

TDK-dolgozat

Sebők Mátyás

Magyarország elektromos fogyasztásának rövid távú előrejelzése mesterséges intelligenciával

EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

INFORMÁCIÓS RENDSZEREK TANSZÉK



Szerző:

Sebők Mátyás

programtervező informatikus BSc

3. évfolyam

Témavezető:

dr. Vincellér Zoltán

mesteroktató

Budapest, 2023

Tartalomjegyzék

1. Bevezetés	3
2. Szakirodalmi áttekintés	5
2.1. Problémafelvetés	5
2.2. Kiértékelési és tanítási módszerek	6
2.2.1. Kiértékelési metrikák	6
2.2.2. Tanítási módszer	7
2.3. Gépi- és mély tanulási modellek	8
2.3.1. Véletlenszerű Erdő Regresszor	8
2.3.2. Konvolúciós háló	9
2.3.3. Temporális konvolúciós háló	10
2.3.4. LSTM	11
2.3.5. GRU	12
2.4. Előrejelzési stratégiák	13
2.4.1. Rekurzív stratégia	13
2.4.2. MIMO stratégia	14
2.4.3. Seq2seq stratégia	15
2.4.4. Többmodelles rekurzív stratégia	16
3. Módszertan	18
3.1. Fejlesztői környezet	18
3.2. Felhasznált adatok	18
3.2.1. MAVIR adatok	19
3.2.2. OMSZ adatok	20
3.2.3. EDA	21
3.3. A kiválasztott megközelítések, optimalizációjuk	24
3.3.1. MIMO, Véletlenszerű Erdő Regresszor	25
3.3.2. MIMO, Konvolúciós Háló	26

3.3.3.	MIMO, Temporális Konvolúciós Háló	27
3.3.4.	MIMO, LSTM	29
3.3.5.	Seq2seq, GRU	29
3.3.6.	Rekurzív stratégia, GRU és LSTM	31
3.3.7.	Többmodelles rekurzív stratégia	32
4.	Összefoglalás és eredmények	35
4.1.	Eredmények	35
4.2.	Kitekintés	39
4.3.	További alkalmazások	39
	Köszönetnyilvánítás	40
	Irodalomjegyzék	41
	Ábrajegyzék	45
	Táblázatjegyzék	46

1. fejezet

Bevezetés

Az idősoros adaton végzett előrejelzés egy gyakran kutatott téma, az elmúlt években több gépi- és mély tanulási modell illetve stratégia is megjelent amelyek jobbnál-jobb eredményeket mutatnak. Az ilyen megközelítések általában jobban alkalmazhatóak mint a különböző statisztikai modellek, mivel tetszőleges függvények közelítését biztosítják és külső változókat tudnak eredményesen figyelembe venni. Problémát jelenthet viszont az ún. fekete doboz működésük, hiszen az ilyen modellek belső reprezentációi nehezen értelmezhetőek, magyarázhatóak. Ezért csak akkor érdemes őket használni, ha sokkal jobb eredményeket mutatnak statisztikai párjaiknál.

Az országos elektromos fogyasztás rövid-távú előrejelzése fontos feladat, mert lehetőséget ad arra, hogy a különböző elektromos áramtermelői és a piaci résztvevők a pillanatnyi helyzetben a lehető legjobb döntéseket hozzák, erőforrásaikat eredményesen használják. A dolgozatban Magyarország áramfogyasztásának előrejelzésével foglalkozom. Célja, hogy a szakterületen gyakran alkalmazott modelleket és stratégiákat összevesse egymással, valamint az ún. rekurzív stratégiának egy új, több modelles változatát kidolgozza és az előbb említett megközelítésekkel összehasonlítsa. Az előrejelzést nem csak kizárólagosan országos szinten lehet elvégezni, hanem általánosítható részletesebb adatokra, amennyiben ez rendelkezésünkre áll, és ekkor egy elkülönített csomópont gondoskodhat a hozzátartozó fogyasztókról (elektromos áramfogyasztók) és termelőkről. Az adatokat az OMSZ és MAVIR által közzétett, publikusan elérhető adatbázisából szerveztem össze egy országos idősorba.

Idősoros adatnak nevezünk olyan adathalmazokat, melyekben az adatok egyenlő lépésközönként szerepelnek idő szerint rendezetten. Ilyen adatokat nem mindig egyszerű találni, mivel hosszantartó mérésekre van szükség, valamint felvetnek problémákat, ha

egy-egy mérés kimarad. Az elektromos fogyasztást befolyásolják külső tényezők is, mint az időjárás, az ünnepnapok és munkanapok, valamint az aktuális időpont és dátum is segíthet az előrejelzésekben. Ezen külső változók is segítik a gépi illetve mély tanulási modelleket az előrejelzésekben, hiszen tetszőleges mennyiségűt figyelembe tudnak venni, ha ennek van értelme az előrejelzés tekintetében.

Általában az idősoros előrejelzések ebben a témában napi vagy annál nagyobb lépésközökkel dolgoznak, illetve az órás előrejelzések is nagyobb távra szólnak. A dolgozatban minden módszert úgy alkalmazok, hogy minden időpillanatban a következő 3 órára adok előrejelzést az előbb említett optimalizáció céljából. A 3 órás távot azért választottam, hogy a megközelítéseket össze lehessen hasonlítani az egy lépésre adott és pár lépéssel előre adott előrejelzéseik alapján is. Fontos viszont emellett, hogy maradjunk rövid távon, hogy az előrejelzések pontosak legyenek a már említett okokból.

Az idősoros előrejelzések feladata speciális stratégiákat és ehhez igazított modelleket követel meg. Általánosságban az 1 dimenziós konvolúciós hálót, melynek többdimenziós párja a képfeldolgozásban elterjedt, valamint a rekurrens hálók különböző fajtáit alkalmazzák ilyen feladatokra. Minden megközelítésnek megvannak a maga előnyei és hátrányai, így legtöbbször csak az adott adathalmazra, feladattípusra tudunk pontos eredményeket kapni, viszont a felfedezett új megközelítések és létrehozott implementációk könnyen kiértékelhetők más adatokon és feladatokon.

2. fejezet

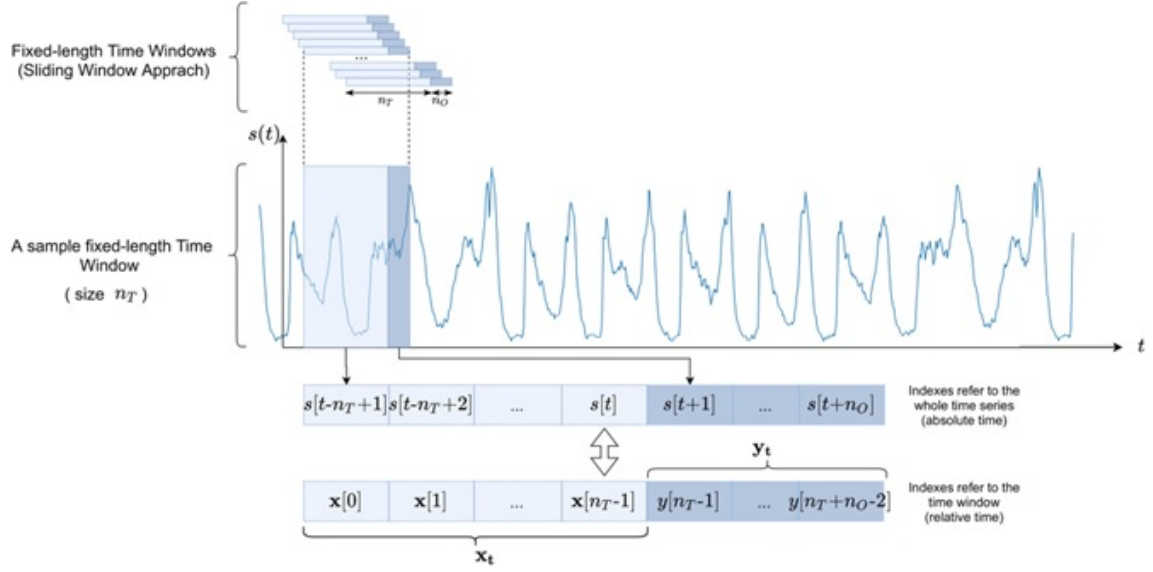
Szakirodalmi áttekintés

2.1. Problémafelvetés

[1] leírja az idősoros előrejelzési feladatot, miszerint a többlépéses idősoros előrejelzésben adott egyenletes felosztású $s = [s[0], s[1], \dots, s[T]]$ $s \subset \mathbb{R}$, ami rendezett idő szerint, és esetünkben órás felosztású. Gépi- és mély tanulás szempontjából fontos, hogy csak fix hosszúságú ablakokban dolgozzunk, ennek hossza legyen n_T . A predikciós ablak hossza legyen n_O . A dolgozatban felügyelt tanítást alkalmazok, legyen egy t időpillanatban $x_t = [s[t - n_T + 1], \dots, s[t]]$ és $y_t = [s[t + 1], \dots, s[t + n_O]]$, ahol $x_t, y_t \subset s$. A mozgó ablakos felosztás szemléltetése a 2.1. ábrán látható.

A modell egy paraméteres függvény f , melynek paramétervektora legyen θ . A gépi- és mély tanulási modellek paramétervektorát általános esetben csak közelíteni lehet, legyen $\hat{\theta}$. A jelöléseket használva egy t időpillanatban a modell kimenete $\hat{y}_t = f(x_t, \hat{\theta})$, ami az eredeti y_t egy közelítése. Mivel $y_t \in \mathbb{R}^{n_O}$, ezért regressziós feladatról van szó, a kimenet nem diszkrét értékekből való.

A probléma kiterjesztésében $s \subset \mathbb{R}^d$, $d - 1$ a külső változók száma. $x_t[i][k]$ jelölje a szekvenciában az i -edik elem k -adik változóját, ahol $i \in [0..n_T)$ és $k \in [0..d)$. Szemléletesen ez azt jelenti, hogy az előrejelezni kívánt változón kívül külső változókat is alkalmazunk, melyek segítik az előrejelzést, például időjárási jellemzőket vagy dátum illetve időpont jellemzőket.



2.1. ábra. A csúsztatott ablakos megközelítés

Forrás: [1]

2.2. Kiértékelési és tanítási módszerek

2.2.1. Kiértékelési metrikák

Az MAE (Mean Absolute Error), átlagos abszolút eltérés egy nagyon szemléletes és könnyen érthető metrika. Minden valós és becsült értékpár különbségének az abszolút értékének átlaga, összefoglalja, hogy a modell átlagosan mennyit téved. Önmagában kevés egy modell kiértékeléséhez, de jó kiindulópont.

$$MAE = \frac{1}{N} \sum_{j=1}^N |y_j - \hat{y}_j| \quad (2.1)$$

Az MSE (Mean Squared Error), átlagos négyzetes eltérés egy nagyon gyakran használt metrika elemzéskor és mély tanulási modellek tanításakor is, ugyanis egy deriválható és konvex függvény. Ez elősegíti a mély tanulási modellek tanítását, a megfelelő $\hat{\theta}$ paramétervektor megtalálását.

$$MSE = \frac{1}{N} \sum_{j=1}^N (y_j - \hat{y}_j)^2 \quad (2.2)$$

Az RMSE (Root Mean Squared Error), az átlagos négyzetes eltérés négyzetgyöke jól összefoglalja egy modell eredményeit. Az átlagos abszolút eltéréssel szemben ez a metri-

ka erőteljesebben reagál a kiugró értékekre, avagy a nagyságrendekkel nagyobb hibákra.

$$RMSE = \sqrt{\frac{1}{N} \sum_{j=1}^N (y_j - \hat{y}_j)^2} \quad (2.3)$$

Az MAPE (Mean Absolute Percentage Error), az átlagos abszolút eltérés százalékban szemlélteti a legjobban egy adott modell teljesítményét, hiszen nem szükséges az adatok nagyságrendjéről semmilyen ismeret.

$$MAPE = \frac{1}{N} \sum_{j=1}^N \frac{|y_j - \hat{y}_j|}{y_j} \times 100 \quad (2.4)$$

Az MPE (Mean Percentage Error), az átlagos eltérés százalékban egy előjeles metrika. Azt szemlélteti, hogy átlagosan a modell mekkora mértékben becsüli alul vagy felül az értékeket. Jobb belátást biztosít a modell előrejelzéseibe és szintén egy könnyen értelmezhető metrika.

$$MPE = \frac{1}{N} \sum_{j=1}^N \frac{y_j - \hat{y}_j}{y_j} \times 100 \quad (2.5)$$

Modellek kiértékelésekor keresztvalidációt (cross validation) szokás alkalmazni. Ennek lényege, hogy véletlenszerűen választunk az adatból olyan részeket, amiken az említett metrikákat kiértékeljük, a maradék részen pedig a modellt tanítjuk. Ezt ismételjük többször, majd az eredményeket átlagolva kapjuk a modell teljesítményét. Erre mutat példát a 2.2. ábra bal oldala.

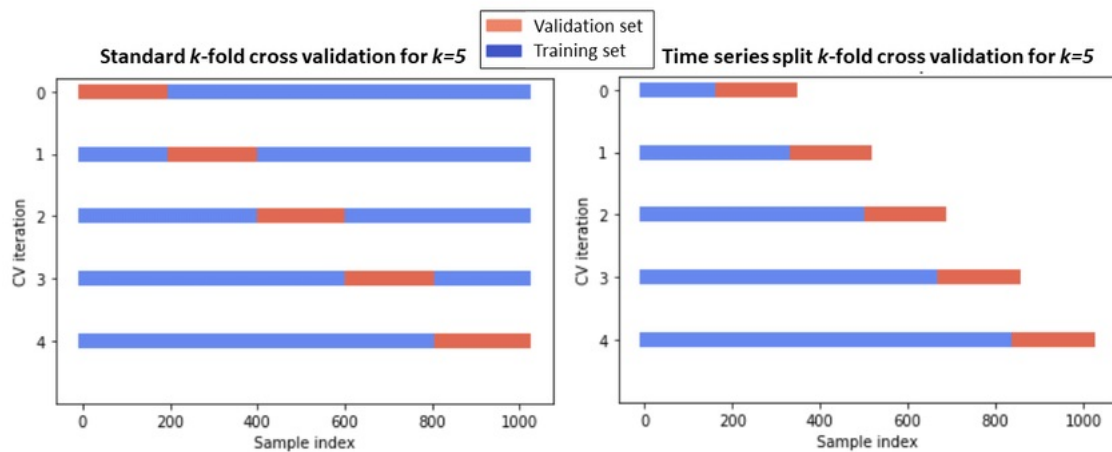
2.2.2. Tanítási módszer

A mély tanulási modellek optimalizációjára az SGD (Stochastic Gradient Descent) különböző fajtái a legelterjedtebbek. [2] az Adam optimalizációs algoritmust mutatja be. [3] bemutatja az Adam algoritmus Nesterov-momentummal történő gyorsítását, ami az Adam algoritmusnál jobb eredményeket ad gyorsaság és pontosság szempontjából is a tesztek szerint. Ezek alapján a dolgozatban az NAdam algoritmussal fogom végezni az f paraméteres modellek $\hat{\theta}$ paramétervektorainak keresését.

A tanításhoz meg kell határozni egy tanító-, egy validációs- és egy teszhalmazt. A tanítóhalmazon végezzük a paraméter optimalizációt vagy tanítást. A validációs halmaz arra szolgál, hogy mutassa a modell teljesítményét olyan adaton tanítás közben, amit még nem látott, nem tanult rá. Ezt a halmazt használjuk az ún. Early Stop, vagy korai megállás

algoritmusában. Automatikusan figyeljük a validációs halmazon hozott eredményeket, és amint rosszabb eredményeket kapunk a korábbiaknál, megállítjuk a modell tanítását, ezzel elkerülve a túltanulás jelenségét. Ezután értékeljük ki a metrikákat (2.2.1.). A 3 részre osztás azért szükséges, mivel ilyen esetben valójában nem csak a tanító-, de a validációs halmazra is optimalizálunk, a teszhalmaznak viszont kötelező olyannak lennie, amit a modell nem látott tanítás közben, hiszen ide helyettesíthetők be azok az adatok, amiket éles működésben használnia kell.

Általában a fent említett 3 halmazt véletlenszerűen célszerű meghatározni, viszont idősoros adatoknál nincs értelme például a 2016-os adatokat előrejelezni úgy, hogy a 2017-es adatokon tanítottuk a modellt. Ezért jött létre az idősoros adatfelosztás keresztvalidációhoz (time series split cross validation). Ennek szemléltetése a 2.2. ábrán látható.



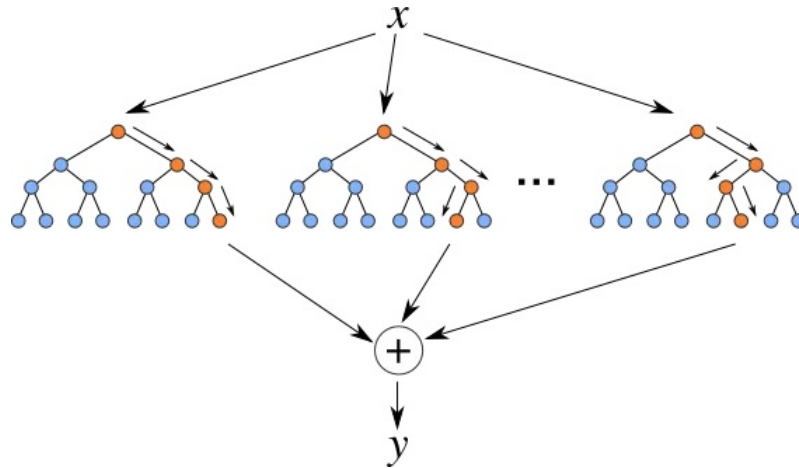
2.2. ábra. Adatfelosztás keresztvalidációhoz
 Bal oldalon: klasszikus keresztvalidációs felosztás
 Jobb oldalon: idősoros keresztvalidációs felosztás
 Forrás: ResearchGate

2.3. Gépi- és mély tanulási modellek

2.3.1. Véletlenszerű Erdő Regresszor

A Véletlenszerű Erdő (*Random Forest*), az egyetlen a dolgozatban szereplő nem mély tanulási algoritmus. Mivel nem mély tanulási algoritmus, ezért a fent említett NAdam optimalizáció helyett az ún. vágás szabályt alkalmazza. [4] mutatja be működését és tanítását.

Több döntési fából álló modell, azért jött létre, mivel a klasszikus döntési fák könnyen túltanulnak az adathalmazon. Ez azt jelenti, hogy a modell csak a tanítóhalmazon teljesít jól, de a teszteken már rossz eredményeket hoz. A több, de sekélyebb döntési fa, amelyeknek nem csak a méretét, de a rendelkezésére álló változókat is leszűkítjük sikerebben általánosítanak az adathalmazon.



2.3. ábra. Véletlenszerű erdő regresszióhoz

Az itt látott + jel összegzést jelent, általában átlagolást alkalmaznak

Forrás: Medium

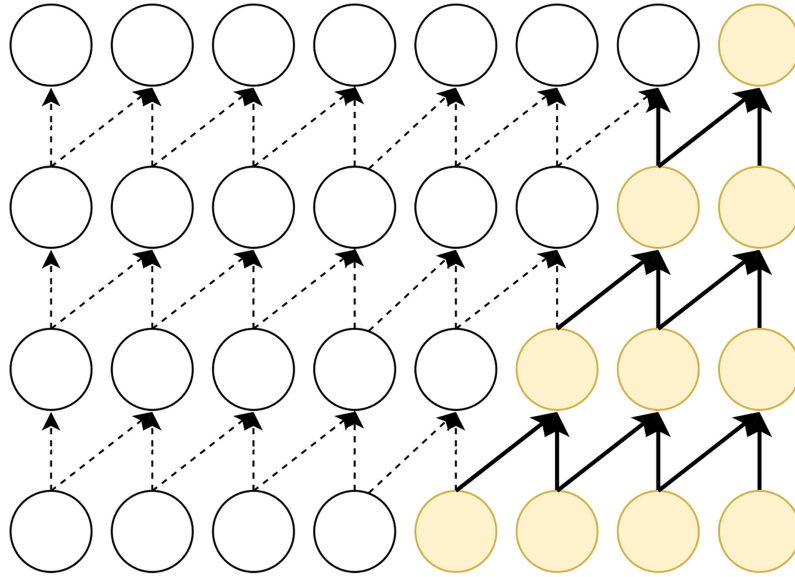
2.3.2. Konvolúciós háló

Az egyszerű többrétegű neurális hálók gyengesége az, hogy nem tudnak mintázatokat felismerni az input bármely részén, hiszen figyelembe veszik az input változók pontos elhelyezkedését. Ez előnyös akkor, ha kevés változónk van és azok sajátos jelentést hordoznak, viszont például képfeldolgozás esetén, ahol minden változó egyenértékű, rosszul teljesítenek. Ezen probléma megoldására jöttek létre a konvolúciós hálók, melyek az emberi látást egy részét próbálják utánozni. Legfontosabb tulajdonsága az ilyen architektúrának a transláció vagy eltolás invariancia, az input bármely részén fel tud ismerni mintázatokat.

[5] a konvolúciós háló konvolúciós, leskáázó és teljesen összekötött rétegekből áll. A konvolúciós réteg feladata a minta felismerése, illetve konfigurációtól függően a dimenzió csökkentése. A leskáázó (*pooling*) réteg feladata az adott területen legerősebb, vagy átlagos minta kiválasztása, ezzel csökkentve a dimenziót és növelve a konvolúciós rétegek "látóterét". A mintázatok felismerése és kiválasztása után teljesen összekötött rétegekkel ad predikciót a modell.

Az architektúra alapvetően képek elemzésére jött létre, ami legalább 2 dimenziós adatot jelent, valamint a színekre is ad általánosítást csatornákkal. A dolgozatban viszont

1 dimenziós adatra lesz alkalmazva az architektúra, az előrejelezni kívánt változó idő-soros adatára. Ezen megközelítést és változatait gyakran alkalmazzák idő-soros adatok előrejelzésére, általában a rövid távú előrejelzésekben teljesít jól, főleg olyan adathalmazokon, ahol nagyrészt az előrejelezni kívánt változó tartalmaz információt a modellnek. Fontos különbség, hogy a konvolúciós rétegeken alkalmazott padding itt csak egy irányból történik, ez az ún. causal convolution, arra szolgál, hogy lépésenként ne hozzunk be a rendszerbe időben előrettekintő érvénytelen adatot [6].



2.4. ábra. Causal convolution

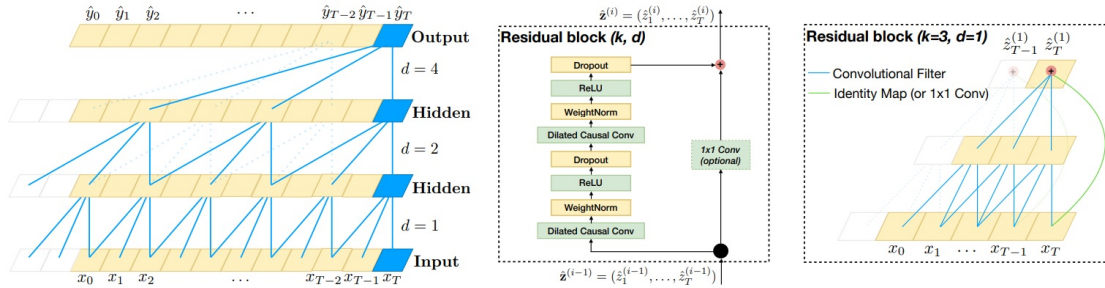
Forrás: [6]

2.3.3. Temporális konvolúciós háló

[7] felvázolja a konvolúciós hálók továbbfejlesztését. A konvolúciós hálónál már említett causal convolution módszerén túl alkalmaznak dilataciót is. Ekkor az egyes rétegek nem egymás mellől veszik az inputokat, mint a 2.4. ábrán, hanem ezek között egyenletes közöket hagyva, ez növeli a különböző rétegek látóterét. Fontos előrelépés a residual (megmaradó) blokkok alkalmazása, ahol egy adott blokk transzformációs kimenetét és bemenetét összegezzük, és így adjuk tovább a következő blokknak, vagy rétegnek. Ez segít enyhíteni az eltűnő gradiens problémáját és a tanítást is gyorsítani tudja, hiszen egy rétegnek nem a teljes transzformációt kell megtanulnia, hanem csak az identitás transzformáció módosítását.

A temporális konvolúciós háló ilyen residual blokkokból áll, ahol két rétegben alkalmaz dilataált konvolúciót, súly normalizációt, ReLU aktivációs függvényt és dropout

rétegeket. A dropout réteg egyes adatpontokat nullára cserél tanítás közben, ezzel kisebb részhálókat tanítva.



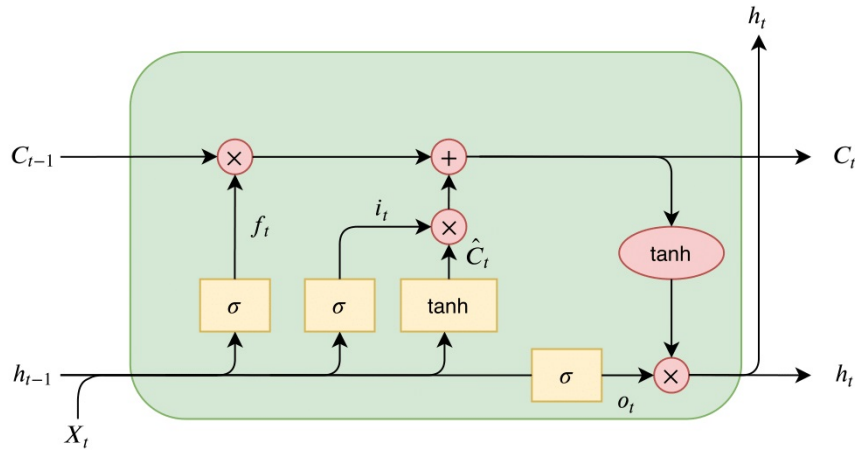
2.5. ábra. Temporális konvolúciós háló részei sorban:
Dilatált konvolúciós réteg, residual blokk, residual kapcsolatra példa
Forrás: [7]

A temporális konvolúciós hálók általában jobb eredményeket hoznak, mint az egyszerű konvolúciós hálók és bizonyos adathalmazokon a rekurrens hálók eredményeit is felülmúlják ([7] 6. oldal).

2.3.4. LSTM

A rekurrens hálóknak nagy előnye, hogy tetszőleges hosszúságú bemenetet tudnak feldolgozni úgy, hogy a rejtett (*hidden*) állapotukat lépésenként frissítik az új bemeneti adatok függvényében. Az egyszerű rekurrens neurális hálók az eltűnő gradiens problémával küzdenek, erre ad egy megoldást az LSTM.

[8] bemutatja az LSTM (*Long short-term memory*), avagy hosszú rövid távú memóriás mély tanulási modell architektúráját és alkalmazását idősoros problémákra. A rejtett állapoton túl egy ún. sejt (*cell*) állapotot is tárol és módosít a bemenet feldolgozása közben. 3 kapuval szabályozza a működést, hogy az állapotok megfelelően frissüljenek. A bemeneti (*input*) kapu felel a sejt állapot új információval történő frissítéséért. A felejtő (*forget*) kapu dobja el a már nem releváns információkat. Ez a 2 kapu adja meg az új sejt állapotot. A kimeneti (*output*) kapu a bemenetből kiszámolt értékek, a rejtett állapot és a sejt állapot függvényében határozza meg az új rejtett állapotot. Az LSTM cella architektúráját a 2.6. ábra mutatja be.

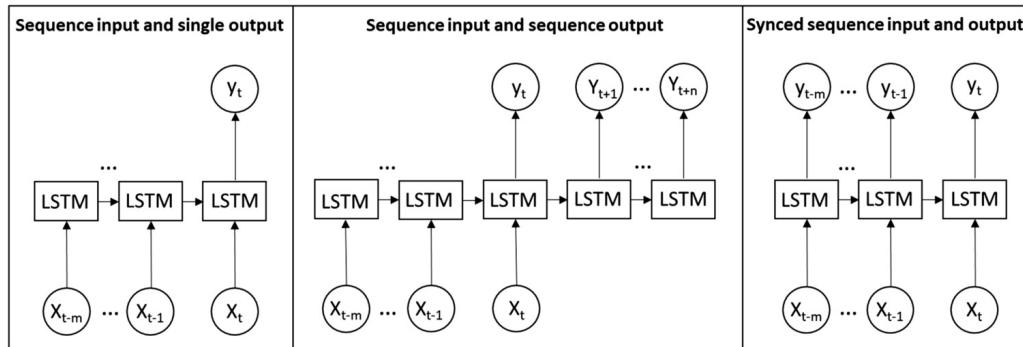


2.6. ábra. LSTM cella, h a rejtett, C a cella állapotot, X a bementi adatot jelzi megfelelő t időpillanatban

f_t a felejtő, i_t a bemeneti, o_t a kimeneti kapu

Forrás: Thorir Mar Ingolfsson

Az említett cellákból építhető fel az LSTM modell, ahol több rétegben és több irányban alkalmazhatóak a cellák, ezzel lehetővé téve sokféle szekvencia feldolgozását, értelmezését. Alkalmazhatjuk egy szekvencia felismerésre és az utolsó kimenetből jelezhetünk előre például egy teljesen összekötött réteggel 1 vagy annál több lépésre. Alkalmazható enkóder-dekóder architektúrában (2.4.3.). Alkalmazható úgy is, hogy a bemeneti szekvenciából azonnal generáljuk a kimenetet minden lépésben.



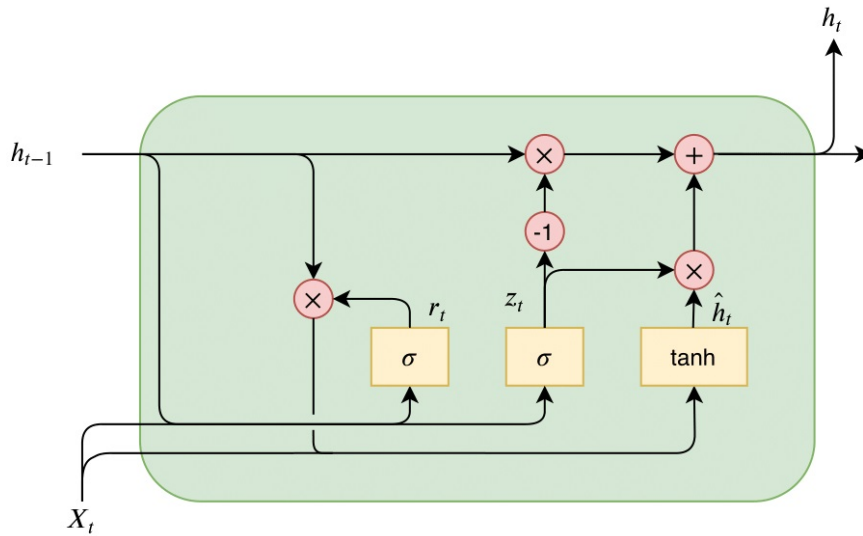
2.7. ábra. LSTM használatának módjai szekvencia feldolgozására

Forrás: [9]

2.3.5. GRU

A GRU (*Gated recurrent unit*), avagy kapus rekurrens egység az LSTM-hez hasonló architektúrát valósít meg. [10] írja le működését, az LSTM-el szemben a GRU nem alkalmaz sejt állapotot, hanem csak rejtett állapottal dolgozik. A 3 kapu helyett 2 szerepel, a frissítő (*update*) kapu adja meg, hogy a beérkező információt figyelembe kell-e venni.

A második a reset kapu megadja, hogy a korábbi rejtett állapotot milyen mértékben kell figyelembe venni. A GRU cell architektúrája a 2.8. ábrán látható.



2.8. ábra. GRU cella, h a rejtett állapotot, X a bementi adatot jelzi megfelelő t időpillanatban

r_t a reset, z_t a frissítő kapu

Forrás: Thorir Mar Ingolfsson

A GRU cellákból hasonlóan építhetők modellek, mint az LSTM cellákból. A két modell magasszintű működését tekintve lényegében ekvivalens, viszont teljesítményüket tekintve általánosságban nem lehet kimondani melyik teljesít jobban. Alkalmazások és adatok függvényében lehet eldönteni, hogy az adott problémára melyik működik eredményesebben.

2.4. Előrejelzési stratégiák

Ebben a részben tárgyaljuk az idősoros adaton végzett előrejelzés különböző stratégiáit. A stratégiák és modellek kombinációja adja azokat a megközelítéseket, amivel a dolgozat hátralévő részében foglalkozom. Ebben a részben nem a modellekre fókuszálunk, tehát a θ paramétervektor nem jelenik meg, f legyen egy előrejelző függvény.

2.4.1. Rekurzív stratégia

Recursive, avagy rekurzív stratégia egy egymodelles stratégia, melyben az előrejelző modell a következő időpillanat előrejelzésére van betanítva [11]. Egy időpillanat előre-

jelzése mindig egyszerűbb, mint egy lépésben egy hosszabb előrejelzés adása, valamint modellt is könnyebb, gyorsabb ilyen feladatra tanítani.

1. Algoritmus Rekurzív stratégia

```

1: for  $i = 0, \dots, n_O - 1$  do
2:    $\hat{y}_t[i] := f(x_t)$  ▷ előrejelzés egy lépésre
3:    $x_t := x_t[1..n_T]$  ▷ ablak csúsztatása, így 1-el rövidebb lesz
4:    $x_t[n_T] := \hat{y}_t[i]$  ▷ ablak bővítése
5: end for
6: return  $\hat{y}_t$  ▷ ebben a külső változók előrejelzése is szerepel

```

Tehát, minden t időpillanatban előállítjuk a $t + 1$ időpillanatra adott előrejelzést, aztán ezt igazságnak véve csúsztatjuk a csúszó ablakot egy lépéssel és $t + 2$ -re is előrejelzést adunk. Ezt iterálva kapjuk meg a teljes előrejelzett vektort. Hátránya, hogy a modellnek a kimeneti változón túl a külső változókat is előre kell jeleznie, hogy tovább tudjunk lépni. Ez sok modell optimalizációs algoritmusban hátrányt jelent, mert olyan változók előrejelzésére is egyenlően optimalizál, amiket nem akarunk használni a végső eredményhez. További hátrány, hogy az ilyen előrejelzésekben a halmozódó hiba eltérítheti a későbbi időpillanatokban adott előrejelzéseket.

2.4.2. MIMO stratégia

MIMO (*Multi-Input Multi-Output*), avagy a több-bemenet több-kimenet stratégia egy egymodelles stratégia. [11] bemutatja a Közvetlen (*Direct*) stratégia működését, mely sokban hasonlít a MIMO-ra, viszont az előbbi több modellt alkalmaz.

2. Algoritmus Közvetlen stratégia

```

1: for  $i = 0, \dots, n_O - 1$  do
2:    $\hat{y}_t[i] := f_i(x_t)$  ▷  $f_i$  az egyes modelleket jelenti
3: end for
4: return  $\hat{y}_t$ 

```

Tehát minden előrejelezett időpillanathoz egy külön modellt kell betanítani és alkalmazni. Ennek továbbfejlesztése a MIMO stratégia, ahol egyetlen modell jelzi előre az összes időpontra a kimeneti változót egy vektorban. Tehát a MIMO stratégia úgy szól, hogy

$$\hat{y}_t = f(x_t), \text{ egyszerű több kimenetes előrejelzés} \quad (2.6)$$

[11] bemutatja a DIRMO (DIRect, miMO), avagy több-kimenetes direkt stratégiát is. Annyiban tér el a direkt stratégiától, hogy az egyes modellek a kimenetvektor egy 1-nél hosszabb szeletét jelzik előre, ebben mutatkozik meg a MIMO jelleg. A dolgozatban a MIMO lesz alkalmazva, hiszen az említett könyvben is ez a stratégia teljesített általánosan jól.

2.4.3. Seq2seq stratégia

Seq2seq, avagy szekvenciából szekvencia stratégia azért jött létre, mert a rekurrens hálók önmagukban nem tudnak tetszőleges hosszú kimenetet adni. Az enkóder-dekóder vagy transzformer modellek nagy népszerűségnek örvendenek a természetes nyelvfeldolgozás területén. Idősoros adatokra is általánosíthatók. A stratégia valójában egy modellként valósul meg, amit a MIMO-hoz hasonlóan alkalmazunk.

Két rekurrens modell szerepel benne, például LSTM vagy GRU, de az enkóder és dekóder modell általában azonos típusú. Az enkóder feladata a kapott bemeneti adatok kódolása egy rejtett állapotra. A dekóder feladata az utolsó bemenetből és a rejtett állapotból lépésenként a kimenet előállítás. Ez tarthat egy bizonyos iterációszámig, vagy egy szekvencia vége jelig, karakterig. [12] LSTM modelleket alkalmaz, én a feladatra GRU modelleket választottam.

3. Algoritmus Seq2seq stratégia

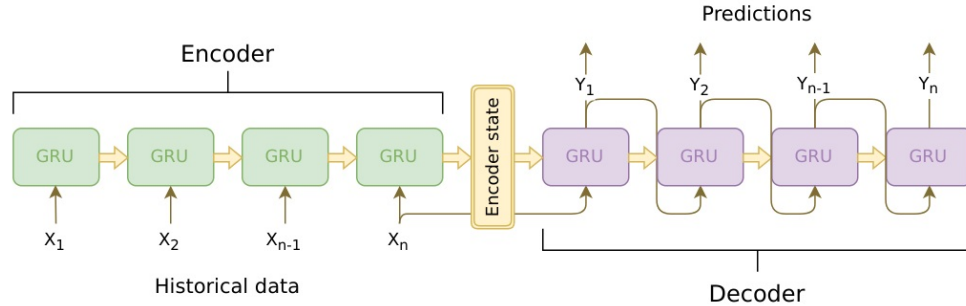
```

1:  $h_t := f_{enc}(x_t, h_0)$  ▷ az enkóderből csak a rejtett állapotra van szükség
2:  $\hat{y}_t[-1] := SELECT(x_t[n_T])$  ▷ csak a kimeneti változónak megfelelő adatot vesszük át
3: for  $i = 0, \dots, n_O - 1$  do
4:    $\hat{y}_t[i] := f_{dec}(\hat{y}_t, h_t)$  ▷  $y_t$  lépésenként bővül
5:   if  $random(0 \dots 1) < teacher\_forcing$  then ▷ teacher forcing
6:      $\hat{y}_t[i] := y_t[i]$ 
7:   end if
8: end for
9: return  $\hat{y}_t[0 \dots]$ 

```

A 2.9. ábra bemutatja szemléletesen egy GRU enkóder-dekóder modell működését idősoros esetben. Hasonló modell építhető két LSTM részből is, ott viszont a rejtett állapoton túl a sejt állapotot is az enkóder hozza létre, a dekóder pedig használja. Fontos itt megemlíteni a teacher-forcing (tanár-kényszerítés) módszerét, ahol tanítás közben a dekóder által előrejelzett adatokat lecseréljük bizonyos, a tanítás során csökkenő eséllyel, a tényleges várt kimenetre. Ez abban segít, hogy a modell egyenletesebben tanuljon hosszabb előrejelzésekre is, hiszen így kevesebb hibát viszünk be a tanítás korai sza-

kaszában [13]. Fontos megjegyezni viszont, hogy az alkalmazott teacher-forcing csak a következő predikcióban számít, az optimalizációt a ténylegesen előrejelzett értékekből számoljuk. Működése a 3. algoritmus 5-6. sorában látható.



2.9. ábra. Seq2seq GRU modell

Forrás: Github - sooftware

2.4.4. Többmodelles rekurzív stratégia

A dolgozatban új stratégiaként a többmodelles rekurzív stratégiát dolgozom ki (*Multi Model Recursion - MMRec*). A rekurzív stratégia nagy előnye, hogy könnyen tanítható, mert csak 1 lépést jelez előre, valamint tetszőleges hosszúságú előrejelzéseket is tud adni. Az első változtatás, hogy az idősorban megjelenő, könnyen számolható változókat ne a modell jelezze előre, például az időpontot és a késleltetett változókat, mint például az elektromos áramfogyasztás 24 órával korábbi értékét könnyen számolhatjuk képletből.

A következő változtatás, hogy annyi modellt hozok létre amennyi változót előre kell jelezni. Például 2 időjárési és 1 elektromos fogyasztási változó esetén 3 modellt alkalmazok. Intuitívan, az elektromos fogyasztás és a csapadék előrejelzéséhez más változókra kell vélhetően a modellnek a hangsúlyt fektetni. Emellett, ha minden modell a saját változóját jelzi előre, akkor az optimalizáció során csak azon lesz a hangsúly, így vélhetően egyenként hatékonyabb modellek keletkeznek.

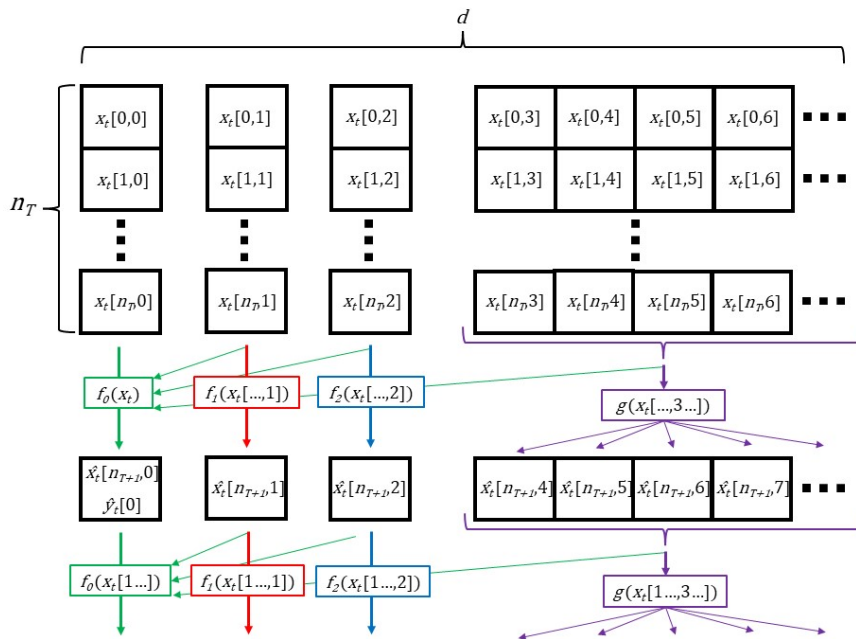
Intuitívan, a stratégia célja az, hogy általánosságban idősorok esetén is hasznos, ha például rendelkezésre áll a külső változókra valamilyen előrejelzés, például időjárás előrejelzés. A stratégia ezt próbálja helyettesíteni azzal, hogy saját magának ad ilyesféle előrejelzéseket azért, hogy csökkentse a halmozódó hiba problémáját a későbbi predikciókban.

4. Algoritmus Többmodelles rekurzív stratégia

```

1: for  $i = 0 \dots n_O - 1$  do
2:   for  $j = 0 \dots k - 1$  do                                ▷  $k$  az előrejelzett változók száma
3:      $\hat{y}_t[i, j] := f_j(x_t)$                                 ▷ adott változó előrejelzése
4:      $x_t := x_t[1..n_T]$                                     ▷ ablak csúsztatása, így 1-el rövidebb lesz
5:      $x_t[n_T] := g(x_t)$                                     ▷  $g$  függvény adja meg az egyértelmű változókat
6:      $x_t[n_T, j] := \hat{y}_t[i, j]$                             ▷ az előrejelzett változók behelyettesítése
7:     if  $\text{random}(0 \dots 1) < \text{teacher\_forcing}$  then      ▷ teacher-forcing
8:        $x_t[n_T, j] := y_t[i, j]$ 
9:     end if
10:  end for
11: end for
12: return  $\hat{y}_t$                                             ▷ vagy megfelelő részének visszaadása
    
```

A dolgozatban úgy határozom meg a stratégiát, hogy a különböző modelleket párhuzamosan lehessen tanítani. Tanításkor felhasználom azt, hogy amennyiben egy modell felhasznál olyan változót, amit egy másik jelez előre, akkor tanításkor vigyünk be a rendszerbe az utóbbi hibáját is, hogy a tanított modell eredményesebben működjön a feladaton. Itt alkalmazok a teacher-forcing-hoz hasonló technikát, amit a 2.4.3. részben részleteztem. A rekurzív predikciók esetében úgy működik, hogy bizonyos előrejelzéseket lecserélek a hozzájuk tartozó várt értékre, ezzel segítve a tanítási folyamatot. Itt is az optimalizációt a tényleges előrejelzésből számoljuk, nem a behelyettesített tökéletes értékből. Működése a 4. algoritmus 7-8. sorában látható.



2.10. ábra. Többmodelles rekurzió

f_k a különböző modellek, g a képlet, ami a könnyen számolható változókat adja meg

3. fejezet

Módszertan

3.1. Fejlesztői környezet

A feladat megvalósításához Python (3.11.5) nyelvet és Jupyter Notebook-okat is használtam. Főbb könyvtárak, amiket alkalmaztam:

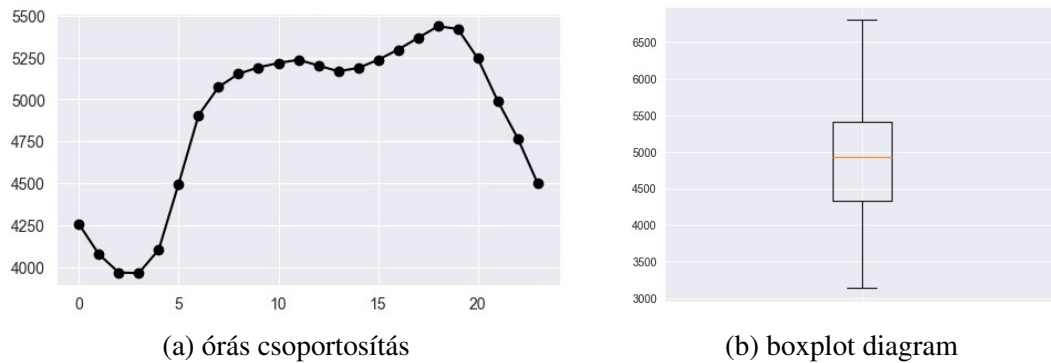
- Numpy: alapvető, gyors tömbműveleteket biztosító, népszerű könyvtár
- Pandas: adatkezelés, -betöltés, -mentés, táblázatkezelés
- Matplotlib: grafikonok készítése, minden megjelenő grafikont ezzel készítettem
- Scikit-learn: gépi-tanulási könyvtár
- Pytorch: mély tanulási könyvtár, a neurális hálós modellek ebben készültek

3.2. Felhasznált adatok

Az elektromos fogyasztási előrejelzéshez Magyarországi adatokat használtam. Az időjárási adatokat az OMSZ (*Országos Meteorológiai Szolgálat*) Meteorológiai Adattár-ból töltöttem le. Az elektromos fogyasztási adatokat a MAVIR (*Magyar Villamosenergia-ipari Átviteli Rendszerirányító zrt.*) adatpublikációjából töltöttem le. Az adatok kezdési dátumának 2015 első napját választottam, mivel 2014-ben az OMSZ sok új állomást létesített és így azoknak az adatait is tudtam használni. Emellett a 2015-ös kezdéssel sok állomás adata áll rendelkezésre a mély tanuláshoz. Utolsó dátumnak 2023 szeptember elsejét választottam, így egy közel 9 év hosszú idősort kaptam.

3.2.1. MAVIR adatok

A MAVIR adatbázisban sokféle mező szerepel, a feladatomhoz a 'nettó terhelés' mezőt választottam, hiszen ehhez a mezőhöz tartozik még a 'MAVIR becslés' mező is. A nettó terhelés Megawattban (MW) szerepel. Röviden ismerkedjünk meg az adatok jellegével.



3.1. ábra. MAVIR által közzétett nettó terhelési adatok ábrázolása

A 3.1. (a) ábráján a Magyarországra 2015 januártól 2023 szeptemberig jellemző napi elektromos fogyasztási görbe látható. Az ilyen ábrák minden országra más tendenciát mutatnak általában. Esetünkben éjszaka éri el minimumát, majd dél körül következik az első tetőzés, utána pedig este 7-8 óra körül éri el maximumát. Biztosak lehetünk abban, hogy a választott modellnek erre a tendenciára rá kell tanulnia.

A 3.1. (b) ábráján a nettó terhelés box (doboz) diagramja látható. A két kiugró vízszintes vonal jelzi a minimumot és maximumot. A középső doboz felső és alsó határait az első és harmadik kvartilisek adják. A narancssárga vonal pedig az átlagot jelenti. Az adatok szórása 729.055, nagyon magas. Fontos itt megjegyezni, hogy az irreálisan kiugró adatok elkerülésére az adatokat a 0.001 és 0.999 kvantilisek közé szorítottam, a boxplot diagram már ezekből az adatokból készült.

Fontos még megemlíteni, hogy a MAVIR adatokban körülbelül 100 hiányzó adatpontot fedeztem fel, és mivel kevés adatról van szó a majdnem 9 éves, órás felosztású adathalmazban, így lineáris interpolációval töltöttem fel a hiányzó értékeket. A lineáris interpoláció egyenletes felosztású adaton a hiányzó adatot megelőző és követő adatok átlagából számolja az új értéket.

3.2.2. OMSZ adatok

Az OMSZ Meteorológiai Adattárában többféle felosztásban elérhetők időjárási adatok a különböző mérőállomásokról. A feladathoz órás felosztást választottam, és mivel 2015 első napját választottam a végső idősor kezdetének, ezért összesen 104 állomás adatát töltöttem le. A különböző mérőállomások más-más méréseket végeznek, így az első feladat a megfelelő változók kiválasztása. 3.1. táblázatban láthatóak a teljesség igénye nélkül az elérhető mérési adatok, kiválasztásuknál nagy szempont, hogy ne legyenek erősen korrelálók (például csak 1 féle hőmérséklet legyen), órás átlagokat vegyünk ahol lehet, és sok állomáson legyenek elérhetőek.

Jel-O	Jel-S	Leírás	Hány állomáson méri
<i>r</i>	<i>prec</i>	órás csapadékösszeg	103
<i>ta</i>	<i>temp</i>	az elmúlt óra átlaghőmérséklete	104
<i>u</i>	<i>rhum</i>	órás pillanatnyi relatív nedvesség	103
<i>sr</i>	<i>grad</i>	órás globálsugárzás összeg	34
<i>we</i>	-	pillanatnyi időkép kódja	11
<i>p0</i>	<i>pres</i>	órás pillanatnyi tengerszinti légnyomás	40
<i>f</i>	<i>wind</i>	órás átlagos szélsébség	101
<i>tviz</i>	<i>tviz</i>	órás pillanatnyi vízhőmérséklet	3

3.1. táblázat. OMSZ állomások elérhető adatai
Jel-O az OMSZ által használt, Jel-S a továbbiakban általam használt

A pillanatnyi időkép kódból (*we*), kevés elérhető érték van, ezért nem használhatóak előrejelzésre, valamint problémát jelentene, hogyan összegezhetőek országos szintre. A vízhőmérséklet 3 helyen elérhető:

- Győr-Moson-Sopron, Likócs: vélhetően a Mosoni-Duna hőmérséklete
- Fejér, Agárd: vélhetően a Velencei-tó hőmérséklete, tovább használok
- Somogy, Siófok: vélhetően a Balaton hőmérséklete, tovább használok

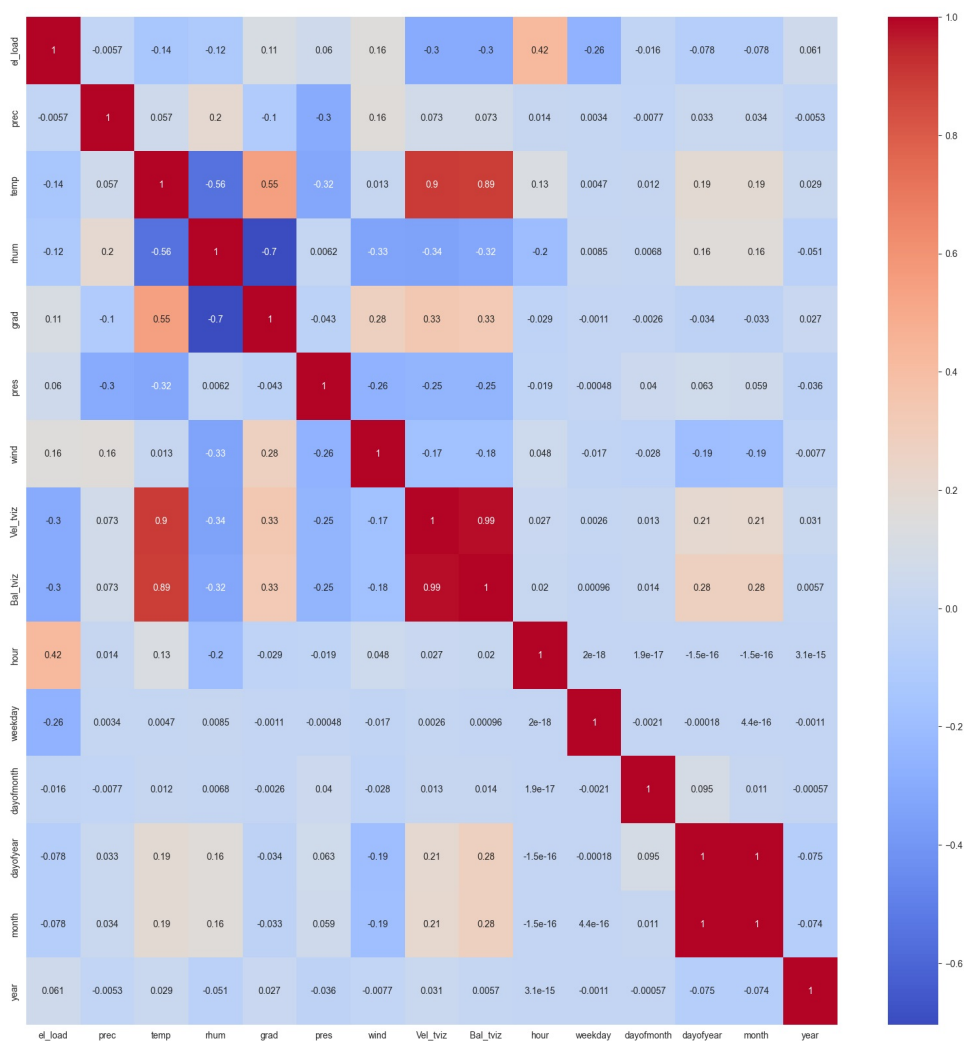
A megmaradt változók relatív sok helyen elérhetőek, ezért jelen állapotban tovább használok őket. Problémát jelentenek viszont a hiányzó értékek, nem minden állomás mér minden változót. A vízhőmérséklet problémája könnyen orvosolható, azokat mindeképp külön változóként kezelem országos szinten is. A többi változót úgy kezelem, hogy országosan átlagolom őket, ekkor a hiányzó értékek nem jelentenek problémát, mert egyszerűen, ha egy időpontban nem elérhető valahol az érték, akkor a többi felett átlagolva nem vesszük figyelembe a hiányzót. Ekkora állomás mennyiségnél ez vélhetően nem okoz gondot az adatot tekintve.

3.2.3. EDA

A következő lépés az EDA (Exploratory Data Analysis), avagy a "feltáró" adatelemzés. Legfőbb lépései az adatok vizualizálása, határértékek vizsgálata, korrelációk feltárása, esetleges változó kiválasztó algoritmusok alkalmazása.

Első lépés a kiválasztott MAVIR és OMSZ adatok egy adathalmazba szervezése, ami az időpontok tudatában egyszerű feladat. Mivel idősoros problémával dolgozok, a dátumból érdemes kinyerni a különböző, időpontot leíró változókat, és hétvégre, ünnepnapra vagy covid időszakra vonatkozó bináris változókat. A kinyert változók tehát: óra, hónap, év, hét napja, hónap napja, év napja, ünnepnap és hétvége jelzők, valamint covid jelző.

Az elektromos fogyasztási adatokkal már megismerkedtünk a 3.1. ábra segítségével. Az időjárás adatok kirajzolása előtt nézzük a korrelációs mátrixot, segítségével könnyen szűrhetők a lineárisan összefüggő változók.



3.2. ábra. Korrelációs mátrix

A vízhőmérsékletek és átlagos hőmérsékletek erősen korrelálnak, 1-et elég megtartani. A relatív páratartalom és globális sugárzás inverzen korrelál nagy mértékben. Az elektromos fogyasztás pedig legerősebben az órával korrelál, ez nem meglepő, hiszen a kettő kapcsolata produkálja a napi jellegzetes Magyarországi fogyasztási görbét.

Idősoros feladatoknál célszerű az előrejelzni kívánt változó mellett annak késleltetett (*lag*) megfelelőit is felvenni, én a 24, 48, 72 és 96 órás késleltetéseket választottam.

A végső változók kiválasztásához segítségül hívom a szekvenciális automata változó kiválasztó algoritmust, melyben egy választott gépi- vagy mély tanulási modellel 0 db változóból kiindulva vizsgáljuk, hogy milyen változókat emeljünk ki az eredetiekből, hogy a lehető legjobb előrejelzést adjuk. [14]

5. Algoritmus Automata változó választás

```

1:  $m := ALLVARIABLES$                                 ▷  $m$  tartalmazza az összes változót
2:  $bestvars := []$                                        ▷ itt tároljuk a választott változókat
3: for  $i = 0 \dots d - 1$  do
4:    $bestmetric = MAX$                                    ▷ lokálisan legjobb eredmény
5:    $localbestvar = NULL$                                 ▷ lokálisan legjobb változó
6:   for  $j$  in  $(m \setminus bestvars)$  do                 ▷ \ halmazművelet
7:      $newvars := bestvars + j$                          ▷ vegyük az új változót hozzá
8:      $metric := TRAINANDEVALUATE(f(s[\dots, newvars]))$  ▷ kiértékelés
9:     if  $metric < bestmetric$  then                     ▷ jobb?
10:       $bestmetric := metric$ 
11:       $localbestvar := j$ 
12:    end if
13:  end for
14:   $bestvars := bestvars + localbestvar$                  ▷ új változó hozzávétele
15:  yield  $i, bestvars, bestmetric$ 
16:  ▷ minden változószámmra adjuk vissza a legjobb talált megoldást
17: end for

```

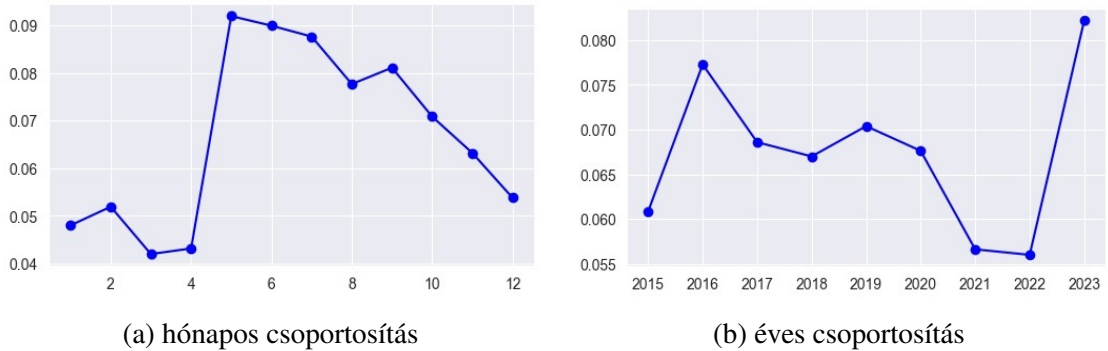
A dolgozatban az `mlxtend` Python könyvtár `SequentialFeatureSelector`-át alkalmaztam Véletlenszerű Erdővel, ahol $n_T = 24$ és $n_O = 3$. Az algoritmust 13 változóig futtattam, mert addigra már biztos volt, hogy további eredményjavulás nem fog bekövetkezni további változók esetén. A 11 választott változó:

- elektromos fogyasztás és 24 órás késleltetése
- csapadék és globális sugárzás
- ünnepnap és hétvége jelzők
- óra, hét napja, év napja, hónap, év

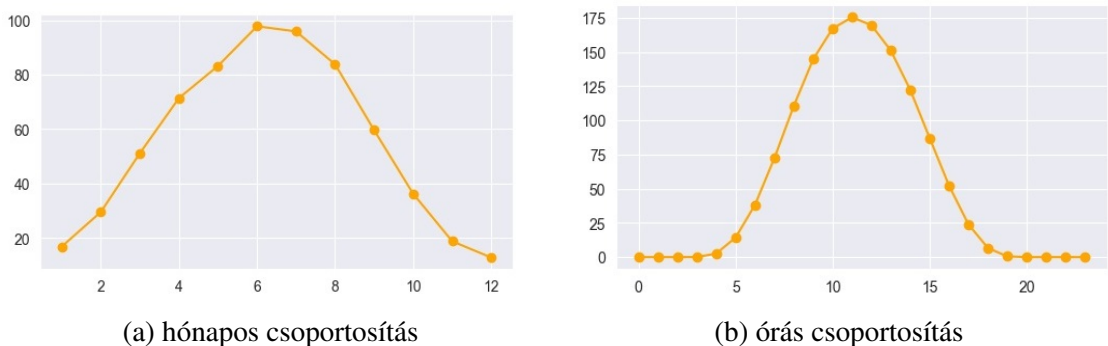
Meglepő módon a covid jelzőt nem választotta az algoritmus, de ez a későbbi kísérletek alapján megalapozottnak bizonyult. Futtattam teszteket rövidebb és hosszabb vissza-

tekintésre is, de $n_T = 12, 36, 48$ közül egyik sem adott jobb eredményeket.

A 11 választott változó 2 db időjárási jellemzőt tartalmaz, a csapadékot és a globális sugárzást. Ez biztató a többmodelles rekurzív algoritmus szempontjából, hiszen csak 3 változót kell majd előrejelezni, ami mindössze 3 modellt jelent. Nézzük is meg a 2 időjárási jellemző legbeszédesebb grafikonjait.



3.3. ábra. Csapadék grafikonok



3.4. ábra. Globális sugárzás grafikonok

A csapadék grafikon csökkenő tendenciát mutat évesen, a 2023-as adatok félrevezetőek lehetnek, mert nem tartalmazzák a szeptember-december időszakot. Hónaposan a csapadék magasabb a nyári időszakban. A globális sugárzási adatok nem túl meglepőek, a nyári időszakban magasabb, valamint éjszaka 0 és napközben délben tetőzik. Az inverz korreláció a páratartalommal azt sugallja, hogy a grafikonja ellentétes görbét mutatna. A korreláció a hőmérséklettel pedig nem meglepő, és a grafikont látva érthető miért nem választotta az automata változó kiválasztás a hőmérsékletet.

3.3. A kiválasztott megközelítések, optimalizációjuk

A kiválasztott megközelítéseket stratégia-modell párokba rendezem, ezek olyan egységek, amik önmagukban tudnak előrejelzést adni. Minden választott modellt optimalizáltam heurisztikus és Grid Search paraméterkereséssel. A Grid Search algoritmus az csinálja, hogy az átadott paraméter értékekből előállítja az összes lehetséges változatot. Ebből saját implementációt is készítettem, mely az Iterátor tervezési mintát követi és könnyen alkalmazható, mivel dictionary típusban adja vissza minden lépésben a soron következő paramétereket. Innen már egy maximum, vagy metrikák esetén minimumkiválasztással könnyen megtalálható a legjobb konfiguráció.

```

1 class Grid:
2     def __init__(self, grid: dict):
3         self._keys: list = list(grid.keys())
4         self._values: list = list(grid.values())
5         self._combinations: list = []
6
7     def __iter__(self):
8         # choose all combinations of hyperparameters without
9         # repetition
10        self._combinations = self._values[0]
11        if len(self._keys) > 1:
12            self._combinations = [[comb] + [item] for item in self._values[1] for comb in self._combinations]
13        if len(self._keys) > 2:
14            for ls in self._values[2:]:
15                self._combinations = [comb + [item] for item in ls for comb in self._combinations]
16
17        return self
18
19    def __next__(self):
20        if len(self._combinations) > 0:
21            return dict(zip(self._keys, self._combinations.pop(0)))
22        else:
23            raise StopIteration()

```

3.1. forráskód. Grid Search Iterátor

Ezeket a kombinációkat egyesével kiértékelve, a modellt betanítva és az RMSE metrikát kiértékelve kapjuk a legjobb paramétervektort és teljesítményét. Kisebb Grid-eken

keresve gyorsabban találhatunk az egyes paraméterekre jó értékeket. Ezen értékeket a végeredményeknél (4.1) fogom feltüntetni, mert a random inicializáció miatt több teljes tanítási és kiértékelési folyamatot kell futtatni.

Az adatra a mély tanulási modellek esetében sztenderdizálást alkalmaztam, ami azt jelenti, hogy változónként az adatokat 0 átlagra és 1 szórásra hozzuk az alábbi képlettel. A tanítóhalmazon számoljuk az átlagot és szórást, majd a validációs és teszhalmazokat is ezekkel az értékekkel sztenderdizáljuk.

$$X = \frac{X - \mu}{\sigma} \quad (3.1)$$

A végeredményeknél (4.1) a metrikák már visszaalakítás után lettek számolva, tehát a modell sztenderdizált adaton dolgozik, prediktál, viszont utána ezeket a predikciókat desztenderdizálja és így adja meg az eredményeket.

3.3.1. MIMO, Véletlenszerű Erdő Regresszor

A Véletlenszerű Erdő paraméterkeresése egyszerűbb feladat a mély tanulási modellekkel szemben, mivel itt egyértelmű a modell konfiguráció, csupán a paramétereit kellett megtalálni. A legfontosabb paramétere egy Véletlenszerű Erdő Regresszornak:

- fák mennyisége: hány különböző fát épít
- max mélység: egy fa maximálisan milyen mély lehet
- minimális vágás: hány változót kell minimálisan megnézni egy vágáshoz
- maximális változók aránya: egy fa a bemeneti változók mekkora részét használhatja

Az említett paramétereken végeztem a keresést, a következő táblázatban feltüntettem, hogy melyik paraméterhez milyen értékeket próbáltam ki. Itt nem feltétlen minden létező kombinációt néztem meg, csak azokat amelyek irányában az eredmények javultak.

Paraméter	Értékei	Jegyzet
<i>fák mennyisége</i>	50, 100, <u>150</u> , 300	túl sok felesleges
<i>max mélység</i>	5, 15, 25, <u>50</u>	2 ^x vágást csinálhat egy fa
<i>minimális vágás</i>	<u>2</u> , 5	nincs nagy hatása itt
<i>maximális változók aránya</i>	$\log_2(n)$, $\sqrt{n}$, 50%, <u>75%</u>	

3.2. táblázat. Véletlenszerű erdő paramétere

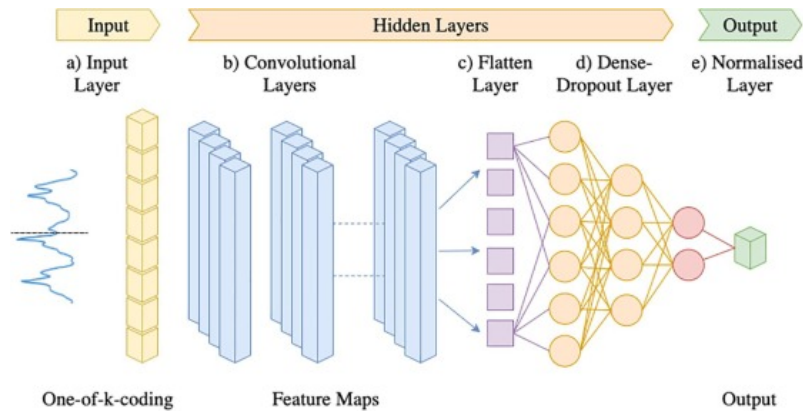
A paraméterkeresés eredményeként szolgáló paramétereket a 3.2. táblázatban aláhúztam. A legjobb eredményeket úgy kaptam, hogy a különböző fákat nem vettem túl

sekélyre, így egy-egy fa túltanulhatott bizonyos feladatokra, de összességében így jobb eredményekkel szolgált a rendszer.

A scikit-learn Python könyvtár implementációját használtam a Véletlenszerű Erdő Regresszor kiértékeléséhez. A könyvtár ad saját implementációt a Grid Search elvégzéséhez és az adatok idősoros felosztására keresztvalidációhoz is, ahogy a 2.2. ábra jobb oldala mutatja. Az adatfelosztási megoldást a továbbiakban is alkalmazom, és a paraméterkereséshez 6-ra választottam a vágások számát, a végeredmény kiértékelésekor viszont ezt nagyobbra veszem, hogy átfogóbb eredményeket mutasson.

3.3.2. MIMO, Konvolúciós Háló

A Konvolúciós Neurális Háló paraméterkeresése egy nehezebb feladat, mivel itt nem csak a különböző paramétereket kell jól meghatározni, hanem a modell felépítését is. Tehát fontos, hogy hány konvolúciós és teljesen összekötött rétegből áll. Előny viszont, hogy ez az egyik leggyorsabban tanítható modell, mivel rendelkezésemre áll CUDA GPU, amit modellekhez könnyedén lehet használni a PyTorch Python könyvtárral. A GPU-k jelentősen gyorsítják a mátrix műveleteket, ami neurális hálóknak esetén nagy előny.



3.5. ábra. Konvolúciós háló architektúra

Forrás: [15]

A konvolúciós hálóknak esetén csak az elektromos fogyasztást használjuk, a külső változókat nem. A modell architektúrákból csak azt a kettőt tüntetem fel, melyek jelentősen jobb eredményeket hoztak az előttem lévőkénél. Azt figyeltem meg, hogy a nagyobb "látótávval" rendelkező rétegek jobban teljesítenek a klasszikus, képfeldolgozásban is használt kisebb méretű filtereknél. A rétegek között ReLU aktivációs függvényt alkalmaztam

Konvolúciós réteg	Konvolúciós réteg	Konvolúciós réteg	Lineáris réteg
1→32 csatorna 10 bal padding 14-es kernel méret maxpool 2	32→64 csatorna 6 bal padding 8-as kernel méret maxpool 2	64→128 csatorna 1 bal padding 5-ös kernel méret maxpool 2	128→3 érték 50% dropout

3.3. táblázat. $n_T = 24$ Konvolúciós háló struktúrája

Konvolúciós réteg	Konvolúciós réteg	Lineáris réteg	Lineáris réteg
1→64 csatorna 6 bal padding 12-es kernel méret maxpool 2	64→256 csatorna 8 bal padding 16-os kernel méret maxpool 2	$7 \times 256 \rightarrow 256$ érték 0.02 zaj 50% dropout	256→3 érték

3.4. táblázat. $n_T = 48$ Konvolúciós háló struktúrája

Ha a modellnek a 24 hosszú szekvencia helyett 48 órás adatot adtam, akkor jobban teljesített. Ez nem meglepő, mivel a konvolúciós hálónál csak a prediktálni kívánt változót használok, ezért kevés adatot kap egy-egy predikcióhoz és a hosszabb visszatekintés pozitívan hat, nem növeli a dimenziószámot túlságosan.

Az egyszerű konvolúciós modell meglepően rossz eredményeket hozott paraméterkeresés közben, ezért is választottam a Temporális Konvolúciós Hálót emellett, mint Konvolúciós architektúra a dolgozatban.

3.3.3. MIMO, Temporális Konvolúciós Háló

A Temporális Konvolúciós Háló sokban hasonlít a Konvolúciós Hálóra, paraméterkeresése viszont egyszerűbb, hiszen az architektúra egyértelműbb. A 2.5. ábrán láthattuk a részeit. A reziduális blokkokból építve kapjuk a hálót. Tehát csak a blokkok számát és azok paramétereit kellett meghatározni.

Az implementációt a [7] publikációhoz tartozó GitHub repository-ból és az ott található Chomp megoldást ZeroPad2d-vel helyettesítő Kaggle implementációból vettem át.

```

1 class TCN(nn.Module):
2     def __init__(self, seq_len=24, pred_len=3, num_channels=(24,) *
      2, kernel_size=3, dropout=0.5, hid_noise=0.0, **kwargs):
3         super(TCN, self).__init__()
4         self.seq_len = seq_len
5         self.pred_len = pred_len

```

```

6         self.tcn = tmd.TemporalConvNet(1, num_channels, kernel_size
          =kernel_size, dropout=dropout)
7         self.hid_noise = tmd.GaussianNoise(hid_noise)
8         self.fc = nn.Linear(num_channels[-1], pred_len)
9
10        def forward(self, x):
11            x = x.reshape(-1, 1, self.seq_len)
12            x = self.tcn(x)
13            x = self.hid_noise(x)
14            return self.fc(x[:, :, -1])

```

3.2. forráskód. Temporális Konvolúciós Háló

A paraméterkeresést tehát a TCN osztályon kellett végezni, paramétereit sorban:

- szekvencia hossz (*seq_len*): bemenet hossza
- csatornák (*num_channels*): egyes rétegek csatornáinak száma és mérete, az egyszerűbb jelölés érdekében például 4-szer 72 csatornát így jelölök: 4×72
- kernel méret (*kernel_size*): konvolúció mérete
- dropout: adatnullázási arány tanításkor
- zaj (*hid_noise*): teljesen összekötött réteg előtti Gauss zajosítás

Paraméter	Értékei
<i>szekvencia hossz</i>	24, <u>48</u>
<i>csatornák</i>	3×48 <u>4</u> 5×20 50 <u>72</u> 100 (24, 48, 72, 96)
<i>kernel méret</i>	3, <u>5</u> , 7, 9
<i>dropout</i>	<u>30%</u> , 50%
<i>zaj</i>	<u>0</u> , 0.05, 0.1

3.5. táblázat. Temporális Konvolúciós Háló paramétereit

Az 5 és annál több rétegű konfigurációk már nem teljesítettek jól, valamint az egyenlő csatornaszámú rétegek jobban teljesítettek, mint ahol változott a csatornák száma. A Temporális Konvolúciós Hálónál nagy előnyt jelentett, ha $n_T = 48$, hiszen külső változók hiányában több adatra volt szüksége. Kipróbáltam azt az esetet is, amikor a bemenetet zajosítom, de valós adat révén a mérési hibák, lebegőpontos számítások, állomások hiányzó adatai annyi hibát hoznak a rendszerbe, hogy az extra Gauss zaj már nagyon félrevezeti a modellt. A legjobb paramétereket a 3.5. táblázatban aláhúzással jelöltem.

3.3.4. MIMO, LSTM

Az LSTM modellt egyszerűbb optimalizálni olyan szempontból, hogy a konfiguráció itt is adott, mint a Temporális Konvolúciós Hálónál. A MIMO stratégia révén, itt a 2.7. ábrán szereplő "szekvenciális input és egy kimenet" megoldásban az egy kimenetét az LSTM-nek átfuttatom egy teljesen összekötött rétegen, hogy a megfelelő 3 hosszú kimenetvektort adjon. Az LSTM réteg implementációját a PyTorch Python könyvtár adja. A konkrét implementáció sokban hasonlít a 3.2. forráskódra. Az optimalizálandó paraméterek tehát:

- LSTM rejtett réteg mérete: az LSTM cellák rejtett és sejt mérete
- rétegek száma: az LSTM megfelelő input feldolgozása után több LSTM cellán haladhatnak át egy inputra
- kétirányúság: az LSTM modellen mindkét irányban végigfuttatjuk az adatot, ekkor az inputot 2-szer dolgozzuk fel
- dropout: adatnullázási arány tanításkor (LSTM rétegek között is)
- rejtett zaj: teljesen összekötött réteg előtti Gauss zajosítás

Paraméter	Értékei
<i>rejtett réteg mérete</i>	10, 15, <u>20</u> , 25, 30
<i>rétegek száma</i>	1, 2, <u>3</u>
<i>kétirányúság</i>	Igen, Nem
<i>dropout</i>	<u>30%</u> , 50%
<i>zaj</i>	0, <u>0.05</u> , 0.1

3.6. táblázat. LSTM Háló paraméterei

Az egyirányú hálónál mindig jobban működtek a kétirányúak, még akkor is, ha az egyirányúnak több paramétert adtam. A modell nagyon jól betanulta a tanítóhalmazt, túl-tanulásra adott jeleket, de magasabb dropout és zaj mellett rosszabbul teljesített. A legjobb paramétereket a 3.6. táblázatban aláhúzással jelöltem.

3.3.5. Seq2seq, GRU

A seq2seq stratégia megvalósításához egy GRU enkóder-dekóder alapú modellt választottam. Az LSTM és GRU modellek általában nagyon hasonló eredményeket adnak, viszont a GRU modellek egyszerűbb architektúrája vélhetően gyorsabb tanítást fog eredményezni. Emellett feltűnt, hogy az adatfelosztáskor az LSTM modellek az első adatrészen nagyon nehezen taníthatók jól, ez a GRU-nál kevésbé jelentkezik.

```

1 class Seq2seq(nn.Module):
2     def __init__(self, features=11, pred_len=3, embedding_size=64,
3         num_layers=1, bidirectional=False, dropout=0.2, in_noise
4         =0.0, out_noise=0.0, **kwargs):
5         super(Seq2seq, self).__init__()
6         self.pred_len = pred_len
7         self.features = features
8         self.embedding_size = embedding_size
9         self.num_layers = num_layers
10        self.enc = GRUEncoder(features, embedding_size, num_layers,
11            bidirectional=bidirectional, dropout=dropout if
12            num_layers > 1 else 0.0, noise=in_noise)
13        self.dec = GRUDecoder(1, embedding_size, num_layers,
14            bidirectional=bidirectional, dropout=dropout if
15            num_layers > 1 else 0.0, noise=out_noise)
16
17    def forward(self, x, y=None, teacher_forcing=0.0):
18        batch_size = x.shape[0]
19        hidden = self.enc(x)
20        dec_input = x[:, -1, 0].reshape(-1, 1, 1)
21        output = torch.zeros(batch_size, self.pred_len).to(
22            MODEL_DEFINITION_DEVICE)
23
24        for i in range(self.pred_len):
25            out, _, _ = self.dec(dec_input, hidden)
26            output[:, i] = out[:, 0]
27            if y is not None and torch.rand(1) < teacher_forcing:
28                dec_input = torch.cat((dec_input, y[:, i].reshape
29                    (-1, 1, 1)), dim=1)
30            else:
31                dec_input = torch.cat((dec_input, out.unsqueeze(1))
32                    , dim=1)
33
34        return output

```

3.3. forráskód. GRU enkóder-dekóder

A GRU enkóder-dekóder megvalósításából csak a Seq2seq osztályt tüntetem fel, a GRUEncoder és GRUDecoder osztályok megvalósítása egyszerű, egymás utáni rétegek definíciójából és alkalmazásukból áll. A paraméteroptimalizációt a Seq2seq osztályon

kellett végezni, melynek paraméterei a következők:

- rejtett reprezentáció mérete (*embedding_size*): a kódolt vektor mérete
- rétegek száma (*num_layers*): LSTM-hez hasonlóan GRU rétegek száma
- kétirányúság (*bidirectional*): ugyanaz, mint az LSTM-nél
- dropout, be- és kimeneti zaj

Paraméter	Értékei
<i>rejtett reprezentáció mérete</i>	<u>10</u> , 12, 18, 20, 24, 64
<i>rétegek száma</i>	<u>1</u> , 2
<i>kétirányúság</i>	<u>Igen</u> , Nem
<i>dropout</i>	<u>50%</u>
<i>kimeneti zaj</i>	0, <u>0.05</u>

3.7. táblázat. GRU seq2seq modell paraméterei

A bemeneti zaj ebben az esetben is nagyon eltéríti a modellt, nem a túltanulást gátolja, hanem rontja a végső modellt. A MIMO, LSTM után meglepő módon ez a modell kis rejtett reprezentációval és 1 rétegű megközelítéssel teljesített a legjobban. A megfelelő paramétereket a 3.7. táblázatban aláhúzással jelöltem.

3.3.6. Rekurzív stratégia, GRU és LSTM

A rekurzív stratégia paraméterkeresése során párhuzamosan teszteltem a GRU és LSTM modelleket is. Implementációban hasonlítanak a 3.2. forráskódban lévő modellre, a legnagyobb változtatás az, hogy a modellek esetében egy 11 hosszú vektort jeleznek előre, ami a következő időpillanatot jelenti. A keresett paraméterek megegyeznek a 3.3.4. listában tárgyalt résszel.

A stratégiában tanításkor teacher-forcing-ot alkalmaztam, tehát egyes lépésekben a modell által előrejelzett vektort az eredetire cseréltem.

Paraméter	Értékei
<i>modell</i>	<u>GRU</u> , LSTM
<i>rejtett réteg mérete</i>	20, 40, 50, 60, <u>70</u> , 80
<i>rétegek száma</i>	1, 2, 3, 4, <u>5</u>
<i>kétirányúság</i>	<u>Igen</u>
<i>dropout</i>	<u>50%</u>
<i>zaj</i>	0, <u>0.05</u>

3.8. táblázat. Rekurzív stratégia paraméterei

Az optimalizácónál felhasználtam az LSTM hálónál kapott eredményeket, ezért nincs feltüntetve az egyirányú modell, vagy a különböző dropout értékek. Mivel a hálónak most sokkal több értéket kell előrejeleznie, ezért a rejtett méret és rétegszám is nagyobb az eddig látottaknál. A legjobb talált paramétereket a 3.8. táblázatban aláhúzással jelöltem.

3.3.7. Többmodelles rekurzív stratégia

A többmodelles rekurzív stratégiában az eddigiekhez képest itt több, különböző architektúrával rendelkező modellt kell optimalizálni és kiértékelni. A 2 db időjárási jellemzőt olyan modellekkel jelzem előre, melyek nem használnak külső változókat, ezeket az egyszerű rekurzív stratégia algoritmusával lehet betanítani és kiértékelni. Erre a feladatra a konvolúciós, temporális konvolúciós, GRU és LSTM hálót teszteltem le, nézzük is a paramétereiket. A Konvolúciós hálók esetén 2 konvolúciós és 1 teljesen összekötött réteget alkalmaztam.

CNN		TCN		GRU, LSTM	
<i>kernel méretek</i>	(12, 6) (10, 10) (6, 12)	<i>kernel méret</i>	<u>3</u> , 5	<i>rejtett réteg mérete</i>	10, 15, <u>20</u>
<i>csatornák</i>	(8, 16) (16, 32)	<i>csatornák</i>	(16, 32) (32, 32)	<i>rétegek száma</i>	<u>2</u>
<i>dropout</i>	<u>50%</u>	<i>dropout</i>	<u>50%</u>	<i>dropout</i>	<u>50%</u>
<i>zaj</i>	<u>0.05</u>	<i>zaj</i>	<u>0.05</u>	<i>zaj</i>	<u>0.05</u>
csapadék, globálsugárzás		csapadék, globálsugárzás		csapadék, globálsugárzás	

3.9. táblázat. Időjárási változók modellei, paramétere
CNN: Konvolúciós Háló, TCN: Temporális Konvolúciós Háló

A csapadék előrejelzésre a Temporális Konvolúciós Háló bizonyult a legjobbnak, egy kicsivel teljesít csak jobban, mint a rekurens modellek. A globálsugárzás előrejelzésében a GRU volt a legjobb, és a Konvolúciós Háló is közel olyan eredményesen teljesített. A kiértékelésben, mivel kis különbség volt a kettő között, a Konvolúciós Hálóval dolgozok tovább, mert az sokkal gyorsabban tanult be a feladatra. A megfelelő változókat, modelleket, és paramétereket modellenként a 3.9. táblázatban aláhúzással jelöltem.

Az elektromos fogyasztás előrejelzésére LSTM és GRU modelleket teszteltem, hiszen ezek több lépésre is jó előrejelzéseket adtak, valamint a külső változókat is jól kezelik. A paramétereket első körben egylépéses előrejelzésre optimalizáltam, aztán rekurzív előrejelzés során további, kisléptékű optimalizációt kellett végeznem.

Paraméter	Értékei
<i>modell</i>	<u>GRU</u> , LSTM
<i>rejtett réteg mérete</i>	10, 15, 25, <u>30</u> , 35, 40, 45, <u>50</u>
<i>rétegek száma</i>	<u>1</u> , <u>2</u> , 3
<i>kétirányúság</i>	Igen, Nem
<i>dropout</i>	<u>50%</u>
<i>zaj</i>	0, <u>0.05</u>

3.10. táblázat. Egylépéses előrejelzési modell paraméterei

A paraméterkeresés során több jól teljesítő paramétervektort találtam, így a továbbiakban több konfigurációt is teszteltem. A metrikák alapján a 2 rétegű, 30-as rejtett réteg méretű modell a legjobb, de az 1 rétegű 50-es rejtett réteg méretű modell is közel ugyanolyan eredményeket hoz, ezért mindkettőt továbbvittem a végső teszteléshez.

```

1 class MultiModelRec(nn.Module):
2     def __init__(self, features=11, pred_len=3, hidden_size=15,
3         num_layers=2, dropout=0.0,
4         noise=0.0, bidirectional=True, **kwargs):
5         super(MultiModelRec, self).__init__()
6         self.out_features = 3
7         self.pred_len = pred_len
8
9         self.gru = GRU(features, 1, hidden_size, num_layers,
10             dropout, noise, bidirectional)
11
12         self.tcn = TCN(24, 1, (32,) * 2, kernel_size=5, dropout=
13             dropout, noise=noise)
14
15         self.conv = ConvNetForRecursion((16, 32), (6, 12), 0.5,
16             0.05)
17
18     def forward(self, x, y, teacher_forcing=0.0):
19         batch_size = x.shape[0]
20
21         if y.shape[2] != self.gru.gru.input_size:
22             pre_calc = torch.concat((
23                 torch.zeros(batch_size, self.pred_len, self.
24                     out_features).to(tl.TRAINER_LIB_DEVICE),
25                 y), dim=2)
26             teacher_forcing = 0.0
27         else:
28             pre_calc = y

```

```
23     output = torch.zeros(batch_size, self.pred_len).to(tl.  
24         TRAINER_LIB_DEVICE)  
25  
26     for i in range(self.pred_len):  
27         out = torch.concat((  
28             self.gru(x),  
29             self.tcn(x[:, :, 1]),  
30             self.conv(x[:, :, 2])  
31         ), dim=1)  
32  
33         output[:, i] = out[:, 0]  
34  
35         x = torch.cat((x[:, 1:], pre_calc[:, i].unsqueeze(1)),  
36             dim=1)  
37         for j in range(self.out_features): # roll teacher-  
38             forcing for each feature  
39             if torch.rand(1) > teacher_forcing:  
40                 x[:, -1, j] = out[:, j]  
41  
42     return output
```

3.4. forráskód. Többmodelles rekurziós modell

A stratégiát implementációs szinten egy PyTorch modellbe szerveztem, így a gradiens visszaterjesztése a teljes modellen egyidőben történhet. Így lehetőséget ad gyorsabb tanításra és arra, hogy olyan modelleket definiáljunk és használjunk könnyedén, amelyek egymás predikcióit is felhasználják tanítás valamint predikció közben.

4. fejezet

Összefoglalás és eredmények

4.1. Eredmények

Ebben a fejezetben tüntetem fel és értékelem ki a modell és stratégia párok teljesítményét. A konfigurációkat 6-szor futtattam és az eredmények átlagát és szórását tüntetem fel. A többszöri futtatás a neurális hálónál használt random inicializáció miatt szükséges. A modellek teljesítménye az idősoros adatfelosztás által megadott adatrészeken történik, ezen eredmények felett is átlagolunk. Megtekintjük a modellek összefoglalt és előrejelzett órákra bontott teljesítményét, valamint az eredményesebb modellek pár predikciós grafikonját és tanulási, valamint predikciós idejét, grafikonját.

Minden modell és stratégia párban azon adatrészek eredményeit használom melyek nem adnak kiugróan rossz eredményeket, ez általában a különböző rekurzív modelleknél fordul elő (GRU és LSTM). Így vagy az első vagy az első és második adatrész eredményeit nem veszem figyelembe attól függően, hogy melyik mutat jobb eredményeket. Azért így teszek, hogy ne büntessem azon modelleket melyek kevés adattal is jól dolgoznak, de ne kerüljenek hátrányba azon modellek melyek több adattal sokkal jobbak az előbb említettekénél.

A MAVIR adatbázisnál (3.2.1) már említett 'MAVIR becslés' mező megfelelő baseline-t (kiindulási pontot) ad arra, hogy a különböző konfigurációkkal mekkora pontosságot kell elérni. Pontos információ nincs arról, hogy a MAVIR ezeket a predikciókat milyen gyakran frissíti és milyen modellel adja, de az biztos, hogy legalább 12 órára ad előrejelzéseket. Predikcióik pontosságát így csak a 4.1. táblázatban tüntetem fel.

A betanított modelleket és részletesebb eredményeiket ebben a [Jupyter Demo Notebook](#)-ban lehet megtekinteni.

Modell és Stratégia	MAE (MW)	RMSE (MW)	MAPE (%)	MPE (%)
<i>MAVIR becslés</i>	252.58±0	300.81±0	4.97±0	-4.70±0
<i>MIMO RF</i>	69.96±15.29	104.32±24.0	1.42±0.32	0.017±0.25
<i>MIMO CNN</i>	103.13±21.5	146.69±27.55	2.12±0.46	0.196±0.416
<i>MIMO TCN</i>	63.92±10.46	92.96±15.57	1.31±0.22	-0.001±0.198
<i>MIMO LSTM</i>	62.62±6.05	88.97±9.19	1.28±0.13	0.05±0.175
<i>Seq2seq GRU</i>	58.75±6.22	84.21±9.83	1.21±0.13	0.08±0.168
<i>Recursive GRU</i>	94.41±14.41	128.39±17.39	1.94±0.28	0.119±0.744
<i>MMRec GRU 1L</i>	65.4±7.68	92.43±10.88	1.34±0.17	0.114±0.292
<i>MMRec GRU 2L</i>	64.79±7.31	90.85±10.27	1.32±0.16	-0.029±0.295

4.1. táblázat. Összefoglalt modell és stratégia teljesítmény

Minden metrikához átlag±szórás van feltüntetve, átlagolva az előrejelzett órákon át
MMRec: többmodelles rekurzív stratégia, 1L és 2L pedig az 1 illetve 2 réteget jelenti

A MAVIR által adott becslésekben az MAE és RMSE értékek relatív közel vannak, ez nem meglepő, hiszen az ilyen célra adott előrejelzéseknél nagy előny, ha a modell stabil. A tesztelt megközelítések közül ebben az összehasonlításban a Seq2seq stratégia bizonyult a legeredményesebbnek, százalékos hibája mindössze 1.2%, még akkor is jól használható, ha olyan megkötést adunk a rendszernek, hogy inkább mindig túlbecsülje az értéket, mert például minimalizálni akarjuk az importált áram mennyiségét.

A Seq2seq megoldáshoz hasonlóan jó eredményeket ad a MIMO-LSTM és a MIMO-TCN konfiguráció, de a TCN modell sokkal nagyobb szórással dolgozik mivel ott külső változókat nem vettük figyelembe. Az új stratégia közül egyértelműen a két rétegű GRU-t fő modellként használó megoldás a legjobb, az eredeti rekurzív stratégiánál viszont mind a kettő sokkal eredményesebb.

Ha csak az átlagokat vizsgáljuk, akkor a Véletlenszerű Erdő eredményei is jónak tűnnek, de itt még a TCN-nél is nagyobb szórást tapasztaltam, amely főként a korai adatrészekben adott kiemelkedő, de a későbbi adatrészekben kapott rossz eredmények nagy különbségéből adódik.

Modell és Stratégia	Óra	MAE (MW)	RMSE (MW)	MAPE (%)	MPE (%)
<i>MIMO RF</i>	1	46.89±7.92	68.31±11.86	0.95±0.17	-0.026±0.151
	2	71.17±15.46	102.58±23.04	1.45±0.33	0.018±0.24
	3	91.82±22.71	131.97±33.18	1.87±0.48	0.058±0.359
<i>MIMO CNN</i>	1	90.24±18.82	125.25±23.93	1.86±0.4	0.163±0.416
	2	103.99±21.74	147.22±27.83	2.14±0.47	0.196±0.411
	3	115.17±24.31	164.84±30.9	2.37±0.52	0.227±0.443
<i>MIMO TCN</i>	1	40.87±4.27	57.0±5.92	0.84±0.09	-0.026±0.111
	2	64.02±10.55	88.68±14.57	1.31±0.22	-0.015±0.196
	3	86.87±16.73	121.59±22.5	1.79±0.36	0.038±0.304
<i>MIMO LSTM</i>	1	46.05±4.12	62.95±5.29	0.95±0.09	0.029±0.147
	2	63.52±5.91	87.93±8.25	1.3±0.13	0.058±0.176
	3	78.29±10.24	109.51±14.75	1.6±0.22	0.063±0.211
<i>Seq2seq GRU</i>	1	41.19±3.06	56.75±3.65	0.85±0.07	0.026±0.114
	2	59.59±6.23	82.55±8.95	1.22±0.13	0.08±0.171
	3	75.48±10.14	105.88±15.03	1.54±0.22	0.133±0.24
<i>Recursive GRU</i>	1	71.76±9.61	95.8±10.81	1.48±0.18	0.096±0.591
	2	96.93±14.65	129.47±16.84	1.99±0.29	0.128±0.762
	3	114.53±19.65	153.2±23.47	2.35±0.39	0.132±0.883
<i>MMRec GRU 1L</i>	1	52.04±4.81	71.13±6.15	1.07±0.11	0.099±0.196
	2	64.37±7.8	89.52±10.53	1.32±0.18	0.101±0.287
	3	79.79±11.71	111.86±15.83	1.63±0.26	0.143±0.409
<i>MMRec GRU 2L</i>	1	51.75±4.89	70.29±6.4	1.06±0.11	-0.004±0.215
	2	63.99±7.41	88.33±9.96	1.31±0.16	-0.038±0.291
	3	78.63±10.48	109.49±14.36	1.6±0.23	-0.046±0.389

4.2. táblázat. Órás bontású modell és stratégia teljesítmény

Minden metrikához átlag±szórás van feltüntetve

MMRec: többmodelles rekurzív stratégia, 1L és 2L pedig az 1 illetve 2 réteget jelenti

A fenti táblázattal jobban megvizsgálhatjuk az egyes modellek működését és a különböző távokra adott előrejelzések értékét. A legjobb rövidtávú előrejelzést a MIMO-TCN megközelítés adja, vélhetően jól rátanult az elektromos fogyasztási grafikon alakjára, tendenciáira. 2 órás távlatban már a Seq2seq stratégia bizonyosul a legjobbnak, ezt a 4.1. táblázatból már előre láthattuk. A 3 órás előrejelzés sem mutat más eredményeket, itt is a Seq2seq a legeredményesebb.

Fontos még megnézni a metrikák óránkénti különbségét, ezzel lehet arra következtetni, hogy esetlegesen a nagyobb távú előrejelzésekben hogyan fognak teljesíteni. A külső változókat nem használó modelleknél magas ez az arány, valamint a Véletlenszerű Erdő is ilyen tendenciát mutat. A Seq2seq és MIMO-LSTM konfigurációk ennél kisebb arányt

mutatnak, viszont a Seq2seq esetén kicsivel nagyobb, mivel az lépésenként építi a predikciót, így a halmozódó hiba egyre növekszik. A rekurzív modellek esetén viszont ezek az arányok kisebbnek bizonyulnak, legalábbis a többmodelles rekurzió esetén, itt a külső változókra alkalmazott specifikusabb modellek segítik a fő modellt abban, hogy a predikciói ne csak a halmozódó hibával érkező fogyasztáson alapuljanak.

Modell és Stratégia	Adatrészek 1-3. Tanítás (p)	Adatrészek 4-6. Tanítás (p)	Adatrészek 7-9. Tanítás (p)	Predikció (mp)
<i>MIMO RF</i>	0.84±0.42	2.44±0.49	4.15±0.59	0.082±0.026
<i>MIMO CNN</i>	1.05±0.45	1.96±0.58	2.96±1.12	0.136±0.011
<i>MIMO TCN</i>	1.57±0.47	2.75±0.66	4.15±1.71	0.258±0.045
<i>MIMO LSTM</i>	1.41±1.05	2.72±0.81	4.83±1.48	0.153±0.015
<i>Seq2seq GRU</i>	3.45±1.61	5.59±2.59	9.45±3.58	0.25±0.019
<i>Recursive GRU</i>	3.02±1.12	7.88±2.02	10.59±2.69	0.731±0.035
<i>MMRec GRU 1L</i>	2.53±1.23	5.06±3.22	10.88±4.23	2.242±0.285
<i>MMRec GRU 2L</i>	2.93±1.87	5.48±3.49	12.27±3.31	1.984±0.044

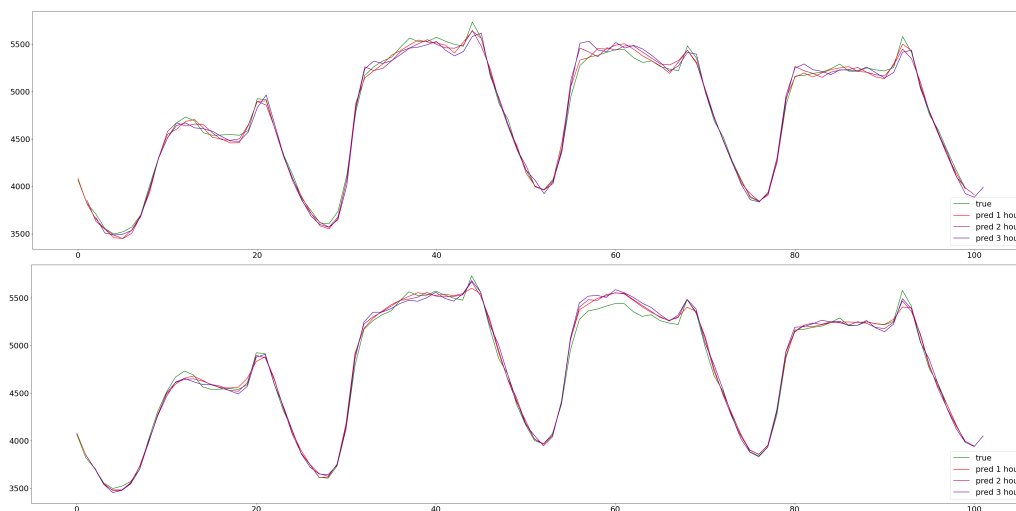
4.3. táblázat. Modell és stratégia tanítási és predikciós idő

Környezet: AMD Ryzen 5 5600H 3.3GHz CPU, Nvidia RTX 3050Ti 4GB GPU

Tanítási időben nincs nagy meglepetés, a komplexebb modellek általánosságban hosszabb ideig tanulnak, de minden konfiguráció tanítása belefér 1 órába, ami akkor fontos, ha egy alkalmazásban szeretnénk alkalmazni és valamilyen időközönként újra akarjuk tanítani. A predikciós idők nagyon megnövekednek a többmodelles rekurziónál, de ebben az alkalmazásban ez sem túl magas, pár másodperc várakozás egy ilyen rendszerben nem okoz gondot.

Összességében erre a feladatra és adathalmazra a Seq2seq GRU enkóder-dekóder architektúra a legeredményesebb. Emellett viszont biztató az, hogy a többmodelles rekurzió a sorban felépített előrejelzéseivel óránként kevesebb hibát halmoz a többi stratégiánál, viszont pont a kiinduló, 1 órás előrejelzésnél ad gyengébb eredményeket. A továbbiakban tekintsünk meg egy-egy predikciós grafikont a Seq2seq és az MMRec 2L konfigurációkról.

A két grafikonon (4.1) a zöld vonal jelzi a kívánt értéket, a piros az 1 órás, sötét lila a 2 órás, világosabb lila a 3 órás előrejelzést jelenti. X tengelyen az idő szerepel órában, Y tengelyen a fogyasztás MW-ban. A grafikon kezdőpontja egy véletlenszerűen választott időpont a 4. adatrészből, tehát az 1-3. adatrészekben tanított modelleket látunk. Általánosságban a délutáni időszak bizonyult a legnehezebbnek predikciós szempontból.



4.1. ábra. Predikciós grafikonok, fenn: Seq2seq, alul: MMRec

4.2. Kitekintés

A többmodelles stratégiát többféleképpen is lehet implementálni, az én implementációm esetében problémát jelent az első óra előrejelzése, viszont jól működik a megváltoztatott teacher-forcing és a modellek is könnyen taníthatóak egymással párhuzamosan, így könnyen használják egymás predikcióit.

A stratégiát érdemes lehet hosszabb szekvenciák predikciójára is tesztelni, ugyanis a vizsgáltak közül ez halmozta fel relatívan a legkevesebb hibát. Léteznek a területen komplexebb modellek is, mint például a figyelemközpontú modellek és a Spacetimeformer modellek [16], ezekkel is össze lehetne hasonlítani.

4.3. További alkalmazások

A többmodelles stratégia a tesztek alapján azt mutatja, hogy legnagyobb előnye a külső változók effektív predikciója és alkalmazása, így hasznos lehet olyan idősorokon ahol a prediktálni kívánt változó erősebben függ valamilyen külső hatástól, például napenergia vagy szélenergia predikció.

Köszönetnyilvánítás

A kutatás, a TDK projekt és a kutatásból készülő publikáció az ELTE Informatikai Kara által ösztöndíjas támogatásban részesült pályázat keretében. Külön köszönet Balázs Istvánnak a TrendEntől a kiindulási probléma felvetéséért és a projekt felügyeletéért.

Felhasznált mesterséges intelligencia eszközök

A dolgozathoz kidolgozott forráskód [17] létrehozásához GitHub Copilot [18] alapú forráskód kiegészítő eszközt használtam. A készítéskor még nem volt elérhető a legújabb csevegés alapú verzió.

Az említetten kívül más, mesterséges intelligencia alapú eszközt nem használtam a dolgozat megírása közben.

Irodalomjegyzék

- [1] Alberto Gasparin, Slobodan Lukovic és Cesare Alippi. „Deep learning for time series forecasting: The electric load case”. *CAAI Transactions on Intelligence Technology* 7.1 (2022), 3–4. old. DOI: <https://doi.org/10.1049/cit2.12060>. eprint: <https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/cit2.12060>. URL: <https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/cit2.12060>.
- [2] Diederik P. Kingma és Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: <https://arxiv.org/abs/1412.6980v9>.
- [3] Timothy Dozat. „Incorporating nesterov momentum into adam”. (2016). URL: <https://openreview.net/forum?id=OM0jvwB8jIp57ZJjtNEZ>.
- [4] Philipp Probst, Marvin N. Wright és Anne-Laure Boulesteix. „Hyperparameters and tuning strategies for random forest”. *WIREs Data Mining and Knowledge Discovery* 9.3 (2019), 1–7. old. DOI: <https://doi.org/10.1002/widm.1301>. eprint: <https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1301>. URL: <https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1301>.
- [5] Jiuxiang Gu és tsai. „Recent advances in convolutional neural networks”. *Pattern Recognition* 77 (2018), 354–357. old. ISSN: 0031-3203. DOI: <https://doi.org/10.1016/j.patcog.2017.10.013>. URL: <https://www.sciencedirect.com/science/article/pii/S0031320317304120>.
- [6] Alberto Gasparin, Slobodan Lukovic és Cesare Alippi. „Deep learning for time series forecasting: The electric load case”. *CAAI Transactions on Intelligence Technology* 7.1 (2022), 11–14. old. DOI: <https://doi.org/10.1049/cit2.12060>. eprint: <https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/cit2.12060>.

- 10.1049/cit2.12060. URL: <https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/cit2.12060>.
- [7] Shaojie Bai, J. Zico Kolter és Vladlen Koltun. *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*. 2018. arXiv: 1803.01271 [cs.LG].
- [8] Shamsul Masum, Ying Liu és John Chiverton. „Multi-step Time Series Forecasting of Electric Load Using Machine Learning Models”. *Artificial Intelligence and Soft Computing*. Szerk. Leszek Rutkowski és tsai. Cham: Springer International Publishing, 2018, 151–153. old. ISBN: 978-3-319-91253-0.
- [9] Wei Li, Amin Kiaghadi és Clint Dawson. „Exploring the best sequence LSTM modeling architecture for flood prediction”. *Neural Computing and Applications* 33.11 (2021), 5571–5580. old. ISSN: 1433-3058. DOI: 10.1007/s00521-020-05334-3. URL: <https://doi.org/10.1007/s00521-020-05334-3>.
- [10] Guizhu Shen és tsai. „Deep Learning with Gated Recurrent Unit Networks for Financial Sequence Predictions”. *Procedia Computer Science* 131 (2018). Recent Advancement in Information and Communication Technology: 897–898. old. ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2018.04.298>. URL: <https://www.sciencedirect.com/science/article/pii/S1877050918306781>.
- [11] Souhaib Ben Taieb és tsai. „A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition”. *Expert Systems with Applications* 39.8 (2012), 7068–7070. old. ISSN: 0957-4174. DOI: <https://doi.org/10.1016/j.eswa.2012.01.039>. URL: <https://www.sciencedirect.com/science/article/pii/S0957417412000528>.
- [12] Zaki Masood és tsai. „A Multi-Step Time-Series Clustering-Based Seq2Seq LSTM Learning for a Single Household Electricity Load Forecasting”. *Energies* 15.7 (2022), 5–6. old. ISSN: 1996-1073. DOI: 10.3390/en15072623. URL: <https://www.mdpi.com/1996-1073/15/7/2623>.
- [13] Thuan Nguyen Thi Hiep, Nhan To Thanh és Tho Quan Thanh. „Forcing-Seq2Seq Model: An Automatic Model of Title Generation for Natural Text Using Deep Learning”. *Proceedings of the Future Technologies Conference (FTC) 2021*,

- Volume 2.* Szerk. Kohei Arai. Cham: Springer International Publishing, 2022, 388–402. old. ISBN: 978-3-030-89880-9.
- [14] Edzard S Gelsema és Laveen N Kanal. *Pattern Recognition in Practice IV: multiple paradigms, comparative studies and hybrid systems*. Elsevier, 1994, 403–413. old. URL: https://books.google.com/books?hl=en&lr=&id=sbajBQAAQBAJ&oi=fnd&pg=PA403&dq=comparative+study+of+techniques+for+large+scale&ots=KdIOYpA8wj&sig=hd0sBP1HX4hcDjx4RLg_chheojc#v=onepage&q=comparative%20study%20of%20techniques%20for%20large%20scale&f=false.
- [15] Roberto Sánchez-Reolid és tsai. „One-dimensional convolutional neural networks for low/high arousal classification from electrodermal activity”. *Biomedical Signal Processing and Control* 71 (2022), 4. old. ISSN: 1746-8094. DOI: <https://doi.org/10.1016/j.bspc.2021.103203>. URL: <https://www.sciencedirect.com/science/article/pii/S1746809421008004>.
- [16] Jake Grigsby, Zhe Wang és Yanjun Qi. „Long-Range Transformers for Dynamic Spatiotemporal Forecasting”. *CoRR* abs/2109.12218 (2021). arXiv: 2109.12218. URL: <https://arxiv.org/abs/2109.12218>.
- [17] Inc. GitHub. *MeepOwned13/es_load_fs_HUN: This is a repository containing code for an ongoing project on forecasting electricity load in Hungary*. https://github.com/MeepOwned13/es_load_fs_HUN. A repository tulajdonosa: Sebők Máttyás - @MeepOwned13, Elérve: 2024.05.18.
- [18] Inc. GitHub. *GitHub Copilot · Your AI pair programmer*. <https://github.com/features/copilot>. Elérve: 2024.05.18.

Ábrák jegyzéke

2.1. A csúsztatott ablakos megközelítés	6
2.2. Adatfelosztás keresztvalidációhoz	8
2.3. Véletlenszerű erdő regresszióhoz	9
2.4. Causal convolution	10
2.5. Temporális konvolúciós háló részei	11
2.6. LSTM cella	12
2.7. LSTM használatának módjai szekvencia feldolgozására	12
2.8. GRU cella	13
2.9. Seq2seq GRU modell	16
2.10. Többmodelles rekurzió	17
3.1. MAVIR által közzétett nettó terhelési adatok ábrázolása	19
3.2. Korrelációs mátrix	21
3.3. Csapadék grafikonok	23
3.4. Globális sugárzás grafikonok	23
3.5. Konvolúciós háló architektúra	26
4.1. Predikciós grafikonok	39

Táblázatok jegyzéke

3.1. OMSZ állomások elérhető adatai	20
3.2. Véletlenszerű erdő paraméterei	25
3.3. $n_T = 24$ Konvolúciós háló struktúrája	27
3.4. $n_T = 48$ Konvolúciós háló struktúrája	27
3.5. Temporális Konvolúciós Háló paraméterei	28
3.6. LSTM Háló paraméterei	29
3.7. GRU seq2seq modell paraméterei	31
3.8. Rekurzív stratégia paraméterei	31
3.9. Időjárási változók modellei, paraméterei	32
3.10. Egylépéses előrejelzési modell paraméterei	33
4.1. Összefoglalt modell és stratégia teljesítmény	36
4.2. Órás bontású modell és stratégia teljesítmény	37
4.3. Modell és stratégia tanítási és predikciós idő	38