

# Machine Learning: Overview

## CS115 - Math for Computer Science

Ngoc Hoang Luong

University of Information Technology (UIT)

*hoangln@uit.edu.vn*

September 8, 2022

# What is machine learning?

*A computer program is said to learn from experience  $E$  with respect to some class of tasks  $T$ , and performance measure  $P$ , if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$ .*

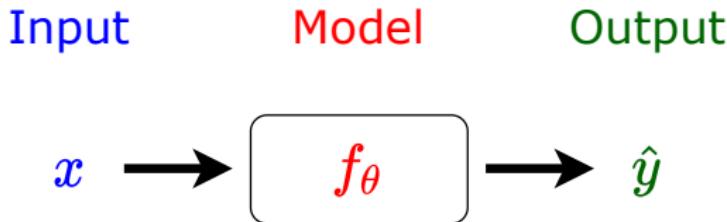
— Tom Mitchell

There are many different kinds of ML, depending on

- the task  $T$  we wish the system to learn
- the performance measure  $P$  we use to evaluate the system
- the training signal or experience  $E$  we give it.

# What is machine learning?

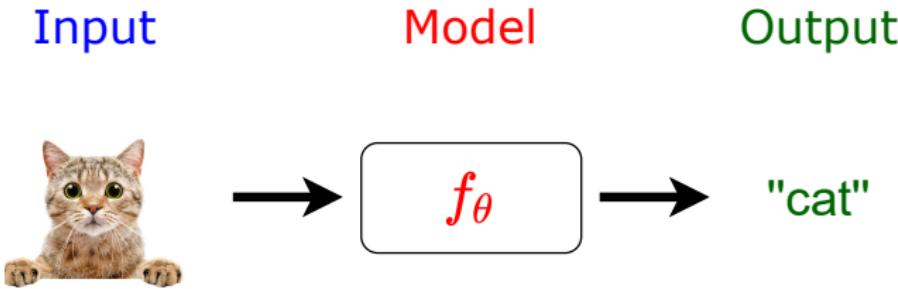
- **Supervised learning.** We get observed inputs  $\mathcal{D} = \{\mathbf{x}_n : n = 1 : N\}$  and corresponding outputs  $y_n$ . We learn a mapping  $f$  from inputs to outputs.



- **Unsupervised learning.** We just get observed inputs  $\mathcal{D}$  without any corresponding outputs. We try to make sense of data.
- **Reinforcement learning.** An **agent** has to learn how to interact with its environment. This can be encoded by a **policy**  $a = \pi(s)$ , which specifies which action  $a$  to perform in response to each possible input  $s$  (environmental state).

# Supervised Learning - Classification

Learning a mapping  $f_{\theta} : \mathbb{R}^{W \times H} \rightarrow \{"\text{cat}", "\text{non-cat}"\}$ .



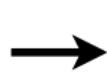
# Supervised Learning - Regression

Learning a mapping  $f_{\theta} : \mathbb{R}^N \rightarrow \mathbb{R}$ .

Input

Model

Output



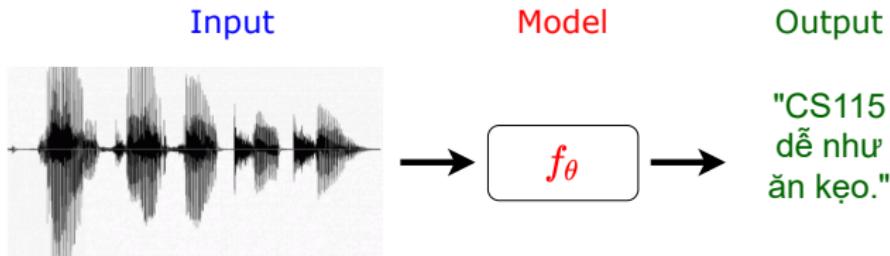
$$f_{\theta}$$



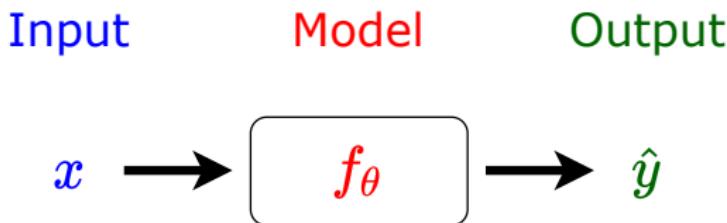
\$50,312

# Supervised Learning - Structured Prediction

Learning a mapping  $f_{\theta} : \mathbb{R}^N \rightarrow \{1, 2, \dots, L\}^M$ .



# Supervised learning



The most common form of ML is **supervised learning**.

- The task  $T$  is to learn a mapping  $f$  from inputs  $x \in \mathcal{X}$  to outputs  $y \in \mathcal{Y}$ : Estimate **parameters**  $\theta$  from training data  $\{(x_n, y_n)\}_{n=1}^N$ .
- The inputs  $x$  are also called the **features**, or **predictors**.
- The output  $y$  is also known as the **label**, **target**, or **response**.
- The experience  $E$  is a set of  $N$  input-output pairs  $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N$ , known as the **training set**.  $N$  is called the **sample size**.
- The performance measure  $P$  depends on the type of output we are predicting.

# Classification

In **classification** problems, the output space is

- a set of  $C$  unordered and mutually exclusive labels known as **classes**  
 $\mathcal{Y} = \{1, 2, \dots, C\}$ .
- If there are just two classes,  $y \in \{0, 1\}$  or  $y \in \{+1, -1\}$ , it is called **binary classification**.



(a)



(b)



(c)

**Figure:** Classifying Iris flowers into 3 subspecies: setosa, versicolor and virginica.

# Classification

## In image classification,

- The  $\mathcal{X}$  is the set of images, which is very high-dimensional.
- For a color image with  $C = 3$  channels (RGB) and  $D_1 \times D_2$  pixels, we have  $\mathcal{X} = \mathbb{R}^D$ , where  $D = C \times D_1 \times D_2$ .
- Learning a mapping  $f : \mathcal{X} \rightarrow \mathcal{Y}$  from images to labels is challenging.

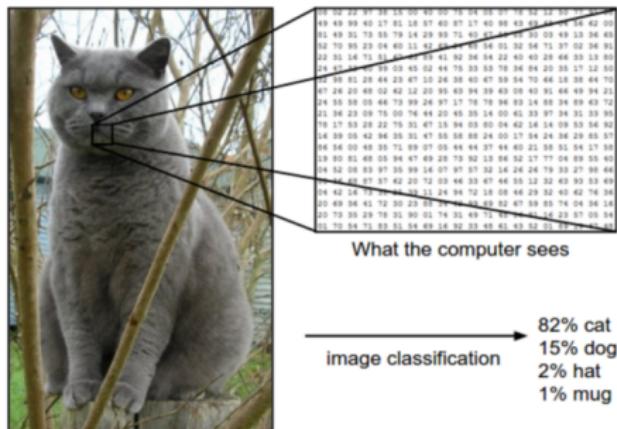


Figure: Illustration of the image classification problem.

# Classification

## Machine Learning:

Sample  
↓  
Label



dog



cat



horse

## Human Learning:

We learn through



Examples

Long Ear Black nose  
↓  
dog



Diagrams

Comparisons

Figure: Machine learning vs. Human learning<sup>1</sup>

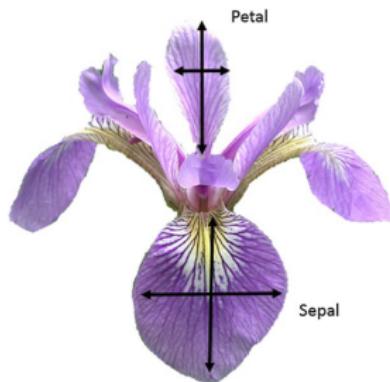
---

<sup>1</sup><https://blog.ml.cmu.edu/2019/03/29/building-machine-learning-models-via-comparisons/>

# Classification

The **Iris dataset** is a collection of 150 labeled examples of Iris flowers, 50 of each type, described by 4 features

- sepal length, sepal width, petal length, petal width.
- The input space is much lower-dimensional  $\mathcal{X} = \mathbb{R}^4$ .
- These numeric features are highly informative.

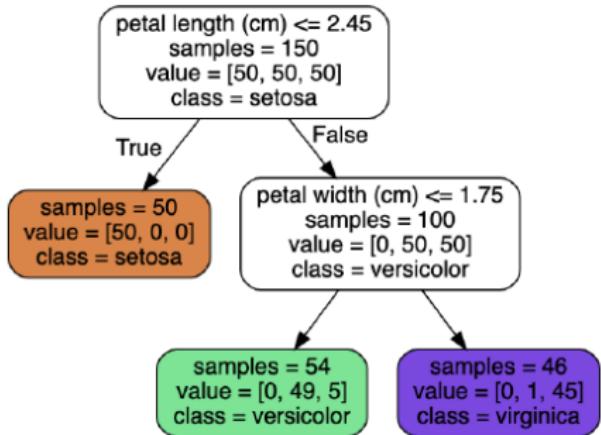


| index | sl  | sw  | pl  | pw  | label      |
|-------|-----|-----|-----|-----|------------|
| 0     | 5.1 | 3.5 | 1.4 | 0.2 | setosa     |
| 1     | 4.9 | 3.0 | 1.4 | 0.2 | setosa     |
|       | ... |     |     |     |            |
| 50    | 7.0 | 3.2 | 4.7 | 1.4 | Versicolor |
|       | ... |     |     |     |            |
| 149   | 5.9 | 3.0 | 5.1 | 1.8 | Virginica  |

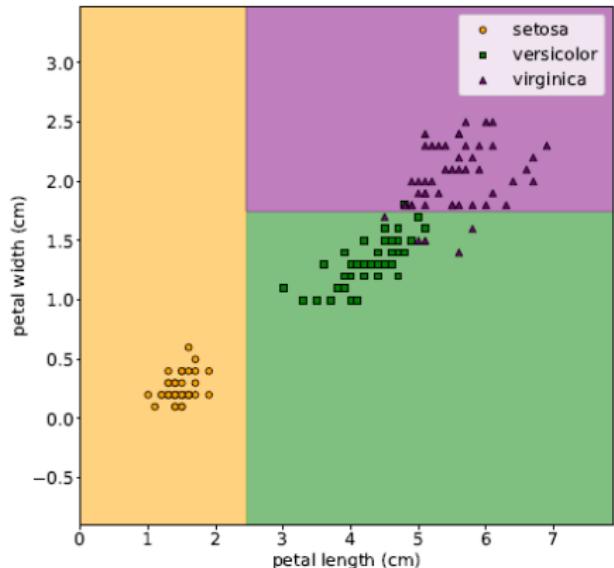
Figure: A subset of the Iris design matrix.

- A small dataset of features can be stored as an  $N \times D$  **design matrix**.

# Learning a classifier



(a)



(b)

Figure: A decision tree on Iris dataset, using just petal length and width features.

# Learning a classifier

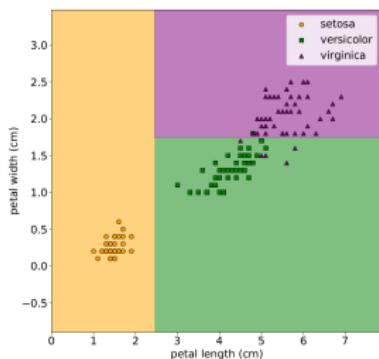


Figure: Decision surface

- We apply a **decision rule** to the Iris dataset

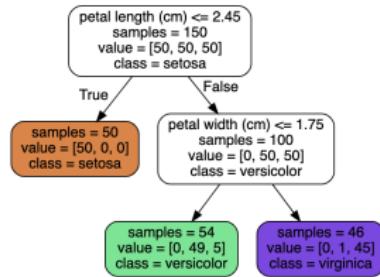
$$f_1(\mathbf{x}; \theta) = \begin{cases} \text{Setosa if petal length} < 2.45 \\ \text{Versicolor or Virginica otherwise} \end{cases}$$

- This rule defines a 1d **decision boundary** at  $x_{\text{petal length}} = 2.45$ , partitioning the input space into 2 regions.
- We add another **decision rule** to inputs that fail the first rule.

$$f_2(\mathbf{x}; \theta) = \begin{cases} \text{Versicolor if petal width} < 1.75 \\ \text{Virginica otherwise} \end{cases}$$

- Both rules induces a 2d **decision surface**.

# Learning a classifier



- We can arrange the rules  $f_1, f_2$  into a tree structure, called a **decision tree**.
- We can represent the tree by storing, for each internal node:
  - the feature index that is used.
  - the corresponding threshold value.
- We denote all these **parameters** by  $\theta$ .

Figure: Decision tree

# Empirical risk minimization

The goal of supervised learning is

- to automatically come up with classification models.
- to reliably predict the labels for any given input.

The **misclassification rate** on the training set can be used to measure performance on this task:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N \mathbb{I}(y_n \neq f(\mathbf{x}_n; \boldsymbol{\theta}))$$

where  $\mathbb{I}(e)$  is the binary **indicator function**:

$$\mathbb{I}(e) = \begin{cases} 1 & \text{if } e \text{ is true} \\ 0 & \text{if } e \text{ is false} \end{cases}$$

This assumes all errors are equal.

# Empirical risk minimization

- It may happen that some errors are more costly than others.
- Suppose that setosa & Versicolor are tasty but Virginica is poisonous.
- We might use the asymmetric **loss function**  $\ell(y, \hat{y})$  as below.

|       |            | Estimate |            |           |
|-------|------------|----------|------------|-----------|
|       |            | Setosa   | Versicolor | Virginica |
| Truth | Setosa     | 0        | 1          | 1         |
|       | Versicolor | 1        | 0          | 1         |
|       | Virginica  | 10       | 10         | 0         |

The **empirical risk** is the average loss of the predictor on the training set:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N \ell(y_n, f(\mathbf{x}_n; \boldsymbol{\theta}))$$

The misclassification rate is equal to the empirical risk when we use **zero-one loss**:

$$\ell_{01}(y, \hat{y}) = \mathbb{I}(y \neq \hat{y})$$

# Empirical risk minimization

The problem of **model fitting** or **training** can be defined as finding a setting of parameters that minimizes the empirical risk on the training set:

$$\hat{\theta} = \operatorname{argmin}_{\theta} \mathcal{L}(\theta) = \operatorname{argmin}_{\theta} \frac{1}{N} \sum_{n=1}^N \ell(y_n, f(\mathbf{x}_n; \theta))$$

This is called **empirical risk minimization**.

However, our true goal is:

- to minimize the expected loss on **future data** that we have not seen,
- to **generalize**, rather than just do well on the training set.

# Uncertainty

We can't perfectly predict the exact output given the input, due to

- lack of knowledge of the input-output mapping (**model uncertainty**)
- and/or intrinsic (irreducible) stochasticity in the mapping (**data uncertainty**).

We capture our uncertainty using **conditional probability distribution**:

$$p(y = c | \mathbf{x}; \boldsymbol{\theta}) = f_c(\mathbf{x}; \boldsymbol{\theta})$$

where  $f : \mathcal{X} \rightarrow [0, 1]^C$  maps inputs to a probability distribution over the  $C$  possible output labels.

# Uncertainty

Since  $f_c(\mathbf{x}; \boldsymbol{\theta})$  returns the probability of class label  $c$ , we require

$$0 \leq f_c \leq 1 \text{ for each } c, \text{ and } \sum_{c=1}^C f_c = 1$$

To avoid this restriction, it is common to require the model to return unnormalized log-probabilities. We can then use the **softmax function** to convert these to probabilities

$$\mathcal{S}(\mathbf{a}) = \left[ \frac{e^{a_1}}{\sum_{c=1}^C e^{a_c}}, \dots, \frac{e^{a_C}}{\sum_{c=1}^C e^{a_c}} \right]$$

$$\mathcal{S} : \mathbb{R}^C \rightarrow [0, 1]^C, \text{ and satisfied } 0 \leq \mathcal{S}(\mathbf{a})_c \leq 1 \text{ and } \sum_{c=1}^C \mathcal{S}(\mathbf{a})_c = 1.$$

The inputs to the softmax,  $\mathbf{a} = f(\mathbf{x}; \boldsymbol{\theta})$ , are called **logits**.

$$p(y = c | \mathbf{x}; \boldsymbol{\theta}) = \mathcal{S}_c(f(\mathbf{x}; \boldsymbol{\theta}))$$

# Uncertainty

The overall model is defined as

$$p(y = c|x; \theta) = \mathcal{S}_c(f(x; \theta))$$

A special case is when  $f$  is an **affine function**

$$f(x; \theta) = b + \mathbf{w}^T \mathbf{x} = b + w_1 x_1 + w_2 x_2 + \dots + w_D x_D$$

where  $\theta = (b, \mathbf{w})$  are the parameters of the model.  $\mathbf{w}$  are called the **weights** and  $b$  is called the **bias**. This model is called **logistic regression**. We can define  $\tilde{\mathbf{w}} = [b, w_1, \dots, w_D]$  and  $\tilde{\mathbf{x}} = [1, x_1, \dots, x_D]$  so that

$$\tilde{\mathbf{w}}^T \tilde{\mathbf{x}} = b + \mathbf{w}^T \mathbf{x}$$

This converts the affine function into a **linear function**. For simplicity, we can just write the prediction function as

$$f(x; \theta) = \mathbf{w}^T \mathbf{x}$$

# Maximum likelihood estimation

- When fitting probabilistic models, we can use the **negative log probability** as our loss function:

$$\ell(y, f(\mathbf{x}; \boldsymbol{\theta})) = -\log p(y|f(\mathbf{x}; \boldsymbol{\theta}))$$

- A good model is one that assigns a high probability to the true output  $y$  for each corresponding input  $\mathbf{x}$
- The average negative log probability of the training set is given by

$$\text{NLL}(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{n=1}^N \log p(y_n|f(\mathbf{x}_n; \boldsymbol{\theta}))$$

- This is called the **negative log likelihood**.
- If we minimize this, we can compute the **maximum likelihood estimate** (MLE):

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} = \operatorname{argmin}_{\boldsymbol{\theta}} \text{NLL}(\boldsymbol{\theta})$$

# Regression

- Suppose we want to predict a real-valued quantity  $y \in \mathbb{R}$  instead of a class label  $y \in \{1, \dots, C\}$ .
- E.g., in the case of Iris flowers,  $y$  might be the average height of the plant, or the degree of toxicity.

This type of problem is known as **regression**.

# Regression

- Regression is very similar to classification, but since the output  $\hat{y} = f(\mathbf{x}; \boldsymbol{\theta})$  is real-valued, we need to use a different loss function.
- Why don't we use the misclassification rate with the zero-one loss?

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N \ell_{01}(y_n, \hat{y}_n) = \frac{1}{N} \sum_{n=1}^N \mathbb{I}(y_n \neq f(\mathbf{x}_n; \boldsymbol{\theta}))$$

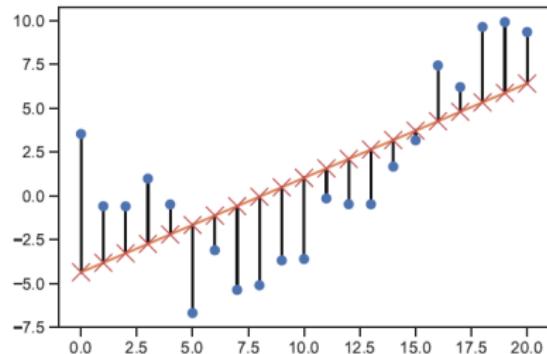
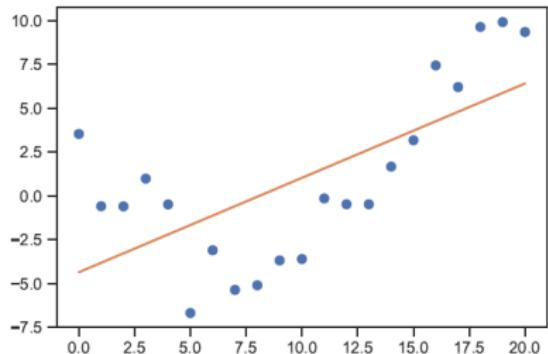
- The most common choice is **quadratic loss**, or  $\ell_2$  loss.

$$\ell_2(y, \hat{y}) = (y - \hat{y})^2$$

- This penalizes large **residuals**  $y - \hat{y}$  more than small ones.
- The empirical risk when using quadratic loss is equal to the **mean squared error** (MSE):

$$\mathcal{L}(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N (y_n - f(\mathbf{x}_n; \boldsymbol{\theta}))^2$$

# Linear Regression



We can fit the 1d data above using a **simple linear regression** model:

$$f(x; \theta) = b + wx$$

where  $w$  is the **slope**, and  $b$  is the **offset**.  $\theta = (w, b)$  are the parameters of the model. By adjusting  $\theta$ , we can minimize the sum of squared errors until we find the **least squares solution**:

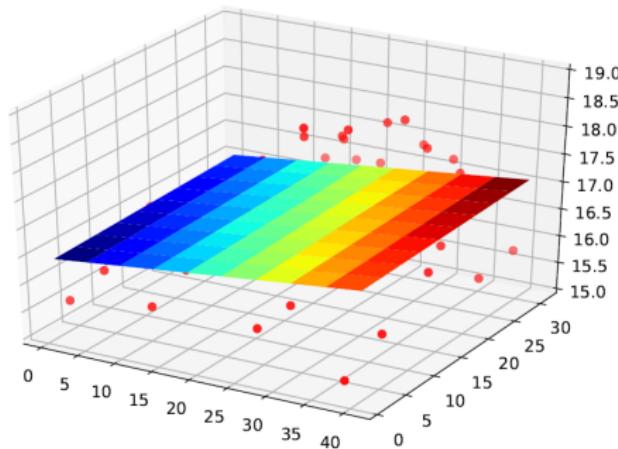
$$\hat{\theta} = \operatorname{argmin}_{\theta} \text{MSE}(\theta)$$

# Linear Regression

If we have multiple input features, we can write

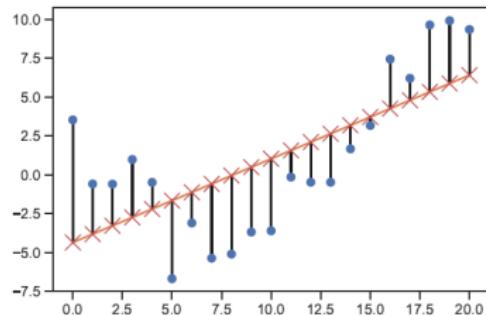
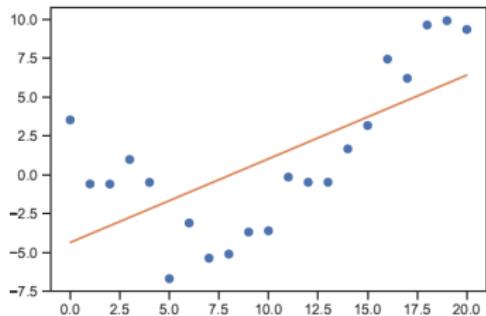
$$f(\mathbf{x}; \boldsymbol{\theta}) = b + w_1x_1 + \dots + w_Dx_D = b + \mathbf{w}^T \mathbf{x}$$

where  $\boldsymbol{\theta} = (\mathbf{w}, b)$ . This is called **multiple linear regression**.



**Figure:** We predict the temperature as a function of 2d location in a room. Linear model:  $f(\mathbf{x}; \boldsymbol{\theta}) = b + w_1x_1 + w_2x_2$

# Polynomial regression



We can improve the fit by a **polynomial regression** model of degree  $D$ :

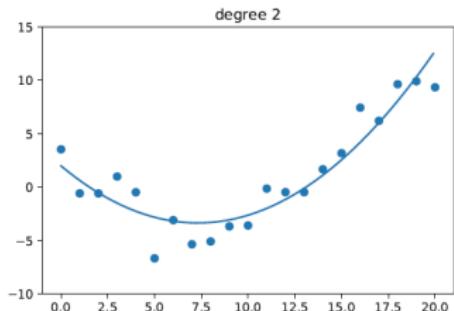
$$f(x; \mathbf{w}) = \mathbf{w}^T \phi(x)$$

where  $\phi(x)$  is a feature vector derived from the input:

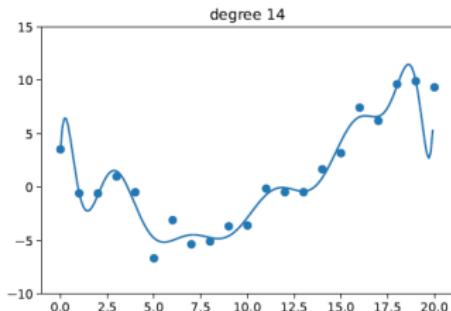
$$\phi(x) = [1, x, x^2, \dots, x^D]$$

This is an example of **feature engineering**.

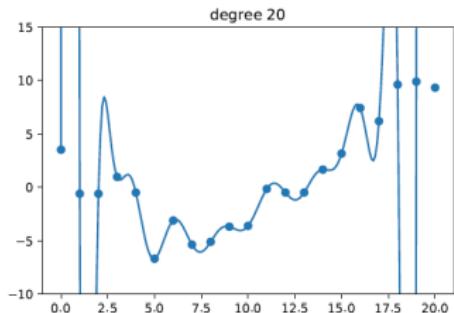
# Polynomial regression



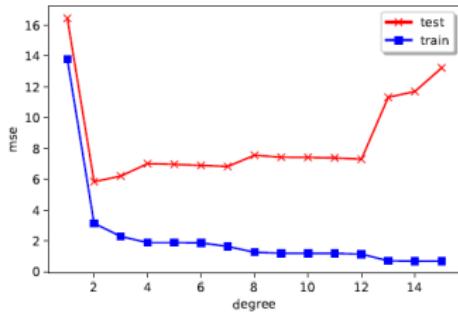
(a)



(b)

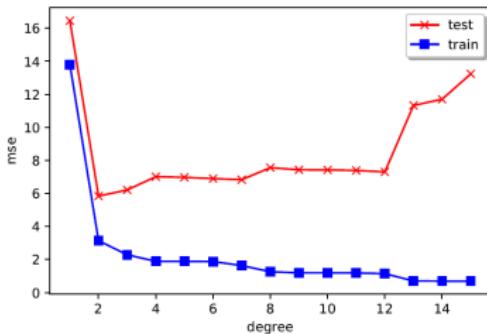


(c)



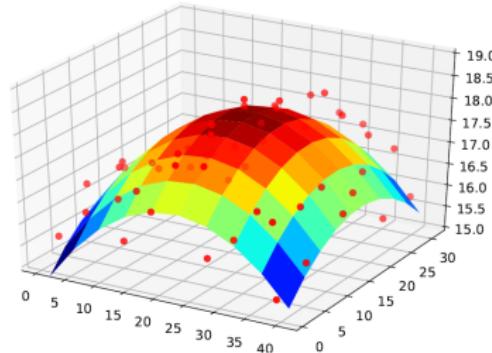
(d)

# Polynomial regression



- Using  $D = 2$  results in a much better fit.
- We can keep increasing  $D$ , and hence the number of parameters in the model, until  $D = N - 1$ .
- In this case, we have one parameter per data point, so we can perfectly **interpolate** the data. The resulting model will have 0 MSE.
- However, the resulting function will not be a good predictor.
- A model that perfectly fits the training data, but which is too complex, is said to suffer from **overfitting**.

# Polynomial regression



- We can apply polynomial regression to multi-dimensional inputs.
- For example, we use a quadratic expansion of the inputs as the prediction function for the temperature model

$$f(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2$$

- The prediction function is still a linear function of the parameters  $\mathbf{w}$ , even though it is a nonlinear function of the original input  $\mathbf{x}$ .

# Deep neural networks

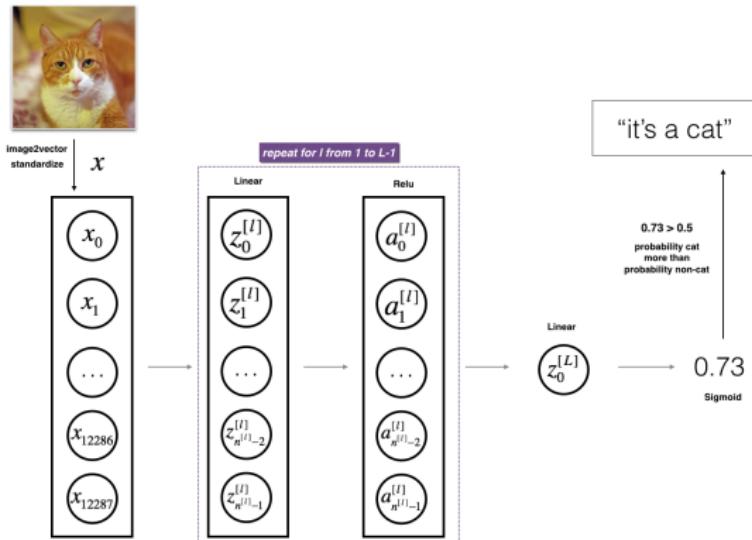
- Previously, we **manually** specify the transformation of the input features (feature engineering). For example, in polynomial regression

$$\phi(\mathbf{x}) = [1, x_1, x_2, x_1^2, x_2^2, \dots]$$

- We can create more powerful models by learning to do such *nonlinear feature extraction* **automatically**.
- We let  $\phi(\mathbf{x})$  have its own set of parameters  $\mathbf{V}$ :

$$f(\mathbf{x}; \mathbf{w}, \mathbf{V}) = \mathbf{w}^T \phi(\mathbf{x}; \mathbf{V})$$

# Deep neural networks

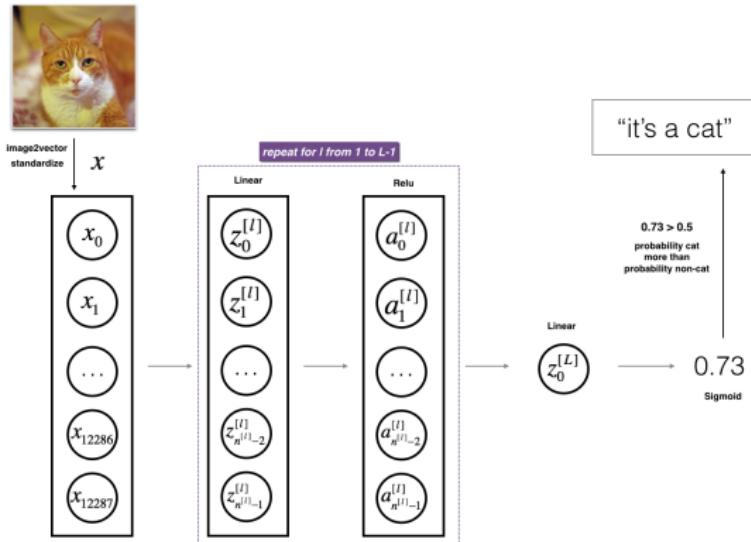


The **feature extractor**  $\phi(\mathbf{x}; \mathbf{V})$  is a composition of simpler functions.  
The resulting model is a stack of  $L$  nested functions:

$$f(\mathbf{x}; \boldsymbol{\theta}) = f(\mathbf{x}; \mathbf{w}, \mathbf{V}) = f_L(f_{L-1}(\dots(f_1(\mathbf{x}))\dots))$$

where  $f_\ell(\mathbf{x}) = f(\mathbf{x}; \boldsymbol{\theta}_\ell)$  is the function at layer  $\ell$ .

# Deep neural networks



$$f(\mathbf{x}; \mathbf{w}, \mathbf{V}) = \mathbf{w}^T \phi(\mathbf{x}; \mathbf{V})$$

Typically, the final layer  $L$  is *linear* and has the form

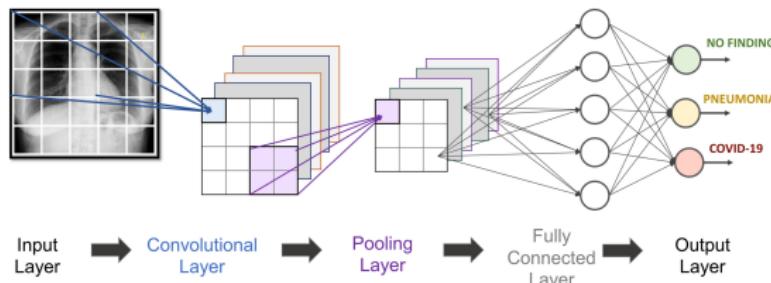
$$f_L(\mathbf{x}) = \mathbf{w}^T f_{1:L-1}(\mathbf{x})$$

where  $f_{1:L-1}(\mathbf{x})$  is the **learned** feature extractor  $\phi(\mathbf{x}; \mathbf{V})$ .

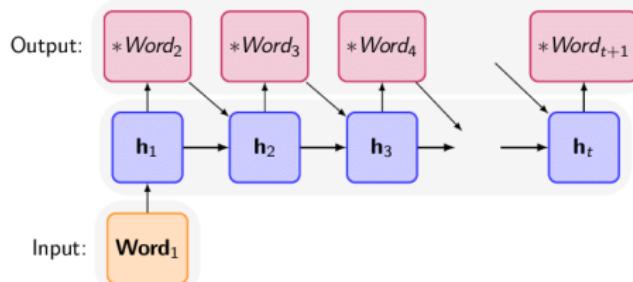
# Deep neural networks

Common DNN variants are:

- Convolutional neural networks (CNNs) for images.



- Recurrent neural networks (RNNs) for sequences.



# Overfitting and Generalization

- We can rewrite the empirical risk equation as

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N \ell(y_n, f(\mathbf{x}_n; \boldsymbol{\theta}))$$

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{\text{train}}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}_{\text{train}}} \ell(y, f(\mathbf{x}; \boldsymbol{\theta}))$$

where  $\mathcal{D}_{\text{train}}$  is the size of the training set  $\mathcal{D}_{\text{train}}$ .

- This makes explicit which dataset the loss is being evaluated on.
- With a complex enough model (or simply by memorizing all training data), we can achieve **zero** training loss.
- But we care about prediction accuracy on new data, that are not in the training set.
- A model that perfectly fits the training data, but which is too complex, is said to suffer from **overfitting**.

# Overfitting and Generalization

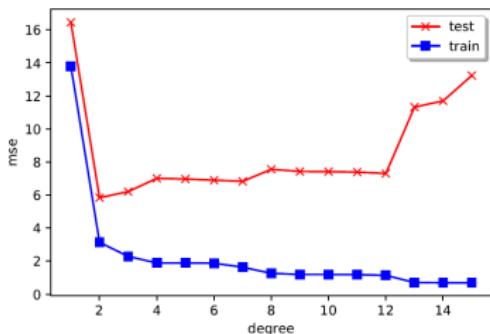
- Let  $p^*(x, y)$  denote the true (and **unknown**) distribution of the training data. Instead of the empirical risk, we have the **population risk** (or the **theoretical** expected loss):

$$\mathcal{L}(\boldsymbol{\theta}; p^*) = \mathbb{E}_{p^*(\mathbf{x}, y)} [\ell(y, f((\mathbf{x}); \boldsymbol{\theta}))]$$

- The difference  $\mathcal{L}(\boldsymbol{\theta}; p^*) - \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{\text{train}})$  is the **generalization gap**.
- If a model has a large generalization gap (i.e., low empirical risk but high population risk), it is a sign of overfitting.
- However, in practice, we don't know  $p^*$ .
- We partition the data we have into two subsets: the training set and the **test set**. We approximate the population risk using the **test risk**:

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{\text{test}}) = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}_{\text{test}}} \ell(y, f(\mathbf{x}; \boldsymbol{\theta}))$$

# Overfitting and Generalization



- The training error goes to 0 as the model becomes more complex.
- The test error has a **U-shaped curve**:
  - On the left, where  $D = 1$ , the model is **underfitting**.
  - On the right, where  $D \gg 1$ , the model is **overfitting**.
  - When  $D = 2$ , the model complexity is “just right”.
- The most complex model has the most **degrees of freedom** → minimum training loss.
- We should pick the model with the minimum test loss.

# Overfitting and Generalization

- In practice, we partition the data into 3 sets: a training set, a test set, and a **validation set**.
- The training set is used for model fitting.
- The validation set is used for model selection.
- The test set is used to estimate future performance (i.e., the population risk).
- The test set is NOT used for model fitting or model selection.

# No free lunch theorem

*All models are wrong, but some models are useful.*

— George Box

- There is no single best model that works optimally for all kinds of problems.
- A set of assumptions (**inductive bias**) that works well in one domain may work poorly in another.
- We pick a suitable model based on domain knowledge, and/or trial and error (using model selection techniques).
- It is important to have many models and algorithmic techniques to choose from.