Engineering Mechanics: Machine Learning

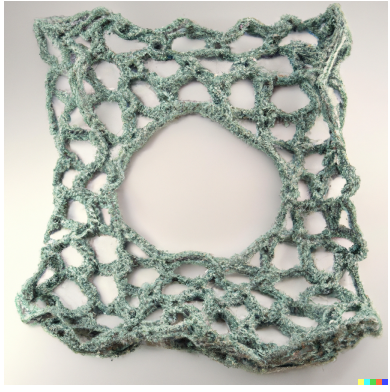
Introduction to ML

Iuri Rocha

A long, long time ago (1.5 years)

Some DALL-E generations we showed at the EM Symposium 2022:

A Finite Element model knitted out of wool



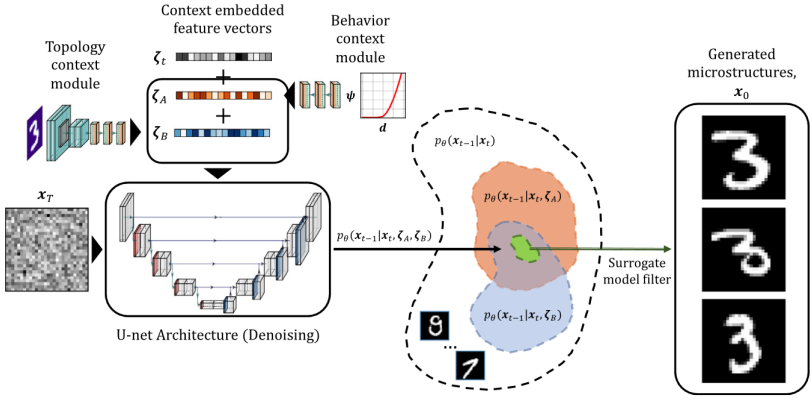
Two teddy bears discovering a new metamaterial

Today

Denoising diffusion for microstructure design:

- Tailored hyperelastic potential

*[Vlassis and Sun (2023), CMAME **413**:116126]*

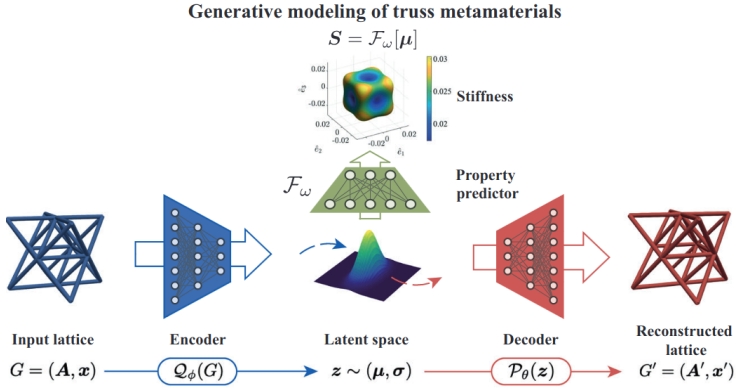


Today

Inverse design of spinodoid metamaterials:

- Tailored stiffness tensor

*[Zheng et al (2023), Nat Comm **2023(14)**:7563]*

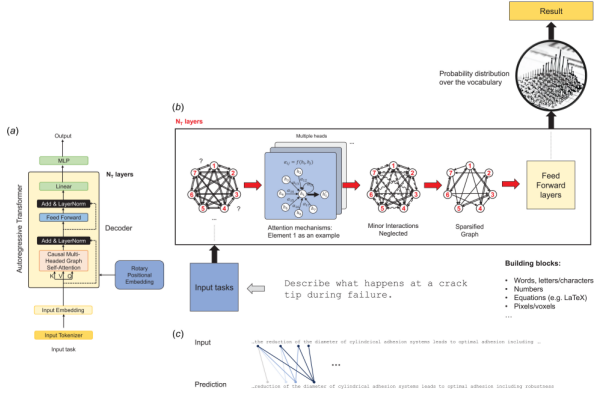


Today

MechGPT, a large language model fine-tuned for mechanics:

- Multimodality, non-trivial connections between different areas of knowledge

*[Buehler (2024), Appl Mech Rev **76(2)**:021001]*

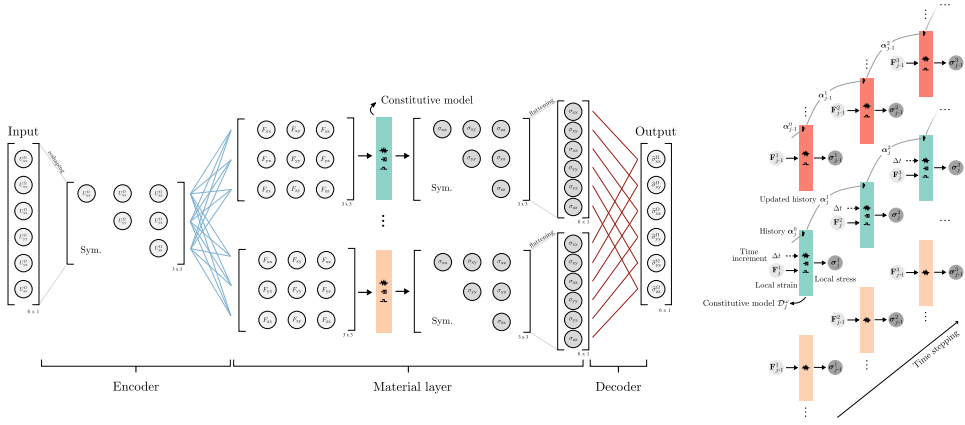


Today

Combining machine learning and physics in creative ways:

- Sparse connectivities, invariances, real material models embedded in architecture

[Maia et al (2024), Coming soon]



Machine Learning — an extremely quick primer

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...)

Machine Learning — an extremely quick primer

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 outputs ⇐ this course
- Classification: Map input features to discrete class labels

Machine Learning — an extremely quick primer

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 outputs ⇐ 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
- These models are often generative

Machine Learning — an extremely quick primer

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 outputs ⇐ 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
- These models are often generative

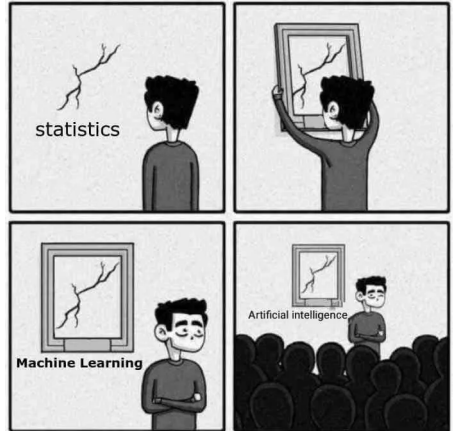
Reinforcement Learning: Learn a task through reward/punishment mechanisms:

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

Introduction to ML

Contents for this part of the course:

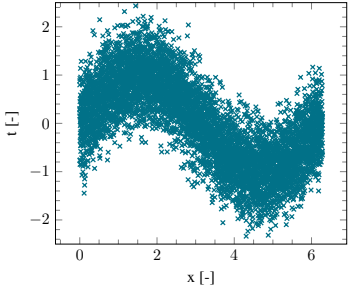
- Decision theory for regression
- Intuitive model building with k-Nearest Neighbors
- Robust model selection, bias-variance tradeoff
- From linear models to neural networks
- Bayesian ML with Gaussian Processes
- The curse of dimensionality, inductive biases



Regression problems

The problem we would like to solve:

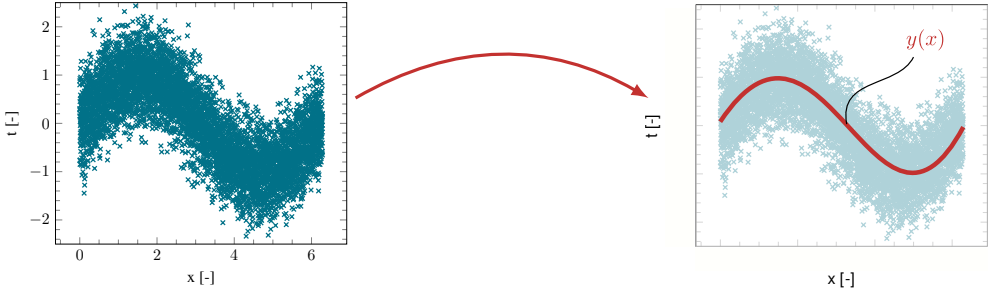
- Given: Some complex process $p(\mathbf{x}, t)$, usually highly nonlinear



Regression problems

The problem we would like to solve:

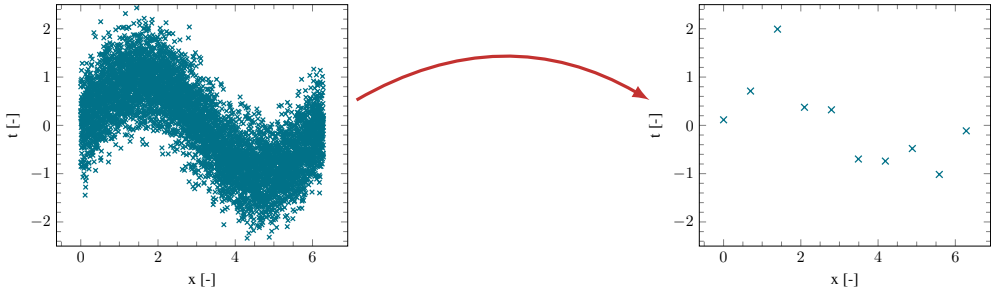
- Given: Some complex process $p(\mathbf{x}, t)$, usually highly nonlinear
- Goal: Construct a model $y(\mathbf{x})$ that explains it



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 explains it
- In practice: We do not know $p(\mathbf{x}, t)$, but only have N observations of it:



Choosing the model $y(x)$

Two main types:

- Parametric models: Knowledge of data encapsulated by a set of parameters: $y(\mathbf{x}, \mathbf{w})$
- Non-parametric models: The whole dataset is directly used to make predictions: $y(\mathbf{x}, \mathcal{D})$

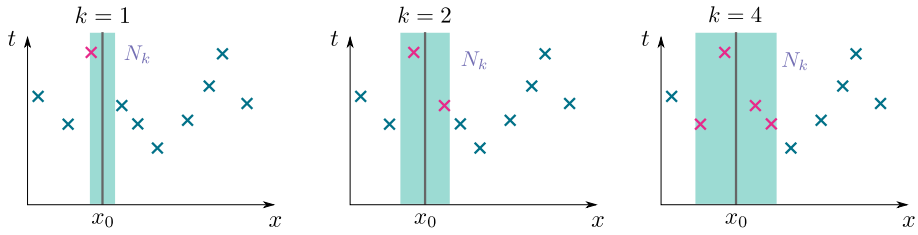
Choosing the model $y(x)$

Two main types:

- Parametric models: Knowledge of data encapsulated by a set of parameters: $y(\mathbf{x}, \mathbf{w})$
- Non-parametric models: The whole dataset is directly used to make predictions: $y(\mathbf{x}, \mathcal{D})$

Let us start with a very simple non-parametric model for $y(\mathbf{x})$:

- For a given $\mathbf{x}_0$, we look at a neighborhood N_k around it until we find k data points



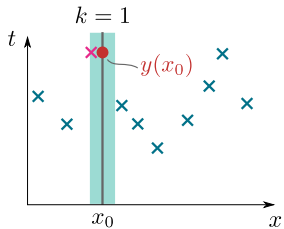
Choosing the model $y(x)$

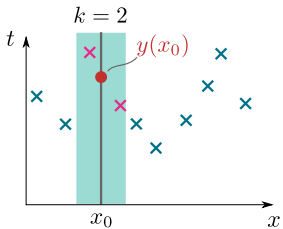
Two main types:

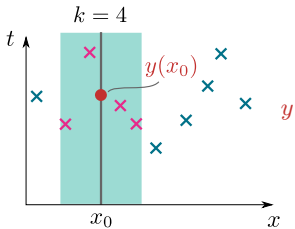
- Parametric models: Knowledge of data encapsulated by a set of parameters: $y(\mathbf{x}, \mathbf{w})$
- Non-parametric models: The whole dataset is directly used to make predictions: $y(\mathbf{x}, \mathcal{D})$

Let us start with a very simple non-parametric model for $y(\mathbf{x})$:

- For a given $\mathbf{x}_0$, we look at a neighborhood N_k around it until we find k data points
- We then average these points, resulting in a k-Nearest Neighbors (kNN) estimator





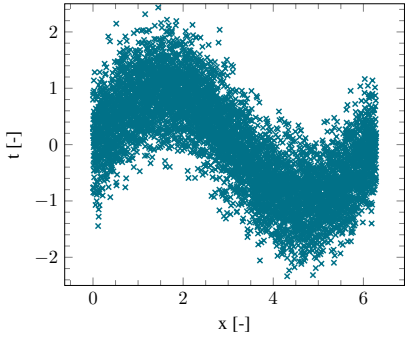


$$y(x_0) = \frac{1}{k} \sum_{x_i \in N_k} t_i$$

Decision Theory for regression

We formalize our problem again from the beginning:

- Given: Some process $p(\mathbf{x}, t)$ we would like to explain with a model
- Goal: Construct a model $y(\mathbf{x})$ that is as close as possible to t

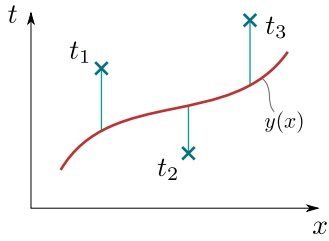


Decision Theory for regression

We formalize our problem again from the beginning:

- Given: Some process $p(\mathbf{x}, t)$ we would like to explain with a model
- Goal: Construct a model $y(\mathbf{x})$ that is as close as possible to t
- How to measure this "closeness"? The squared loss function is a popular choice:

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



Decision Theory for regression

We formalize our problem again from the beginning:

- Given: Some process $p(\mathbf{x}, t)$ we would like to explain with a model
- Goal: Construct a model $y(\mathbf{x})$ that is as close as possible to t
- Here it is natural to go for the expectation:

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

Decision Theory for regression

We formalize our problem again from the beginning:

- Given: Some process $p(\mathbf{x}, t)$ we would like to explain with a model
- Goal: Construct a model $y(\mathbf{x})$ that is as close as possible to t
- Here it is natural to go for the expectation:

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

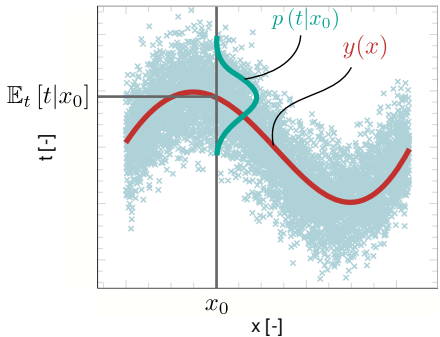
- Solving for the regression function $y(\mathbf{x})$ gives:

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

Decision Theory for regression

We formalize our problem again from the beginning:

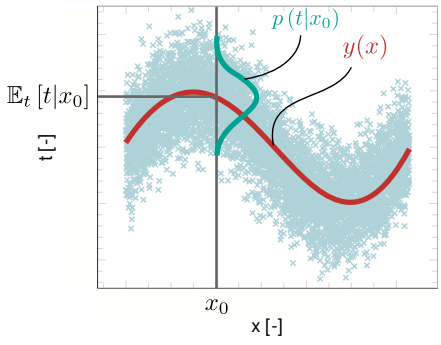
- Given: Some process $p(\mathbf{x}, t)$ we would like to explain with a model
- Goal: Construct a model $y(\mathbf{x})$ that is as close as possible to t



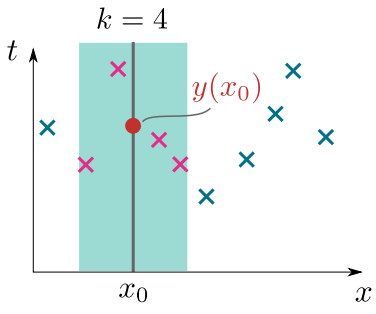
Decision Theory for regression

We formalize our problem again from the beginning:

- Given: Some process $p(\mathbf{x}, t)$ we would like to explain with a model
- Goal: Construct a model $y(\mathbf{x})$ that is as close as possible to t



parallel with kNN?

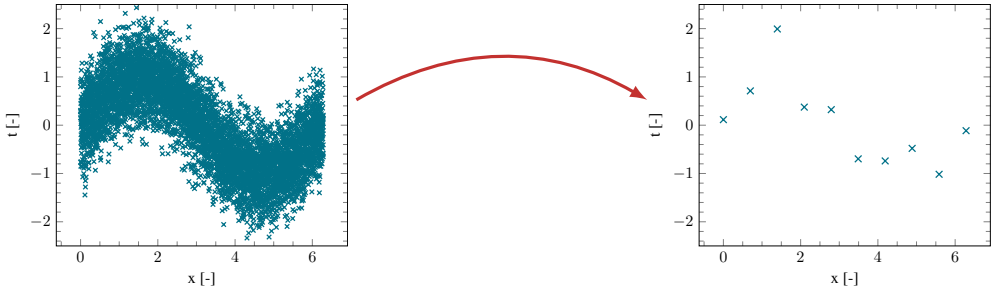


Decision Theory for regression

We formalize our problem again from the beginning:

- Goal: Construct a model $y(\mathbf{x})$ that is as close as possible to t
- In practice we do not know $p\,(x, t)$ exactly and make decisions based on limited data:

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

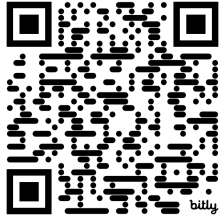


Now let us try this out

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:

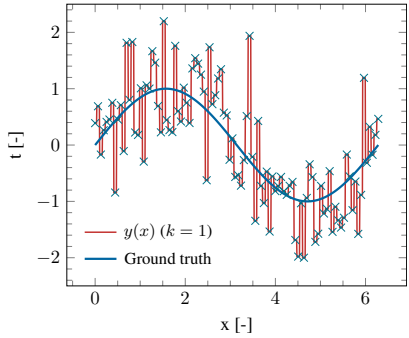
$$\mathbb{E}\left[L\right] \approx \frac{1}{N}\sum_{i}^{N}\left(y(\mathbf{x}_i, k) - t_i\right)^2$$



Overfitting and underfitting

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

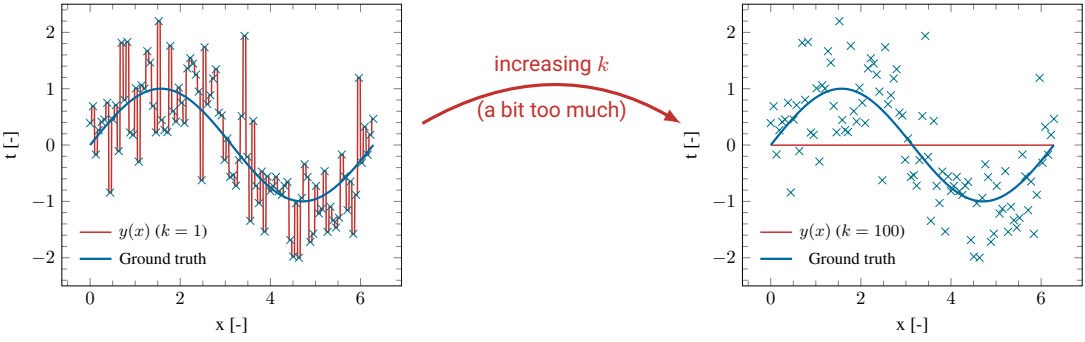
- Model fits the noise in the dataset and cannot generalize
- The error is exactly zero, but this is not a good model



Overfitting and underfitting

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

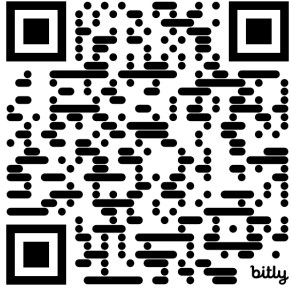
- Model fits the noise in the dataset and cannot generalize
- The error is exactly zero, but this is not a good model
- Too much freedom? What if we increase k?



Let us do this one more time

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

- Look at the second interactive plot
- Change the value of k until it is as close as possible to the ground truth



Model selection

In practice we do not know the ground truth, so 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 $k = 1$

Model selection

In practice we do not know the ground truth, so 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 $k = 1$

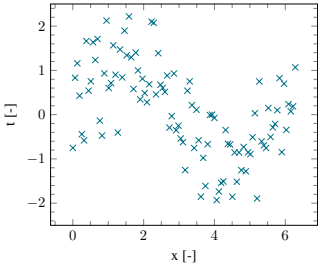
The solution is to introduce a validation dataset:

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

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

Model selection

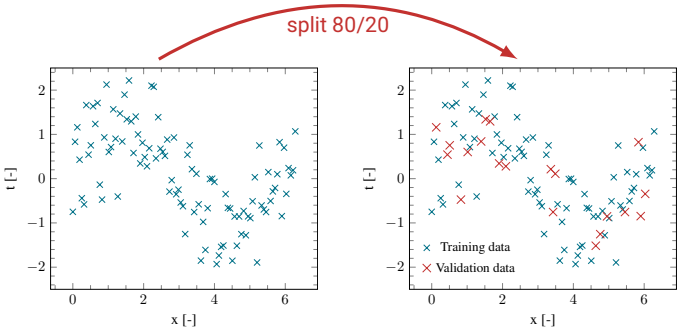
But how do we pick a validation set?



100 samples

Model selection

But how do we pick a validation set?

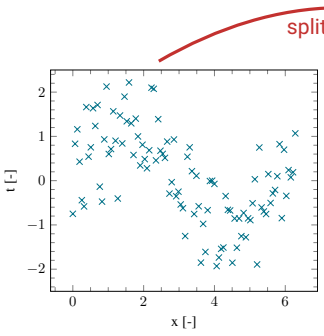


100 samples

80 training samples
20 validation samples

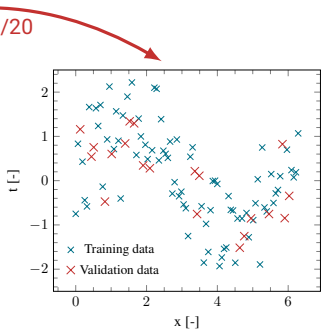
Model selection

But how do we pick a validation set?



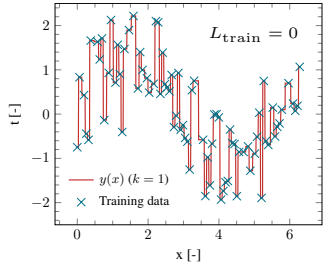
100 samples

split 80/20



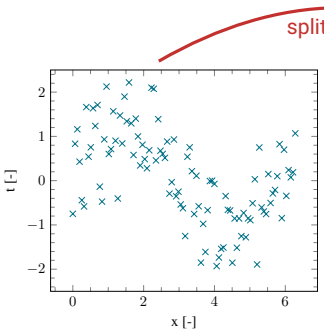
80 training samples
20 validation samples

train



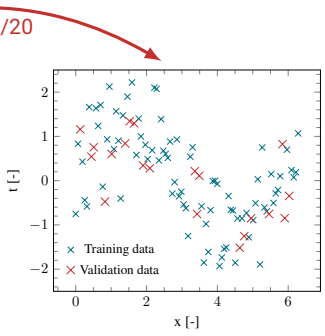
Model selection

But how do we pick a validation set?



100 samples

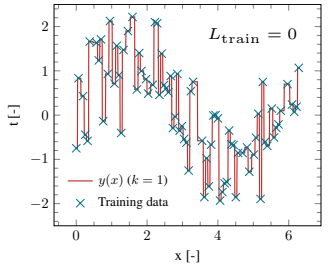
split 80/20

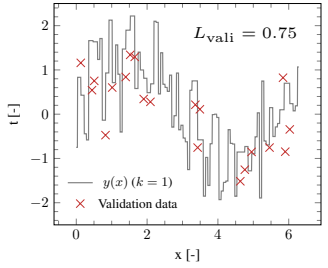


80 training samples
20 validation samples

train

validate

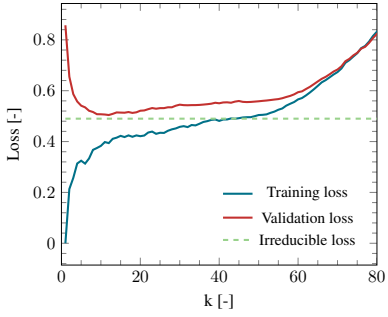




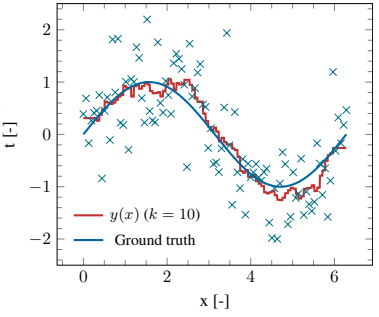
Model selection

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



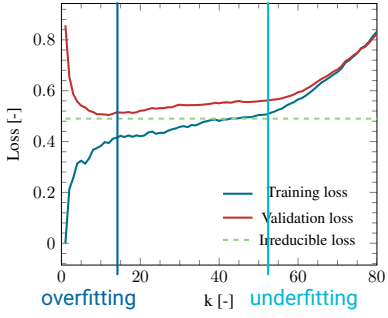
pick k at minimum loss

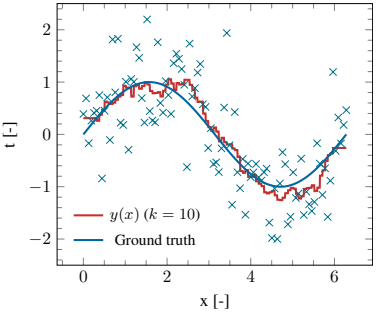


Model selection

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

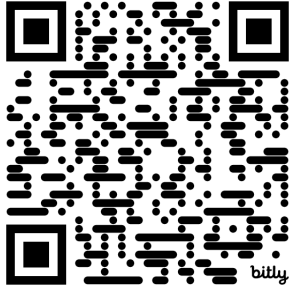




Let us do it one last time

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

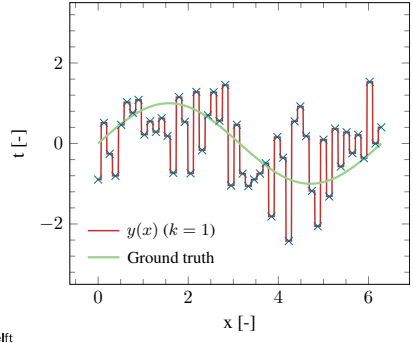


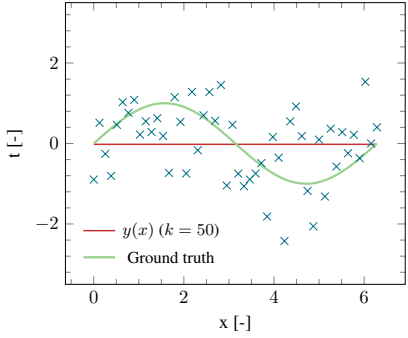
The bias-variance tradeoff

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?

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



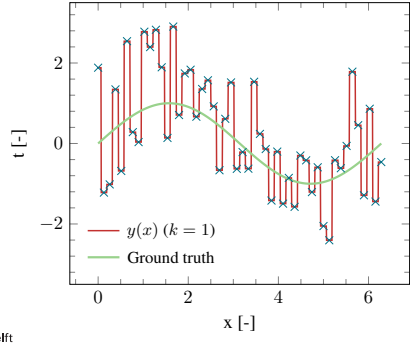


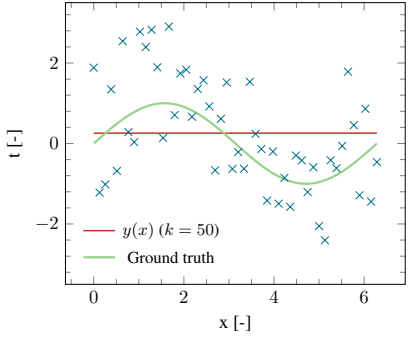
The bias-variance tradeoff

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?

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



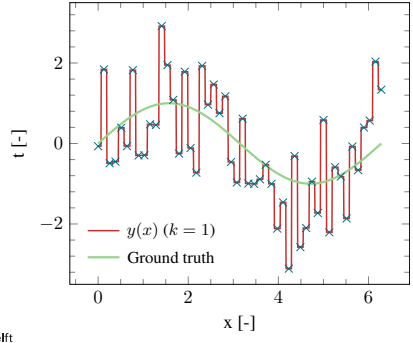


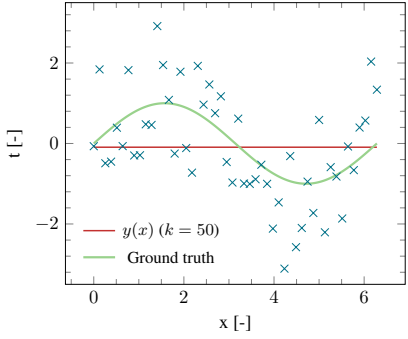
The bias-variance tradeoff

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?

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



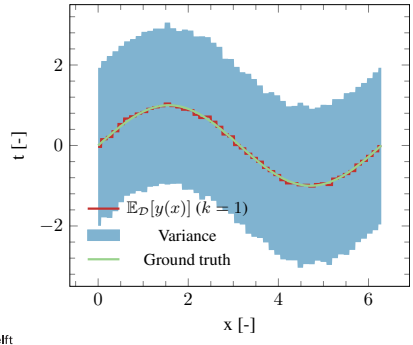


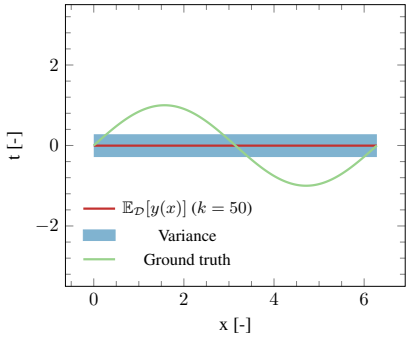
The bias-variance tradeoff

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?

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



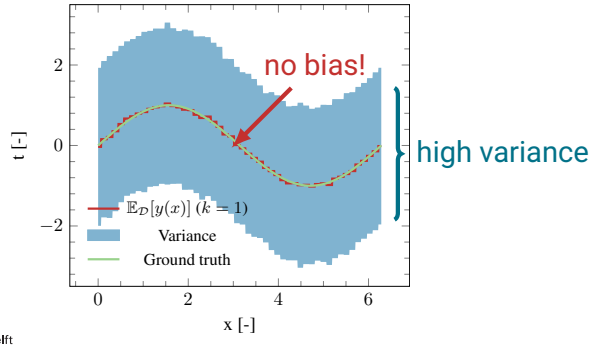


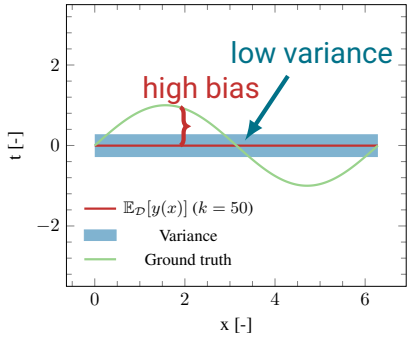
The bias-variance tradeoff

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?

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





Linear basis function models

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

- Parametric model, linear in its arguments:

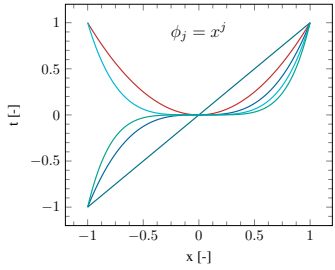
$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \cdots + w_D x_D$$

Linear basis function models

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

- Parametric model, linear in its arguments:

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

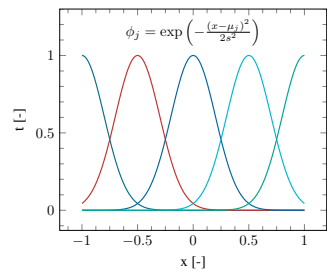


Here we make them more flexible:

- 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



Linear basis function models

Observation model:

- We adopt a parametric model and assume additive Gaussian noise:

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

Linear basis function models

Observation model:

- We adopt a parametric model and assume 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:

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

Linear basis function models

Observation model:

- We adopt a parametric model and assume 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:

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

We are implicitly assuming:

- Noise is Gaussian
- Response is unimodal

Maximum Likelihood Estimation

Computing the likelihood of our data:

- The probability density of a given value t is:

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

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

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:

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

- 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:

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

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:

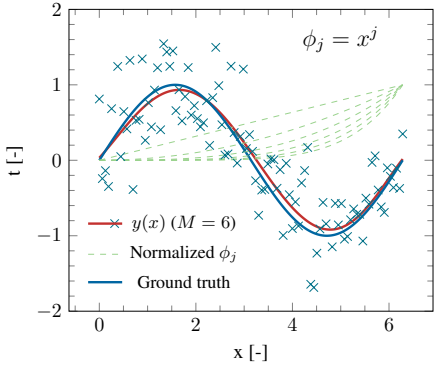
$$\ln p\left(\mathcal{D} \mid \mathbf{w}\right) = \sum_{n=1}^{N} \ln \mathcal{N}\left(t_n \mid \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}\right) = \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi) - \beta \left\{ \frac{1}{2} \sum_{n=1}^{N} \left(t_n - \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n)\right)^2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

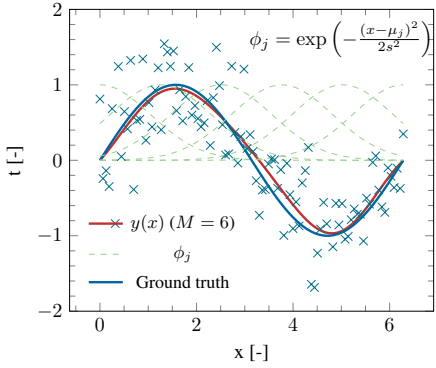
- Maximizing the likelihood is therefore equivalent to minimizing the error in red
- This is where the usual loss function for ML regression comes from

Maximum Likelihood Estimation

How does this look like? An example:

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

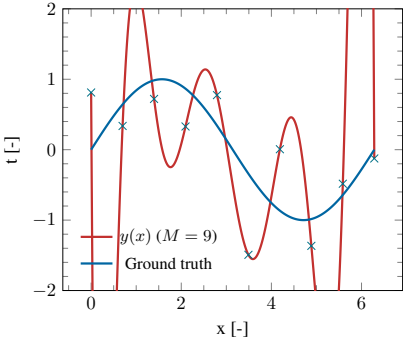


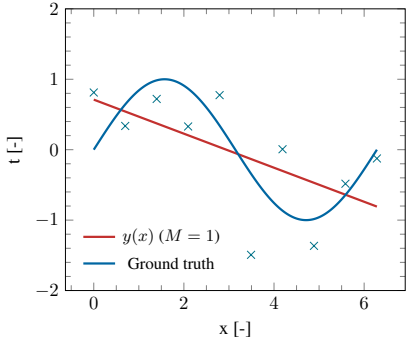


Overfitting and underfitting MLE models

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





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)

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:

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

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

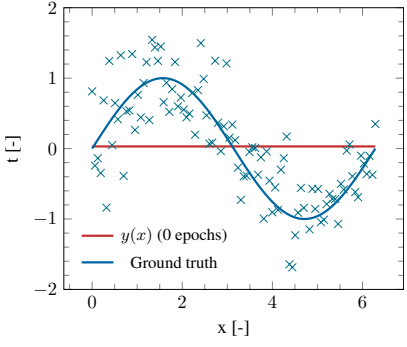
Variations:

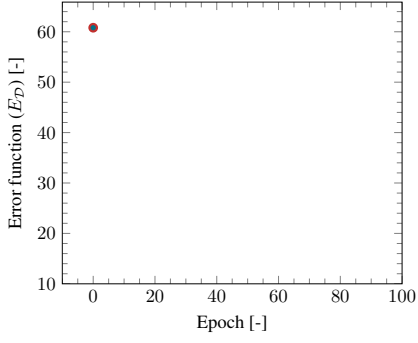
- $N_{\mathcal{B}} = 1$: Online stochastic gradient descent
- $1 < N_{\mathcal{B}} < N$: Minibatch SGD (most popular)
- $N_{\mathcal{B}} = N$: Full batch gradient descent

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 $N_{\mathcal{B}} = 10$

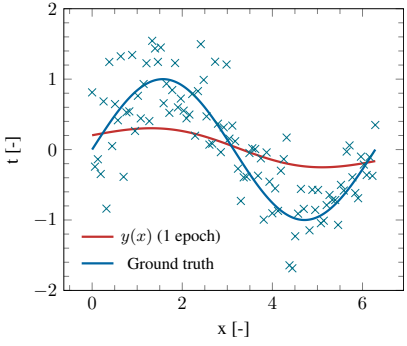


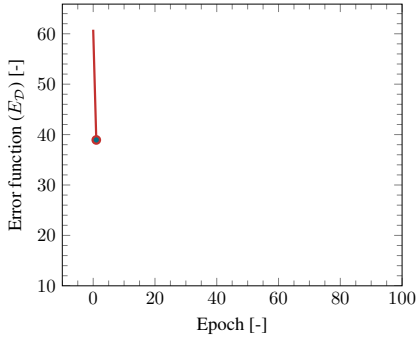


Stochastic Gradient Descent

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 $N_{\mathcal{B}} = 10$

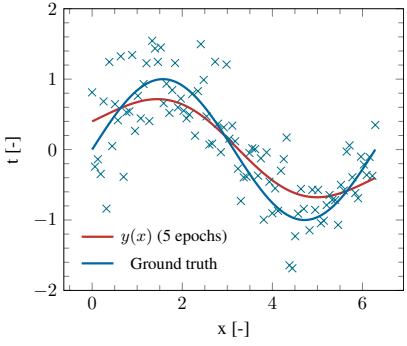


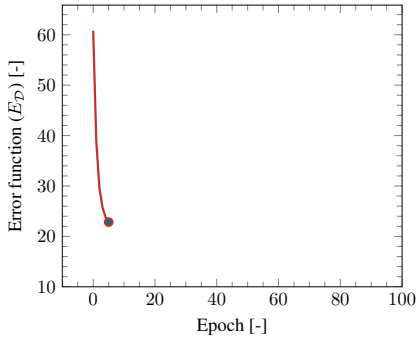


Stochastic Gradient Descent

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 $N_{\mathcal{B}} = 10$

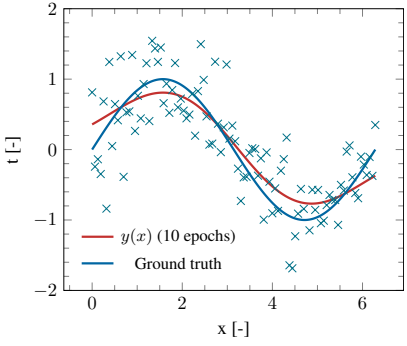


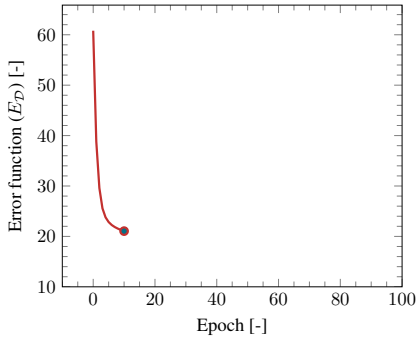


Stochastic Gradient Descent

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 $N_{\mathcal{B}} = 10$

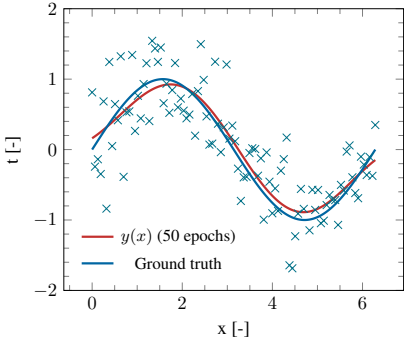


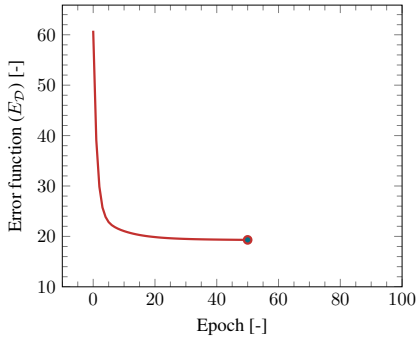


Stochastic Gradient Descent

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 $N_{\mathcal{B}} = 10$

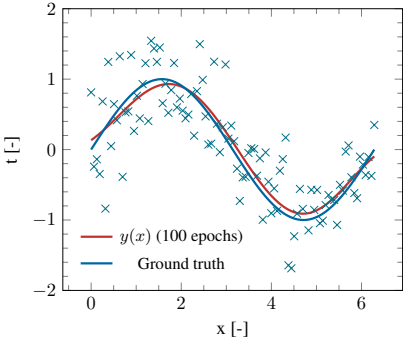


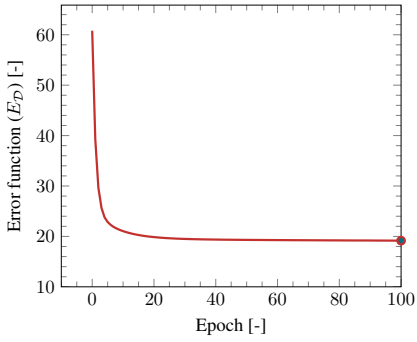


Stochastic Gradient Descent

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 $N_{\mathcal{B}} = 10$

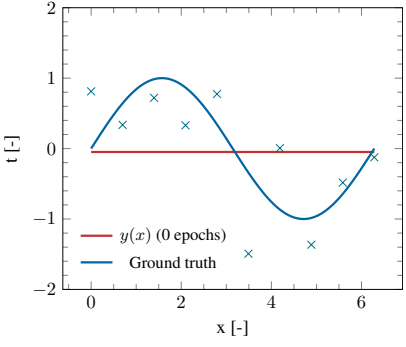


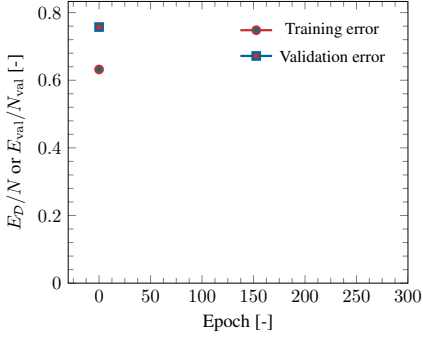


Stochastic Gradient Descent

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

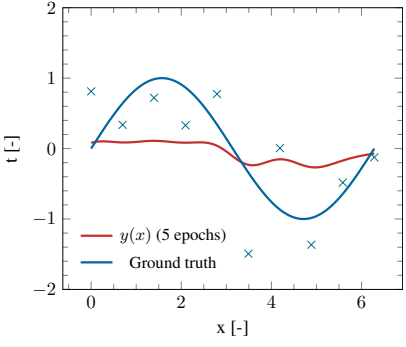


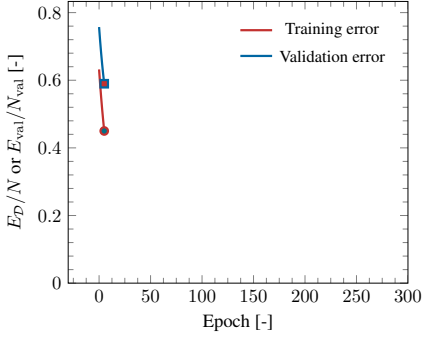


Stochastic Gradient Descent

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

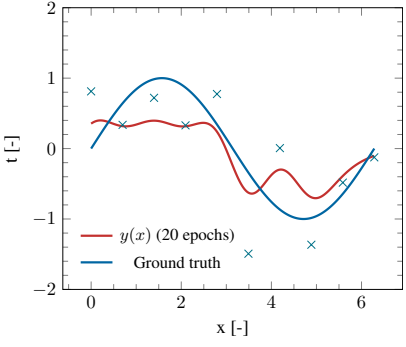


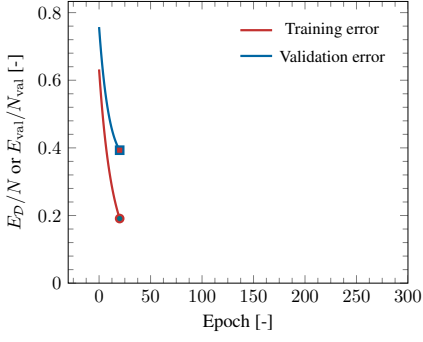


Stochastic Gradient Descent

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

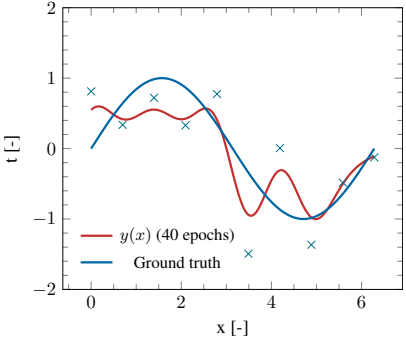


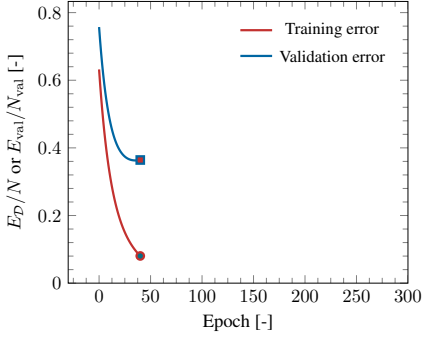


Stochastic Gradient Descent

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

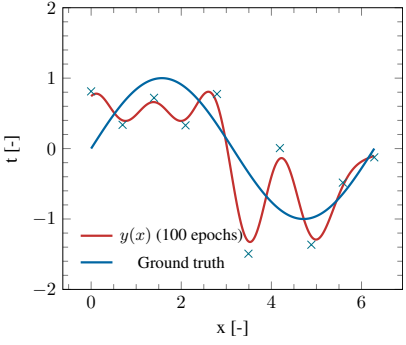


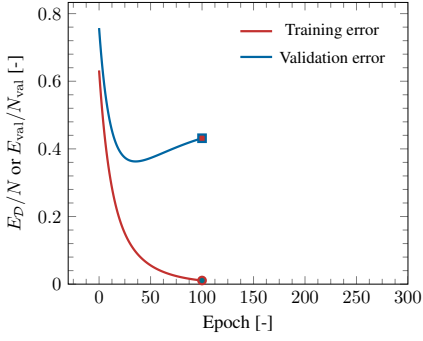


Stochastic Gradient Descent

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

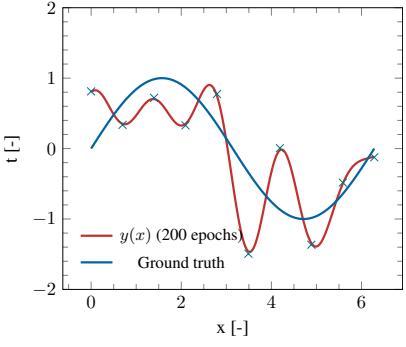


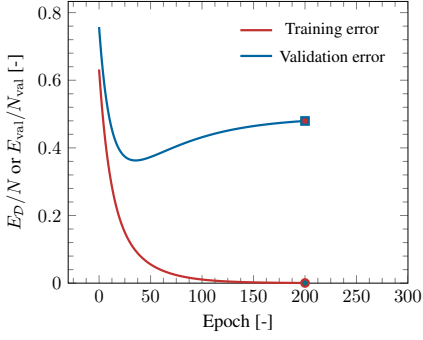


Stochastic Gradient Descent

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

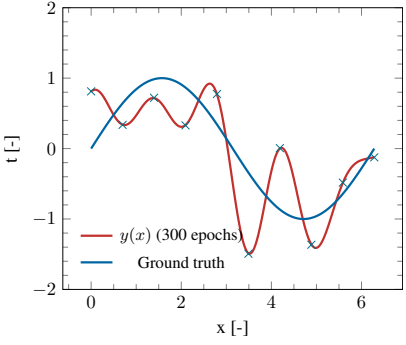


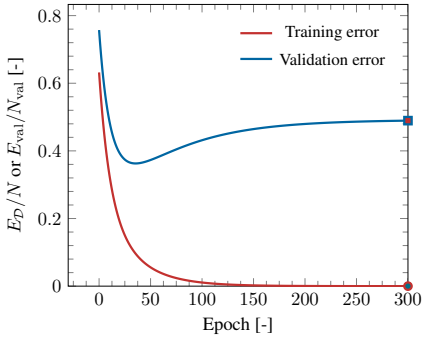


Stochastic Gradient Descent

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



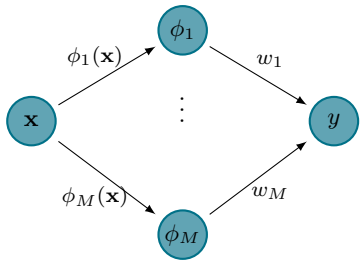


Adaptive basis functions

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

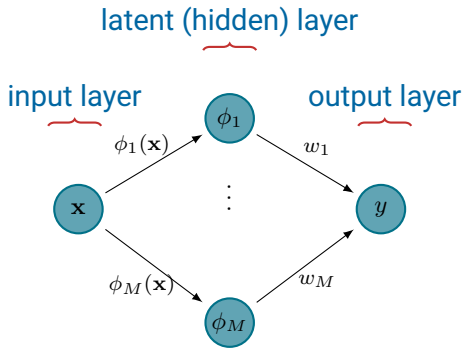
$$y = \phi_1(\mathbf{x})w_1 + \phi_2(\mathbf{x})w_2 + \cdots + \phi_M(\mathbf{x})w_M$$



Adaptive basis functions

For now, only half of the model is trainable:

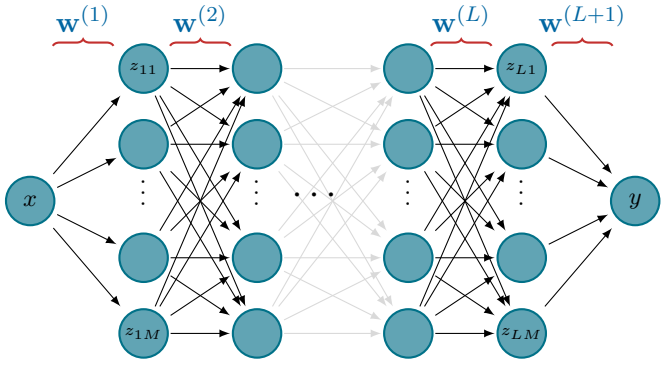
- Input to hidden encoding ($\phi_1 \cdots \phi_M$) fixed, hidden to output decoding ($\mathbf{w}$) trained
- What if we could also train the first half?



Artificial Neural Networks

Replacing basis functions by several layers of nonlinear transformations:

- Neural Network: layers of neurons linked by weighted synaptic connections
- Computing gradients becomes more complex, but all layers now have trainable weights



Neural Networks – Activation functions

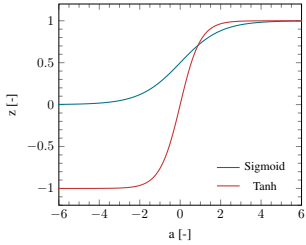
For a given neuron, forward propagation happens in two steps:

- A linear combination of values from the previous layer:

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

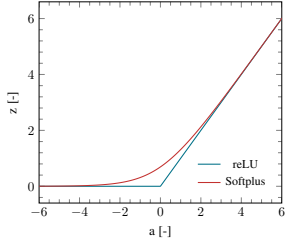
- A nonlinear transformation with an activation function:

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



Choosing the activation function:

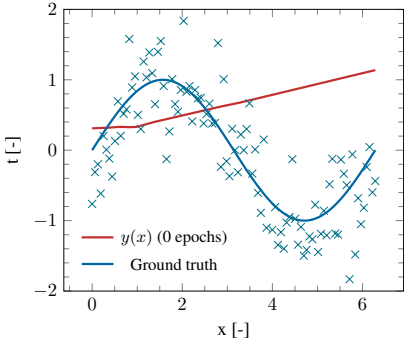
- Application dependent
- Can be seen as a hyperparameter

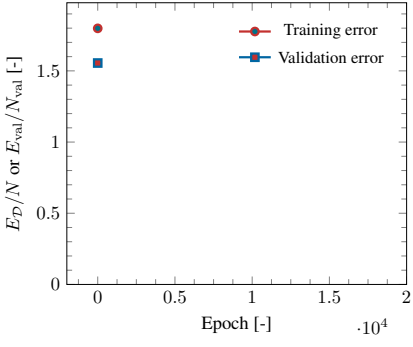


Neural Networks – Example

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

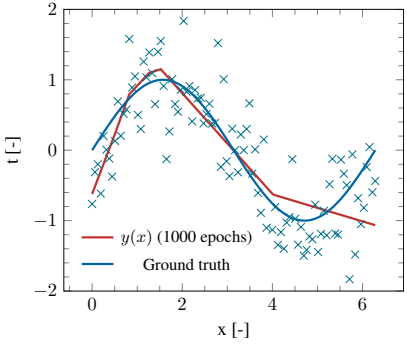


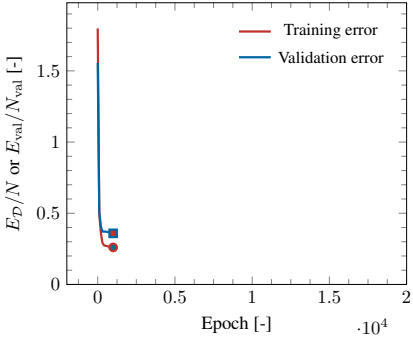


Neural Networks – Example

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

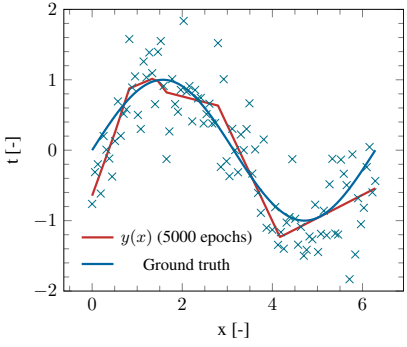


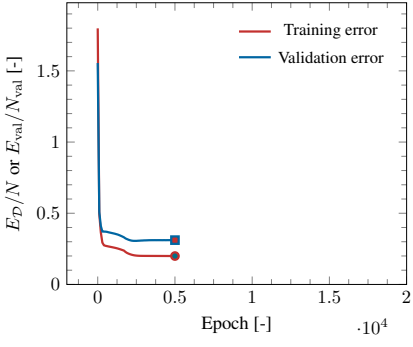


Neural Networks – Example

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

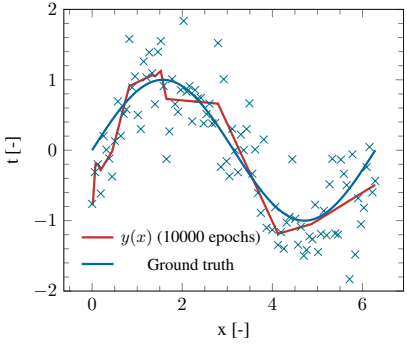


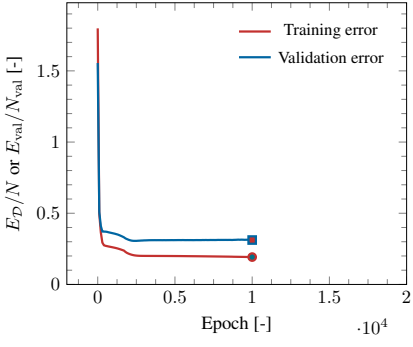


Neural Networks – Example

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

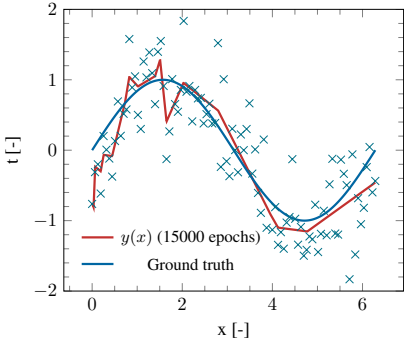


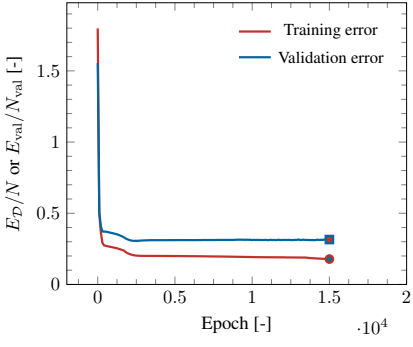


Neural Networks – Example

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

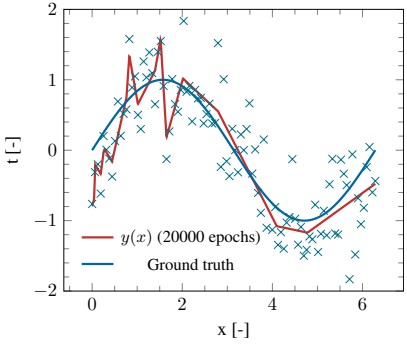


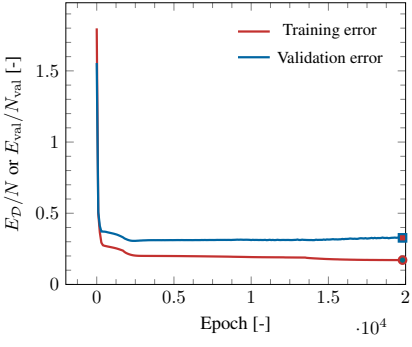


Neural Networks – Example

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





Break

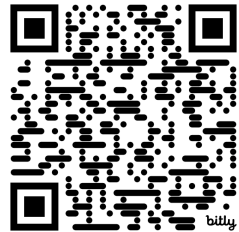
Let us take a 20-minute break

For later:

- Read the content in the book pages
- Play with a bunch of other interactive plots
- Look at Bayesian linear regression and try the exercises

Up next:

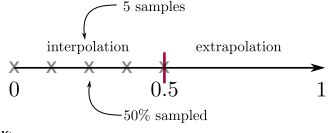
- The curse of dimensionality
- Breaking the curse — Bayesian ML and inductive biases
- Gaussian Processes for regression

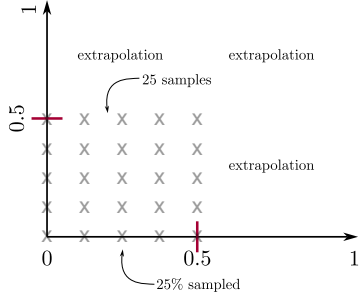


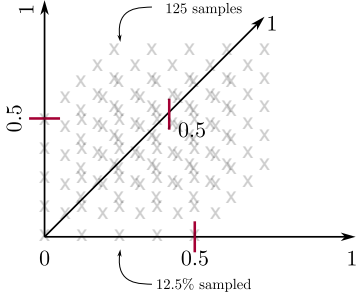
The Curse of Dimensionality

When building ML models, we rely on covering our feature space well enough. However...

- We need exponentially more samples to keep up as we go to higher dimensions







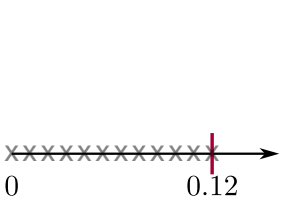
The Curse of Dimensionality

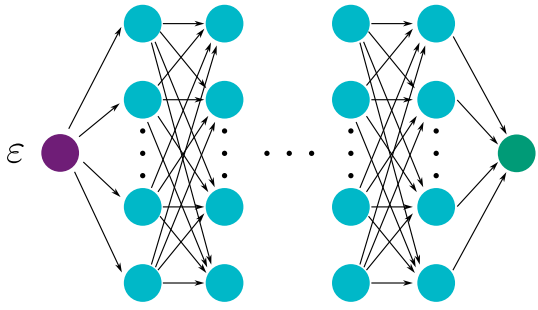
When building ML models, we rely on covering our feature space well enough. However...

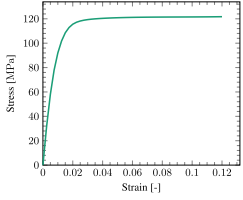
- We need exponentially more samples to keep up as we go to higher dimensions

This is a huge issue for all sorts of tasks:

- From surrogate modeling to image processing







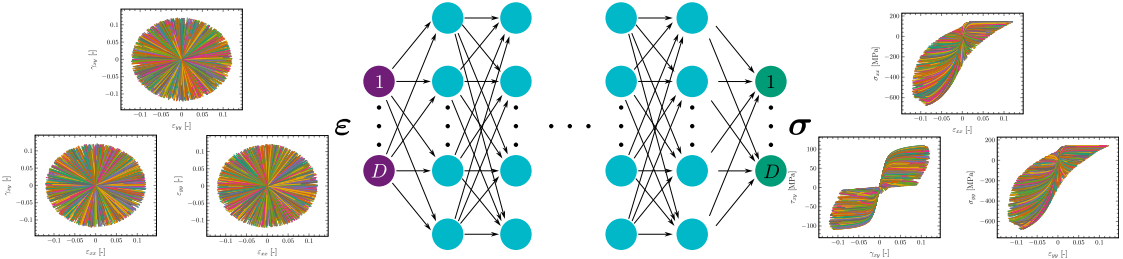
The Curse of Dimensionality

When building ML models, we rely on covering our feature space well enough. However...

- We need exponentially more samples to keep up as we go to higher dimensions

This is a huge issue for all sorts of tasks:

- From surrogate modeling to image processing



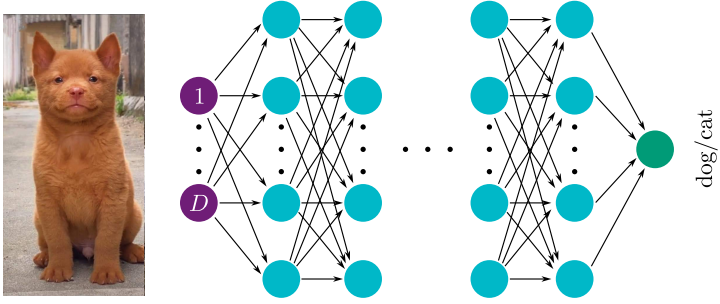
The Curse of Dimensionality

When building ML models, we rely on covering our feature space well enough. However...

- We need exponentially more samples to keep up as we go to higher dimensions

This is a huge issue for all sorts of tasks:

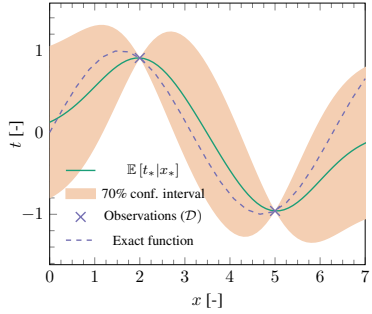
- From surrogate modeling to image processing



Breaking the curse

Bad news: fully breaking the curse is impossible. But with some extra tricks it can be alleviated:

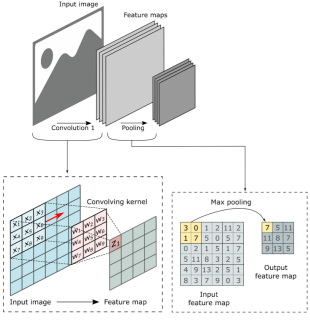
- Introduce bias through prior beliefs ⇒ Bayesian ML (up next)
- Assume Euclidean or non-Euclidean spatial bias ⇒ Convolutional NNs, Graph NNs
- Explain patterns in lower dimensions with dimensionality reduction ⇒ PCA, Autoencoders
- Assume the data is explained by latent time dependencies ⇒ 1D CNNs, Recurrent NNs
- Introduce bias coming from physics knowledge ⇒ PINNs, hybrid models

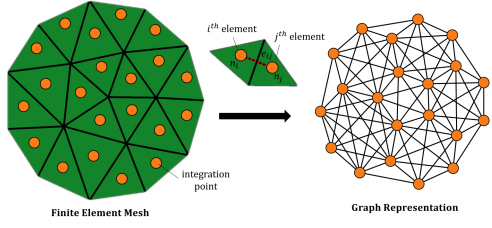


Breaking the curse

Bad news: fully breaking the curse is impossible. But with some extra tricks it can be alleviated:

- Introduce bias through prior beliefs ⇒ Bayesian ML (up next)
- Assume Euclidean or non-Euclidean spatial bias ⇒ Convolutional NNs, Graph NNs
- Explain patterns in lower dimensions with dimensionality reduction ⇒ PCA, Autoencoders
- Assume the data is explained by latent time dependencies ⇒ 1D CNNs, Recurrent NNs
- Introduce bias coming from physics knowledge ⇒ PINNs, hybrid models





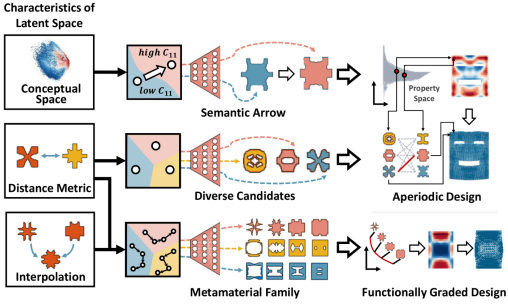
[Alzubaidi et al (2022), Rock Mech Rock Eng **55**:3719-3734]

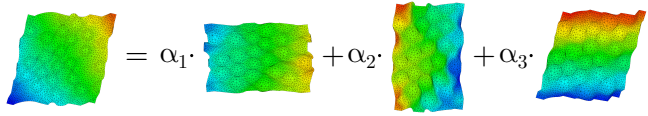
[Vlassis and Sun (2022), arXiv:2208.00246v1]

Breaking the curse

Bad news: fully breaking the curse is impossible. But with some extra tricks it can be alleviated:

- Introduce bias through prior beliefs ⇒ Bayesian ML (up next)
- Assume Euclidean or non-Euclidean spatial bias ⇒ Convolutional NNs, Graph NNs
- Explain patterns in lower dimensions with dimensionality reduction ⇒ PCA, Autoencoders
- Assume the data is explained by latent time dependencies ⇒ 1D CNNs, Recurrent NNs
- Introduce bias coming from physics knowledge ⇒ PINNs, hybrid models





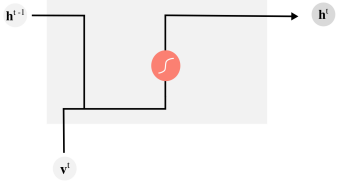
[Wang et al (2020), CMAME **372**:113377]

[Rocha et al (2019), CMAME **345**:644-670]

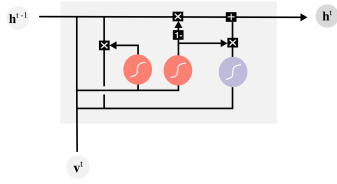
Breaking the curse

Bad news: fully breaking the curse is impossible. But with some extra tricks it can be alleviated:

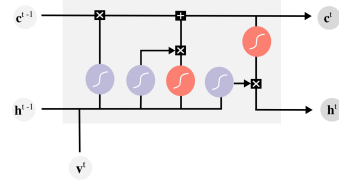
- Introduce bias through prior beliefs ⇒ Bayesian ML (up next)
- Assume Euclidean or non-Euclidean spatial bias ⇒ Convolutional NNs, Graph NNs
- Explain patterns in lower dimensions with dimensionality reduction ⇒ PCA, Autoencoders
- Assume the data is explained by latent time dependencies ⇒ 1D CNNs, Recurrent NNs
- Introduce bias coming from physics knowledge ⇒ PINNs, hybrid models



Classical RNN



Gated Recurrent Unit (GRU)



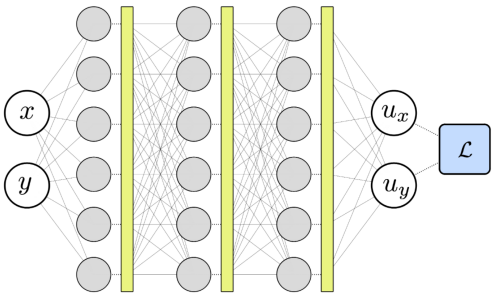
Long Short Term Memory (LSTM)

[Maia et al (2022), CMAME **407**:115934]

Breaking the curse

Bad news: fully breaking the curse is impossible. But with some extra tricks it can be alleviated:

- Introduce bias through prior beliefs ⇒ Bayesian ML (up next)
- Assume Euclidean or non-Euclidean spatial bias ⇒ Convolutional NNs, Graph NNs
- Explain patterns in lower dimensions with dimensionality reduction ⇒ PCA, Autoencoders
- Assume the data is explained by latent time dependencies ⇒ 1D CNNs, Recurrent NNs
- Introduce bias coming from physics knowledge ⇒ PINNs, hybrid models



[Haghighat et al (2020), arXiv:2003.02751v2]

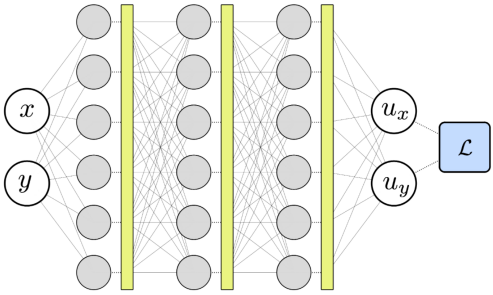
$$\sigma_{ij,j} + f_i = 0$$
$$\sigma_{ij} = \lambda \delta_{ij} \varepsilon_{kk} + 2\mu \varepsilon_{ij}$$
$$\varepsilon_{ij} = \tfrac{1}{2}\left(u_{i,j} + u_{j,i}\right)$$
$$\mathcal{L} = |\sigma_{ij,j} + f_i|_\Omega$$

Collocation loss

Breaking the curse

Bad news: fully breaking the curse is impossible. But with some extra tricks it can be alleviated:

- Introduce bias through prior beliefs ⇒ Bayesian ML (up next)
- Assume Euclidean or non-Euclidean spatial bias ⇒ Convolutional NNs, Graph NNs
- Explain patterns in lower dimensions with dimensionality reduction ⇒ PCA, Autoencoders
- Assume the data is explained by latent time dependencies ⇒ 1D CNNs, Recurrent NNs
- Introduce bias coming from physics knowledge ⇒ PINNs, hybrid models



[Haghighat et al (2020), arXiv:2003.02751v2]

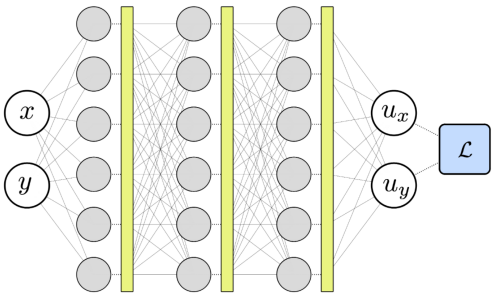
$$\sigma_{ij,j} + f_i = 0$$
$$\sigma_{ij} = \lambda \delta_{ij} \varepsilon_{kk} + 2\mu \varepsilon_{ij}$$
$$\varepsilon_{ij} = \tfrac{1}{2}\left(u_{i,j} + u_{j,i}\right)$$
$$\mathcal{L} = |\sigma_{ij,j} + f_i|_\Omega + |u - u^*|_{\Gamma_u}$$

Collocation loss — Dirichlet loss

Breaking the curse

Bad news: fully breaking the curse is impossible. But with some extra tricks it can be alleviated:

- Introduce bias through prior beliefs ⇒ Bayesian ML (up next)
- Assume Euclidean or non-Euclidean spatial bias ⇒ Convolutional NNs, Graph NNs
- Explain patterns in lower dimensions with dimensionality reduction ⇒ PCA, Autoencoders
- Assume the data is explained by latent time dependencies ⇒ 1D CNNs, Recurrent NNs
- Introduce bias coming from physics knowledge ⇒ PINNs, hybrid models



[Haghighat et al (2020), arXiv:2003.02751v2]

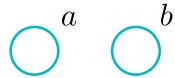
$$\sigma_{ij,j} + f_i = 0$$
$$\sigma_{ij} = \lambda \delta_{ij} \varepsilon_{kk} + 2\mu \varepsilon_{ij}$$
$$\varepsilon_{ij} = \tfrac{1}{2}\left(u_{i,j} + u_{j,i}\right)$$
$$\mathcal{L} = |\sigma_{ij,j} + f_i|_\Omega + |u - u^*|_{\Gamma_u} + |\sigma_{ij} - \sigma^*_{ij}|_{\Gamma_\sigma}$$

Collocation loss | Dirichlet loss | Neumann loss

Interlude — Probabilistic Graphs

Graphs with more than one variable imply a joint probability distribution:

- Two independent variables:

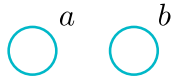


$$p(a, b) = p(a)p(b)$$

Interlude — Probabilistic Graphs

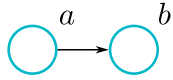
Graphs with more than one variable imply a joint probability distribution:

- Two independent variables:

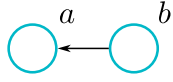


$$p(a, b) = p(a)p(b)$$

- We can use arrows to indicate dependency, causality or correlation through the Product Rule:



$$p(a, b) = p(a)p(b|a)$$



$$p(a, b) = p(b)p(a|b)$$

Interlude — Bayes' Theorem

Consider a model with two variables x and y. We can use the Product Rule of probability to write:

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

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

Interlude — Bayes' Theorem

Consider a model with two variables x and y. We can use the Product Rule of probability to write:

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

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

Substituting one expression into the other we get the very important result:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

Interlude — Bayes' Theorem

Consider a model with two variables x and y. We can use the Product Rule of probability to write:

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

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

Substituting one expression into the other we get the very important result:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

We can use Bayes' Theorem to model and update beliefs and uncertainties:

- The prior $p(y)$ expresses what we know about y before observing x
- The likelihood $p(x|y)$ models how much y can explain x
- The posterior $p(y|x)$ expresses how the knowledge about y changed after observing x

Interlude — Bayes' Theorem

Consider a model with two variables x and y. We can use the Product Rule of probability to write:

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

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

Substituting one expression into the other we get the very important result:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

We can use Bayes' Theorem to model and update beliefs and uncertainties:

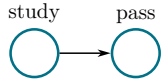
- The prior $p(y)$ expresses what we know about y before observing x
- The likelihood $p(x|y)$ models how much y can explain x
- The posterior $p(y|x)$ expresses how the knowledge about y changed after observing x

Bayes' Theorem states our willingness to change what we know about y after observing x:

$$\text{posterior} \propto \text{likelihood} \times \text{prior}$$

Bayes' Theorem — Example

Let us consider a simple probabilistic model for predicting student performance:

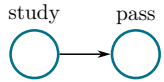


prior

$$p(\text{study}, \text{pass}) = p(\text{study})p(\text{pass}|\text{study})$$

Bayes' Theorem — Example

Let us consider a simple probabilistic model for predicting student performance:



prior

$$p(\text{study}, \text{pass}) = p(\text{study})p(\text{pass}|\text{study})$$

Let us set some reasonable expectations:

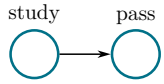
$$p(S = 1) = 0.6 \qquad p(S = 0) = ?$$

$$p(P = 1|S = 1) = 0.9 \quad p(P = 1|S = 0) = 0.3$$

$$p(P = 0|S = 1) = ? \quad p(P = 0|S = 0) = ?$$

Bayes' Theorem — Example

Let us consider a simple probabilistic model for predicting student performance:



prior

$$p(\text{study}, \text{pass}) = p(\text{study})p(\text{pass}|\text{study})$$

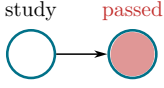
Let us set some reasonable expectations:

$$p(S = 1) = 0.6 \qquad p(S = 0) = ?$$

$$p(P = 1|S = 1) = 0.9 \quad p(P = 1|S = 0) = 0.3$$

$$p(P = 0|S = 1) = ? \quad p(P = 0|S = 0) = ?$$

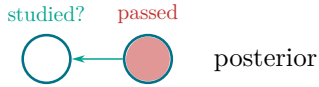
Now suppose we observe someone passed. How certain should we be that they studied?



observation

Bayes' Theorem — Example

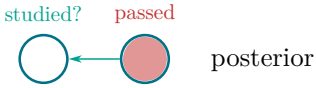
Keeping our prior assumptions ignores important evidence. We use Bayes' Theorem instead:



$$p(S = 1|P = 1) = \frac{p(P = 1|S = 1)p(S = 1)}{p(P = 1)}$$

Bayes' Theorem — Example

Keeping our prior assumptions ignores important evidence. We use Bayes' Theorem instead:



$$p(S=1|P=1) = \frac{p(P=1|S=1)p(S=1)}{p(P=1)}$$

$p(P=1,S) = p(S)p(P=1|S)$

Get together with someone and use the values below (10 minutes):

- Use the Sum Rule to compute the marginal $p(P=1)$ from the joint $P(P=1,S)$
- Use the values of $p(P=1|S=1)$ and $p(S=1)$ from below
- Put it all together above and compute the posterior $p(S=1|P=1)$

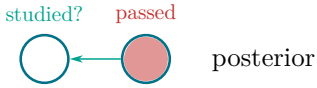
$$p(S=1) = 0.6 \qquad p(S=0) = 0.4$$

$$p(P=1|S=1) = 0.9 \quad p(P=1|S=0) = 0.3$$

$$p(P=0|S=1) = 0.1 \quad p(P=0|S=0) = 0.7$$

Bayes' Theorem — Example

Keeping our prior assumptions ignores important evidence. We use Bayes' Theorem instead:



$$p(S = 1|P = 1) = \frac{p(P = 1|S = 1)p(S = 1)}{p(P = 1)}$$

Get together with someone and use the values below (10 minutes):

- Use the Sum Rule to compute the marginal $p(P = 1)$ from the joint $P(P = 1, S)$
- Use the values of $p(P = 1|S = 1)$ and $p(S = 1)$ from below
- Put it all together above and compute the posterior $p(S = 1|P = 1)$

How much did our knowledge shift? How does this compare to the prior? Does it make sense?

$$p(S = 1|P = 1) = \frac{0.9 \cdot 0.6}{0.9 \cdot 0.6 + 0.3 \cdot 0.4} = 0.818$$

Bayesian linear regression

Finally, let us use Bayes' Theorem properly and end up with a distribution for $\mathbf{w}$:

$$p(\mathbf{w}|\mathbf{t}) = \frac{p(\mathbf{t}|\mathbf{w})p(\mathbf{w})}{p(\mathbf{t})} \quad \text{with} \quad p(\mathbf{w}) = \mathcal{N}\left(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}\right) \text{ and } p(\mathbf{t}|\mathbf{w}) = \mathcal{N}\left(\mathbf{t}|\mathbf{\Phi}\mathbf{w}, \beta^{-1}\mathbf{I}\right)$$

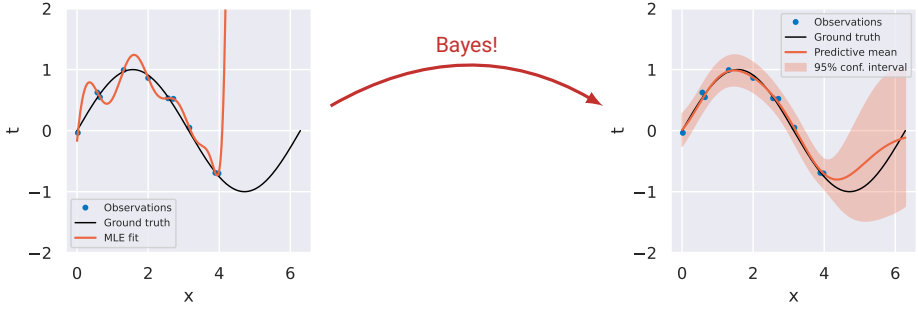
Bayesian linear regression

Finally, let us use Bayes' Theorem properly and end up with a distribution for $\mathbf{w}$:

$$p(\mathbf{w}|\mathbf{t}) = \frac{p(\mathbf{t}|\mathbf{w})p(\mathbf{w})}{p(\mathbf{t})} \quad \text{with} \quad p(\mathbf{w}) = \mathcal{N}\left(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}\right) \text{ and } p(\mathbf{t}|\mathbf{w}) = \mathcal{N}\left(\mathbf{t}|\mathbf{\Phi}\mathbf{w}, \beta^{-1}\mathbf{I}\right)$$

The result takes the simple form:

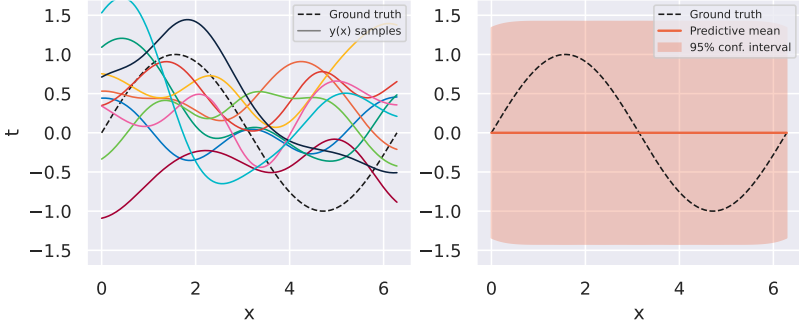
$$p(\mathbf{w}|\mathbf{t}) = \mathcal{N}\left(\mathbf{w}|\mathbf{m}, \mathbf{S}\right) \quad \text{with} \quad \mathbf{m} = \beta\mathbf{S}\mathbf{\Phi}^{\mathrm{T}}\mathbf{t} \text{ and } \mathbf{S}^{-1} = \alpha\mathbf{I} + \beta\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}$$



Sampling models from the posterior

Since $\mathbf{w}$ is probabilistic, we have a bag of models at our disposal:

- Take a sample $\widetilde{\mathbf{w}}$ from $\mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$
- Compute $y(\mathbf{x}, \widetilde{\mathbf{w}})$ for our whole range of $\mathbf{x}$

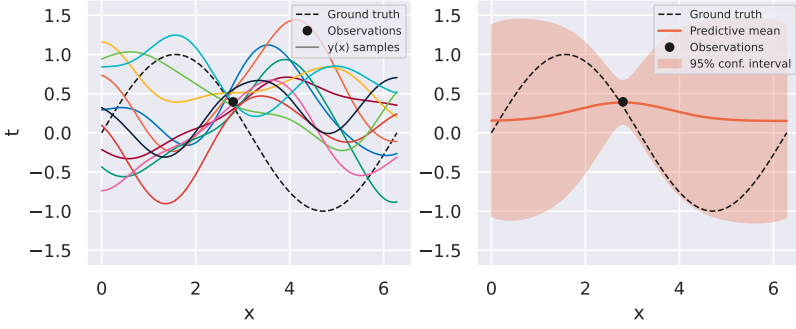


Sampling models from the posterior

Since $\mathbf{w}$ is probabilistic, we have a bag of models at our disposal:

- Take a sample $\widetilde{\mathbf{w}}$ from $\mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$
- Compute $y(\mathbf{x}, \widetilde{\mathbf{w}})$ for our whole range of $\mathbf{x}$

After conditioning we can do the same for the posterior, and that can evolve as more samples are added:

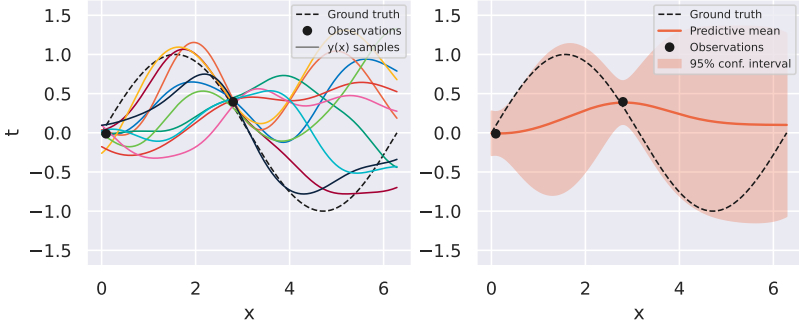


Sampling models from the posterior

Since $\mathbf{w}$ is probabilistic, we have a bag of models at our disposal:

- Take a sample $\widetilde{\mathbf{w}}$ from $\mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$
- Compute $y(\mathbf{x}, \widetilde{\mathbf{w}})$ for our whole range of $\mathbf{x}$

After conditioning we can do the same for the posterior, and that can evolve as more samples are added:

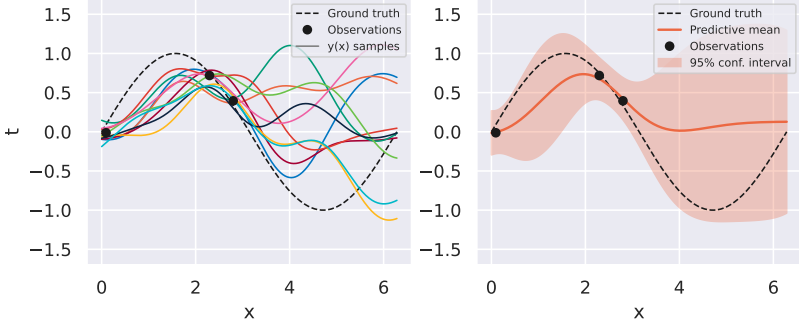


Sampling models from the posterior

Since $\mathbf{w}$ is probabilistic, we have a bag of models at our disposal:

- Take a sample $\widetilde{\mathbf{w}}$ from $\mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$
- Compute $y(\mathbf{x}, \widetilde{\mathbf{w}})$ for our whole range of $\mathbf{x}$

After conditioning we can do the same for the posterior, and that can evolve as more samples are added:

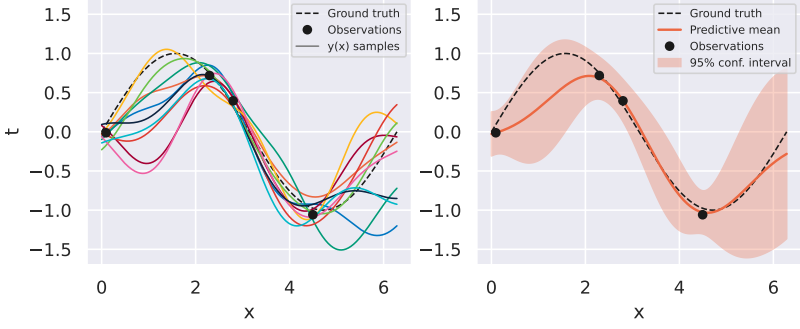


Sampling models from the posterior

Since $\mathbf{w}$ is probabilistic, we have a bag of models at our disposal:

- Take a sample $\widetilde{\mathbf{w}}$ from $\mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$
- Compute $y(\mathbf{x}, \widetilde{\mathbf{w}})$ for our whole range of $\mathbf{x}$

After conditioning we can do the same for the posterior, and that can evolve as more samples are added:

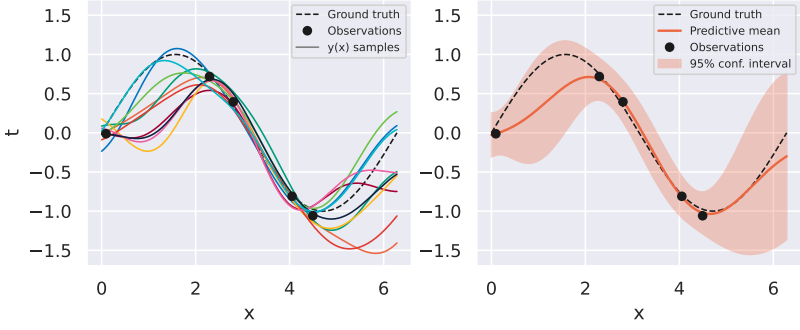


Sampling models from the posterior

Since $\mathbf{w}$ is probabilistic, we have a bag of models at our disposal:

- Take a sample $\widetilde{\mathbf{w}}$ from $\mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$
- Compute $y(\mathbf{x}, \widetilde{\mathbf{w}})$ for our whole range of $\mathbf{x}$

After conditioning we can do the same for the posterior, and that can evolve as more samples are added:



From weights to functions

Now we gather all values of y for a single realization of $\mathbf{w}$ into a vector:

$$\mathbf{y} = \mathbf{\Phi}\mathbf{w} \quad \text{with} \quad \mathbf{\Phi}_{nk} = \phi_k(\mathbf{x}_n)$$

From weights to functions

Now we gather all values of y for a single realization of $\mathbf{w}$ into a vector:

$$\mathbf{y} = \mathbf{\Phi}\mathbf{w} \quad \text{with} \quad \mathbf{\Phi}_{nk} = \phi_k(\mathbf{x}_n)$$

Because $\mathbf{\Phi}$ is a linear deterministic operator, $\mathbf{y}$ is also Gaussian:

$$p(\mathbf{y}) = \mathcal{N}\left(\mathbf{y}|\mathbf{0}, \alpha^{-1}\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\right)$$

- Note that $\mathbf{w}$ is now gone. This is now officially a non-parametric model!
- Instead of sampling from $p(\mathbf{w})$, we can now sample models from $p(\mathbf{y})$

From weights to functions

Now we gather all values of y for a single realization of $\mathbf{w}$ into a vector:

$$\mathbf{y} = \mathbf{\Phi}\mathbf{w} \quad \text{with} \quad \mathbf{\Phi}_{nk} = \phi_k(\mathbf{x}_n)$$

Because $\mathbf{\Phi}$ is a linear deterministic operator, $\mathbf{y}$ is also Gaussian:

$$p(\mathbf{y}) = \mathcal{N}\left(\mathbf{y}|\mathbf{0}, \alpha^{-1}\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}\right)$$

- Note that $\mathbf{w}$ is now gone. This is now officially a non-parametric model!
- Instead of sampling from $p(\mathbf{w})$, we can now sample models from $p(\mathbf{y})$

We can write the above in a more compact way:

- Define a kernel that relates two values of $\mathbf{x}$ and returns a scalar:

$$k(\mathbf{x}, \mathbf{x}') = \frac{1}{\alpha}\boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}')$$

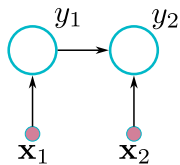
- Express $p(\mathbf{y})$ in terms of the kernel:

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K}) \quad \text{with} \quad K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$$

From weights to functions

As always, we can represent this model with a graph

- Starting with just a few variables

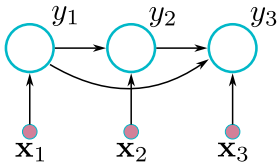


$$p(y_1, y_2) = \mathcal{N}\left(\mathbf{y}_{12} \mid \begin{bmatrix} m(\mathbf{x}_1) \\ m(\mathbf{x}_2) \end{bmatrix}, \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) \end{bmatrix}\right) \equiv \mathcal{N}\left(\mathbf{y}_{12} \mid \mathbf{m}_{12}, \mathbf{K}_{12}\right)$$

From weights to functions

As always, we can represent this model with a graph

- Starting with just a few variables

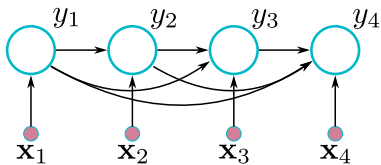


$$p(\mathbf{y}_{12}, y_3) = \mathcal{N}\left(\mathbf{y}_{123} \mid \begin{bmatrix}\mathbf{m}_{12}\\ m(\mathbf{x}_3)\end{bmatrix}, \begin{bmatrix}\mathbf{K}_{12} & k(\mathbf{x}_1,\mathbf{x}_3)\\ & k(\mathbf{x}_2,\mathbf{x}_3)\\ k(\mathbf{x}_3,\mathbf{x}_1) \quad k(\mathbf{x}_3,\mathbf{x}_2) & k(\mathbf{x}_3,\mathbf{x}_3)\end{bmatrix}\right) \equiv \mathcal{N}\left(\mathbf{y}_{123} \mid \mathbf{m}_{123}, \mathbf{K}_{123}\right)$$

From weights to functions

As always, we can represent this model with a graph

- Starting with just a few variables

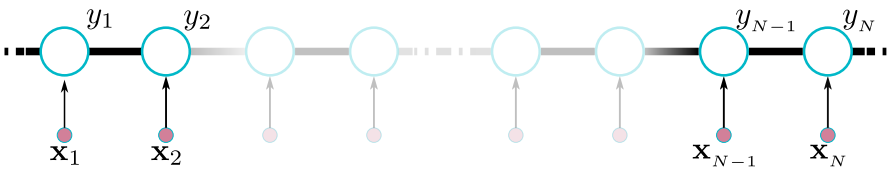


$$p(\mathbf{y}_{123}, y_4) = \mathcal{N}\left(\mathbf{y}_{1234} \mid \begin{bmatrix}\mathbf{m}_{123} \\ m(\mathbf{x}_4)\end{bmatrix}, \begin{bmatrix} & \mathbf{K}_{123} & & k(\mathbf{x}_1, \mathbf{x}_4) \\ & & & k(\mathbf{x}_2, \mathbf{x}_4) \\ & & & k(\mathbf{x}_3, \mathbf{x}_4) \\ k(\mathbf{x}_4, \mathbf{x}_1) & k(\mathbf{x}_4, \mathbf{x}_2) & k(\mathbf{x}_4, \mathbf{x}_3) & k(\mathbf{x}_4, \mathbf{x}_4) \end{bmatrix}\right) \equiv \mathcal{N}\left(\mathbf{y}_{1234} \mid \mathbf{m}_{1234}, \mathbf{K}_{1234}\right)$$

From weights to functions

As always, we can represent this model with a graph

- Starting with just a few variables
- We can generalize this for any number of nodes. Assuming zero mean:

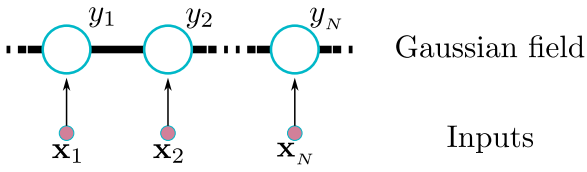


$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K})$$

From weights to functions

As always, we can represent this model with a graph

- Starting with just a few variables
- We can generalize this for any number of nodes. Assuming zero mean
- Finally, we can use a more compact version:



Gaussian Processes

Because we can include any number of variables in the joint, we can fully switch to a function view:

- A process is a probability density over functions
- Sampling a function means sampling an arbitrary number of points from it

Gaussian Processes

Because we can include any number of variables in the joint, we can fully switch to a function view:

- A process is a probability density over functions
- Sampling a function means sampling an arbitrary number of points from it

A Gaussian Process (GP) is a process dictated by joint Gaussian densities:

- Any number of points from a GP is jointly Gaussian

Gaussian Processes

Because we can include any number of variables in the joint, we can fully switch to a function view:

- A process is a probability density over functions
- Sampling a function means sampling an arbitrary number of points from it

A Gaussian Process (GP) is a process dictated by joint Gaussian densities:

- Any number of points from a GP is jointly Gaussian

A GP is fully described by a mean function and a covariance kernel:

$$y(\mathbf{x}) \sim \mathcal{GP}\left(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\right)$$

- The kernel dictates the correlation between function values
- Edge case: $k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \delta(\mathbf{x} - \mathbf{x}') \Rightarrow \mathbf{K} = \sigma_f^2 \mathbf{I}$ (white noise random walk)

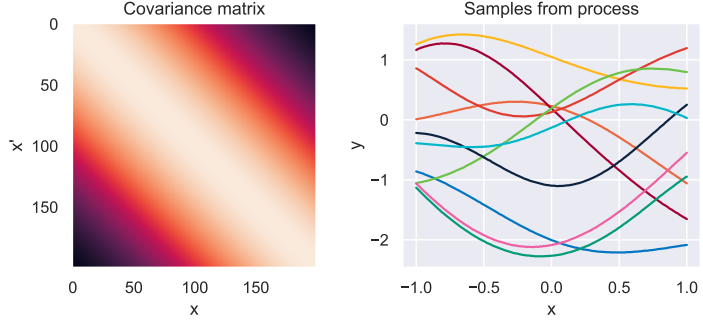
Kernel engineering — squared exponential

A very popular kernel in many applications, powerful and flexible:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right)$$

- Hyperparameters: process variance σ_f^2, length scale ℓ
- Equivalent to an RBF kernel with infinite (!) basis functions

process variance $\sigma_f^2 = 1$, length scale $\ell = 1$



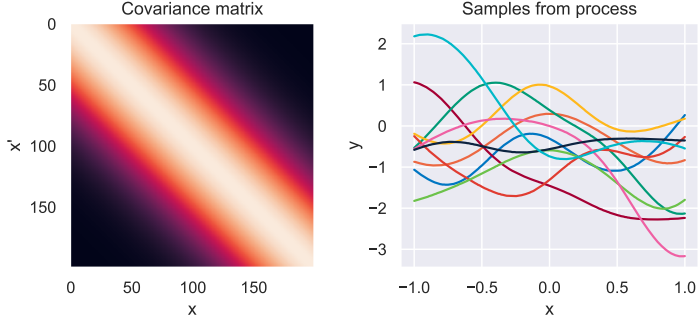
Kernel engineering — squared exponential

A very popular kernel in many applications, powerful and flexible:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right)$$

- Hyperparameters: process variance σ_f^2, length scale ℓ
- Equivalent to an RBF kernel with infinite (!) basis functions

process variance $\sigma_f^2 = 1$, length scale $\ell = 0.5$



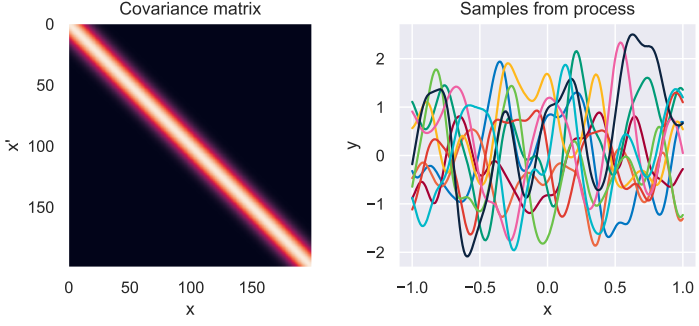
Kernel engineering — squared exponential

A very popular kernel in many applications, powerful and flexible:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right)$$

- Hyperparameters: process variance σ_f^2, length scale ℓ
- Equivalent to an RBF kernel with infinite (!) basis functions

process variance $\sigma_f^2 = 1$, length scale $\ell = 0.1$



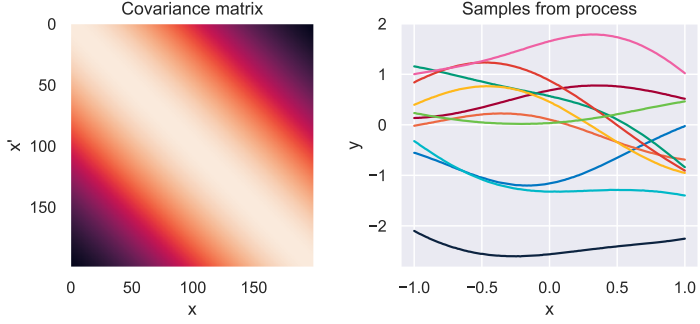
Kernel engineering — Matérn

A kernel for when differentiability is an issue:

$$k(\mathbf{x}, \mathbf{x}') = \frac{\sigma_f^2}{\Gamma(\nu)2^{\nu-1}} \left(\frac{\sqrt{2\nu}}{\ell} \|\mathbf{x} - \mathbf{x}'\| \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}}{\ell} \|\mathbf{x} - \mathbf{x}'\| \right)$$

- Hyperparameters: process variance σ_f^2, length scale ℓ, differentiability parameter ν

$\nu \to \infty$ (infinitely differentiable)



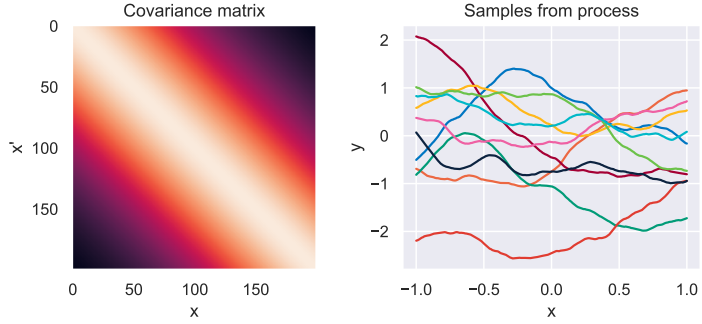
Kernel engineering — Matérn

A kernel for when differentiability is an issue:

$$k(\mathbf{x}, \mathbf{x}') = \frac{\sigma_f^2}{\Gamma(\nu)2^{\nu-1}} \left(\frac{\sqrt{2\nu}}{\ell} \|\mathbf{x} - \mathbf{x}'\| \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}}{\ell} \|\mathbf{x} - \mathbf{x}'\| \right)$$

- Hyperparameters: process variance σ_f^2, length scale ℓ, differentiability parameter ν

$\nu = 1.5$ (once differentiable)



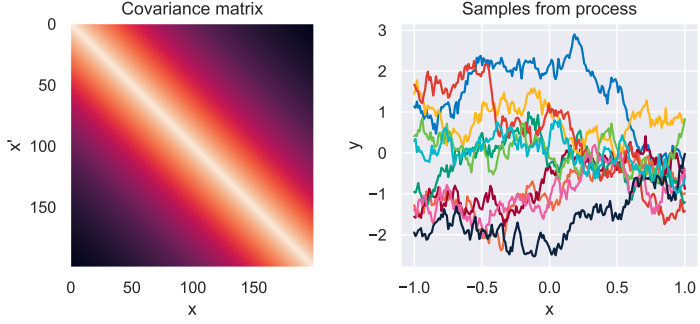
Kernel engineering — Matérn

A kernel for when differentiability is an issue:

$$k(\mathbf{x}, \mathbf{x}') = \frac{\sigma_f^2}{\Gamma(\nu)2^{\nu-1}} \left(\frac{\sqrt{2\nu}}{\ell} \|\mathbf{x} - \mathbf{x}'\| \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}}{\ell} \|\mathbf{x} - \mathbf{x}'\| \right)$$

- Hyperparameters: process variance σ_f^2, length scale ℓ, differentiability parameter ν

$\nu = 0.5$ (not differentiable)



Gaussian processes for regression

For regression, we need to observe some targets and predict for new inputs

- Assume targets are Gaussian-distributed around $y(\mathbf{x})$:

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

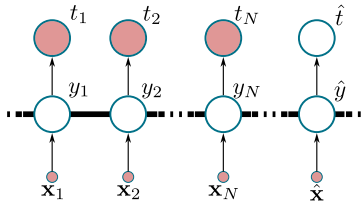
Gaussian processes for regression

For regression, we need to observe some targets and predict for new inputs

- Assume targets are Gaussian-distributed around $y(\mathbf{x})$:

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

- The targets are conditionally independent on $y(\mathbf{x})$ (i.i.d. assumption):



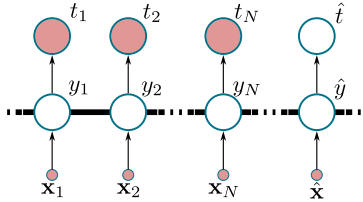
Gaussian processes for regression

For regression, we need to observe some targets and predict for new inputs

- Assume targets are Gaussian-distributed around $y(\mathbf{x})$:

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

- The targets are conditionally independent on $y(\mathbf{x})$ (i.i.d. assumption):



- From the standard expressions for the Gaussian, we have a marginal over the targets:

$$p(\mathbf{t}) = \int p(\mathbf{t}|\mathbf{y})p(\mathbf{y})\,\mathrm{d}\mathbf{y} = \mathcal{N}\left(\mathbf{t}|\mathbf{0}, K(\mathbf{X},\mathbf{X}) + \beta^{-1}\mathbf{I}\right)$$

GPs for regression — joint distribution

How to make new predictions? We need a joint between $\mathbf{t}$ and $\hat{t}$:

$$p(\mathbf{t}, \hat{t}) = \mathcal{N}\left(\mathbf{t}, \hat{t} \mid \mathbf{0}, \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) + \beta^{-1}\mathbf{I} & K(\mathbf{X}, \hat{\mathbf{x}}) \\ K(\hat{\mathbf{x}}, \mathbf{X}) & k(\hat{\mathbf{x}}, \hat{\mathbf{x}}) + \beta^{-1} \end{bmatrix}\right)$$

GPs for regression — joint distribution

How to make new predictions? We need a joint between $\mathbf{t}$ and $\hat{t}$:

$$p(\mathbf{t}, \hat{t}) = \mathcal{N}\left(\mathbf{t}, \hat{t} \mid \mathbf{0}, \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) + \beta^{-1}\mathbf{I} & K(\mathbf{X}, \hat{\mathbf{x}}) \\ K(\hat{\mathbf{x}}, \mathbf{X}) & k(\hat{\mathbf{x}}, \hat{\mathbf{x}}) + \beta^{-1} \end{bmatrix}\right)$$

Note the meaning and sizes of these submatrices:

- $K(\mathbf{X}, \mathbf{X}) + \beta^{-1}\mathbf{I}$: Correlation between training inputs, with added observation noise ($N \times N$)
- $K(\mathbf{X}, \hat{\mathbf{x}})$: Correlation between training targets and the new target ($N \times 1$)
- $K(\hat{\mathbf{x}}, \mathbf{X})$: Correlation between the new target and training targets ($1 \times N$)
- $k(\hat{\mathbf{x}}, \hat{\mathbf{x}})$: Variance of the new target, with added observation noise (1×1)

GPs for regression — joint distribution

How to make new predictions? We need a joint between $\mathbf{t}$ and $\hat{t}$:

$$p(\mathbf{t}, \hat{t}) = \mathcal{N}\left(\mathbf{t}, \hat{t} \mid \mathbf{0}, \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) + \beta^{-1}\mathbf{I} & K(\mathbf{X}, \hat{\mathbf{x}}) \\ K(\hat{\mathbf{x}}, \mathbf{X}) & k(\hat{\mathbf{x}}, \hat{\mathbf{x}}) + \beta^{-1} \end{bmatrix}\right)$$

Note the meaning and sizes of these submatrices:

- $K(\mathbf{X}, \mathbf{X}) + \beta^{-1}\mathbf{I}$: Correlation between training inputs, with added observation noise ($N \times N$)
- $K(\mathbf{X}, \hat{\mathbf{x}})$: Correlation between training targets and the new target ($N \times 1$)
- $K(\hat{\mathbf{x}}, \mathbf{X})$: Correlation between the new target and training targets ($1 \times N$)
- $k(\hat{\mathbf{x}}, \hat{\mathbf{x}})$: Variance of the new target, with added observation noise (1×1)

Note that we are predicting a single new value at $\hat{\mathbf{x}}$

- What would be the sizes of these matrices if we were predicting at 1000 new locations?

GPs for regression — predictive posterior

We have a joint but this is all still our a prior distribution:

- We still have a whole density $p(\mathbf{t})$ for the training targets

GPs for regression — predictive posterior

We have a joint but this is all still our a prior distribution:

- We still have a whole density $p(\mathbf{t})$ for the training targets

Since we know $\mathbf{t}$, we should update our prior using Bayes' Theorem:

$$p(\hat{t}|\mathbf{t}) = \frac{p(\mathbf{t}|\hat{t})p(\hat{t})}{p(\mathbf{t})}$$

GPs for regression — predictive posterior

We have a joint but this is all still our a prior distribution:

- We still have a whole density $p(\mathbf{t})$ for the training targets

Since we know $\mathbf{t}$, we should update our prior using Bayes' Theorem:

$$p(\hat{t}|\mathbf{t}) = \frac{p(\mathbf{t}|\hat{t})p(\hat{t})}{p(\mathbf{t})}$$

But given that we already have a joint Gaussian, we can just use our old conditioning expressions:

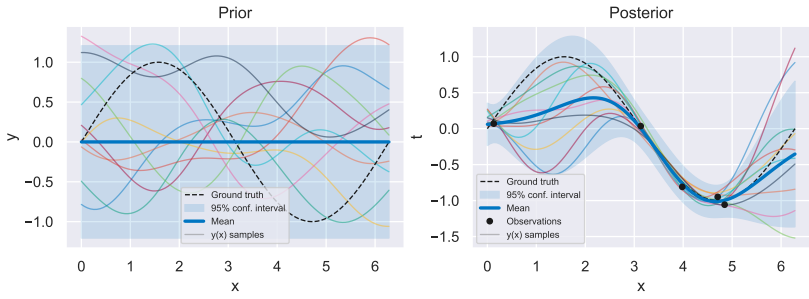
$$p(\hat{t}|\mathbf{t}) = \mathcal{N}\left(\hat{t}|\hat{m}, \hat{\sigma}^2\right)$$

$$\hat{m} = K\left(\hat{\mathbf{x}}, \mathbf{X}\right)\left[K\left(\mathbf{X}, \mathbf{X}\right) + \beta^{-1}\mathbf{I}\right]^{-1}\mathbf{t}$$

$$\hat{\sigma}^2 = k\left(\hat{\mathbf{x}}, \hat{\mathbf{x}}\right) - K\left(\hat{\mathbf{x}}, \mathbf{X}\right)\left[K\left(\mathbf{X}, \mathbf{X}\right) + \beta^{-1}\mathbf{I}\right]^{-1}K\left(\mathbf{X}, \hat{\mathbf{x}}\right) + \beta^{-1}$$

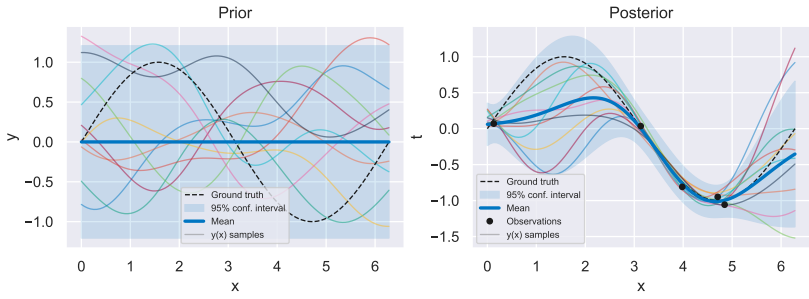
GPs for regression — example

Let us go back to our running sine-wave example. We observe $N = 5$ data points with a SE GP:



GPs for regression — example

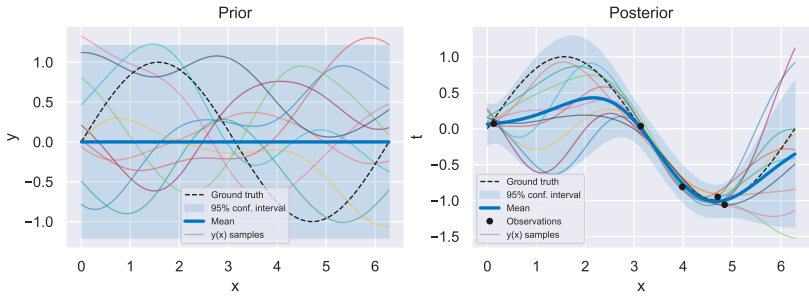
Let us go back to our running sine-wave example. We observe $N = 5$ data points with a SE GP:



- We have a prior and a posterior process, we can sample from both

GPs for regression — example

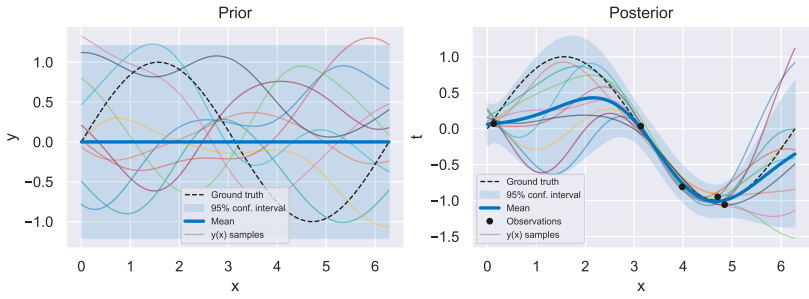
Let us go back to our running sine-wave example. We observe $N = 5$ data points with a SE GP:



- We have a prior and a posterior process, we can sample from both
- All posterior samples pass close to the observations (this closeness is proportional to β)

GPs for regression — example

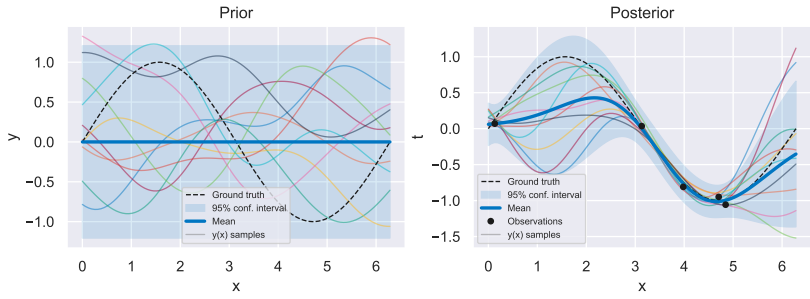
Let us go back to our running sine-wave example. We observe $N = 5$ data points with a SE GP:



- We have a prior and a posterior process, we can sample from both
- All posterior samples pass close to the observations (this closeness is proportional to β)
- Away from data the samples spread out and the variance increases

GPs for regression — example

Let us go back to our running sine-wave example. We observe $N = 5$ data points with a SE GP:



- We have a prior and a posterior process, we can sample from both
- All posterior samples pass close to the observations (this closeness is proportional to β)
- Away from data the samples spread out and the variance increases
- The model avoids overfitting even for this very small dataset

Learning and model selection

Hyperparameters can be learned without a validation set!

- We look at the marginal likelihood, which is quite easy in this case:

$$p(\mathbf{t}) = \mathcal{N}\left(\mathbf{t}|\mathbf{0}, K(\mathbf{X}, \mathbf{X}) + \beta^{-1}\mathbf{I}\right)$$

Learning and model selection

Hyperparameters can be learned without a validation set!

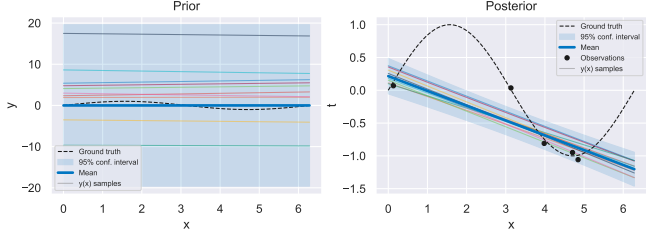
- We look at the marginal likelihood, which is quite easy in this case:

$$p(\mathbf{t}) = \mathcal{N}\left(\mathbf{t}|\mathbf{0}, K(\mathbf{X}, \mathbf{X}) + \beta^{-1}\mathbf{I}\right)$$

We take the $\log$ of this PDF and get to an expression we can maximize:

$$\ln p(\mathbf{t}) = -\frac{1}{2}\ln|\mathbf{K} + \beta^{-1}\mathbf{I}| - \frac{1}{2}\mathbf{t}^{\mathrm{T}}\left(\mathbf{K} + \beta^{-1}\mathbf{I}\right)^{-1}\mathbf{t} - \frac{N}{2}\ln(2\pi)$$

- This is called empirical Bayes or Type-2 MLE



Learning and model selection

Hyperparameters can be learned without a validation set!

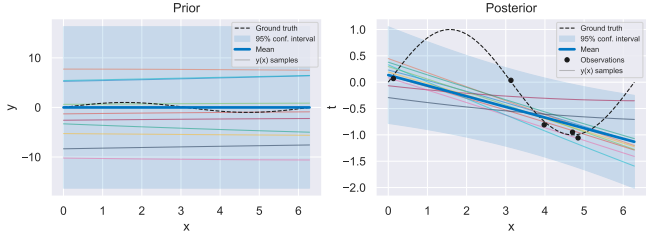
- We look at the marginal likelihood, which is quite easy in this case:

$$p(\mathbf{t}) = \mathcal{N}\left(\mathbf{t}|\mathbf{0}, K(\mathbf{X}, \mathbf{X}) + \beta^{-1}\mathbf{I}\right)$$

We take the $\log$ of this PDF and get to an expression we can maximize:

$$\ln p(\mathbf{t}) = -\frac{1}{2}\ln|\mathbf{K} + \beta^{-1}\mathbf{I}| - \frac{1}{2}\mathbf{t}^{\mathrm{T}}\left(\mathbf{K} + \beta^{-1}\mathbf{I}\right)^{-1}\mathbf{t} - \frac{N}{2}\ln(2\pi)$$

- This is called empirical Bayes or Type-2 MLE



Learning and model selection

Hyperparameters can be learned without a validation set!

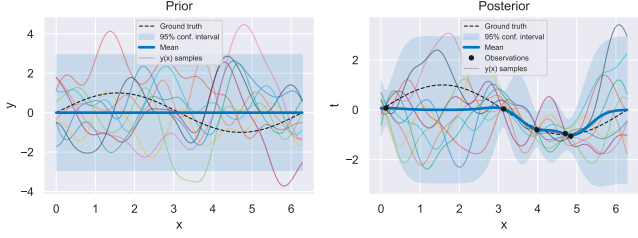
- We look at the marginal likelihood, which is quite easy in this case:

$$p(\mathbf{t}) = \mathcal{N}\left(\mathbf{t}|\mathbf{0}, K(\mathbf{X},\mathbf{X}) + \beta^{-1}\mathbf{I}\right)$$

We take the $\log$ of this PDF and get to an expression we can maximize:

$$\ln p(\mathbf{t}) = -\frac{1}{2}\ln|\mathbf{K} + \beta^{-1}\mathbf{I}| - \frac{1}{2}\mathbf{t}^{\mathrm{T}}\left(\mathbf{K} + \beta^{-1}\mathbf{I}\right)^{-1}\mathbf{t} - \frac{N}{2}\ln(2\pi)$$

- This is called empirical Bayes or Type-2 MLE



Learning and model selection

Hyperparameters can be learned without a validation set!

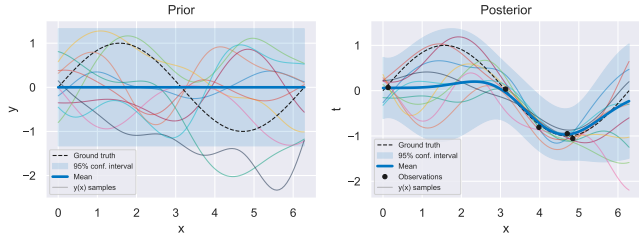
- We look at the marginal likelihood, which is quite easy in this case:

$$p(\mathbf{t}) = \mathcal{N}\left(\mathbf{t}|\mathbf{0}, K(\mathbf{X},\mathbf{X}) + \beta^{-1}\mathbf{I}\right)$$

We take the $\log$ of this PDF and get to an expression we can maximize:

$$\ln p(\mathbf{t}) = -\frac{1}{2}\ln|\mathbf{K} + \beta^{-1}\mathbf{I}| - \frac{1}{2}\mathbf{t}^{\mathrm{T}}\left(\mathbf{K} + \beta^{-1}\mathbf{I}\right)^{-1}\mathbf{t} - \frac{N}{2}\ln(2\pi)$$

- This is called empirical Bayes or Type-2 MLE



Learning and model selection

Hyperparameters can be learned without a validation set!

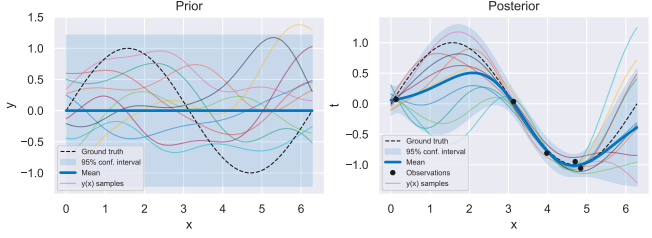
- We look at the marginal likelihood, which is quite easy in this case:

$$p(\mathbf{t}) = \mathcal{N}\left(\mathbf{t}|\mathbf{0}, K(\mathbf{X},\mathbf{X}) + \beta^{-1}\mathbf{I}\right)$$

We take the $\log$ of this PDF and get to an expression we can maximize:

$$\ln p(\mathbf{t}) = -\frac{1}{2}\ln|\mathbf{K} + \beta^{-1}\mathbf{I}| - \frac{1}{2}\mathbf{t}^{\mathrm{T}}\left(\mathbf{K} + \beta^{-1}\mathbf{I}\right)^{-1}\mathbf{t} - \frac{N}{2}\ln(2\pi)$$

- This is called empirical Bayes or Type-2 MLE



End of this part

Key takeaways:

- ML inherits much of its foundation from statistics
- Bias/variance tradeoff is an everyday struggle
- Bayesian ML can lead to very robust models
- Nevertheless, high-dimensional feature spaces require clever solutions

Up next:

- Introducing physics-based bias to ML models
- Structural bias through operator architectures
- Regression with ML across the scales

We hope you enjoyed this part!