دانشگاه صنعتی اصفهان

دانشکده مهندسی برق و کامپیوتر

عنوان: تکلیف اول درس پیچیدگی محاسباتی

نام و نام خانوادگی: علیرضا ابره فروش

شماره دانشجویی: ۹۸۱۶۶۰۳

نیم‌سال تحصیلی: پاییز ۱۴۰۲

مدرّس: دکتر رامین جوادی

۱

$c_1.n^{2^n}.2^{c_2 n.c_3^n} \in 2^{2^{O(n)}} \iff$

$\log(c_1) + 2^n \log(n) + c_2 n.c_3^n \in 2^{O(n)} \overset{c_3 \geq 2}{\iff}$

$n.c_3^n \in 2^{O(n)} \iff$

$\log(n) + n\log(c_3) \in O(n) \iff$

$n \in O(n)$

$\square$

۲

Let A and B be two languages in P. We aim to prove that there exists a polynomial-time reduction from A to B.

Since A is in P, there exists a deterministic polynomial-time Turing machine M_A that decides A. Similarly, since B is in P, there exists a deterministic polynomial-time Turing machine M_B that decides B.

Given a string z, we can check whether z is in A using M_A in polynomial time.

Now, let's define our reduction function f as follows:

Given an input string z:

- If z is in A, output x (since z is in A, $f(z)$ should be in B).
- If z is not in A, output y (since z is not in A, $f(z)$ should not be in B).

Since M_B runs in polynomial time, and the construction of $f(z)$ also runs in polynomial time, the function f is a polynomial-time computable function.

Therefore, we've shown that there exists a polynomial-time reduction from A to B.

۳

۱.۳

Assume, for the sake of contradiction, that B_1 is decidable.

If B_1 is decidable, then there exists a Turing machine M_{B_1} that decides it.

We can use M_{B_1} to decide A_1 as follows:

Given an input $< M >$, where M is a Turing machine:

1. Construct a Turing machine N such that $L(N)$ is the complement of $L(M)$, i.e., $L(N) = \Sigma^* \setminus L(M)$.
2. Encode $< M >$ and the description of N as input for M_{B_1}.
3. If M_{B_1} accepts, output "accept". If M_{B_1} rejects, output "reject".

If M_{B_1} accepts, it means that N accepts all strings, i.e., $L(N) = \Sigma^*$. This implies that $L(M)$ is empty, and thus $< M >\in A_1$.

If M_{B_1} rejects, it means that N does not accept all strings, i.e., $L(N) \neq \Sigma^*$. This implies that $L(M)$ is not empty, and thus $< M >\notin A_1$.

However, we know that A_1 is undecidable, which contradicts our assumption that B_1 is decidable.

Therefore, our initial assumption that B_1 is decidable must be false. Hence, B_1 is undecidable.

۲.۳

۴

Use the construction: The TM M' encodes the k tapes of M (including its input and output tapes) on a single tape by using locations $1, k+1, 2k+1, \cdots$ to encode the first tape, locations $2, k+2, 2k+2, \cdots$ to encode the second tape etc. For every symbol a in M's alphabet, M' will contain both the symbol a and the symbol $\hat{a}$. In the encoding of each tape, exactly one symbol will be of the "hat type," indicating that the corresponding head of M is positioned in that location. M' uses the input and output tape in the same way M does. To simulate one step of M, the machine M' makes two sweeps of its work tape: first it sweeps the tape in the left-to-right direction and records to its state register the k symbols that are hatted. Then M' uses M's transition function to determine the new state, symbols, and head movements and sweeps the tape back in the right-to-left direction to update the encoding accordingly."

We really only need to make a few changes to this to make M' into an oblivious TM M'':

• When M' updates the encoding in its right-to-left sweep, it may need to move the m̂arker to the right. However, M'' cannot stop and move right when it finds the hat, because it is oblivious. Therefore, the head of M'' must, on its right-to-left sweep, move LRL every time M' would move L, so its movement pattern will look like LRLLRLLRL... That way, the head always has the opportunity to move the m̂arker either left or right when it is encountered.

• We need to know how far out M'' needs to sweep. One option is to calculate $T(n)$ in advance and place a special marker at spot $kT(n)+1$, so that we always sweep out to $T(n)+1$ and back to the beginning. Alternatively, we simply place a marker at spot $k+1$ in step 1, and at the end of every sweep right, we have M'' move the marker k cells to the right. This will work because the original machine M cannot have gone beyond spot i in step i of its calculation.

• We also need to handle the case where M might halt at different times depending on the input. We know that M is computable in time $T(n)$ for some time constructible T . This means that there exists a machine M_T such that on all inputs of length n, M_T halts in exactly $T(n)$ steps. We can simulate M and M_T simultaneously on the same input by adding m tapes (this includes input/output tapes) for simulation of M_T to M''. States of the machine M'' can then be considered as pairs of states of the above form corresponding to M and M_T. M' will continue sweeping back and forth in the oblivious manner described above until M_T halts. Because we're sweeping out to a maximum distance of $kT(n)$, and we do it $T(n)$ times, the machine M'' runs in time $O(T(n)2)$.

منابع

[1] https://www.shiksha.com/online-courses/articles/relu-and-sigmoid-activation-function/

[2] https://medium.com/@amanatulla1606/vanishing-gradient-problem-in-deep-learning-understanding-intuition-and-solutions-da90ef4ecb54

[3] https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

[4] https://wandb.ai/ayush-thakur/dl-question-bank/reports/ReLU-vs-Sigmoid-Function-in-Deep-Neural-Networks–VmlldzoyMDk0MzI

[5] https://medium.com/swlh/why-are-neural-nets-non-linear-a46756c2d67f