Forgalomelőrejelzés beágyazásokkal

Szeiler Pál
Témavezető: Béres Ferenc, Molnár András József, Benczúr András

1 A feladat leírása

A feladat egy úthálózat forgalmának előrejelzése. Ehhez rendelkezésre állnak az úthálózaton rögzített pozíciójú szenzorokból származó adatok az áthaladó forgalomról. Ezen információkon túl adott még a szenzorok elhelyezkedése is. A témában a két legfontosabb adathalmaz a METR-LA és a PEMS-BAY [5]. Mi a METR-LA egy részhalmazával foglalkoztunk. Ebben az adathalmazban 207 szenzor 4 hónapnyi adata állt rendelkezésre öt perces időablakokban. Nekünk az egy óra alatt gyűjtött adatokból kellett a következő egy órát prediktálni. Az adat periodikus, azaz meg lehet figyelni, hogy napközben alacsony az átlagsebesség a mért helyeken, míg kora reggel és éjszaka magas. Az adathalmazban előre adott volt a train/validation/test vágás.

1.1 A szenzorok elhelyezkedése

Minden szenzorhoz adottak voltak a koordinátái, valamint a szenzorok közötti távolságok. A METR-LA esetében elérhető a szenzoroknak egy olyan gráfja, ahol két csúcs között csak akkor megy él, ha a csúcsokhoz tartozó szenzorok közötti távolság egy adott határérték alatt van. Az így kapott gráf egy nagy összefüggő komponensből áll valamint egy izolált csúcsból. Mivel a gráf nem tükrözi az úthálózat struktúráját, - ugyanis előfordulhat, hogy két csúcs között vezet él annak ellenére, hogy a megfelelő szenzorok két párhuzamos út forgalmát mérik, - a feladathoz hozzátartozott, hogy egy olyan módszert találjunk, amely egy csúcshoz meg tudja határozni azon másik csúcsokat, melyek adatai hatással vannak az adott csúcsra. Ezért a feladathoz idősorok modellezésére alkalmas neurális hálókon túl, - például rekurrens hálók, konvolúciós hálók, - gráf embeddingeket is használtunk.

2 Gráfbeágyazás

Annak érdekében, hogy a neurális hálóknak információt adjunk a szenzorgráfról, különböző gráfbeágyazásokat használtunk. Egy gráfbeágyazás során minden csúcshoz rendelünk vektorokat, ami az élsúlyokat figyelembe veszi. A legtöbb beágyazó algoritmusnál 32 dimenziós vektorokat generáltunk. A használt embeddingek:

- DeepWalk [1]
- NetMF [6]

- Diff2Vec [7]
- Role2Vec [4]
- Walklets [3]

Ezeken kívül kipróbáltunk még párat (pl. GLEE [8], GraRep [2]), viszont azok teljesítménye jócskán elmaradt a többitől.
A fenti algoritmusok működése eltérő, és a gráf más-más tulajdonságait fogják meg. Az úthálózat egyes útjain a forgalom nagysága feltehetőleg közel azonos, ez viszont nem feltétlenül igaz az egymáshoz közeli csúcsokra abban az esetben, ha azok nem ugyanazon az úton vannak. A kipróbált algoritmusok közül a **NetMF** mátrix faktorizáción alapszik, míg a **Deepwalk** a gráf csúcsaiból indított véletlen sétákkal tanulja meg a gráf szerkezetét, és az aktuális reprezentációt updateli minden egyes sétával. A **Walklets** is csúcsonként megadott számú véletlen sétát generál, de a séták néhány csúcsát átugorja. A **Diff2Vec** véletlen séták helyett egy csúcs szomszédságában tesz meg egy Euler sétát. Ezek a módszerek elsősorban a sűrűbb részgráfokat veszik észre, szemben a **Role2Vec** algoritmussal, amelyben minden csúcshoz rendelünk egy típust és a véletlen séták itt csúcsok helyett típusok sorozatai lesznek. Emiatt a gráfban például a nagyonn kereszteződések lesznek hasonlóak, vagy azon csúcsok, melyek egy út olyan szakaszán vannak, ahol nincs a közelben kereszteződés.

3 Használt módszerek

A félév elején egy DCRNN [5] implementációt próbáltunk ki, amely 2017-ben state of the art modellnek számított. A jelenlegi state of the art modell egy gráf konvolúciós rekurrens neurális háló [10][9].
A félév további részében pedig sűrű rétegekből álló, konvolúciós valamint rekurrens neurális hálókat implementáltunk. A modelleket több metrikában hasonlítottuk össze. Az összehasonlításhoz használt metrikák:

- MAE (Mean Absolute Error)
- MSE (Mean Squared Error)
- RMSE (Root Mean Squared Error)

A különböző modellek futásidejét is összehasonlítottuk. A hálózati információ nélkül tanított modellek jóval gyorsabbak voltak az embeddinggel tanított modelleknél, köszönhetően az input lényegesen kisebb méretének.
A hálók struktúráját úgy építettük fel, hogy könnyen össze tudjuk hasonlítani, hogy egy adott réteg mennyivel járult hozzá a teljesítmény növekedéséhez/csökkenéséhez. Ezek alapján fontosnak találtuk a regularizációs módszereket, mint a batchnorm[1] és a dropout[2]. Ezek lehetővé tették a gyorsabb tanulást és a hiperparaméterek könnyebb beállítását. Más regularizációt, például early stopping-ot nem használtunk,

[1] https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm1d.html
[2] https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html

de a teszteléshez az adott beállítások mellett - az embedding típusa és normalizáció - a validációs halmazon legjobb teljesítményt elérő modelleket választottuk. Az epochok számának limitálása és a regularizáció miatt ezek a modellek rendre a legutolsó tanítási fázisban kapott súlyokat használták.
Az adatok könnyebb kezelhetősége érdekében minden háló első rétege egy fully connected réteg volt, ami az inputot összenyomta és a formáját átalakította, így minegyik hálónk inputja (batchméret, idő, featureszám, csúcsszám), ahol a featureszám a gráf egy csúcsánál rendelkezésre álló adatok száma.

3.1 A normalizálás szerepe

A hálókat az input normalizálásával, valamint anélkül is tanítottuk. Azokban az esetekben, amikor embeddinget is használtunk, az embedding vektorokat nem normalizáltuk, valamint a validációt és a tesztelést is a visszatranszformált adatokon végeztük el, hogy az eredményül kapott loss érték összehasonlítható legyen a nem normalizált inputon történő tanítás eredményével. A hatásai eltérőek voltak az használt embedding függvényében, de az minden embedding esetén igaz volt, hogy a normalizált adatokon gyorsabb volt a konvergencia és könnyebb volt a hiperparaméterek beállítása. Valamint az is megfigyelhető volt, hogy ugyan normalizált inputon a modellek közel ugyanúgy teljesítettek a validációs adathalmazon mint nem normalizált inputon, a teszt halmazon nyújtott teljesítményük viszont jelentősen rosszabb volt.

3.2 Fully connected modell

Az első modell, amit kipróbáltunk, csak sűrű rétegekből[3] állt. Fully connected hálókat nem szokás idősorok prediktálásához használni, nekünk egyrészt összehasonlítási alapként szolgált a rekurrens és konvolúciós hálókkal, másrészt felhasználtuk a többi modellhez dimenziócsökkentő eljárásként.

3.3 Rekurrens neurális hálók

Rekurrens neurális hálóknak több verzióját is kipróbáltuk.
Az első változatban nem használtunk preprocesszálást, azaz egy sűrű réteg után az LSTM[4] réteg közvetlenül megkapta a csúcsokhoz tartozó featureöket. Ez az embeddingek használatakor nagy volt, ami a futásidőt növelte valamint a featureszám függött az embedding dimenziójától. Ennek kiküszöbölésére használtuk a fent említett sűrű modellt preprocesszálásra, így már egy rögzített featureszámot kapott az lstm réteg. A hátránya viszont, hogy közvetlenül nem látta az lstm az embeddinget. Ezt a változatot kipróbáltuk úgy is, hogy egy előre betanított sűrű rétegekből álló modellt használtunk aminek a súlyait lefagyasztottuk, de így a háló teljesítménye romlott.

[3] https://pytorch.org/docs/stable/generated/torch.nn.Linear.html
[4] https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

3.4 Konvolúciós háló

Idősorok prediktálásához a rekurrens hálókon kívül az egy dimenziós konvolúció[5] is használható. Itt szintén problémát okozott a nagy featureszám, így a rekurrens hálóhoz hasonlóan itt is néztünk preprocesszálást használó, illetve nem használó változatot. Mindkét verzióban 15 perces ablakokat használtunk és padeltük az inputot, hogy a konvolúciós rétegek outputja a következő 60 percet prediktálja. A konvolúciós rétegek után használtunk még egy sűrű réteget, ami a kívánt méretűre nyomja össze a konvolúció outputját.

4 A hálók teljesítménye

A hálók teljesítményét a fenti három metrikában hasonlítottuk össze a validációs és teszt halmazon. Azt vártuk, hogy egyrészt az LSTM és a konvolúciós hálók jobb teljesítményt nyújtanak, mint a fully connected modell. Továbbá ha a hálóknak rendelkezésére áll a hálózatról valamilyen információ, az is javítja a pontosságot. A három metrika közül elsődlegesen a MAE-t használtuk. Minden mérést tízszer végeztünk, az ábrákon az átlagos scoreokat jelenítjük meg. Minden esetben 30 epochig tanítottuk a hálókat, ugyanis ez volt az az epochszám, amikor már a validációs halmazon a MAE loss felvette a minimumát, és innentől a hálók már túltanultak.

A modellek közötti különbségek

Összességében elmondható, hogy a fully connected modell jól teljesített mind futásidőben, mind MAE értékben. A konvolúciós modellel tudtunk ezen javítani, viszont ahogy a 2. ábrán is látszik, előfordul, hogy a kezdetben inicializált súlyokkal a háló semmit sem tanul. Az LSTM nyújtotta a legrosszabb teljesítményt, 4.3 alatti MAE értéket nem tudtunk vele elérni, emiatt a tanításáról nem is mutatunk be ábrákat.

Normalizálás

Az 1. figurán látható a normalizáció hatása. A normalizált adatokon könnyebben tanult a háló és jól láthatóan nincsenek akkora kilengések, mint a nem normalizált esetben. Valamint MAE érték is rendre alacsonyabb. Viszont a teszt halmazon a nem normalizált adatokon tanított hálók sokkal jobban teljesítettek, a konvolúciós modell vonatkozó számait a 1. táblázat tartalmazza. Az eltérés okára nem jöttünk rá, feltehetőleg a tesz és validációs halmazoknak eloszlásbeli különbsége miatt van.

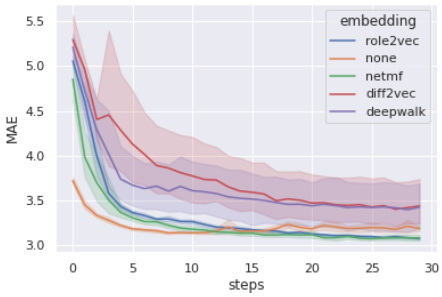
Embeddingek

Az 1. és 2. figurákon látható, hogy a validációs halmazon hálózati információ nélkül a fully connected és a konvolúciós modell a kísérletek átlagát tekintve nagyon jól teljesített, a MAE scoreban a legkisebb szórást is a gráf ismerete nélkül érték el a modellek. A fully connected modell esetén csupán két embedding, a **Role2Vec** és

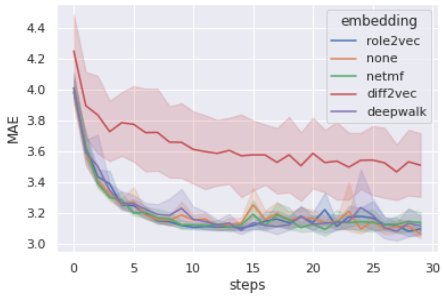
[5] https://pytorch.org/docs/stable/generated/torch.nn.Conv1d.html

	None	DeepWalk	NetMF	Diff2Vec	Role2Vec	Walklets	GraRep
Norm +	4.254	4.380	4.212	4.543	4.267	4.368	6.860
Norm -	3.486	3.502	3.497	3.546	3.415	3.509	6.776

Table 1: A konvolúciós modell legjobb teljesítményei a teszt halmazon (MAE).



(a) Normalizáció



(b) Normalizáció nélkül

Figure 1: A Fully connected modell teljesítménye a validációs halmazon.

a **NetMF** teljesített jobban, a konvolúciós hálónál viszont csak a **NetMF** tudta megközelíteni.

4.1 Teljesítmény a teszt halmazon

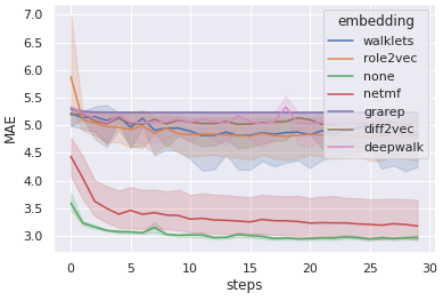
A teszt halmazon az átlagos teljesítményt tekintve az embeddingek közül a **NetMF** bizonyult a legjobbnak. Ezt követte a **Role2Vec**, a **DeepWalk** és a **Walklets**. Amint az a 3. ábrán látható, az embeddingek használata jelentősen növeli a szórást, feltehetőleg a jóval nagyobb inputméret miatt. A 1. táblázatban foglaltuk össze a konvolúciós modellnek a teszt halmazon elért legjobb eredményeit (MAE).

4.2 Futásidő

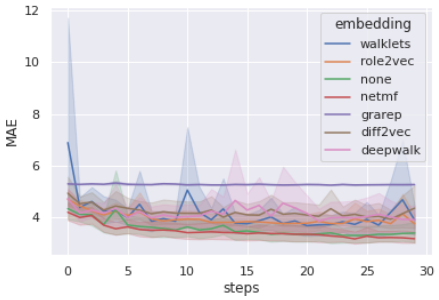
Az 2. és 3. táblázatokban foglaltuk össze az átlagos futásidőket másodperben 30 epochra. Nem meglepő módon a legjobb futásidőt akkor kaptuk, amikor nem alkalmaztunk embeddinget, viszont a különböző beágyazások használatakor nem voltak nagy különbségek a futásidőben. A normalizált és nem normalizált input közötti differencia feltehetően az adathalmaz transzformációja miatt van.

	None	DeepWalk	NetMF	Diff2Vec	Role2Vec
Norm +	584	865	818	944	911
Norm -	541	808	776	888	863

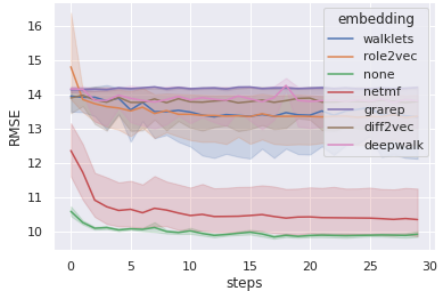
Table 2: A Fully connected modell átlagos futásideje másodpercben.



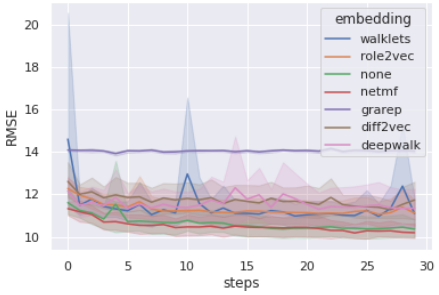
(a) Normalizáció



(b) Normalizáció nélkül

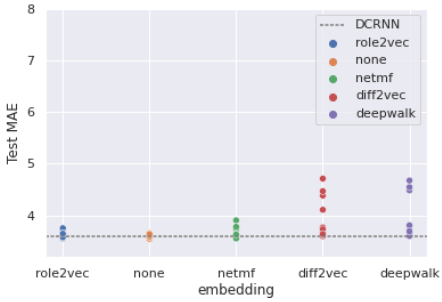


(c) Normalizáció

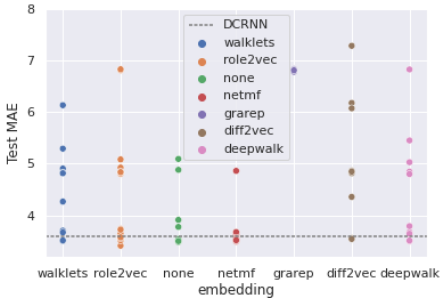


(d) Normalizáció nélkül

Figure 2: A konvolúciós modell teljesítménye a validációs halmazon.



(a) Fully connected modell



(b) Konvolúciós modell

Figure 3: A modellek teljesítménye normalizáció nélkül a teszt halmazon.

	None	DeepWalk	NetMF	Diff2Vec	Role2Vec	Walklets	GraRep
Norm +	838	1103	1091	1182	1111	1155	1203
Norm -	779	1050	1034	1122	1064	1099	1139

Table 3: A konvolúciós modell átlagos futásideje másodpercben.

5 Összefoglalás

A félév célja az volt, hogy találjunk olyan embeddingeket, amelyek a forgalomelőrejelzéshez hasznos információt kódolnak az úthálózatokról. Annak ellenére, hogy a hálózati információ nélkül tanított modelljeink is elfogadható teljesítményt nyújtottak, A **NetMF** és a **Role2Vec** beágyazó algoritmusok használatával tudtunk javítani a pontosságon.

Irodalomjegyzék

[1] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "DeepWalk". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, Aug. 2014. DOI: 10.1145/2623330.2623732. URL: https://doi.org/10.1145%2F2623330.2623732.

[2] Shaosheng Cao, Wei Lu, and Qiongkai Xu. "GraRep: Learning Graph Representations with Global Structural Information". In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. CIKM '15. Melbourne, Australia: Association for Computing Machinery, 2015, pp. 891–900. ISBN: 9781450337946. DOI: 10.1145/2806416.2806512. URL: https://doi.org/10.1145/2806416.2806512.

[3] Bryan Perozzi et al. *Don't Walk, Skip! Online Learning of Multi-scale Network Embeddings*. 2016. DOI: 10.48550/ARXIV.1605.02115. URL: https://arxiv.org/abs/1605.02115.

[4] Nesreen K. Ahmed et al. *Learning Role-based Graph Embeddings*. 2018. DOI: 10.48550/ARXIV.1802.02896. URL: https://arxiv.org/abs/1802.02896.

[5] Yaguang Li et al. "Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting". In: *International Conference on Learning Representations (ICLR '18)*. 2018.

[6] Jiezhong Qiu et al. "Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec". In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM. 2018, pp. 459–467.

[7] Benedek Rozemberczki and Rik Sarkar. "Fast Sequence Based Embedding with Diffusion Graphs". In: *International Conference on Complex Networks*. 2018, pp. 99–107.

[8] Leo Torres, Kevin S Chan, and Tina Eliassi-Rad. "GLEE: Geometric Laplacian Eigenmap Embedding". In: *Journal of Complex Networks* 8.2 (Mar. 2020). Ed. by Ernesto Estrada. DOI: 10.1093/comnet/cnaa007. URL: https://doi.org/10.1093%2Fcomnet%2Fcnaa007.

[9] Renhe Jiang et al. "MegaCRN: Meta-Graph Convolutional Recurrent Network for Spatio-Temporal Modeling". In: *arXiv preprint arXiv:2212.05989* (2022).

[10] Renhe Jiang et al. "Spatio-Temporal Meta-Graph Learning for Traffic Forecasting". In: *arXiv preprint arXiv:2211.14701* (2022).