It’s Not What Machines Can Learn, It’s What We Cannot Teach

Supplemental Material

Gal Yehuda 1  Moshe Gabel 2  Assaf Schuster 1

Proof of Lemma 5.

**Lemma 4.** There exists an NP-hard language \( L_1 \) and a function \( \delta(n) \to 0 \) as \( n \to \infty \), such that for any sufficiently long \( w \) generated by any randomized polynomial process,

\[
\Pr[w \in L_1] \leq \delta(n).
\]

The proof is similar to the proof of Theorem 1 in (Itsykson et al., 2016). The main difference is that we construct a decidable language, in contrast to the language generated in (Itsykson et al., 2016).

**Proof.** For every \( n \), the output of a randomized algorithm \( P \) is a random variable \( P_n \): for \( w \in \{0,1\}^n \), \( \Pr[P_n = w] \) is the probability that given the length \( n \), \( P \) outputs \( w \). Let \( K \subseteq \{0,1\}^n \) be a set of words of length \( n \); \( \Pr[P_n \in K] \) is the probability that a random word \( w \) drawn by \( P_n \) is in \( K \).

Given two random variables \( X, Y \) such that \( X, Y \) take values in \( \{0,1\}^n \), the statistical distance between \( X \) and \( Y \) is defined as (Itsykson et al., 2016):

\[
\Delta(X, Y) = \max_{K \subseteq \{0,1\}^n} |\Pr[X \in K] - \Pr[Y \in K]|.
\]

Using Theorem 9 in (Itsykson et al., 2016) when \( a = \frac{1}{2} \) and \( b = 1 \) we obtain the following corollary.

**Corollary 5.** For every randomized algorithm \( P \) that runs in time \( O(n^{\log^{0.5} n}) \) there exist infinitely many words that \( P \) can only generate with probability less than \( \epsilon(n) \), where \( \epsilon(n) \to 0 \) as \( n \to \infty \).

We construct the randomized algorithm \( P \) as follows. Let \( \mathcal{M} \) be an enumeration of all probabilistic Turing machines \( \mathcal{M} = M_1, M_2, M_3, ... \), under a standard enumeration of Turing machines, and let \( g(n) \) be a function that satisfies \( g(n)\epsilon(n) \to 0 \) and \( g(n) \to \infty \) (where \( \epsilon(n) \) is the function from Corollary 5). Example of such function is \( g(n) = \frac{1}{\log(\epsilon(n))} \). We define \( \delta(n) = g(n)\epsilon(n) \), by the definition of \( g(n), \delta(n) \to 0 \).

On input \( n \), the algorithm \( P \) uniformly chooses \( M_i \) for \( 1 \leq i \leq g(n) \) and runs \( M_i \) on the input \( n \) (with the random bits \( M_i \) needs) for \( O(n^{\log^{0.5} n}) \) steps. If \( M_i \) returned a word \( w < n \), \( P \) pads it with \( n - |w| \) zeros and returns the result. If \( M_i \) returned a word \( w > n \), \( P \) trims \( |w| - n \) characters from \( w \) and returns it. Finally, if \( M_i \) did not halt, \( P \) returns \( w = 1^n \).

\( P \) satisfies the following properties:

1. For every randomized polynomial algorithm \( P' \) and for every \( w \in \{0,1\}^n \) when \( n \) is large enough,

\[
\Pr[P'_n = w] \geq \frac{1}{g(n)} \Pr[P'_n = w].
\]

2. \( P \) runs in time \( O(n^{\log^{0.5} n}) \).

We show that the first property holds as follows. Let \( P' \) be a randomized polynomial algorithm that runs in time \( O(n^c) \), and let \( n_0 \) be the first index that \( P' \) appears in the enumeration \( \mathcal{M} \). For \( w, |w| = n \geq g(n_0) \) and \( n^{\log^{0.5} n} \geq n^c \), the probability of \( P \) to generate \( w \) is at least the probability to choose the machine \( P' \), \( \frac{1}{g(n)} \), multiplied by the probability that the machine \( P' \) generates \( w: \Pr[P'_n = w] \). Note we give \( P' \) enough time to complete the computation by choosing \( n \) such that \( n^{\log^{0.5} n} \geq n^c \).

The second property holds by the definition of \( P \).

By Corollary 5 there exists a randomized algorithm \( P^* \) such that for infinitely many \( n's \) \( n_1, n_2, n_3, ... \), it holds that

\[
\Delta(P^*_n, P_n) \geq 1 - \epsilon(n).
\]

It means that for each such \( n \), there exists a set of strings \( K_n \) such that \( \Pr[P_n \in K_n] \leq \epsilon(n) \).

Define \( L_1 \) as the union of all \( K_n \).

Let \( w \in L_1 \) of length \( n \) for sufficiently large \( n \), and let \( P' \) be a randomized polynomial algorithm.

\[
\Pr[w = P'_n] \leq g(n) \Pr[w = P_n] \tag{1}
\]

\[
\leq g(n)\epsilon(n) \tag{2}
\]

\[
\delta(n) \to 0 \tag{3}
\]

Where (1) follows from the first property of \( P \), (2) follows from the definition of \( L \), and (3) is the definition of \( \delta(n) \).
Additional Details on CQC

For reproducibility, we include full details of our case study on Conjunctive Query Containment (CQC).

Encoding Query Tokens Table 1 shows the mapping between query tokens and their representation as one-hot vectors.

Table 1. Token representation. Each token with index $j$ is mapped to a vector with 1 in position $j$ and all other elements are zero. The dictionary size and the length of the vectors is $d = 42$.

<table>
<thead>
<tr>
<th>Type</th>
<th>Tokens</th>
<th>Index range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables</td>
<td>$x_0$ ... $x_{32}$</td>
<td>6–11, 14–40</td>
</tr>
<tr>
<td>Relations</td>
<td>$Q \ R_0 \ R_1$</td>
<td>12, 5, 4</td>
</tr>
<tr>
<td>Operators</td>
<td>$\land$ $:$</td>
<td>1, 13</td>
</tr>
<tr>
<td>Parentheses</td>
<td>( )</td>
<td>2, 3</td>
</tr>
<tr>
<td>Constants</td>
<td>0 1</td>
<td>41, 42</td>
</tr>
</tbody>
</table>

Sampling Balanced Query Pairs from $\mu$ We exploit the phase transition phenomenon to define a parametric family of query pairs $\mu(m_1, m_2)$ such that sampling $(p, q)$ from $\mu(m_1, m_2)$ with $m_1 \geq m_2$ guarantees the following:

- $p$ has $m_1$ conjunctions and $q$ has $m_2$ conjunctions.
- The probability that $p \subseteq q$ is approximately 0.5.
- The process for generating positive and negative examples is the same.

Intuitively, for a conjunctive query $p$ with a fixed number of conjunctions, the fewer variables is uses, the more “constrained” it is. For example, let $p(x_1) = R_1(x_1, x_2, x_3)$ and $q(x_1) = R_1(x_1, x_1, x_2)$. While every tuple in $R_1$ will satisfy $p$, only tuples whose first and second element are the same will satisfy $q$.

Given a fixed set of relations $R$, we define the distribution $G(X, m)$ over conjunctive queries with $m$ conjunctions, where $X$ is a set of variables as follows: first, choose $m$ relations from $R$ uniformly and with repetitions; then, conjunction variables for each conjunction uniformly and with repetitions from $X$. The constraintness of $G(X, m)$ is defined as $\alpha = \frac{m}{n}$.

Let $p \sim G(X_1, m_1)$ and $q \sim G(X_2, m_2)$ be a query pair, and let $\alpha_1$ and $\alpha_2$ be the respective constraintness. We observe that the probability of $p \subseteq q$ depends on the ratio of $\alpha_2$ and $\alpha_1$. When $\frac{\alpha_2}{\alpha_1} \gg c$ for a constant $c$, with high probability $p \subseteq q$, when $\frac{\alpha_2}{\alpha_1} \ll c$ with high probability $p \nsubseteq q$, and when $\frac{\alpha_2}{\alpha_1} \approx c$, the probability of $p \subseteq q$ is approximately 0.5. We empirically determined that for $m_1 \geq m_2$, $c \approx \frac{5}{15}$.

Finally, we define the distribution $\mu(m_1, m_2)$ over pairs of conjunctive queries $(p, q)$ as sampling $p \sim G(X_1, m_1)$ and $q \sim G(X_2, m_2)$ with $X_1$ and $X_2$ such that $\frac{m_2}{m_1} \approx c$. Since positive and negative samples are generated with the same structure and the same constraintness, syntactic features alone are unlikely to help classification.

Data Augmentation for Conjunctive Query Pairs Given a query $q$, we define the following rewrites:

- $\text{MergeVar}(q)$: Pick two variables $x, y \in vars(q)$, replace every occurrence of $y$ by $x$.
- $\text{SplitVar}(q)$: Pick a new variable $w \notin vars(q)$, and a variable $x \in vars(q)$. Each occurrence of $x$ is unchanged with probability 0.5 or replaced with $w$.
- $\text{AddConj}(q)$: Pick a conjunction $R(\ell_1, \ell_2, \ell_3)$ and add it to $q$.
- $\text{DelConj}(q)$: Pick a conjunction in $p$ and remove it.
- $\text{Shuffle}(q)$: Shuffle the order of conjunctions in $p$.

For $(p, q)$ where $p \subseteq q$, we use the following set of class-preserving rewrites: $(p, \text{MergeVar}(p), q), (p, \text{SplitVar}(p)), (p, \text{AddConj}(p), q), (p, \text{DelConj}(q), p), (p, \text{Shuffle}(p), q)$, and $(p, \text{Shuffle}(q))$.

For $(p, q)$ where $p \nsubseteq q$, we use the following class-preserving rewrites: $(p, \text{MergeVar}(q), (p, \text{SplitVar}(p), q), (p, \text{AddConj}(q)), (p, \text{Shuffle}(p), q)$, and $(p, \text{Shuffle}(q))$.