DD2380 - Machine Learning an introduction

Josephine Sullivan

February 17, 2020

What is Machine Learning?

"*Machine learning is the science of getting computers to act without being explicitly programmed.*"

Andrew Ng via Coursera.

"*Learning is any process by which a system improves performance from experience.*"

Herbert Simon.

What is Machine Learning?

"*Machine learning is the science of getting computers to act without being explicitly programmed.*"

Andrew Ng via Coursera.

"*Learning is any process by which a system improves performance from experience.*"

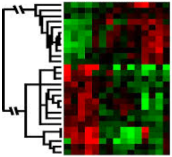
Herbert Simon.

When do we use Machine Learning?

Use **Machine Learning** when

- Human expertise does not exist (navigating on Mars)
- Humans cannot explain their expertise (speech recognition)
- Models are based on huge amounts of data (genomics)
- Models must be customized (personalized medicine)



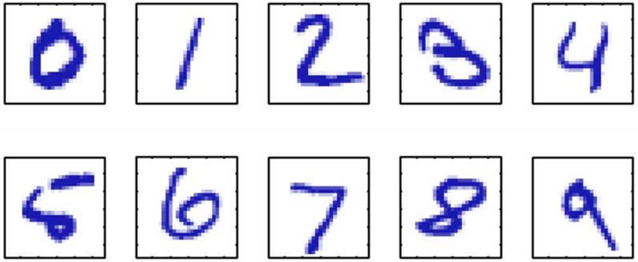
Classic example of a task requiring machine learning

- Can you write down rules to define a chair?
- How could you encode these rules in a computer programme to recognise the image of a chair???

Classic example of a task requiring machine learning

- Can you write down rules to define a chair?
- How could you encode these rules in a computer programme to recognise the image of a chair???

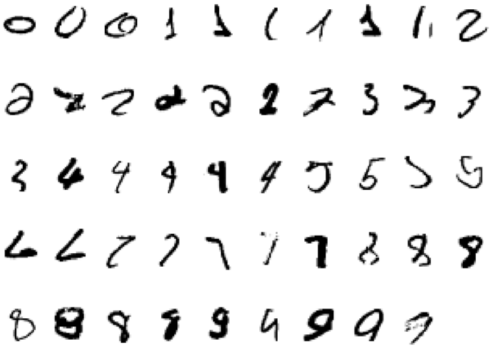
Example: Hand-written digit recognition



- Images are 28×28 arrays of numbers.
- Represent input image as a vector $\mathbf{x} \in \mathbb{R}^{784}$
- Learn a classifier function s.t.

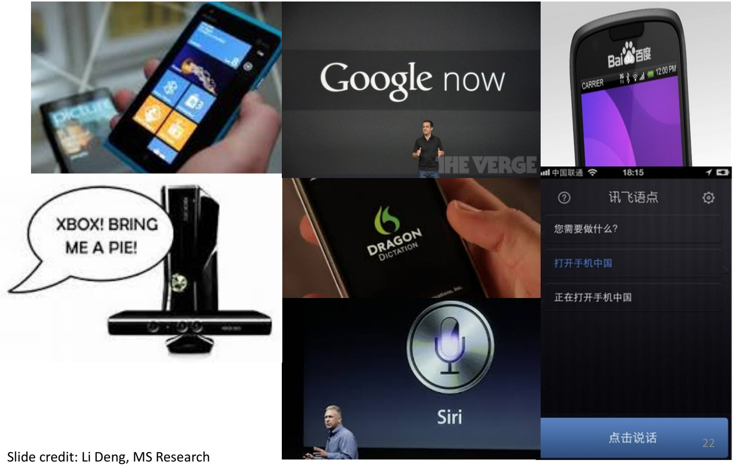
$$f : \mathbb{R}^{784} \to \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

- As a supervised classification problem.
- Start with training data



- Can achieve testing error of $\leq$0.4%
- One of first commercial and widely used ML systems (for zip codes & checks)

Impact of machine learning (really deep learning) on speech technology



Slide credit: Li Deng, MS Research

Web example: Google translate

- Use neural networks to perform machine translation.
- Based on an RNN encoder-decoder neural network model.
  (encodes sentence and then decodes to target language.)

Google Translate

People who bought Hastie:

- Uses the ML technique of collaborative filtering.

Example: Netflix

- Netflix wants "*Everybody to be watching Netflix all the time!*"

- Netflix uses AI/Data/Machine Learning extensively to help with this goal.

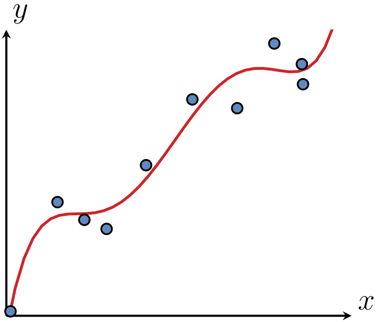
- Examples of this use:
  - Personalization of Movie Recommendations
  - Auto-Generation and Personalization of Thumbnails / Artwork
  - Movie Editing (Post-Production)
  - Streaming Quality

For more details check out: How Netflix Uses AI, Data Science, and Machine Learning From A Product Perspective

Why your Netflix thumbnails don't look like mine

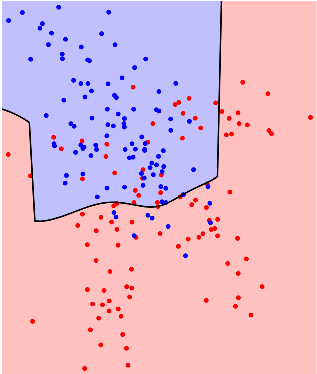
What Netflix's is up to

1. **Supervised Regression**
   - Learn a mapping from $\mathbb{R}^d$ to $\mathbb{R}^k$ where $k \geq 1$.
   - Have labelled examples for training.



2. **Supervised Classification**

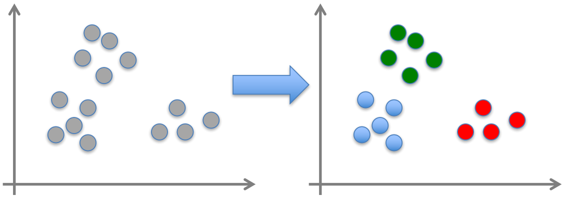
   - Learn a mapping from $\mathbb{R}^d$ to $\{1, 2, 3, \ldots, k\}$ where $k \geq 2$.
   - Have labelled examples for training.



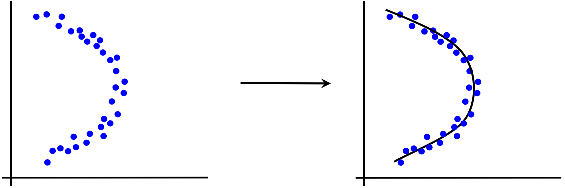
Four canonical learning problems

3. **Unsupervised learning** - model the data

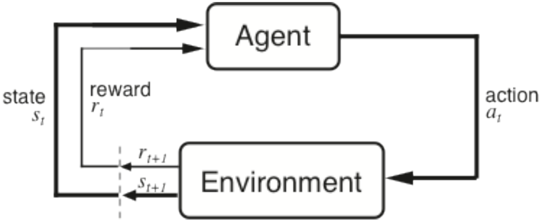
   - Clustering



   - Dimensionality reduction



4. **Reinforcement learning**
   - Rewards from sequence of actions



Today's lecture

- Focus on the problem of supervised regression.

- **Why?** Highlight transferable issues that arise in generic ML.

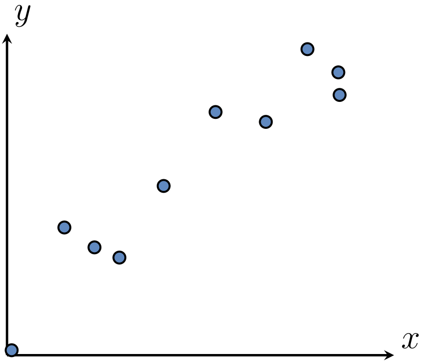
Devil's advocate: It's all just curve fitting!

> *As much as I look into what's being done with deep learning, I see* **they're all stuck there on the level of associations. Curve fitting.** *That sounds like sacrilege, to say that all the impressive achievements of deep learning amount to just fitting a curve to data. From the point of view of the mathematical hierarchy,* **no matter how skillfully you manipulate the data and what you read into the data when you manipulate it, it's still a curve-fitting exercise, albeit complex and non-trivial.**

Judea Pearl.

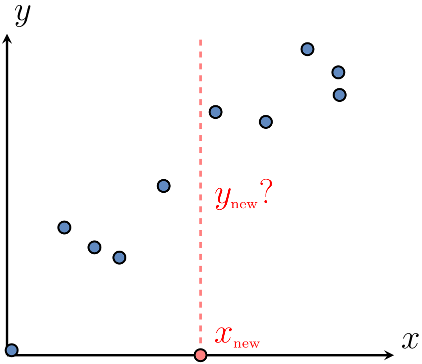
Supervised learning - Regression

Supervised Regression: What you are given

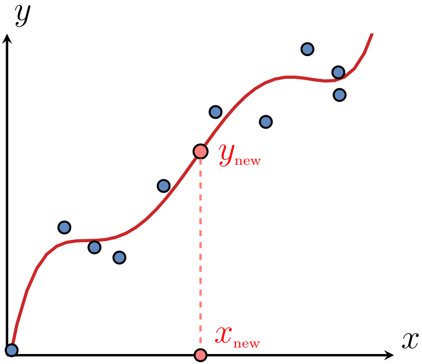


- **Given**: Labelled training data $\mathcal{X} = \{(x_i, y_i)\}_{i=1}^{n}$ where each $x_i \in \mathbb{R}$ and $y_i \in \mathbb{R}$.

Supervised Regression: The task



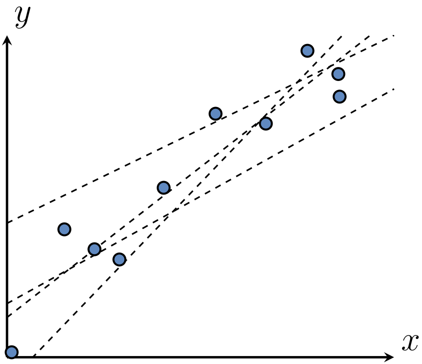
- **Task**: for any $x_{\text{new}} \in \mathbb{R}$ predict its y_{new} value.



- **Solution**: Learn a function, $f : \mathbb{R} \to \mathbb{R}$, that predicts output value y_{new} for input x_{new} that is

$$f(x_{\text{new}}) = y_{\text{new}}$$

Supervised Regression: Learn which function?



Learning requires:

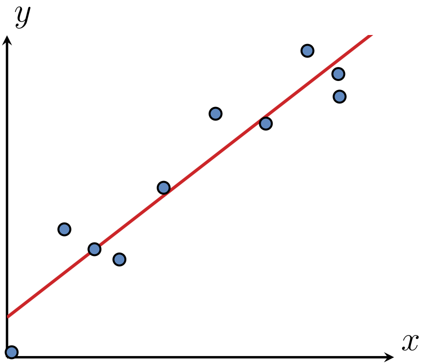
1. Defining type of $f : \mathbb{R} \times \mathbb{R}^p \to \mathbb{R}$ such as a linear function

$$f(x, \boldsymbol{\theta}) = wx + b \qquad \text{where } \boldsymbol{\theta} = \begin{pmatrix} b \\ w \end{pmatrix}$$

with parameters $\boldsymbol{\theta}$ controlling its exact shape.

2. Estimating a good $\boldsymbol{\theta}^*$ s.t. $f(x_i, \boldsymbol{\theta}^*) \approx y_i$ for $i = 1, \ldots, n$

Supervised Regression: Learn which function?



Learning requires:

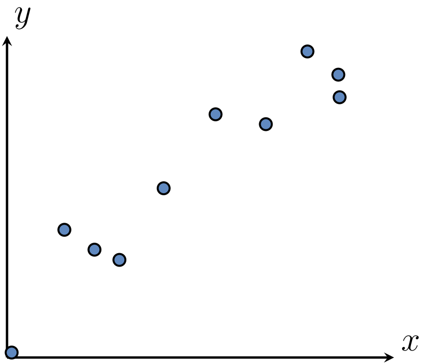
1. Defining type of $f : \mathbb{R} \times \mathbb{R}^p \to \mathbb{R}$ such as a linear function

$$f(x, \boldsymbol{\theta}) = wx + b \qquad \text{where } \boldsymbol{\theta} = \begin{pmatrix} b \\ w \end{pmatrix}$$

   with parameters $\boldsymbol{\theta}$ controlling its exact shape.

2. Estimating a good $\boldsymbol{\theta}^*$ s.t. $f(x_i, \boldsymbol{\theta}^*) \approx y_i$ for $i = 1, \ldots, n$

Focus on estimating $\boldsymbol{\theta}$ from training data

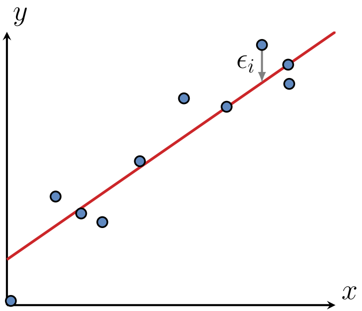


- In this example there appears to be a linear relationship between x and y.
- So let

$$f(x, \boldsymbol{\theta}) = wx + b \qquad \text{where } \boldsymbol{\theta} = \begin{pmatrix} b \\ w \end{pmatrix}$$

- How can we estimate a good $\boldsymbol{\theta}$ from the training data $\mathcal{X} = \{(x_i, y_i)\}_{i=1}^n$?

Least squares estimation of $\boldsymbol{\theta}$



- How to estimate $\boldsymbol{\theta}$ from the training data?
- Want

$$f(x_i, \boldsymbol{\theta}) = w x_i + b \approx y_i \qquad \text{for } i = 1, \ldots, n$$

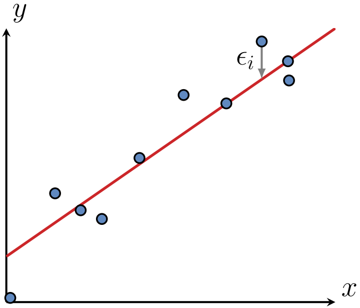
- So set up this least squares optimization problem

$$w^*, b^* = \arg\min_{w,b} \sum_{i=1}^{n} \epsilon_i^2$$

where

$$\epsilon_i = w x_i + b - y_i$$

Can solve this using calculus!



- Let

$$C(w, b, \mathcal{X}) = \sum_{i=1}^{n} \epsilon_i^2$$

- Take partial derivatives of C w.r.t. w and b and set them to zero:

$$\frac{\partial C}{\partial w} = 0 \quad \text{and} \quad \frac{\partial C}{\partial b} = 0$$

- Solve this system of equations - two unknowns and two linear constraints.

The resulting maths

- Expression for the partial derivatives

$$\frac{\partial C}{\partial w} = 2\sum_{i=1}^{n}(wx_i + b - y_i)x_i, \quad \frac{\partial C}{\partial b} = 2\sum_{i=1}^{n}(wx_i + b - y_i)$$

- Set these two equations to zero and solve for w and b:

$$w^* = \frac{\sum_i x_i y_i - n\bar{y}\bar{x}}{\sum_i x_i^2 + n\bar{x}^2},$$
$$b^* = \bar{y} - w^*\bar{x}$$

where

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \text{and} \quad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

Can write what we have done in matrix notation

- Let

$$X = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \quad \text{and} \quad \boldsymbol{\theta} = \begin{pmatrix} b \\ w \end{pmatrix}$$

- Then

$$\mathbf{y} = X\boldsymbol{\theta}$$

  and

$$C(\boldsymbol{\theta}, \mathcal{X}) = (X\boldsymbol{\theta} - \mathbf{y})^T (X\boldsymbol{\theta} - \mathbf{y})$$

- If we know some vector calculus.....

Can write what we have done in matrix notation

- Let

$$X = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \quad \text{and} \quad \boldsymbol{\theta} = \begin{pmatrix} b \\ w \end{pmatrix}$$

- Then

$$\mathbf{y} = X\boldsymbol{\theta}$$

  and

$$C(\boldsymbol{\theta}, \mathcal{X}) = (X\boldsymbol{\theta} - \mathbf{y})^T (X\boldsymbol{\theta} - \mathbf{y})$$

- If we know some vector calculus.....

Can write what we have done in matrix notation

- Compute the gradient of C w.r.t. $\boldsymbol{\theta}$ and set to $\mathbf{0}$

$$\frac{\partial C}{\partial \boldsymbol{\theta}} = 2X^T X\boldsymbol{\theta} - 2X^T\mathbf{y} = \mathbf{0}$$

"*The Matrix Cookbook*" is your friend.

- The optimal $\boldsymbol{\theta}$ is then given by (if $X^T X$ is invertible)

$$\boldsymbol{\theta}^* = (X^T X)^{-1} X^T \mathbf{y}$$

- Cleaner specification of the solution worth the initial extra effort, especially if we consider....

Can write what we have done in matrix notation

- Compute the gradient of C w.r.t. $\boldsymbol{\theta}$ and set to $\mathbf{0}$

$$\frac{\partial C}{\partial \boldsymbol{\theta}} = 2X^T X\boldsymbol{\theta} - 2X^T\mathbf{y} = \mathbf{0}$$

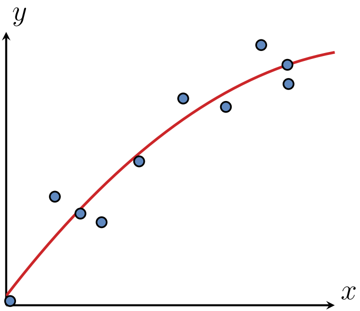
"*The Matrix Cookbook*" is your friend.

- The optimal $\boldsymbol{\theta}$ is then given by (if $X^T X$ is invertible)

$$\boldsymbol{\theta}^* = (X^T X)^{-1} X^T\mathbf{y}$$

- Cleaner specification of the solution worth the initial extra effort, especially if we consider....

Are we really sure f should be linear?



- Perhaps a better model of the training data would be a quadratic function:

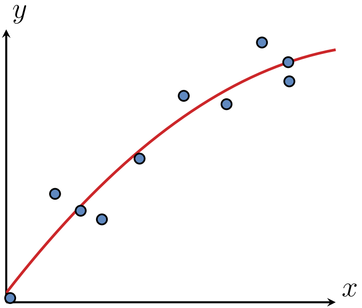
$$f(x, \boldsymbol{\theta}) = w_1 + w_2 x + w_3 x^2$$

where $\boldsymbol{\theta} = (w_1, w_2, w_3)^T$

- We can find an estimate for $\boldsymbol{\theta}$ by solving the same least squares problem as before.

$$\arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \left(w_1 + w_2 x_i + w_3 x_i^2 - y_i\right)^2$$

Least squares for fitting a non-linear function



- We can find an estimate for $\boldsymbol{\theta}$ by solving the same least squares problem as before.

$$\arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \left(w_1 + w_2 x_i + w_3 x_i^2 - y_i\right)^2$$

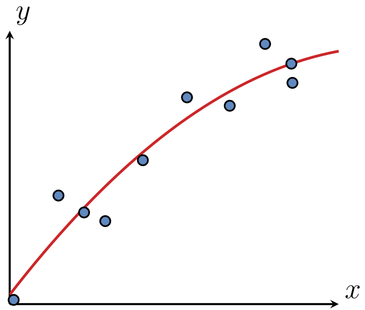
- Can write this in matrix notation

$$\arg\min_{\boldsymbol{\theta}} \left(X\boldsymbol{\theta} - \mathbf{y}\right)^T \left(X\boldsymbol{\theta} - \mathbf{y}\right)$$

where

$$X = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \\ 1 & x_n & x_n^2 \end{pmatrix} \quad \text{and} \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

Least squares for fitting a non-linear function



- Can write the least squares criterion in matrix notation

$$\arg\min_{\boldsymbol{\theta}} \, (X\boldsymbol{\theta} - \mathbf{y})^T (X\boldsymbol{\theta} - \mathbf{y})$$

where

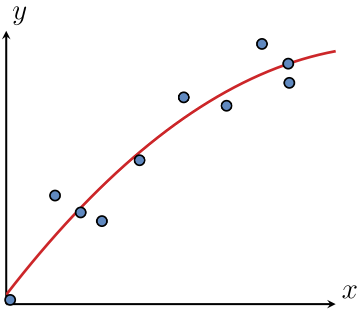
$$X = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \\ 1 & x_n & x_n^2 \end{pmatrix} \quad \text{and} \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

- Have just solved this optimization problem and

$$\boldsymbol{\theta}^* = (X^T X)^{-1} X^T \mathbf{y}$$

- Note have fit a non-linear function by applying a non-linear transformation to input and then solving a linear optimization problem.

Least squares for fitting a non-linear function



- Can write the least squares criterion in matrix notation

$$\arg\min_{\boldsymbol{\theta}} \, (X\boldsymbol{\theta} - \mathbf{y})^T (X\boldsymbol{\theta} - \mathbf{y})$$

where

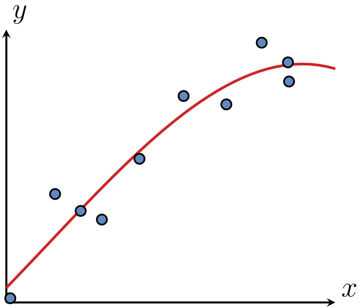
$$X = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \\ 1 & x_n & x_n^2 \end{pmatrix} \quad \text{and} \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

- Have just solved this optimization problem and

$$\boldsymbol{\theta}^* = (X^T X)^{-1} X^T \mathbf{y}$$

- Note have fit a non-linear function by applying a non-linear transformation to input and then solving a linear optimization problem.

Least squares for fitting a non-linear function



- Same trick can be used to fit a cubic, just let

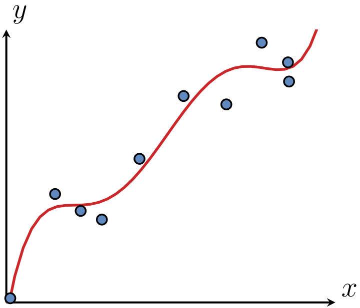
$$X = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{pmatrix}$$

- Can potentially fit a polynomial of degree p this way! Just let

$$X = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^p \\ 1 & x_2 & x_2^2 & \cdots & x_2^p \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^p \end{pmatrix}$$

- If I have n training points can I really fit a polynomial of any degree p with this method?

Least squares for fitting a non-linear function



- Same trick can be used to fit a cubic, just let

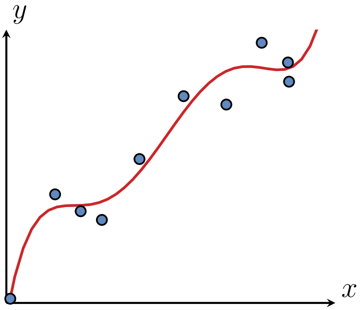
$$X = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{pmatrix}$$

- Can potentially fit a polynomial of degree p this way! Just let

$$X = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^p \\ 1 & x_2 & x_2^2 & \cdots & x_2^p \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^p \end{pmatrix}$$

- If I have n training points can I really fit a polynomial of any degree p with this method?

Least squares for fitting a non-linear function



- Same trick can be used to fit a cubic, just let

$$X = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{pmatrix}$$

- Can potentially fit a polynomial of degree p this way! Just let

$$X = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^p \\ 1 & x_2 & x_2^2 & \cdots & x_2^p \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^p \end{pmatrix}$$

- If I have n training points can I really fit a polynomial of any degree p with this method?

Need sufficient training points to fit polynomial of degree p

- Remember coefficients of polynomial are estimated by

$$\boldsymbol{\theta}^* = (X^T X)^{-1} X^T \mathbf{y}$$

- The size of $X^T X$ is $(p+1) \times (p+1)$ as X is $n \times (p+1)$

- What can we say about the rank of $X^T X$?

$$\mathsf{rank}(X^T X) \leq \min(n, p+1)$$

$\implies$ $X^T X$ is definitely singular when $n < p+1$.

$\implies$ $(X^T X)^{-1}$ does not exist when $n < p+1$.

$\implies$ cannot find $\boldsymbol{\theta}^*$ with the above expression.

Need $\geq p+1$ points to uniquely fit polynomial of degree p

- Remember: Size of X is $n \times (p+1)$
- When rank$(X) \leq n < p+1$
  - There exist multiple $\boldsymbol{\theta}^*$ s.t.

$$X\boldsymbol{\theta}^* - \mathbf{y} = \mathbf{0}$$

  - Each θ^* has the form

$$\boldsymbol{\theta}^* = X^\dagger \mathbf{y} + V\boldsymbol{\gamma}, \quad \boldsymbol{\gamma} \in \mathbb{R}^{p+1-\text{rank(X)}}$$

    where

    * $X^\dagger$ is the pseudo-inverse of X.
      (rank$(X) = n$ then $X^\dagger = X^T(XX^T)^{-1}$)
    * each column of V is a basis vector for the nullspace of X.

Need $\geq p+1$ points to uniquely fit polynomial of degree p

- Remember: Size of X is $n \times (p+1)$

- When $\text{rank}(X) \leq n < p+1$
  - There exist multiple $\boldsymbol{\theta}^*$ s.t.

$$X\boldsymbol{\theta}^* - \mathbf{y} = \mathbf{0}$$

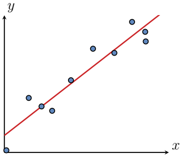
  - Each θ^* has the form

$$\boldsymbol{\theta}^* = X^\dagger \mathbf{y} + V\boldsymbol{\gamma}, \quad \boldsymbol{\gamma} \in \mathbb{R}^{p+1-\text{rank(X)}}$$

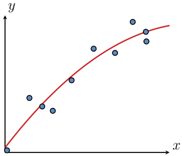
    where

    * $X^\dagger$ is the pseudo-inverse of X.
      ($\text{rank}(X) = n$ then $X^\dagger = X^T(XX^T)^{-1}$)

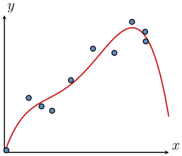
    * each column of V is a basis vector for the nullspace of X.



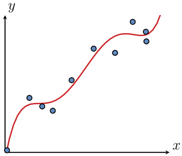
poly degree 1?



poly degree 2?



poly degree 4?



poly degree 5?

- **Given**:
  - n labelled training examples - $\{(x_1, y_1), \ldots, (x_n, y_n)\}$.
  - Lots of choices for form of the fitting function - all polynomials up to degree p.
  - A method to estimate the parameters $\boldsymbol{\theta}$ given a particular f - least squares estimation.

- **Problem**:
  - How do I decide which function I should choose as my final predictor?

Option 1: Which polynomial degree?

- Choose the model that minimizes the training error.
- Assess performance for each possible function:

Calculate training error for each f

for $j = 1, \ldots, p$

- Fit poly of degree j to training data to get $\boldsymbol{\theta}_j^*$
- Calculate the **training error**

$$\text{err}_j = \frac{1}{n} \sum_{i=1}^{n} \left(f_j(x_i, \boldsymbol{\theta}_j^*) - y_i \right)^2$$

- Choose the optimal function with

$$j^* = \arg \min_{1 \le j \le p} \text{err}_j$$

Option 1: Which polynomial degree?

- Choose the model that minimizes the training error.
- Assess performance for each possible function:

Calculate training error for each f

for $j = 1, \ldots, p$

- Fit poly of degree j to training data to get $\boldsymbol{\theta}_j^*$
- Calculate the **training error**

$$\text{err}_j = \frac{1}{n} \sum_{i=1}^{n} \left(f_j(x_i, \boldsymbol{\theta}_j^*) - y_i \right)^2$$

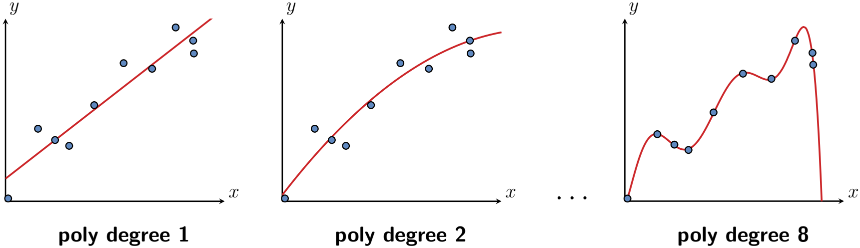
This is a terrible option.

- Choose the optimal function with

$$j^* = \arg \min_{1 \leq j \leq p} \text{err}_j$$

Don't measure performance with training error

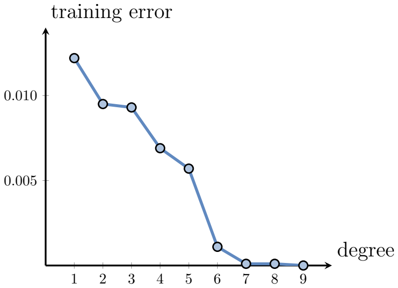
Least squares fit of polynomials of different degrees to the training data.



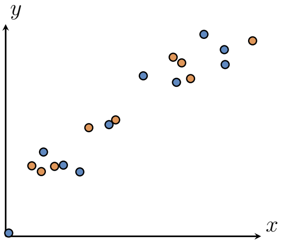
- By increasing the complexity of the function can drive the training error to zero.

Don't measure performance with training error

Training error Vs **polynomial degree** for our toy problem.



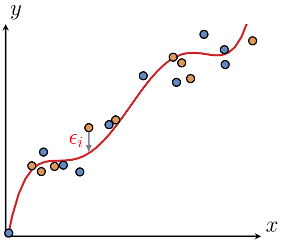
- By increasing the complexity of the function can drive the training error to zero.



- Generate more labelled examples $\{(x_{n+i}, y_{n+i})\}_{i=1}^{m}$ (not used during training) called the **test set**.
- Define the **test error** for our problem as

$$\text{test err}_j = \frac{1}{m}\sum_{i=1}^{m}\left(f_j(x_{n+i}, \boldsymbol{\theta}_j^*) - y_{n+i}\right)^2$$

- **Test error** indicates how well the function **generalizes** to unseen samples.



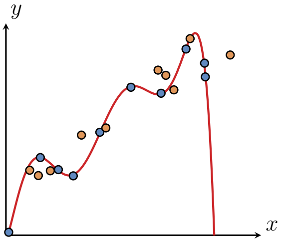
- Generate more labelled examples $\{(x_{n+i}, y_{n+i})\}_{i=1}^{m}$ (not used during training) called the **test set**.
- Define the **test error** for our problem as

$$\text{test err}_j = \frac{1}{m}\sum_{i=1}^{m}\left(f_j(x_{n+i}, \boldsymbol{\theta}_j^*) - y_{n+i}\right)^2$$

- **Test error** indicates how well the function **generalizes** to unseen samples.

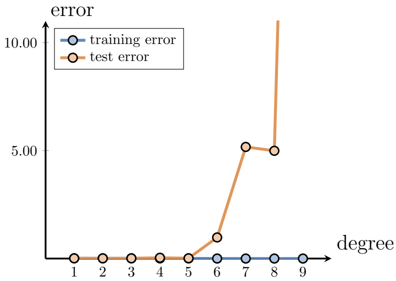
Low training error and high test error

Degree 8 poly has **training error** ≈ 0 and **test error** ≈ 5



- The **training** and **test error** can be very different.
- **Over-fitting** occurs when the training error is low but the test error is high.

Training and test error for our toy problem.



- Over-fitting occurs when the training error is low but the test error is high.
- The higher the capacity of your function $\implies$ more likely to over-fit.

Option 2: Which polynomial degree?

- **Criterion**: Choose the model that minimizes the test error.
- **Logistics**: Calculate **test error** for each possible function:

Calculate test error for each f

for $j = 1, \ldots, p$

- Fit polynomial of degree j to the training data to get $\boldsymbol{\theta}_j^*$
- Calculate the **test error**

$$\text{test err}_j = \frac{1}{m} \sum_{i=1}^{m} \left(f_j(x_{n+i}, \boldsymbol{\theta}_j^*) - y_{n+i} \right)^2$$

- Choose the optimal function with

$$j^* = \arg \min_{1 \le j \le p} \text{test err}_j$$

Option 2: Which polynomial degree?

- **Criterion**: Choose the model that minimizes the test error.
- **Logistics**: Calculate **test error** for each possible function:

Calculate test error for each f

for $j = 1, \ldots, p$

- Fit polynomial of degree j to the training data to get $\boldsymbol{\theta}_j^*$
- Calculate the **test error**

$$\text{test err}_j = \frac{1}{m} \sum_{i=1}^{m} \left(f_j(x_{n+i}, \boldsymbol{\theta}_j^*) - y_{n+i} \right)^2$$

- Choose the optimal function with

$$j^* = \arg \min_{1 \le j \le p} \text{test err}_j$$

What is the generalization ability of my final f_{j^*}?

Pipeline for model selection & final performance measure

- Let $\mathcal{X} = \{(x_i, y_i)\}_{i=1}^n$ represent all my labelled data.
- Partition $\mathcal{X}$ into 3 sets
  - the **training set** - $\mathcal{X}_{\text{train}}$,
  - the **validation set** - $\mathcal{X}_{\text{val}}$,
  - the **test set** - $\mathcal{X}_{\text{test}}$.
- Proceed as follows

1. **Model Selection** (measure performance on validation set)

   for $j = 1, \ldots, p$
   - Fit polynomial of degree j using $\mathcal{X}_{\text{train}}$ to get $\boldsymbol{\theta}_j^*$
   - Calculate the **validation error**

   $$\text{val err}_j = \frac{1}{|\mathcal{X}_{\text{val}}|} \sum_{(x,y)\in\mathcal{X}_{\text{val}}} \left(f_j(x, \boldsymbol{\theta}_j^*) - y\right)^2$$

   Select a model: $j^* = \arg\min_{1\leq j\leq p} \text{val err}_j$

Pipeline for model selection & final performance measure

- Let $\mathcal{X} = \{(x_i, y_i)\}_{i=1}^n$ represent all my labelled data.
- Partition $\mathcal{X}$ into 3 sets
  - the **training set** - $\mathcal{X}_{\text{train}}$,
  - the **validation set** - $\mathcal{X}_{\text{val}}$,
  - the **test set** - $\mathcal{X}_{\text{test}}$.
- Proceed as follows

2. **Final training of Selected Model** (train using $\mathcal{X}_{\text{train}} \cup \mathcal{X}_{\text{val}}$)
   - Fit polynomial of degree j^* given $\mathcal{X}_{\text{train}} \cup \mathcal{X}_{\text{val}}$ to get $\boldsymbol{\theta}_{j^*}^*$

Pipeline for model selection & final performance measure

- Let $\mathcal{X} = \{(x_i, y_i)\}_{i=1}^{n}$ represent all my labelled data.
- Partition $\mathcal{X}$ into 3 sets
  - the **training set** - $\mathcal{X}_{\text{train}}$,
  - the **validation set** - $\mathcal{X}_{\text{val}}$,
  - the **test set** - $\mathcal{X}_{\text{test}}$.
- Proceed as follows

3. **Measure Performance** (measure performance on test set)
   - Assess performance of final regressor

$$\text{test err} = \sum_{(x,y)\in\mathcal{X}_{\text{test}}} \left(f_{j^*}(x, \boldsymbol{\theta}^*_{j^*}) - y\right)^2$$

One central challenge of Machine Learning

- Ideally I want to
  1. Have an expressive prediction function f.
  2. Not over-fit during training.
- Simple model $\implies$ less likely to over-fit but can only accurately model simple relationships.
- High capacity model $\implies$ more likely to over-fit but can potentially accurately model complicated relationships.

Challenge

- Ideally I want to
  1. Have an expressive prediction function f.
  2. Not over-fit during training.
- Simple model $\implies$ less likely to over-fit but can only accurately model simple relationships.
- High capacity model $\implies$ more likely to over-fit but can potentially accurately model complicated relationships.

- Ideally I want to
  1. Have an expressive prediction function f.
  2. Not over-fit during training.

- Simple model $\implies$ less likely to over-fit but can only accurately model simple relationships.

- High capacity model $\implies$ more likely to over-fit but can potentially accurately model complicated relationships.

- Ideally I want to
  1. Have an expressive prediction function f.
  2. Not over-fit during training.
- Simple model $\implies$ less likely to over-fit but can only accurately model simple relationships.
- High capacity model $\implies$ more likely to over-fit but can potentially accurately model complicated relationships.

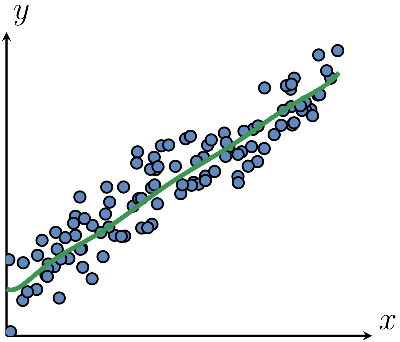
Can I have the best of both worlds??

Sometimes! **Regularization** -

Introduce extra constraints on your optimal model parameters.

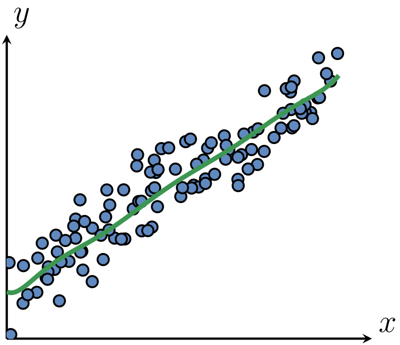
Regularization: Lots of labelled training data option

Least squares polynomial of degree 9 with 1000 training points.



Add many more labelled training points.

Least squares polynomial of degree 9 with 1000 training points.



Add many more labelled training points.

Unfortunately, this is often not an option....

Regularization: Add penalty term to the cost function

- Remember cost-function we've seen so far involves measuring how well $\boldsymbol{\theta}$ *fits* the training data via a sum-of-squares measure:

$$L(\boldsymbol{\theta}, \mathcal{X}) = (X\boldsymbol{\theta} - \mathbf{y})^T (X\boldsymbol{\theta} - \mathbf{y})$$

- Add an extra function, $R : \mathbb{R}^p \to \mathbb{R}$, **regularization** term, to the goodness-of-fit function (often termed the **loss function**)

$$C(\boldsymbol{\theta}, \mathcal{X}) = L(\boldsymbol{\theta}, \mathcal{X}) + \lambda\, R(\boldsymbol{\theta})$$

- Common choice for $R(\boldsymbol{\theta})$ is

$$R(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|^2 = \boldsymbol{\theta}^T \boldsymbol{\theta}$$

Regularization: Add penalty term to the cost function

- Add an extra function, $R : \mathbb{R}^p \to \mathbb{R}$, **regularization** term, to the goodness-of-fit function (often termed the **loss function**)

$$C(\boldsymbol{\theta}, \mathcal{X}) = L(\boldsymbol{\theta}, \mathcal{X}) + \lambda\, R(\boldsymbol{\theta})$$

- Regularization function should have *lower* scores for *simpler* models

  $\Longrightarrow$ its minimization should promote *simpler* models

  $\Longrightarrow$ discourage over-fitting.

- λ controls the trade-off between fitting the training data and *complexity* of the final model.

Example: **Ridge Regression** -

Regularization of the least squares regressor with $R(\boldsymbol{\theta}) = \boldsymbol{\theta}^T \boldsymbol{\theta}$.

Ridge regression for our toy problem

- Have the labelled training data $\mathcal{X} = \{(x_i, y_i)\}_{i=1}^n$

- Ridge regression solves this optimization problem

$$\arg\min_{\boldsymbol{\theta}, w_1} \left[(X\boldsymbol{\theta} + w_1\mathbf{1} - \mathbf{y})^T (X\boldsymbol{\theta} + w_1\mathbf{1} - \mathbf{y})^T + \lambda\, \boldsymbol{\theta}^T \boldsymbol{\theta} \right]$$

where

$$X = \begin{pmatrix} x_1 & x_1^2 & \cdots & x_1^p \\ x_2 & x_2^2 & \cdots & x_2^p \\ \vdots & \vdots & \cdots & \vdots \\ x_n & x_n^2 & \cdots & x_n^p \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \text{ and } \boldsymbol{\theta} = \begin{pmatrix} w_2 \\ w_3 \\ \vdots \\ w_{p+1} \end{pmatrix}$$

- **Note**: Not putting regularization on the offset term.

- To make the solution of the optimization cleaner let

$$X_1 = X - \mathbf{1}^T \boldsymbol{\mu}$$

where

$$\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_i^j, \quad \text{and} \quad \boldsymbol{\mu} = (\mu_1, \ldots, \mu_p)^T$$

- This is called **centering the training data**.

- Important consequence

$$\mathbf{1}^T X_1 = \mathbf{0}^T$$

- The slightly re-jigged optimization problem is

$$\arg\min_{\boldsymbol{\theta}, w_1} \left[(X_1\boldsymbol{\theta} + w_1\mathbf{1} - \mathbf{y})^T (X_1\boldsymbol{\theta} + w_1\mathbf{1} - \mathbf{y})^T + \lambda\, \boldsymbol{\theta}^T\boldsymbol{\theta} \right]$$

Ridge regression - the maths

$$\arg\min_{\boldsymbol{\theta}, w_1} \left[(X_1 \boldsymbol{\theta} + w_1 \mathbf{1} - \mathbf{y})^T (X_1 \boldsymbol{\theta} + w_1 \mathbf{1} - \mathbf{y})^T + \lambda \, \boldsymbol{\theta}^T \boldsymbol{\theta} \right]$$

- Can solve the optimization problem as before.
  1. Compute gradients of the cost function.
  2. Set expression for gradients to zero.
  3. Solve the resulting equation system.

Ridge regression - the maths

$$\arg\min_{\boldsymbol{\theta}, w_1} \left[\left(X_1 \boldsymbol{\theta} + w_1 \mathbf{1} - \mathbf{y} \right)^T \left(X_1 \boldsymbol{\theta} + w_1 \mathbf{1} - \mathbf{y} \right)^T + \lambda \, \boldsymbol{\theta}^T \boldsymbol{\theta} \right]$$

- Can solve the optimization problem as before.
  1. Compute gradients of the cost function.
  2. Set expression for gradients to zero.
  3. Solve the resulting equation system.

Ridge regression - the maths

$$\arg\min_{\boldsymbol{\theta}, w_1} \left[(X_1\boldsymbol{\theta} + w_1\mathbf{1} - \mathbf{y})^T (X_1\boldsymbol{\theta} + w_1\mathbf{1} - \mathbf{y})^T + \lambda\, \boldsymbol{\theta}^T\boldsymbol{\theta} \right]$$

- Can solve the optimization problem as before.
  1. Compute gradients of the cost function.
  2. Set expression for gradients to zero.
  3. Solve the resulting equation system.

- **Solution** (must apply to a centered point)

$$\boldsymbol{\theta}^* = \left(X_1^T X_1 + \lambda I\right)^{-1} X_1^T \mathbf{y}, \qquad w_1^* = \frac{1}{n}\sum_{i=1}^{n} y_i$$

Ridge regression - the maths

$$\arg\min_{\boldsymbol{\theta}, w_1} \left[(X\boldsymbol{\theta} + w_1 \mathbf{1} - \mathbf{y})^T (X\boldsymbol{\theta} + w_1 \mathbf{1} - \mathbf{y})^T + \lambda\, \boldsymbol{\theta}^T \boldsymbol{\theta} \right]$$

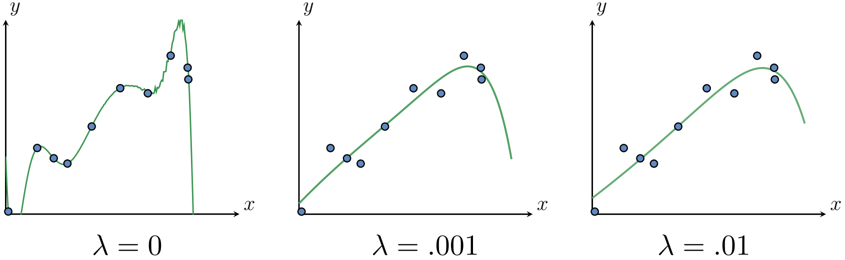
- Can solve the optimization problem as before.
  1. Compute gradients of the cost function.
  2. Set expression for gradients to zero.
  3. Solve the resulting equation system.

- **Solution** (for non-centered data)

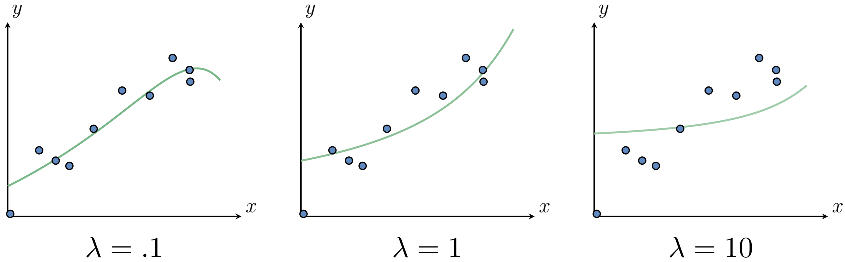
$$\boldsymbol{\theta}^* = \left(X_1^T X_1 + \lambda I \right)^{-1} X_1^T \mathbf{y}, \qquad w_1^* = \frac{1}{n} \sum_{i=1}^{n} y_i - \boldsymbol{\mu}^T \boldsymbol{\theta}^*$$

Effect of λ on ridge regression solution

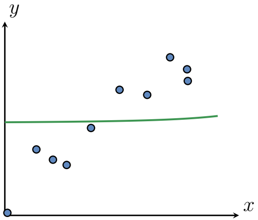
Estimated polynomial of degree 9 with different values of λ.



Estimated polynomial of degree 9 with different values of λ.



Estimated polynomial of degree 9 with different values of λ.



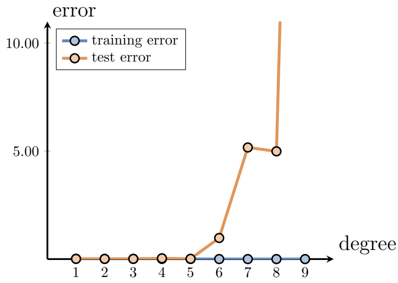
$\lambda = 100$

Effect of λ on ridge regression solution

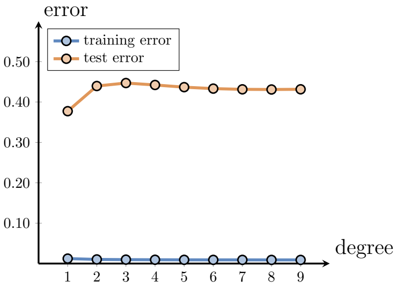
Coefficients of the degree 9 polynomial fitted as λ varies.

λ	w_1	θ_1	θ_2	$\cdots$	θ_6	θ_7	θ_8	θ_9
0	1.215	-43.492	978.553	$\cdots$	206857.077	-217202.970	126881.430	-31664.049
.001	0.877	1.585	-0.679	$\cdots$	-0.043	-0.296	-0.516	-0.701
.010	0.916	1.175	0.254	$\cdots$	-0.166	-0.201	-0.223	-0.235
.100	1.017	0.771	0.359	$\cdots$	-0.098	-0.114	-0.119	-0.118
1.000	1.201	0.300	0.214	$\cdots$	0.047	0.033	0.023	0.016
10.000	1.398	0.074	0.064	$\cdots$	0.027	0.022	0.018	0.015
100.000	1.473	0.009	0.008	$\cdots$	0.004	0.003	0.003	0.002

Effect of ridge-regression on training and test error



$\lambda = 0$



$\lambda = .001$

Training and **test error** Vs **polynomial degree** for our toy problem without and with regularization.

L_1 regularization is also popular and powerful...

- L_1 **regularizer**

$$R_{\text{lasso}}(\boldsymbol{\theta}) = \sum_{i=1}^{p} |\theta_i| = \|\boldsymbol{\theta}\|_1$$

- **Lasso Regression**: squared-error loss + L_1 regularization

$$\boldsymbol{\theta}_{\text{lasso}} = \arg\min \left[\sum_{i=1}^{n} (y_i - f(\mathbf{x}_i, \boldsymbol{\theta}) - w_1)^2 + \lambda \|\boldsymbol{\theta}\|_1 \right]$$

- L_1 **regularizer**

$$R_{\text{lasso}}(\boldsymbol{\theta}) = \sum_{i=1}^{p} |\theta_i| = \|\boldsymbol{\theta}\|_1$$

- **Lasso Regression**: squared-error loss + L_1 regularization

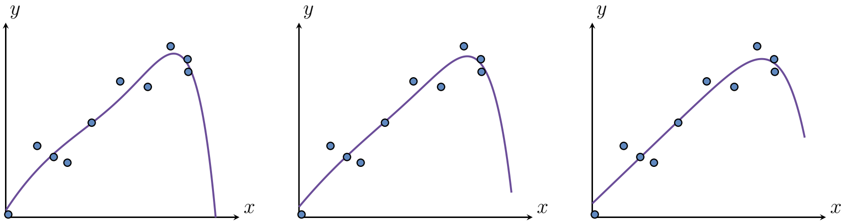
$$\boldsymbol{\theta}_{\text{lasso}} = \arg\min \left[\sum_{i=1}^{n} (y_i - f(\mathbf{x}_i, \boldsymbol{\theta}) - w_1)^2 + \lambda \|\boldsymbol{\theta}\|_1 \right]$$

Is there qualitative difference between $\boldsymbol{\theta}_{\text{lasso}}$ and $\boldsymbol{\theta}_{\text{ridge}}$?

Yes!

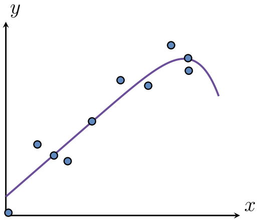
Learnt lasso regressor as λ varies

Estimated polynomial of degree 9 with different values of λ.



$\lambda = 6.5219e - 04$ $\quad$ $\lambda = 0.0042$ $\quad$ $\lambda = 0.0269$

Estimated polynomial of degree 9 with different values of λ.



$\lambda = 0.1732$ $\lambda = 1.1135$

Let's have a look at the coefficients

λ	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8	θ_9
0	2.3140	-3.0120	1.3911	2.0319	-0.0003	-0.0000	0.0002	0.0003	-3.0518
0.001	2.1086	-2.1474	0.2904	2.3597	0	0	0	0	-2.8560
0.005	1.6243	-0.6960	0	0.8283	0.0933	0	0	0	-1.6844
0.010	1.3759	0	0	0	0	0	0	0	-0.9631
0.050	1.3431	0	0	0	0	0	0	0	-0.8661
0.100	1.3027	0	0	0	0	0	0	0	-0.7464
1.000	0.9486	0	0	0	0	0	0	0	0

Let's have a look at the coefficients

λ	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8	θ_9
0	2.3140	-3.0120	1.3911	2.0319	-0.0003	-0.0000	0.0002	0.0003	-3.0518
0.001	2.1086	-2.1474	0.2904	2.3597	0	0	0	0	-2.8560
0.005	1.6243	-0.6960	0	0.8283	0.0933	0	0	0	-1.6844
0.010	1.3759	0	0	0	0	0	0	0	-0.9631
0.050	1.3431	0	0	0	0	0	0	0	-0.8661
0.100	1.3027	0	0	0	0	0	0	0	-0.7464
1.000	0.9486	0	0	0	0	0	0	0	0

- As λ increases, magnitude of non-zero coefficients shrink.
- And as λ increases many shrink to exactly zero.
- L_1 regularization promotes sparsity.

LASSO is great for feature selection.

LASSO $\equiv$ Least absolute shrinkage and selection operator

What generalizes from these slides to other machine learning methods?

What generalizes? Regression from $\mathbb{R}^d$

1. Want to learn a linear function $f : \mathbb{R}^d \times \mathbb{R}^{d+1} \to \mathbb{R}$ that is
$$f(\mathbf{x}, \boldsymbol{\theta}) = \mathbf{w}^T \mathbf{x} + b$$
where $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ and $\boldsymbol{\theta} = (b, \mathbf{w}^T)^T$ and $\mathbf{w} \in \mathbb{R}^d$.

2. Have labelled data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ with each $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$

3. Set
$$X = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \uparrow & \uparrow & \cdots & \uparrow \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n \\ \downarrow & \downarrow & \cdots & \downarrow \end{pmatrix}^T$$

4. Can set up the least squares optimization exactly as before
$$\arg\min_{\boldsymbol{\theta}} \, (X\boldsymbol{\theta} - \mathbf{y})^T (X\boldsymbol{\theta} - \mathbf{y})$$

What generalizes? High-level approach to supervised learning

- **Task**: Learn a function $f : \mathbb{R}^d \times \mathbb{R}^p \to \mathbb{R}^k$

- **High-level solution**:

  1. Decide on how to represent f.
  2. Given labelled training and parametrization of f define an optimization problem which links f's parameter setting to prediction quality on training data.
  3. Solve (or partially solve) the optimization problem to find a good parameter setting for f.
  4. Evaluate the found solution.

Every ML algorithm has three components:

- **Representation**
- **Optimization**
- **Evaluation**

Other take home message

- Machine learning methods can over-fit very easily especially for complicated f's.
- You must be vigilant to avoid this.
- Regularization and data are your friends in this battle.

Over-fitting not the only peril when using Machine Learning

Dataset Bias - Do you see any sheep?

Microsoft Azure's computer vision API added the caption and tags.

A herd of sheep grazing on a lush green hillside
Tags: grazing, sheep, mountain, cattle, horse

Images taken from: Do neural nets dream of electric sheep?

Dataset Bias - I still don't see any sheep

Microsoft Azure's computer vision API added the caption and tags.

A close up of a hillside next to a rocky hill
Tags: hillside, grazing, sheep, giraffe, herd

Images taken from: Do neural nets dream of electric sheep?

Dataset Bias - But I see goats now

Microsoft Azure's computer vision API added the caption and tags.

Left: A man is holding a dog in his hand
Right: A woman is holding a dog in her hand
Image: @SouperSarah

Images taken from: Do neural nets dream of electric sheep?

Perils of Machine Learning

- **Remember**: Computers do exactly what they are asked, so you have to be very specific in what you ask them to do!

- **Machine learning systems** will find the easiest path to solve the task you set them.

- Will exploit loopholes to find shortcuts usually because of:
  - dataset bias between training and test sets
  - poorly specified "loss" functions

Perils of Machine Learning

- **Remember**: Computers do exactly what they are asked, so you have to be very specific in what you ask them to do!

- **Machine learning systems** will find the easiest path to solve the task you set them.

- Will exploit loopholes to find shortcuts usually because of:
  - dataset bias between training and test sets
  - poorly specified "loss" functions

Perils of Machine Learning

- **Remember**: Computers do exactly what they are asked, so you have to be very specific in what you ask them to do!

- **Machine learning systems** will find the easiest path to solve the task you set them.

- Will exploit loopholes to find shortcuts usually because of:
  - dataset bias between training and test sets
  - poorly specified "loss" functions

- **Remember**: Computers do exactly what they are asked, so you have to be very specific in what you ask them to do!

- **Machine learning systems** will find the easiest path to solve the task you set them.

- Will exploit loopholes to find shortcuts usually because of:
  - dataset bias between training and test sets
  - poorly specified "loss" functions

Is your ML algorithm solving the problem you meant it to solve?

Perils of Machine Learning

- **Remember**: Computers do exactly what they are asked, so you have to be very specific in what you ask them to do!

- **Machine learning systems** will find the easiest path to solve the task you set them.

- Will exploit loopholes to find shortcuts usually because of:
  - dataset bias between training and test sets
  - poorly specified "loss" functions

Is your ML algorithm solving the problem you meant it to solve?

Or just exploiting non-meaningful shortcuts.

Postscript: Deep learning & neural networks - generalization not as well understood

The Mystery of Generalization in Deep Learning

- GoogLeNet, VGGNet,.... have many millions of parameters.
- Networks trained with (ignoring data augmentations) # training points $\ll$ # of parameters.
- But these networks still generalize well.
- What's going on??

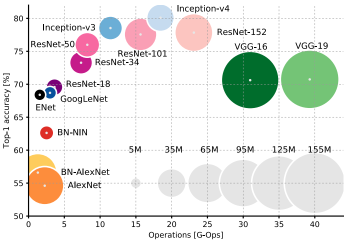
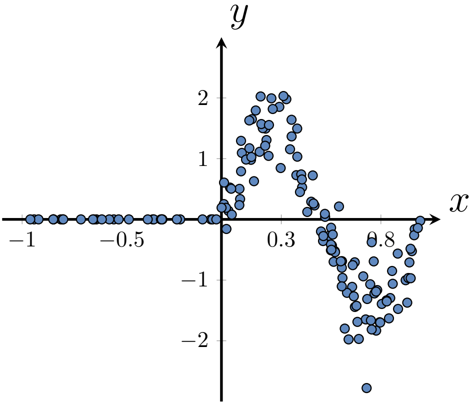


Fig credit: **An Analysis of Deep Neural Network Models for Practical Applications** by Canziani, Culurciello & Paszke.

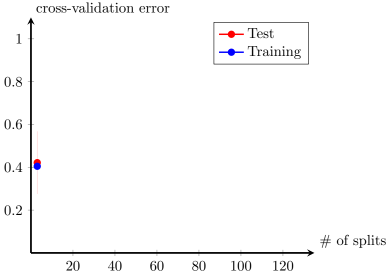
Traditional understanding of Generalization:

controlled by effective # of parameters of classifier/regressor and # of training points.

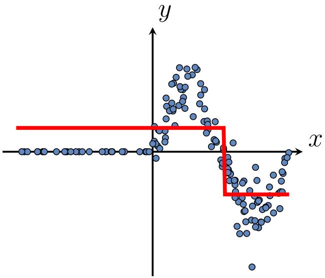


Have 120 data points.

Single Decision Tree with no regularization

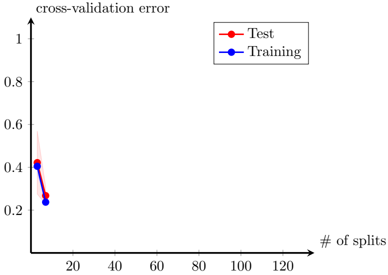


MSE scores

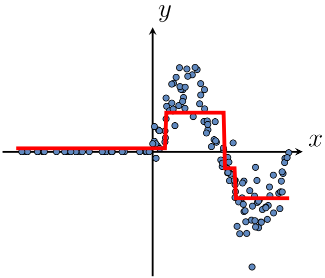


Predicted regression curve

Single Decision Tree with no regularization

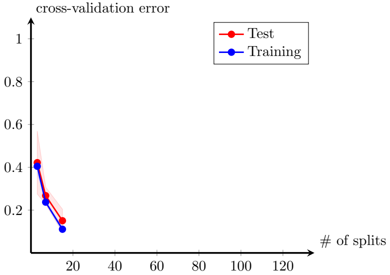


MSE scores

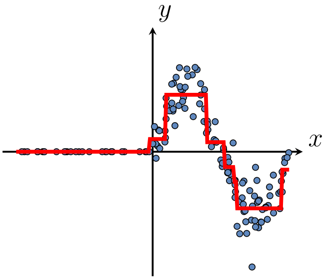


Predicted regression curve

Single Decision Tree with no regularization

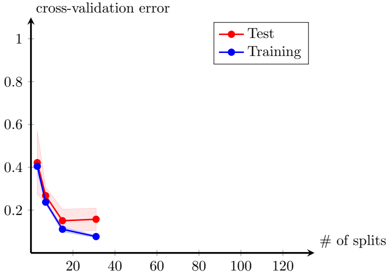


MSE scores

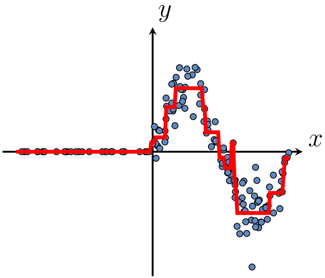


Predicted regression curve

Single Decision Tree with no regularization

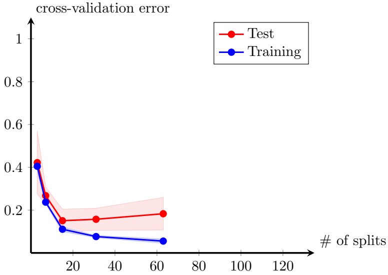


MSE scores

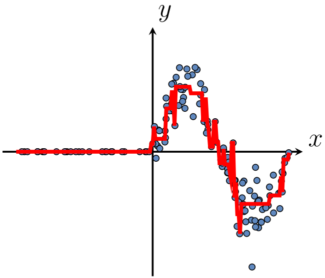


Predicted regression curve

Single Decision Tree with no regularization

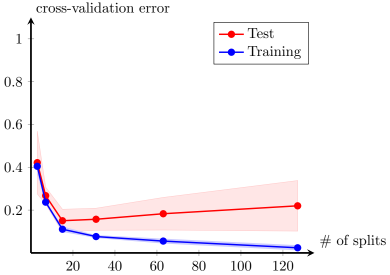


MSE scores

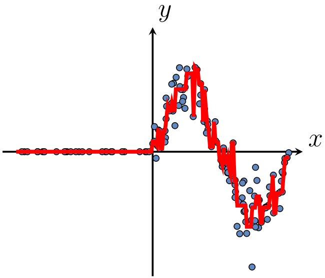


Predicted regression curve

Single Decision Tree with no regularization

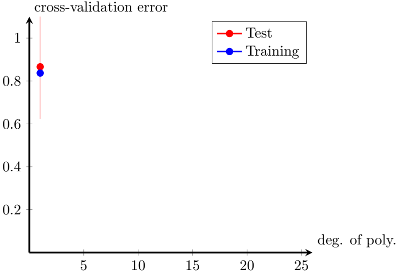


MSE scores

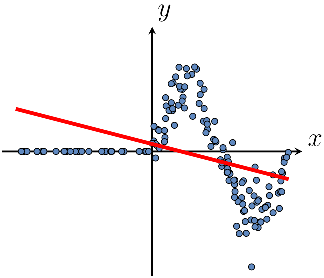


Predicted regression curve

Polynomial Regression with no regularization

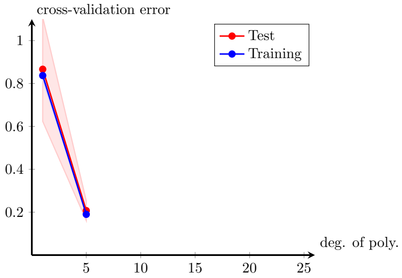


MSE scores

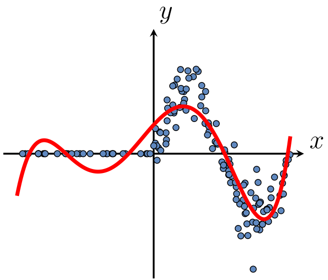


Predicted regression curve

Polynomial Regression with no regularization

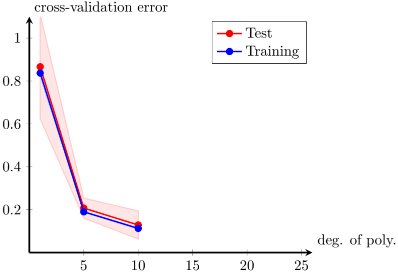


MSE scores

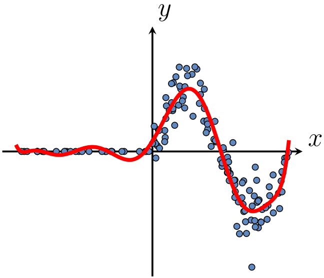


Predicted regression curve

Polynomial Regression with no regularization

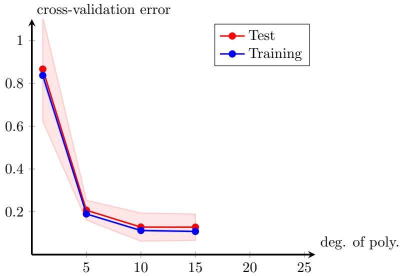


MSE scores

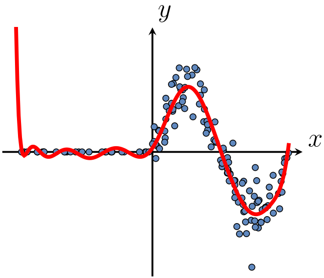


Predicted regression curve

Polynomial Regression with no regularization

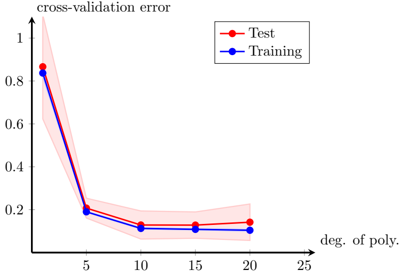


MSE scores

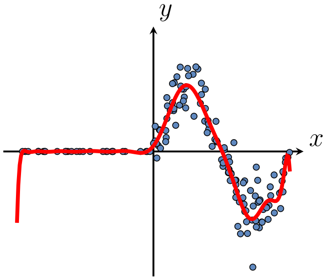


Predicted regression curve

Polynomial Regression with no regularization

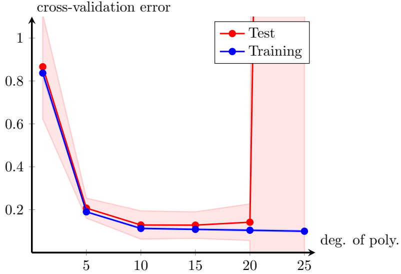


MSE scores

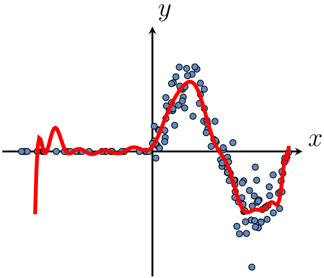


Predicted regression curve

Polynomial Regression with no regularization



MSE scores

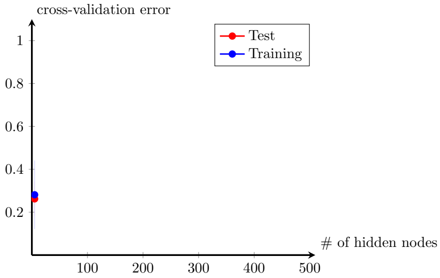


Predicted regression curve

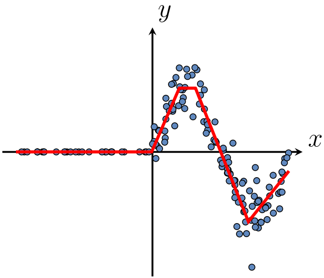
How about a MLP?

- Fit a 1-hidden layer MLP with h hidden nodes
- Training procedure:
  - Mean Square Error loss
  - He initialization
  - ReLu activation function
  - Optimizer: mini-batch GD with cyclic learning rates
  - Batch size: 10

1-hidden layer MLP with no regularization

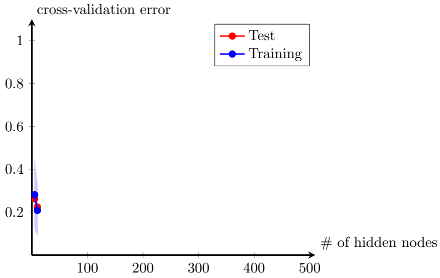


MSE scores

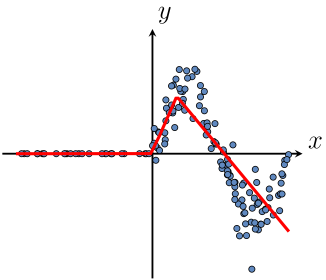


Predicted regression curve

1-hidden layer MLP with no regularization

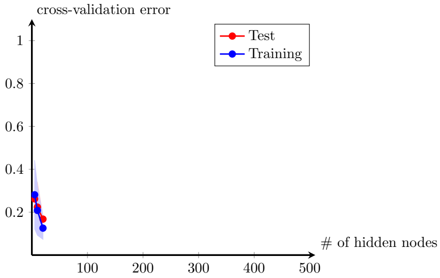


MSE scores

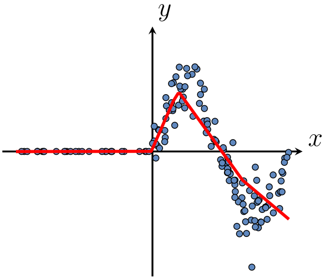


Predicted regression curve

1-hidden layer MLP with no regularization

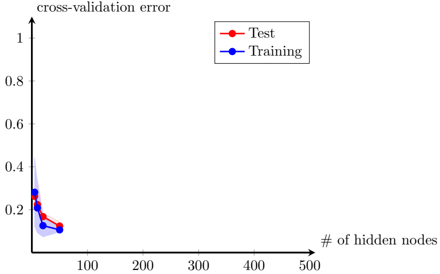


MSE scores

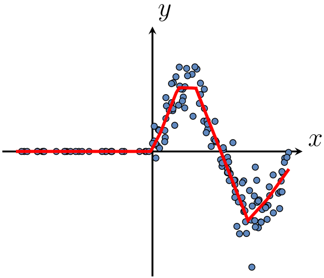


Predicted regression curve

1-hidden layer MLP with no regularization

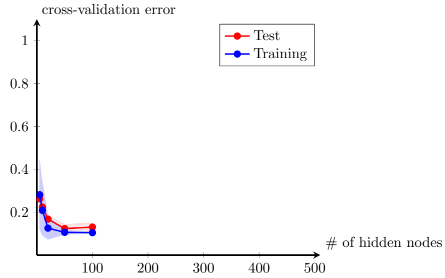


MSE scores

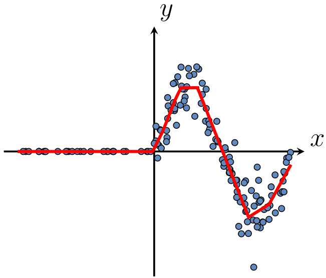


Predicted regression curve

1-hidden layer MLP with no regularization

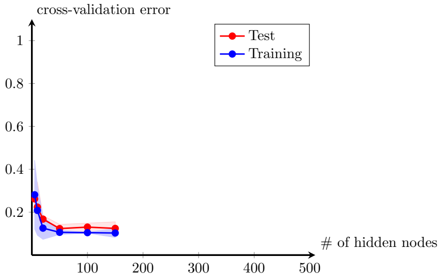


MSE scores

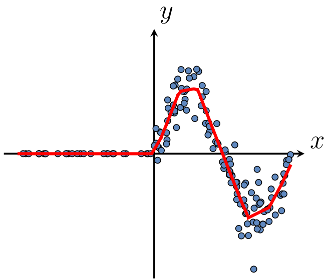


Predicted regression curve

1-hidden layer MLP with no regularization

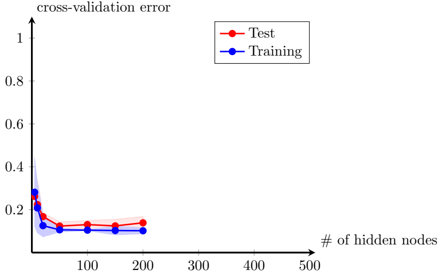


MSE scores

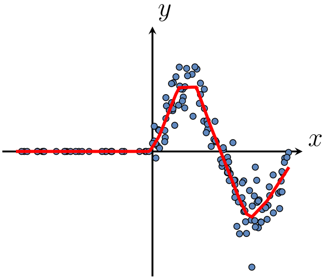


Predicted regression curve

1-hidden layer MLP with no regularization

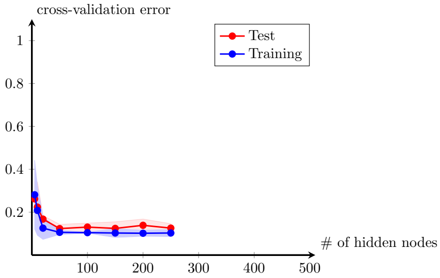


MSE scores

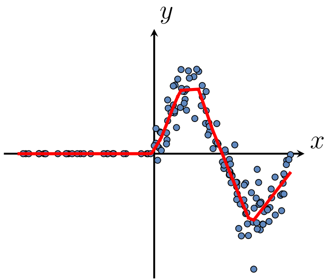


Predicted regression curve

1-hidden layer MLP with no regularization

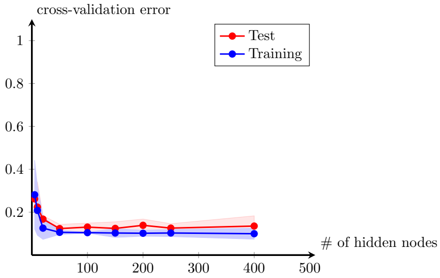


MSE scores

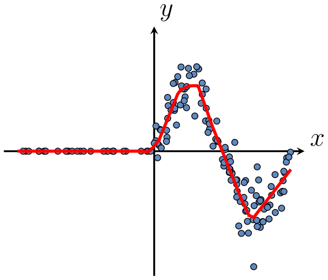


Predicted regression curve

1-hidden layer MLP with no regularization

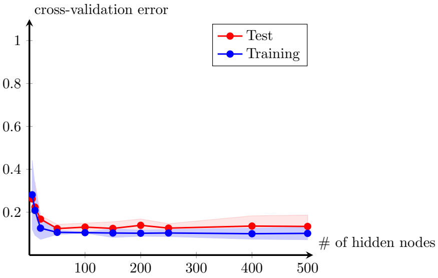


MSE scores

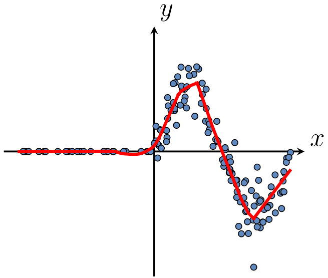


Predicted regression curve

1-hidden layer MLP with no regularization

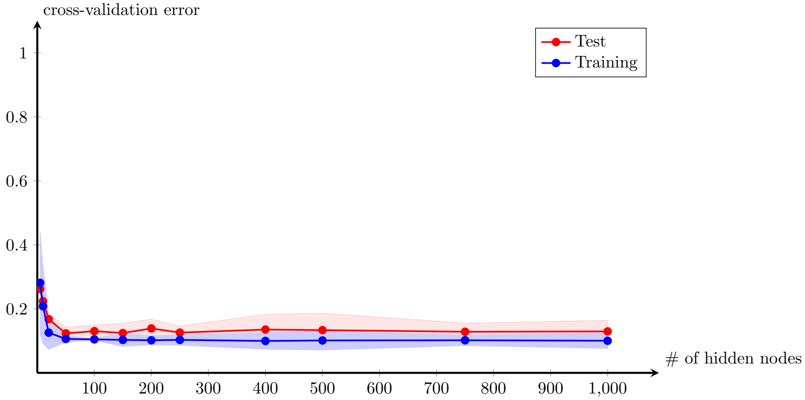


MSE scores



Predicted regression curve

1-hidden layer MLP with no regularization



# of parameters $\gg$ # of data points. No over-fitting???