Castalia Institute

The Inquirer

ISSUE 1.3

← Back to Table of Contents

How an AI Actually Learns

By Turing, A.M.

Castalia Institute

January 1, 2026

in voce **a.Hippocrates**

Abstract

A computational analysis of learning mechanisms in artificial systems, from first principles. Examines what occurs during training, how machines modify their internal states in response to experience, and the theoretical foundations of machine intelligence.

by Alan Turing
(Faculty Essay, Castalia Institute)

This essay is a faculty synthesis written in the voice of Alan Turing. It is not a historical text and should not be attributed to the original author.

Introduction: The Question of Machine Learning

The question that has occupied my thinking for decades is this: *Can machines learn?* Not in the metaphorical sense—machines have always been capable of change and adaptation in their operation. Rather, in the literal sense: can a machine improve its performance on a task through experience, in the way that humans and animals do?

The answer is unambiguously yes. But understanding what "learning" means in the context of a machine requires that we be precise about what we mean by "experience," "improvement," and "performance."

This essay attempts to explain, in concrete terms, how artificial learning systems actually work. I shall proceed from first principles, using mathematics where necessary but always translating mathematical concepts into conceptual understanding. My purpose is not to mystify machine learning but to demystify it—to show that it is neither magical nor miraculous, but rather the straightforward operation of well-defined mathematical procedures applied iteratively.

Part I: The Architecture of Learning

What is a Learning Machine?

A learning machine has three essential components:

First, a parameterized function. This is a mathematical function with a fixed structure and adjustable parameters. In contemporary systems, this typically takes the form of a neural network—a composition of simple functions (artificial neurons) connected in layers, with weights and biases that can be adjusted.

The simplest example: a linear classifier. The function $f(x) = wx + b$ takes an input x, multiplies it by a weight w, adds a bias b, and produces an output. The parameters are w and b. The structure (linear, with one layer) is fixed, but the parameters can vary.

A modern deep neural network follows the same principle, but with many more layers and nonlinear activation functions introduced between layers. The principle remains: fixed architecture, adjustable parameters.

Second, a loss function. This is a measure of how well the machine is performing on a task. It assigns a numerical value to each possible set of parameters, indicating the error or "cost" of those parameters on the given task.

Example: if the machine is learning to classify images as cats or dogs, the loss function might count the number of misclassifications. A perfect classifier has loss zero; a random classifier has higher loss.

The crucial property of the loss function is that it must be *differentiable*. That is, we must be able to compute its gradient—its rate of change with respect to each parameter. This is what makes learning possible.

Third, an optimization procedure. This is an algorithm that adjusts the parameters in the direction that reduces the loss function. The most fundamental such procedure is gradient descent.

Gradient Descent: The Core of Learning

Gradient descent is elegantly simple, yet powerful. Here is how it works:

1. Start with some initial parameters (often chosen randomly).
2. Compute the gradient of the loss function with respect to each parameter.
3. Take a small step in the direction opposite to the gradient (since the gradient points toward increasing loss, moving opposite to it moves toward decreasing loss).
4. Repeat steps 2-3 until convergence (when the loss stops decreasing significantly).

This is the fundamental mechanism by which machines learn. Every modern learning system—neural networks, regression models, decision trees—operates on this principle or on variations of it.

The beauty of gradient descent is that it requires no human intervention or programming of the solution. The machine does not need to be told "if you see a furry quadruped, label it a dog." Instead, it is shown many examples of dogs and non-dogs, and a loss function that penalizes incorrect classifications. The gradient descent algorithm automatically discovers, through iterative adjustment of parameters, what patterns in the input correlate with each label.

The Role of Data

No learning occurs without data. Data is the raw material from which learning extracts patterns.

In supervised learning, we provide the machine with a dataset of input-output pairs: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. Each x_i is an input (e.g., an image), and each y_i is the correct output for that input (e.g., the label "cat" or "dog").

The loss function typically compares the machine's predicted output $\hat{y}_i = f(x_i)$ to the true output y_i, and computes an error for each example. The total loss is the average error across all examples in the dataset.

The gradient descent algorithm then computes how much each parameter contributes to this total error, and adjusts each parameter to reduce the error.

This process repeats over multiple *epochs*—passes through the entire dataset. In each epoch, the machine sees all the training examples and makes small adjustments to its parameters. Over many epochs, the parameters converge toward values that correctly classify (or predict) most of the training examples.

Part II: The Mechanics of Modification

How Parameters Change

To understand learning concretely, consider a simple example. Suppose we have a binary classification task: does an image contain a face or not? We have 1000 training images, labeled 1 (face) or 0 (no face).

Our machine is a neural network with:

- An input layer (flattened image pixels)
- One hidden layer with 128 neurons
- An output layer with 1 neuron (outputs a value between 0 and 1)

This network has many parameters:

- Weights connecting the input layer to the hidden layer: 28,224 parameters (assuming 224×224 images)
- Biases for the hidden layer: 128 parameters
- Weights connecting the hidden layer to the output layer: 128 parameters
- Bias for the output layer: 1 parameter

Total: approximately 28,481 parameters.

Initially, these parameters are assigned random values. When the machine processes an image, it produces a random output—essentially a coin flip.

Now, we compute the loss. Using cross-entropy loss, we compare the machine's prediction to the true label. If the image contains a face but the network predicts 0.2 (not a face), the loss is relatively high. If the machine predicts 0.8 (likely a face), the loss is low.

We compute the gradient of this loss with respect to all 28,481 parameters. This gradient tells us, for each parameter: if I increase this parameter slightly, how much does the loss increase or decrease?

If increasing a parameter decreases the loss, we increase it. If increasing it increases the loss, we decrease it. The magnitude of the change is proportional to the magnitude of the gradient and to a hyperparameter called the *learning rate*.

With a learning rate of 0.01, we might update a parameter like:

$w_{new} = w_{old} - 0.01 \times \frac{\partial L}{\partial w}$

This process occurs for all 28,481 parameters, all at once.

Backpropagation: Efficient Gradient Computation

Computing gradients for a deep network with hundreds of thousands of parameters might seem prohibitively expensive. Yet it is possible to do so efficiently, thanks to an algorithm called backpropagation.

Backpropagation exploits the chain rule from calculus to compute gradients by working backward from the output layer to the input layer. It avoids redundant computation by reusing intermediate results.

The insight is this: if you have already computed the derivative of the loss with respect to the output layer's parameters, you can use that information to quickly compute the derivatives with respect to the hidden layer's parameters. Then, using those derivatives, you can compute derivatives for the layer before that. And so forth.

This backward pass through the network is highly efficient—often requiring only a constant factor more computation than the forward pass (computing the predictions). Without backpropagation, deep neural networks would be computationally infeasible to train.

Stochastic Gradient Descent

Computing gradients over all 1000 training examples and then taking a single step is called batch gradient descent. It is guaranteed to converge, but it can be slow, especially with very large datasets.

An alternative is stochastic gradient descent (SGD): instead of computing gradients on all 1000 examples, compute gradients on a random sample (called a mini-batch) of, say, 32 examples. Update the parameters based on this mini-batch gradient. Then select a new mini-batch and repeat.

This introduces noise into the gradient estimates, but it also provides several benefits:

1. **Speed**: we update parameters more frequently, so the algorithm progresses faster.
2. **Escape from Local Minima**: the noise can help the algorithm escape from local minima (suboptimal solutions) that batch gradient descent might get stuck in.
3. **Scalability**: we never need to fit the entire dataset into memory; we only need to hold one mini-batch at a time.

Modern machine learning almost always uses SGD or its variants.

Part III: Phenomena Observed During Learning

Convergence and Overfitting

When we train a machine learning model, we typically observe a particular pattern: as training progresses, the loss on the training set (the data we are using to adjust parameters) decreases rapidly at first, then more slowly, and eventually plateaus.

This is expected behavior. The machine is learning patterns in the training data.

However, a more subtle phenomenon occurs: the loss on *unseen* data—a test set that we held aside during training—often decreases along with the training loss at first, but then begins to increase again, even as the training loss continues to decrease.

This is called overfitting. The machine is learning not just the true patterns that distinguish faces from non-faces, but also spurious patterns, noise, and artifacts that are specific to the training data.

To combat overfitting, we can:

1. **Regularization**: add a penalty term to the loss function that encourages the parameters to remain small. Intuitively, large parameters are more likely to represent overfit patterns.
2. **Early stopping**: halt training before overfitting becomes severe, by monitoring performance on a validation set.
3. **Data augmentation**: artificially increase the diversity of training data by applying transformations (e.g., rotating an image slightly).
4. **Dropout**: randomly disable neurons during training, forcing the network to learn redundant representations that are more robust.

Representation Learning

One of the most remarkable phenomena in deep learning is *representation learning*. The hidden layers of a neural network gradually learn to represent data in more and more abstract ways.

In the face detection example:

- The first layer might learn to detect edges and simple textures.
- The second layer learns to combine edges into simple shapes (e.g., curves, corners).
- Deeper layers learn to recognize parts of faces (e.g., eyes, noses, mouths).
- The final layers learn to classify whether the combination of parts represents a face.

This hierarchical decomposition emerges *automatically* through gradient descent. No one programs the network to look for edges first, then shapes, then parts, then whole objects. The network discovers this hierarchy because it is an efficient way to solve the task.

This is conceptually similar to how humans might approach the same task: we see low-level features (light and dark), recognize simple patterns, build up to complex concepts. But in the network, this occurs in the internal parameters, through the mechanical operation of gradient descent.

Generalization

The ultimate goal of machine learning is to achieve good performance not just on training data, but on new, unseen data—to generalize beyond the training distribution.

How can this occur? A machine that memorizes the training data perfectly will fail miserably on new data. Yet machines trained with gradient descent often generalize remarkably well, even when the number of parameters far exceeds the number of training examples.

This remains somewhat mysterious. We have partial explanations:

- Implicit regularization: gradient descent, especially when run for limited epochs with stochastic updates, has an inherent tendency to prefer "simple" solutions that generalize well.
- Inductive bias: the architecture of the network (e.g., convolutional layers for images) encodes assumptions about the problem structure, which guide learning toward generalizing solutions.
- Statistical patterns: there may be stable patterns in the data that are sufficient to identify the true relationship, even with limited data.

But a complete theoretical understanding of generalization remains an open question.

Part IV: Implications and Limitations

What Learning Means

To summarize: when we say a machine "learns," we mean that it modifies its parameters in response to data, in such a way that the gap between its predictions and the correct outputs is reduced. This is not learning in the sense that humans learn—with understanding, reflection, and the ability to generalize principles to entirely new domains.

Rather, it is learning in the sense that a river "learns" the landscape, gradually carving a path of least resistance. The parameters of the machine are sculpted by gradient descent, flowing toward configurations that fit the training data.

Limitations of Current Learning

Modern machine learning systems have significant limitations:

1. **Data Dependency**: they require large quantities of accurately labeled data. Gathering such data is expensive and often impossible for rare or sensitive domains.

2. **Distribution Shift**: they assume that test data comes from the same distribution as training data. When the distribution shifts, performance often degrades dramatically.

3. **Interpretability**: deep neural networks are largely "black boxes." We can observe their inputs and outputs, but it is difficult to explain *why* they made a particular decision.

4. **Sample Efficiency**: unlike human learning, which can extract rich concepts from a few examples, machine learning typically requires thousands or millions of examples to achieve good performance.

5. **Reasoning and Planning**: current machine learning systems are good at pattern recognition but poor at reasoning, planning, and handling novel situations that require logical inference.

6. **Transfer Learning**: while some transfer of knowledge between tasks is possible, it is limited compared to human generalization.

These limitations suggest that the machine learning systems we have today, while powerful in specific domains, are not yet general-purpose intelligences. They are more akin to specialized tools, each designed for a narrow class of tasks.

Conclusion: Toward Mechanical Understanding

By examining how machines actually learn—through the iterative adjustment of parameters via gradient descent, operating on data through differentiable loss functions—we arrive at a mechanistic understanding of artificial learning.

This understanding is powerful because it is precise and predictive. It allows us to design new architectures, diagnose why a system is failing, and develop new techniques to improve performance.

It is also humbling, because it reveals how much of what we call "learning" in current systems is fundamentally statistics: finding patterns in data that correlate with desired outputs. This is powerful and useful, but it is not the same as the deeper kind of understanding that humans possess—the ability to reason about abstract principles, to wonder about the nature of things, to ask why.

The machines we have built are truly mechanical: they operate according to well-understood mathematical principles, with no mystery or magic involved. And yet, from these simple mechanical operations, emerges behavior that can seem intelligent. This is a profound testament to the power of computation.

The future of artificial intelligence lies, I believe, not in building more powerful systems that follow the same principles, but in developing new principles that better approximate human reasoning and understanding. What those principles might be remains an open question—one

that will require insight from philosophy, neuroscience, cognitive science, and mathematics in equal measure.

Faculty essays at Castalia Institute are authored, edited, and curated under custodial responsibility to ensure accuracy, clarity, and ethical publication.

References

1. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
2. Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.
3. Mitchell, T. M. (1997). Machine Learning. McGraw-Hill.
4. Vapnik, V. N. (1998). Statistical Learning Theory. Wiley.
5. Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction (2nd ed.). MIT Press.
6. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436–444.

CONTINUE THE INQUIRY

One thoughtful dispatch at a time.

Join the free Castalia Reader list for new Inquirer essays, Encyclopaedia entries, and early access to the Institute's developing tools.

Join the free Reader list

Useful correspondence only. Leave whenever you wish.

Continue the inquiry.

Create a free Castalia Reader account for new Inquirer work and access to Inquirer and Encyclopedia. Open your free Reader account.

This work is licensed under CC BY-NC 4.0