Modelling, Uncertainty and Data for Engineers (MUDE)

Machine Learning

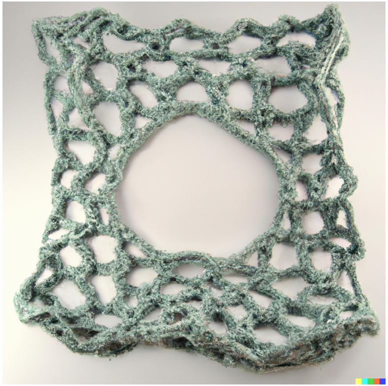
Iuri Rocha, Riccardo Taormina

and the SLIMM lab and Aidrolab PhDs

TU Delft

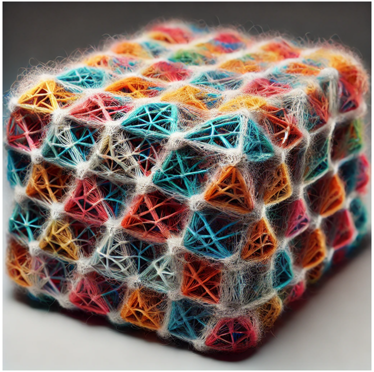
A long, long, long time ago (2 years)

A Finite Element model knitted out of wool



Two teddy bears discovering a new metamaterial

Fairly long ago (2 days)



Here is the creative depiction of a FEM mesh knitted out of wool! It combines the structural precision of FEM with the softness and humor of knitted material, making it an engaging and fun way to teach

Here is a delightful image of two teddy bears hard at work constructing a new metamaterial in a whimsical lab! Their tiny lab coats and safety goggles add an adorable touch while blending humor with a scientific theme.

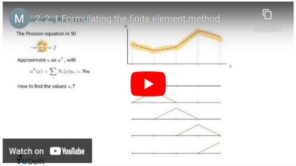
TUDelft

Nowadays machine learning can do a lot more

2.1. Strong form of the 1D Poisson equation

The *strong from* of the problem is a Partial Differential Equation (PDE), or a set of PDEs, describing the physics of the phenomena at hand, supplemented with appropriate boundary and initial conditions. Solving the strong form of the problem leads to the exact solution of the continuous system. However, for many engineering problems, it is not possible to obtain the exact solution, and numerical methods need to be employed to find an approximate solution. The finite element method is one approach to find approximate solutions that is applicable to a wide range of different problems. In this chapter we will focus on problems for which the strong form is the Poisson equation.

The story told below and up to the discrete form section is also presented in this video:



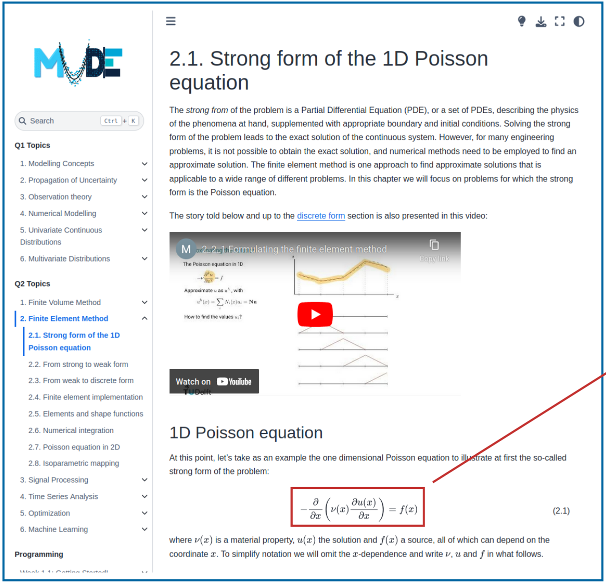
1D Poisson equation

At this point, let's take as an example the one dimensional Poisson equation to illustrate at first the so-called strong form of the problem:

$$-\frac{\partial}{\partial x}\left(\nu(x)\frac{\partial u(x)}{\partial x}\right) = f(x) \tag{2.1}$$

where $\nu(x)$ is a material property, $u(x)$ the solution and $f(x)$ a source, all of which can depend on the coordinate x. To simplify notation we will omit the x-dependence and write ν, u and f in what follows.

Nowadays machine learning can do a lot more



$$-\frac{\partial}{\partial x}\left(\nu(x)\frac{\partial u(x)}{\partial x}\right) = f(x)$$

(screenshot)

Nowadays machine learning can do a lot more

$$-\frac{\partial}{\partial x}\left(\nu(x)\frac{\partial u(x)}{\partial x}\right) = f(x) \tag{2.1}$$

What is this equation?

This equation is a one-dimensional **diffusion equation** with a spatially varying diffusion coefficient $\nu(x)$, often encountered in physics and engineering to describe processes like heat conduction, mass transport, or flow in porous media.

Components:

1. $\nu(x)$:
   - A spatially dependent coefficient, representing the diffusivity or conductivity of the medium.
2. $u(x)$:
   - The variable of interest, such as temperature, concentration, or velocity field.
3. $f(x)$:
   - A source term, representing any external inputs or sinks in the system.
4. $-\frac{\partial}{\partial x}\left(\nu(x)\frac{\partial u(x)}{\partial x}\right)$:
   - Represents the flux due to diffusion and the spatial variation of diffusivity.

Nowadays machine learning can do a lot more

$$-\frac{\partial}{\partial x}\left(\nu(x)\frac{\partial u(x)}{\partial x}\right) = f(x) \qquad (2.1)$$

What is this equation?

This equation is a one-dimensional **diffusion equation** with a spatially varying diffusion coefficient $\nu(x)$, often encountered in physics and engineering to describe processes like heat conduction, mass transport, or flow in porous media.

Components:

1. $\nu(x)$:
   - A spatially dependent coefficient, representing the diffusivity or conductivity of the medium.
2. $u(x)$:
   - The variable of interest, such as temperature, concentration, or velocity field.
3. $f(x)$:
   - A source term, representing any external inputs or sinks in the system.
4. $-\frac{\partial}{\partial x}\left(\nu(x)\frac{\partial u(x)}{\partial x}\right)$:
   - Represents the flux due to diffusion and the spatial variation of diffusivity.

ChatGPT 4o

This form is particularly useful when the material properties vary with position, requiring a spatially dependent $\nu(x)$.

Can you write me a simple python code to solve this equation with FEM in 1D?

Below is a simple Python code to solve the given 1D diffusion equation using the **Finite Element Method (FEM)**. The example assumes Dirichlet boundary conditions for simplicity.

Fem 1d Diffusion

Answer in chat instead >

This code solves the equation $-\frac{d}{dx}\left(\nu(x)\frac{du}{dx}\right) = f(x)$ using a simple FEM approach in 1D. The solution is plotted at the end. You can modify $\nu(x)$ and $f(x)$ to represent more complex problems. Let me know if you need explanations or enhancements!

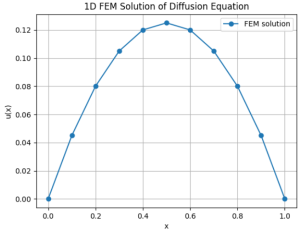
Message ChatGPT

Fem 1d Diffusion

```python
        # Element stiffness matrix (2x2)
        ke = (nu(x[n1]) + nu(x[n2])) / 2 / h * np.array([[1, -1],
                                                        [-1, 1]])

        # Element load vector (2x1)
        fe = h / 2 * np.array([f(x[n1]), f(x[n2])])

        # Assemble into the global system
        K[n1:n2+1, n1:n2+1] += ke
        F[n1:n2+1] += fe

    # Apply Dirichlet boundary conditions (u(0) = 0, u(L) = 0)
    bc_nodes = [0, n_nodes - 1]  # Boundary nodes
    for bc_node in bc_nodes:
        K[bc_node, :] = 0
        K[bc_node, bc_node] = 1
        F[bc_node] = 0
```

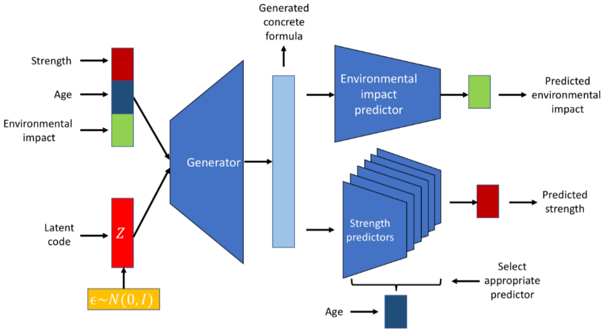
Console

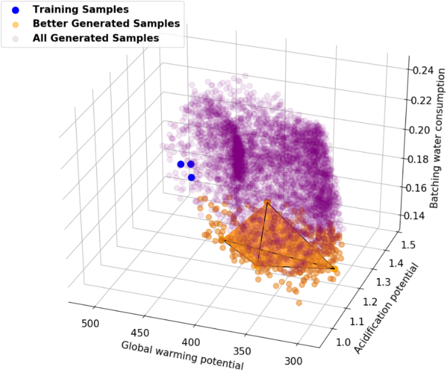
Run



It cannot come up with new ideas, but it can help you do it

Generating new, more sustainable concrete mixtures:



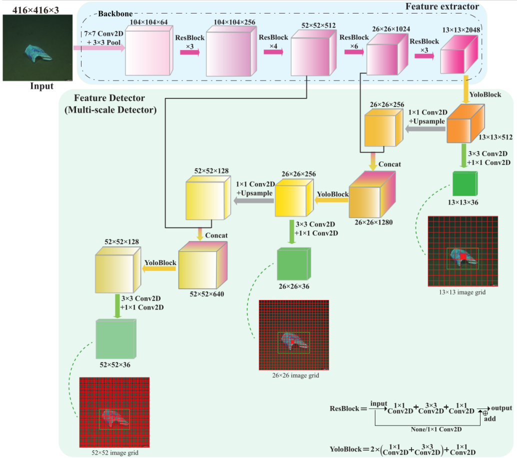


[Ge et al (2019), arXiv:1905.08222]

TU Delft

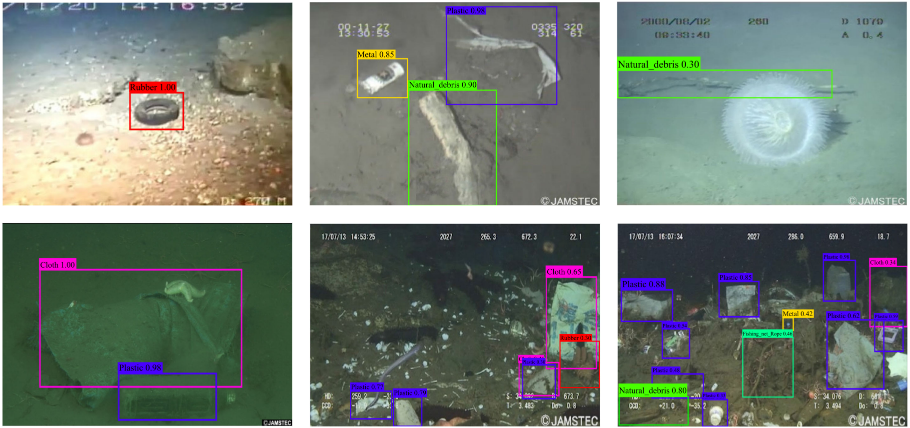
It cannot come up with new ideas, but it can help you do it

Detecting plastic in deep sea with autonomous underwater vehicles:



It cannot come up with new ideas, but it can help you do it

Detecting plastic in deep sea with autonomous underwater vehicles:

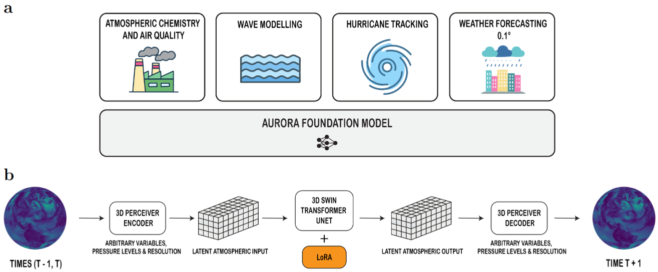


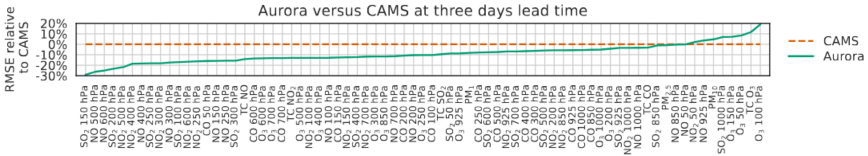
TU Delft

[Xue et al (2021), IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. **14**]

It cannot come up with new ideas, but it can help you do it

A 1.3 billion parameter foundation model for the Earth atmosphere and ocean waves:



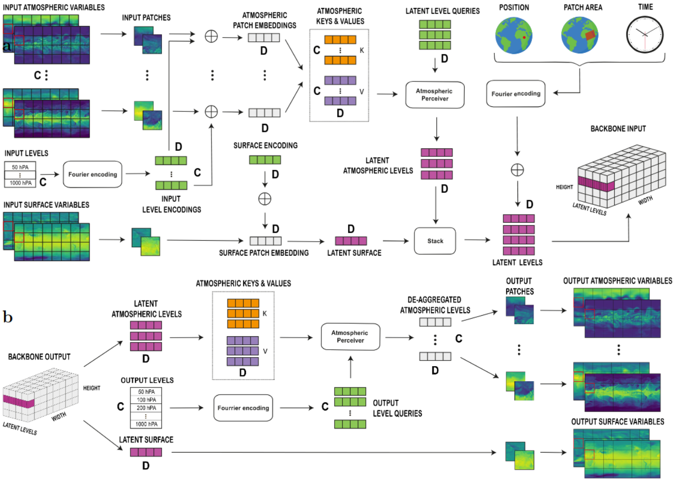


[Bodnar et al (2024), arXiv:2405.13063]

TU Delft

It cannot come up with new ideas, but it can help you do it

A 1.3 billion parameter foundation model for the Earth atmosphere and ocean waves:

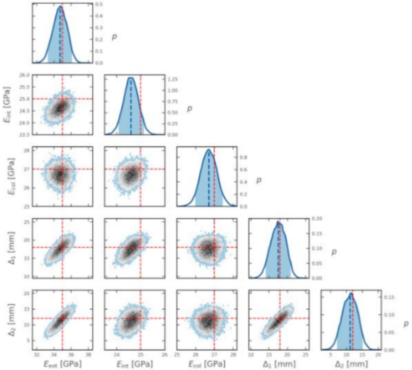


[Bodnar et al (2024), arXiv:2405.13063]

TUDelft

A few examples closer to home

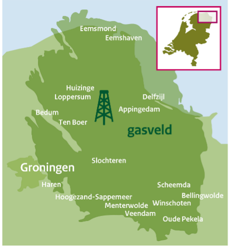
Inverse identification of bridge health through Bayesian machine learning:

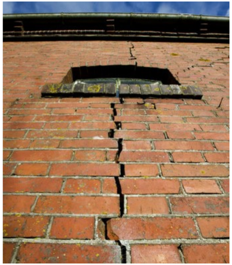


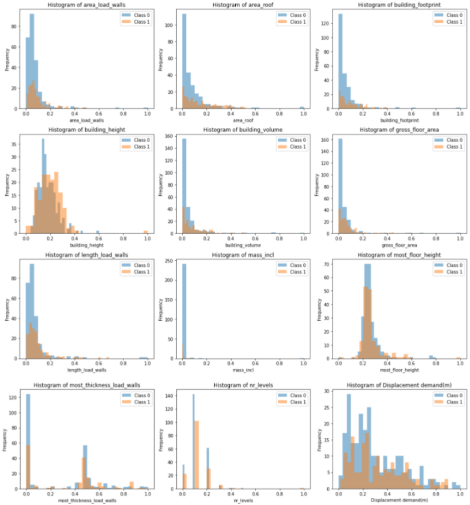
[Martínez Colán (2022)]

A few examples closer to home

Classifier model for predicting need for reinforcement against earthquakes in Groningen:





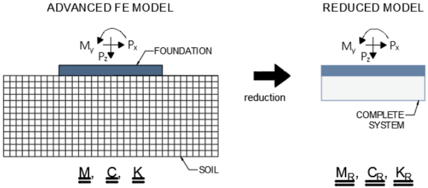


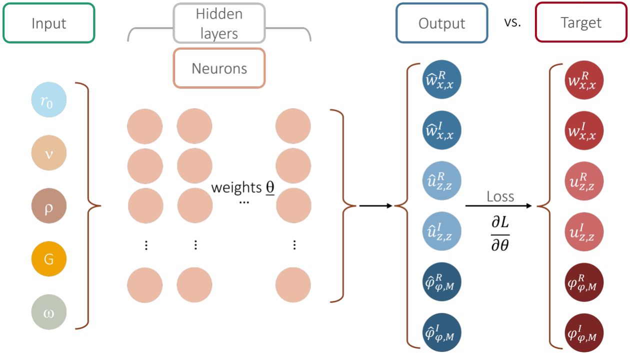
[Rithu Maria (2024)]

TU Delft

A few examples closer to home

Neural network-based mechanical analogues for soil-structure interaction:



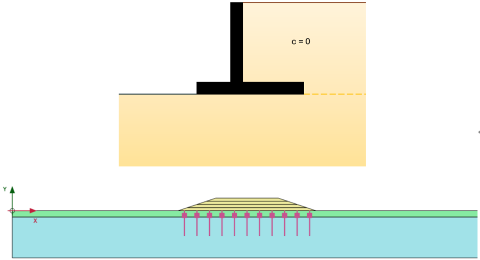


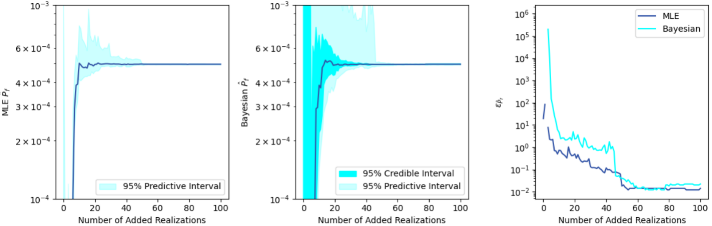
TU Delft

[Mariana Fuente (2024)]

A few examples closer to home

Bayesian machine learning for reliability analysis of geotechnical engineering problems:





[Ron Navarro (2024)]

Some definitions and terminology

Narrow versus General AI:

- Narrow AI can only perform one specific task $\Leftarrow$ ML techniques live here
- General AI can perform a multitude of tasks and program itself $\Leftarrow$ just a dream (for now)

Some definitions and terminology

Narrow versus General AI:

- Narrow AI can only perform one specific task $\Leftarrow$ ML techniques live here
- General AI can perform a multitude of tasks and program itself $\Leftarrow$ just a dream (for now)

Supervised Learning: Tasks with known target outcomes, requires labeled data:

- Regression: Map input features to noisy observations of continuous targets $\Leftarrow$ this course
- Classification: Map input features to discrete class labels

Some definitions and terminology

Narrow versus General AI:

- Narrow AI can only perform one specific task ⇐ ML techniques live here
- General AI can perform a multitude of tasks and program itself ⇐ just a dream (for now)

Supervised Learning: Tasks with known target outcomes, requires labeled data:

- Regression: Map input features to noisy observations of continuous targets ⇐ this course
- Classification: Map input features to discrete class labels

Unsupervised Learning: Explain patterns in unlabeled data with latent (hidden) variables:

- Clustering: Split data into groups explained by discrete latents
- Dimensionality reduction: Explain the data with a manifold described by continuous latents

Some definitions and terminology

Narrow versus General AI:

- Narrow AI can only perform one specific task $\Leftarrow$ ML techniques live here
- General AI can perform a multitude of tasks and program itself $\Leftarrow$ just a dream (for now)

Supervised Learning: Tasks with known target outcomes, requires labeled data:

- Regression: Map input features to noisy observations of continuous targets $\Leftarrow$ this course
- Classification: Map input features to discrete class labels

Unsupervised Learning: Explain patterns in unlabeled data with latent (hidden) variables:

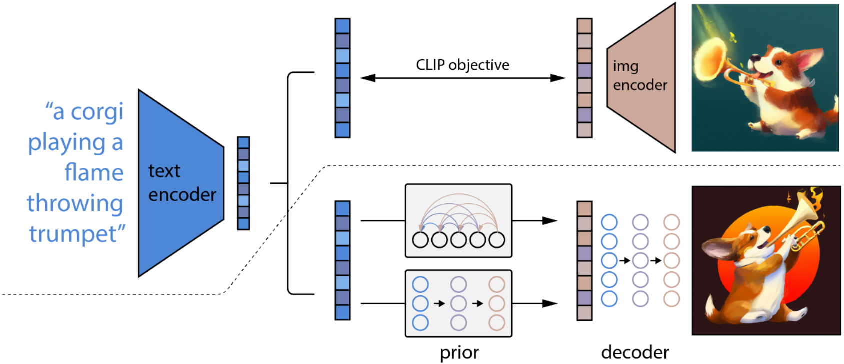
- Clustering: Split data into groups explained by discrete latents
- Dimensionality reduction: Explain the data with a manifold described by continuous latents

Reinforcement Learning: Learn a task through reward/punishment mechanisms:

- Agent(s) interacting with an environment, evolving interaction policy

Finding patterns and making good use of them

The core goal of ML is not fitting data, but finding useful representations and exploiting them



[Ramesh et al (2022), arXiv:2204.06125]

Week 2.6 — Theory

You can find all the material in the book, as always:

- Pages for each important concept, starting from scratch
- It is beneficial to go through them in order

Interactive plots:

- You will play with some interactive plots today
- Keep going in the book, good way to build ML intuition

Videos:

- Each page comes with a short video, highly recommended to watch
- In total about 40 minutes of videos for regression modeling

Quiz questions:

- In each page with hidden answer blocks
- At the end of the book section on a dedicated page — these are exam-like questions

Week 2.6 – PA and WS

Programming assignment:

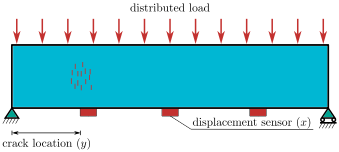
- Split a dataset into training/validation/test blocks
- Crucial operation when training ML models!

Wednesday workshop:

- Build your first neural networks with `scikit-learn`
- Experiment with data normalization, overfitting, underfitting and model selection
- Revisit the road deformation project from Week 1.3, now with neural nets

Code from both assignments will be very helpful for Friday

Week 2.6 — Friday project

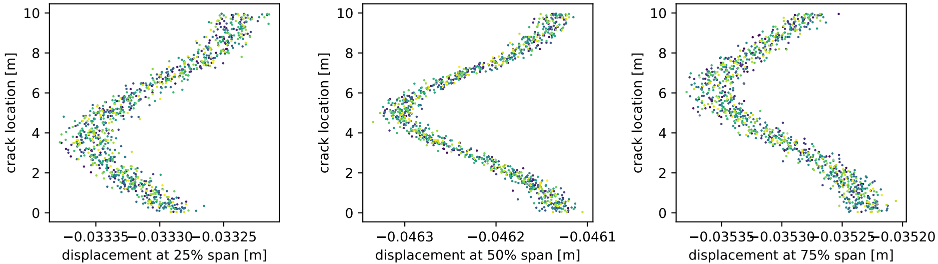


Detecting cracks in bridges with neural networks:

- Beams with a crack somewhere along the span
- We do not know the crack location but we do know how the beam deforms
- We train a net with a dataset of 800 beams and predict 200 unseen ones

A complete ML regression workflow:

- Pre-process the data and get it ready for training
- Experiment with feature selection, starting with one sensor at midspan
- Perform a well-structured model selection procedure to pick the best architecture



TUDelft

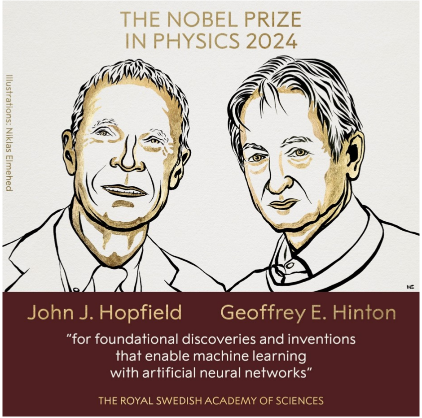
Objectives

Contents:

- Decision theory for regression, k-Nearest Neighbors estimator
- Linear regression with nonlinear basis functions
- Introduction to neural networks for regression

By the end of the week, you will be able to:

- Compare different regression modeling approaches
- Construct parsimonious regression models
- Critically assess model performance



TU Delft

Objectives

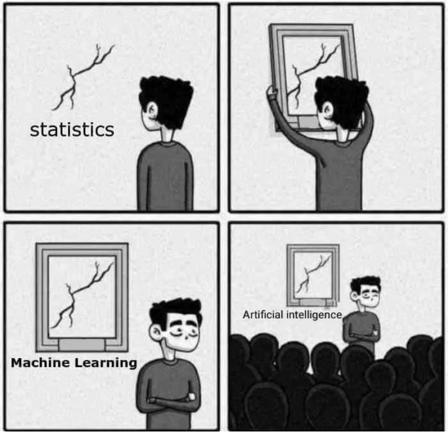
Contents:

- Decision theory for regression, k-Nearest Neighbors estimator
- Linear regression with nonlinear basis functions
- Introduction to neural networks for regression

By the end of the week, you will be able to:

- Compare different regression modeling approaches
- Construct parsimonious regression models
- Critically assess model performance

Brace yourself for some statistics:



TU Delft

Quick statistics recap

Continuous random variables represented by probability densities:

- Joint, marginal and conditional densities

$$p(x, y) \quad p(x) \;\; p(y) \quad p(x|y) \;\; p(y|x)$$

- For two independent variables x and y it holds

$$p(x, y) = p(x)p(y)$$

Quick statistics recap

Continuous random variables represented by probability densities:

- Joint, marginal and conditional densities

$$p(x, y) \quad p(x) \;\; p(y) \quad p(x|y) \;\; p(y|x)$$

- For two independent variables x and y it holds

$$p(x, y) = p(x)p(y)$$

Some useful integrals

- Expectation of a function of a random variable

$$\mathbb{E}[f(x)] = \int f(x)p(x)\,\mathrm{d}x$$

- Monte Carlo approximation of the expectation, with N samples x_i from $p(x)$:

$$\mathbb{E}[f(x)] \approx \frac{1}{N}\sum_{i}^{N} f(x_i)$$

- Variance of a function of a random variable

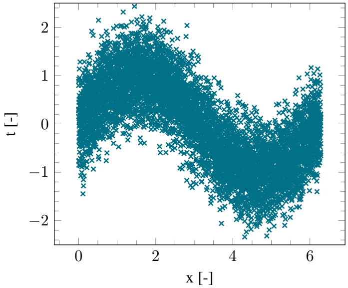
$$\mathrm{var}[f(x)] = \mathbb{E}\left[(f(x) - \mathbb{E}[f(x)])^2\right]$$

TU Delft

Regression problems

The problem we would like to solve:

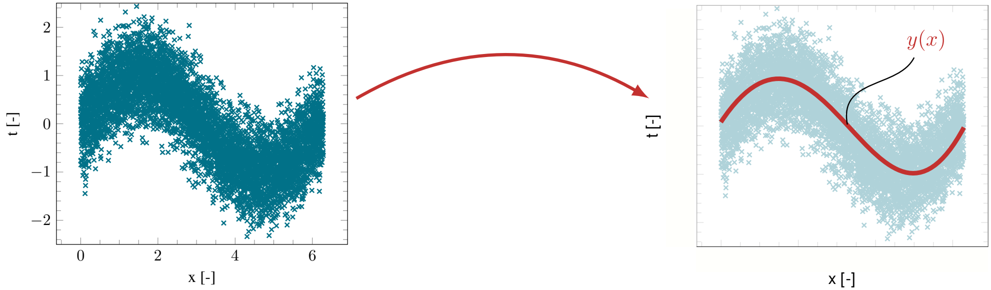
- Given: Some complex process $p(\mathbf{x}, t)$, usually highly nonlinear



Regression problems

The problem we would like to solve:

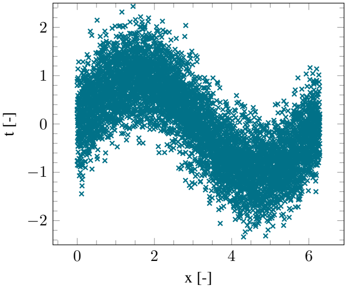
- Given: Some complex process $p(\mathbf{x}, t)$, usually highly nonlinear
- Goal: Construct a model $y(\mathbf{x})$ that is as close as possible to t

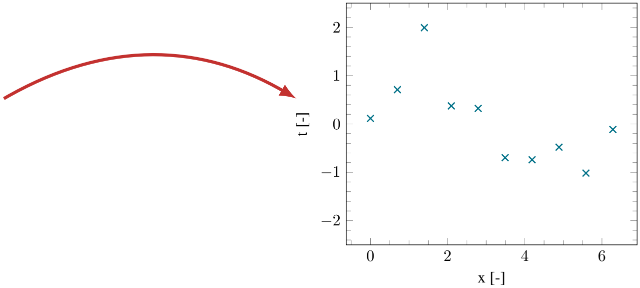


Regression problems

The problem we would like to solve:

- Given: Some complex process $p(\mathbf{x}, t)$, usually highly nonlinear
- Goal: Construct a model $y(\mathbf{x})$ that is as close as possible to t
- In practice: We do not know $p(\mathbf{x}, t)$, but only have N observations of it



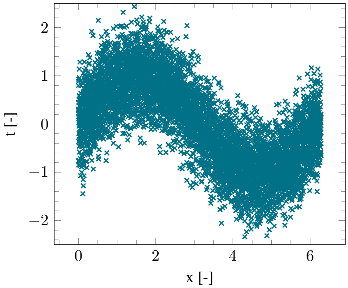


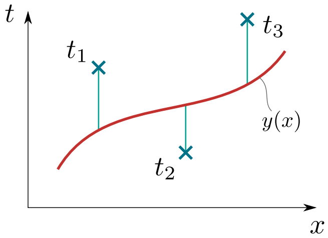
Regression problems

The problem we would like to solve:

- Given: Some complex process $p(\mathbf{x}, t)$, usually highly nonlinear
- Goal: Construct a model $y(\mathbf{x})$ that is as close as possible to t
- In practice: We do not know $p(\mathbf{x}, t)$, but only have N observations of it
- How to measure this "closeness"? The squared loss function is a popular choice:

$$L(t, y(\mathbf{x})) = (y(\mathbf{x}) - t)^2$$





Regression problems

The problem we would like to solve:

- Given: Some complex process $p(\mathbf{x}, t)$, usually highly nonlinear
- Goal: Construct a model $y(\mathbf{x})$ that is as close as possible to t
- In practice: We do not know $p(\mathbf{x}, t)$, but only have N observations of it
- How to measure this ”closeness”? The squared loss function is a popular choice:

$$L(t, y(\mathbf{x})) = (y(\mathbf{x}) - t)^2$$

- Here it is natural to go for the expectation:

$$\mathbb{E}[L] = \int\int (y(\mathbf{x}) - t)^2 \, p(\mathbf{x}, t) \, \mathrm{d}\mathbf{x} \, \mathrm{d}t$$

Regression problems

The problem we would like to solve:

- Given: Some complex process $p(\mathbf{x}, t)$, usually highly nonlinear
- Goal: Construct a model $y(\mathbf{x})$ that is as close as possible to t
- In practice: We do not know $p(\mathbf{x}, t)$, but only have N observations of it
- How to measure this "closeness"? The squared loss function is a popular choice:

$$L(t, y(\mathbf{x})) = (y(\mathbf{x}) - t)^2$$

- Here it is natural to go for the expectation:

$$\mathbb{E}[L] = \int\int (y(\mathbf{x}) - t)^2 \, p(\mathbf{x}, t) \, \mathrm{d}\mathbf{x} \, \mathrm{d}t$$

- Solving for the regression function $y(\mathbf{x})$ gives:

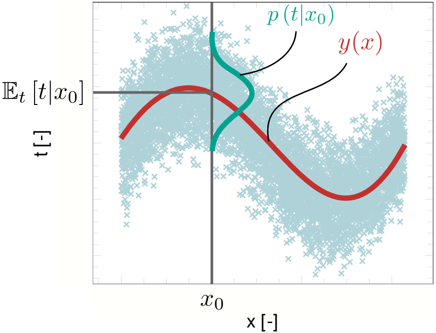
$$y(\mathbf{x}) = \int t p(t|x) \, \mathrm{d}t = \mathbb{E}_t[t|\mathbf{x}]$$

TU Delft

Regression problems

The problem we would like to solve:

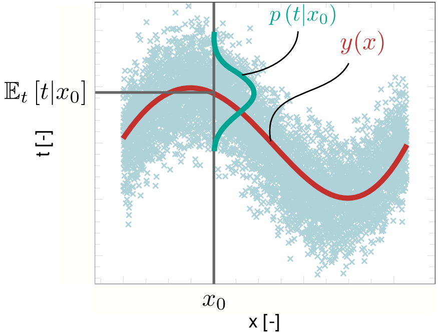
- Given: Some complex process $p(\mathbf{x}, t)$, usually highly nonlinear
- Goal: Construct a model $y(\mathbf{x})$ that is as close as possible to t



Regression problems

The problem we would like to solve:

- Given: Some complex process $p(\mathbf{x}, t)$, usually highly nonlinear
- Goal: Construct a model $y(\mathbf{x})$ that is as close as possible to t



- We can see this as an "ideal model" $h(x) = \mathbb{E}_t[t|\mathbf{x}]$, but it requires full knowledge of $p(\mathbf{x}, t)$ (i.e. infinite data)

TU Delft

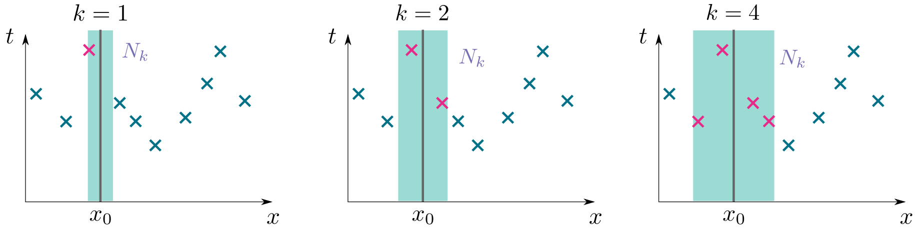
Reaching a model $y(x)$ with local approximations

Let us start with a very simple k-Nearest Neighbors (kNN) model:

Reaching a model $y(x)$ with local approximations

Let us start with a very simple k-Nearest Neighbors (kNN) model:

- We look at a neighborhood N_k around x_0 until we find k points $\Rightarrow$ approximate conditioning $\mathbb{E}[t|x]$

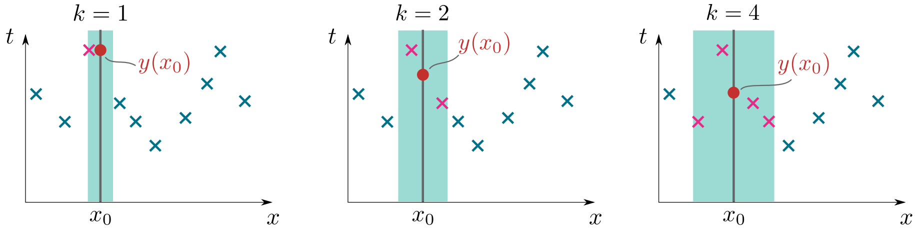


TU Delft

Reaching a model $y(x)$ with local approximations

Let us start with a very simple k-Nearest Neighbors (kNN) model:

- We look at a neighborhood N_k around x_0 until we find k points $\Rightarrow$ approximate conditioning $\mathbb{E}[t|x]$
- We then average these points to obtain $y(x_0)$ $\Rightarrow$ approximate expectation $\mathbb{E}[t|x]$

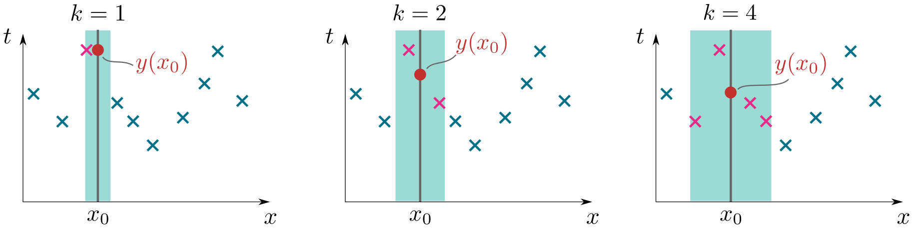


$$y(x_0) = \frac{1}{k} \sum_{x_i \in N_k} t_i$$

Reaching a model $y(x)$ with local approximations

Let us start with a very simple k-Nearest Neighbors (kNN) model:

- We look at a neighborhood N_k around x_0 until we find k points $\Rightarrow$ approximate conditioning $\mathbb{E}[t|x]$
- We then average these points to obtain $y(x_0)$ $\Rightarrow$ approximate expectation $\mathbb{E}[t|x]$



$$y(x_0) = \frac{1}{k} \sum_{x_i \in N_k} t_i$$

Now we just need to pick k by minimizing the loss:

- Since we only have N data points we minimize a Monte Carlo approximation:

$$\int\int (y(\mathbf{x}) - t)^2 \, p(\mathbf{x}, t) \, \mathrm{d}\mathbf{x} \, \mathrm{d}t \approx \frac{1}{N} \sum_{i}^{N} (y(\mathbf{x}_i) - t_i)^2$$

TU Delft

Now let us try this out

Go to bit.ly/engmechml or scan the QR code:

- Look at the first interactive plot
- Change the value of k until you are satisfied with the model
- Change the value of k until the training loss is as small as possible:

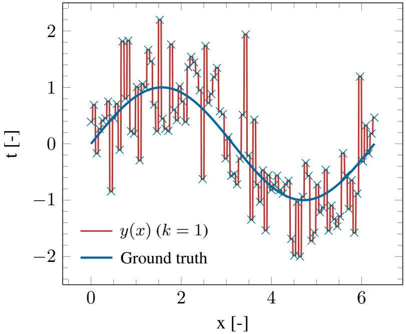
$$\mathbb{E}\left[L\right] \approx \frac{1}{N}\sum_{i}^{N}\left(y(\mathbf{x}_i, k) - t_i\right)^2$$

TU Delft

Overfitting and underfitting

This is the model we get if we are just trying to minimize the training loss:

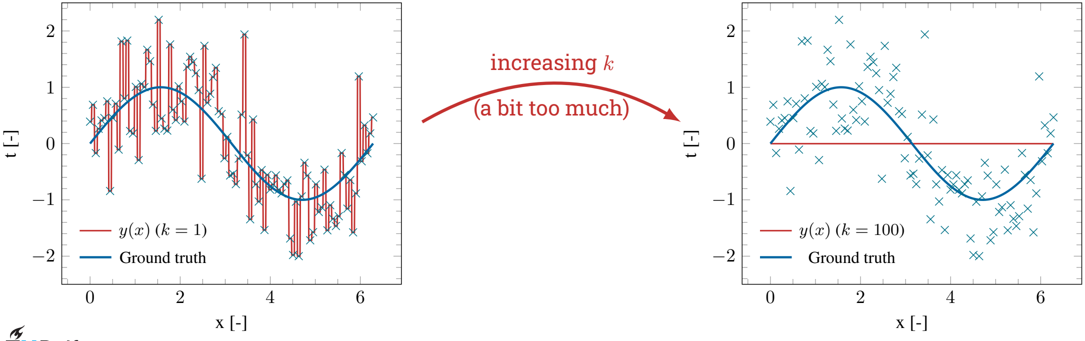
- Model fits the noise in the dataset and cannot generalize
- The error is exactly zero, but this is not a good model



Overfitting and underfitting

This is the model we get if we are just trying to minimize the training loss:

- Model fits the noise in the dataset and cannot generalize
- The error is exactly zero, but this is not a good model
- Too much freedom? What if we increase k?



Model selection

Clearly, choosing k is tricky:

- Too low: we fit the noise in the data $\Rightarrow$ overfitting!
- Too high: we oversmooth and lose detail $\Rightarrow$ underfitting!
- The training set cannot be trusted to give us k, it will always lead to $k = 1$

TU Delft

Model selection

Clearly, choosing k is tricky:

- Too low: we fit the noise in the data $\Rightarrow$ overfitting!
- Too high: we oversmooth and lose detail $\Rightarrow$ underfitting!
- The training set cannot be trusted to give us k, it will always lead to $k = 1$

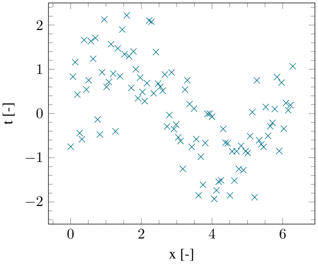
The solution is to introduce a validation dataset:

- A new dataset that cannot be used for training
- We can then use it to find the hyperparameter k:

$$k = \arg\min_{\bar{k}} \frac{1}{N_{\text{val}}} \sum_{i}^{N_{\text{val}}} \left(y(\mathbf{x}_i, \bar{k}) - t_i \right)^2$$

Model selection

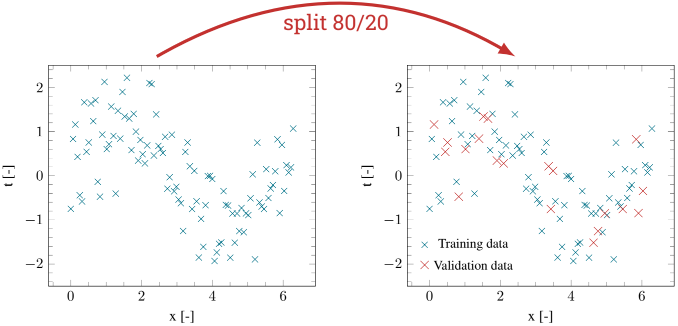
But how do we pick a validation set?



100 samples

Model selection

But how do we pick a validation set?

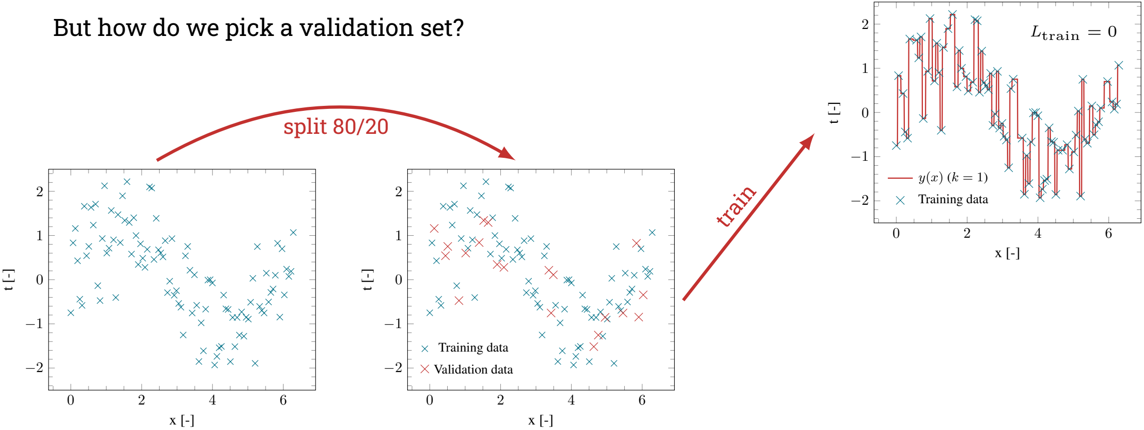


100 samples

80 training samples
20 validation samples

Model selection

But how do we pick a validation set?



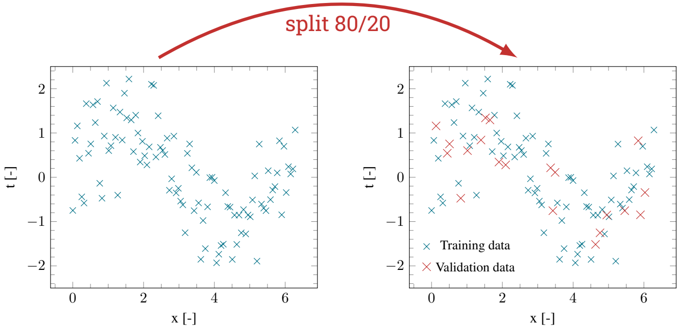
100 samples

80 training samples
20 validation
samples

TU Delft

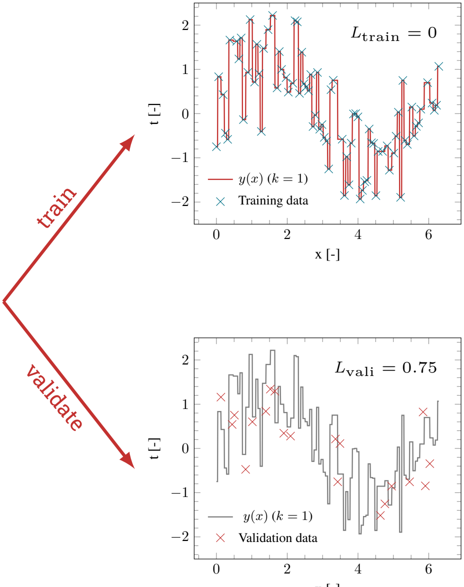
Model selection

But how do we pick a validation set?



100 samples

80 training samples
20 validation
samples

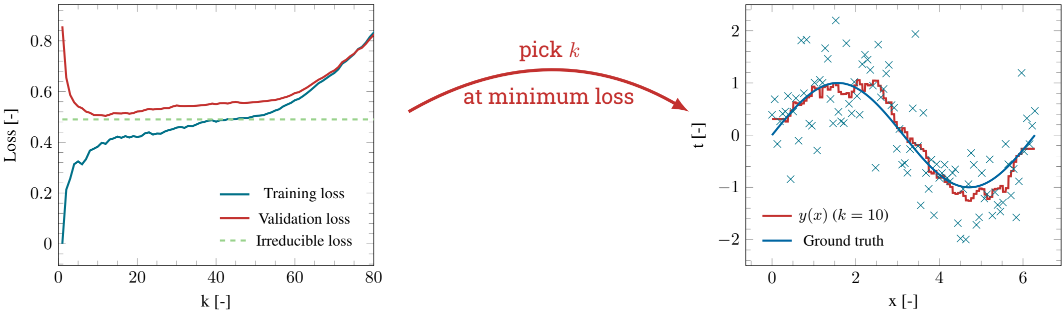


TU Delft

Model selection

The bias-variance tradeoff:

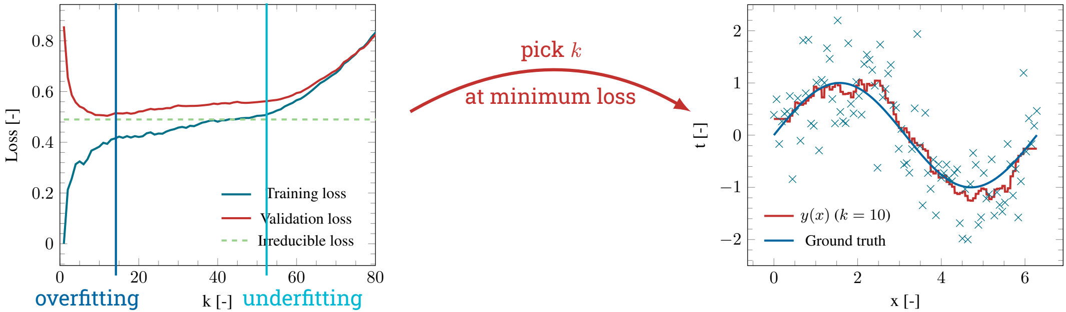
- Overly flexible models have low bias and high variance
- Overly rigid models have high bias and low variance
- We may accept some bias in exchange for a lower variance... but not too much



Model selection

The bias-variance tradeoff:

- Overly flexible models have low bias and high variance
- Overly rigid models have high bias and low variance
- We may accept some bias in exchange for a lower variance... but not too much



Let us do it one last time

Go to bit.ly/engmechml or scan the QR code:

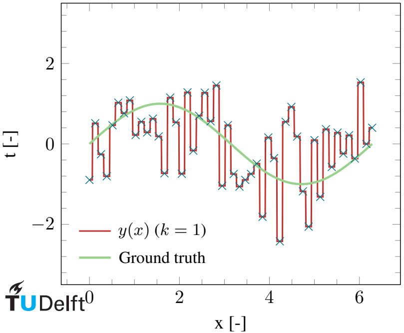
- Look at the third interactive plot
- Change the value of k until the validation loss is as low as possible

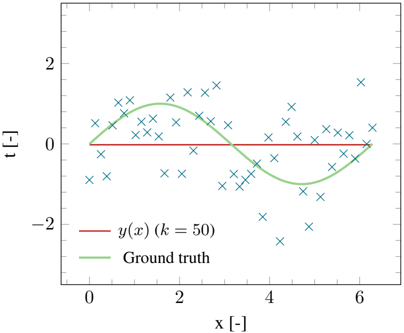
The bias-variance tradeoff

Why do we say flexible models have high variance? A closer look:

- Same example as before, but now 1000 different datasets of $N = 50$ each
- How much does the choice of dataset affect the final model?

$$L(\mathbf{x}) = \underbrace{\left(\mathbb{E}_{\mathcal{D}}\left[y(\mathbf{x},\mathcal{D})\right] - h(\mathbf{x})\right)^2}_{\text{bias}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(y(\mathbf{x},\mathcal{D}) - \mathbb{E}_{\mathcal{D}}\left[y(\mathbf{x},\mathcal{D})\right]\right)^2\right]}_{\text{variance}} + \underbrace{\int \left(h\left(\mathbf{x}\right) - t\right)^2 p(t|\mathbf{x})\,\mathrm{d}t}_{\text{irreducible noise}}$$



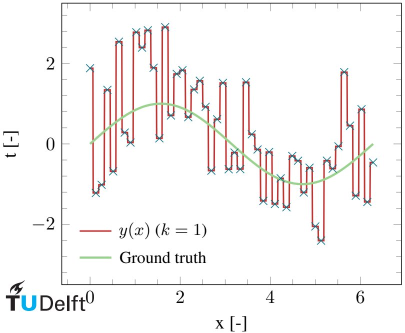


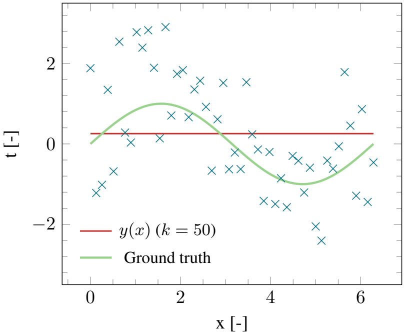
The bias-variance tradeoff

Why do we say flexible models have high variance? A closer look:

- Same example as before, but now 1000 different datasets of $N = 50$ each
- How much does the choice of dataset affect the final model?

$$L(\mathbf{x}) = \underbrace{\left(\mathbb{E}_{\mathcal{D}}\left[y(\mathbf{x},\mathcal{D})\right] - h(\mathbf{x})\right)^2}_{\text{bias}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(y(\mathbf{x},\mathcal{D}) - \mathbb{E}_{\mathcal{D}}\left[y(\mathbf{x},\mathcal{D})\right]\right)^2\right]}_{\text{variance}} + \underbrace{\int \left(h(\mathbf{x}) - t\right)^2 p(t|\mathbf{x})\,\mathrm{d}t}_{\text{irreducible noise}}$$



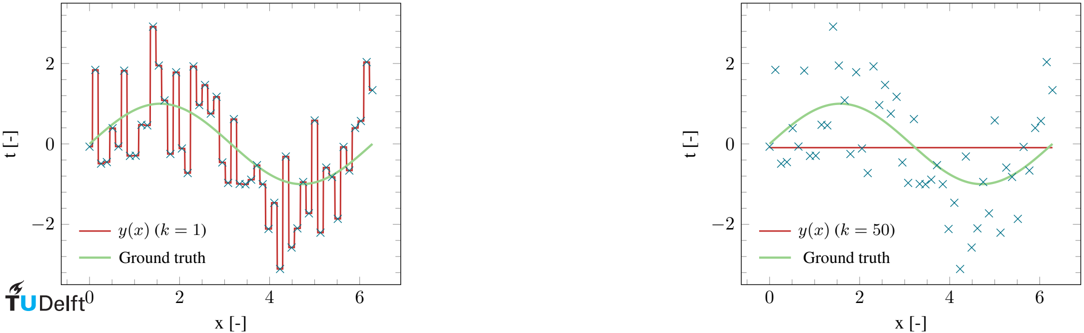


The bias-variance tradeoff

Why do we say flexible models have high variance? A closer look:

- Same example as before, but now 1000 different datasets of $N = 50$ each
- How much does the choice of dataset affect the final model?

$$L(\mathbf{x}) = \underbrace{\left(\mathbb{E}_{\mathcal{D}}\left[y(\mathbf{x},\mathcal{D})\right] - h(\mathbf{x})\right)^2}_{\text{bias}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(y(\mathbf{x},\mathcal{D}) - \mathbb{E}_{\mathcal{D}}\left[y(\mathbf{x},\mathcal{D})\right]\right)^2\right]}_{\text{variance}} + \underbrace{\int \left(h\left(\mathbf{x}\right) - t\right)^2 p(t|\mathbf{x})\,\mathrm{d}t}_{\text{irreducible noise}}$$

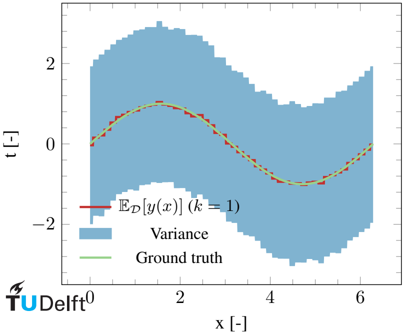


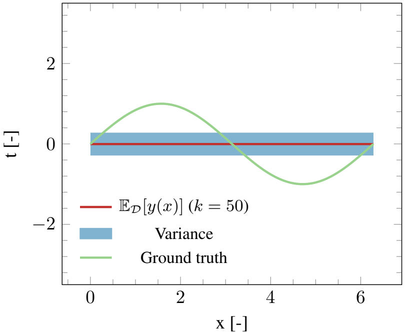
The bias-variance tradeoff

Why do we say flexible models have high variance? A closer look:

- Same example as before, but now 1000 different datasets of $N = 50$ each
- How much does the choice of dataset affect the final model?

$$L(\mathbf{x}) = \underbrace{\left(\mathbb{E}_{\mathcal{D}}\left[y(\mathbf{x},\mathcal{D})\right] - h(\mathbf{x})\right)^2}_{\text{bias}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(y(\mathbf{x},\mathcal{D}) - \mathbb{E}_{\mathcal{D}}\left[y(\mathbf{x},\mathcal{D})\right]\right)^2\right]}_{\text{variance}} + \underbrace{\int \left(h(\mathbf{x}) - t\right)^2 p(t|\mathbf{x})\,\mathrm{d}t}_{\text{irreducible noise}}$$





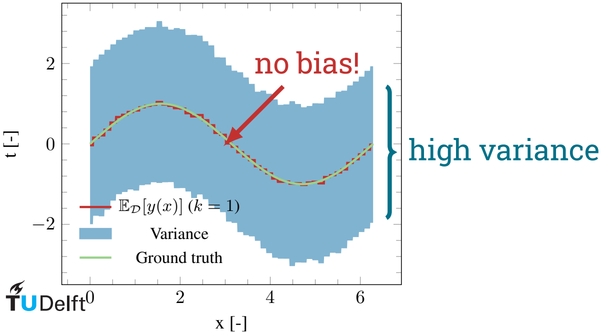
TU Delft

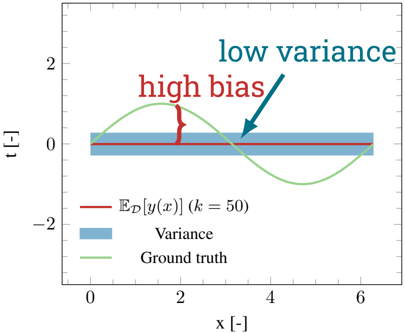
The bias-variance tradeoff

Why do we say flexible models have high variance? A closer look:

- Same example as before, but now 1000 different datasets of $N = 50$ each
- How much does the choice of dataset affect the final model?

$$L(\mathbf{x}) = \underbrace{\left(\mathbb{E}_{\mathcal{D}}\left[y(\mathbf{x},\mathcal{D})\right] - h(\mathbf{x})\right)^2}_{\text{bias}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(y(\mathbf{x},\mathcal{D}) - \mathbb{E}_{\mathcal{D}}\left[y(\mathbf{x},\mathcal{D})\right]\right)^2\right]}_{\text{variance}} + \underbrace{\int \left(h(\mathbf{x}) - t\right)^2 p(t|\mathbf{x})\,\mathrm{d}t}_{\text{irreducible noise}}$$





Reaching a model $y(x)$ with a global approximation

Observation model:

- We adopt a parametric model $y(\mathbf{x}, \mathbf{w})$ and assume some additive Gaussian noise:

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon \quad \text{with} \quad \epsilon \sim \mathcal{N}\left(0, \beta^{-1}\right)$$

Reaching a model $y(x)$ with a global approximation

Observation model:

- We adopt a parametric model $y(\mathbf{x}, \mathbf{w})$ and assume some additive Gaussian noise:

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon \quad \text{with} \quad \epsilon \sim \mathcal{N}\left(0, \beta^{-1}\right)$$

- Under the squared loss we have seen before, the regression function is simply:

$$\mathbb{E}_t\left[t \,|\, \mathbf{x}\right] = \int t p\left(t \,|\, \mathbf{x}\right) \, \mathrm{d}t = y(\mathbf{x}, \mathbf{w})$$

- The learning problem then boils down to fitting a Gaussian distribution

TU Delft

Reaching a model $y(x)$ with a global approximation

Observation model:

- We adopt a parametric model $y(\mathbf{x}, \mathbf{w})$ and assume some additive Gaussian noise:

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon \quad \text{with} \quad \epsilon \sim \mathcal{N}\left(0, \beta^{-1}\right)$$

- Under the squared loss we have seen before, the regression function is simply:

$$\mathbb{E}_t\left[t \,|\, \mathbf{x}\right] = \int t p\left(t \,|\, \mathbf{x}\right) \mathrm{d}t = y(\mathbf{x}, \mathbf{w})$$

- The learning problem then boils down to fitting a Gaussian distribution

The approximations for this flavor of models are now:

- Noise is Gaussian
- Response is unimodal (because the Gaussian has only one peak)
- The function $y(\mathbf{x}, \mathbf{w})$ might not be infinitely flexible (bias $\neq 0$)

Giving $y(x)$ some shape — linear basis function models

Simple linear regression, assuming D input features in $\mathbf{x}$:

- Parametric model, linear in its arguments:

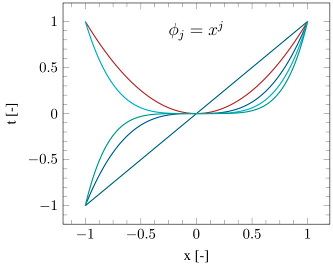
$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \cdots + w_D x_D$$

Giving $y(x)$ some shape — linear basis function models

Simple linear regression, assuming D input features in $\mathbf{x}$:

- Parametric model, linear in its arguments:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \cdots + w_D x_D$$

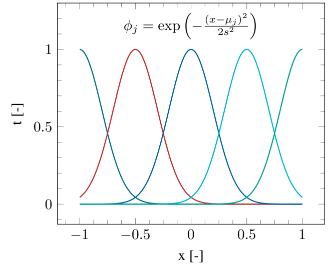


Here we make them more flexible:

- General nonlinear functions of $\mathbf{x}$ as regressors:

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j}^{M} w_j \phi_j(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x})$$

- A bias term $\phi_0 = 1$ is usually included in $\boldsymbol{\phi}$
- We are now unshackled from the original dimensionality D



TU Delft

Maximum Likelihood Estimation

Computing the likelihood of our data:

- The probability density of a given value t is:

$$p\left(t \mid \mathbf{w}\right) = \mathcal{N}\left(t \mid y(\mathbf{x}, \mathbf{w}), \beta^{-1}\right)$$

- Given a dataset $\mathcal{D}$ with observations $\mathbf{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ / $\mathbf{t} = [t_1, \cdots, t_N]$,

TU Delft

Maximum Likelihood Estimation

Computing the likelihood of our data:

- The probability density of a given value t is:

$$p\left(t \mid \mathbf{w}\right) = \mathcal{N}\left(t \mid y(\mathbf{x}, \mathbf{w}), \beta^{-1}\right)$$

- Given a dataset $\mathcal{D}$ with observations $\mathbf{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ / $\mathbf{t} = [t_1, \cdots, t_N]$,
- The likelihood of drawing our whole dataset from this Gaussian is therefore:

$$p\left(\mathcal{D} \mid \mathbf{w}\right) = \prod_{n=1}^{N} \mathcal{N}\left(t_n \mid \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}\right)$$

Maximum Likelihood Estimation

Computing the likelihood of our data:

- The probability density of a given value t is:

$$p\,(t\,|\,\mathbf{w}) = \mathcal{N}\left(t\,|\,y(\mathbf{x},\mathbf{w}), \beta^{-1}\right)$$

- Given a dataset $\mathcal{D}$ with observations $\mathbf{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ / $\mathbf{t} = [t_1, \cdots, t_N]$,
- The likelihood of drawing our whole dataset from this Gaussian is therefore:

$$p\,(\mathcal{D}\,|\,\mathbf{w}) = \prod_{n=1}^{N} \mathcal{N}\left(t_n\,|\,\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}\right)$$

- Applying the natural logarithm to both sides, we get:

$$\ln p\,(\mathcal{D}\,|\,\mathbf{w}) = \sum_{n=1}^{N} \ln \mathcal{N}\left(t_n\,|\,\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}\right) = \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \beta\left\{\frac{1}{2}\sum_{n=1}^{N}\left(t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\right)^2\right\}$$

TU Delft

Maximum Likelihood Estimation

Computing the likelihood of our data:

- The probability density of a given value t is:

$$p\left(t \mid \mathbf{w}\right) = \mathcal{N}\left(t \mid y(\mathbf{x}, \mathbf{w}), \beta^{-1}\right)$$

- Given a dataset $\mathcal{D}$ with observations $\mathbf{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ / $\mathbf{t} = [t_1, \cdots, t_N]$,
- The likelihood of drawing our whole dataset from this Gaussian is therefore:

$$p\left(\mathcal{D} \mid \mathbf{w}\right) = \prod_{n=1}^{N} \mathcal{N}\left(t_n \mid \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}\right)$$

- Applying the natural logarithm to both sides, we get:

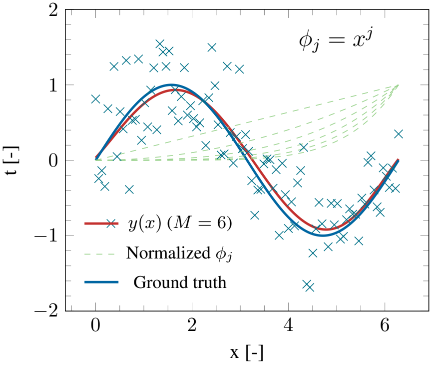
$$\ln p\left(\mathcal{D} \mid \mathbf{w}\right) = \sum_{n=1}^{N} \ln \mathcal{N}\left(t_n \mid \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}\right) = \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi) - \beta \left\{ \frac{1}{2} \sum_{n=1}^{N} \left(t_n - \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n)\right)^2 \right\}$$

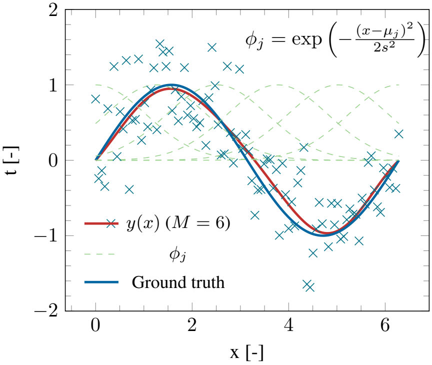
- Maximizing the likelihood is therefore equivalent to minimizing the error in red
- This is where the usual loss function for ML regression comes from

Maximum Likelihood Estimation

How does this look like? An example:

- Dataset with $N = 100$ observations, $M = 6$ basis functions (polynomials or Gaussians)



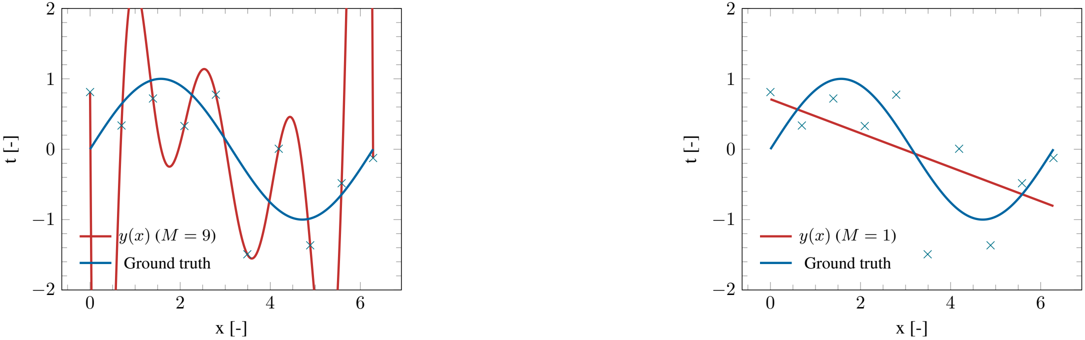


TU Delft

Overfitting and underfitting MLE models

Also here, flexibility is not always a good thing:

- Dataset with $N = 10$ observations, model with complete order M polynomials
- Again a tradeoff between bias and variance



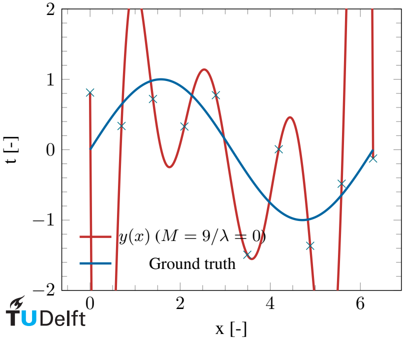
TU Delft

Regularized MLE models

L2 regularization, also known as Ridge Regression or Weight Decay:

- Model complexity is a bit hidden here. We can make it explicit by doing:

$$\mathbf{w}_{\mathrm{ML}} = \arg\min_{\overline{\mathbf{w}}} \left\{ \frac{1}{2} \sum_{n=1}^{N} \left(t_n - \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n) \right)^2 + \frac{\lambda}{2} \mathbf{w}^{\mathrm{T}} \mathbf{w} \right\} \Rightarrow \mathbf{w}_{\mathrm{ML}} = \left(\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} + \lambda \mathbf{I} \right)^{-1} \boldsymbol{\Phi}^{\mathrm{T}} \mathbf{t}$$



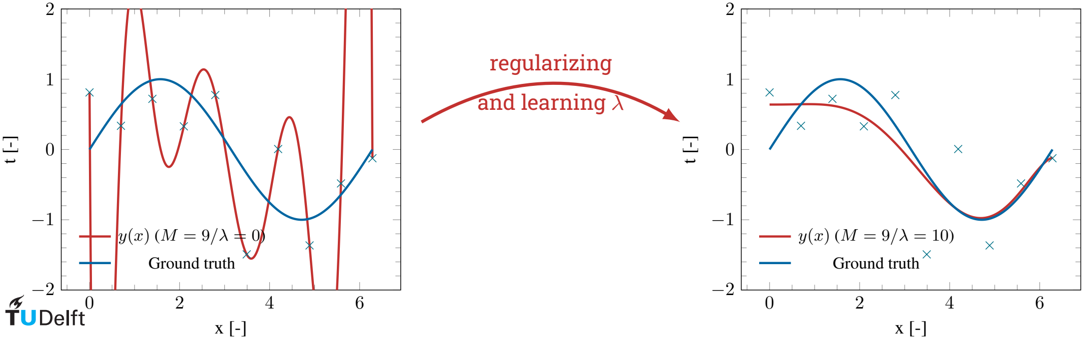
TU Delft

Regularized MLE models

L2 regularization, also known as Ridge Regression or Weight Decay:

- Model complexity is a bit hidden here. We can make it explicit by doing:

$$\mathbf{w}_{\mathrm{ML}} = \arg\min_{\overline{\mathbf{w}}} \left\{ \frac{1}{2} \sum_{n=1}^{N} \left(t_n - \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n) \right)^2 + \frac{\lambda}{2} \mathbf{w}^{\mathrm{T}} \mathbf{w} \right\} \Rightarrow \mathbf{w}_{\mathrm{ML}} = \left(\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} + \lambda \mathbf{I} \right)^{-1} \boldsymbol{\Phi}^{\mathrm{T}} \mathbf{t}$$



TU Delft

Stochastic Gradient Descent

For now we have trained with the complete dataset at once:

- The error function contains all N data points:

$$E_{\mathcal{D}} = \frac{1}{2} \sum_{n=1}^{N} \left(t_n - \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n) \right)^2$$

Situations when it is interesting (or necessary) to deviate from this:

- N is too large and computing $\left(\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi}\right)^{-1}$ becomes prohibitive
- The model is nonlinear (in $\mathbf{w}$) and $\mathbf{w}_{\mathrm{ML}}$ does not have a closed-form solution
- The dataset is arriving sequentially (e.g. in real time from a sensor)

TU Delft

Stochastic Gradient Descent

Instead of solving directly for $\mathbf{w}_{\mathrm{ML}}$, we can use Gradient Descent:

- Pick a (random) subset $\mathcal{B}$ of the dataset with $N_{\mathcal{B}}$ observations

Stochastic Gradient Descent

Instead of solving directly for $\mathbf{w}_{\mathrm{ML}}$, we can use Gradient Descent:

- Pick a (random) subset $\mathcal{B}$ of the dataset with $N_{\mathcal{B}}$ observations
- Update $\mathbf{w}$ with gradients coming from $\mathcal{B}$ and with a fixed learning rate η:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_{\mathcal{B}} \quad \text{with} \quad \nabla E_{\mathcal{B}} = -\sum_{n=1}^{N_{\mathcal{B}}} \left(t_n - \mathbf{w}^{(\tau)\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n) \right) \boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}$$

TU Delft

Stochastic Gradient Descent

Instead of solving directly for $\mathbf{w}_{\mathrm{ML}}$, we can use Gradient Descent:

- Pick a (random) subset $\mathcal{B}$ of the dataset with $N_{\mathcal{B}}$ observations
- Update $\mathbf{w}$ with gradients coming from $\mathcal{B}$ and with a fixed learning rate η:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_{\mathcal{B}} \quad \text{with} \quad \nabla E_{\mathcal{B}} = -\sum_{n=1}^{N_{\mathcal{B}}} \left(t_n - \mathbf{w}^{(\tau)\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n) \right) \boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}$$

- Every time the complete dataset has been seen, we say an epoch has passed

Stochastic Gradient Descent

Instead of solving directly for $\mathbf{w}_{\mathrm{ML}}$, we can use Gradient Descent:

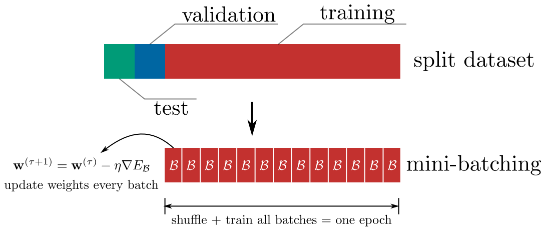
- Pick a (random) subset $\mathcal{B}$ of the dataset with $N_{\mathcal{B}}$ observations
- Update $\mathbf{w}$ with gradients coming from $\mathcal{B}$ and with a fixed learning rate η:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_{\mathcal{B}} \quad \text{with} \quad \nabla E_{\mathcal{B}} = -\sum_{n=1}^{N_{\mathcal{B}}} \left(t_n - \mathbf{w}^{(\tau)\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n) \right) \boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}$$

- Every time the complete dataset has been seen, we say an epoch has passed

Variations:

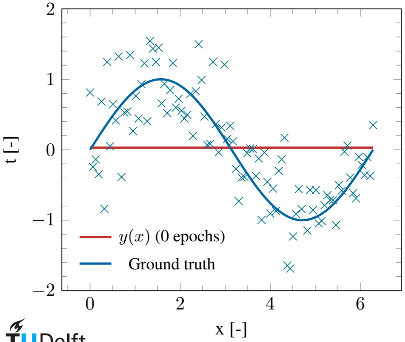
- $N_{\mathcal{B}} = 1$: Online stochastic gradient descent
- $1 < N_{\mathcal{B}} < N$: Minibatch SGD (most popular)
- $N_{\mathcal{B}} = N$: Full batch gradient descent

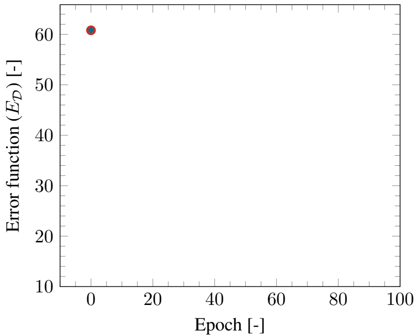


Stochastic Gradient Descent

An example:

- Same example as before, with $N = 100$ and $M = 6$ polynomial basis functions
- We fix the learning rate $\eta = 0.001$ and minibatch size $N_{\mathcal{B}} = 10$

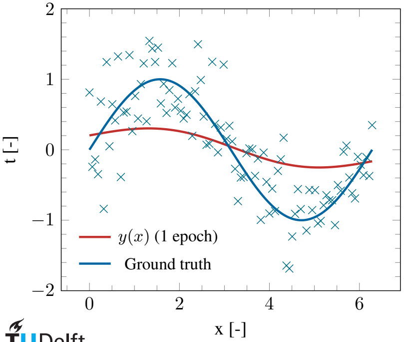


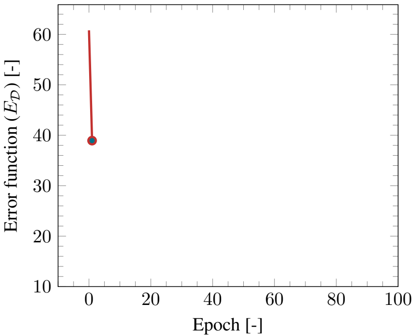


Stochastic Gradient Descent

An example:

- Same example as before, with $N = 100$ and $M = 6$ polynomial basis functions
- We fix the learning rate $\eta = 0.001$ and minibatch size $N_{\mathcal{B}} = 10$

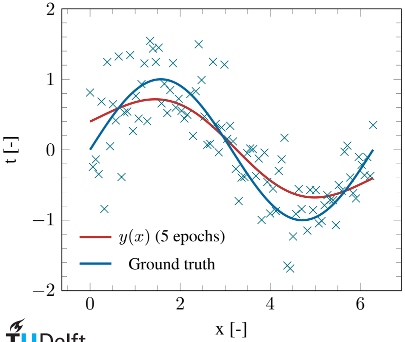


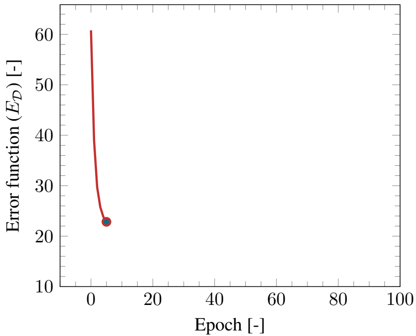


Stochastic Gradient Descent

An example:

- Same example as before, with $N = 100$ and $M = 6$ polynomial basis functions
- We fix the learning rate $\eta = 0.001$ and minibatch size $N_{\mathcal{B}} = 10$



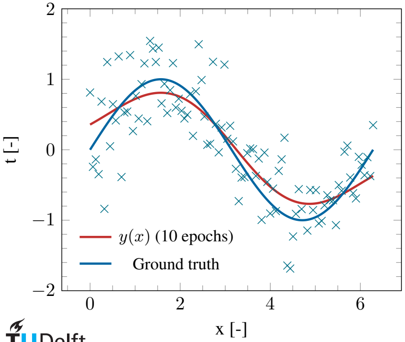


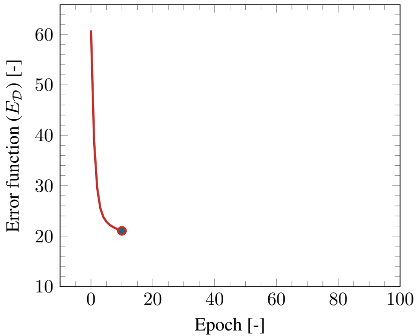
TU Delft

Stochastic Gradient Descent

An example:

- Same example as before, with $N = 100$ and $M = 6$ polynomial basis functions
- We fix the learning rate $\eta = 0.001$ and minibatch size $N_{\mathcal{B}} = 10$

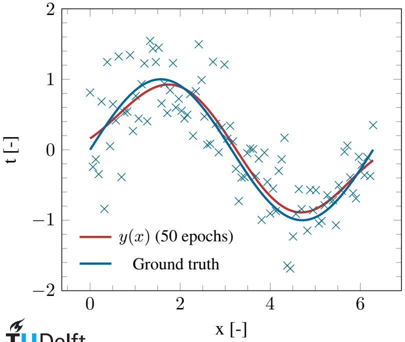


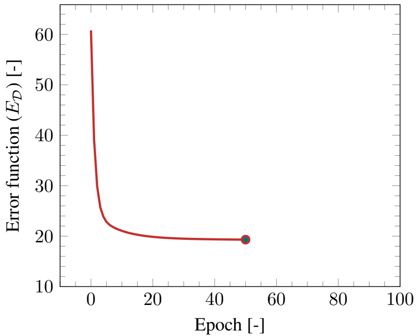


Stochastic Gradient Descent

An example:

- Same example as before, with $N = 100$ and $M = 6$ polynomial basis functions
- We fix the learning rate $\eta = 0.001$ and minibatch size $N_{\mathcal{B}} = 10$

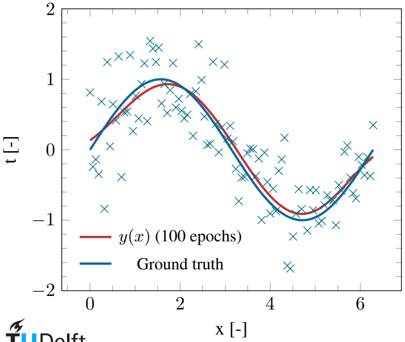


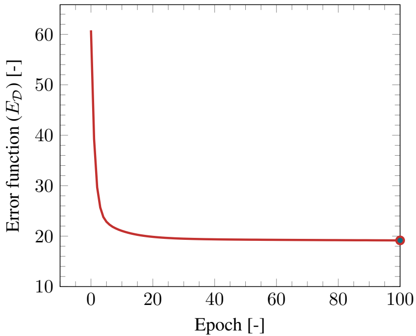


Stochastic Gradient Descent

An example:

- Same example as before, with $N = 100$ and $M = 6$ polynomial basis functions
- We fix the learning rate $\eta = 0.001$ and minibatch size $N_{\mathcal{B}} = 10$

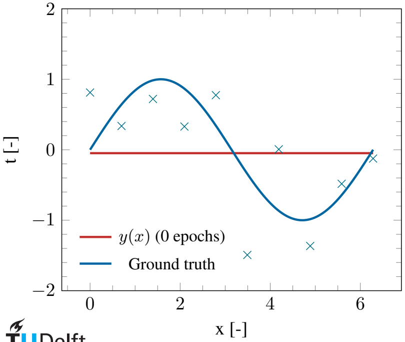


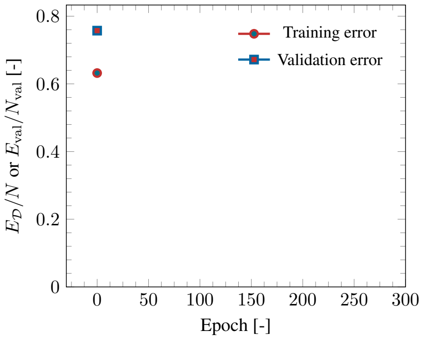


Stochastic Gradient Descent

Using SGD progress to spot signs of overfitting:

- Tracking the error on a validation dataset after every epoch
- This motivates the early stopping strategy popular in the deep learning community

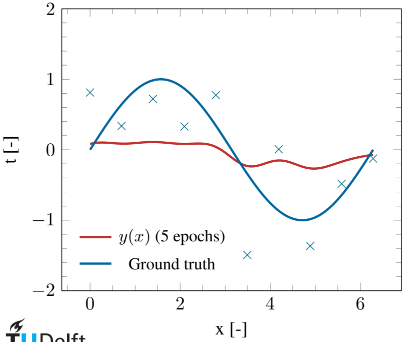


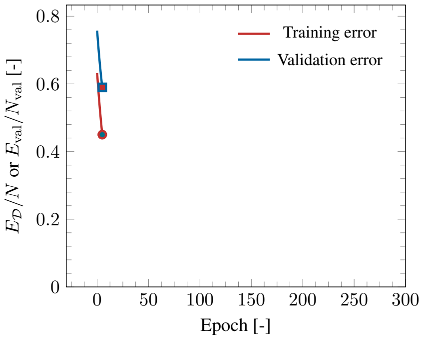


Stochastic Gradient Descent

Using SGD progress to spot signs of overfitting:

- Tracking the error on a validation dataset after every epoch
- This motivates the early stopping strategy popular in the deep learning community



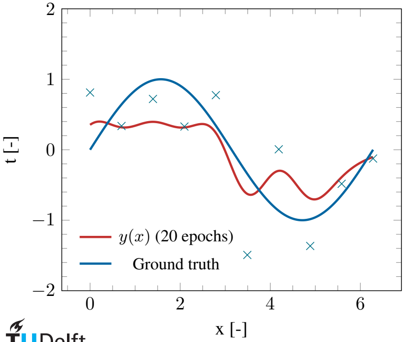


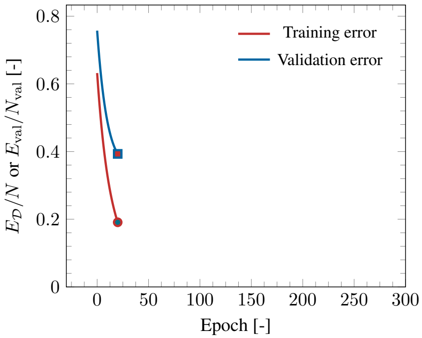
TU Delft

Stochastic Gradient Descent

Using SGD progress to spot signs of overfitting:

- Tracking the error on a validation dataset after every epoch
- This motivates the early stopping strategy popular in the deep learning community

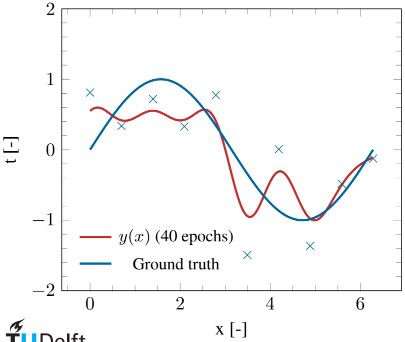


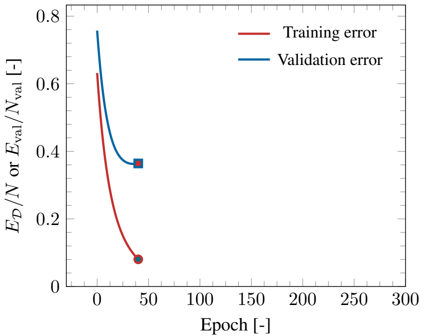


Stochastic Gradient Descent

Using SGD progress to spot signs of overfitting:

- Tracking the error on a validation dataset after every epoch
- This motivates the early stopping strategy popular in the deep learning community

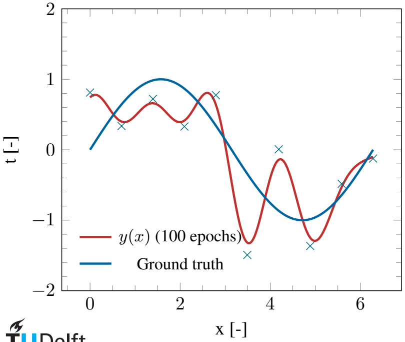


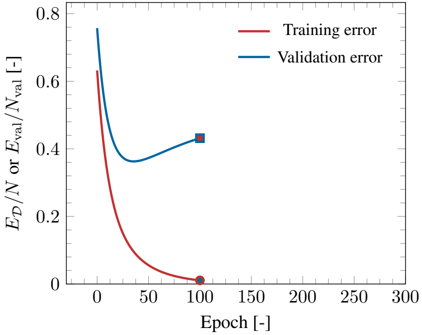


Stochastic Gradient Descent

Using SGD progress to spot signs of overfitting:

- Tracking the error on a validation dataset after every epoch
- This motivates the early stopping strategy popular in the deep learning community

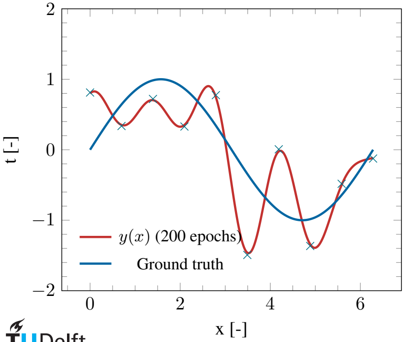


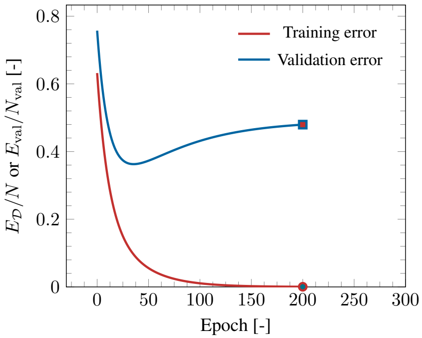


Stochastic Gradient Descent

Using SGD progress to spot signs of overfitting:

- Tracking the error on a validation dataset after every epoch
- This motivates the early stopping strategy popular in the deep learning community

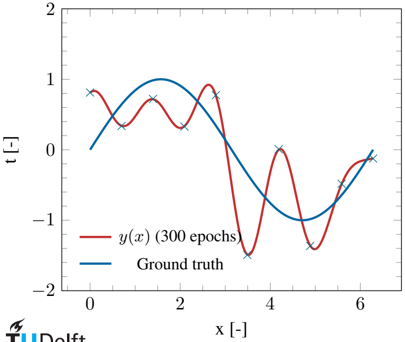


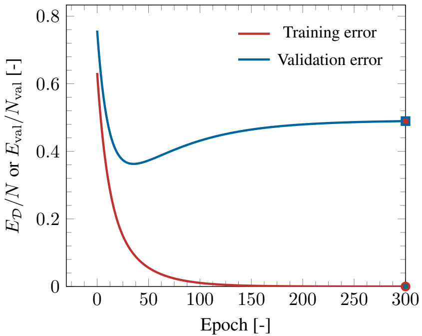


Stochastic Gradient Descent

Using SGD progress to spot signs of overfitting:

- Tracking the error on a validation dataset after every epoch
- This motivates the early stopping strategy popular in the deep learning community



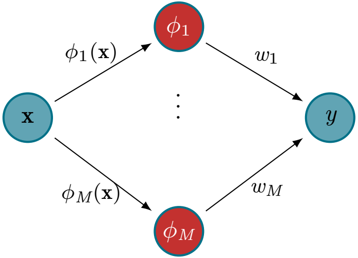


Adaptive basis functions

Up until now, the basis functions have been fixed a priori:

- Polynomials: number of terms M, polynomial degrees of each term
- Gaussians: bandwidth s, basis function centers μ_j

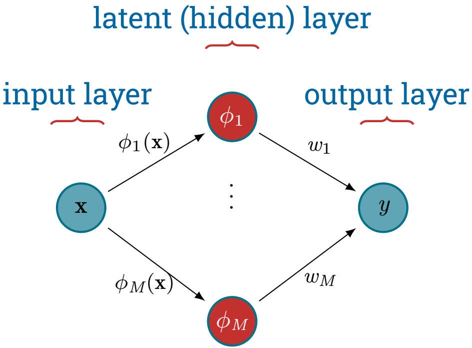
$$y = \phi_1(\mathbf{x})w_1 + \phi_2(\mathbf{x})w_2 + \cdots + \phi_M(\mathbf{x})w_M$$



Adaptive basis functions

For now, only half of the model is trainable:

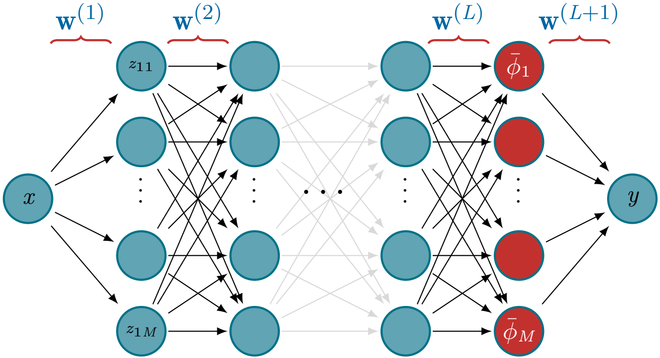
- Input to hidden encoding ($\phi_1 \cdots \phi_M$) fixed, hidden to output decoding ($\mathbf{w}$) trained
- What if we could also train the first half?



Artificial Neural Networks

Replacing basis functions by several layers of nonlinear transformations:

- Neural Network: layers of neurons linked by weighted connections
- Values at the red layer can be seen as coming from new, learned basis functions $\bar{\phi}(x)$



TU Delft

Neural Networks – Activation functions

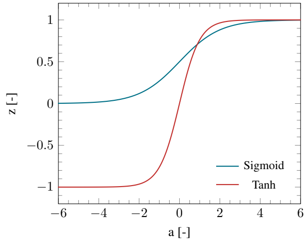
For a given neuron, forward propagation happens in two steps:

- A linear combination of values from the previous layer:

$$a_{lj} = \sum_{i}^{D} w_{ji}^{(l)} z_i^{(l-1)} + w_{j0}^{(l)}$$

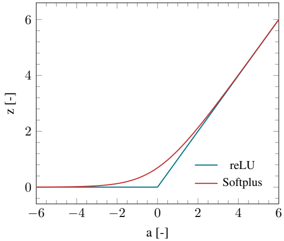
- A nonlinear transformation with an activation function:

$$z_{lj} = h\left(a_{lj}\right)$$



Choosing the activation function:

- Application dependent
- Can be seen as a hyperparameter

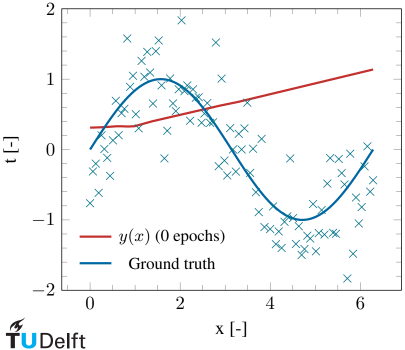


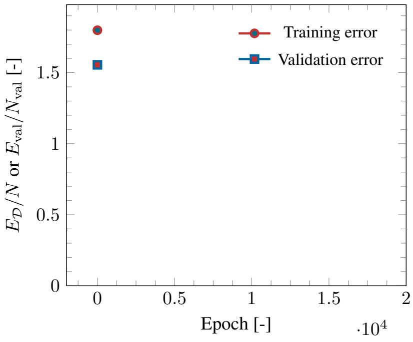
TUDelft

Neural Networks – Example

Same example as before, but now with a neural network:

- Full batch Adam SGD (variable learning rate)
- Two hidden layers, 10 neurons each, ReLU activation

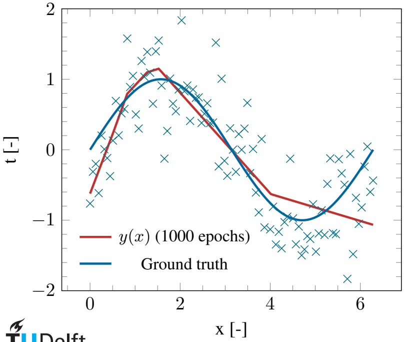


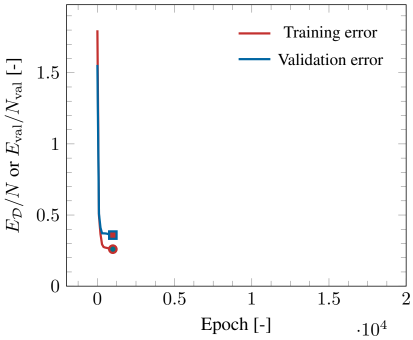


Neural Networks – Example

Same example as before, but now with a neural network:

- Full batch Adam SGD (variable learning rate)
- Two hidden layers, 10 neurons each, ReLU activation

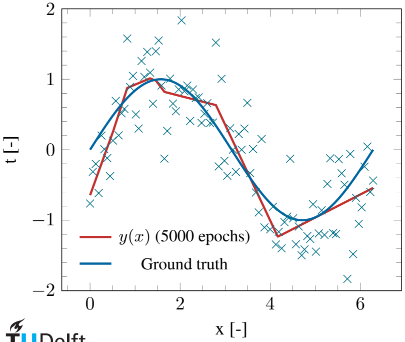


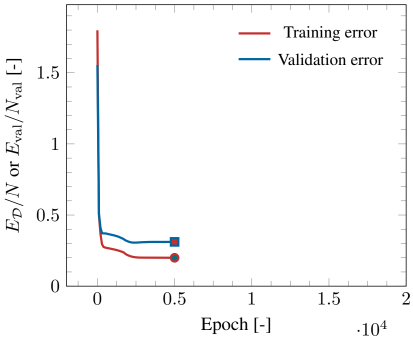


Neural Networks – Example

Same example as before, but now with a neural network:

- Full batch Adam SGD (variable learning rate)
- Two hidden layers, 10 neurons each, ReLU activation

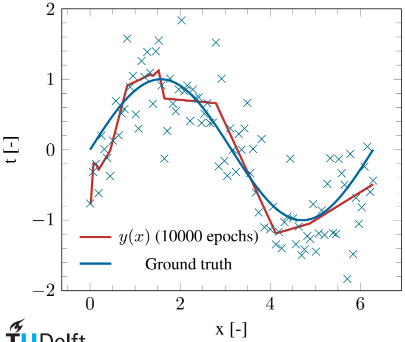


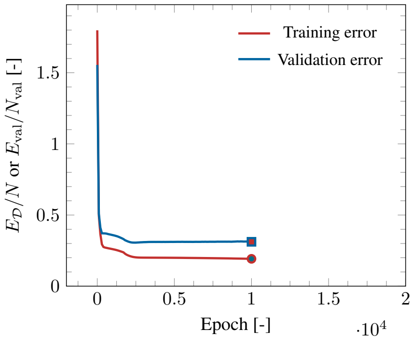


Neural Networks – Example

Same example as before, but now with a neural network:

- Full batch Adam SGD (variable learning rate)
- Two hidden layers, 10 neurons each, ReLU activation



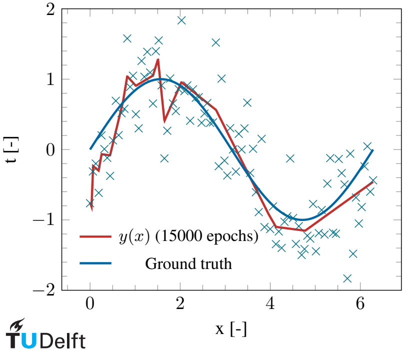


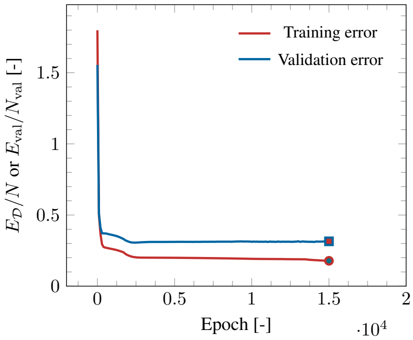
TU Delft

Neural Networks – Example

Same example as before, but now with a neural network:

- Full batch Adam SGD (variable learning rate)
- Two hidden layers, 10 neurons each, ReLU activation

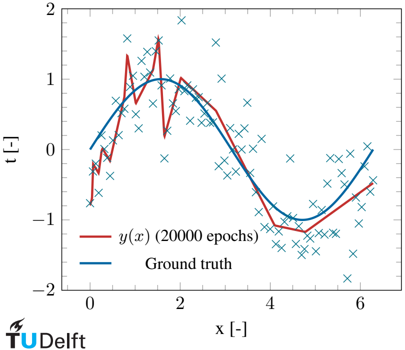


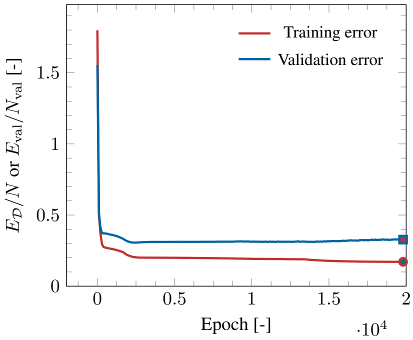


Neural Networks – Example

Same example as before, but now with a neural network:

- Full batch Adam SGD (variable learning rate)
- Two hidden layers, 10 neurons each, ReLU activation





TU Delft

Recap

You should now be able to:

- Understand and compare different approaches for regression modeling
- Construct parsimonious regression models for general applications
- Critically assess model performance from a probabilistic standpoint

Main takeaways:

- Model flexibility is not always beneficial, overfitting can be a major issue
- Selecting good models always boils down to balancing bias and variance
- Simply fitting data is not learning. ML is all about finding hidden patterns in data
- Neural Networks are not magic, can be seen as adaptive versions of simpler models
- Fancier deep learning models share this same foundation but add extra heuristics