When Models Meet Data

Rahul Shome

Slides by Jo Ciucă

Australian National University

comp3670@anu.edu.au

Recap: Check your understanding

(A) Orthogonal projections are linear projections.

(B) The result will no longer change when applying orthogonal projection multiple times (>1).

(C) Given a subspace to project on, orthogonal projection gives the minimum information loss (l_2).

(D) Gram-Schmidt Orthogonalization outputs the same number of basis vectors as the input.

(E) Projections allow us to visualize better and understand high-dimensional data.

Recap: Check your understanding

(A) Orthogonal projections are linear projections. YES

(B) The result will no longer change when applying orthogonal projection multiple times (>1). YES

(C) Given a subspace to project on, orthogonal projection gives the minimum information loss (l_2). YES

(D) Gram-Schmidt Orthogonalization outputs the same number of basis vectors as the input. YES

(E) Projections allow us to visualise better and understand high-dimensional data. YES

Further resources for Linear Algebra to hone the intuition

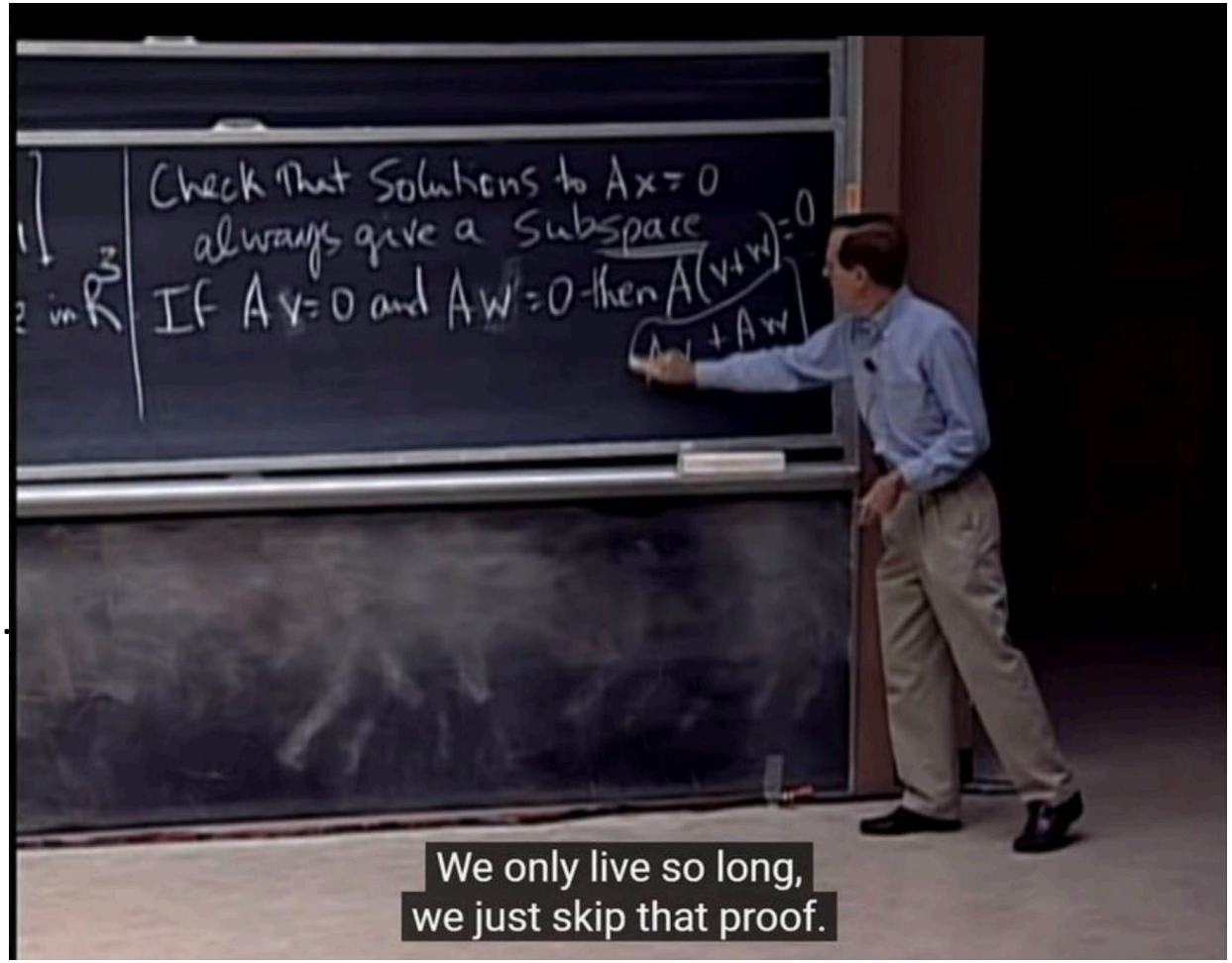
Prof Gilbert Strang MIT course on Linear Algebra

3blue1brown Chapter I

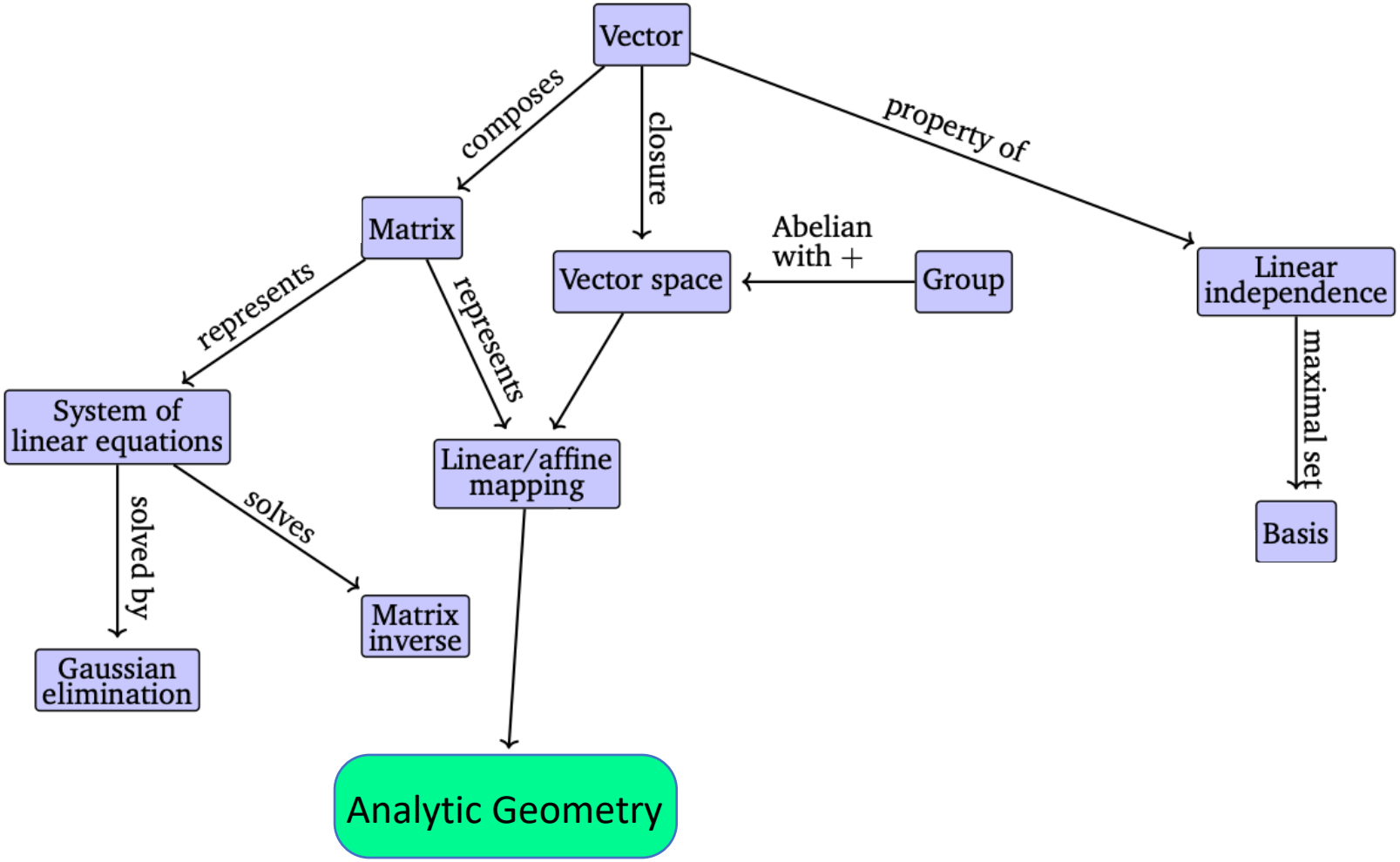
Linear Algebra done right - Axler

Linear Algebra - Serge Lang
Linear Algebra - Shilov

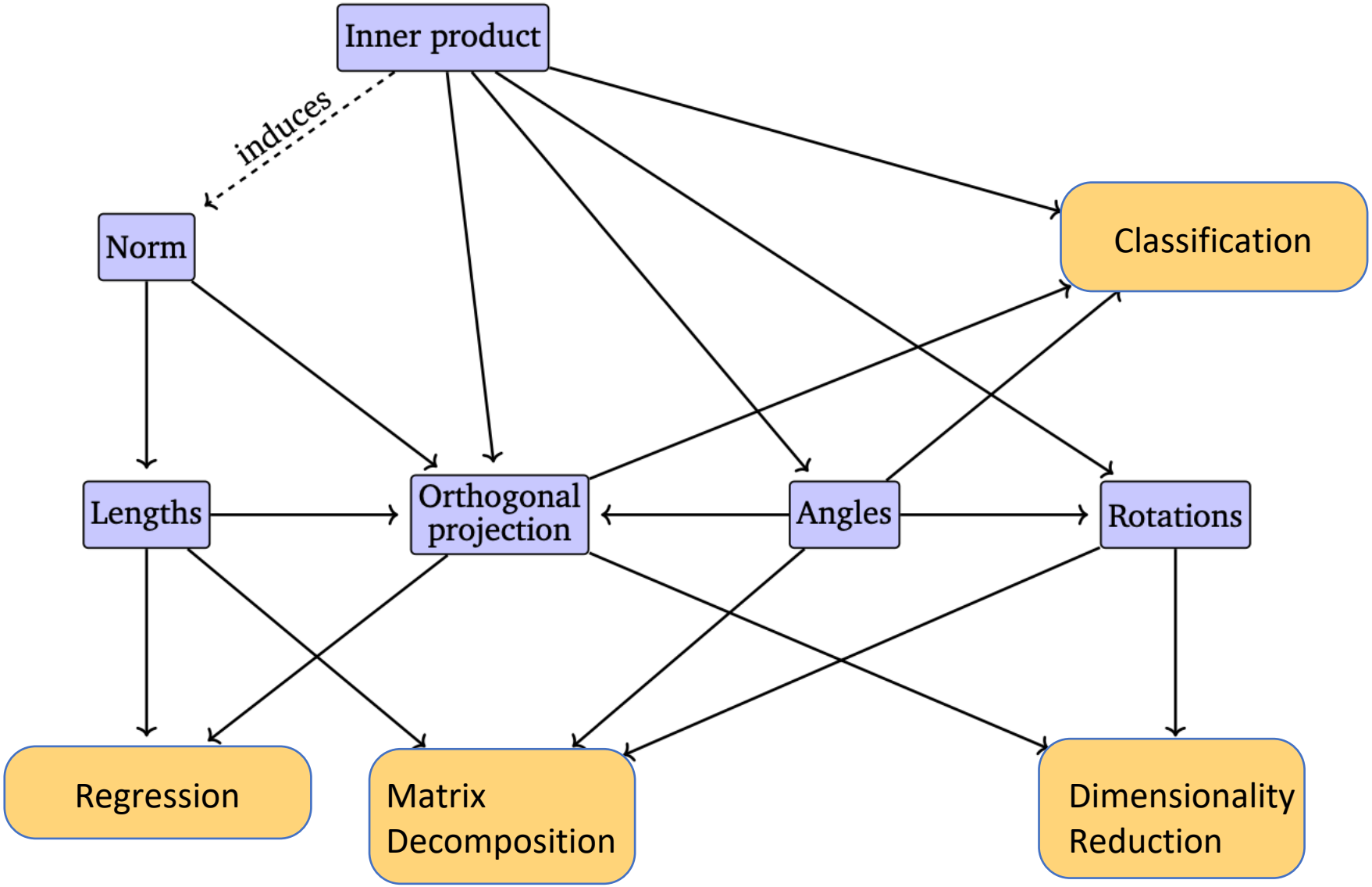
Deep Learning Book - Bengio et al.



The story so far



Where we are going



Linear Algebra

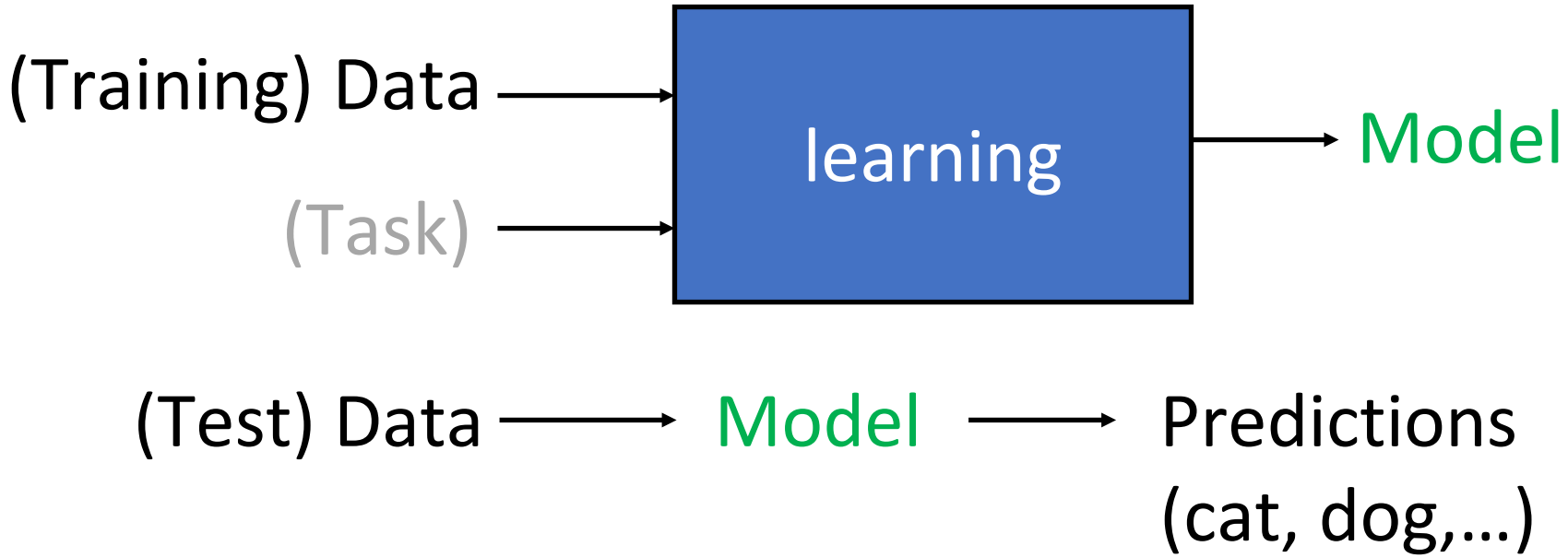
Optimization

Probability

8.1 Data, Models, and Learning

- A machine learning system has three major components:
- Data, models, learning

- A model is obtained by learning from the training data
- A prediction is made by applying a learned model on test data



8.1 Data, Models, and Learning

- We aim to learn good models.
- How is good defined? We need to have performance metrics on the test data. Examples include:
- Classification accuracy
- Distance from the ground truth
- Test time (efficiency)
- Model size
- ……
- New performance metrics are constantly being proposed by the machine learning community.

8.1.1 Data as Vectors

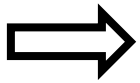
- Data, read by computers, should be in a numerical format.
- See the tabular format below

Name	Degree	Postcode	Age	Annual salary
Aditya	MSc	W21BG	36	89563
Bob	PhD	EC1A1BA	47	123543
Chloé	BEcon	SW1A1BH	26	23989
Daisuke	BSc	SE207AT	68	138769
Elisabeth	MBA	SE10AA	33	113888

- Row: an instance
- Column: a particular feature
- Apart from tabular format, machine learning can be applied to many types of data, e.g., genomic sequences, text and image contents of a webpage, and social media graphs, citation networks...

- We convert the table into numerical format

Name	Degree	Postcode	Age	Annual salary
Aditya	MSc	W21BG	36	89563
Bob	PhD	EC1A1BA	47	123543
Chloé	BEcon	SW1A1BH	26	23989
Daisuke	BSc	SE207AT	68	138769
Elisabeth	MBA	SE10AA	33	113888



Degree	Latitude (in degrees)	Longitude (in degrees)	Age	Annual Salary (in thousands)
2	51.5073	0.1290	36	89.563
3	51.5074	0.1275	47	123.543
1	51.5071	0.1278	26	23.989
1	51.5075	0.1281	68	138.769
2	51.5074	0.1278	33	113.888

- Some columns can be is quantized to -1 and +1
- Degree from BS, MS to PhD: 1, 2, 3
- Postcode corresponds to Latitude and Longitude on the map
- Name is removed because of privacy and because it does not contain useful information for the machine learning system. (considerations?)

[1] Chen et al., What's in a Name? First Names as Facial Attributes. CVPR 2013

Figure 4: The top-3 predicted names for some face images. The correct prediction is highlighted in green, while the actual first name is shown in red if it is not ranked within the top-3 predictions. The first 2 rows give some good examples where our top-3 predictions include the actual name, and the bottom 2 rows are randomly selected from our test set. Even when our predictions are wrong, reasonable names are predicted (e.g., appropriate gender or age).

Open discussion?

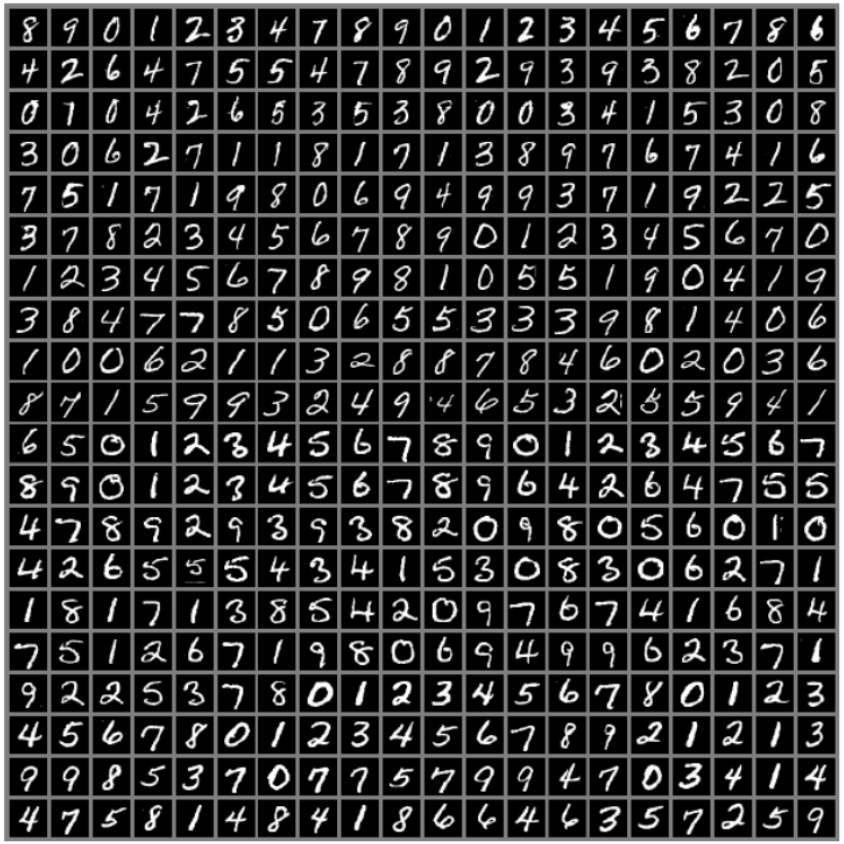
- We use N to denote the number of examples in a dataset and index the examples with lowercase $n = 1, \cdots, N$

Degree	Latitude (in degrees)	Longitude (in degrees)	Age	Annual Salary (in thousands)
2	51.5073	0.1290	36	89.563
3	51.5074	0.1275	47	123.543
1	51.5071	0.1278	26	23.989
1	51.5075	0.1281	68	138.769
2	51.5074	0.1278	33	113.888

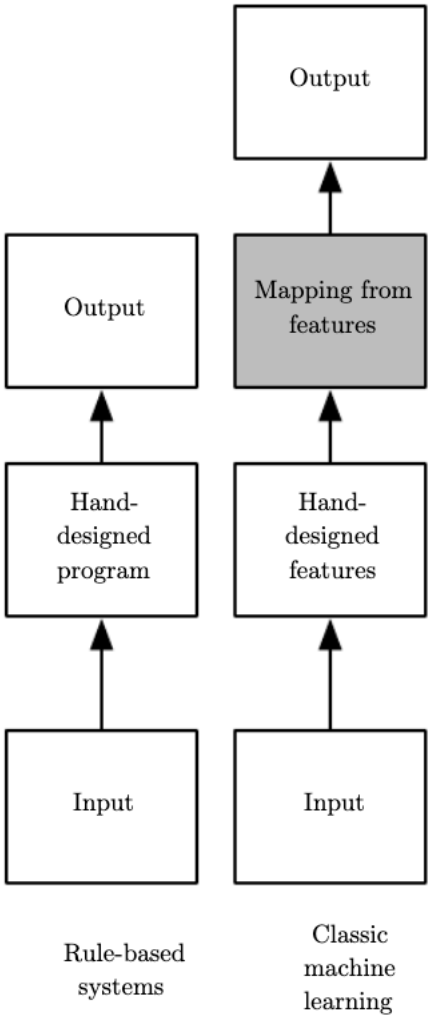
- Each row is a particular individual $\boldsymbol{x}_n$ referred to as an example or data point in machine learning
- The subscript n refers to the fact that this is the nth example out of a total of N examples in the dataset
- Each column represents a particular feature of interest about the example, and we index the features as $d = 1, \cdots, D$
- Each example is a D-dimensional vector

Shaded = learning from data

Classical Machine Learning vs Representation Learning



MNIST dataset, ~70k images



- Consider the problem of predicting annual salary from age

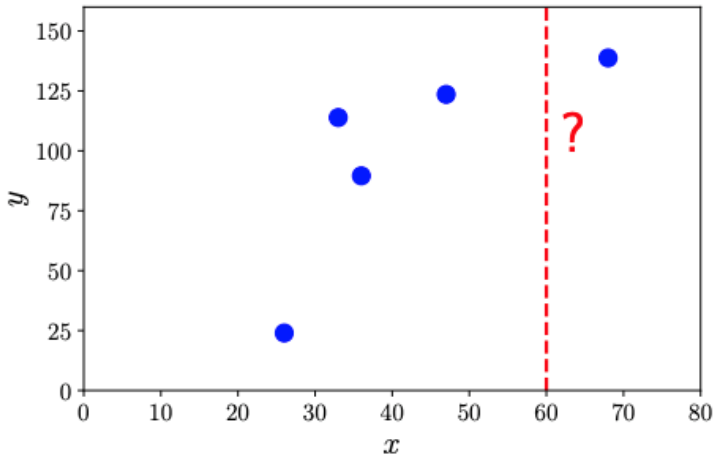
D columns

N rows

Degree	Latitude (in degrees)	Longitude (in degrees)	Age	Annual Salary (in thousands)
2	51.5073	0.1290	36	89.563
3	51.5074	0.1275	47	123.543
1	51.5071	0.1278	26	23.989
1	51.5075	0.1281	68	138.769
2	51.5074	0.1278	33	113.888

- A supervised learning algorithm
- We have a label y_n (the salary) associated with each example $\boldsymbol{x}_n$ (e.g., age).
- A dataset is written as a set of example-label pairs $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n), \ldots, (\boldsymbol{x}_N, y_N)\}$
- The table of examples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ are concatenated and written as $X \in \mathbb{R}^{N \times D}$

We are interested in:
What is the salary (y) at age 60 ($x = 60$)?



x: age
y: salary

8.1.2 Models as Functions

- Once we have data in an appropriate vector representation, we can construct a predictive function (known as a predictor).
- Here, a model means a predictor.
- A predictor is a function that, when given a particular input example (in our case, a vector of features), produces an output.

- For example,

$$f: \mathbb{R}^D \rightarrow \mathbb{R}$$

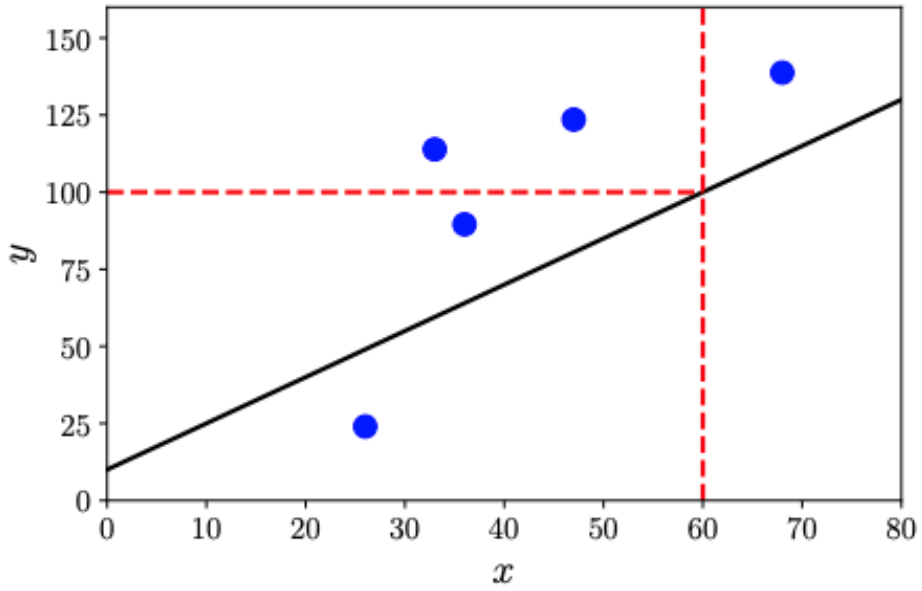
where the input $\boldsymbol{x}$ is a D-dimensional vector, and the output is a real-valued scalar. That is, the function f is applied to $\boldsymbol{x}$, written as $f(\boldsymbol{x})$ and returns a real number.

8.1.2 Models as Functions

- We mainly consider the special case of linear functions

$$f(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x} + \theta_0$$

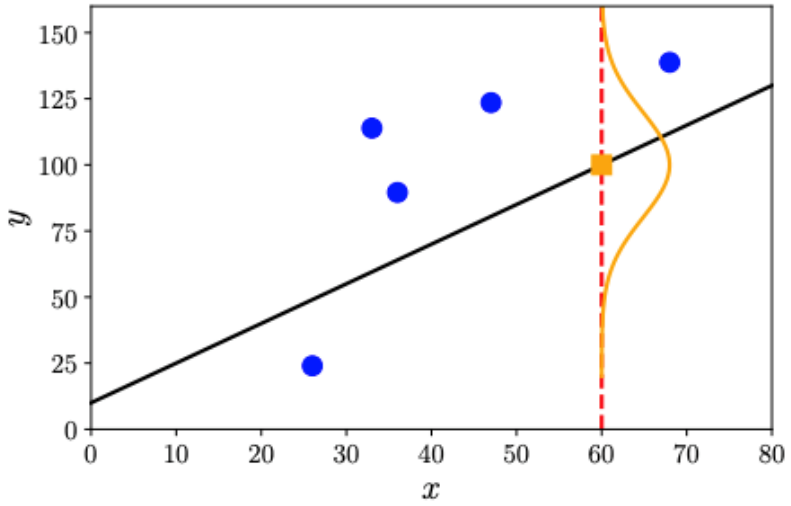
- Example: predicting salary $f(\boldsymbol{x})$ from age $\boldsymbol{x}$.



Black and solid diagonal line is an example predictor.
$f(60) = 100$

8.1.3 Models as Probability Distributions

- The observed data is usually a combination of the true underlying data and noise, i.e., $\tilde{x} = x + n$
- We wish to reveal x from $\tilde{x}$
- So we would like to have predictors that express some sort of uncertainty, e.g., to quantify the confidence we have about the value of the prediction for a particular test data point.



Example function (black solid diagonal line) and its predictive uncertainty at $x = 60$ (drawn as a Gaussian).

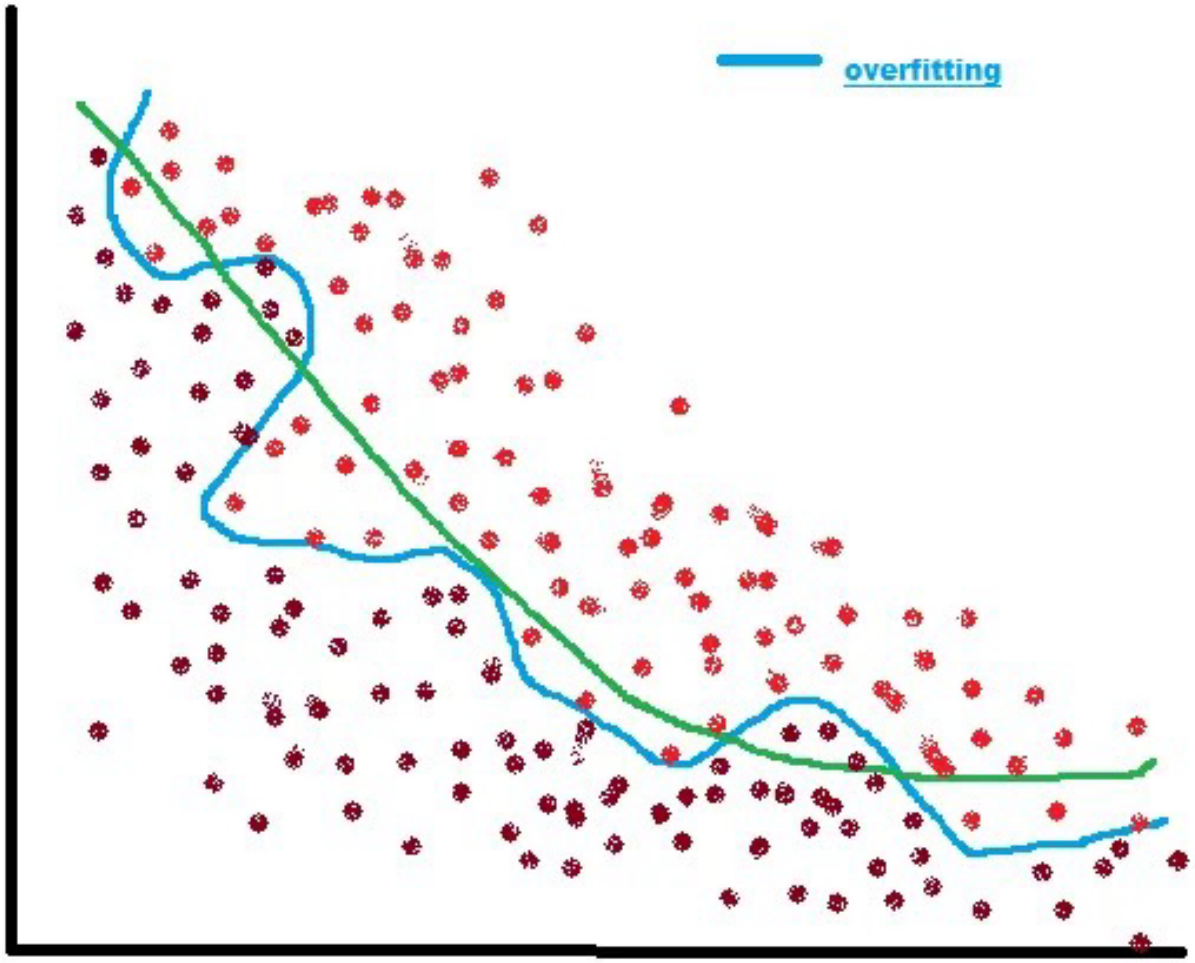
- Instead of considering a predictor as a single function, we could consider predictors to be probabilistic models.
- We will learn probability in later lectures

8.1.4 Learning is Finding Parameters

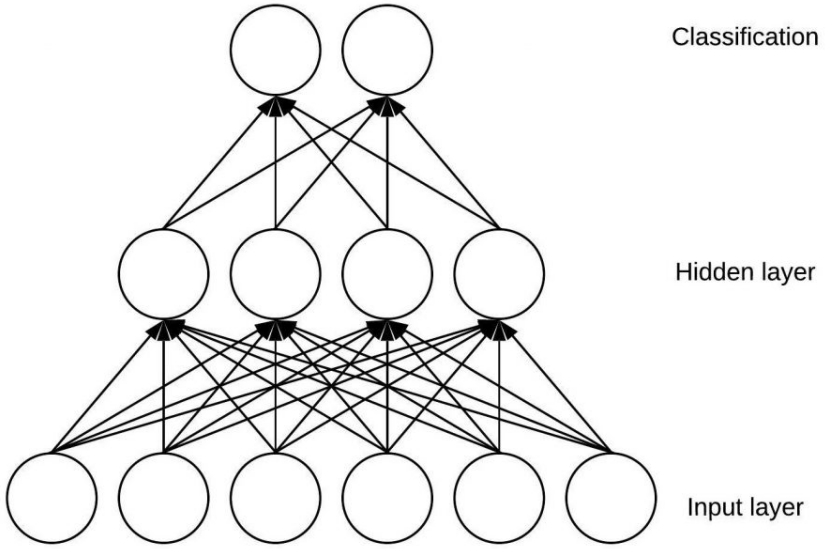
- The goal of learning is to find a model and its corresponding parameters such that the resulting predictor will perform well on unseen data.
- 3 algorithmic phases when discussing machine learning algorithms
- Prediction or inference
- Training or parameter estimation
- Hyperparameter tuning or model selection

- Prediction phase: we use a trained predictor on previously unseen test data
- The training or parameter estimation phase: we adjust our predictive model based on training data. We will introduce the empirical risk minimization for finding good parameters.
- We use cross-validation to assess predictor performance on unseen data.
- We also need to balance between fitting well on training data and finding "simple" explanations of the phenomenon. This trade-off is often achieved using regularization.

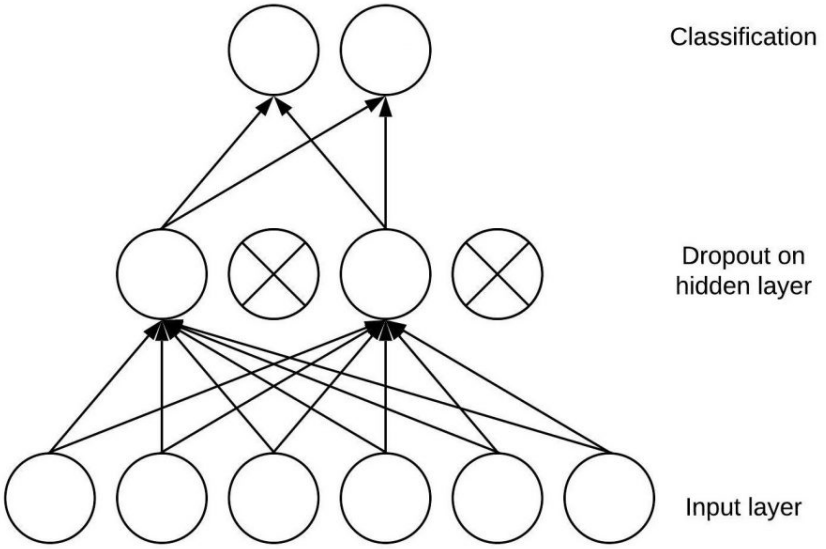
Regularization helps with overfitting.



Regularization in neural networks: dropout.



Without Dropout



With Dropout

Credit: Wenkang Wei

- Hyperparameter tuning or model selection
- We need to make high-level modeling decisions about the structure of the predictor. For example
- Number of layers to be used in deep learning
- Number of components in a Gaussian Mixture Model
- Weight of regularization terms

Hyperparameter

- The problem of choosing among different models/hyperparameters is called model selection

- Difference between parameters and hyperparameters
- Parameters are to be numerically optimized ($\sim 10^6$ weights in a deep network)
- Hyperparameters generally use search techniques and validation sets (neural architecture search [2]).

[2] Zoph et al., Neural architecture search with reinforcement learning, Arxiv 2016

8.2 Empirical Risk Minimization

- What does it mean to learn?
- Estimating parameters based on training data.
- Four questions will be answered
- What is the set of functions we allow the predictor to take? – Hypothesis class of functions
- How do we measure how well the predictor performs on the training data? -- Loss functions for training
- How do we construct predictors from only training data that performs well on unseen test data? -- regularization
- What is the procedure for searching over the space of models? -- Cross-Validation

8.2.1 Hypothesis Class of Functions

- We are given N examples $\boldsymbol{x}_n \in \mathbb{R}^D$ and corresponding scalar labels $y_n \in \mathbb{R}$.
- Supervised learning: we have pairs $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)$

- We want to estimate a predictor $f(\cdot, \boldsymbol{\theta}): \mathbb{R}^D \to \mathbb{R}$, parametrized by $\boldsymbol{\theta}$
- We hope to be able to find a good parameter $\boldsymbol{\theta}^*$ such that we fit the data well, that is

$$f(\boldsymbol{x}_n, \boldsymbol{\theta}^*) \approx y_n \text{ for all } n = 1, \ldots, N$$

- We use $\hat{y}_n = f(\boldsymbol{x}_n, \boldsymbol{\theta}^*)$ to represent the output of the predictor

Example (least-squares regression)

- When the label y_n is real-valued, a popular choice of function class for predictors is affine functions (linear functions).

$f(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x} + \theta_0$

- For more compact representations, we concatenate an additional unit feature $x^{(0)} = 1$ to $\boldsymbol{x}_n$, i.e.,

$$\boldsymbol{x}_n = \left[x_n^{(0)}, x_n^{(1)}, x_n^{(2)}, \dots, x_n^{(D)}\right]^{\mathrm{T}} = \left[1, x_n^{(1)}, x_n^{(2)}, \dots, x_n^{(D)}\right]^{\mathrm{T}}$$

- The parameter vector is $\boldsymbol{\theta} = [\theta_0, \theta_1, \theta_2, \dots, \theta_D]^T$
- We can write the predictor as follows

$$f(\boldsymbol{x}_n, \boldsymbol{\theta}) = \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n$$

which is equivalent to the affine model

$$f(\boldsymbol{x}_n, \boldsymbol{\theta}) = \theta_0 + \sum_{d=1}^{D} \theta_d x_n^{(d)} = \theta_0 x_n^{(0)} + \sum_{d=1}^{D} \theta_d x_n^{(d)} = \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n$$

Example (least-squares regression)

$$f(\boldsymbol{x}_n, \boldsymbol{\theta}) = \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n$$

- The predictor takes the vector of features representing a single example $\boldsymbol{x}_n$ as input and produces a real-valued output,

$$f\colon \mathbb{R}^{D+1} \to \mathbb{R}$$

- $f(\boldsymbol{x}_n, \boldsymbol{\theta}) = \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}_n$ is a linear predictor
- There are many non-linear predictors, such as the neural networks

8.2.2 Loss Function for Training

- In training, we aim to learn a model that fits the data well.
- To define "fits the data well", we specify a loss function

$$\ell\left(y_n, \hat{y}_n\right)$$

- Input: ground truth label y_n of a training example

  the prediction $\hat{y}_n$ of this training example
- Output: a non-negative number, called loss. It represents how much error we have made on this particular prediction
- To find good parameters $\boldsymbol{\theta}^*$, we need to minimize the average loss on the set of N training examples

- We usually assume training examples $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)$ are independent and identically distributed (i.i.d).

- Under the i.i.d assumption, the empirical mean is a good estimate of the population mean.
- We can use the empirical mean of the loss on the training data
- Given a training set $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$, we use the notation of an example matrix

$\boldsymbol{X} \coloneqq [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^{\mathrm{T}} \in \mathbb{R}^{N \times D}$

and a label vector

$\boldsymbol{y} = [y_1, \ldots, y_N]^{\mathrm{T}} \in \mathbb{R}^N$

- The average loss is given by

$$\mathbf{R}_{emp}(f, \boldsymbol{X}, \boldsymbol{y}) = \frac{1}{N} \sum_{n=1}^{N} \ell\left(y_n, \hat{y}_n\right)$$

where $\hat{y}_n = f(\boldsymbol{x}_n, \boldsymbol{\theta})$. The above equation is called the empirical risk. The learning strategy is called empirical risk minimization.

Example - Least-Squares Loss

- We use the squared loss function

$$\ell\left(y_n, \hat{y}_n\right) = \left(y_n - \hat{y}_n\right)^2$$

- We aim to minimize the empirical risk, which is the average of the losses over the training data.

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \frac{1}{N}\sum_{n=1}^{N} \ell\left(y_n, \hat{y}_n\right) = \min_{\boldsymbol{\theta}\in\mathbb{R}^D} \frac{1}{N}\sum_{n=1}^{N} \left(y_n - f(\boldsymbol{x}_n, \boldsymbol{\theta})\right)^2$$

Using the linear predictor $f(\boldsymbol{x}_n, \boldsymbol{\theta}) = \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_n$, we obtain the optimization problem

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \frac{1}{N}\sum_{n=1}^{N} \left(y_n - f(\boldsymbol{x}_n, \boldsymbol{\theta})\right)^2$$

- This equation can be equivalently expressed in matrix form

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \frac{1}{N}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|^2$$

- This is known as the least-squares problem. There exists a closed-form analytic solution for this by solving the normal equations. We will discuss it in later lectures

- We actually want to find a predictor f that minimizes the expected risk (or the population risk)

$$\mathbf{R}_{\text{true}}(f) = \mathbb{E}_{x,y}\left[\ell\left(y, f(\boldsymbol{x})\right)\right]$$

where y is the ground truth label and $f(\boldsymbol{x})$ is the prediction based on the example $\boldsymbol{x}$.

- $\mathbf{R}_{\text{true}}(f)$ is the true risk, if we can access an infinite amount of data
- The expectation $\mathbb{E}$ is over the infinite set of all possible data and labels.

- We actually want to find a predictor f that minimizes the expected risk (or the population risk)

$$\mathbf{R}_{\text{true}}(f) = \mathbb{E}_{x,y}\left[\ell\left(y, f(\boldsymbol{x})\right)\right]$$

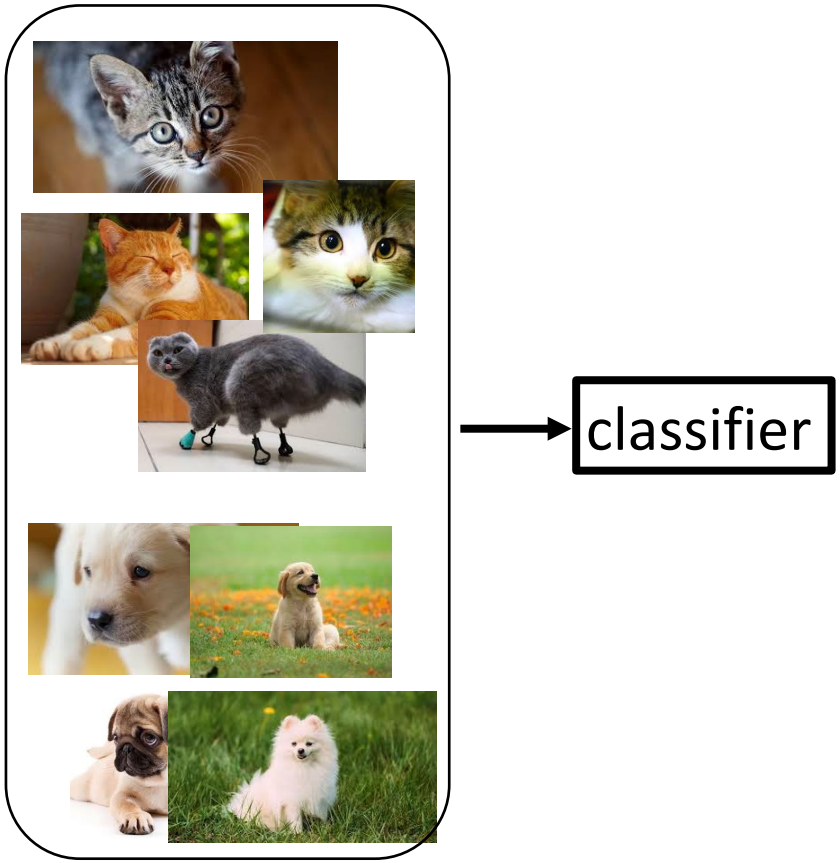
where y is the ground truth label and $f(\boldsymbol{x})$ is the prediction based on the example $\boldsymbol{x}$.

- $\mathbf{R}_{\text{true}}(f)$ is the true risk, if we can access an infinite amount of data
- The expectation $\mathbb{E}$ is over the infinite set of all possible data and labels.

>> Discussion

- Machine learning applications have different types of performance measure.
  - For classification: accuracy, AUC, F1 score, etc.
  - For detection: mean average precision, mIoU, etc.
  - For image denoise/super resolution: SSIM, PSNR, etc.
- In principle, the loss function should correspond to the measure.
- However, there are often mismatches between loss functions and the measures – due to implementation/optimization considerations.

We start with training a classifier



Training data

Deng and Zheng, “Are labels always necessary for classifier accuracy evaluation?” CVPR 2021.

We do a bit testing….

Correct prediction

classifier

Dog 0.05
Cat 0.95

Cat

Testing image

Prediction result = Ground truth

Wrong prediction

classifier

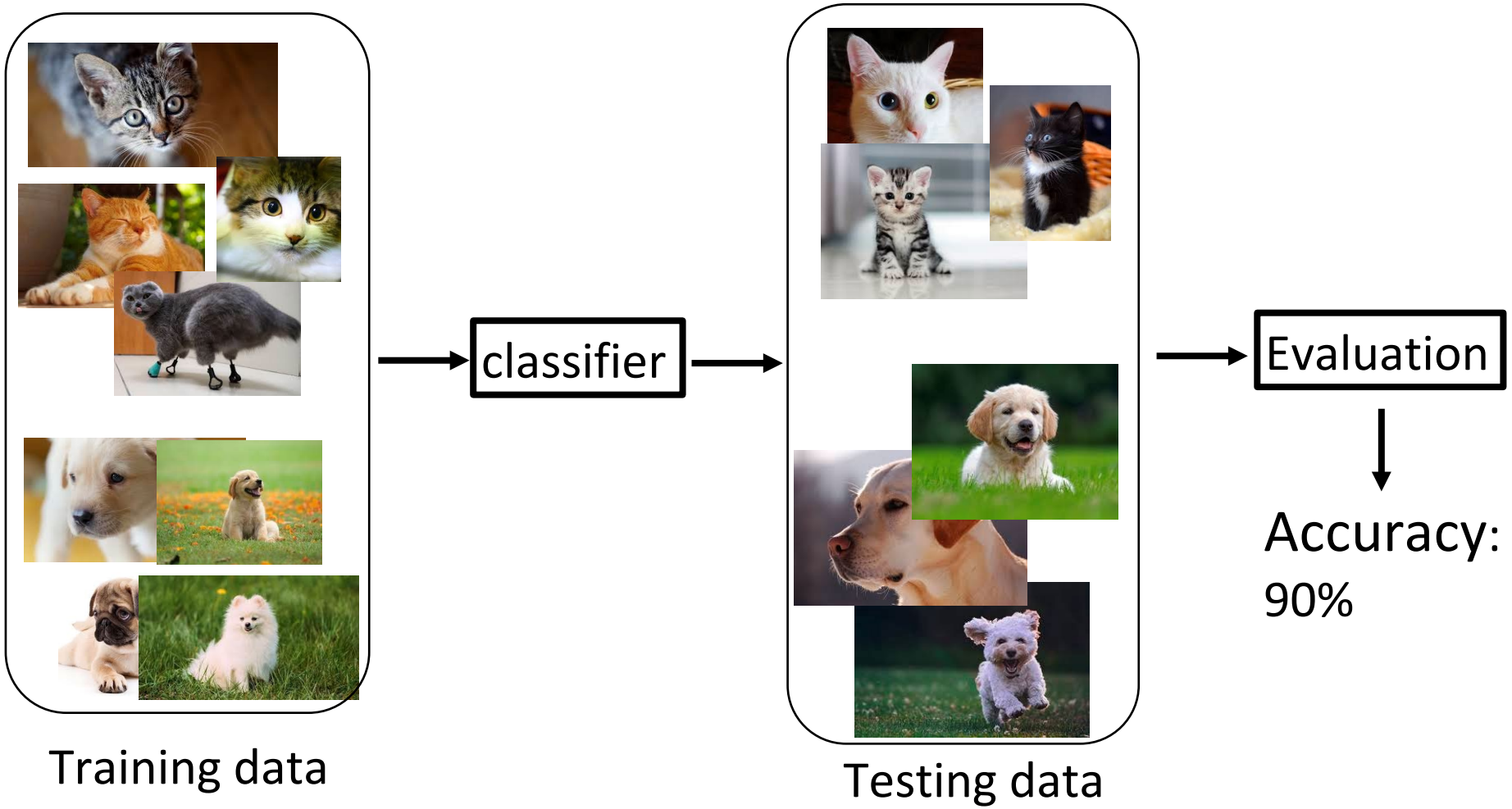
Dog 0.85
Cat 0.15

Cat

Testing image

Prediction result ≠ Ground truth

We now evaluate a model

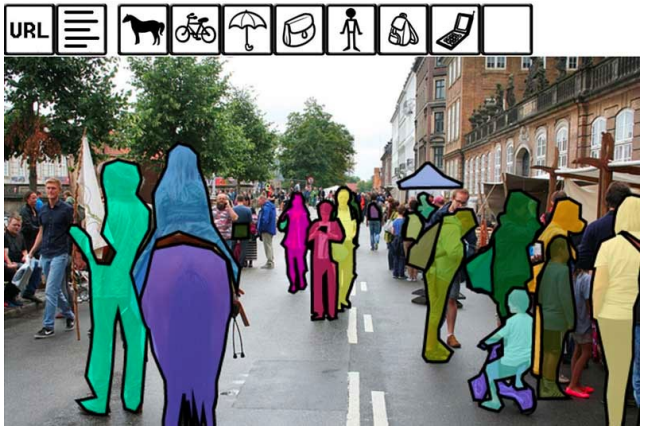


Ground truths provided

Is this way of evaluation feasible?

- Yes, when you have ground truths

ImageNet



MSCOCO

Ground truths provided

LFW

Overfitting

Worry about the data

- The aim of a machine learning predictor is to perform well on unseen data.
- We simulate the unseen data by holding out a proportion of the whole dataset.
- This hold-out set is called the test set.
- In practice, we split data into a training set and a test set.

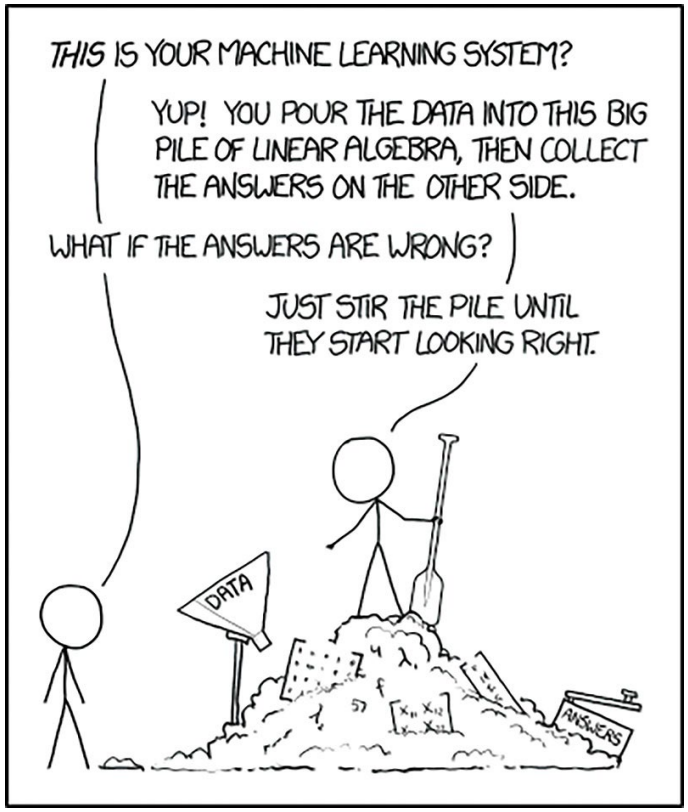
Worry about the data

- Training set: the model “sees” and “learns” from this data.
- Test set: not seen during training, used to evaluate the unbiased g

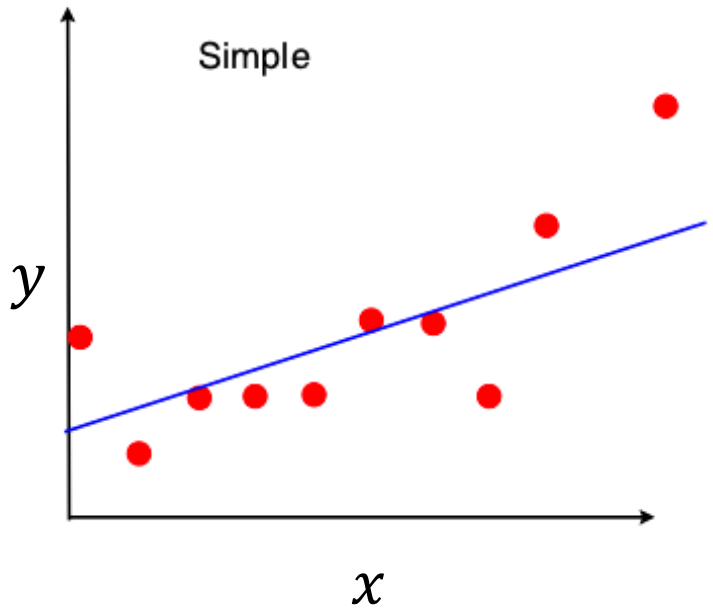
- Validation set: used to provide an unbiased evaluation of a model while tuning model hyperparameters.
- We can use cross-validation to get the validation set.

Worry about the data

- The user should not cycle back to a new round of training after having observed the test set.
- **Never test with the training set.**



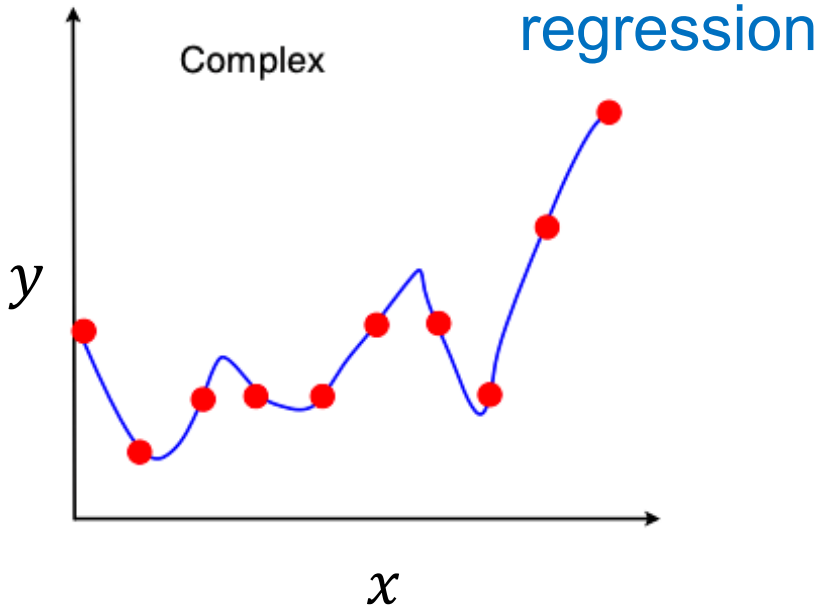
- Empirical risk minimization can lead to overfitting.
- The predictor fits too closely to the training data and does not generalize well to new data.



This simple model fits the training data less well.

A larger empirical risk.

A good machine learning model.



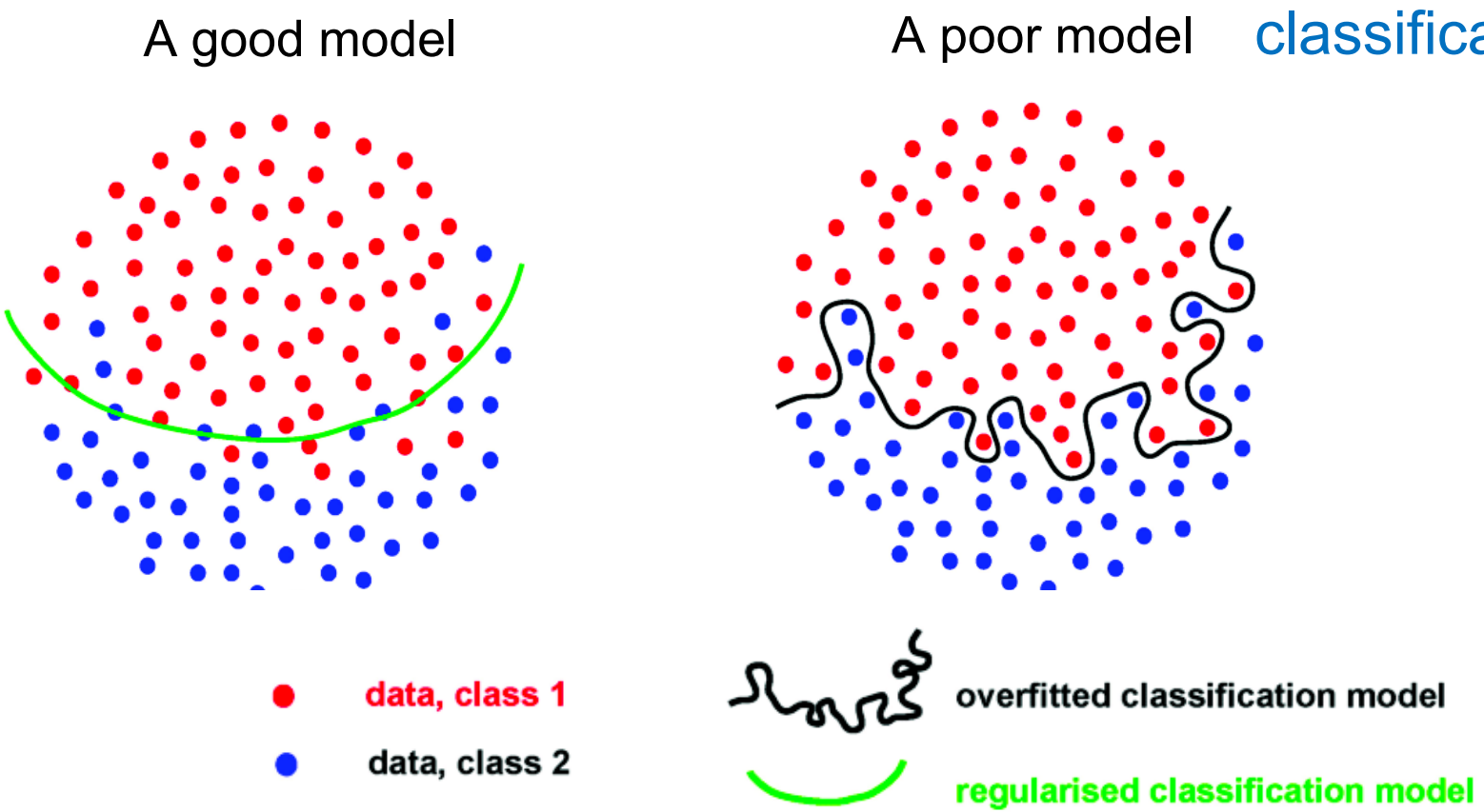
regression

This complex model fits the training data very well.

A very small empirical risk.

A poor machine learning model due to overfitting.

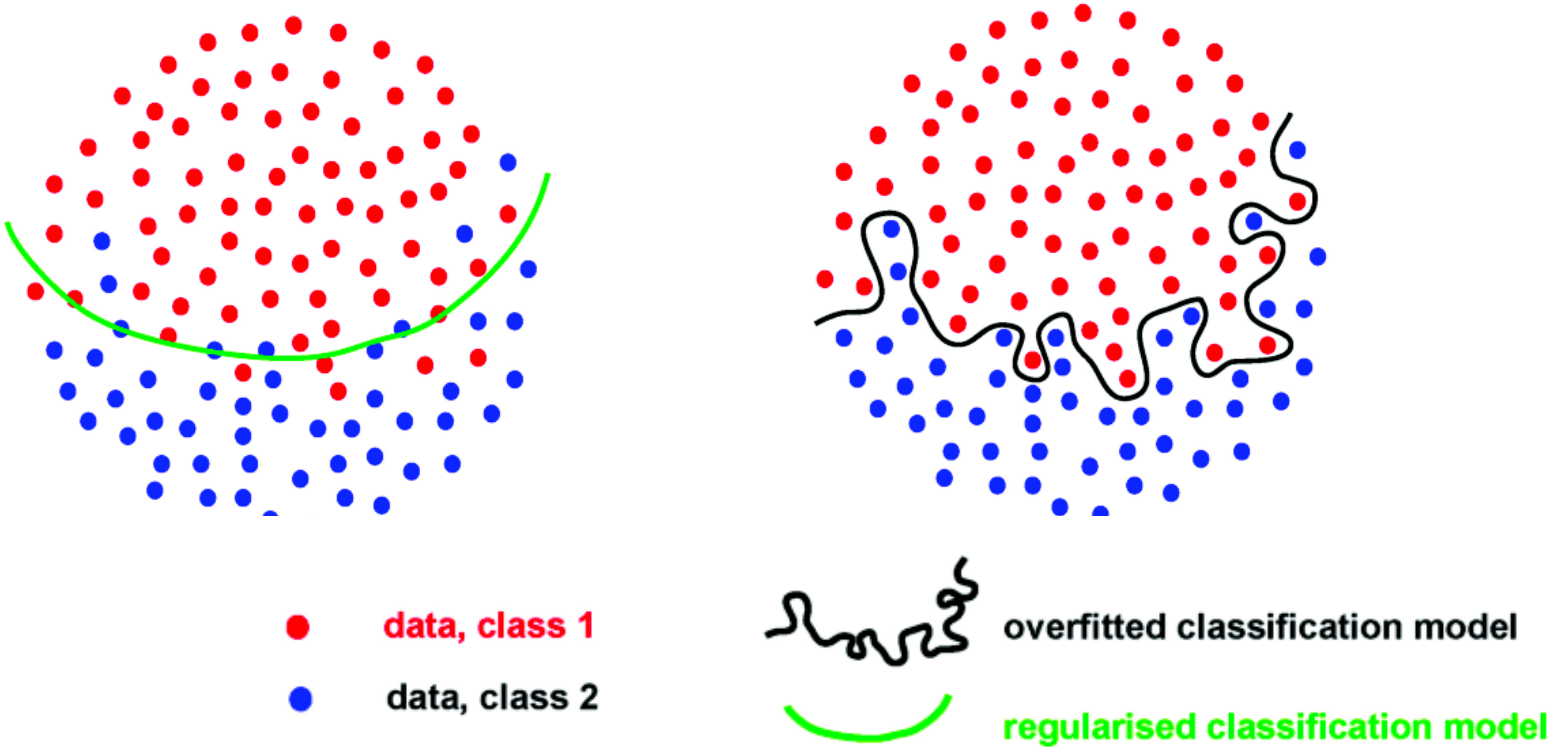
- Empirical risk minimization can lead to overfitting.
- The predictor fits too closely to the training data and does not generalize well to new data.



- Empirical risk minimization can lead to overfitting.
- The predictor fits too closely to the training data and does not generalize well to new data.

A good model

A poor model classification



>> Discussion

Regularization

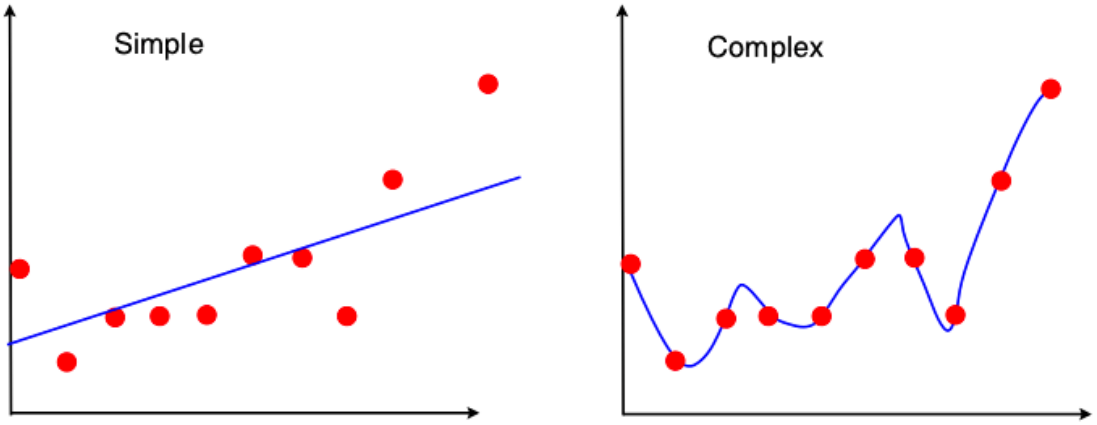
8.2.3 Regularization to Reduce Overfitting

- When overfitting happens, we have
  - very small average loss on the training set but large average loss on the test set

- Given a predictor f, overfitting occurs when
  - the risk estimate from the training data $\mathbf{R}_{\text{emp}}(f, \boldsymbol{X}_{\text{train}}, \boldsymbol{y}_{\text{train}})$ underestimates the expected risk $\mathbf{R}_{\text{true}}(f)$. In other words,
  - $\mathbf{R}_{\text{emp}}(f, \boldsymbol{X}_{\text{train}}, \boldsymbol{y}_{\text{train}})$ is much smaller than $\mathbf{R}_{\text{true}}(f)$ which is estimated using $\mathbf{R}_{\text{emp}}(f, \boldsymbol{X}_{\text{test}}, \boldsymbol{y}_{\text{test}})$

- Overfitting occurs usually when
  - we have little data and a complex hypothesis class

- How to prevent overfitting?



- We can bias the search for the minimizer of empirical risk by introducing a penalty term.
- The penalty term makes it harder for the optimizer to return an overly flexible predictor.
- The penalty term is called regularization.
- Regularization is an approach that discourages complex or extreme solutions to an optimization problem.

- Example
- Least-squares problem

$$\min_{\boldsymbol{\theta}} \frac{1}{N} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|^2$$

- Example
- Least-squares problem

$$\min_{\boldsymbol{\theta}} \frac{1}{N} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|^2$$

- To regularize this formulation, we add a penalty term

$$\min_{\boldsymbol{\theta}} \frac{1}{N} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|^2 + \lambda \|\boldsymbol{\theta}\|^2$$

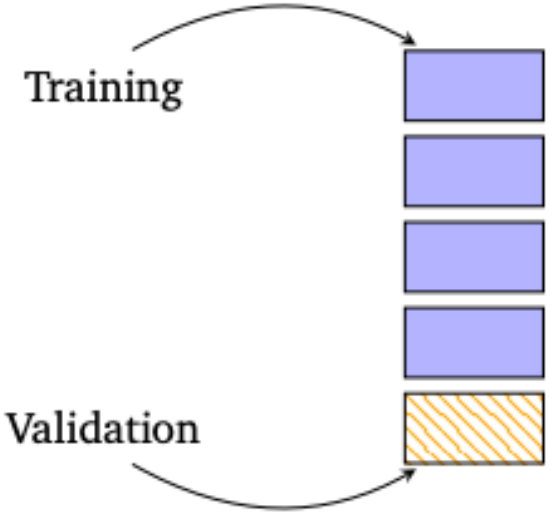
- The addition term $\|\boldsymbol{\theta}\|^2$ is called the regularizer or penalty term, and the parameter regularizer λ is the regularization parameter.

- λ enables a trade-off between minimizing the loss on the training set and the amplitude of the parameters $\boldsymbol{\theta}$
- It often happens that the amplitude of the parameters in $\boldsymbol{\theta}$ becomes relatively large if we run into overfitting
- λ is a hyperparameter

Cross-validation

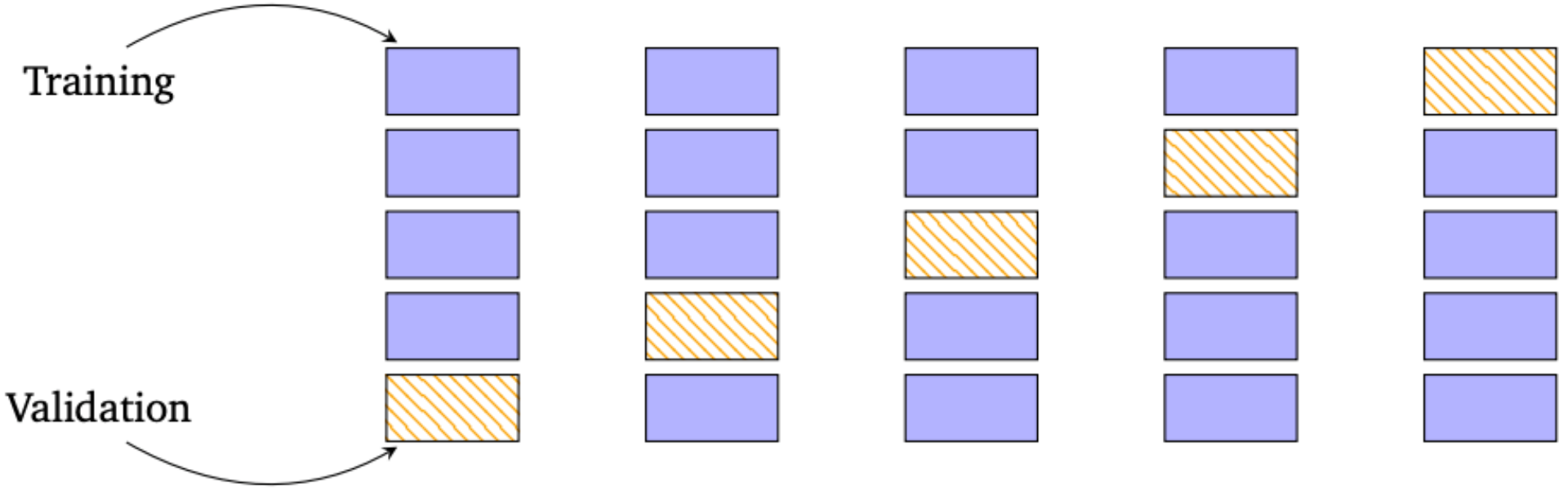
8.2.4 Cross-Validation to Assess the Generalization Performance

- We mentioned that we split a dataset into a training set and a test set.
- We measure the final generalization error by applying the predictor to test data.
- We can use the validation set for estimating the generalization error during the hyperparameter tuning phase.
- If no hyperparameter tuning phase, the test data is sometimes referred to as the validation set.
- We want the training set to be large.
- That leaves the validation set small.
- A small validation set makes the result less stable (large variances).



- Basically, we want the training set to be large
- We want the validation to be large, too
- How to solve these contradictory objectives?
- Cross-validation: K-fold cross-validation

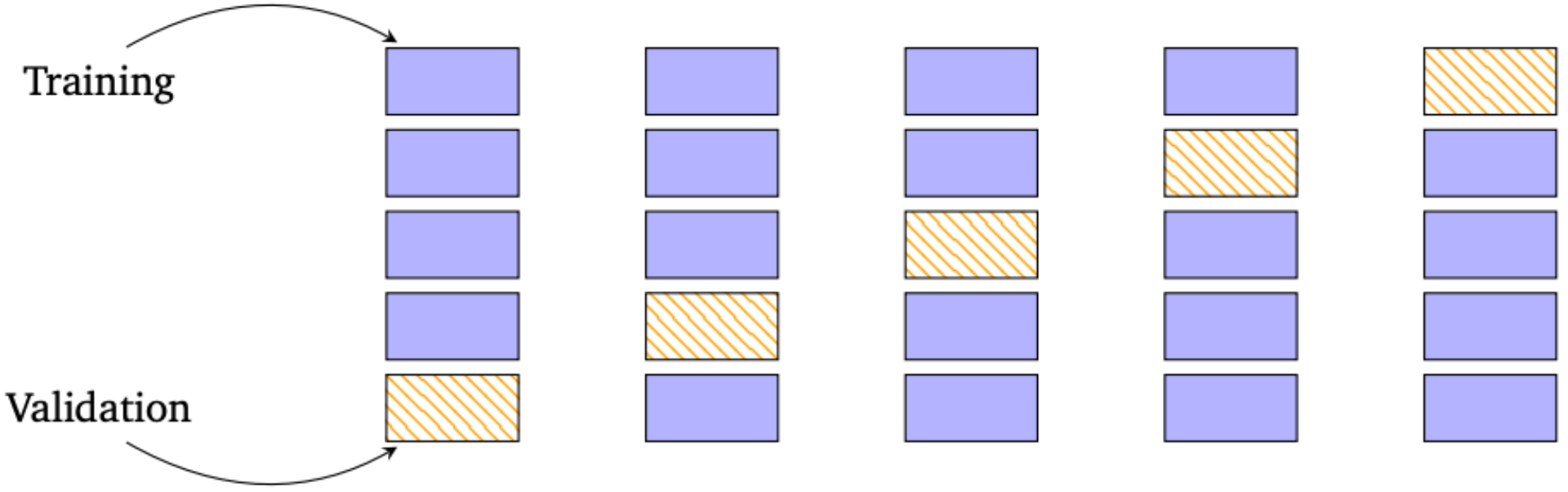
Example: $K = 5$



Cross-validation

- K-fold cross-validation partitions the data into K chunks
- $K-1$ trunks form the training set $\mathcal{R}$
- The last trunk is the validation set $\mathcal{V}$
- This procedure is repeated for all K choices for the validation set, and the performance of the model from the K runs is averaged

Example: $K = 5$



Cross-validation

- Formally, we partition our dataset into two sets $\mathcal{D} = \mathcal{R} \cup \mathcal{V}$, such that they do not overlap, i.e., $\mathcal{R} \cap \mathcal{V} = \phi$
- We train on our model on $\mathcal{R}$ (training set)
- We evaluate our model on $\mathcal{V}$ (validation set)
- We have K partitions. In each partition k:
  - The training set $\mathcal{R}^{(k)}$ produces a predictor $f^{(k)}$
  - $f^{(k)}$ is applied to the validation set $\mathcal{V}^{(k)}$ to compute the empirical risk $\mathrm{R}\left(f^{(k)}, \mathcal{V}^{(k)}\right)$

- All the empirical risks are averaged to approximate the expected generalization error.

$$\mathbb{E}_V[R(f, \mathcal{V})] \approx \frac{1}{K} \sum_{k=1}^{K} \mathrm{R}\left(f^{(k)}, \mathcal{V}^{(k)}\right)$$

Cross-validation – key insights

- The training set is limited -- not producing the best $f^{(k)}$
- The validation set is limited – producing an inaccurate estimation of $\mathrm{R}\left(f^{(k)}, \mathcal{V}^{(k)}\right)$
- After averaging, the results are stable and indicative
- An extreme: leave-one-out cross-validation, where the validation set only contains one example.

- A potential drawback – computation cost
  - The training can be time-consuming
  - Difficult to evaluate many model hyperparameters.
- This problem can be solved by parallel computing, given enough computational resources

Check your understanding

- When your model works poorly on the training set, your model will also work poorly on the test set.
- When your model works poorly on the training set, your model may be overfitting.
- Overfitting happens when your model is too complex, given your training data.
- Regularization alleviates overfitting by improving the complexity of your training data.
- We get more stable test accuracy if K increases in K-fold cross-validation.
- In 2-fold cross-validation, you can obtain 2 results from the 2 test sets, which may differ significantly.

Check your understanding

- When your model works poorly on the training set, your model will also work poorly on the test set. Probably Y
- When your model works poorly on the training set, your model may be overfitting. N
- Overfitting happens when your model is too complex, given your training data. Y
- Regularization alleviates overfitting by improving the complexity of your training data. N
- We get more stable test accuracy if K increases in K-fold cross-validation. Y
- In 2-fold cross-validation, you can obtain 2 results from the 2 test sets, which may differ significantly. Y