EÖTVÖS LORÁND TUDOMÁNYEGYETEM
TERMÉSZETTUDOMÁNYI KAR

Varga Botond

Idősorok elemzése és prediktálása mély neurális hálózatokkal

SZAKDOLGOZAT

Matematika BSc, matematikai elemző szakirány

TÉMAVEZETŐ:

Lukács András
SZÁMÍTÓGÉPTUDOMÁNYI TANSZÉK

BUDAPEST, 2019

Tartalomjegyzék

1. Bevezetés **2**
1.1. Téma ismertetés, motivációk . . . 2
1.2. Történeti bevezetés a mesterséges intelligenciához . . . 2

2. A szélenergia termelésbecslésének feladata és modellezése **3**
2.1. A feladat . . . 4
2.2. Megoldási stratégia . . . 4

3. Az adathalmaz **5**
3.1. Az adat felépítése . . . 5
3.2. Vizsgált adatok alapstatisztikái . . . 6
3.3. Adatok előkészítése . . . 7
3.4. Az adat normalizálása . . . 8

4. A neurális hálózatok alapjai **8**
4.1. Adattudományi alapfogalmak . . . 8
4.2. A perceptron modell . . . 11
4.3. Az alap neurális hálózat . . . 12
4.4. Aktivációs függvény . . . 14
4.5. A gradiens módszer . . . 15
4.6. A hibavisszaterjesztés . . . 16
4.7. Rekurrens neurális hálózatok . . . 18
4.8. Konvolúciós hálózatok . . . 19
4.9. A Long Short Term Memory hálózatok . . . 20

5. A megépített modell **22**
5.1. A neurális háló . . . 22

6. Kiértékelés **23**
6.1. A modell tanítás és kiértékelés módszertana . . . 23
6.2. A kiértékeléssel kapcsolatos fogalmak . . . 24
6.3. Kiértékelés . . . 27
6.4. Fejlesztési lehetőségek . . . 28

1. Bevezetés

1.1. Téma ismertetés, motivációk

Napjainkban az emberiségnek nagyon komoly kihívásokkal kell szembenéznie: ilyen például a globalizáció problémái, az éghajlat változás (globális felmelegedés, elsivatagosodás, vízválság, tengerszint emelkedés), népességi mozgások, a környezetszennyezés. Ebbe a sorba tartozik a véges fosszilis energiaforrások elapadása is.

Már a XX. században megkezdődött a fosszilis energiaforrások kiváltása az eddig olyan megújuló energiaforrásokkal, mint a napenergia és a szélenergia.E folyamat terjedése az EU-ban és Magyarországon is megfigyelhető, köszönhetően az EU megújuló energiatermelést támogató irányelveinek és az ehhez rendelkezésre bocsátott pénzügyi forrásoknak, vissza nem térítendő támogatásoknak is. Az EU csatlakozás óta hazánkban is megjelentek a szélerőmű-parkok, melyek több szélerőművet kötnek egy közös rendszerbe. Azok a szélerőmű-parkok, melyek tengerpart mentén vagy tengerben helyezkednek el, folyamatos termelést tudnak biztosítani, mivel szinte állandóan fúj a szél. Magyarország azonban nem rendelkezik ilyen környezeti adottságokkal, így kezelnie kell a szélenergia termelés változó mértékéből adódó problémát, miszerint a szélerőmű által leadott teljesítmény rövidebb és hosszabb időintervallumok szintjén is nagyon változó. Magyarországnak és a szélenergia szempontjából hasonló környezeti adottságokkal rendelkező országoknak nagy pontosságra törekvő szélenergia termelésbecslő rendszereket kell kialakítaniuk, hogy az országos vagy területi villamosenergia ellátó rendszerhez való kapcsolódás zökkenő mentes és tervezhető legyen az energiagazdálkodás.

A nyílt energiapiacon alapvetően a kereslet-kínálat aránya határozza meg az árakat. A szélenergia termelésbecslő rendszerek gazdasági jelentősége tehát abban rejlik, hogy segítségével pontosabban lehet megbecsülni az energia árak jövőbeni alakulását, ami mind a szélerőmű üzemeltetőjének, mind pedig a fogyasztónak (jellemzően az állam) fontos adat a kiadások és bevételek tervezéséhez. Az energia árak alakulásának csak az egyik, de ma már nem elhanyagolható tényezője a szélenergia termelés.

1.2. Történeti bevezetés a mesterséges intelligenciához

Jelen szakdolgozat témája szélenergia termelésbecslés mesterséges intelligencia alkalmazásával, az ismert legjobb gyakorlatnak megfelelően konvolúciós visszacsatolásos LSTM neurális hálózattal (Long Short Term Memory recurrent neural network), mely a mélytanulásos hálózatok egy speciális, az idősoros feladatok megoldására kifejlesztett változata. Az első fejezetben rövid bevezetőt fogok nyújtani. A második fejezetben pontosan definiálom a feladatot, amelyet megoldok. Ezután ismertetem az adathalmazt, amellyel dolgozunk. A negyedik fejezetben a mély tanulási alapfogalmakat és modelleket mutatom be. Ezután

részletes ismertetést adok azokról a hálózatokról, amelyeket kipróbáltam. A hatodik fejezetben kiértékelem a kapott eredményeket.

Az MI területén a kutatók az 1950-es évek végén, az 1960-as évek elején tették meg az első lépéseket, gyakorlatilag a szobaszámítógépek megjelenésével egyidőben.

Frank Rosenblatt 1958-ban nyilvánosságra hozta a perceptron alapú lineáris algoritmusát, melyről akkoriban a New York Times azt írta, hogy a perceptron embrionális állapota egy olyan jövőbeni számítógépnek, amely képes lesz járni, beszélni, látni, írni, önmagát reprodukálni és tudatában lesz saját létezésének.[6] Marvin Minsky és Seymour Pappert 1969-ben publikált Perceptron című könyvében pesszimista jóslatokat fogalmazott meg a perceptron hálózatok jövőjével, alkalmazhatóságával és korlátaival kapcsolatban, s a közvélekedés szerint ezeknek a gondolatoknak a hatására a témakör kutatása háttérbe szorult. Kunihiko Fukushima 1979-ben ismertette a neocognitron neurális hálózati architektúrát, mellyel a kézírás/alakzat felismerés területén kísérleztek, s amelyet a konvolúciós neurális hálózatok elődjének tekintenek. David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams 1988-ban a Nature természettudományos szakfolyóiratban publikálta "Learning representations by back-propagating errors" című cikkét , melyben leírták a hibavisszaterjesztés algoritmusát, ami kulcsfontosságú szerepet játszik napjaink neurális hálózati architektúráiban. 1989-ben Yann LeCun és munkatársai az ATT Bell Labs-ban publikálták a "Backpropagation Applied to Handwritten Zip Code Recognition" című tanulmányt[14], amiben leírják azt a neurális hálózatot, amelyet az US Postal számára készítettek a kézírásos irányítószámok felismeréséhez. LeCun, Bottou, Bengio és Haffner 1998-ban publikálta a „Gradient-based learning applied to document recognition" című cikküket, melyben leírják a csekkekre kézzel írt számjegyek felismerésére alkalmazott, konvolúciós gradiens alapú tanulási algoritmusra épülő neurális hálózatot.[3] A 2010-es évek a mesterséges intelligencia történetében a mélytanulásos neurális hálókról szóltak, azoknak különböző architektúrájáról, jellemzően konvolúciós, visszacsatolásos és megerősítéses architektúrák alkalmazásáról.

2. A szélenergia termelésbecslésének feladata és modellezése

Az idősorok szekvenciális adathalmazok, amelyeket tipikusan egyenlő időközönként mérünk. Jelölése $x(t)$,$t = 0, 1, 2, 3, ...$ vektorokból álló sorozat, ahol a t az eltelt időt reprezentálja. Az $x(t)$-ket valószínüségi változóként kezeljük. Az olyan idősorokat, amelyek egyetlen változóra vonatkozó méréseket tartalmaznak, egyváltozós (univariate) idősoroknak nevezzük, míg azoknál az idősoroknál, ahol több változóra vonatkozóan is vannak mérések, többváltozós (multivariate) idősornak nevezzük. Egy idősor lehet folytonos vagy diszkrét. Folytonos idősor esetében a méréseket minden egyes időpillanatban elvégezhetjük, míg a diszkrét idősoroknál a méréseket csak adott időpillanatokban végezzük el. Folytonos idősorra jó példa

lehet egy hőmérsékletet rögzítő műszer adata, egy folyó vízhozama, vagy egy kémiai folyamat. Diszkrét idősornak tekinthetjük egy városnak a populációját, egy vállalat termékértékesítését, vagy két valuta közötti árfolyam változását. Diszkrét idősoroknál a mérések általában fix időközönként történnek, például óránkénti, napi, heti, havi vagy éves különbséggel. Egy idősorban lévő változóról feltesszük, hogy folytonos változóként mérik a valós számok halmazán. Érdemes megemlíteni, hogy egy folytonos idősort egyszerűen lehet diszkretizálni úgy, hogy összeintegráljuk valamilyen időközönként[5] .

2.1. A feladat

A dolgozatban megvizsgálom, hogy mennyire pontosan lehet előrejelezni, megjósolni a szélerőművek teljesítményét historikus adatok alapján. Adott egy adathalmaz, amely órákra lebontva tartalmazza az átlaghőmérsékletet, légnyomást, szélirányt, szélsebességet és természetesen az aznap generált áram mennyiségét. A feladat nem más, mint az, hogy neurális hálózatok segítségével meg tudjuk mondani, hogy holnap, holnapután, vagy egy hét múlva mennyi áramot fog generálni az erőmű. Ennek a feladatnak a megoldására több, state-of-the-art hálózatot és módszert is ki próbáltam.

2.2. Megoldási stratégia

A szakdolgozat, rekurrens neurális hálózatok (Recurrent Neural Network) kialakítására, és vizsgálatára szorítkozik, annak segítségével ad becslést (szélirány, szélerő, időtartam, termelt energia mennyisége) alapján a közeli és távoli jövőben termelt szélenergia várható mennyiségre. Az utóbbi években több hálózati architektúra született, dolgozatomban egy, az idősorok előrejelzéséhez használt architektúra családból fogok többet kipróbálni, nevezetesen az LSTM (Long Short-Term Memory) modellekből. Ez a modellcsalád részhalmaza egy még tágabb csoportnak, a Rekurrens Neurális Hálózatokénak. A standard előrecsatolt (feed forward) hálókhoz képest a rekurrens hálózatokban vannak visszacsatolások. Érdemes megemlíteni, hogy idősorok előrejelzésén kívül beszédfeldolgozásra, kézírás megfejtésére is használják. Az általam használt hálózat bemenetként paraméterezhető hosszúságú vektorokat kap, amelyeket az idősorból gyártok le. Ezt a halmazt nevezzük a "magyarázó halmaznak". Bemenetként még minden ilyen vektorhoz, rendelünk egy értéket, amely az úgynevezett célváltozónk. A dolgozat ezen pontjától a nyers idősort DF-nek, a magyarázó halmazt X-nek, a célváltozót pedig y-nak fogom nevezni.

Megjegyzés

A dolgozat során nem csak az LSTM hálózatok architektúráján, hanem az input halmaz szerkezetén is változtattam. A neurális hálózatot úgy programoztam, hogy ne csak fix méretű vektorokat tudjon kapni inputként, hanem bármilyen hosszúságút is, ezért ugyanazt

az architektúrát nem kell többször kódolni. A kapott eredmények összehasonlítása során szempont lesz az is, hogy milyen szerkezetű adatot és hogyan kapott meg a modell.

3. Az adathalmaz

A felhasznált adathalmazt egy, *GitHub*-on lévő *repository* lehet elérni :
https://github.com/ShashwatArghode/Wind-Energy-Prediction-using-LSTM

3.1. Az adat felépítése

DateTime
A rekordhoz tartozó mérés pontos ideje, órás pontossággal.

Air Temperature
A mérés idejét megelőző órában a levegő átlaghőmérséklete, Celsius fokban.

Air Pressure
A mérés idejét megelőző órában az átlagos légnyomás, atmoszférában mérve.

Wind Speed
A mérés idejét megelőző órában a szél átlagos sebessége, méter/szekundumban.

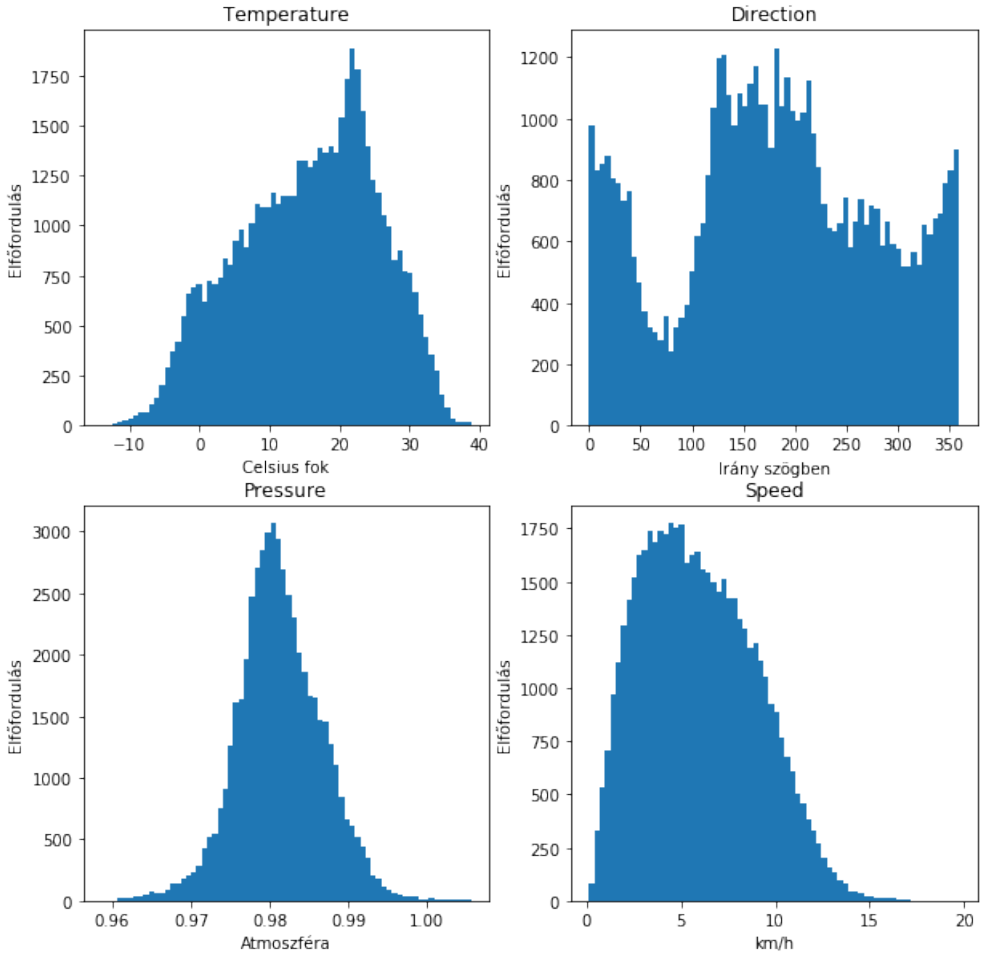
Wind Direction
A szél irányának, a szélerőművel bezárt átlagos szöge, fokban.

Power Generated
A mérés idejét megelőző órában az erőmű által összesen generált áram, kilowattban mérve.

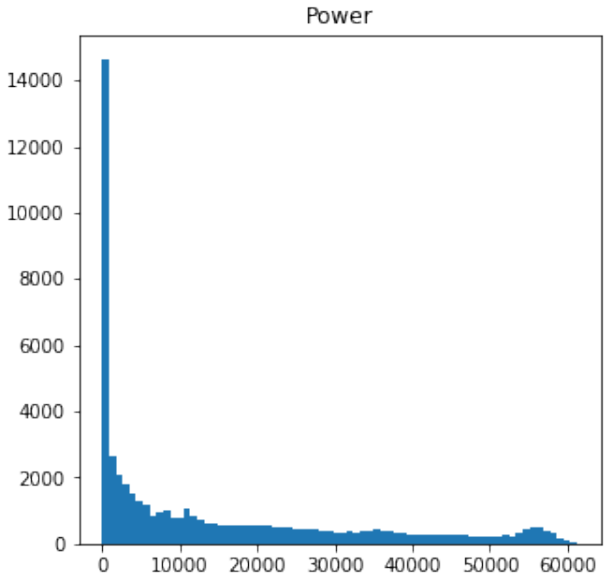
Az adathalmazban szereplő adatok közül már első ránézésre látható, hogy a legmeghatározóbb egyértelműen a Wind Speed, vagyis a szélsebesség lesz. A mai legmodernebb szélerőművek már képesek forogni a tengelyük körül azért, hogy a lehető leghatékonyabban működjenek. A szakdolgozat által felhasznált adatban nem ilyen szélerőműveket vizsgáltak, ezért fontos tényező marad a szél iránya is.

3.2. Vizsgált adatok alapstatisztikái

A modell építésének megkezdése előtt érdemes megvizsgálni az adatot, amivel dolgozunk. Ez több ponton is nagyon hasznosnak bizonyulhat. Első körben, ahhoz hogy jobban értsük pontosan milyen is az adat jellege, legyen képünk arról, hogy a változóink milyen eloszlásból származnak. Ezt érdemes összevetni azzal, ami intuitívnak érződik, és ha a kettő különbözik, akkor megideologizálni, hogy miért viselkedhet mégis másképp. Ezen kívül fontos lehet abból a szempontból is, hogy megértsük, hogy a modellünk miért, hogyan működhet.
A hisztogramokon látható, hogy a két uralkodó szélirány dominált, a dél-nyugati és az északi. A mért szélsebességek nagyrésze a 0-15 $\frac{km}{h}$ intervallumra esik, módusza körülbelül 5.5 $\frac{km}{h}$. A termelt energia mennyiségében a nullás érték dominál, és egészen 60000 kilowattig történtek mérések.



1. ábra. A magyarázó változók hisztogramjai



2. ábra. A célváltozó hisztogramja

3.3. Adatok előkészítése

Jól látható, hogy a generált áram mennyiségénél van egy kiugró érték 0-nál. Ez két dolog együttes hatásából fakad. Az egyik és a modell minőségét befolyásoló tényező az, hogy vannak napok amikor nem történt meg a mérés, így nincs adat arról, hogy mennyi áramot generált az erőmű. A probléma az, hogy az adathalmazban elég szakszerűtlen módon nullás számmal lettek feltöltve ezek az értékek, ahol nem történt mérés, így ezeket nem lehet megkülönböztetni azoktól az esetektől, amikor történt mérés, csak a körülmények miatt egyáltalán nem generálódott áram. Az ebből fakadó nehézségeket a következő módon sikerült megoldanom. Egy átlagos nagyméretű szélerőmű beindításához (forgásához) minimum 3,5 méter/szekundumos szélsebességre van szükség. Ezt a tényt felhasználva azokat az eseteket, ahol ennél gyengébb szél fújt, nem manipuláltam, hiszen valószínűleg ott helyes az adat. Azokat az eseteket, ahol ennél nagyobb volt a szél sebessége, KNN módszerrel helyettesítettem.A KNN a klasszifikáló algoritmusok egyik legalapvetőbb egyede. A KNN algoritmus diszkrét esetben úgy dolgozik, hogy egy ismeretlen címkéjű adatpontnak veszi a valamilyen metrikával mért, K darab legközelebbi szomszédját. A vizsgált pont címkéje ezek után a K legközelebbi szomszédban legtöbbször előforduló címke lesz.

A leggyakrabban használt módszerek a mediánnal, vagy átlaggal helyettesítik a hiányzó adatokat. Ezzel ellentéteben én a KNN módszerrel úgy töltöm fel a hiányzó adatokat, hogy veszem az adatpont legközelebbi hat szomszédját, és e pontoknak megvizsgálom a generált áram attribútumait, majd kiszámolom az átlagukat. Ezt minden hiányzó pontra elvégzem

és az így kapott az átlagok kerülnek a nulla értékek helyére.

Az adatokat CSV (Comma Separated Values) formátumban tudtam letölteni, ezért a legkézenfekvőbb az volt, hogy az adat importálását és előkészítését a Pandas Python környezetében végeztem el. A hat évnyi, órás felbontásban rögzített adatokat a tisztítás után úgy kellett átalakítani, hogy az megfeleljen a modellnek. Ehhez egy függvényt írtam, amelynek egyik paramétereként a Dataframe-t, másik paraméterként pedig a lookback-et kell megadni. A lookback annyit jelent, hogy a tanítás során hány előző mérésből próbálja meg megjósolni a következőt. Azért fontos, hogy ez paraméterezhető legyen, mert ez lesz az egyik olyan paramétere a modellnek, amellyel sokat mértünk.

3.4. Az adat normalizálása

A modell jobb teljesítménye értekében a hiányos adatok pótlása után az adat normalizálása következett.A gépi tanulás és mély neurális hálózatok kontextusában a normalizálás annyit jelent, hogy a különböző mért adatokat egy nagyságrendre hozzuk. Ennek a legnagyobb jelentősége a modell tanításánál van.

Amikor egy neurális hálózat elkezd tanulni az adatból, akkor gyakorlatilag egy leképezést tanul meg, ami az input adatokat a célváltozókba képezi le. Ha az input adat nagy nagyságrendbeli értékeket vesz fel (akár milliós nagyságrend is lehet), akkor jó eséllyel a hálózatunk szintúgy nagy súlyokat tanul meg. Ez azt eredményezheti, hogy a hálózatunk instabillá válik, vagyis rossz teljesítményt nyújt a tanulás során, és érzékenyebb az input adatra.

Az adatomat a Scikit-Learn Python csomag segítségével normalizáltam. Itt több módszer is van a normalizálásra, jelen dolgozatban min-max scaling-et alkalmaztam. Ez az egyik legegyszerűbbb módja az adat skálázásának, ami a következő képlettel működik:

$$\hat{X} = \frac{X - min(X)}{max(X) - min(X)}$$

Az X egy vektor, aminek minden eleméből kivonjuk X minimumát, majd elosztjuk a maximumának és minimumának különbségével.

4. A neurális hálózatok alapjai

4.1. Adattudományi alapfogalmak

Adatutudomány

Az adattudomány közelmúltban felvirágzó tudomány, amely azzal foglalkozik, hogy jel-

lemzően nagy mennyiségű adatból milyen nem triviális mintázatokat, információt lehet kinyerni. Fontos, hogy az információ ne triviális, de valamilyen szempontból hasznos legyen.

Machine Learnin/Gépi tanulás

A gépi tanulás egy részterülete a számitástudománynak, amely azzal foglalkozik, hogy algoritmusokat építsen, amelyek hasznosak, és valamilyen jelenségről gyűjtött példákból táplálkoznak. Ezek a példák jöhetnek a természetből, lehetnek ember által készitett, de akár egy másik algoritmus által generáltak is. A gépi tanulást úgy is lehet definiálni, hogy először gyűjtünk egy adathalmazt, majd azon algoritmikusan épitünk egy modellt. A a kapott modellt ezután felhasználjuk valamilyen gyakorlati probléma megoldásához.. [7]

Tanulás

Azt mondjuk, hogy egy számítógépes program **tanul** az E *(Experience)* tapasztalatokból a T *(Tasks)* feladatok tekintetében és a teljesítményének mértéke P *(Performance)*, ha a T feladatokon a teljesítménye, azaz P, az E tapasztalat által (azok hatására), javul. [8]

Neuron

A neurális hálók rendszerét csomópontok/neuronok és azokat összekötő kapcsolatok/élek sokasága alkotja. Ezeket hívjuk a gépi tanulásban neuronoknak, ezek a hálókban rétegekbe rendeződnek, és a rétegek közt történik az információ továbbítás. [11]

A hibafüggvény

A matematikában, főleg az optimalizációs feladatokban a hiba-, vagy költségfüggvény egy olyan függvény, ami egy jelenséghez, vagy adatpontokhoz egy valós számot rendel. Ez a valós szám reprezentálja a modell hibáját a vizsgált adatpontokon. Egy optimalizációs feladat mindig ezt a költségfüggvényt próbálja minimalizálmi. Abban az esetben, amikor az optimalizációs feladat egy függvény maximalizálása, akkor célfüggvénynek nevezzük.
Ebben a szakdolgozatban a RMSE(Root Mean Squared Error), vagyis az átlagos négyzetes hiba négyzetgyökét, az átlagos reziduális hibát (Mean Residual Error) és a R^2-t értéket fogom megpróbálni optimalizálni.

A négyzetes hiba gyöke (Root Mean Squared Error)

Egy modell RMSE hibáját a következőképpen lehet kiszámítani

$$\frac{1}{N}\sum_{i=1}^{N}((f(x_i)-y_i)^2)$$

, ahol $f(x_i)$ a modellünk outputja, y_i pedig a pontos érték.

Átlagos reziduális hiba

A reziduális hibát a következőképpen számoljuk

$$\frac{1}{N}\sum_{i=1}^{N}(f(x_i) - y_i)$$

R^2 érték

Ahhoz, hogy definiálni tudjuk az R^2 -et, előbb szükséges bevezetni néhány fogalmat.

Egy adathalmaznak n eleme van $y_1, ..., y_n$ (nevezzük őket y_i)-nek. Ezeket egy vektorba rendezve kapjuk a $Y = (y_1, y_2, y_3, ..., y_n)$, vektort.A Y vektor minden eleméhez tartozik egy prediktált érték is, amit az adott feladatra illesztett modell jósol. Ezeket az értékeket is rendezhetjük egy vektorba, ez legyen a $f = (f_1, f_2, f_3, ..., f_n)$ vektor.

4.1. Definíció. Reziduális

Az $e_i = y_i$ -f_i $i = 1....., N$, értékeket reziduálisoknak nevezzük. A reziduálisokat az e vektorban tároljuk.

A következő kifejezés az adat átlagát adja meg.

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

A valós adat négyzetes eltérése az átlagtól.:

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2,$$

A jósolt adatok négyzetes eltérése az átlagtól:

$$SS_{\text{reg}} = \sum_i (f_i - \bar{y})^2,$$

A reziduálisok négyzetének összege:

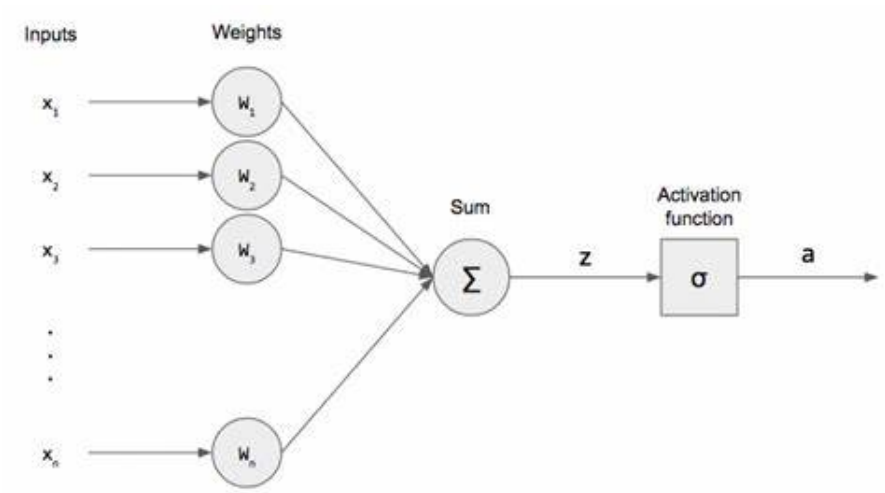
$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

Ezekkel a definíciókkal az R^2 a következőképpen adható meg :

$$R^2 \equiv 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

4.2. A perceptron modell

A modern értelemben vett perceptron egy algoritmus amely bináris klasszifikációs feladatot old meg.



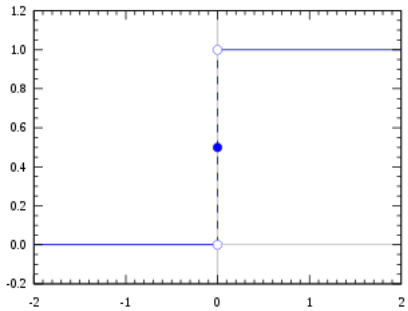
3. ábra. A perceptron modell

Az ábrán az inputs oszlopban vannak az $(x_1, x_2, x_3, ..., x_n)$ bináris input értékek. A weights oszlopban pedig a pontosan ugyanolyan számú $(w_1, w_2, w_3, ..., w_n)$ súlyok. Mindkét oszlopra lehet egy n hosszú vektorként gondolni. Az első lépésben minden input értéket $(x_1, x_2, x_3, ..., x_n)$, megszorzunk a hozzá tartozó súlyokkal $(w_1, w_2, w_3, ..., w_n)$. A szorzások elvégzése után a kapott értékeket összeadjuk, ezt az állapotot nevezzük aktiváció előtti (preactivation) állapotnak. Érdemes megemlíteni, hogy ha az input értékekre és a súlyokra vektorként tekintünk, akkor ez tulajdonképpen a két vektor skaláris szorzata. A skaláris szorzás elvégzése után dönthetünk úgy, hogy hozzáadunk egy bias paramétert, ami egy

konstans. A matematikai korrektség érdekében ezt a bias konstanst belecsempészhetjük a képletbe úgy, hogy az input vektorhoz hozzáveszünk egy x_0 elemet, és ehhez az elemhez hozzárendelünk egy egy értékű w_0 súlyt.

$$\mathbf{Z} = \sum_{i=0}^{N} (x_i \cdot w_i)$$

Az eredeti perceptron algoritmus ezután a bináris klasszifikációt úgy oldja meg, hogy ha az összeg pozitív akkor a címke 1, ha negatív akkor a címke 0 lesz. Ezt a függvényt $Heavisidestep$-nek hívják. A perceptron ebben a formában csak egy lineáris szeparátorként fogható fel.



4. ábra. Heaviside step függvény

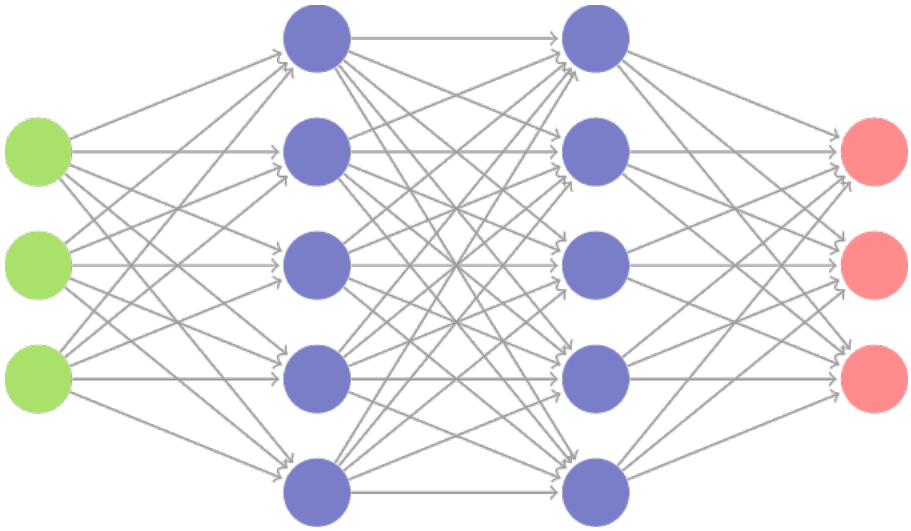
Nugodt szívvel kijelenthető, hogy a perceptron a szakdolgozat legfontosabb modellje, hiszen minden egyes neurális hálózat, amelyet használni fogok ilyen perceptronok hálózataként áll elő.

4.3. Az alap neurális hálózat

A mesterséges neurális hálózatok olyan számítási rendszerek, amelyeket az élőlények agyában működő neurális hálózatok inspiráltak, de nem ugyanolyanok. Ezek a rendszerek "megtanulnak" bizonyos feladatokat, annak alapján, hogy példákat adunk nekik, de nem programozunk bele semmilyen feladatspecifikus algoritmust. Annál a képfelismerési feladatnál, hogy „Van-e macska a képen?" úgy tanulnak, hogy inputként megkapnak képeket, amelyeket már megcímkéztek, vagyis „macskás" vagy „macska nélküli" képekként definiálták azokat, majd ezekből a rendszer megtanulja, hogy önállóan hogyan különböztesse meg a

kétféle képet.

A neurális hálózatok jellemzően három rétegre oszthatóak fel. Az első réteg az input réteg. Ez a réteg kapja meg a nyers inputot, az adatot, amelyből tanítani akarjuk a modellt. A rejtett réteg vagy rétegek, azok amelyek az input és output rétegek között helyezkednek el. Esszenciálisan ezek a rétegek végzik az információ kinyerését. Az utolsó réteg pedig az output, vagy kimeneti réteg, ez a réteg szolgáltatja a modell outputját. Ezekbe a rétegekbe tetszőleges számú neuront pakolhatunk, továbbá a rejtett rétegek száma is szabadon variálható.



5. ábra. Egy neurális hálózat

Ezen a képen az előző bekezdésben tárgyaltak szerint a zöld csomópontok a bementi neuronok, a kékek a rejtett neuronok, és a pirosak a kimeneti neuronok. Látható, hogy ennél a példánál mindössze két darab rejtett réteg van.

Ez a példa egy közönséges *előrecsatolt* vagy $feed-forward$ háló. Jól látható, hogy két egymást követő réteg között bármely két neuron össze van, kötve az ilyen rétegeket *dense* vagy *sűrű* rétegeknek nevezzük.

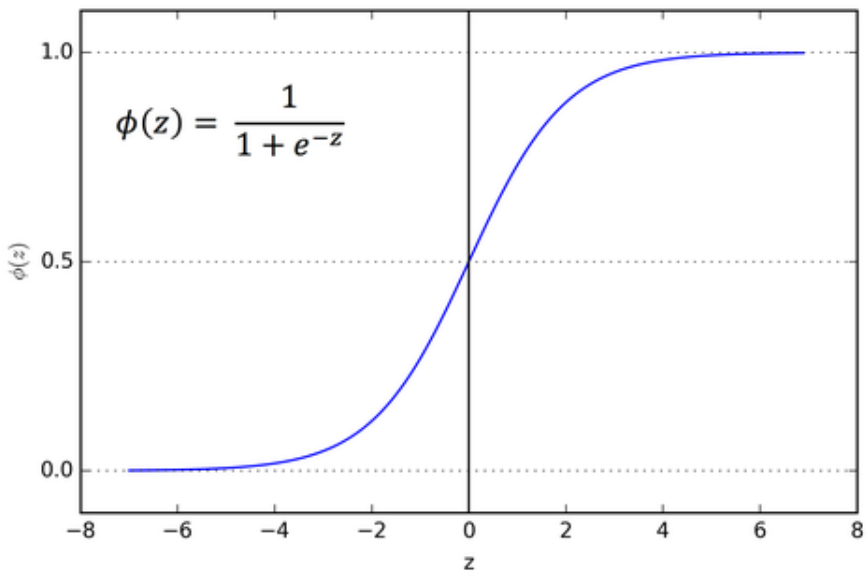
A bemeneti réteg minden neuronjának van valami numerikus értéke. Ezeket az értékeket hívjuk $(x_1, x_2, x_3, ..., x_n)$-nek, de a mi példánkban x_3-ig tart. Azért, hogy a jelölés általánosítható legyen a háló minden neuronjára, be kell vezetnünk még egy indexet. A neuronokat nevezzük x_i^z-nek, ahol az i index jelöli, hogy hányadik neuronról van szó, és a z, hogy hányadik rétegben. Ezekhez a neuronokhoz tartozik valahány él vagy kapcsolat és ezeknek van egy súlyuk. A neuronok közötti éleket nevezzük w_{ij}^z-nek, ahol az i index jelöli azt, hogy hányadik neuronból és a j index, hogy hányadik neuronba fut az él. A felső index azt jelöli, hogy hányadik rétegbe fut az él.

Ezzel a jelölésrendszerrel az első rejtett réteg első neuronjának, vagyis x_1^2-nek az értéke

$$\sum_{i=1}^{3}(x_i^1 \cdot w_{i1}^2).$$

Fontos megjegyezni, hogy eddig pontosan úgy működik egy neuron, mint a perceptron, de most ezt az értéket az úgynevezett *aktivációs* függvénybe adja be, és az aktivációs függvény outputja lesz a neuron végső értéke. Aktivációs függvénynek sok mindent meg lehet adni, de most csak a két legelterjedtebbet fogom ismertetni.

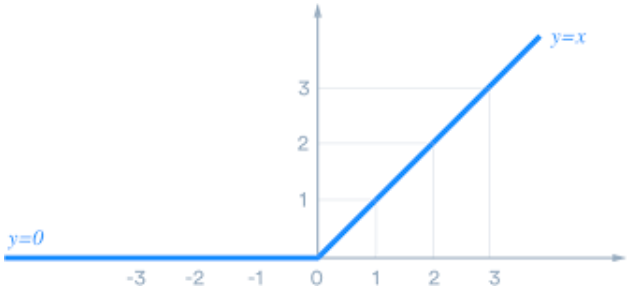
4.4. Aktivációs függvény



6. ábra. Sigmoid[1]

A sigmoid függvény legfőbb előnye, hogy 0 és 1 közé képez. Ezen tulajdonsága miatt nagyon sok olyan esetben használják, amikor valaminek a valószínűségét kell kimenetként megjósolni, hiszen a valószínűség is 0 és 1 között létezik. A sigmoid az egész értelmezési tartományán differenciálható, tehát bármelyik pontjában meg tudjuk állapítani a meredekségét. A sigmoid függvény egyik hátránya, hogy előfordulhat, hogy a sigmoid aktivációs függvényt használó neurális hálózat tanulása megakad. A sigmoid függvény egyik általánosabb verziója

a softmax, amit a többosztályos (multiclass) klasszifikációs feladatánál szokták használni.

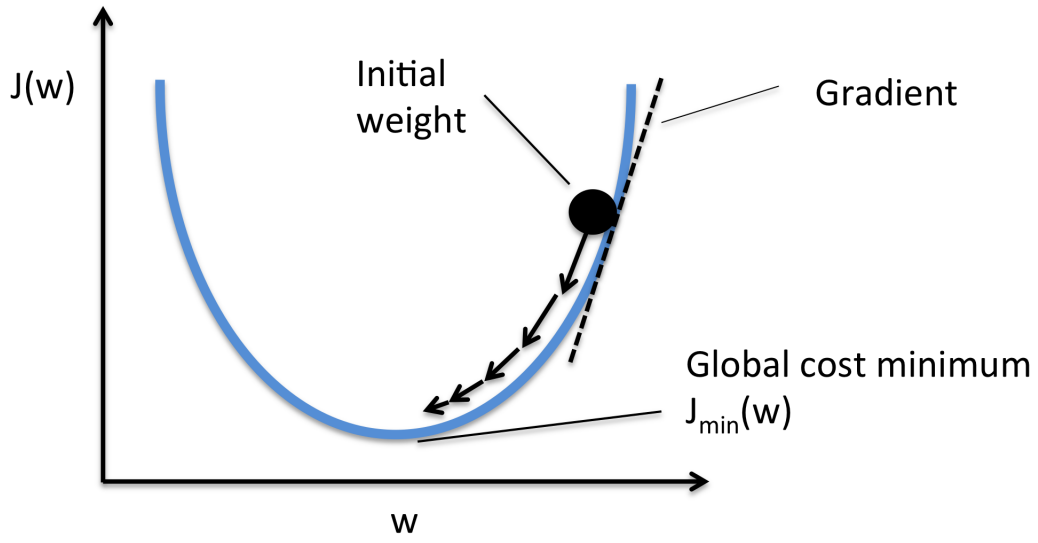


7. ábra. Rectified Linear Unit (ReLU)[2]

A 7. ábrán látható *ReLU* függvényt matematikailag a $f(x) = max(0, x)$ kifejezéssel lehet megadni. A *ReLU* függvényt gyakran használjuk a konvolúciós hálóknál. Nagy előnye, hogy számításigénye alacsony, így gyorsabban lehet tanítani a modellt. A ReLU lineáris $x > 0$-ra és ez azt jelenti, hogy a derivált sosem lesz nulla, vagy végtelen. A ReLU függvény további előnye a sigmoiddal szemben, hogy nem szenved az *eltűnő gradiens* problémában. Vannak esetek amikor az is kedvező lehet, hogy a ReLU egy ritkán aktiváló függvény. Ezalatt azt kell érteni, hogy 0 a kimenet ha $x < 0$, ilyenkor azt mondjuk hogy a neuron nem aktiválódott.[10]

4.5. A gradiens módszer

A gradiens módszer egy széleskörben alkalmazott optimalizációs, iteratív eljárás. A költségfüggvény egy adott pontjában az algoritmus kiszámolja a függvény gradiensét az adott pontban, majd lép egyet az ellenkező irányba. Ezt a két lépést sokszor egymást után elvégezve a hiba konvergál egy lokális minimumba. A legfontosabb paraméter a gradiens módszer használata közben az a *stepsize*, amely azt adja meg, hogy mekkorát lépjen az adott irányba az algoritmus. Amennyiben a stepsize túl kicsi, nagyon lassan konvergál, amennyiben viszont túl nagy, akkor nem is biztos, hogy konvergál az algoritmus.



8. ábra. A gradiens módszer
[13]

Az ábrán a gradiens módszer működése látszik. Az initial weight pont jelenti a paraméterek állapotát egy iteráció elején. Ebben a pontban a gradiens a költségfüggvény minden változója szerinti parciális deriváltakat tartalmazza, a vizualizálhatóság kedvéért itt egy w paraméter van. A gradiens vektor a "legmeredekebb" irányba mutat felfelé, a gradiens módszer ezzel ellentétes irányba lépteti a súlyokat. Ha a W egy vektort jelöl, amiben a modell aktuális $w_1, w_2, ..., w_n$, súlyai vannak és a $J(W)$ költségfüggvényt kell minimalizálni. Ekkor a $w_1, w_2, ..., w_n$ súlyok a következőképpen frissülnek:

$$w_j := w_j - \lambda \frac{\partial J(w)}{\partial w_j},$$

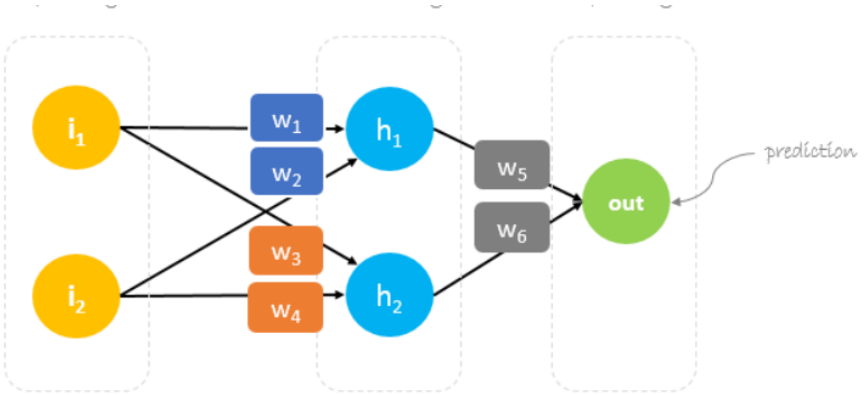
ahol λ paraméterrel lehet a lépés méretét szabályozni.

4.6. A hibavisszaterjesztés

Amikor egy előrecsatolt hálózat feldolgoz egy x vektort, hogy előállítsa belőle az $\bar{y}$ outputot, az információ előrefelé halad a hálózatban. Az x inputok tartalmazzák a kezdetleges információt, amit a háló előre propagál a rejtett rétegeken keresztül egészen a kimenetig, hogy

előállítsa $\bar{y}$-t. Ezt úgy hívják, hogy *információelőreterjesztés*. A hálózat tanítása alatt addig propagáljuk előre az információt, amíg egy skalárt ad vissza, a $J(\theta)$ költségfüggvény értékét. A hibavisszaterjesztés algortimus lehetővé teszi, hogy a hibától visszafelé terjedjen az információ, és ezzel kiszámítsa a gradienst[8].

A neurális hálózatok gradiens módszert használnak a tanuláshoz. Egy neurális hálózatban az állítható paraméterek a súlyok és a bias értékek. Ezeket fogja behangolni a back propagation úgy, hogy a lehető legjobb teljesítményt érjék el. Indítsuk el a hálózatot egy véletlenül inicializált súlyozással. Ezzel a súlyozásssal kiszámoljuk a modell hibáját. A back-propagation az összetett függvények deriválási szabályát alkalmazva kiszámolja a parciális deriváltakat minden él szerint, majd a gradiens módszert alkalmazva úgy változtatja ezeket, hogy a modell egy lokális minimumba konvergáljon.



9. ábra. Back-propagation
[4]

Neurális hálózatoknál a hibavisszaterjesztés az összetett függvények deriváltjára vonatkozó lánc szabállyal történik.A gradiens módszer miatt a w_1 súly frissítése a következőképpen történik:

$$w_1 := w_1 - \lambda \frac{\partial J(w)}{\partial w_1}$$

Ebben a kifejezésben a lánc szabályból adódóan

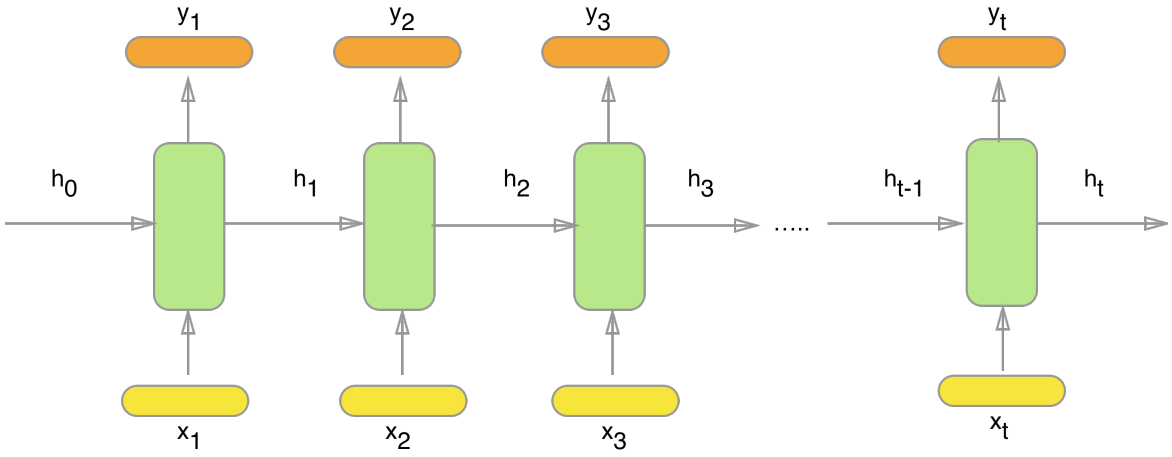
$$\frac{\partial J(w)}{\partial w_1} = \frac{\partial J(w)}{\partial prediction} \cdot \frac{\partial prediction}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1}.$$

Analóg módon kiszámítva minden w_i súlyra megkapható a költségfüggvény gradiense az adott pontban. A kapott gradienssel ellentétes irányba lépve megtörténik a gradiens módszer egy iterációja.

4.7. Rekurrens neurális hálózatok

A rekurrens neurális hálózatok (RNN-ek) olyan hálózati architektúrák, amelyeket a szekvencia feldolgozás motivált. Sok alkalmazása van, többek között a beszédfeldolgozás, beszédgenerálás, szöveges termékértékelésekből elégedettségi szint meghatározása és automatikus képfelirat generálás.

Ha ugyanazokat a lépéseket végeznénk el minden input adatra, az abban az esetben lenne jó irány, ha az adatpontok páronként függetlenek lennének azonos eloszlással. Mivel idősorok és szekvenciák esetében ez nem igaz, ezért jobb architektúrát kellett keresni. Az egyszerű előrecsatolt hálózatoknak van egy közös hátrányuk, nevezetesen, hogy nincs memóriájuk. Ezt úgy kell érteni, hogy ugyanazokat a lépéseket hajtják végre minden egyes adatpontra, anélkül, hogy figyelembe vennék, "emlékeznének" az előző pontokra. Minden output az aktuális inputból és modell paraméterekből nyerhető ki, azaz $\bar{y}_i = f(x_t, w)$. A rekurrens hálózatok is egyesével haladnak végig a szekvencia elemein, de ezeknél minden output érték nem csak az aktuális inputnak függvénye, hanem az előzőké is: $\bar{y}_i = f(x_t, x_{t-1}, w)$

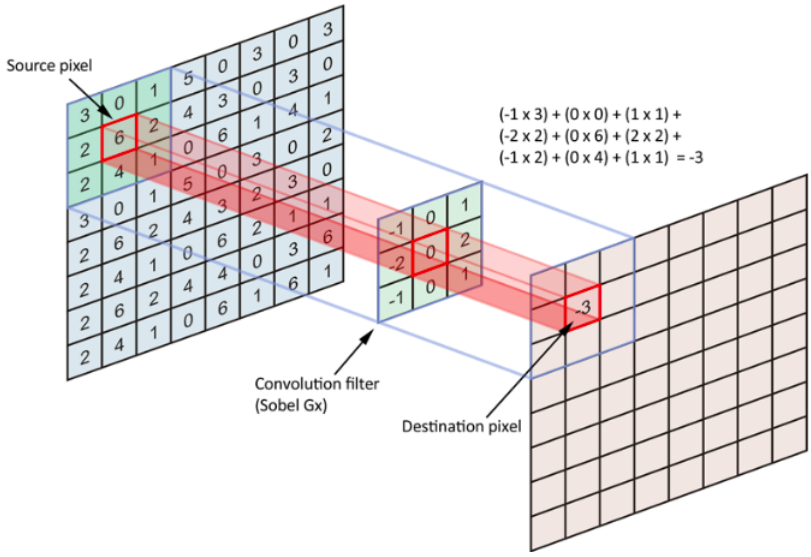


10. ábra. Egy rekurrens háló[9]

A 10. ábrán az $x_1, x_2, ..., x_t$ változók jelölik az inputokat. $y_1, y_2, ..., y_t$ jelölik a modell outputját, és a $h_0, h_1, h_2, h_3, ..., h_t$ tartalmazza az előző értékekből származó információt.

4.8. Konvolúciós hálózatok

A konvolúciós hálók, vagy konvolúciós mély neurális hálózatok olyan speciális architektúrák, amelyeket arra fejlesztettek, hogy olyan adaton dolgozzanak, amelynek valamilyen rácsszerű topológiája van. Példaként felhozhatóak az idősor adatok, amelyeket lehet 1 dimenziós rácshálóként értelmezni, vagy egy kép, amely 2 dimenziós rácsszerkezetet alkot. A konvolúciós hálózatok gyakorlati alkalmazásai nagyon sikeresek voltak. A "konvolúciós hálózat" azt indikálja, hogy a hálózat felhasznál egy matematikai műveletet, a konvolúciót. A konvolúció egy speciális lineáris művelet. A konvolúciós hálózatok, leegyszerüsítve olyan hálózatok, amelyek konvolúciót használnak az általános mátrix szorzás helyett legalább egy rétegben.[8]



11. ábra. A konvolúciós réteg

A 10. ábrán látható egy konvolúciós művelet kiszámítása. A konvolúciós filter végigmegy a bemeneti adaton, ami a példa esetében egy kép pixeleit jelenti, és áttranszformálja azt. Ez a transzformáció a konvolúció műveletével történik, vagyis a két $3x3$-as mátrix nem mátrixszorzással szorzódik össze. A nyers adat körüli $3x3$-mas mátrix legyen az A, a filter

legyen F és a végeredmény legyen p. Ezt a műveletet a következő alakban lehet felírni :

$$p = \sum_{i=1}^{3} \sum_{j=1}^{3} (A_{i,j} \cdot F_{i,j})$$

Ezzel a módszerrel a konvolúciós hálózat meg tudja tanulni az adat lokális viselkedését, lokális feature-eket tud kinyerni az adatból.

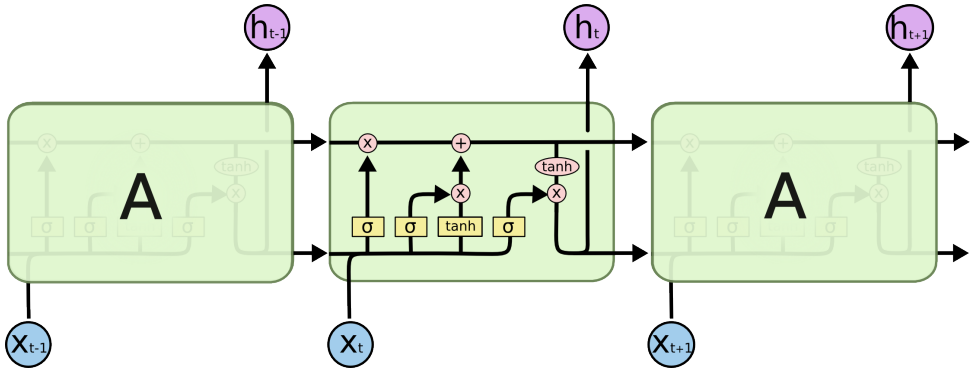
4.9. A Long Short Term Memory hálózatok

Az emberi gondolkodás egyik legfőbb tulajdonsága, hogy a szavakat, mondatokat és gondolatokat kontextusba helyezi, vagyis hozzákapcsolja a környezethez és az előzményekhez, mert az emberi agy rendelkezik memóriával. Folyószövegek olvasásánál az emberi agy felhasználja a már elolvasott, processzált információt ahhoz, hogy egy következő szónak az információ tartalmát minél pontosabban tudja értelmezni. A gondolatok és információk megmaradnak.

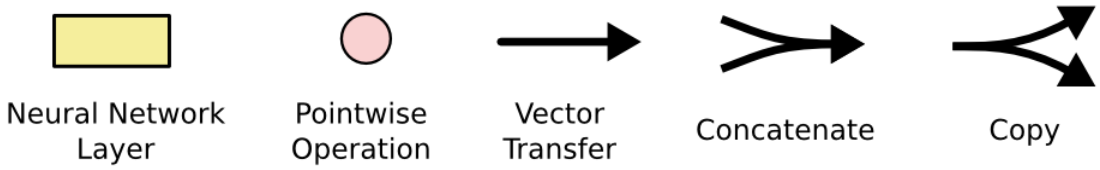
Ezzel szemben a tradícionális neurális hálózatok erre nem képesek, és ez az egyik legnagyobb hiányosságuk. Vegyük például azt a feladatot, amelyben klasszifikálni akarjuk, hogy egy film során éppen milyen jelenet játszódik. Ebben az esetben egy tradícionális neurális hálózat nem tudja felhasználni az előző jelenetek tartalmát annak megállapítására, hogy egy későbbi időpontban mi történik Ennek a problémának az orvoslására fejlesztették ki a rekurrens hálózatokat.

A Long Short-Term Memory hálózatok, röviden "LSTM" hálózatok, olyan különleges fajtája a rekurrens hálózatoknak, amely képes hosszútávú memóriát tárolni. Először Hochreiter és Schmidhuber fogalmazta meg egy ilyen háló működését 1997-ben, és azóta számos kutató finomította, és találta meg azokat az alkalmazásait, amiknél nagyon jól működik.

Az LSTM hálózatokat arra tervezték, hogy hosszú távú memóriával rendelkezzenek. Ezeknek a hálózatoknak az alapértelmezett viselkedése, hogy eltárolják az információt (az előzményeket) hosszabb időszakokra.



12. ábra. Egy LSTM lánc[12]



13. ábra. A jelölések[12]

A 13. és 14. ábrán a LSTM hálózatok lényegi elemeit látjuk. A 13. ábrán látjuk a működést, a 14. ábrán pedig az ahhoz tartozó notációt. A 13. ábrán minden nyíl egy vektort tartalmaz, amelyet vektort az egyik csomópont kimenetéből egy másik csomópont bemenetére visz át. A rózsaszínű körök a pontonkénti műveleteket jelölik. Az ábrán látható sárga téglalapok neurális hálózati réteg, amelyet a hálózat tanul. Amikor egy fekete vonal kettéválik, akkor mindkét irányba viszi a teljes vektort, amelyet eredetileg tartalmazott. Amikor két nyíl összeolvad, az konkatenációt jelent.

A LSTM hálózatok kulcs ötlete a horizontálisan (felül) haladó cella állapot. A cella állapot olyan mint egy futószalag vagy "conveyor belt": végighalad az egész láncon és csak minimális lineáris változtatásokat hajt végre rajtuk. Az információ nagyon könnyen keresztül áramolhat rajta változatlanul. A LSTM cellának megvan az a képessége, hogy megváltoztassa, hozzáadjon, vagy elvegyen információt. Ezeket a változtatásokat az úgynevezett "kapuk" szabályozzák. A kapuk biztosítják a lehetőséget arra, hogy opcionálisan átengedjék az információt. Ezek a kapuk egy szigmoid aktivációs függvénnyel ellátott neurális hálózati rétegből, és pontonkénti szorzás műveletből állnak.

A sigmoid réteg kimenete egy 0 és 1 közötti szám, aminek az a jelentése, hogy mennyit engedjenek át a kapuk az egyes komponensekből. A 0 érték azt jelenti, hogy semennyit, az 1 érték pedig azt, hogy az egészet. Egy LSTM hálózatban három ilyen kapu van, amely a cella állapot "karbantartásáért" felel.

5. A megépített modell

Az a modell, amely leginkább illik a szakdolgozat feladatához, felhasználja mind a konvolúciós és LSTM hálózatokat is. A feladat egy több, vagy egyváltozós idősor előrejelzése. 2019-ben koreai kutatók nagyon jó eredményeket értek el a lakossági áramfelhasználás előrejelzésében. Az általuk épített modell képezi az alapját a szakdolgozat által vizsgált modellnek is. A neurális hálózatnak lesznek konvolúciós rétegei, ezek nyerik ki a hasznos lokális karakterisztikákat és a LSTM rétegek teszik lehetővé az adatok időbeliségének felhasználását. Ebben a fejezetben a megvalósított modell informatikai hátteréről lesz szó. A neurális hálózat python programnyelven lett megírva. Ennek a programnyelvnek sok a gépi tanulási által ihletett könyvtárát felhasználtam. Az adatok beolvasását a *pandas* nevű könyvtár segítségével végeztem. A pandas legnagyobb előnye, hogy a számos előre implementált függvény mellett a háttérben C fut, ezért sokkal gyorsabb mint a python saját függvényei. Az adatok preprocesszálását és tisztításában a pandas mellett egy másik, a *numpy* könyvtár függvényei segítettek. A hiányzó adatok pótlásánál használt KNN módszert egy kifejezetten gépi tanulásra íródott könyvtár a $Sci - kitlearn$ implementációjával történt.

A neurláis hálózatot a két legnagyobb Deep Learning-re íródott könvytár egyikével, a $Keras$-al végeztem. Ebben a könvytárban a különböző rétegek vannak implementálva, így a fejlesztőnek, csak a sorrendet és a rétegek fajtáját kell kitalálnia, ezután viszonylag egyszerűen lehet kódolni. Az általam használt rétegek a *keras* könyvtárból többek között a Conv, az LSTM és a TimeDistributed rétegek.

A neurális hálózat betanítása után a hibafüggvényeket ismét a $Sci - kitlearn$ könvtár beépített függvényeivel számolódtak. Az általam elkészített ábrákat a *matplotlib* nevezetű könyvtár segítségével készítettem el.

Érdemes megemlíteni, hogy a fentebb említett könvtárakban előre megírt függvények nélkül az implementálás feladata sokkalta nehezebb lett volna.

5.1. A neurális háló

A megépített neurális háló architektúra a következő rétegekből áll: az input vektor egy konvolúciós rétegen megy keresztül először. A konvolúciós réteg 64 filtert használ, és 2 hosszúságú a kernel. Ebben a rétegben a ReLU vagyis Rectified Linear Unit aktivációs függvény van, a konvolúciók outputja még ezen a függvényen megy keresztül. Ezután

egy pooling réteg következik, amely a konvolúciós hálózatokban gyakran előfordul a zaj csökkentésére. A pooling 2 hosszúságű vektorokkal dolgozik és max poolingot használ. A maxpooling azt jelenti, hogy a két vektor közül a nagyobb lesz csak benne a kimenetben. A neurális hálózatban ez a három réteg ismétlődik újra, hogy még magasabb reprezentációs szintű karakterisztikákat tudjon megtanulni. Ez után következnek a LSTM rétegek tangens hiperbolikusz aktivációs függvénnyel. A LSTM rétegek után a sűrű rétegek következnek. Előbb egy 32 neuronból álló, majd egy 60 neuronból álló sűrű rétegen folyik keresztül az információ. A felvázolt modell összesen 192.988 paraméterrel rendelkezik.A következő táblázatban az egyes rétegek fontos paraméterei láthatóak.

Réteg típusa	Filterek száma	Kernel mérete	Stride
Konvulúció réteg	64	(2,1)	1
Aktivációs függvény (ReLU)	-	-	-
Pooling (Max)	-	(2,1)	2
Konvolúciós	64	(2,1)	1
Aktivációs függvény (ReLU)	-	-	-
Pooling (Max)	-	(2,1)	2
Long Short Term Memory	-	-	-
Aktivációs függvény (Tanh)	-	-	-
Dense (32)	-	-	-
Dense (60)	-	-	-

6. Kiértékelés

6.1. A modell tanítás és kiértékelés módszertana

Ebben a fejezetben röviden összefoglalom, hogy miként érdemes megépíteni és kiértékelni egy gépi tanulási feladatot.

Az egyik legfontosabb dolog, hogy a feladat jól legyen specifikálva. Ha bármilyen apró részlet nincs egyértelműen letisztázva, akkor az később problémákhoz vezethet. Ezek elkerülése érdekében érdemes előre alaposan megtervezni az egész folyamatot. Amennyiben a modell üzleti felhasználásra készül, akkor az is lényeges, hogy tisztában legyünk annak üzleti értékével is.

A feladat pontos meghatározása után lehet elkezdeni a modell építését. Elsőként több részre kell bontani a rendelkezésre álló adatot. Ez jellemzően három részt jelent: a *Tanító*, a *Teszt* és a *Validációs* halmazokat. Ezek a halmazok egymástól diszjunktak, és uniójuk kiadja a teljes adathalmazt. Ezeknek az aránya függ a feladattól és a rendelkezésre álló adat mennyiségétől, de jellemző megválasztás lehet a $\frac{3}{5}, \frac{1}{5}, \frac{1}{5}$

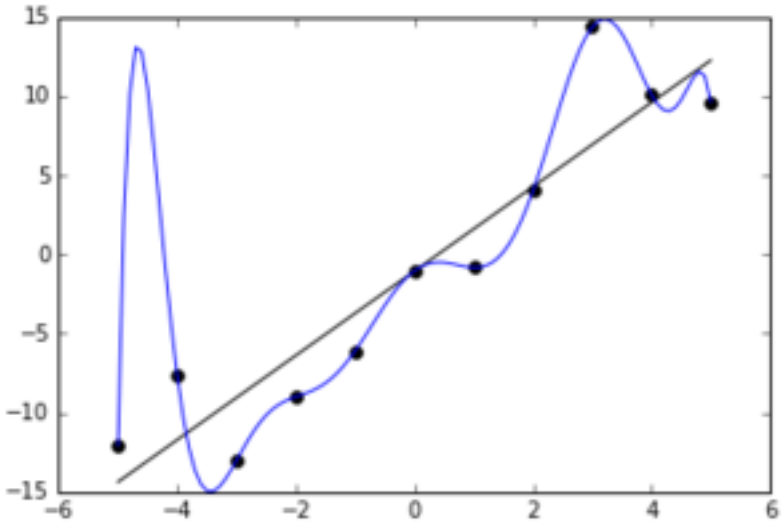
Az összes rendelkezésre álló adat $\frac{3}{5}$-e tanító halmaz. A megépített modell ezeket az adatpontokat kapja meg inputként, és ezeken végzi el a tanulást. Fontos viszont, hogy a modell kiértékelését, tesztelését semmiképp sem szabad ezen az adathalmazon végezni, hiszen a modell lényege, hogy általános mintázatokat tanuljon, vagyis addig ismeretlen adatpontokra működjön jól.

Az $\frac{1}{5}$-ét az adatnak a teszt halmazába tesszük, az egyes modelleket ezeken az adatokon próbáljuk ki és értékeljük ki valamilyen költségfüggvénnyel. Ez a halmaz diszjunkt a tanító halmaztól, a modell számára ismeretlen adatpontokat tartalmaz, így jól mérhető, hogy mennyire teljesít jól az algoritmus.

Az utolsó $\frac{1}{5}$-e az adatnak a validációs halmazba kerül. Első meggondolásra nem biztos, hogy világos, miért van szükség validációs és teszt halmazra is. A válasz egyszerű. Amikor hangoljuk a használt modell hiperparamétereit, folyamatosan úgy állítgatjuk, hogy a teszt halmazon minél jobban teljesítsen. Ez akarva-akaratlanul is azt eredményezi, hogy a hiperparamétereket rátanítjuk az adathalmazra. Emiatt, a hiperparaméterek nem az általános adatpontokra fognak jól teljesíteni, hanem csak a teszt halmazban lévő pontokra. Ezért kell a validációs halmaz, amelyen akkor értékeljük ki a modellt, amikor a teszt halmazon már elfogadható eredményt értünk el.

6.2. A kiértékeléssel kapcsolatos fogalmak

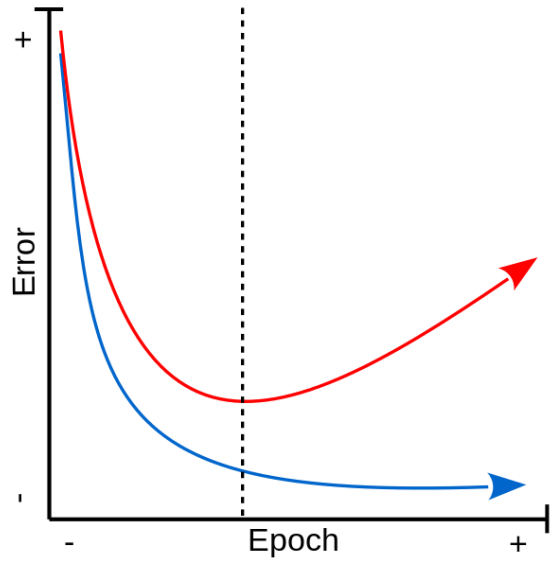
Ebben a fejezetben a gépi tanulási feladatoknál jellemző fogalmak, problémák tárgyalására kerül sor. A legfontosabb dolog a tanítási feladatnál, hogy a modell általános legyen. Az overfitting azt jelenti, hogy a felépített módszer túlságosan rátanult a tanító halmazra. Ezt érdemes úgy elképzelni, hogy $n+1$ adatpontra mindig állítható egy n-edfokú polinom, olyan módon, hogy minden adatponton pontosan átmenjen. Habár így a modell hibája 0 lesz a tanító halmazon, az algoritmus elhasal új adatpontok esetében.



14. ábra. Overfitting

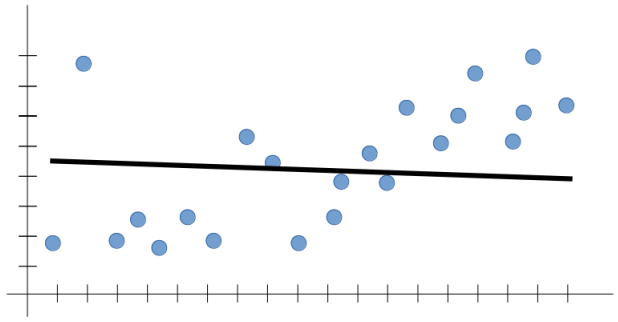
Az ábrán jól látható, hogy az egyenes nem illeszkedik tökéletesen az adatpontokra, de jól megfogja a tendenciát. A kék görbe tökéletesen illeszkedik minden pontra, mégsem jó megközelítés.

Ha megvizsgáljuk a tanulási iterációk alatt, hogy hogyan változik a tanító halmazon számolt hiba és a teszt halmazon számolt hiba, és ezt ábrázoljuk, akkor overfitting esetén a következő ábrát kapjuk:



15. ábra. Tanulási folyamat

Ahogy az x tengelyen haladunk előre, egyre több iterációt elvégezve egy ponton túl hiába csökken a tanító hiba, a teszt hiba nőni fog.
Az underfitting az overfitting ellentéte. Akkor történik underfitting, ha a kiépített modell nem elég komplex ahhoz, hogy megtalálja az adatban lévő összefüggéseket.



16. ábra. Underfitting

Az ábrán az illesztett egyenes jól láthatóan nem tudja pontosan/ megfelelően jól megfogni az adatban rejlő tulajdonságokat, vagyis ekkor underfitting történik. Az underfitting esetében a tanulási folyamat ábráján az látszana, hogy a tanító és teszt halmazon is nagy a hiba, és két hiba nagyságrendileg megegyezik.

Összegezve tehát nem mindíg könnyű megtalálni a tökéletes komplexitású modellt. Ha eltúlozzuk , és a modell kifejezőereje sokkal nagyobb a feladat komplexitásánál, akkor túltanulunk, amennyiben viszont sokkal kisebb, akkor nem fogjuk tudni a modellt jól illeszteni a tanító adatra sem. Akkor lehet a legjobb eredményt elérni, ha sikerül megtalálni a két állapot közötti ideális pontot.

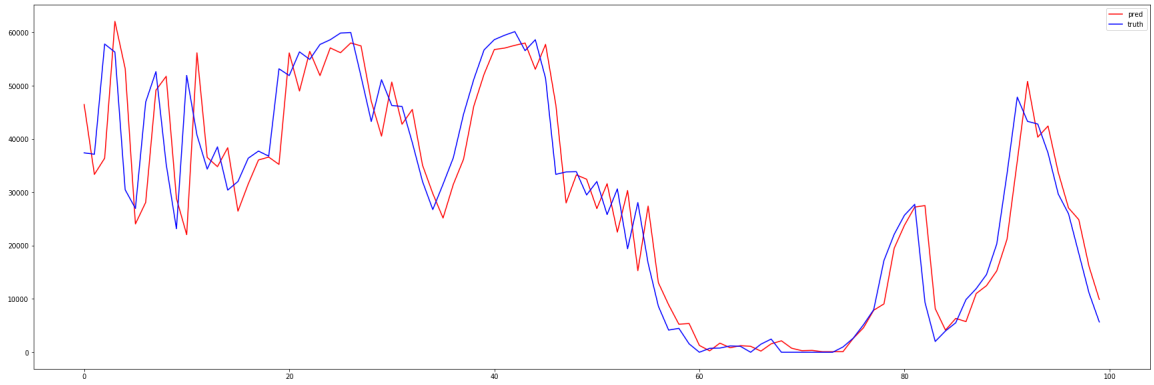
6.3. Kiértékelés

Ebben a fejezetben a megalkotott modell által elért eredményekről lesz szó. Az idősor előrejelzési feladatok többféleképpen lehetnek motiváltak. Egyes tőzsdei ügyleteknél jellemzően hosszab távra kivánjuk előrejelezni az árfolyamok alakulását, napok, hetek, akár hónapokra előre. A szélenergia termelésbecslésnél viszont a feladat nem motiválja, hogy hosszú távra előre becsüljük, hanem sokkal inkább azt, hogy a következő napokban mekkora lesz a termelés. Ennek megfelelően a predikciók során viszonlag rövidtávú előrejelzéseket adtunk.

A következő táblázatban a különböző visszatekintési paraméterekkel elvégzett méréseket lehet látni.

Visszatekintési idő	R^2	RMSE	Átlagos reziduális hiba
2	0.77432458395763	8553.1638	255.58081581
4	0.8198949694576432	6569.9981	199.1236953
6	0.8457704999388764	5734.8763	110.92506213
8	0.8375807259155191	6903.3621	167.19378246
12	0.6893842721356885	12865.6395	489.19734613
24	0.6375807259155191	15823.4928	601.58081581

A táblázatból látszik, hogy az optimális visszatekintési paraméter beállitás a hat óra. Ez azt jelenti, hogy a neurális hálózat a predikciót megelőző hat mérésből tud a legjobban előrejelezni. A következő ábrán a legjobb mérés által elért predikciót és a valós teszt értékeket lehet látni.



17. ábra. A predikciók hat órás visszatekintéssel

A 17. ábrán egyértelműen látszik, hogy a neurális hálózat milyen jól tudja prediktálni tudja a termelt áram mennyiségét. Amennyiben valaki valós időben látja az adatokat, és abból végez előrejelzést a közeli jövőre, elég nagy magabiztossággal hagyatkozhat erre a modellre.

6.4. Fejlesztési lehetőségek

Érdemes belegondolni, hogy milyen módokon lehetne még jobb eredményt elérni, mik azok a dolgok amikkel tovább lehetne fejleszteni a modellt. Nagyon hatékony dolog lehet a méréseket sűrűbbé tenni. Ez egyrészt azt is jelentené, hogy egységnyi idő alatt több adatpontot lehet gyűjteni, másrészt finomabb, részletesebb információkat tartalmazhat a neurális hálózat számára. Ezen kívül az időjárási körülményeket, előrejelzéseket is fel kellene használni, hiszen végsőroron a legmeghatározóbb a termelés szempontjából a szél sebessége. Jelen szakdolgozatba ezek több okból sem fértek bele. Egyrészt az adat órás felbontású volt, és nem voltak meg az eszközök az önálló adatgyűjtésre. Ebből kifolyólag az interneten található adatokra kellett hagyatkozni, amihez nem volt csatolva időjárási előrejelzés. A szakdolgozat elején megfogalmazott feladatra sikerült megoldást adni. A legnehezebb feladat a neurális hálózat megépítése volt, amit kellő ráfordítással könnyen tovább lehet fejleszteni, és akár ipari-gazdasági környezetben is bevezethető.

Hivatkozások

[1] Activation functions in neural networks. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. Accessed: Sep 6, 2017.

[2] Activation functions in neural networks. https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7. Accessed: Nov 30, 2017.

[3] Gradient based learning applied to document recognition.

[4] Backpropagation step by step, 2019. https://hmkcode.com/ai/backpropagation-step-by-step/.

[5] Ratnadip Adhikari. An introductory study on time series modeling and forecasting. 2013.

[6] James Blackmon. Rosenblatt's perceptron. http://jcblackmon.com/general/rosenblatts-perceptron/.

[7] Andriy Burkov. *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[9] Simeon Kostadinov. How recurrent neural networks work, 2017. https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7.

[10] Ayyüce Kızrak. Comparison of activation functions for deep neural networks. https://towardsdatascience.com/comparison-of-activation-functions-for-deep-neural-networks-706ac4284c8a, 2018.

[11] Tom M. Mitchell. *Machine learning, International Edition*. McGraw-Hill Series in Computer Science. McGraw-Hill, 1997.

[12] Prabhu. Understanding lstm networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015.

[13] Sebastian Raschka. Gradient descent and stochastic gradient descent, 2017. http://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization.

[14] J. S. Denker D. Henderson R. E. Howard Y. LeCun, B. Boser. backpropagation applied to hand written zipcode recognition.