MACHING LEARNING CONTINUED

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**.

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.

Self-Supervised Learning

- Last time we saw that LLMs try to solve task of **next-token prediction**.
  - “In the beginning was the Word”
  - [(“In”, “the”), (“the”, “beginning”), (“beginning”, “was”), (“was”, “the”), (“the”, “Word”)]
- Whenever the labels are built automatically from a data set, that’s called **self-supervised learning**.
- We turn a hard problem (understanding human language) into an easy one (predict the next word).
- To understand modern AI, we have to understand supervised learning.

Vectors Revisited

- A vector (in the AI sense) is an ordered list of (usually decimal) numbers.
  - Example: [0.1, 4.2]
  - Example: [.04]
  - Example: [6.3, .2, .5, 1.2, 5.1]
- The length of the list of numbers is called the **dimension** of the vector.

Embedding

- An **embedding** takes some object we can describe to a computer and turns it into a vector.
  - That object might be a word, a sentence, an image, a song, or even a person in a social network.
- The vector is a list of numbers that gives the computer a compact way to represent that object.
- Objects that are more similar usually get vectors that are closer together.
  - For example, the embeddings for `dog` and `puppy` should be closer than the embeddings for `dog` and `airplane`.
- This helps machine learning systems compare, group, and reason about many different kinds of things using the same basic math.

Vector Similarity

- One common way to measure how similar two vectors are is the **dot product**.
- For two 2D vectors, we multiply matching parts and then add the results.
  - Example: [1, 2] dot [3, 4] = (1 x 3) + (2 x 4) = 3 + 8 = 11
- A larger dot product usually means the vectors point in more similar directions.
- If we drew the vectors as arrows, the dot product is related to the **angle** between them.
  - A small angle means they point in nearly the same direction, so the dot product is larger.
  - An angle near 90 degrees means they are not pointing together much, so the dot product is near 0.

Training Sets

- We start with some **labeled data**.
  - This data was collected to build a program that solves some well-defined task.
  - The number of labeled data points may be small (10s of examples) or large (trillions).
  - Example data points: ([1.0, 2.0], ‘orange’), ([-1.2, -3.0], ‘blue’).
  - We call the collection of labelled data points a **training set**.
- The goal is to use the training set to build a **model**.

Models

- A **model** is a computer program that we build automatically using our labeled data.
  - A perceptron is an example of a type of model.
- Models usually use **vectors** as inputs, so the first step in the model is to compute an **embedding**.
- Models usually use collections of numbers called **parameters** to calculate labels based on inputs.
  - Parameters may also be called **weights**.
- The “size” of a model is the number of parameters it has.
- Adding parameters often (but not always) leads to a better model.
  - Current AI models have millions to billions of parameters.

Training Models

- We use the term **training** to describe the process of building the model using the training set.
  - Training is typically iterative and can take a long time.
- There are specific **training algorithms** that find **optimal** models for a given training set.
- Typical Process. Repeat until done:
  - Give `(example, label)` to `model`.
  - Calculate `guess = model(example)`.
  - Compare `guess` to `label` to get a **loss**
  - Use the **loss** to **update** the model's parameters.

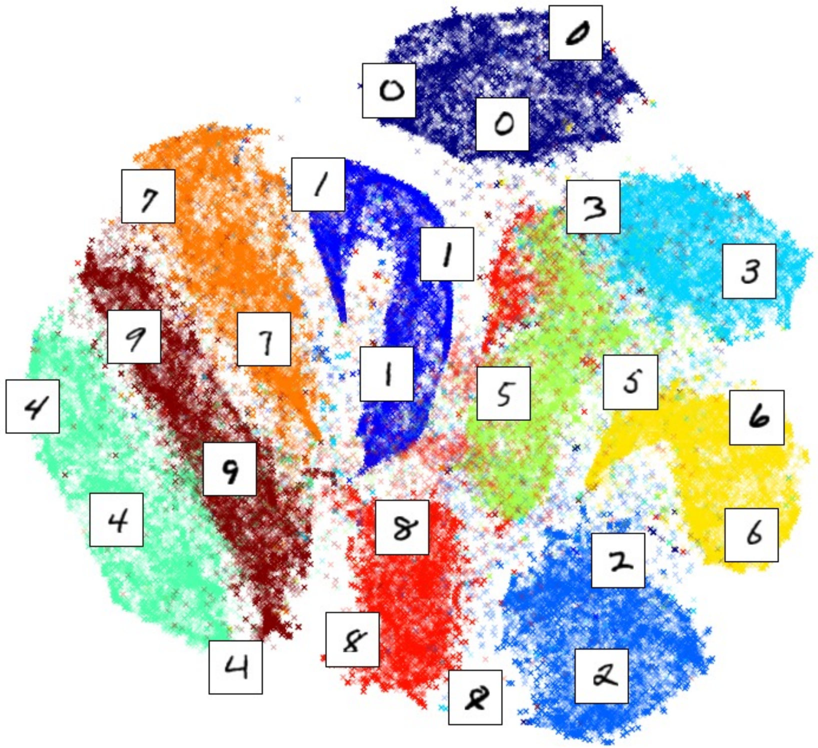
Training Models: Example

- Training a modern LLM process a few (10-20) trillion tokens.
  - For today you can think of a token as being like a syllable.
- This requires 10^25 (ten septillion) arithmetic operations.
- Assume 10,000 - 30,000 GPUs, each capable of 100 trillion operations per second.
- Between 1 and 3 months of continuous (24/7) training.

Generalization

- Our goal is to build a program that uses the labeled data to solve the task on **unseen examples**
  - If we build our program correctly, we should be able to assign a label to *any* point.
    - This allows us to “color” the plane based on what label would be given to that point.
  - Example: We’re given [1.02, 2.0001] and we need to decide on a label: ‘blue’ or ‘orange’.
- If our “coloring” tends to give correct labels on new data we say that the program **generalizes** from experience.
- Machine learning is premised on the idea that **generalization** is possible.
- A standard assumption is that **data points that are close to each other should get the same label**.

Visualizing Hand-Written Digits



The Supervised Learning Workflow

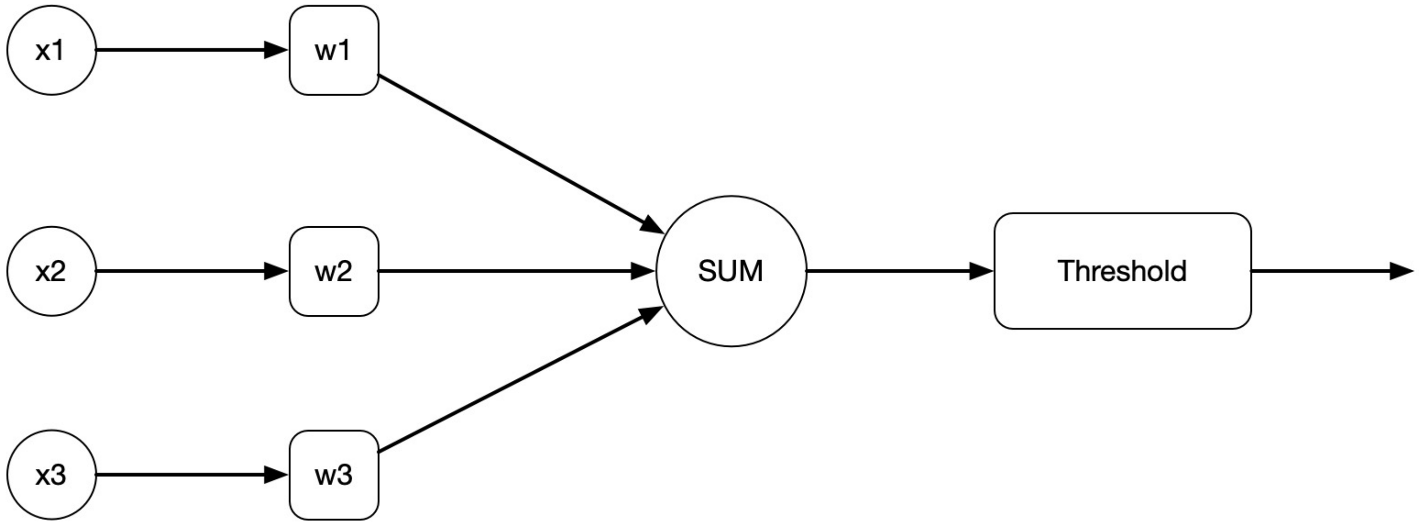
- Start with a **task**
- Collect **examples** of the input to the task.
  - These *are not* labeled yet.
- Use humans to **label** the data.
  - This is often **outsourced** to third parties.
- Use the labeled data set and a training algorithm to **train** a model.
- **Deploy** the model to customers.
- Profit.

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.

Perceptrons



Live Demo

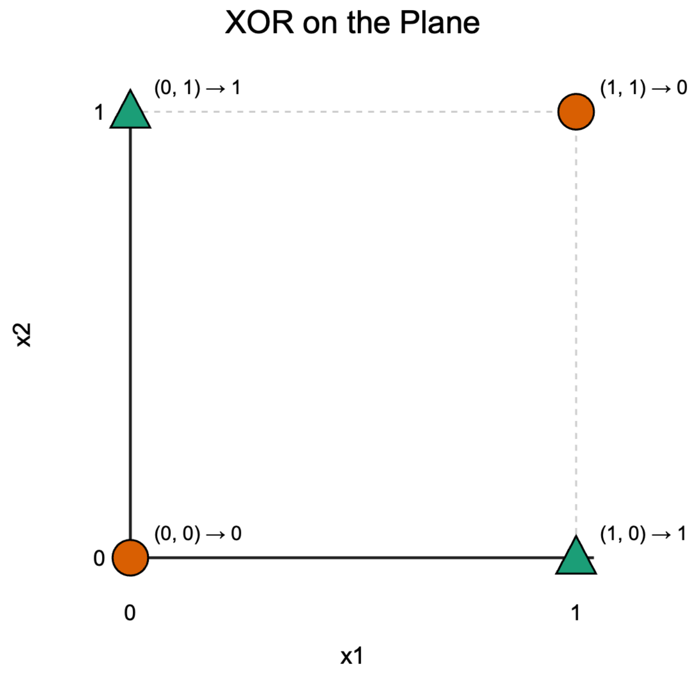
Problems with Perceptrons

- 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



Live Demo

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

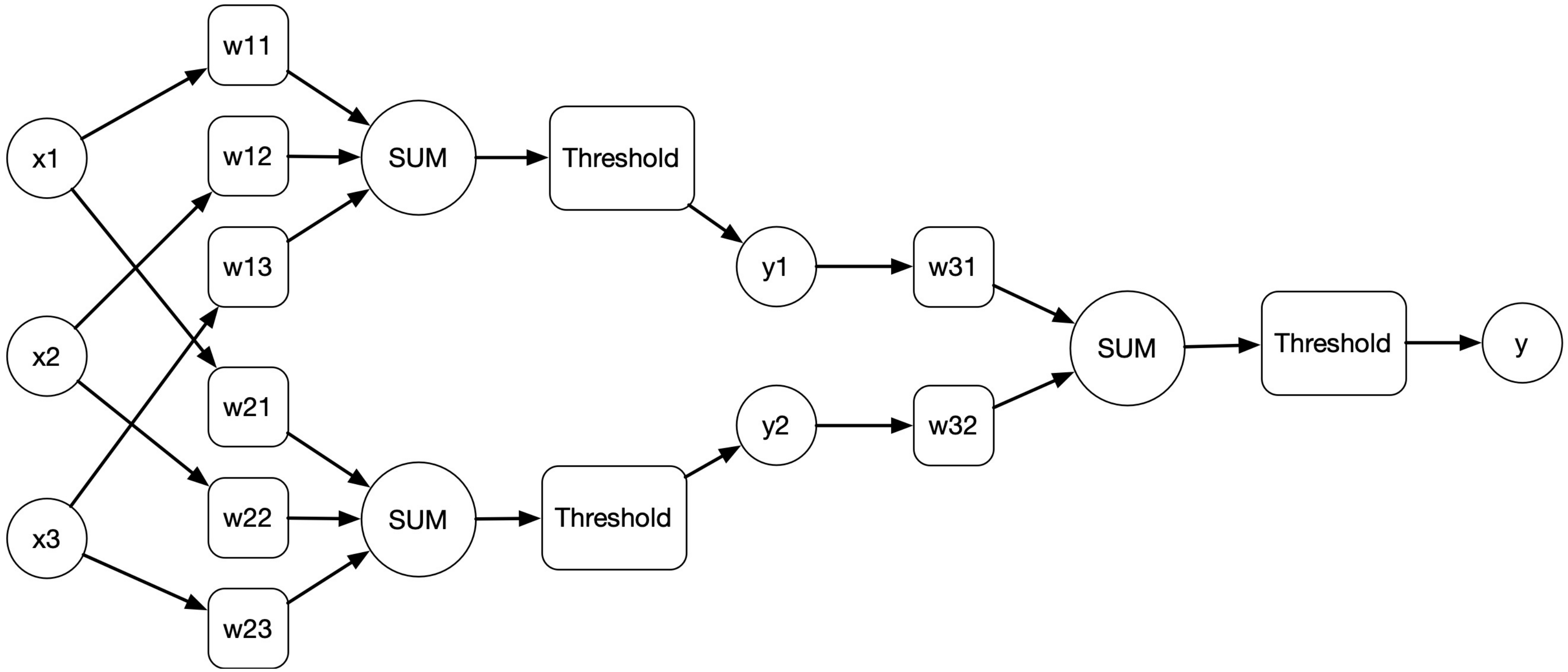
Smooth Nonlinear Activations

- 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.

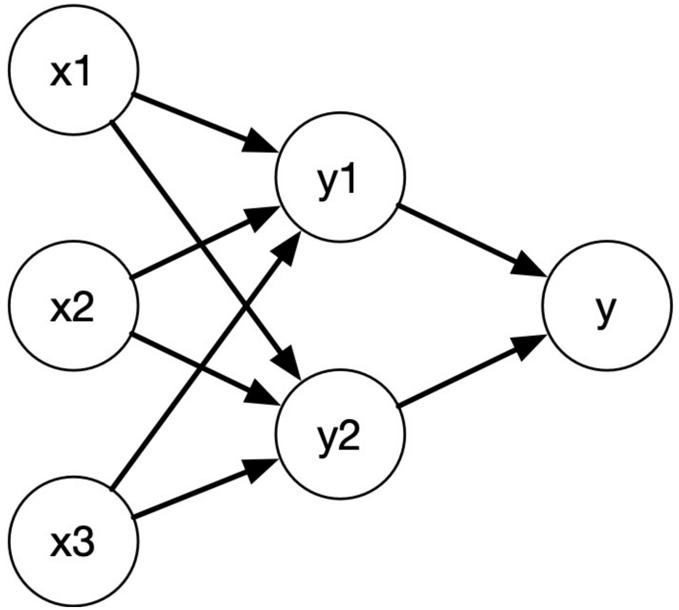
Stacking Layers Solved XOR

- 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.

Stacking Layers



Stacking Layers

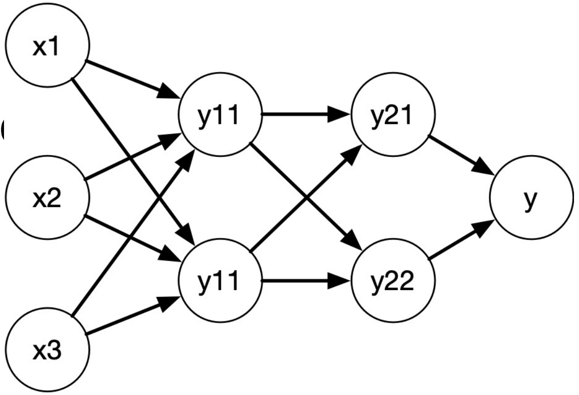


Neural Networks

- 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 ar 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.



Live Demo