IP-feladatok futásidejének becslése

Becsó Gergely, Dankó Dorottya

December 2022

1. A feladat

Az Adatbányászat tantárgy projektmunkájának keretében egészértékű programozási feladatok (IP = integer programming) futásidejének becslését végeztük el Python programnyelven. Ehhez regressziós modelleket és neurális hálózatokat használtunk.

1.1. Az adathalmaz

Az adatokat magunk generáltuk. 5000 db adott sűrűségű (0,35), 100×200-as ritka mátrixot generáltunk, amelyek elemei a 0 és 1 számok. Minden mátrixhoz sorsoltunk egy 200-dimenziós, 0 és 100 közötti egész elemekből álló célfüggvényt is. Az IP-feladatok tehát $A\mathbf{x} \leq \mathbf{b}$, $\max(\mathbf{cx})$ alakúak voltak, ahol $A \in \{0,1\}^{100\times200}$, $\mathbf{b} = \mathbf{1}$, $\mathbf{c} \in \{0,1\}^{200}$. Ezekre meghívtuk a SCIP7 IP-solvert és feljegyeztük, hogy az adott feladatokat mennyi idő alatt oldja meg és mennyi LP-feladatot old meg eközben. A target adatokról készítettünk egy kis statisztikát, ez látható a lenti táblázatban.

	Átlag	Szórásnégyzet
Futásidő	0.139839	0.021164
LP-k száma	12.6888	238.7407

2. A megoldás

2.1. Előfeldolgozás

Első lépésben a .lp formátumban lévő, IP feladatokat tartalmazó fájlokból kinyertük a számunkra fontos adatokat egy pandas dataframe-be: kétdimenziós numpy tömbök készültek a mátrixokból, egydimenziósak a célfüggvényekből.

A klasszikus modellekhez ezután „kilapítottuk”: egy-egy új oszlopot hoztunk létre mind a 20000 mátrixelemnek és a 200 célfüggvényelemnek. Így létrejött az 5000×20205 méretű dataframe, ami az adatokat tartalmazza. Ez a lépés nagy számítógépes erőforrást igényelt, a futásideje is hosszú volt, de az adathalmazt ezután le tudtuk menteni egy .csv fájlba, amit később beolvashattunk gyorsan.

A neurális hálózatokhoz ezután a célfüggvényt összefűztök a megfelelő mátrixszal, majd létrehoztunk egy egyedi pytorch Dataset-t, amely a pytorch DataLoader osztálya számára értemezhetővé teszi az adatot.

2.2. Klasszikus regressziók

Két regressziós modellt próbáltunk ki az adathalmazunkon: a lineáris regressziót és az extreme gradient boosting (XGBoost) modellt. Mivel a futásidők nagyon alacsonyak voltak (nagy részük fél másodperc alatti) és a szórásuk is kicsi volt, elvégeztük a regressziókat a használt LP feladatok számára is. Tanuló- és tesztadatokra osztottuk a dataframe-ünket. Az első 4500 soron tanult, az utolsó 500 sort használtuk tesztelésre.

A lineáris regresszióhoz a Scikit Learn LinearModel könyvtárát használtuk, az XGBoosthoz pedig az xgboost könyvtár XGBRegressor alkönyvtárát. Az alábbi táblázat mutatja az eredményeket. Az MSE (mean squared error) jelenti az átlagos négyzetes eltérést, az MAE (mean absolute error) az átlagos abszolút eltérést.

	MSE (futásidő)	MAE (futásidő)	MSE (LP-k)	MAE (LP-k)
LinearRegression	0.023387	0.12330	271.9823	13.2126
XGBRegressor	0.02059	0.11538	241.16588	12.6485

2.3. Neurális hálók

Összesen kilenc háló variációt próbáltunk ki az adathalmazon. A korábbiakhoz hasonlóan az első 4500 példán tanítottunk és az utolsó 500-on értékeltünk ki. A hálózatokban a MSE veszteségfüggvényt használtuk, Adam optimizert 10^{-3} kezdeti tanulási rátával és 10^{-3} "weight decay"-el. A tanításokat egy NVIDIA GeForce RTX 2080 Ti GPU segítségével végeztük.

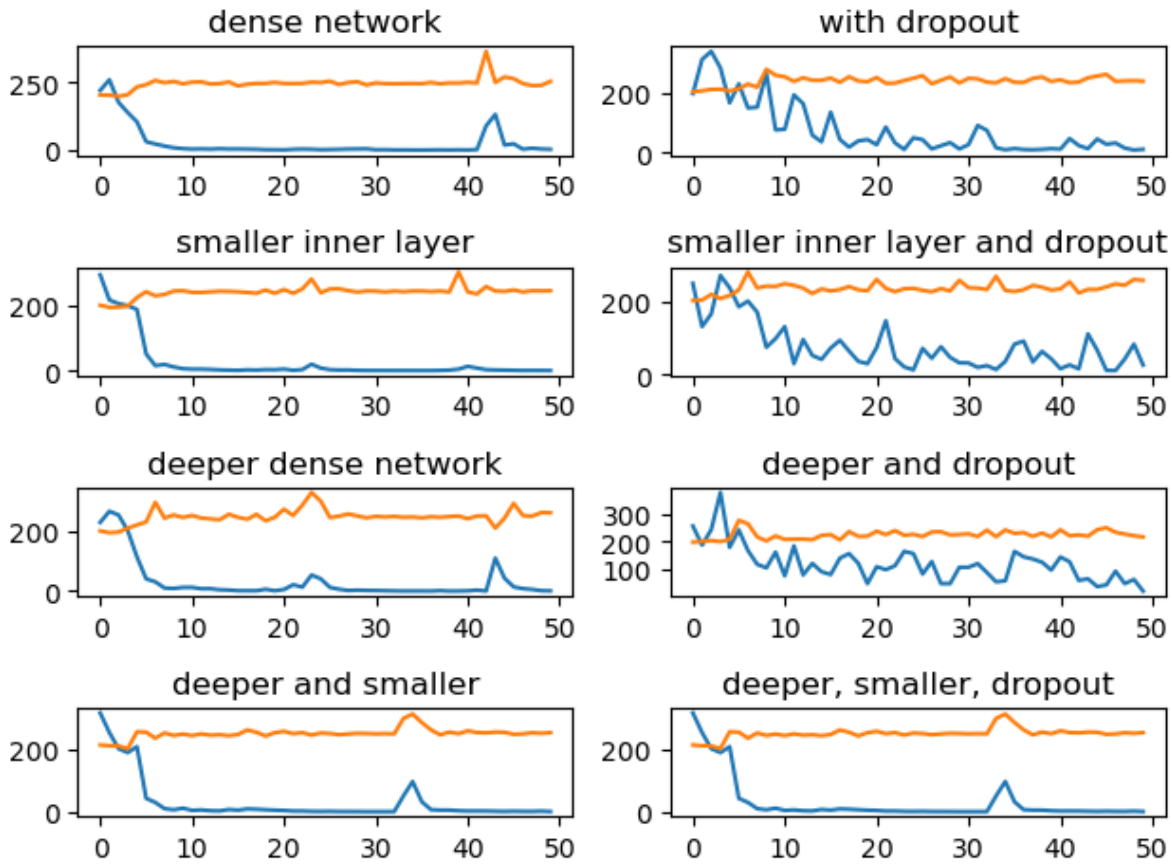
Először egy egyszerű, sűrű hálóval próbálkozunk, 20200 bemenettel, egy 512 széles belső réteggel és egy 1 elemű kimenettel. Mivel a modell a feladatból semmit sem tudott megtanulni, azaz rögtön elkezdődött a túltanulás folyamata, háromféle módon próbáltunk javítani:

- a belső réteg 128-ra való csökkentésével,
- dropout rétegek hozzáadásával,
- több belső réteg alkalmazásával.

Ez összesen nyolc variácó.

A belső réteg csökkentése sikeresen csökkentette a háló információtárolási képességét, így a tanítás végére kevésbé tudott túltanulni, ugyanakkor még mindig nem tudott lényegi információt megszerezni a feladatról. A dropout jelentősen lassított a tanulási folyamaton és nem is sikerült olyan alaposan túltanulni. Egy belső réteg hozzáadása pedig semmilyen lényegi változást nem okozott a tanítás menetében.

Alább látható a tanítás alatt elért tanítási és teszt error. Itt a kék jelöli a tanításit, a narancssárga a teszhibát.

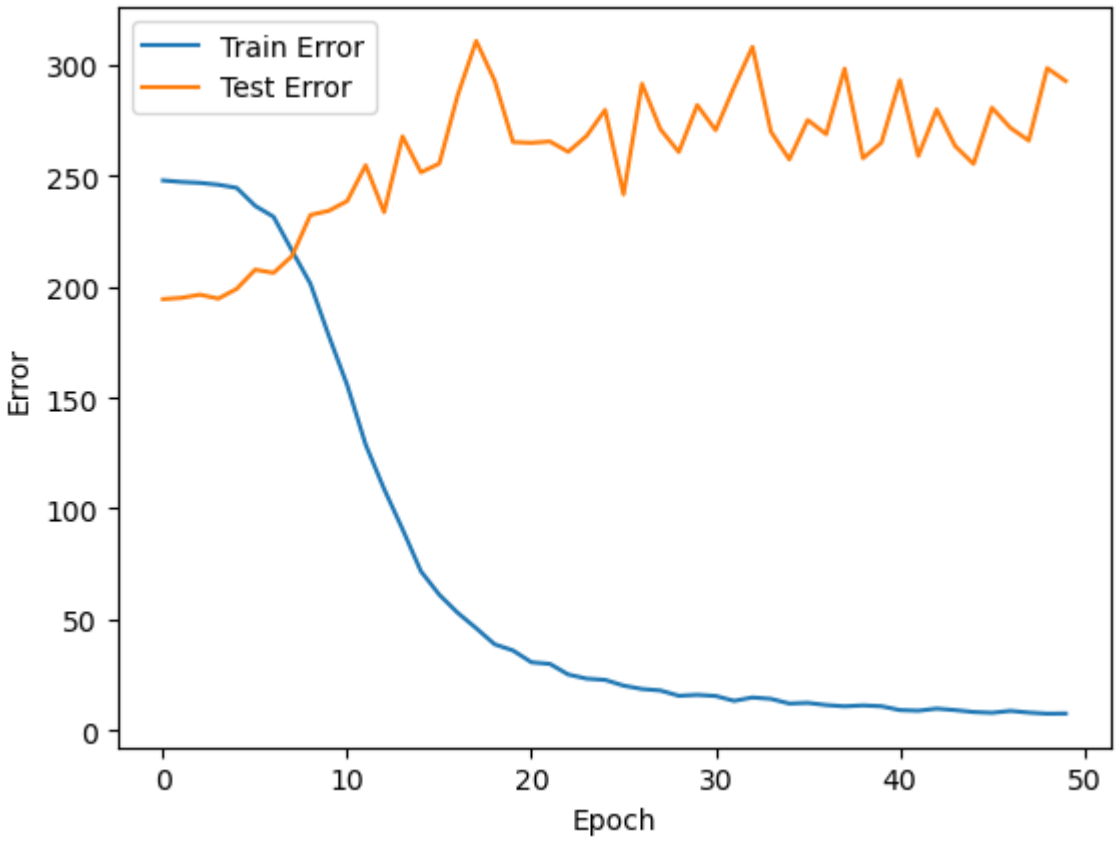


Ezen kívül egy összetettebb rendszert is megpróbáltunk betanítani:

A mátrix sorait, az IP feladat feltételeit egy LSTM hálózat segítségével 128-dimenziós vektorokba kódoltuk. Ez azért volt hasznos, mert ha működik, így ezt a hálót lehet alkalmazni más méretű feladatokra is. Ennek a kimenetét egy attention réteg segítségével átalakítottuk, így az egyes feltételek kódolásában a többi feltétel kontextusát is figyelembe vettük. Ezután a beágyazásokat egy sűrű réteggel egyetlen számmá húztuk össze, majd a beágyazások számait aggregáltuk szintén egy sűrű réteggel egyetlen számmá.

Az implementálás alatt valamilyen hibát vétettünk és nem lehetett 1 méretű batchnél nagyobbakkal dolgozni.

A tanulás ívét itt láthatjuk:



Sajnos a korábbi modellekhez hasonlóan ez a modell sem volt képes lényegi tudást megtanulni a feladatról, szépen látszik a túltanulás folyamata.

3. Konklúzió

A kipróbált modellek egyike sem tudott lényegi tudást kinyerni az adatból. Lehetséges, hogy az adatgenerálásnál csúszott be hiba, esetleg a címkék keveredtek össze, vagy nincs lényegi összefüggés a vizsgált IP-feladatok alakja és a futásidő között.