Parameter estimation of partial differential equations via neural networks

Alexander Glushko, Dmitry I. Kabanov

Final project on the Stochastic Numerics course

6 August 2019

RWTH AACHEN UNIVERSITY

Outline

- Problem of parameter estimation of partial differential equations (inverse problem)
- How neural networks are used for the problem
- Intro to neural networks
- Example 1: linear heat equation
- Example 2: viscous Burger's equation

Inverse problem

Consider data

$$\boldsymbol{D} = \{(t_i, x_i), u_i\}, \quad i = 1, ..., N$$

observed from the model given by a partial differential equation of the form

$$u_t + \mathcal{N}(u; \boldsymbol{\lambda}) = 0,$$

with some initial and boundary conditions and **unknown** $\boldsymbol{\lambda}$

Goal: estimate $\boldsymbol{\lambda}$

$u = u(x, t)$ the solution of the equation (observations)
$\mathcal{N}(u; \boldsymbol{\lambda})$ a nonlinear algebraic-differential operator
$\boldsymbol{\lambda}$ the vector of unknown parameters

The optimal $\boldsymbol{\lambda}$ found through maximization of the posterior distribution (Bayes' rule)[1]

$$\rho(\boldsymbol{\lambda}|\boldsymbol{D}) \propto \rho(\boldsymbol{D}|\boldsymbol{\lambda}) \times \rho(\boldsymbol{\lambda}).$$

Furthermore, we assume

$$\begin{aligned} &1)\ u_i = u(x_i, t_i; \boldsymbol{\lambda}) + \epsilon_i, \quad i = 1, \ldots, N, \quad \epsilon_i \sim N(0, \sigma^2), \\ &2)\ \text{for all } \boldsymbol{\lambda} \text{ prior for } \boldsymbol{\lambda}: \quad \rho(\boldsymbol{\lambda}) = \text{const}, \end{aligned}$$

so that, the problem of finding $\boldsymbol{\lambda}$ is a nonlinear unconstrained optimization problem

$$\arg\min_{\boldsymbol{\lambda}} \quad \sum_{i=1}^{N} \left[u_i - u(x_i, t_i; \boldsymbol{\lambda})\right]^2,$$

[1] D. Sivia and J. Skilling. *Data analysis: a Bayesian tutorial*. OUP Oxford, 2006.

Analytical solution of the optimization problem (3) can be expensive, so we replace $u(x_i, t_i; \boldsymbol{\lambda})$ with a feedforward neural network[2].

To ensure that $u_{\mathrm{NN}}(x, t; \boldsymbol{\theta})$ is close to the exact solution of Eq. (2), we set the problem of estimating of the unknown parameters $\boldsymbol{\lambda}$:

$$\underset{\boldsymbol{\lambda}, \boldsymbol{\theta}}{\arg\min} \quad \sum_{i=1}^{N} \left[u_i - u_{\mathrm{NN}}(x_i, t_i; \boldsymbol{\theta}) \right]^2 \tag{1a}$$

$$\text{subject to} \quad \left\| u_{\mathrm{NN},t} + \mathcal{N}(u_{\mathrm{NN}}; \boldsymbol{\lambda}) \right\| \leq \epsilon \tag{1b}$$

for some $\epsilon \ll 1$.

---

[2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016; M. Raissi, P. Perdikaris, and G. E. Karniadakis. "Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations". In: *arXiv preprint arXiv:1711.10566* (2017).

From constrained to unconstrained optimization problem

1. Introduce auxiliary function

$$f_{\text{NN}}(x, t; \boldsymbol{\lambda}, \boldsymbol{\theta}) = u_{\text{NN},t} + \mathcal{N}(u_{\text{NN}}; \boldsymbol{\lambda}),$$

where we plug the neural network u_{NN}.

2. Clearly, we want $\|f_{\text{NN}}\| \approx 0$ for all (x, t).
Relax it: *not for all* (x, t) but only at the observation points.
As a result, we replace constrained problem with unconstrained problem via Lagrangian relaxation

$$\arg\min_{\boldsymbol{\lambda},\boldsymbol{\theta}} \sum_{i=1}^{N} \left[u_i - u_{\text{NN}}(x_i, t_i; \boldsymbol{\theta})\right]^2 + \gamma \sum_{i=1}^{N} \left[f_{\text{NN}}(x_i, t_i; \boldsymbol{\lambda}, \boldsymbol{\theta})\right]^2,$$

where γ is a regularization hyperparameter that is chosen via cross-validation.

Solving optimization problems

- Optimization problem (i. e., network training) is done via Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm.
- Quasi-Newton algorithm that approximates the Hessian matrix of the objective function to obtain second-order convergence

Description of neural networks

Let us recall the neural network equation:

$$u_{\mathsf{NN}}(x,t;\boldsymbol{\theta}) = g_L \circ g_{L-1} \circ \cdots \circ g_1,$$

where

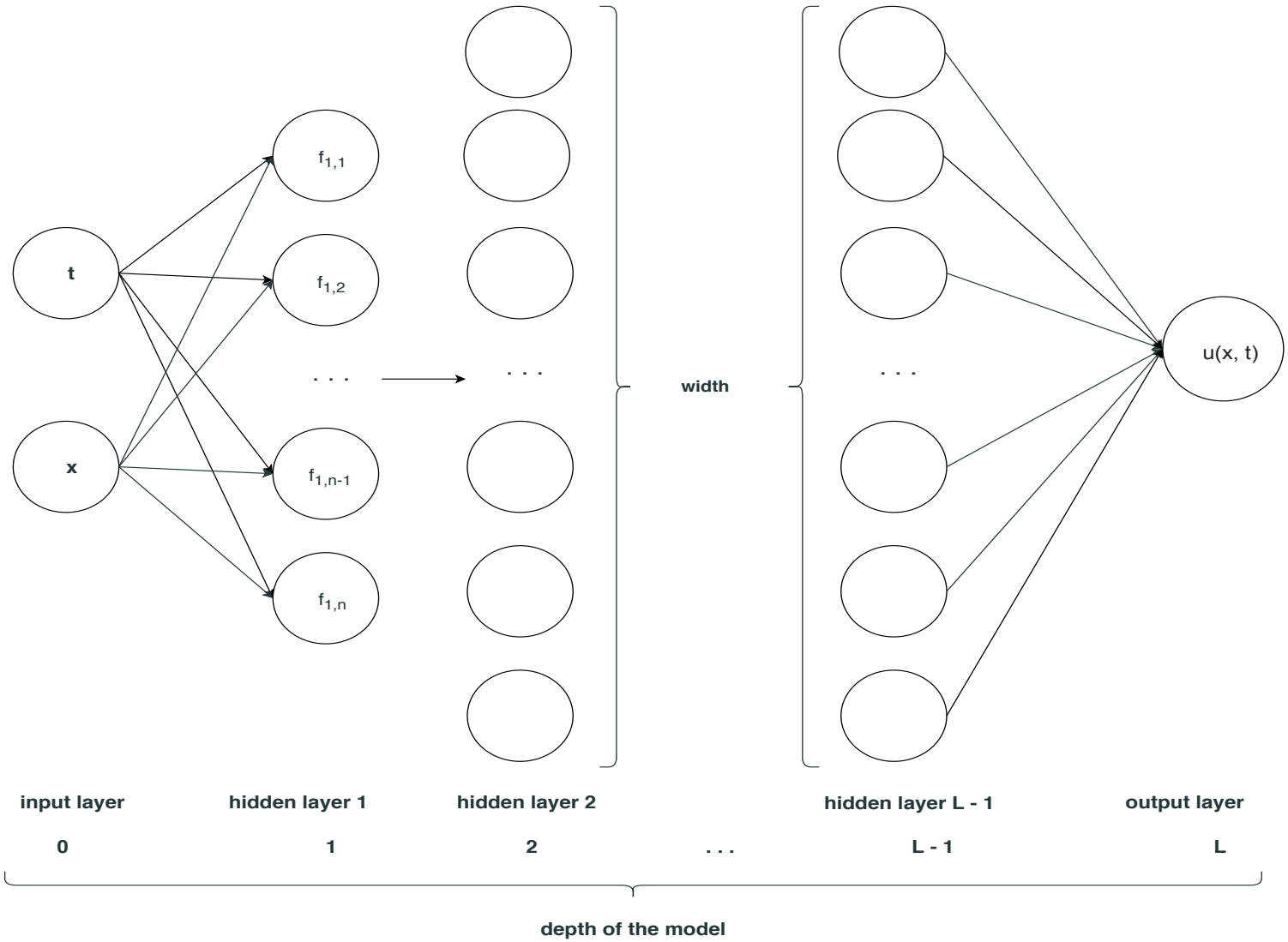
$$g_\ell(z;\boldsymbol{\theta}_\ell) = \sigma(W_\ell z + b_\ell), \quad \ell = 1,\ldots,L.$$

Here

- L - is the number of layers in the network,
- 0s layer and Lth layer are input and output layers, respectively,
- layers from 1 to $L-1$ - are hidden layers,
- $\sigma = \tanh(z)$ - is a nonlinear activation function applied componentwise.

The neural-network parameter $\boldsymbol{\theta}$ contains the components of matrices $W_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ and bias vectors $b_\ell \in \mathbb{R}^{n_\ell}$, where n_ℓ (number of "neurons") denotes the width of the ℓ^{th} layer.

The general scheme of the feedforward NN (a.k.a. multilayer perceptron, a.k.a. fully connected NN) that we use:



Why neural network is better than solving full PDE problem?

Neural network must be evaluated only at the observation points.

PDE problem must be computed on a grid, which includes observation points as a subset

Neural network rough computational complexity

Let us denote $n^{(k)}$ the number of neurons in the kth layer, L as the number of layers. Then the number of multiplications of weights is:

$$n_{mul} = \sum_{k=1}^{L} n^{(k)} n^{(k-1)} n^{(k-2)} + n^{(1)} n^{(0)}.$$

The number of activation function use is:

$$n_g = \sum_{k=1}^{L} n^{(k)}.$$

Then, we assume that weight matrices are square (the number of neurons is the same in each layer), and $L \sim n$:

$$n_{mul} = Ln^3 \sim n^4, \quad n_g = Ln \sim n^2.$$

Therefore, the computational complexity is: $n_{mul} + n_g \sim O(n^4) + O(n^2) \Longleftrightarrow O(n^4)$.

Hyperparameter γ is chosen via K-fold cross-validation

$$\arg\min_{\boldsymbol{\lambda},\boldsymbol{\theta}} \sum_{i=1}^{N} \left[u_i - u_{\mathsf{NN}}(x_i, t_i; \boldsymbol{\theta})\right]^2 \; + \; \gamma \sum_{i=1}^{N} \left[f_{\mathsf{NN}}(x_i, t_i; \boldsymbol{\lambda}, \boldsymbol{\theta})\right]^2$$

Parameter γ controls the balance between matching the data and satisfying the model constraint.

Hyperparameter γ is chosen via K-fold cross-validation

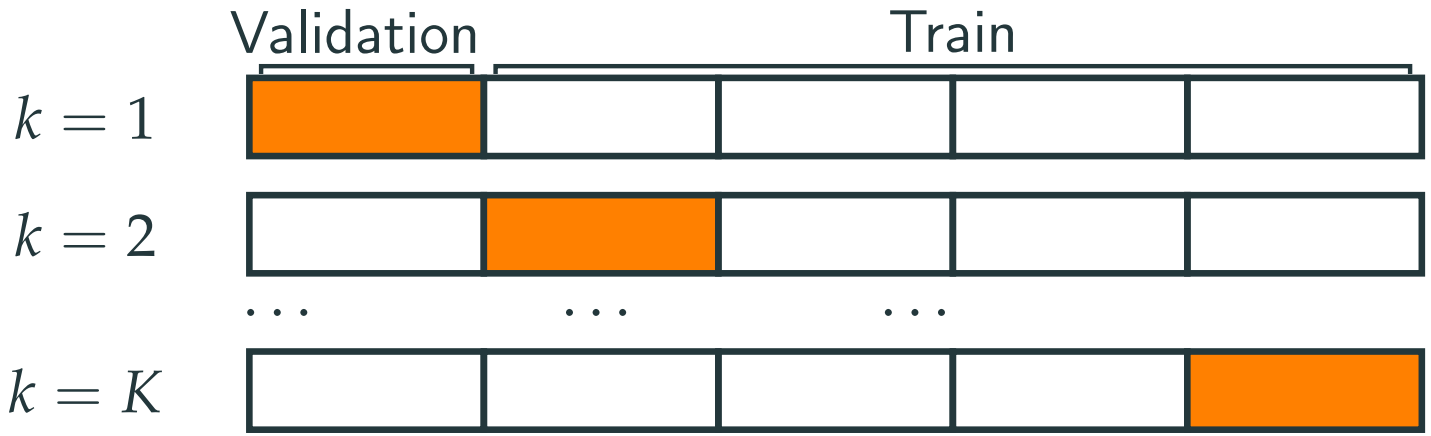
$$\arg\min_{\boldsymbol{\lambda},\boldsymbol{\theta}} \sum_{i=1}^{N} \left[u_i - u_{\mathsf{NN}}(x_i, t_i; \boldsymbol{\theta})\right]^2 + \gamma \sum_{i=1}^{N} \left[f_{\mathsf{NN}}(x_i, t_i; \boldsymbol{\lambda}, \boldsymbol{\theta})\right]^2$$

Parameter γ controls the balance between matching the data and satisfying the model constraint.

How to choose the best γ?

Discretize $\gamma \in \mathbb{R}$ in some range and choose the optimal value through the K-fold cross-validation.

Hyperparameter γ is chosen via K-fold cross-validation



For given γ in some range:

1. Split the data in $K = 5$ folds
2. Repeat K times:
   - train the neural network on "white" folds
   - compute prediction error on "orange" fold
3. Average the errors over K folds

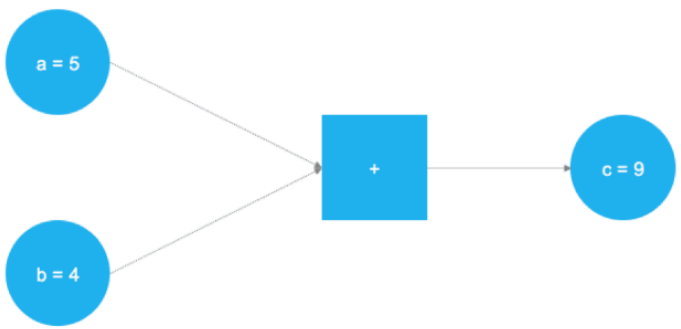
In the end: choose γ with the minimal averaged prediction error.

Estimate uncertainty of λ via bootstrap

- After estimating model parameter $\boldsymbol{\lambda}$, we approximate uncertainty of $\boldsymbol{\lambda}$ via bootstrap technique.
- By sampling from observations with replacement multiple times, we estimate the distribution of $\boldsymbol{\lambda}$ and 95 % percentile confidence interval.
- We use samples of size $0.8N$.
- We sample 100 times.

We use Tensorflow for all computations

- You specify functions as computational graphs
- It allows to compute derivatives automatically
- Can compute on videocards (it is faster than usual processors)
- Very useful for neural networks



Example 1. Heat equation

Heat equation

We consider the linear heat equation

$$u_t - \lambda u_{xx} - g(x,t) = 0, \quad x \in [-1;1], \quad t \in [0,1]$$
$$u(x,0) = 0$$
$$u(0,t) = u(1,t) = 0$$

with source term $g(x,t) = (4\pi^2 - 1)\,\sin(2\pi x)\,\exp(-t)$

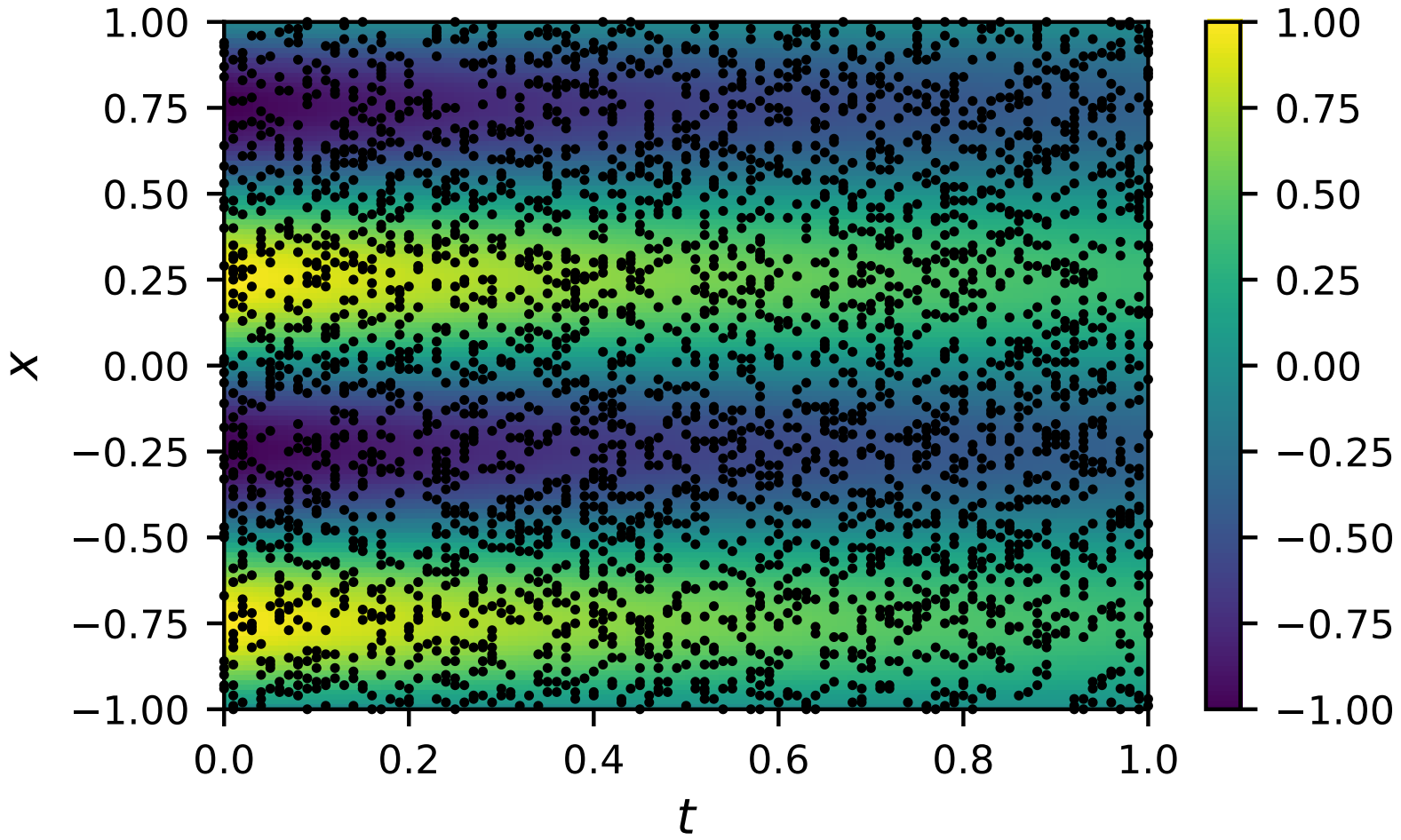
True λ is 1

Exact solution: $u = \sin(2\pi x)\,\exp(-t)$

Observations

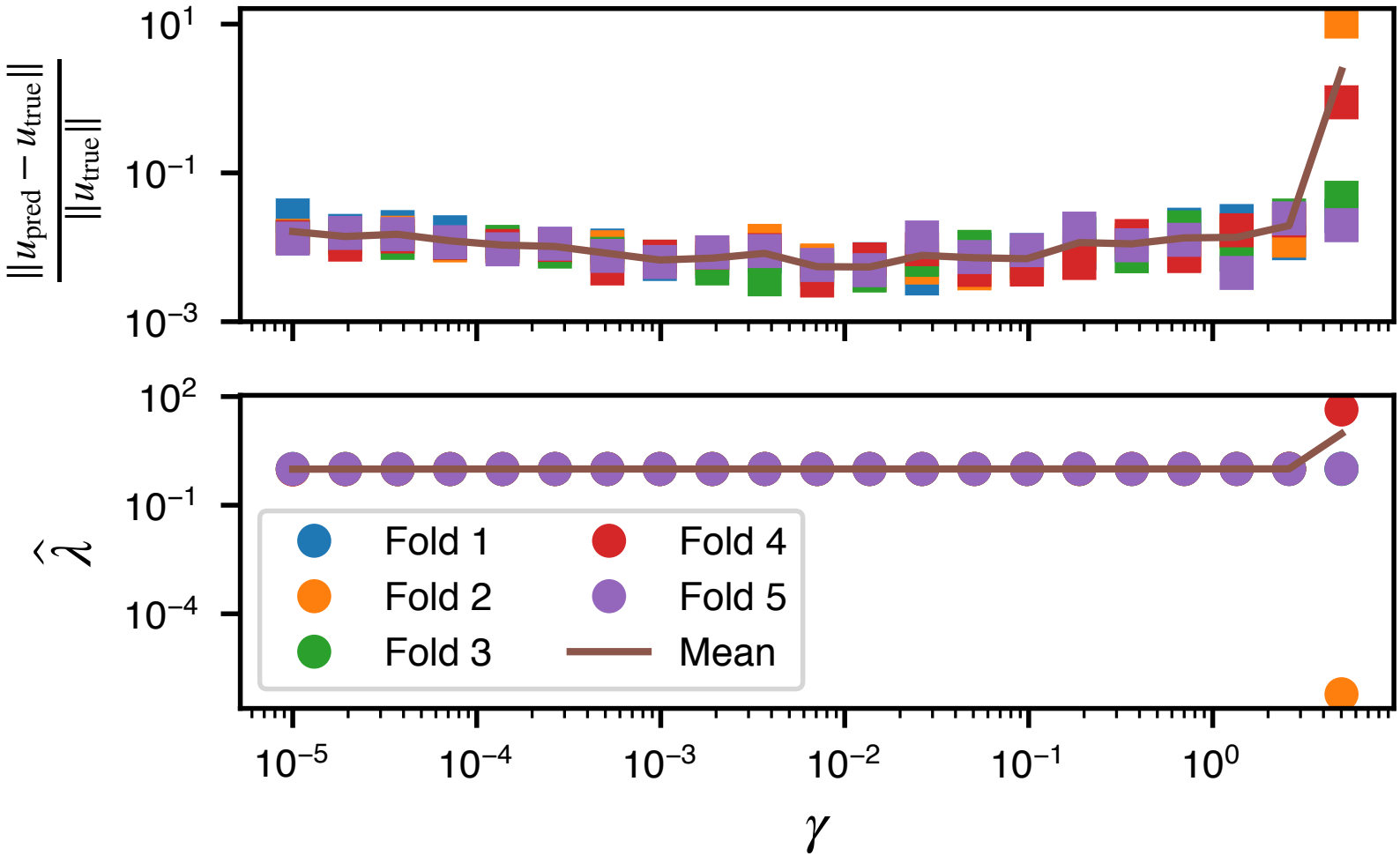
1. Compute u on the uniform grid with 201×101 points in $x \times t$
2. Network complexity [2, 20, 20, 1]: 2 hidden layers with 20 neurons in each
3. In total $2 \times 20 + 20 + 20 \times 20 + 20 + 20 \times 1 + 1 = 501$ parameters in $\boldsymbol{\theta}$
4. Sample 6400 observations uniformly
5. Large number of observations allows to avoid overfitting

Observations



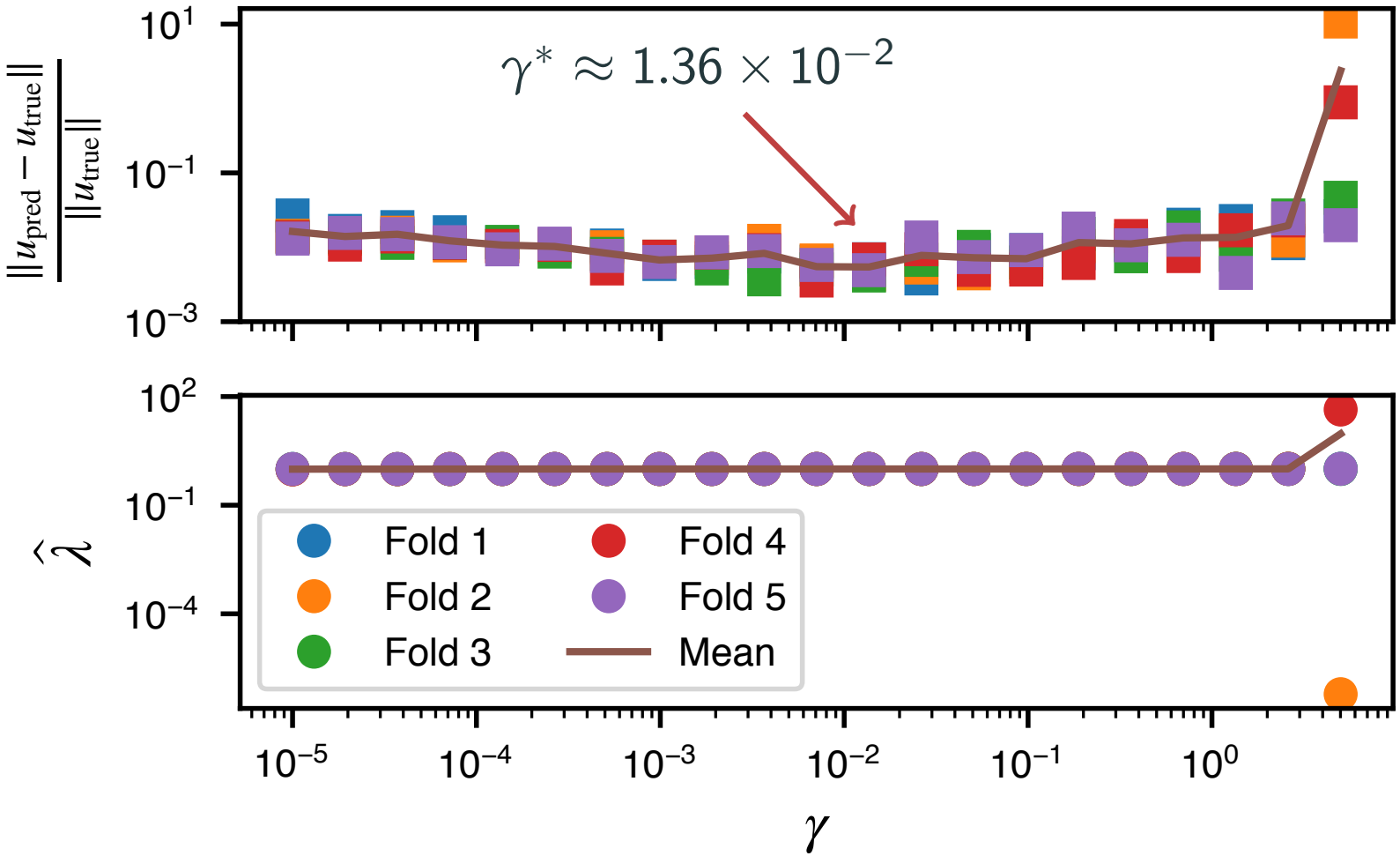
Cross-validation for γ

Clean data: just exact solution



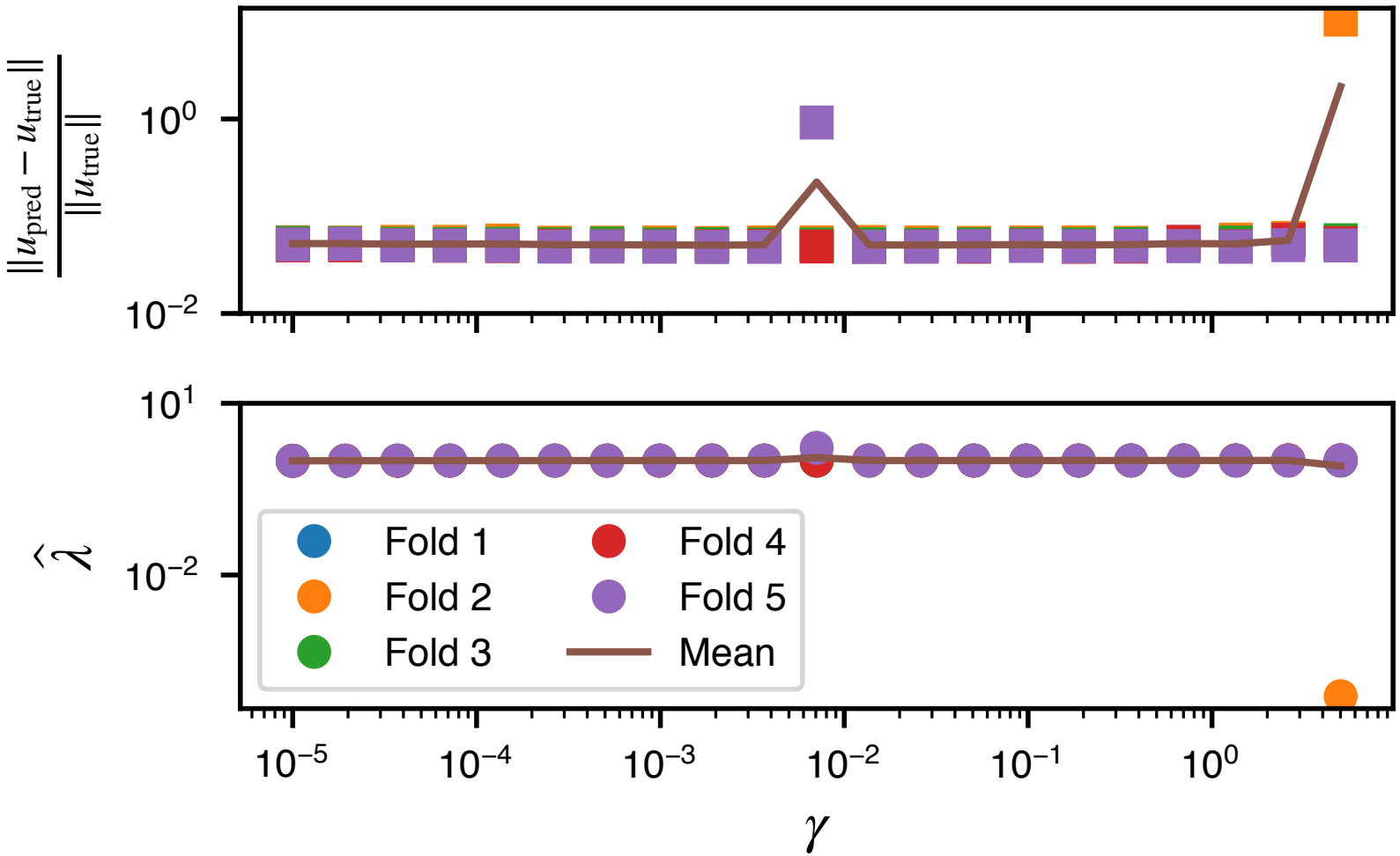
Cross-validation for γ

Clean data: just exact solution



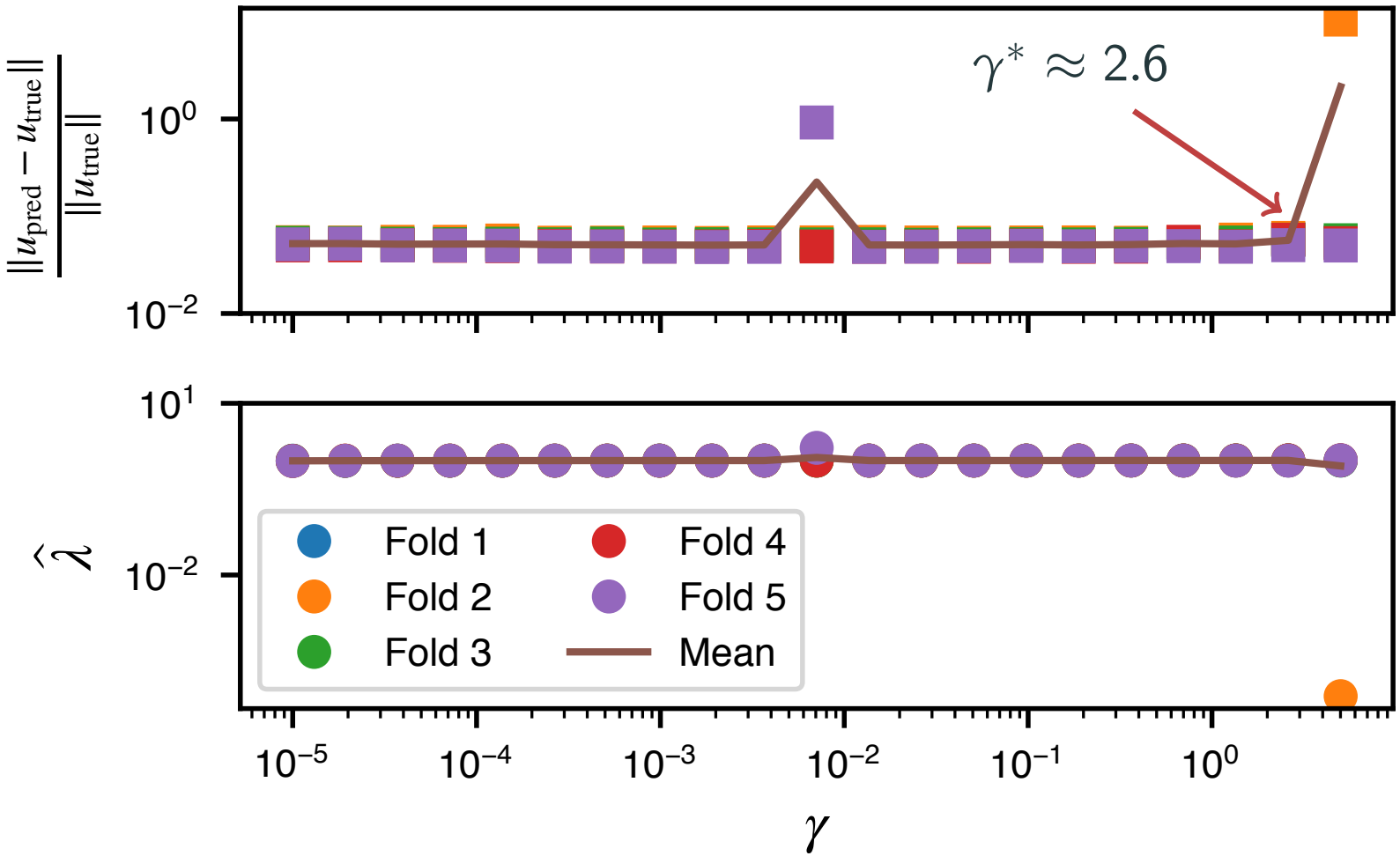
Cross-validation for γ, noisy observations

Noisy data: $u_{\text{obs}} = u + N(0, \sigma^2), \quad \sigma = 0.05u$



Cross-validation for γ, noisy observations

Noisy data: $u_{\text{obs}} = u + N(0, \sigma^2), \quad \sigma = 0.05u$



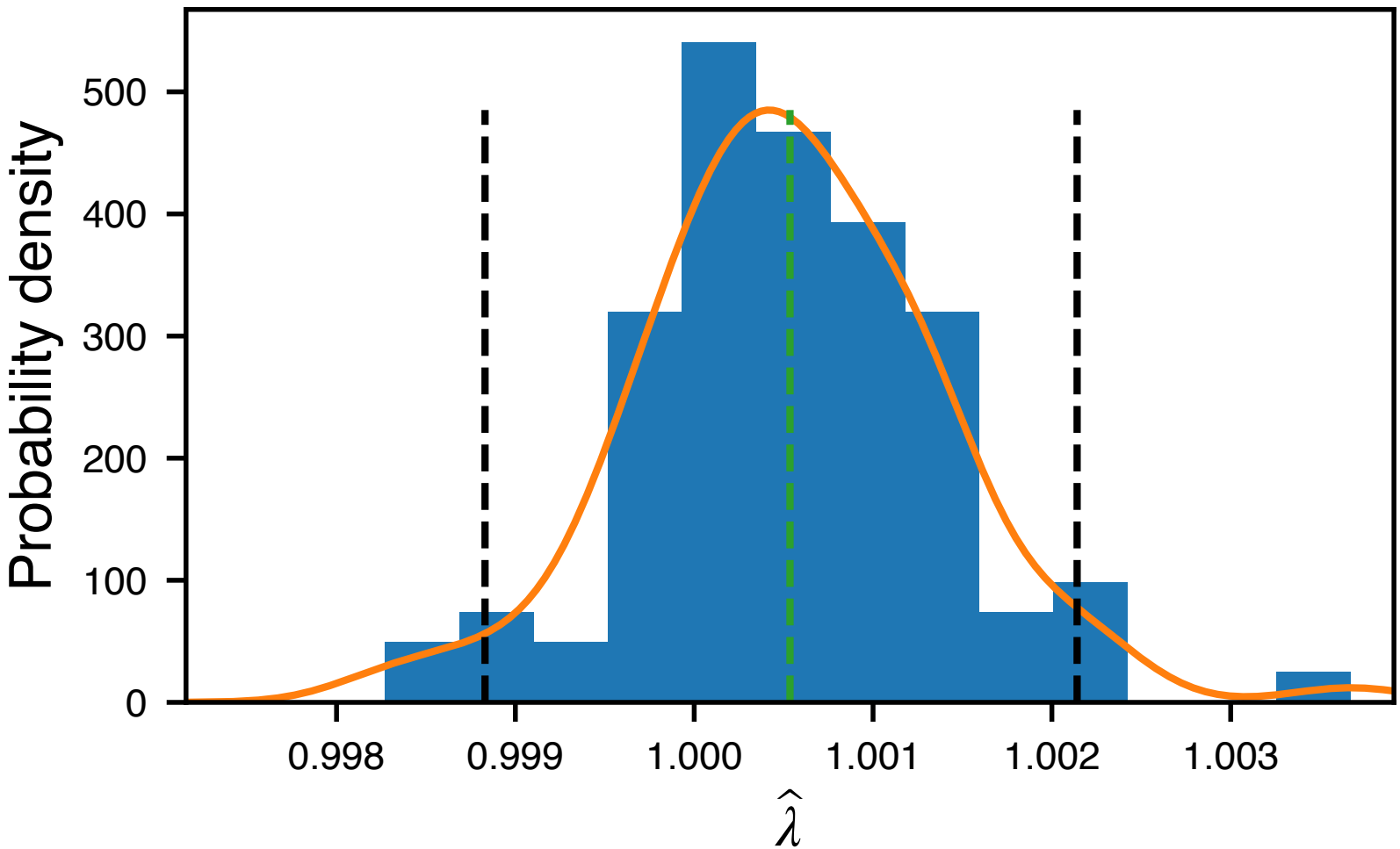
Uncertainty on $\widehat{\lambda}$ via bootstrap

- Both for clean and noisy data with corresponding optimal γ
- Simulate distribution of $\widehat{\lambda}$ via 100 values
- Detect and remove outliers via interquartile range method
- Compute bootstrap percentile intervals

Uncertainty on $\widehat{\lambda}$ via bootstrap, clean data

Clean data: just exact solution

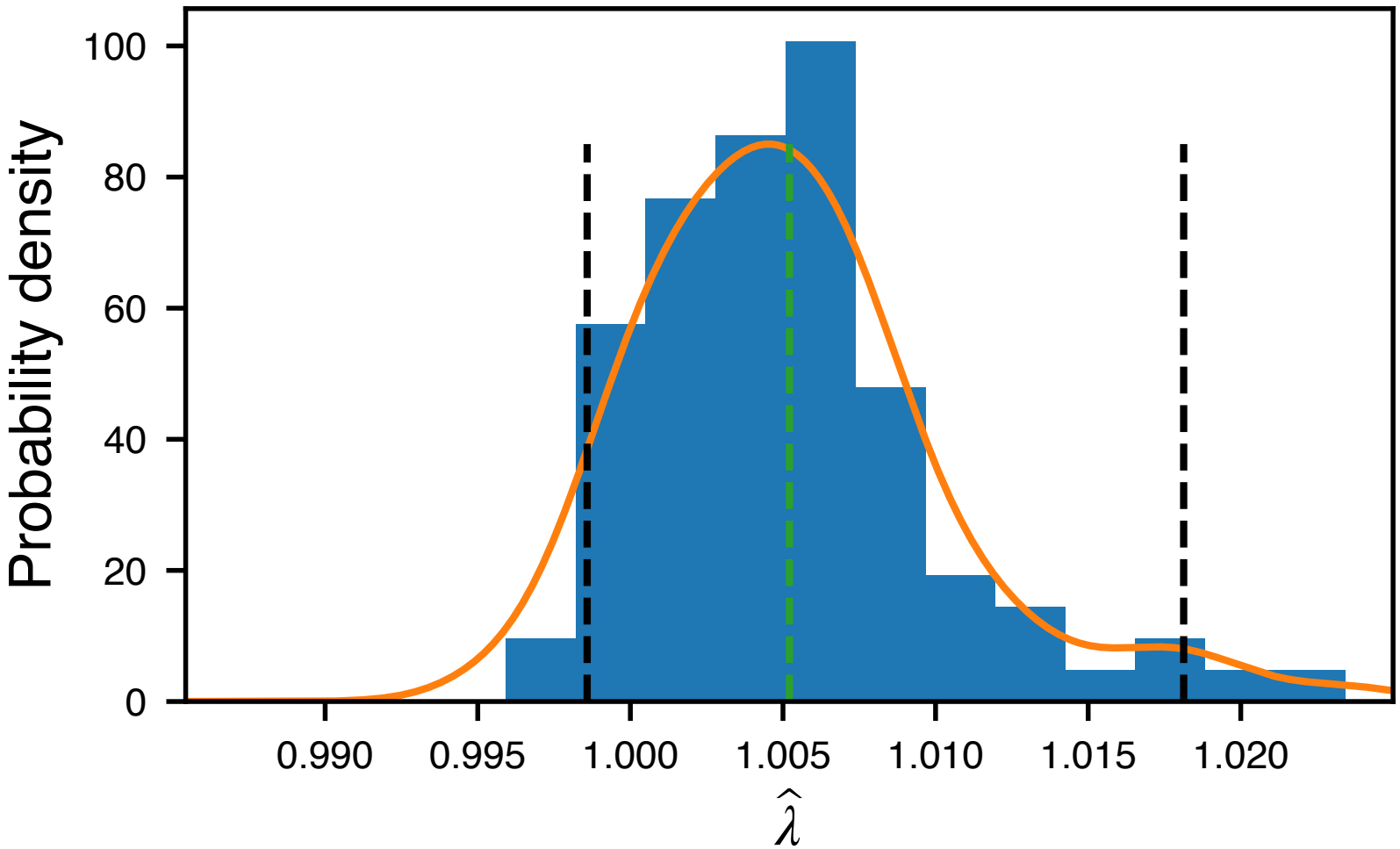
Mean $\widehat{\lambda} \approx 1.0005$



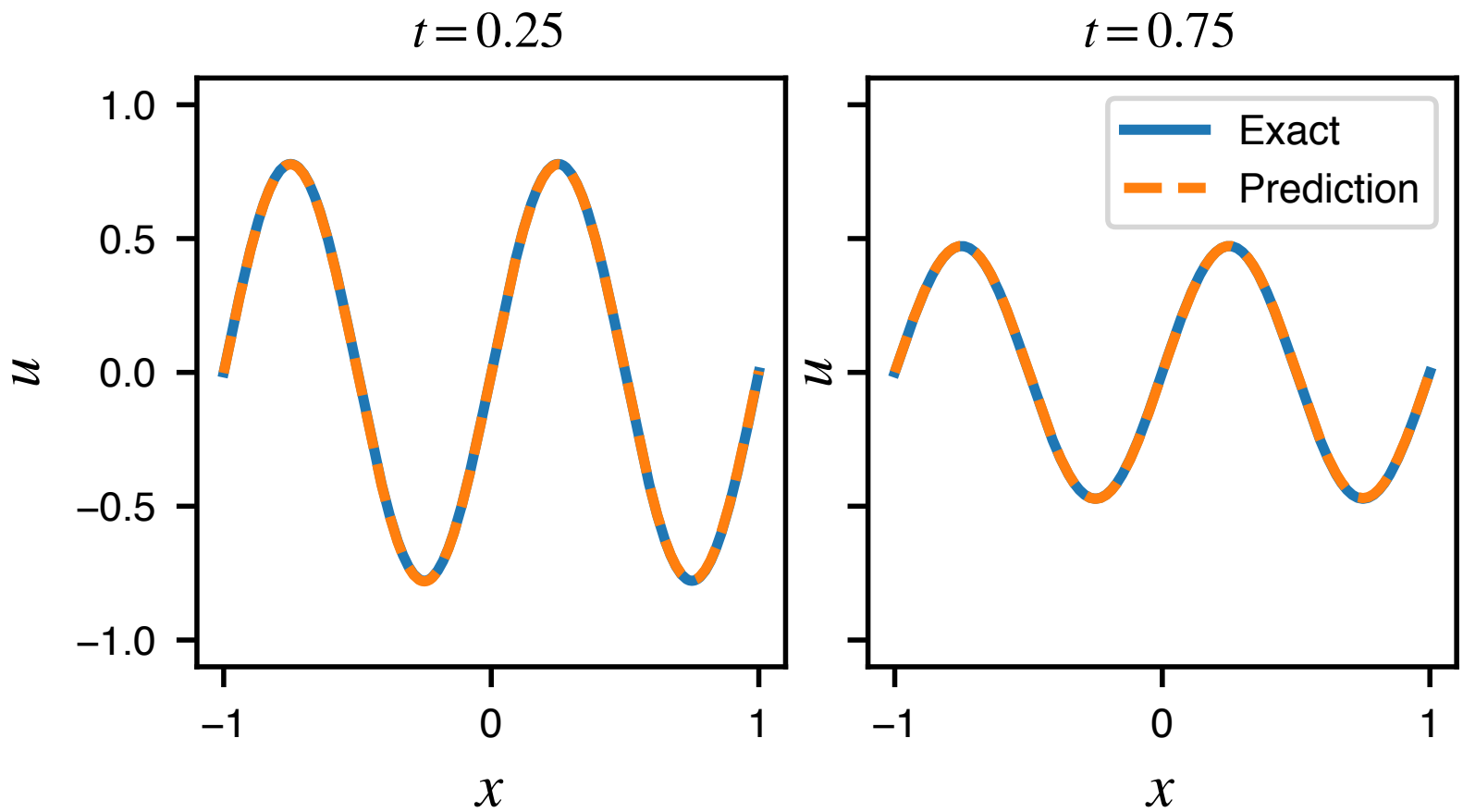
Uncertainty on $\widehat{\lambda}$ via bootstrap, noisy data

Noisy data: $u_{\text{obs}} = u + N(0, \sigma^2), \quad \sigma = 0.05u$

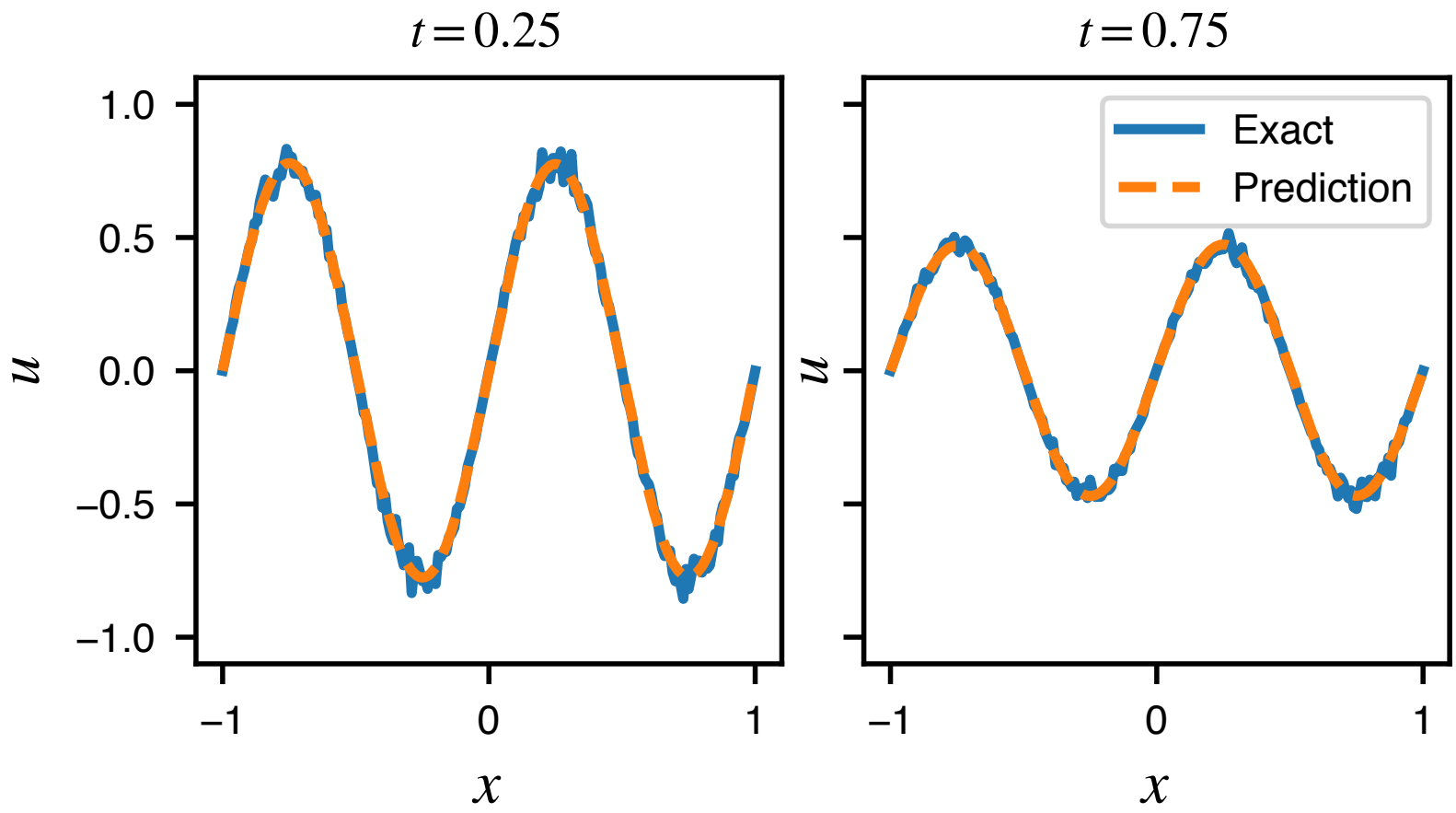
Mean $\widehat{\lambda} \approx 1.0052$ (10x larger error than for clean data)



Predictions, clean data



Predictions, noisy data



Example 2. Burgers' equation

Burgers' equation

We consider the input data of the Burgers' equation:

$$
\begin{aligned}
&u_t + \lambda_1 u u_x - \lambda_2 u_{xx} = 0, \quad x \in [-1, 1], \quad t \in [0, T] \\
&u(0, x) = -\sin(\pi x) \\
&u(t, -1) = u(t, 1) = 0
\end{aligned}
$$

with true values of the sought-for parameters

$$
\lambda_1 = 1, \quad \lambda_2 = (0.01/\pi) \approx 0.00318.
$$

The exact solution is

$$
u(t, x) = \frac{\pi \lambda_2}{\pi \lambda_2 e^{\{\pi^2 \lambda_2 t\}} - (e^{\{\pi^2 \lambda_2 t\}} - 1) cos(\pi x)}
$$

Burgers' equation neural network configuration

1. The following network configuration was used: $[2, 10, 10, 10, 10, 1]$.
2. The number of the parameters in $\boldsymbol{\theta}$ is: $2 * 10 + (10 * 10 + 10) * 3 + 10 * 1 + 1 = 370$
3. 3700 observations were used.

Observations

Below, the exact solution is depicted on the uniform grid with 201 points in x and 101 points in t.

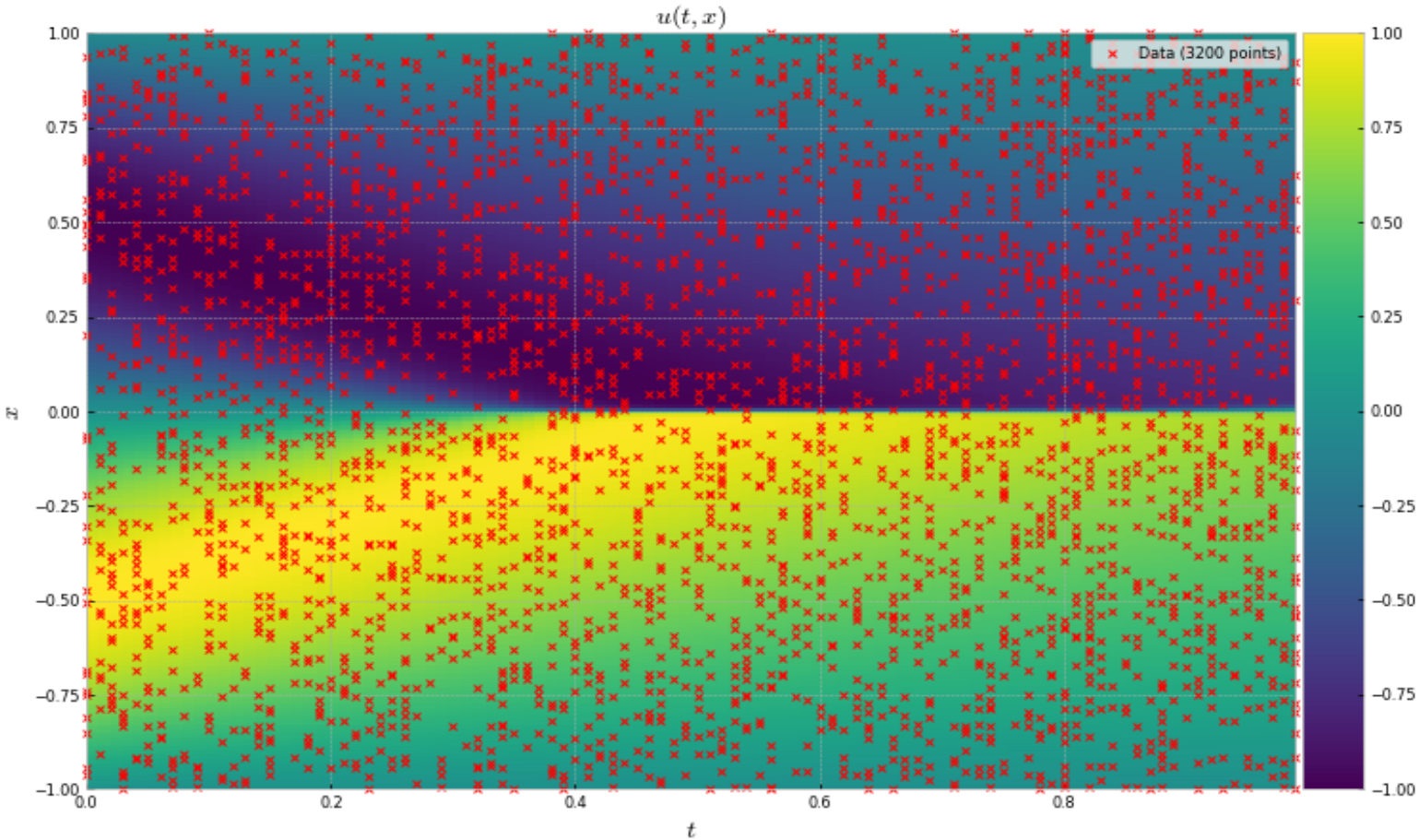
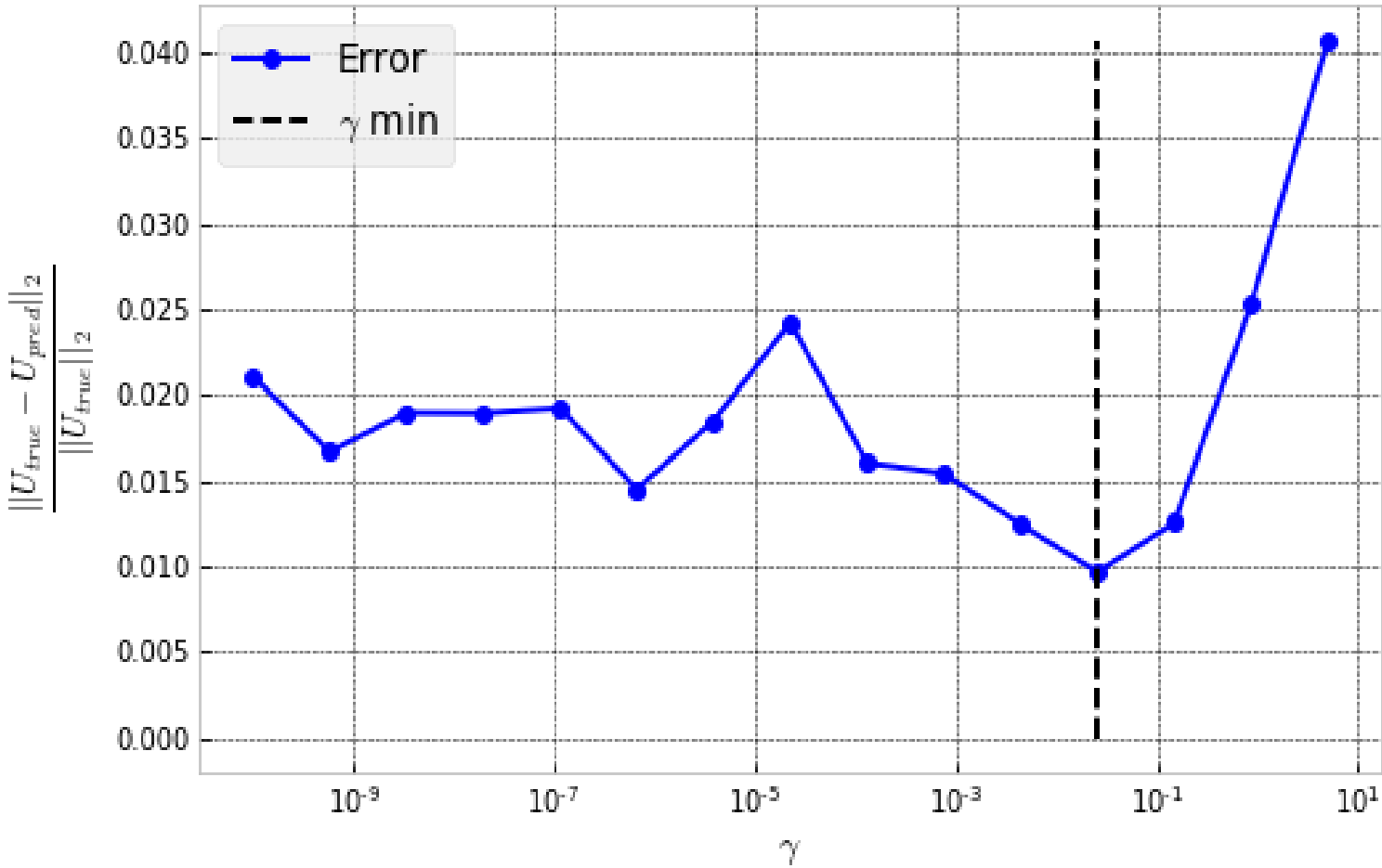


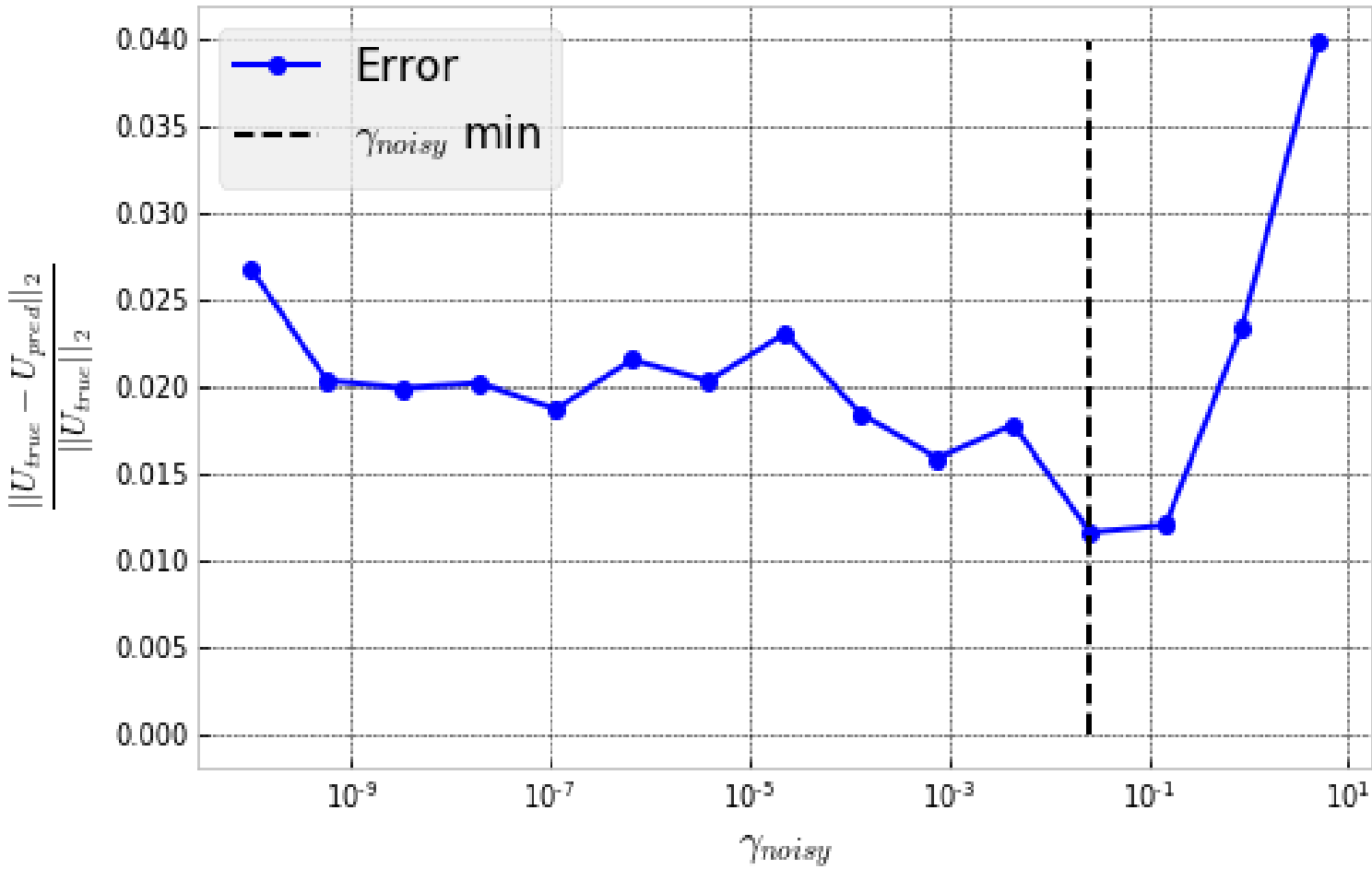
Figure 1: The exact solution of Burgers' equation and 3200 observations.

Hyperparameter γ for the optimization problem was computed via cross-validation technique, using 5 folds.



Cross-validation result for the hyperparameter $\gamma = 0.0255$.

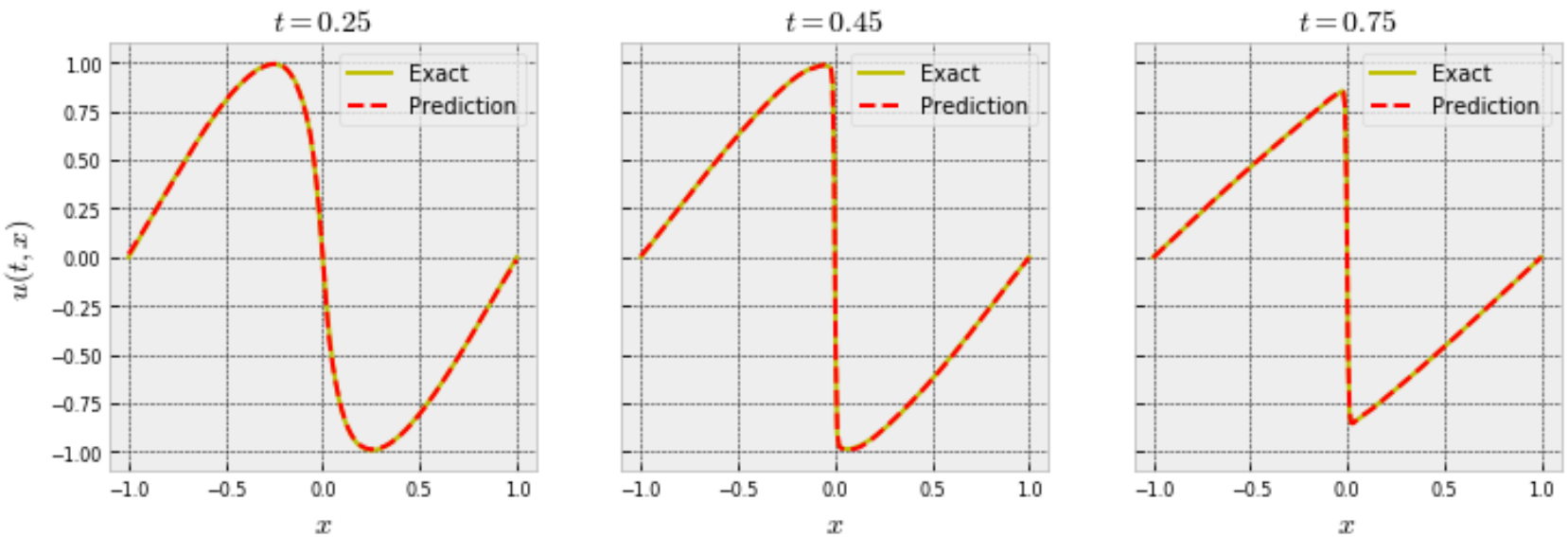
Parameter γ for noisy data: $u_{obs} = u + N(0, \sigma^2)$,



Cross-validation result for the hyperparameter $\gamma = 0.0255$ on the noisy data.

Clean data results $\gamma = 0.02553$

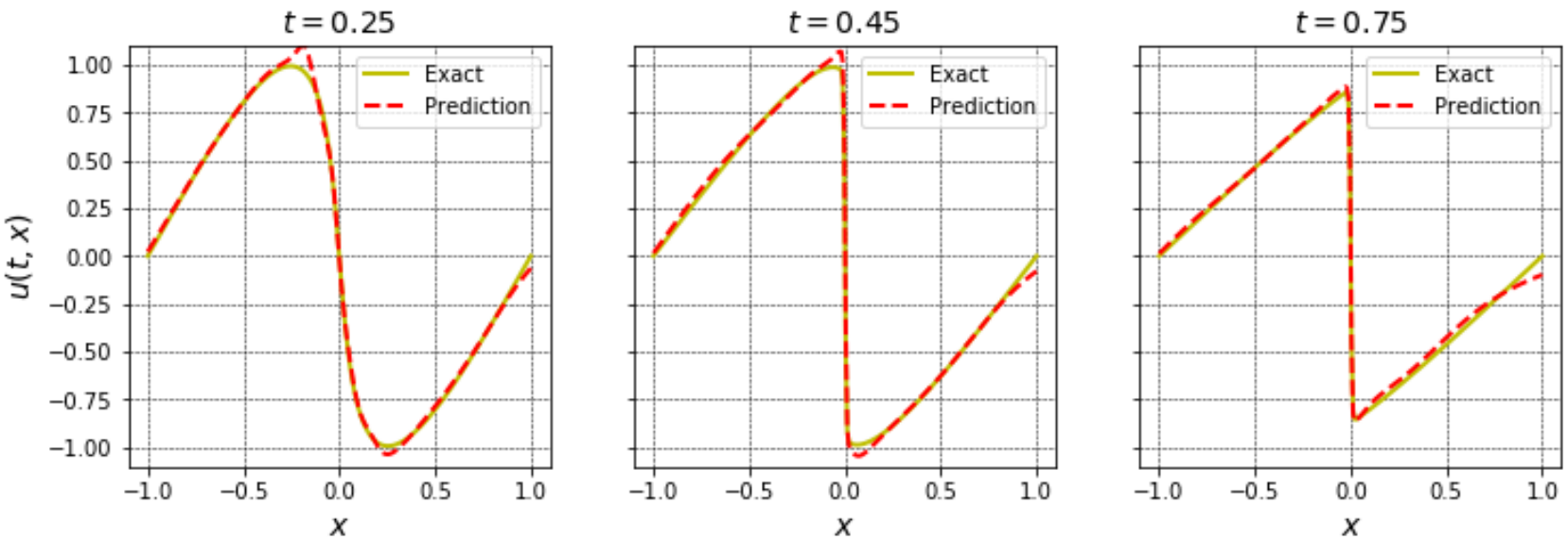
Comparison between the exact and the predicted solutions at the different time instants t = 0.25, 0.50, 0.75. Training the network with $\gamma = 0.0255$ gives prediction $\lambda_1 = 0.986564,\ \lambda_2 = 0.00338$ with MSE $u(x,t) \approx 0.011\,\%$.



Noisy data results

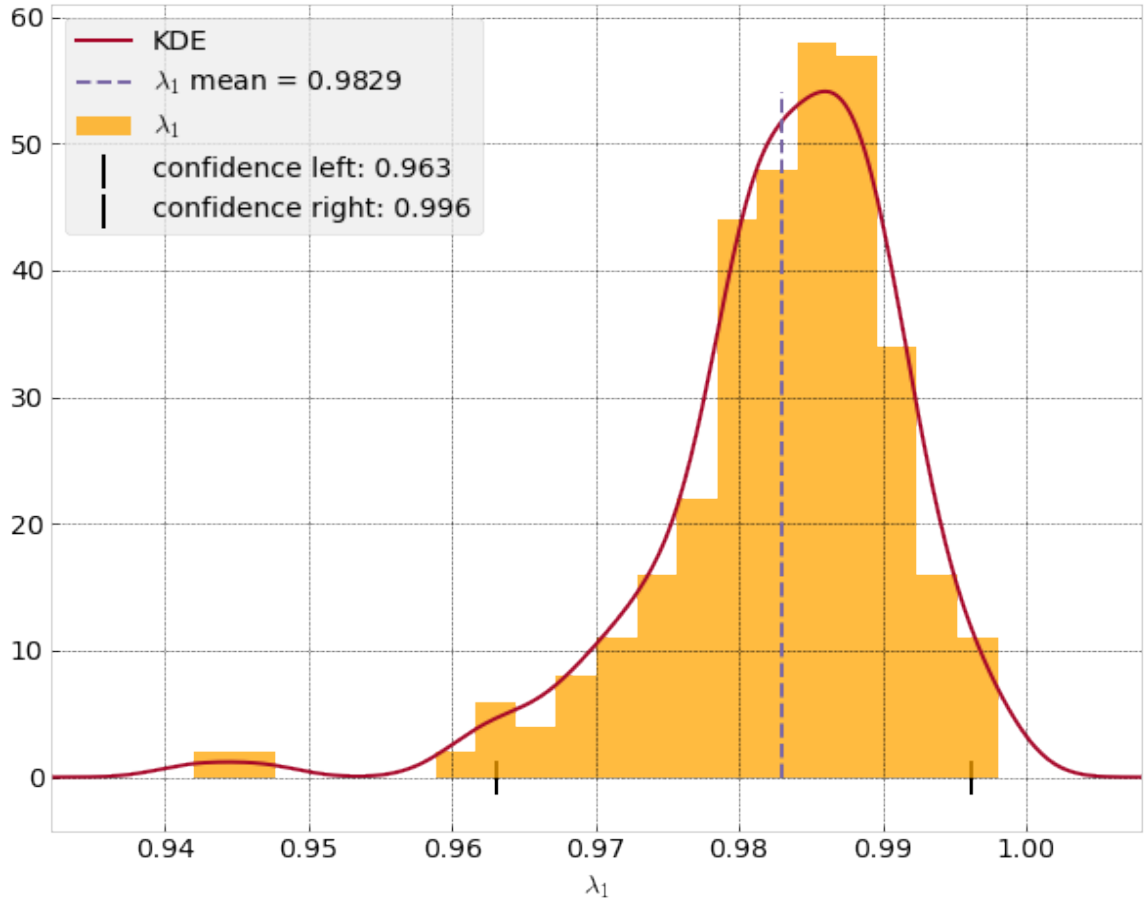
Comparison between the exact and the predicted solutions for the noisy data.

$$u_{\text{obs}} = u + N(0, \sigma^2)$$

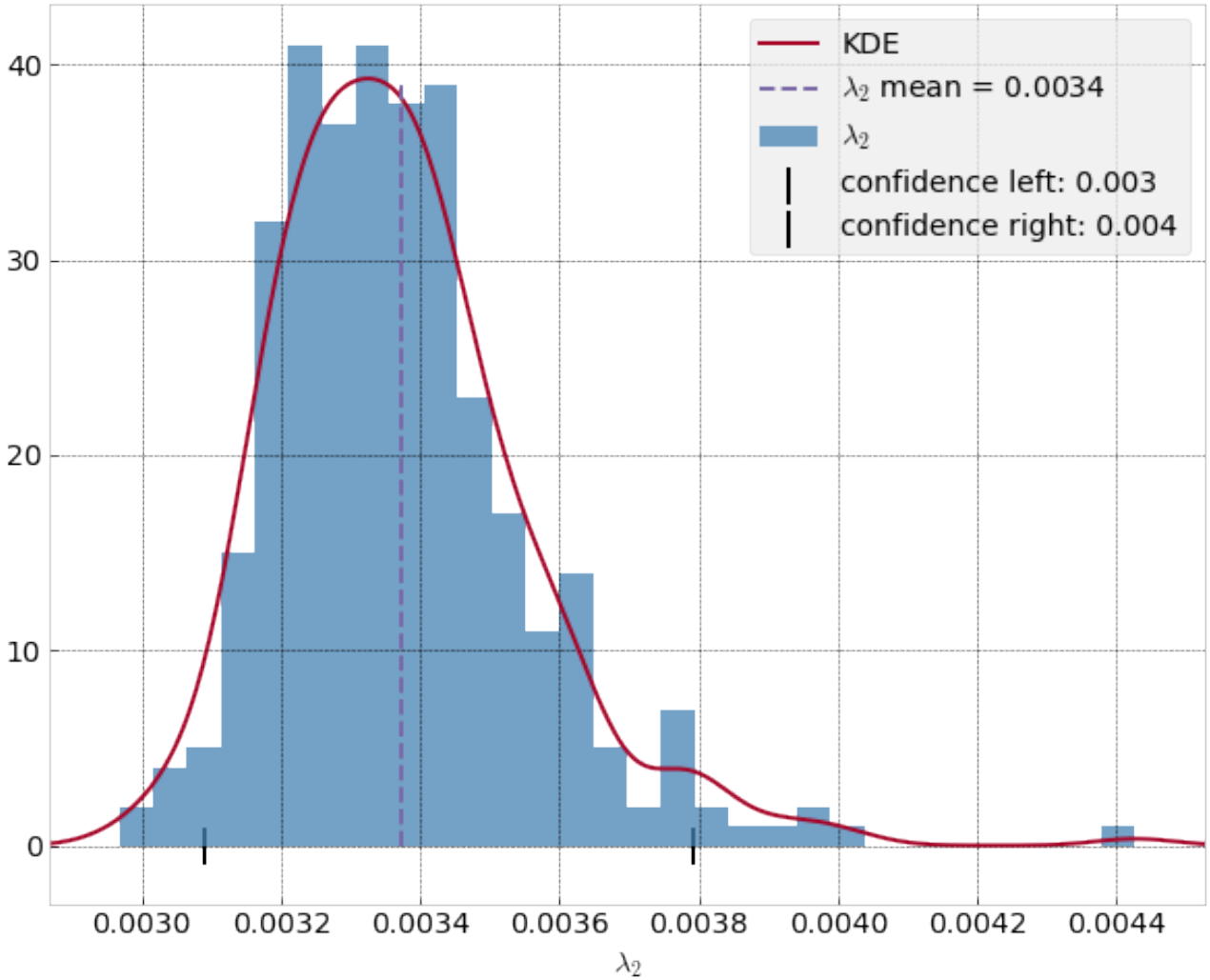


Solution error $\frac{||U_{pred} - U_{true}||_2}{||U_{true}||_2}$ is 0.049.

We have got the following bootstrapped results:



Bootstrapped parameter λ_1 for Burgers' equation.



Bootstrapped parameter λ_2 for Burgers' equation.

Neural Network configuration vs λ error

Below one can see the % error in λ_1 and λ_2 for different neural network configurations. In the first column the number of the hidden layers is presented, in the second row the number of the neurons in each hidden layer is presented.

	% error in λ_1			% error in λ_2		
Layers \ Neurons	5	10	20	5	10	20
1	19.43	31.82	25.32	118.02	79.2	74.33
2	24.93	8.75	3.21	56.05	28.35	23.48
4	3.42	0.82	0.29	13.56	2.43	4.21
6	3.26	2.16	1.29	19.53	2.79	0.66
8	6.17	1.41	2.83	6.57	1.26	1.28

Neural Network vs predicted solution error

Below one can see the % error in solution for different neural network configurations.

	% error in solution $u(x,t)$		
Layers \ Neurons	5	10	20
1	19.16	15.98	13.45
2	6.95	2.95	2.12
4	1.53	1.05	0.74
6	2.05	1.04	0.56
8	2.61	0.84	0.91

From the results above one can see that the optimal configuration of the model is: 6 hidden layers with 20 neurons in each.

Conclusions

- We apply neural networks to problem of parameter estimation.
- Adding physical constraint leads to efficient network training.
- Due to physical constraint, small number of observations works (small from neural-network point of view).
- Accurate estimates are reached with clean and noisy data.
- Parameter distributions look similar to normal distribution.

Important references

- Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2017). *Physics informed deep learning (part II): Data-driven discovery of nonlinear partial differential equations*.
  arXiv:1711.10566

- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.
  `http://www.deeplearningbook.org/`

- Hagan M. et al. (2014). *Neural network design*. 2nd ed.
  `http://hagan.okstate.edu/nnd.html`

Thank you!