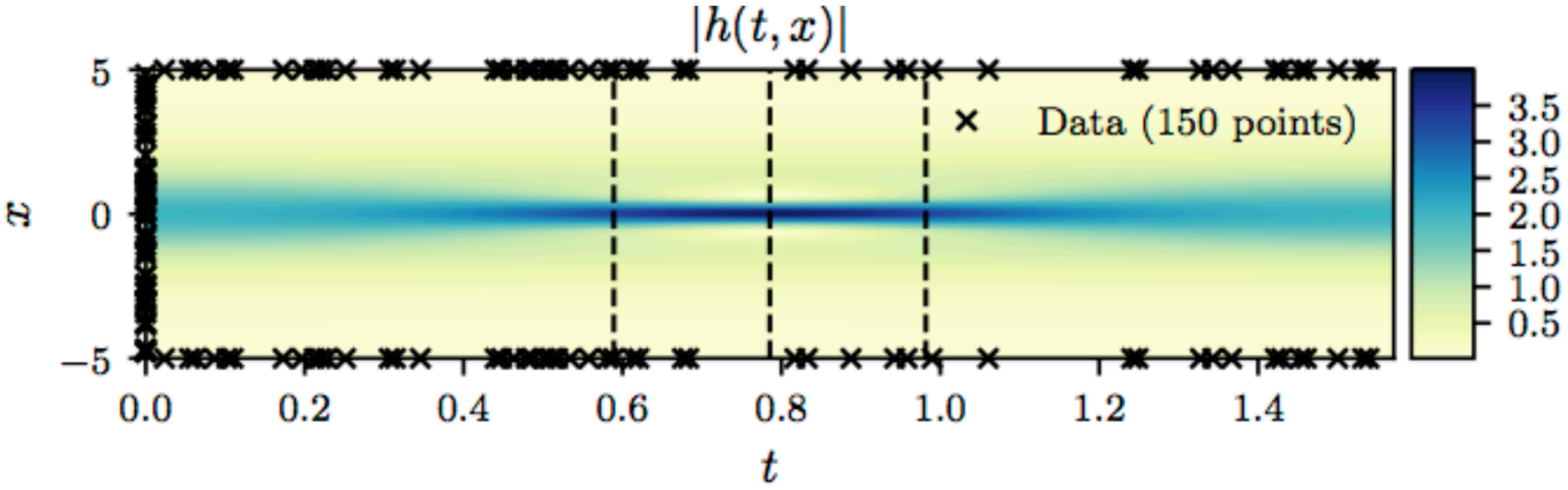
Neural network approximation:

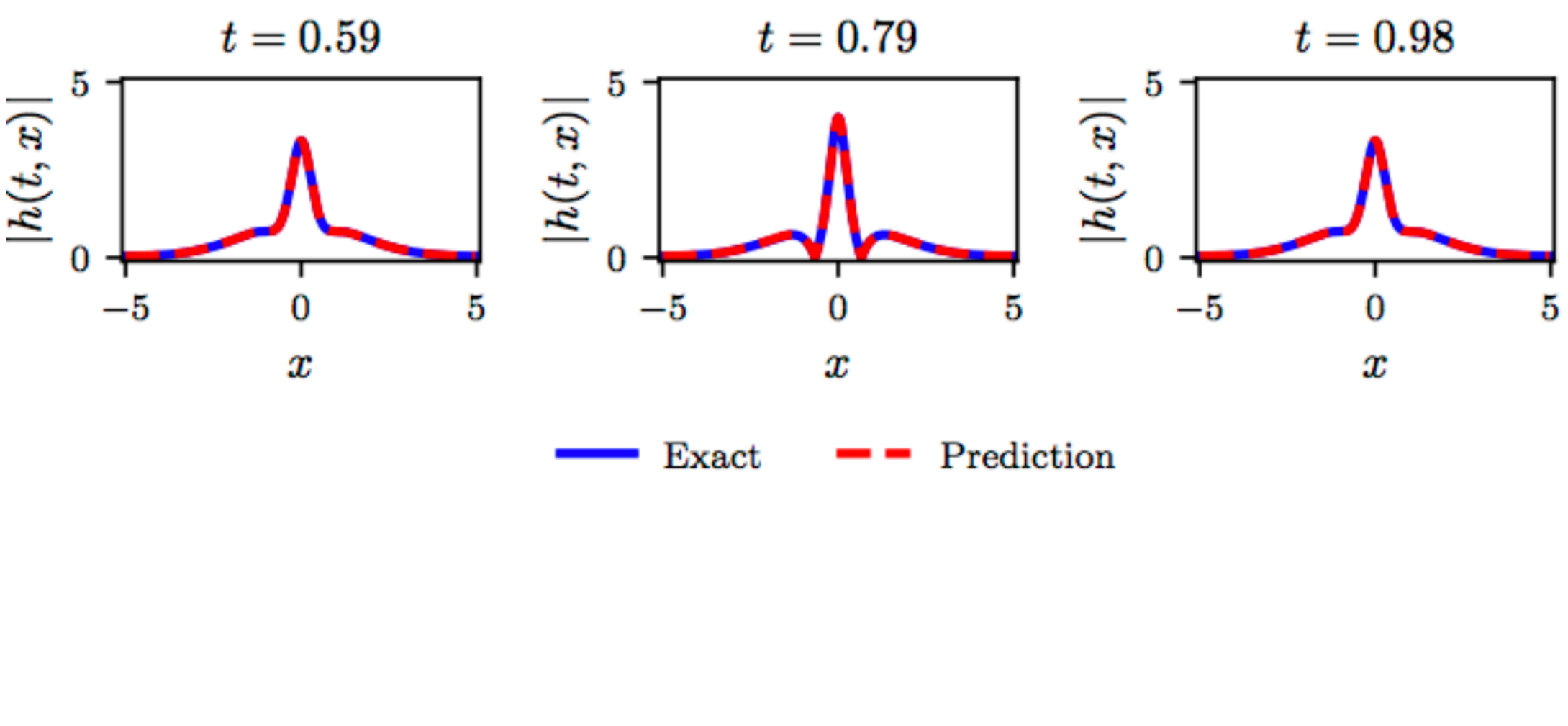
$$u(t,x) \approx \hat{u}_\theta(t,x) \qquad f(\theta) := \hat{u}_t + \hat{u}\hat{u}_x - (0.01/\pi)u_{xx}$$

$$L(\theta) = \|f(\theta)\|^2 + \|\hat{u}(0,x) - u(0,x)\|^2 + \|\hat{u}(t,-1)\|^2 + \|\hat{u}(t,1)\|^2$$


$u(t,x)$
Data (100 points)
x
t
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75


$t = 0.25$
$t = 0.50$
$t = 0.75$
$u(t,x)$
x
1
0
−1
Exact
Prediction


$|h(t,x)|$
5
0
−5
x
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
t
Data (150 points)
3.5
3.0
2.5
2.0
1.5
1.0
0.5


$t = 0.59$
$t = 0.79$
$t = 0.98$
$|h(t,x)|$
5
0
−5
0
5
x
Exact
Prediction

Data-driven solutions for PDEs: Discrete Time Models

$$u_t + \mathcal{N}[u;\lambda] = 0$$

$$u(t) \approx \{u_0, u_1, \ldots, u_n, \ldots u_N\}$$

Euler method

$$u_{n+1} = u_n - \Delta t \mathcal{N}[u_n;\lambda]$$

Runge-Kutta methods:

$$u^{n+c_i} = u^n - \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u^{n+c_j}], i = 1,...,q$$

$$u^{n+1} = u^n - \Delta t \sum_{j=1}^{q} b_j \mathcal{N}[u^{n+c_j}]$$

$$u_i^n := u^{n+c_i} + \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u^{n+c_j}], i = 1,...,q$$

$$u_{q+1}^n := u^{n+1} + \Delta t \sum_{j=1}^{q} b_j \mathcal{N}[u^{n+c_j}]$$

$$u^n = u_i^n, \; i = 1,...,q$$

$$u^n = u_{q+1}^n$$

multi-output neural network

$$[u^{n+c_1}(x), u^{n+c_2}(x), \ldots, u^{n+c_q}(x), u^{n+1}(x)]$$

$$u_i^n := u^{n+c_i} + \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u^{n+c_j}], i = 1,...,q$$

$$u_{q+1}^n := u^{n+1} + \Delta t \sum_{j=1}^{q} b_j \mathcal{N}[u^{n+c_j}]$$

$$[u_1^n(x), u_2^n(x), \ldots, u_q^n(x), u_{q+1}^n(x)]$$

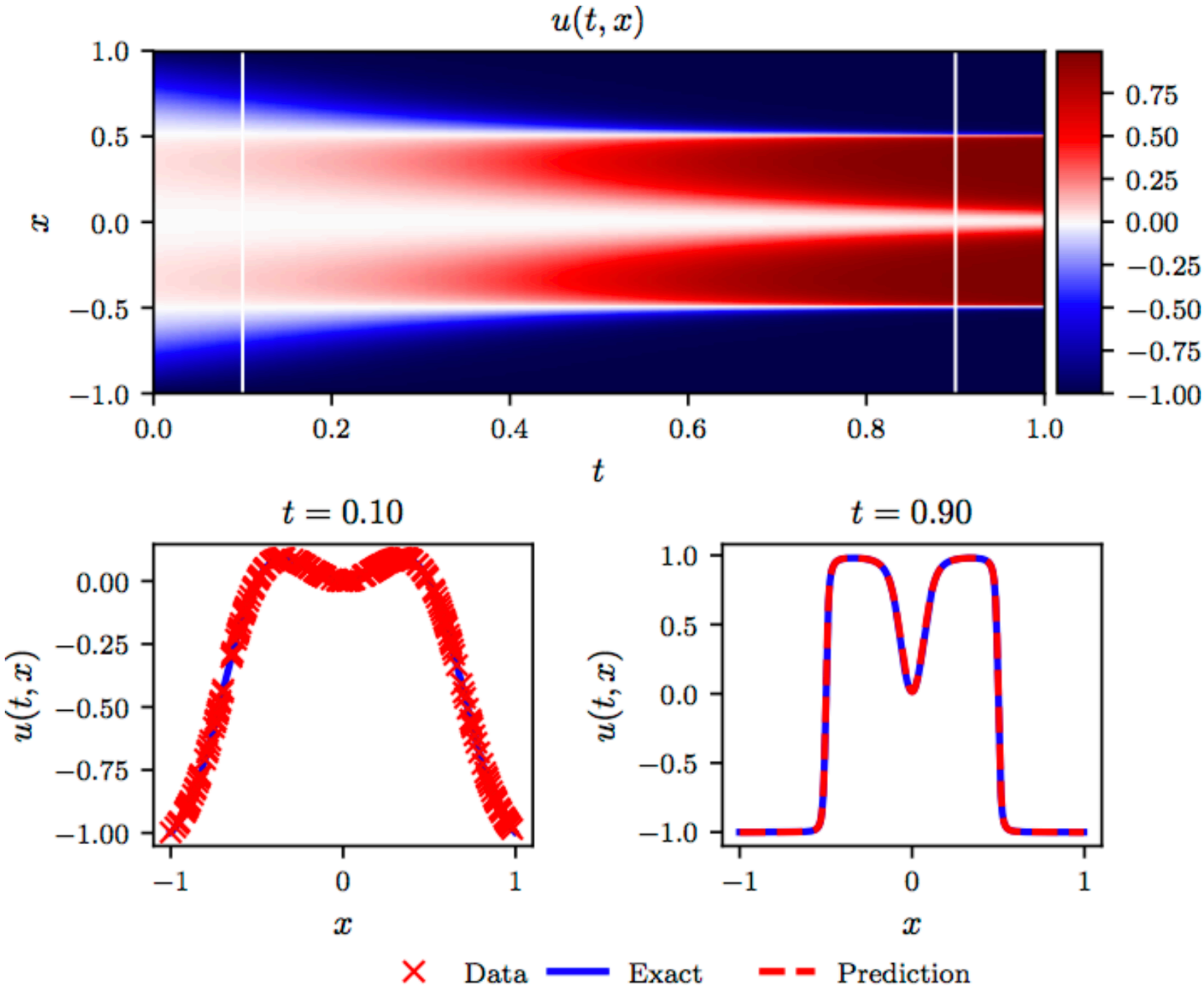
$$u^n = u_i^n,\ i = 1,\ldots,q$$

$$u^n = u_{q+1}^n$$

Temporal error accumulation :

$$\mathcal{O}(\Delta t^{2q})$$

$$\Delta t = 0.8,\ q = 500,\ \Delta t^{2q} = 0.8^{1000} \approx 10^{-97}$$


$u(t,x)$
1.0
0.5
0.0
−0.5
−1.0
x
0.0
0.2
0.4
0.6
0.8
1.0
t
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
$t = 0.10$
$t = 0.90$
0.00
−0.25
−0.50
−0.75
−1.00
$u(t,x)$
−1
0
1
x
1.0
0.5
0.0
−0.5
−1.0
$u(t,x)$
−1
0
1
x
Data
Exact
Prediction