MACHINE LEARNING

What is Human Learning?

- What does it mean to say that a human learns?
- How do we learn?

Aristotle and Aquinas on Learning

- **Aristotle** argued that learning begins with **sense experience**: we perceive particular things, remember them, and from many memories build **experience**.
- From experience, the mind can grasp more general patterns, moving from particular cases toward broader understanding.
- **Aquinas** adopts this basic picture and emphasizes that human knowledge starts with the senses and is then formed into concepts by the intellect.
- On this view, learning is a movement from experience to stable understanding, not just the storage of isolated facts.

Learning Changes the Brain

- Learning changes the brain by strengthening some connections between neurons and weakening others.
- As we practice or gain new experiences, patterns of activity that happen together become easier to repeat later.
- Over time, these changing connections make some responses faster, more accurate, and more automatic.
- At a high level, learning is the process of updating the brain’s internal wiring based on experience.

Hand-Written Programs vs Learning Programs

- In a **traditional hand-written program**, the programmer explicitly writes the rules the computer should follow.
- The behavior of the system comes mainly from human-designed logic.
- In a **learning-based program**, the programmer still chooses the overall setup, but the system improves part of its behavior from data or feedback.
- Instead of writing every rule directly, we let the system adjust itself through **experience**.

Definition

- A popular definition from Tom Mitchell: A computer program is said to **learn** from experience E with respect to some class of tasks T and some performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.
- Need to define **experience**.
- Need to define a class of **tasks**.
- Need to define a **performance measure**.

What Is an Experience?

- An **experience** is the data or feedback the program uses to improve.
- The experience is not the final goal of the system; it is the source of information from which the system adjusts its behavior.
- Different learning settings provide different kinds of experience:
  - labeled examples
  - unlabeled examples
  - rewards or penalties from interaction
- The key idea is that the system is not programmed with every rule by hand; instead, it improves by processing experience.

What Is a Class of Tasks?

- A class of **tasks** is the family of problems on which we want the system to perform well.
- It describes the general kind of input-output behavior we care about, not just one isolated case.
- Examples of classes of tasks include:
  - classifying emails as spam or not spam
  - predicting house prices from property features
  - choosing moves in game positions
- A particular email or a particular house is one instance; the class of tasks is the broader kind of problem.

What Is a Performance Measure?

- A **performance measure** tells us how well the system is doing on the tasks in the class `T`.
- It gives a concrete way to evaluate improvement.
- Examples of performance measures include:
  - accuracy for classification
  - mean squared error for prediction
  - win rate for game playing
- In Mitchell’s definition, learning requires improvement that can be observed through the performance measure.

Three Main Types of Machine Learning

- **Supervised learning** learns from labeled examples.
- **Unsupervised learning** looks for structure in unlabeled data.
- **Reinforcement learning** learns by acting and receiving rewards or penalties.
- These differ mainly in the kind of **experience** they use, the **tasks** they target, and the **performance measures** used to judge improvement.
- A newer fourth option, **self-supervised learning**, bridges **supervised** and **unsupervised** learning.

Supervised Learning

- Experience `E`: labeled examples
  - Example: emails paired with labels such as `spam` or `not spam`
  - Example: house features paired with sale prices
  - Example: images paired with object categories such as `cat`, `dog`, or `car`
- Tasks `T`: predict an output from an input
- Performance measure `P`: accuracy, error, cross-entropy loss

Supervised Learning

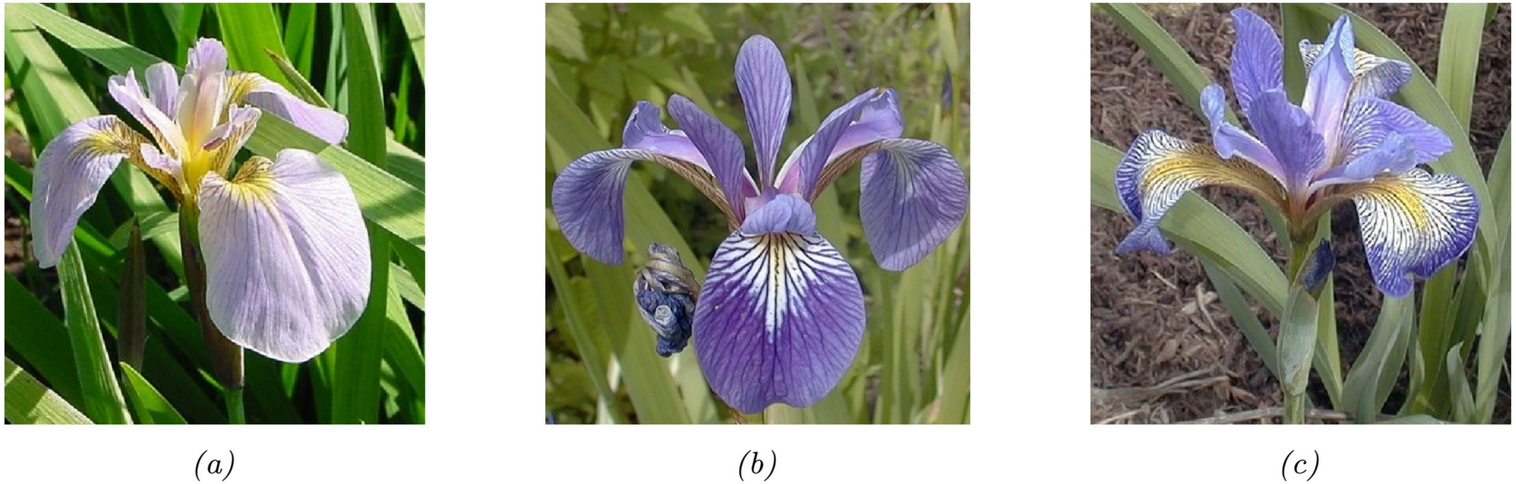
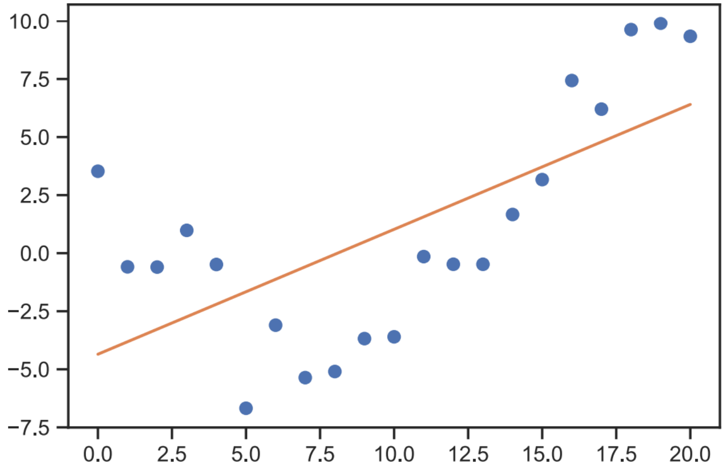
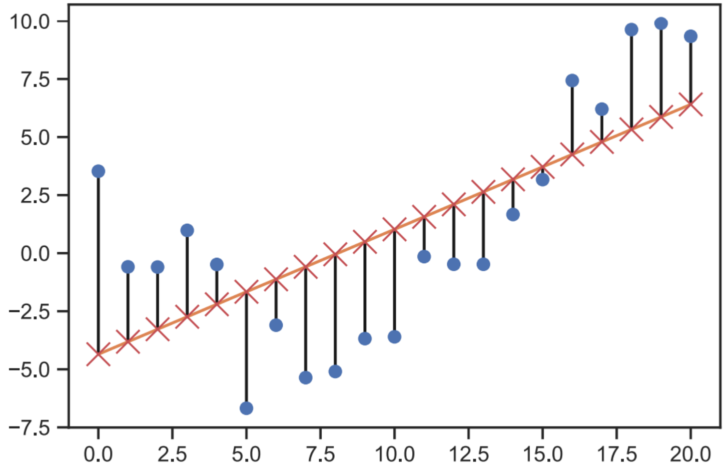


Figure 1.1: Three types of Iris flowers: Setosa, Versicolor and Virginica. Used with kind permission of Dennis Kramb and SIGNA.

Supervised Learning: Regression

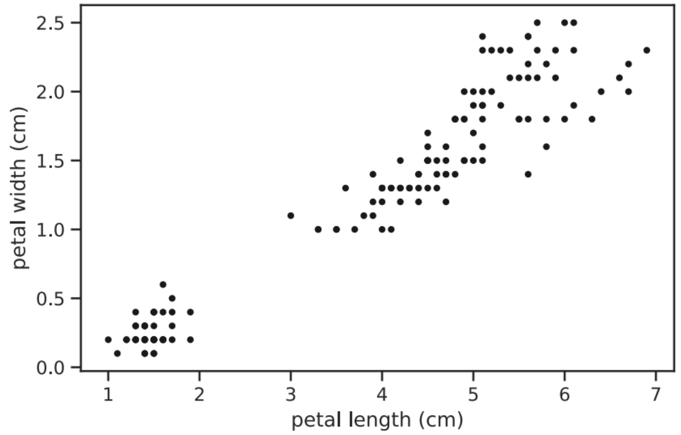




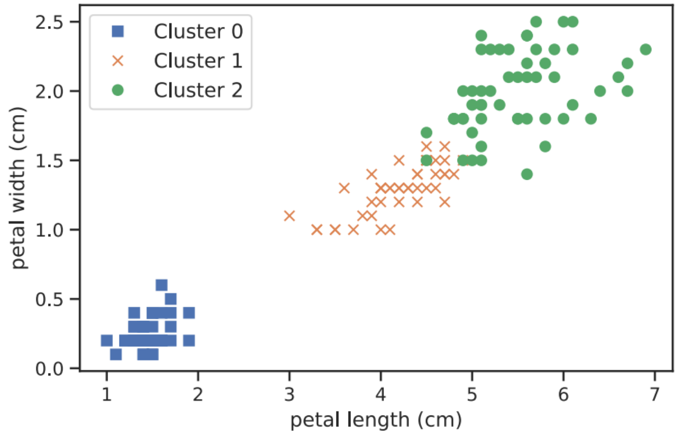
Unsupervised Learning

- Experience `E`: unlabeled data
- Tasks `T`: discover structure, groups, patterns, or lower-dimensional representations
- Performance measure `P`: often indirect, such as clustering quality, reconstruction error, or usefulness for later tasks

Unsupervised Learning



(a)

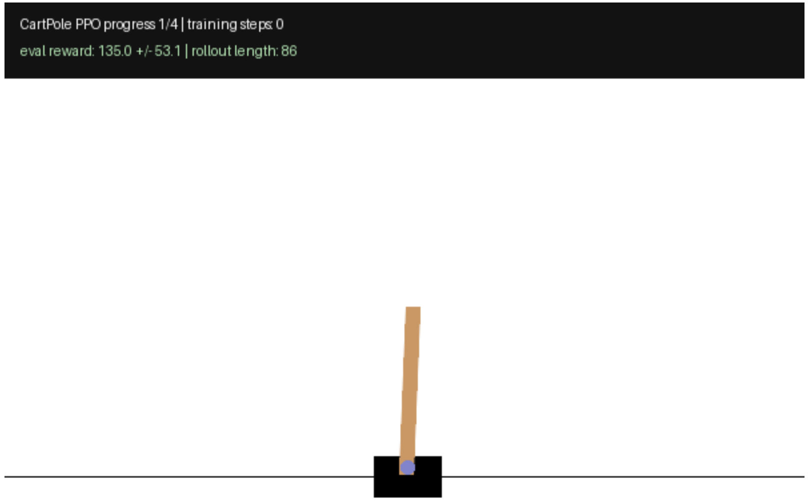


(b)

Reinforcement Learning

- Experience `E`: rewards, penalties, and state transitions from interaction
- Tasks `T`: choose actions over time to achieve good long-term outcomes
- Performance measure `P`: cumulative reward or average return

Reinforcement Learning



```
CartPole PPO progress 1/4 | training steps: 0
eval reward: 135.0 +/- 53.1 | rollout length: 86
```

Self-Supervised Learning

- **Self-supervised learning** starts with unlabeled data, so it resembles **unsupervised learning**.
- But it creates prediction targets from the data itself, so it also resembles **supervised learning**.
- It is a bridge between the two:
  - no human-provided labels are needed
  - the system still learns by solving a prediction problem
- This idea is central to many modern AI systems, including large language models.

Self-Supervised Learning Is Relatively New

- The basic idea of learning from data is old, but **self-supervised learning** became a major focus much more recently.
- It grew in importance as researchers gained access to very large unlabeled datasets and enough compute to train large models on prediction-based objectives.
- This shift helped make modern foundation models and large language models practical.

NEURAL NETWORKS

Perceptrons

- A **perceptron** is a simple example of a machine-learning model, especially for **supervised learning**.
- Experience `E`:
  - labeled training examples
  - for example, input feature vectors paired with class labels
- Tasks `T`:
  - classify inputs into one of two categories
  - for example, decide whether a point is in class `+1` or class `0`
- Performance measure `P`:
  - classification accuracy or classification error on the task
- The key idea is that the perceptron is not programmed with the separating rule by hand; it adjusts its weights from experience.

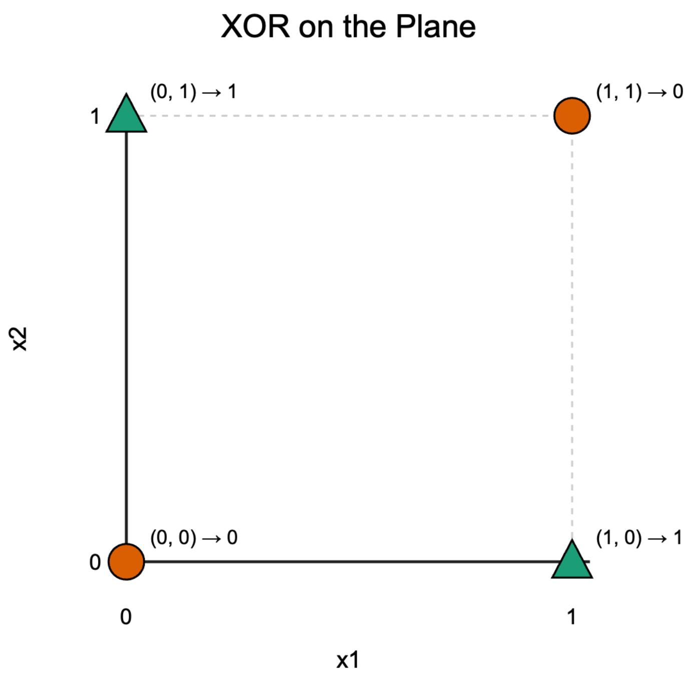
Problems with Perceptrons

- A single perceptron can only represent a **linear decision boundary**.
- It cannot solve problems like **XOR**, where the classes are not linearly separable.
- It is limited to relatively simple input-output relationships.
- This motivates combining many simple units into larger **neural networks** that can represent more complex functions.

The XOR Problem

- The **XOR** function outputs `1` when the two input bits are different, and `0` when they are the same.
- Its truth table is:
  - `(0, 0) -> 0`
  - `(0, 1) -> 1`
  - `(1, 0) -> 1`
  - `(1, 1) -> 0`
- If we plot these four cases as points in the plane, no single straight line separates the `1` outputs from the `0` outputs.
- That is why a single perceptron cannot represent XOR.

XOR



Minsky and Papert’s Critique

- In *Perceptrons*, **Marvin Minsky** and **Seymour Papert** emphasized important limitations of single-layer perceptrons, including problems like **XOR**.
- Their critique was mathematically serious, but it was often taken more broadly as evidence that neural-network approaches were a dead end.
- As a result, interest and funding for neural-network research dropped for a time.
- Later work showed that multilayer networks with nonlinear activations could get around these limits.

Two Ideas That Get Around XOR

- The XOR problem pushed neural networks in two important directions.
- First, we need **nonlinearity**, so the model is not just one linear decision rule.
- Second, we need **stacking**, so multiple units and layers can work together to build more complex features.
- The next slides explain each of these ideas.

Two Parts of a Perceptron

- A perceptron can be understood as having two main parts.
- First, it applies a **linear transformation**:
  - it computes a weighted sum of the inputs, plus a bias
- Second, it applies an **activation function**:
  - it turns that numeric score into an output decision

Smooth Nonlinear Activations

- One important change was moving beyond a hard binary threshold activation.
- Instead, neural networks often use **smooth nonlinear activation functions**.
- A smooth nonlinear activation changes the model in a way that a purely linear system cannot imitate.
- Without nonlinearity, even many stacked layers would still collapse into one linear transformation.

Stacking Layers Solved XOR

- A second key idea was to stack multiple layers of units instead of relying on a single perceptron.
- Hidden layers can learn intermediate features that make the final classification easier.
- With nonlinear activations and multiple layers, the network can represent decision boundaries that are not just single straight lines.
- This was an early example of why layered neural networks are more expressive than single linear classifiers.

Neural Networks

- A modern **neural network** is built by stacking many **linear layers** and **nonlinear activation functions**.
- The linear layers transform the representation from one stage to the next.
- The nonlinearities prevent the whole model from collapsing into one linear transformation.
- Modern AI models often use substantial depth, meaning many such layers are stacked together.
- The linear layers contain adjustable **parameters** such as weights and biases.
- Those parameters are the quantities that are **learned** from experience during training.

Deep Learning

- **Deep learning** refers to neural networks with many layers.
- The word **deep** refers to the depth of the model, meaning how many transformations are stacked on top of one another.
- A deeper network can build more complex representations by composing many simpler steps.
- In that sense, deep learning extends the same stacking idea that helped neural networks get around XOR.