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

Supplemental Material

Gal Yehuda  Moshe Gabel  Assaf Schuster

Proof of Lemma 5.

Lemma 5. 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 6. 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 $\mathcal{M} = M_1, M_2, M_3, \ldots$, 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 6). 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 $P'$, $\frac{1}{g(n_0)}$, 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 $g(n)^{\log^{0.5} n} \geq n^c$.

The second property holds by the definition of $P$.

By Corollary 6 there exists a randomized algorithm $P^*$ such that for infinitely many $n$’s $n_1, n_2, n_3, \ldots$, 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.

$$\Pr[w = P'_n] \leq g(n) \Pr[w = P_n] \geq g(n)\epsilon(n) \to 0.$$ (1)

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

\qed
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>x0 ... x32</td>
<td>6-11, 14-40</td>
</tr>
<tr>
<td>Relations</td>
<td>Q R0 R1</td>
<td>12, 5, 4</td>
</tr>
<tr>
<td>Operators</td>
<td>∧ : 1, 13</td>
<td></td>
</tr>
<tr>
<td>Parentheses</td>
<td>( ) 2, 3</td>
<td></td>
</tr>
<tr>
<td>Constants</td>
<td>0 1 41, 42</td>
<td></td>
</tr>
</tbody>
</table>

Sampling Balanced Query Pairs from µ We exploit the phase transition phenomenon to define a parametric family of query pairs µ(m1, m2) such that sampling (p, q) from µ(m1, m2) with m1 ≥ m2 guarantees the following:

- p has m1 conjunctions and q has m2 conjunctions.
- The probability that p ⊆ 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(x1) = R1(x1, x2, x3) and q(x1) = R1(x1, x1, x2). While every tuple in R1 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 α = m/n.

Let p ~ G(X1, m1) and q ~ G(X2, m2) be a query pair, and let α1 and α2 be the respective constraintness. We observe that the probability of p ⊆ q depends on the ratio of α2 and α1. When α2 ⪆ c for a constant c, with high probability p ⊆ q, when α2 ≪ c with high probability p ⪅ q, and when α2 ≈ c, the probability of p ⊆ q is approximately 0.5. We empirically determined that for m1 ≥ m2, c ≈ 5/15.

Finally, we define the distribution µ(m1, m2) over pairs of conjunctive queries (p, q) as sampling p ~ G(X1, m1) and q ~ G(X2, m2) with X1 and X2 such that α1 ≈ 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:

- MergeVar(q): Pick two variables x, y ∈ vars(q), replace every occurrence of y by x.
- SplitVar(q): Pick a new variable w /∉ vars(q), and a variable x ∈ vars(q). Each occurrence of x is unchanged with probability 0.5 or replaced with w.
- AddConj(q): Pick a conjunction R(ℓ1, ℓ2, ℓ3) and add it to q.
- DelConj(q): Pick a conjunction in p and remove it.
- Shuffle(q): Shuffle the order of conjunctions in p.

For (p, q) where p ⊆ q, we use the following set of class-preserving rewrites: (MergeVar(p, q)), (p, SplitVar(q)), (AddConj(p), q), (p, DelConj(q)), (Shuffle(p), q), and (p, Shuffle(q)). For (p, q) where p ⪅ q, we use the following class-preserving rewrites: (p, MergeVar(q)), (SplitVar(p), q), (p, AddConj(q)), (Shuffle(p), q), and (p, Shuffle(q)).