Lecture 3: ML Basics Continued

Lecturer: Jonathan Shafer **Date:** February 10, 2026
Scribes: David Koplow, Karthik Vedula

Notation Notes

The past two lectures have used formal math and set notation that some students may be unfamiliar with. It is explained here in addition to some important ideas.

- X denotes the **instance (input) space**. Elements $x \in X$ are data points.
- Y denotes the **label space**. For classification, typically $Y = \{0, 1\}$; for regression, $Y \subseteq \mathbb{R}$.
- The notation Y^X is **function-set notation**: $Y^X = \{ f \mid f : X \to Y \}$, the set of all functions from X to Y.
- A **hypothesis** is a function $h \colon X \to Y$. The hypothesis class is denoted $\mathcal{H} \subseteq Y^X$.
- D denotes an unknown **data distribution** over $X \times Y$. Writing $(x, y) \sim D$ means a labeled example is drawn from this distribution.
- $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ denotes a **training sample** of size m, drawn i.i.d. from D. The notation $S \sim D^m$ means m independent draws from D.
- $\ell(\hat{y}, y)$ denotes a **loss function** measuring error between prediction $\hat{y}$ and true label y.
- $L_D(h)$ denotes the **population (true) risk** of hypothesis h, defined as

$$L_D(h) = \mathbb{E}_{(x,y)\sim D}[\ell(h(x), y)].$$

- $L_S(h)$ denotes the **empirical risk** of h on sample S,

$$L_S(h) = \frac{1}{m} \sum_{i=1}^{m} \ell(h(x_i), y_i).$$

- For a finite set $A \subseteq X$, $\mathcal{H}|_A$ denotes the set of all labelings of subset A induced by hypotheses in $\mathcal{H}$.
- $\sigma(\cdot)$ denotes an **activation function**, most commonly the sigmoid $\sigma(x) = (1 + e^{-x})^{-1}$.

1. Machine-Learning Basics Continued

Definition 1.1 (Restriction of Hypothesis Class)**.** For a hypothesis class $\mathcal{H} \subseteq Y^X$, a hypothesis $h \in \mathcal{H}$, and a subset $A \subset X$, we say

- $h|_A = g$ such that $g \colon A \to Y$ and for all $a \in A$, we have $g(a) = h(a)$,
- $\mathcal{H}|_A = \{h|_A \colon h \in \mathcal{H}\}$.

Using this, we can define *shattering* as follows:

Definition 1.2 (Shattering)**.** For a hypothesis class $\mathcal{H} \subseteq \{0,1\}^X$ and finite subset $A \subseteq X$, we say that $\mathcal{H}$ **shatters** A is $|\mathcal{H}|_A| = 2^{|A|}$. In English, this means no matter how you label the subset of points A, if the hypothesis class (the models that you are considering) can fit the data exactly, then A shatters $\mathcal{H}$. Another way to say this is that no matter how you label the points in the subset, the class contains a hypothesis that explains all of them simultaneously.

We can now complete the definition of *VC dimension*:

Definition 1.3 (Vapnik–Chervonenkis (VC) Dimension)**.** For a hypothesis class $\mathcal{H} \subseteq \{0,1\}^X$, we have

$$\mathrm{VC}(\mathcal{H}) = \sup\{k \in \mathbb{N} : \exists A \subseteq X \text{ s.t. } |A| = k \text{ and } \mathcal{H} \text{ shatters } A\}.$$

In English, the VC dimension of a hypothesis class is the largest size of a set that the class can shatter. Equivalently, as we keep adding points to a set A, there is a smallest size $k+1$ for which some labeling of the points cannot be realized by any hypothesis in $\mathcal{H}$. The VC dimension is then k. If $\mathcal{H}$ shatters arbitrarily large-sized sets, then its VC dimension is infinite.

VC Dimension Examples

Single hypothesis ($|\mathcal{H}| = 1$). Since $\mathcal{H}$ contains only one function, it induces only one labeling on any subset $A \subseteq X$. Therefore, no nonempty set can be shattered, regardless of the dimension or structure of X. Hence $\mathrm{VC}(\mathcal{H}) = 0$.

Two hypotheses ($|\mathcal{H}| = 2$).
Let $\mathcal{H} = \{h_1, h_2\}$. Fix any finite subset $A \subseteq X$. Each hypothesis h_i assigns a label in $\{0,1\}$ to every point in A, so each h_i produces exactly one labeling of A. Therefore,

$$|\mathcal{H}|_A| \leq 2,$$

because there are only two hypotheses available (so at most two induced labelings).
To *shatter* a set A of size k, we would need

$$|\mathcal{H}|_A| = 2^k,$$

since there are 2^k possible binary labelings of k points.
If $k \geq 2$, then $2^k \geq 4$, but we just saw $|\mathcal{H}|_A| \leq 2$. Hence no set of size 2 (or larger) can be shattered.
On the other hand, if $k = 1$, then by definition of distinct hypotheses, there must exist some $x \in X$ such that $h_1(x) \neq h_2(x)$. Picking $A = \{x\}$, since we have $h_1(x) \neq h_2(x)$, then $\mathcal{H}|_{\{x\}} = \{0,1\}$, so $\{x\}$ is shattered. Thus $\mathrm{VC}(\mathcal{H}) = 1$.

Threshold functions on the real line. Consider

$$\mathcal{H} = \{h_t(x) = \mathbb{1}(x \geq t) : t \in \mathbb{R}\}.$$

Each h_t labels all points to the right of t as 1 and all points to the left of t as 0.
Step 1: One point can be shattered. Fix any single point $A = \{x\}$. There are two labelings:

- Label x as 1: choose $t \le x$, then $h_t(x) = 1$.
- Label x as 0: choose $t > x$, then $h_t(x) = 0$.

So $\mathcal{H}$ shatters every set of size 1.

Step 2: No two-point set can be shattered. Take two distinct points. There are four possible labelings of $\{x_1, x_2\}$:

$$(0,0),\ (0,1),\ (1,0)\ (1,1).$$

Assume without loss of generality $x_1 < x_2$. Thresholds can realize three of these:

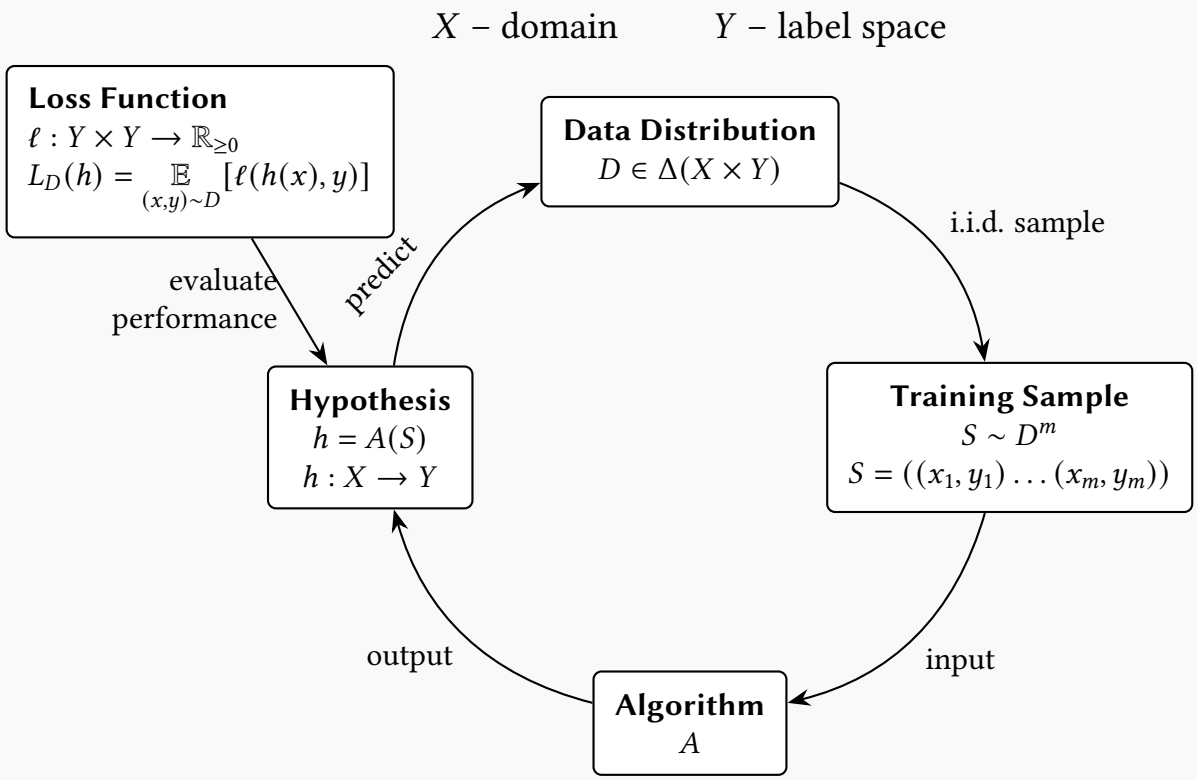
- $(0,0)$ by choosing $t > x_2$,
- $(0,1)$ by choosing $x_1 < t \le x_2$,
- $(1,1)$ by choosing $t \le x_1$.

But the labeling $(1,0)$ is impossible: if $h_t(x_1) = 1$, then $x_1 \ge t$. Since $x_2 > x_1$, we also have $x_2 \ge t$, which forces $h_t(x_2) = 1$. Thus a threshold can never label a smaller point as 1 while labeling a larger point as 0. Therefore, no set of size 2 is shattered, but some set of size 1 is shattered, so

$$\mathrm{VC}(\mathcal{H}) = 1.$$

Review: PAC Learning

Recall the process of learning a function based on inputs and outputs sampled from a distributions as outlined last lecture:



In learning theory, we care about several key **resources**:

- **Data:** The number of samples required (sample complexity).

- **Compute:** Training time and inference time.
- **Memory:** Space required to store the model and process data.
- **Randomness:** Random bits used by the algorithm.

Ideally, we want perfect classification ($L_D(h) = 0$). However, due to constraints, we need to relax this goal.

Limitation	Relaxation	Term
Finite sample size	$L_D(h) \leq \varepsilon$	Approximately correct
Sampling randomness	$\Pr[\text{success}] \geq 1 - \delta$	Probably correct
Finite Memory	ignore (for simplicity of definition)	
Finite Compute	ignore (for simplicity of definition)	

Definition 1.4 (PAC Learning)**.** For sets X and Y, mapping $\ell \colon Y \times Y \to \mathbb{R}_{\geq 0}$, and hypothesis class $\mathcal{H} \subseteq Y^X$, we say that an algorithm $A \colon (X \times Y)^* \to Y^X$ **probably approximately-correct (PAC)** learns $\mathcal{H}$ if for all ε and δ, there exists $m = m(\varepsilon, \delta) \in \mathbb{N}$ such that for all $\mathcal{H}$-realizable data distribution D, we have

$$\Pr_{S \sim D^m}\left[L_D(A(S)) \leq \varepsilon\right] \geq 1 - \delta.$$

Definition 1.5 (Agnostic PAC Learning)**.** When perfect classification is impossible (i.e., $L_D(h) > 0$ for all $h \in \mathcal{H}$), we compete with the best hypothesis in $\mathcal{H}$. An algorithm A **agnostic PAC learns** $\mathcal{H}$ if for all $\varepsilon, \delta > 0$, there exists $m = m(\varepsilon, \delta)$ such that for all distributions D:

$$\Pr_{S \sim D^m}\left[L_D(A(S)) \leq \inf_{h \in \mathcal{H}} L_D(h) + \varepsilon\right] \geq 1 - \delta.$$

Note that the sample complexity does not depend on the distribution we are trying to learn, and this is called *distribution-free learning*.

Review: Sample Complexity Bounds

Sample Complexity Bounds and ERM

The sample complexity required for PAC learning is tightly characterized by the VC dimension.

Theorem 1.6 (Sample Complexity)**.** *The sample complexity for PAC learning is defined by the following:*

1. ***Realizable Case:*** $m(\varepsilon, \delta) = \Theta\left(\frac{VC(\mathcal{H}) + \log(1/\delta)}{\varepsilon}\right)$.
2. ***Agnostic Case:*** $m(\varepsilon, \delta) = \Theta\left(\frac{VC(\mathcal{H}) + \log(1/\delta)}{\varepsilon^2}\right)$.

There is a very simple algorithm that achieves the sample complexities defined above called Empirical Risk Minimization (ERM).

Definition 1.7 (ERM)**.** The **Empirical Risk Minimization (ERM)** rule selects a hypothesis

that minimizes the empirical risk on the training sample S:

$$\mathrm{ERM}_{\mathcal{H}}(S) \in \arg\min_{h\in\mathcal{H}} L_S(h), \quad \text{where } L_S(h) = \frac{1}{m}\sum_{i=1}^{m} \ell(h(x_i), y_i).$$

If there are multiple minimizers, we break ties arbitrarily. While ERM is often statistically consistent, it faces computational challenges:

- **NP-Hardness:** For certain hypothesis classes and losses (e.g., minimizing 0-1 loss for axis-aligned rectangles in high dimensions), ERM is NP-hard.
- **Undecidability:** For pathological classes, finding the empirical risk minimizer can be formally undecidable.
- **Cryptographic Hardness:** Problems like **Learning Parity with Noise (LPN)** are conjectured to be hard. LPN can be explicitly formulated as a search problem over $\mathbb{F}_2^n$: given sample pairs $(x, \langle s, x\rangle + e \pmod 2)$ where $s \in \mathbb{F}_2^n$ is a secret, $x \sim \mathbb{F}_2^n$ is uniformly random, and e is a sparse noise vector, it is computationally hard to find s.
- **Learning vs. Computation Tensions:** While statistical theory (PAC bounds) tells us how much data is needed, computational complexity dictates whether we can efficiently find a good hypothesis.

2. Probabilistic Classification

Review: Regression and Classification

Regression

In *regression*, the label space is continuous, typically $Y \subseteq \mathbb{R}$. A regressor is a hypothesis $h\colon X \to Y$.

Squared loss. The standard loss is

$$\ell_{\mathrm{sq}}(\hat{y}, y) = (\hat{y} - y)^2.$$

Accordingly, the population risk and empirical risk are

$$L_D(h) = \mathbb{E}_{(x,y)\sim D}\big[\ell_{\mathrm{sq}}(h(x), y)\big], \qquad L_S(h) = \frac{1}{m}\sum_{i=1}^{m} \ell_{\mathrm{sq}}(h(x_i), y_i).$$

Definition 2.1 (Ordinary Least Squares (OLS))**.** For a parametric class (e.g. linear predictors), OLS chooses parameters that minimize the empirical squared loss:

$$\hat{h} \in \arg\min_{h\in\mathcal{H}} \frac{1}{m}\sum_{i=1}^{m} (h(x_i) - y_i)^2.$$

Classification

In *classification*, the label space is discrete, e.g. binary $Y = \{0, 1\}$ or multiclass $Y = \{1, \dots, k\}$. A classifier is a hypothesis $h \colon X \to Y$.

0–1 loss. The standard loss is the indicator of mistake:

$$\ell_{0,1}(\hat{y}, y) = \mathbb{1}(\hat{y} \neq y).$$

Thus,

$$L_D^{0,1}(h) = \mathbb{E}_{(x,y)\sim D}\big[\ell_{0,1}(h(x), y)\big] = \Pr_{(x,y)\sim D}[h(x) \neq y],$$

and

$$L_S^{0,1}(h) = \frac{1}{m}\sum_{i=1}^{m} \ell_{0,1}(h(x_i), y_i) = \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}\big(h(x_i) \neq y_i\big).$$

Scores and decision rules. Often we first compute real-valued scores and then convert them to a label. In the binary case, a common form is

$$h(x) = \mathbb{1}\big(f(x) \geq 0\big), \qquad f \colon X \to \mathbb{R}.$$

In the multiclass case, we use scores $\{f_c\}_{c=1}^{k}$ with

$$h(x) = \arg\max_{c \in \{1,\dots,k\}} f_c(x).$$

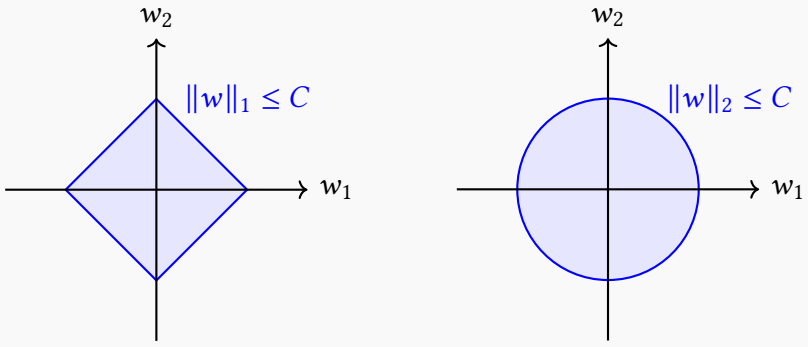
Regularization

To reduce overfitting, we often penalize model complexity. For linear predictors parameterized by w:

$$\min_{w} \; L_S(w) + \lambda \|w\|_p.$$

where $\lambda > 0$ controls the strength of regularization. Common choices for p:

- **Ridge** ($p = 2$): $\|w\|_2$ penalty.
- **Lasso** ($p = 1$): $\|w\|_1$ penalty; encourages sparsity.



Lasso (L_1) **Ridge (L_2)**

Now, instead of outputting a single label, as we have done so far, we output a distribution on labels. The label space Y is still discrete (e.g. binary output flags), but the hypothesis class is no longer discrete i.e. $\mathcal{H} \subseteq [0,1]^X$. Conceptually, this expresses **uncertainty** due to the continuous nature and is good for **optimization**, as we will see soon in *logistic regression*:

Definition 2.2 (Cross-Entropy)**.** The **cross-entropy loss/log loss/surprisal** for an outcome y and prediction $\hat{p}$ is given by

$$\ell(y,\hat{p}) = y \ln \frac{1}{\hat{p}} + (1-y)\ln \frac{1}{1-\hat{p}} = \begin{cases} \ln\left(\frac{1}{\hat{p}}\right), & \text{if } y = 1 \\ \ln\left(\frac{1}{1-\hat{p}}\right), & \text{if } y = 0 \end{cases}.$$

Similarly, we can define this for a hypothesis $h \in \mathcal{H}$ and data source D:

$$L_D(h) := \mathbb{E}_{(x,y)\sim D}\left[\ell(y,h(x))\right] = \mathbb{E}\left[\ln\left(\frac{1}{\hat{p}_y}\right)\right].$$

We are now ready to define *logistic regression*:

Definition 2.3 (Logistic Regression)**.** Let $\sigma\colon \mathbb{R} \to [0,1]$ denote the **sigmoid function**, defined by $\sigma(x) = \frac{1}{1+e^{-x}}$. For $x \in X = \mathbb{R}^n$ and $Y = \{0,1\}$, we define the **logistic regression hypothesis class** $\mathcal{H}$ as follows:

$$\forall w \in \mathbb{R}^n, b \in \mathbb{R}, \qquad h_{w,b}(x) := \sigma(\langle w, x\rangle + b),$$
$$\mathcal{H} := \left\{h_{w,b}\colon w \in \mathbb{R}^n, b \in \mathbb{R}\right\}.$$

Note that there is no closed-form solution for the empirical-risk minimizer (ERM), so stochastic methods are typically used, such as stochastic gradient descent (SGD).

3. Neural Networks (NN)

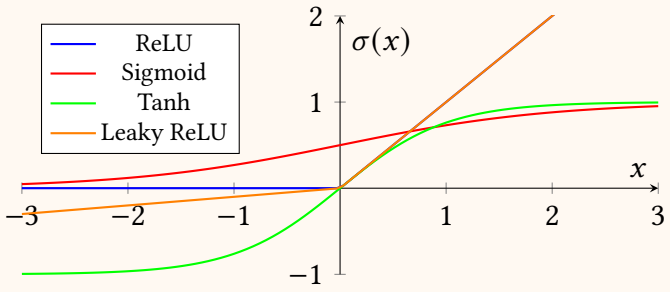
Logistic regression is a simple (one-neuron) example of a neural network.

Definition 3.1 (Neural Network Components)**.** A **neuron** is a composition of a linear function with a non-linear function. This non-linear function is known as an **activation function**.

Activation Function Examples

- Rectified Linear Unit (ReLU): $R(x) := \max\{0, x\}$,
- Leaky ReLU: $L_\alpha(x) := \max\{\alpha x, x\}$ with $0 < \alpha \ll 1$,
- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$,
- Hyperbolic tangent: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.
- Softmax: multivariate sigmoid, mapping

$$\sigma(x_1, x_2, \ldots, x_k) := \left(\frac{e^{x_1}}{\sum_{j=1}^k e^{x_j}}, \ldots, \frac{e^{x_n}}{\sum_{j=1}^k e^{x_j}}\right).$$



Finally, a **neural network** is a composition of many neurons.

Architecture Examples

We now discuss some of the most common architectures of neural networks:

Fully-connected Multi-level Perceptron (MLP). With input (layer 0) $x_1 = a_1^0, \ldots, x_{s_0} = a_{s_0}^0$, the next layer (layer 1) is composed of $a_1^1, a_2^1, \ldots, a_{s_1}^1$, where for each $i \in [s_1]$, we have

$$a_i^1 = \sigma\left(z_i^1\right) = \sigma\left(w_{1i}^0 a_1^0 + \cdots + w_{s_1 i}^0 a_{s_1}^0 + b_i^0\right),$$

where the w_{ij}^k's are the **learned weights** and the b_j^i's are the **biases**. We repeat this as many times as necessary until the penultimate layer $a_1^L, \ldots, a_{s_L}^L$, where we apply a final softmax later to output a probability distribution over the possible outputs Y. We can write this as a **feed-forward** algorithm as follows:

Algorithm 1: Feed Forward Process

Input : x (input vector), $\{W^{(i)}\}$ (weights), L (total layers)
Output: Final activation $a^{(L)}$

$a^{(0)} \leftarrow x$;
for $i \leftarrow 1$ **to** L **do**
  $z^{(i)} \leftarrow W^{(i)} a^{(i-1)}$;
  $a^{(i)} \leftarrow \sigma^{(i)}(z^{(i)})$
end
return $a^{(L)}$;

Recurrent Neural Network (RNN). Run neural networks multiple times and use the results from previous runtimes to influence future activations.

Convolutional Neural Network (CNN). Used for image identification, details skipped in the interest of time.

3.1. Training Neural Networks

The most common algorithm used to train or learn the desired weights/biases of the neural network is *stochastic gradient descent (SGD)*:

Definition 3.2 (Stochastic Gradient Descent (SGD))**.** Let $W \in \mathbb{R}^n$, $\eta > 0$ be the *step size*, $T \in \mathbb{N}$ be the number of steps, and b be the mini-batch size. The process of **stochastic gradient descent (SGD)** is as follows:

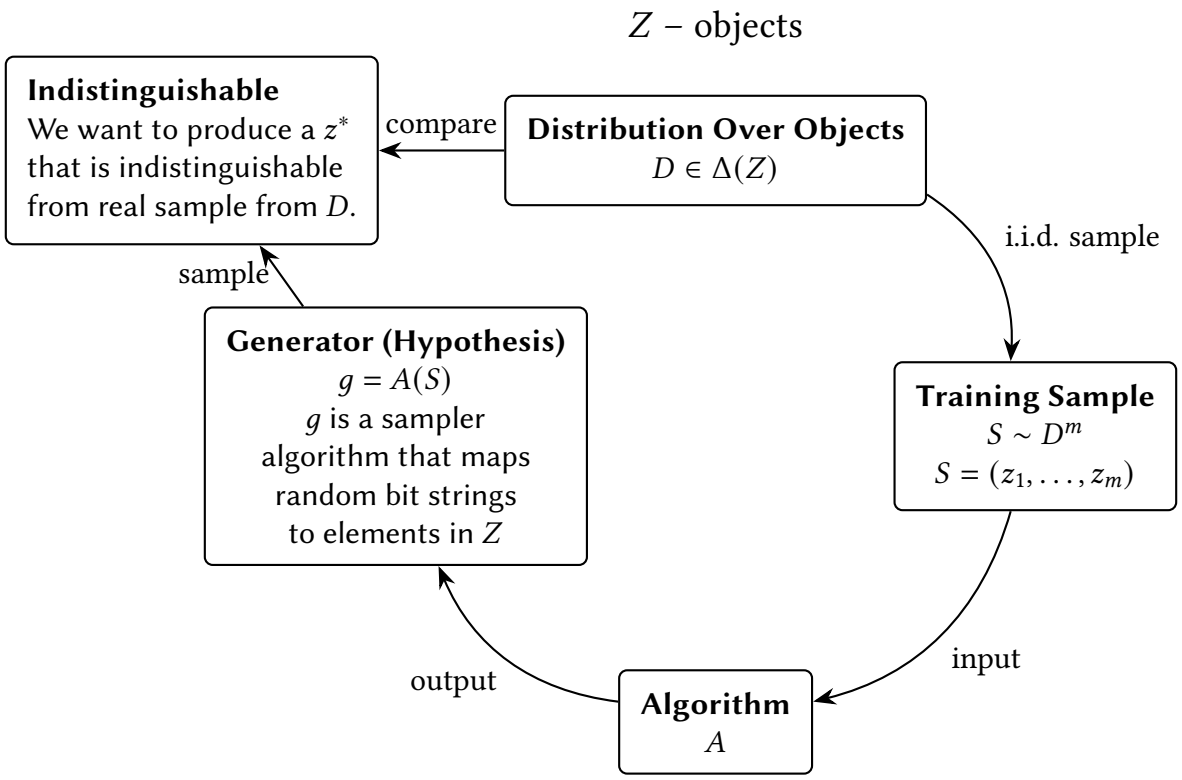
Algorithm 2: Stochastic Gradient Descent (SGD)

Input : $W \in \mathbb{R}^n$, $\eta > 0$ (step size), $T \in \mathbb{N}$ (steps), b (batch size)
Result : Optimized weights $W^{(T)}$

Initialize $W^{(0)}$ randomly;
for $t \leftarrow 1$ **to** T **do**
  Sample $z = (z_1^{(t)}, z_2^{(t)}, \ldots, z_b^{(t)}) \sim D^b$;
  Select $v^{(t)} \in \partial_W(\ell(W^{(t-1)}, z))$;
  `// ∂_W denotes subgradient w.r.t. W`
  `// When non-differentiable, pick any supporting vector`
  $W^{(t)} \leftarrow W^{(t-1)} - \eta \cdot v^{(t)}$;
end
return $W^{(T)}$;

4. Data Generation

The process of data generation can be described in a similar architecture. The data source outputs data without labels, we get a sample S, we run it through the algorithm, and then we output a hypothesis (which we call a *generator*) and we generate a new input/image. This process is successful if the things generated "look" the same as what came from the data source or be indistinguishable. There are some attempts to define this, but there is no universally correct answer to it, and it is very application-dependent. The generation process is visualized below.



One example of a data generation process is the following: https://thispersondoesnotexist.com/. The core goal here is **indistinguishability** between the generated and real samples.