VENTSPILS AUGSTSKOLA
INFORMĀCIJAS TEHNOLOĢIJU FAKULTĀTE

MAĢISTRA DARBS

DAUDZKANĀLU EEG SIGNĀLU APSTRĀDES PIELIETOJUMS, IZMANTOJOT DZIĻĀS APMĀCĪBAS ALGORITMUS

Autors

Ventspils Augstskolas
Informācijas tehnoloģiju fakultātes
maģistra studiju programmas
„Datorzinātnes”
2. kursa studente
Linda Kalasnikova
Matr.nr. 17080003

_____
(paraksts)

Fakultātes dekāns doc. Dr.phys. Māris Ēlerts

_____
(paraksts)

Zinātniskais vadītājs Dr. sc. comp. Gundars Bergmanis-Korāts

_____
(paraksts)

Recenzents _____
(ieņemamais amats, zinātniskais nosaukums, vārds, uzvārds)

_____
(paraksts)

Ventspils
2019

Saturs

Anotācija

Darba nosaukums: Daudzkanālu EEG signālu apstrādes pielietojums, izmantojot dziļās apmācības algoritmus.

Darba autors: Linda Kalašņikova

Vadītājs: Dr. sc. comp. Gundars Bergmanis-Korāts

Darba apjoms: 59 lpp., 29 attēli, 9 tabulas, 51 bibliogrāfiskās norādes, 6 pielikumi.

Atslēgas vārdi: MIEGA EEG, HIPNOGRAMMA, KONVLOŪCIJA,ILGI-ĪSLAICĪGĀ ATMIŅA.

Maģistra darba ietvaros tiek apskatīta tēma par smadzenes-dators saskarni (angļu: Brain-Computer Interface), kuras pamatā ir daudzkanālu elektroencefalogrammas (EEG) signālu apstrāde.

Tēma ir aktuāla un nozīmīga pacientiem, kas cieš no miega apnojas. Šis miega traucējums izpaužas kā pēkšņa elpošanas apstāšanās miega laikā. Miega apnoja ietekmē miega fāzes. Klīnikā tās, savukārt, nosaka vizuāli analizējot elektroencefalogrammas datus. Šī iemesla dēļ ir nepieciešamība pēc automatizēta miega fāžu novērtējuma, kas atļautu veikt miega novērtējumu daudz efektīvāk.

Pēdējā laikā augstu popularitāti guvušās dziļās apmācības metožu pielietojums sniedz iespēju apstrādāt datus gan industrijā, gan medicīnā. Šī iemesla dēļ maģistra darba ietvaros tiks apskatītas un pielietotas dažādas dziļās apmācības metodes miega fāžu klasificēšanā.

Eksistē risinājumi (aprakstīti [1] un [2]), kuriem nepieciešama veikt datu priekšapstrādi, lai izgūtu īpašības. Šai problēmai ir arī risinājumi, kuri pielieto dziļās apmācības metodes uz EEG datiem [3], [4], taču mazo datu dēļ rezultāti nav pietiekoši labi. Tādēļ autore apskata dziļās apmācības modeļus un pieejamos datus, kurus var pielietot modeļu apmācībai.

Darba mērķis ir miega EEG laika rindu mērījumos atpazīt nomoda un miega fāzes.

Darbā tiek izvirzīti sekojoši uzdevumi, lai sasniegtu mērķi:

1. Izpētīt pieejamo miega EEG signālu datubāzes.
2. Izpētīt un iepazīties ar esošajiem dziļās apmācības risinājumiem laika rindu apstrādē.
3. Izstrādāt metožu un algoritmu kopumu laika rindu signālu klasifikācijā.
4. Sagatavot testa datus algoritmu apmācībai.
5. Veikt modeļu apmācību un analizēt iegūtos rezultātus.

Anotation

Name: Multichannel approaches for EEG processing using Deep Learning.

Author: Linda Kalašņikova

Supervisor: Dr. sc. comp. Gundars Bergmanis-Korāts

Workload: 59 pages, 29 images, 9 tables, 51 bibliographical references, 6 attachments.

Key Words: SLEEP EEG, HYPNOGRAM, CONVOLUTION, LONG SHORT-TERM MEMORY.

In Master Thesis will be reviewed topic which is related to the Brain-Computer Interface that is based on the processing of multi-channel electroencephalogram (EEG) signals.

The topic is meaningful and topical for patients suffering from sleep apnea. This sleep disorder is manifested as a sudden stop of breathing during sleep. Sleep apnea affects the sleep phases. In the clinic they are determined by visual analysis of electroencephalogram data. For this reason, there is a need for an automated sleep phase assessment, which would allow for a more effective sleep assessment.

Recently, the use of highly popular deep training methods provides the opportunity to process data in both industry and medicine. For this reason within the Master's thesis, different deep learning methods for classifying sleep stages will be considered and applied.

There are solutions (described in [1] and [2]) that require data pre-processing to retrieve features. There are also solutions to this problem that use deep learning methods on EEG data [3], [4] but due to small data the results are not good enough. Therefore, the author will explore different deep learning models and available data that can be used for model training.

The aim of the work is to recognize wake and sleep stages in Sleep EEG time series measurements.

The following tasks are set to achieve the aim:

1. Explore available sleep EEG signal databases.
2. Explore and get familiar with existing deep learning solutions for processing time series.
3. Develop a set of methods and algorithms for time series signal classification.
4. Prepare test data for algorithm training.
5. Conduct training, evaluate the obtained results.

Izmantotie saīsinājumi un termini

EEG - Elektroencefalogrāfija (no angļu Electroencephalography)
REM - Ātrās acu kustības (no angļu Rapid eye movement)
NREM - Ne ātrās acu kustības (no angļu Non Rapid eye movement)
RNN - Rekurentie neironu tīkli (no angļu Recurrent Neural Network)
LSTM - Ilgi-īslaicīgā atmiņa (no angļu Long Short-Term Memory)
CNN - Konvolūciju Neironu Tīkli (no angļu Convolution Neural Network)
tanh - Hiperboliska tangentes funkcija (no angļu Hyperbolic Tangent function)
EDF - Eiropas Datu Formāts (no angļ. European Data Format)
MSE - Vidējā kvadrāta kļūda (no angļ. Mean squared error)
1D - Viendimensionāls
2D - Divdimensionāls
3D - Trīsdimensionāls

Ievads

Daudzkanālu signālu apstrāde ir fundamentāla datu analīzes problēma, kas sastopama inženierijas jomās kā, piemērām, dažādās industriālās aplikācijās, tā arī medicīnā [5].

Šī darba ietvaros tiek apskatīta tēma saistīta ar smadzenes-dators saskarni (angļu: Brain-Computer Interface) ar mērķi miega EEG laika rindu mērījumos atpazīt nomoda un miega fāzes.

EEG signālus pamatā lieto smadzeņu stāvokļa un smadzeņu traucējuma monitoringam un diagnostikai. Taču šādus signālus iespējams izmantot arī smadzenes-dators saskarnē, kur EEG signālos novērotā smadzeņu aktivitāte tiek interpretēta datoram saprotamās komandās.

Tēma ir aktuāla un nozīmīga pacientiem, kas cieš no miega apnojas. Šis miega traucējums izpaužas kā pēkšņa elpošanas apstāšanās miega laikā. Tai ir tendence pasliktināties laika gaitā un dažos gadījumos tā var apdraudēt dzīvību. Miega apnoja ietekmē miega fāzes. Klīnikā tās, savukārt, nosaka vizuāli analizējot elektroencefalogrammas datus. Šī iemesla dēļ ir nepieciešamība pēc automatizēta miega fāžu novērtējuma, kas atļautu veikt miega novērtējumu daudz efektīvāk.

Pēdējā laikā augstu popularitāti guvušās dziļās apmācības metožu pielietojums sniedz iespēju apstrādāt datus gan industrijā, gan medicīnā. Šī iemesla dēļ maģistra darba ietvaros tiks apskatītas un pielietotas dažādas dziļās apmācības metodes miega fāžu klasificēšanā.

Eksistē risinājumi (aprakstīti [1] un [2]), kuriem nepieciešama veikt datu priekšapstrādi, lai izgūtu īpašības. Šai problēmai ir arī risinājumi, kuri pielieto dziļās apmācības metodes uz EEG datiem [3], [4], taču mazo datu dēļ rezultāti nav pietiekoši labi. Tādēļ autore apskata dziļās apmācības modeļus un pieejamos datus, kurus var pielietot modeļu apmācībai.

Rezultātā iegūtās zināšanas un algoritmu pielietojums var tikt pielietots arī citu sensoru datiem, tādējādi radot vērtīgu zināšanu un metožu bāzi pētniecībai un pielietojumiem citās industrijas nozarēs.

Lai sasniegtu darba mērķi nepieciešams veikt sekojošus uzdevumus:

1. Izpētīt pieejamo miega EEG signālu datubāzes.
2. Izpētīt un iepazīties ar esošajiem dziļās apmācības risinājumiem laika rindu apstrādē.
3. Izstrādāt metožu un algoritmu kopumu laika rindu signālu klasifikācijā.
4. Sagatavot testa datus algoritmu apmācībai.
5. Veikt modeļu apmācību un analizēt iegūtos rezultātus.

Maģistra darbs sastāv no 3 nodaļām. 1. nodaļā tiek apskatīts miega EEG un miega fāzes - Nomoda, REM, N1, N2 un N3. 2. nodaļā apskata dziļās apmācības algoritmus, kas sevī ietver Konvolūcijs Neironu Tīklus, Rekurentos Neironu Tīklus un Ilgo Īstermiņa Atmiņu. 3. nodaļā tiek aprakstīta datu kopas sagatavošana, tīklu apmācība, testēšana un rezultātu attēlošana.

1. Miega EEG

Miega EEG ir smadzeņu elektriskās aktivitātes ieraksts, kamēr cilvēks ir nomoda un miega stāvoklī. Miegs tiek uzskatīts par mainītu vai samazinātas apziņas stāvokli.

Miega EEG tests parasti tiek veikts slimnīcā, izmantojot standarta EEG ierīci [6]. Šajā procedūrā izmanto elektrodus, kas reģistrē elektriskā potenciāla izmaiņu uz galvas virsmas pret references elektrodu. Elektrodi tiek novietoti uz galvas ādas. Kā daļu no pārbaudes, var veikt pacienta filmēšanu, lai palīdzētu diagnosticēt slimību. Pirms testa pacients var lietot medikamentus aizmigšanai. Pārbaude ilgst vienu vai divas stundas. Kad pacients ir pamodies, viņš parasti var doties mājās.

Ir dažāda veida miega EEG ieraksti:

1. **Bezmiega EEG** - Lai veiktu šo pārbaudi, nepieciešams izvairīties no gulēšanas vai arī gulēt tikai noteiktas stundas naktī pirms testa. Parasti ārsti specificē kādu režīmu pacientam vajadzētu ievērot pirms EEG.

2. **Dabiska miega EEG** - Lai veiktu šo testu nav nepieciešams īpašs režīms, cilvēks var aizmigt dabiski un nav nepieciešams pirms testa gulēt mazāk nekā parasti.

No EEG izmeklējumiem ir iespējams noteikt smadzeņu viļņu paternus, kas rodas miega laikā un nomodā. Šajos paternos novērojamās izmaiņas var parādīt reģionu, kas ir bojāts un iespējams ir lēkmju rašanās iemesls. EEG var arī uzrādīt pīķus vai asus viļņus, kas liecina par lēkmju risku [7]. Tā kā tie rodas nejauši, tos var vai nevar redzēt vienā īsā ieraksta laikā.

Procentuāli laiks, ko cilvēks pavada miegā tiek saukts par miega efektivitāti [8]. Miega efektivitāti aprēķina, dalot kopējo miega laiku ar kopējo laiku gultā. Miega efektivitāte, kas pārsniedz 90%, tiek uzskatīta par normālu un optimālu.

1.1. Miega fāzes

Miega režīms ir sadalīts divos galvenajos miega stāvokļos - ātrās acu kustības un ne ātrās acu kustības stāvoklī. NREM ir sadalīts trīs posmos: N1 fāzē, N2 fāzē un N3 fāzē. N1 fāze tiek uzskatīta par miera pārejas posmu, šo fāzi raksturo īpašības, kuras var novērot gan nomoda stāvoklī, gan miega stāvoklī.

Nomoda

Šajā fāzē muskuļu aktivitāte un izziņas reakcija ir augsta un modrība ir atkarīga no daudziem dažādiem faktoriem. Biežu kustību un muskuļu aktivitātes dēļ EEG signālus var būt grūti lasīt.

EEG signālus šai periodā raksturo parasti augstas frekvences ar jauktu spriegumu 1.1.. Šajā posmā Elektromiogrāfija uzrāda to, ka amplitūdas ir augstas un gaisa plūsma mainās atkarībā no aktivitātes līmeņiem [10].

Nomoda fāzei ir raksturīgi alfa viļņu paterni. Alfa viļņus var novērot, tad kad cilvēks relaksējas un atrodas nomoda stāvoklī ar aizvērtām acīm.

Pamošanās pēc miega sākuma ir laiks minūtēs, kuru pavada cenšoties aizmigt, kad ir sasniegts miega stāvoklis. Šo fāzi bieži vien raksturo zemāks aktivitātes līmenis nekā dienas laikā. Pieauguša cilvēka vidējais miega aiztures laiks ir 10-20 minūtes.

Nomoda stāvoklis sastāda divas trešdaļas no 24 stundu cikla vai aptuveni 16 stundas dienā [8]. Vidēji cilvēks guļ vismaz 90% no miega perioda, tas ir laiks no miega sākuma līdz pamošanās brīdim. Citiem vārdiem sakot, šai fāzei vajadzētu aizņemt mazāk nekā 10% no miega perioda.

Ātrās acu kustības

REM fāzi bieži sauc par paradoksālu miegu, jo smadzeņu darbība līdzinās smadzeņu darbībai laikā, kad esam nomodā, bet muskuļi ir paralizēti. Muskuļu tonuss šajā posmā ir ļoti zems. REM fāze miega sākuma posmā parasti ir īsāka un kamēr miega periods turpinās, arī laiks, kas tiek pavadīs R fāzē, palielinās. Miega fāze REM ir saistīts ar atmiņu, izzināšanas spēju un ķermeņa atjaunošanos miega laikā. REM fāzes laikā notiek prāta atjaunošana. Pētījumi liecina, ka atmiņa pasliktinās, kad REM fāze samazinās.

REM fāzi raksturo galvenokārt ātras acu kustības. Fiziskais REM ir fāze, kad acis ātri pārvietojas uz priekšu un atpakaļ. Tomēr tonizējošajā REM periodā ir diezgan izplatīts tas, ka nav redzamas ātras acu kustības. Zāģveida viļņi elektroenciogrammās ir raksturīga viļņu forma, kas bieži parādās REM fāzē un ir visbiežāk novērojama tieši fiziskajā REM fāzē.

Citas REM fāzes raksturojošās lietas ir muskuļu atonija, zemsprieguma, jauktas frekvences EEG modelis 1.1., ķermeņa temperatūras samazināšanās, mainītas elpošanas modelis. Ja pacienti ir pamodušies no šī miega posma, viņi bieži vien var atcerēties sapņus.

REM fāze ir apmēram 20-25% no miega perioda. Bet REM fāze mēdz samazināties, ja rodas miega traucējumi [8].

N1

N1 fāze ir apmēram 5-10% no miega perioda. N1 fāzes laikā cilvēkam var būt iespēja dzirdēt, atpazīt un reaģēt uz ārējiem stimuliem. EEG paterni var būt diezgan mainīgi, un tie var strauji novirzīties, tāpēc dažreiz ir grūti tos interpretēt. EEG laikā, kad pacients atrodas N1 fāzē, var novērot salīdzinoši zemu sprieguma modeli ar jauktām frekvencēm un ātru EEG aktivitāti. 1.1. [8]. N1 arī sauc par pārejas posmu no nomoda stāvokļa. Bieži vien cilvēku var pamodināt no šī stāvokļa un viņš var pat neapzināties, ka bija aizmidzis.

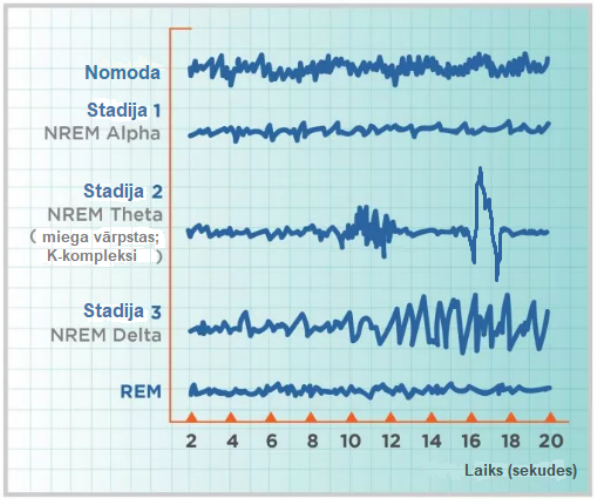
N2

N2 fāzi raksturo nedaudz dziļāka miega pakāpe, kurā iespējams saskarties ar miega vārpstām [10] un K kompleksiem. Miega vārpstas ir unikālas EEG viļņu formas, kas norāda uz ārējo

stimulu, piemēram, skaņu bloķēšanu 1.1.. K kompleksi ir asi, viengabalaini vai polifāziski lēni viļņi ar asu negatīva augšupvērstu novirzi, kam seko lēnākā pozitīva lejupvērsta novirze. K kompleksi, pat bez miega vārpstām, ir pietiekams radītājs, lai atpazītu N2 pakāpi. N2 fāzē daudz mazāk apzinās, kas notiek apkārt, nekā tad, kad cilvēks atrodas N1 fāzē. N2 fāze ir apmēram 40-50% no kopējā miega perioda.

N3

N3 fāze sevī ietver 3. un 4. fāzi. EEG, kad iestājas N3 fāze ir novērojamas mēreni zemas frekvences un liela amplitūdas aktivitāte 1.1.. Fāze N3, kas formāli saucas par deltas miegu, tiek uzskatīts par dziļāko miega posmu, jo šajā posmā ir grūti izsaukt jebkādu reakciju [10]. N3 parasti ir posms, no kura visgrūtāk ir pamodināt cilvēku. Cilvēka pamodināšana no N3 posma bieži izraisa apjukumu un dezorientāciju. N3 fāze ir, apmēram, 20-25% no miega perioda un dominē nakts pirmajā trešdaļā. N3 fāzei ir tendence ievērojami samazināties ar vecumu.



Att. 1.1. Viļņu paterni miega laikā
[9]

1.2. Hipnogramma

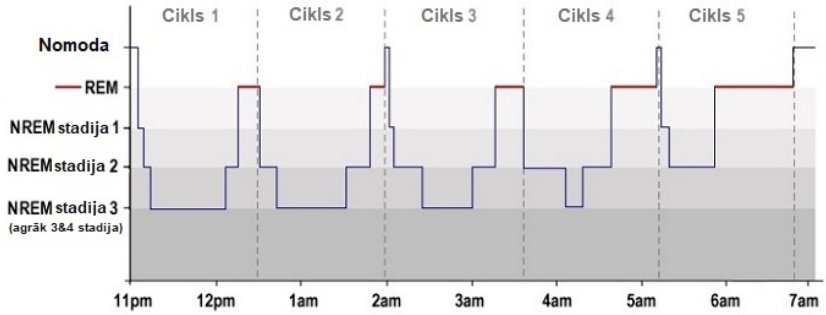
Hipnogramma ir polisomnogrāfijas forma jeb grafiks [11], kas atspoguļo miega posmus kā laika funkciju. Hipnogrammas tika izveidotas, lai vieglāk būtu apskatīt kā elektroencefalogrammas laikā mainās smadzeņu viļņu darbība, kad cilvēks ir aizmidzis.

Hipnogrammas parasti iegūst, vizuāli novērtējot ierakstus no elektroencefalogrammas, elektrookulogrāfijas un elektromiogrāfijas [12]. Šo trīs avotu izejas dati tiek vienlaikus ierakstīti un pēc tam attēloti grafikā kā hipnogramma.

Hipnogramma vienā lappusē ļauj attēlot vairākus mainīgos lielumus, ieskaitot miega posmus, elpošanas procesus, pozitīvos elpošanas ceļu spiedienus, mehāniskas kustības, oksimetriju, sirds ritma izmaiņu mērījumus, elektroencefalogrāfijas jaudas spektru un ķermeņa stāvoklis [13].

Citiem vārdiem var teikt, ka hipnogramma ir visa miega pētījuma saspiests grafiskais kopsavilkums. Hipnogrammas struktūru var apskatīt attēlā 1.2., kur uz Y ass ir attēlotas miega fāzes un uz X ass laiks, kad sākas un beidzas attiecīgā miega fāze.

Šī darba ietvaros hipnogrammas dati tiks izmantoti, lai veiktu ierakstīto EEG signālu sadalīšanu pa apakš signāliem, kur katrs apakš signāls atbilst konkrētai miega fāzei, tādējādi būs iespējams noteikt kādā fāzē, konkrētā laika brīdī, atrodas cilvēks. Iegūtā datu kopa tiks izmantota neironu tīklu apmācībai un testēšanai.



Att. 1.2. Hipnogramma ar miega ciklu sadalījumu [14]

2. Dziļā apmācība

Mašīnmācīšanās (no angļ. Machine Learning) pēta algoritmus un statistikas modeļus, kurus datorsistēmas izmanto, lai efektīvi veiktu konkrētus uzdevumus. Mašīnmācīšanās pamatnoteikums ir veidot algoritmus, kas var saņemt ievades datus, izmantot statistisko analīzi un prognozēt izvades datus bez specifiskas modeļa programmēšanas. Mašīnmācīšanās metodes tiecas iemācīties datos esošo struktūru, tādējādi pielāgojot modeli datu modelim. Ja modeļiem tiek padoti jauni dati, tie spēj patstāvīgi iemācīties jaunajos datos esošo struktūru.

Dziļā apmācība ir specifiska mašīnmācīšanās joma, kurā tiek izstrādātas metodes un algoritmi, kas ļauj no datiem izgūt īpašības un tās tiecas iemācīties datos esošu struktūru (patterns) [15]. Dziļā apmācība atklāj sarežģītas struktūras lielās datu kopās, izmantojot atpakaļ izplatīšanās algoritmu, lai noteiktu kā nepieciešams mainīt tīkla parametrus, lai iegūtu īpašības, kas reprezentē ievades datus.

Pēc Dziļie konvolūcijas tīkli tiek veiksmīgi izmantoti attēlu, video, runas un audio apstrādē, savukārt rekurentie tīkli tiek izmantoti secīgu datu klasifikācijā, piemēram, teksta, runas, video un laika rindu klasifikācijā, notikumu noteikšanā, u.tml.

2.1. Konvolūciju neironu tīkli

Pēdējo desmit gadu laikā Konvolūciju neironu tīklu pielietojums kļuvis populārs vairākās jomās, kas saistītas ar paraugu atpazīšanu. Tas ietver attēlu apstrādi un balss atpazīšanu. Šis sasniegums ir mudinājis gan pētniekus, gan izstrādātājus pievērsties lielākiem modeļiem, lai atrisinātu sarežģītus uzdevumus, kas nebija iespējami ar klasiskajiem neironu tīkliem [16].

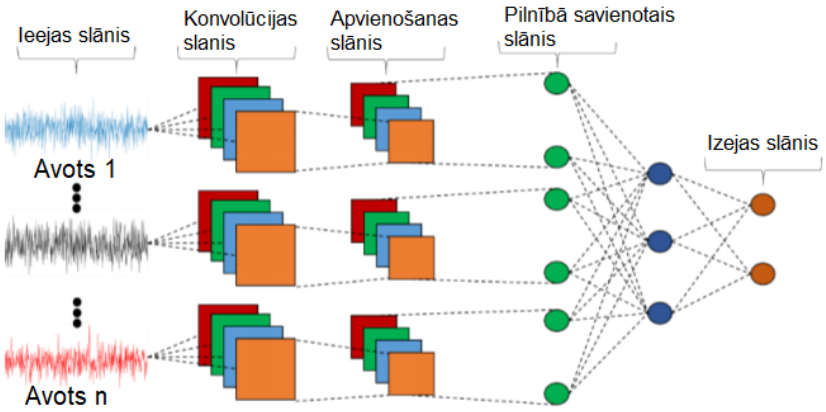
Konvolūcijas neironu tīkls (no angļ. Convolutional Neural Network) ir dziļas apmācības algoritms, kas var kā ieejas datus izmantot, piemēram, attēlus un piešķir nozīmi (apmācāmus svarus un nobīdes) atsevišķiem attēla aspektiem vai objektiem un tādā veidā algoritms spēj atšķirt vienu no otra.

CNN prasa daudz mazāku datu priekšapstrādi, salīdzinot ar citiem klasifikācijas algoritmiem. Izmantojot primitīvas metodes filtri tiek manuāli sagatavoti, bet CNN gadījumā pats algoritms spēj apgūt šos filtrus jeb īpašības. Kā arī, izmantojot atbilstošus filtrus, CNN spēj veiksmīgi uztvert datu telpiskās un laika atkarības.

2.1.1. Arhitektūra

Konvolūciju tīkls sastāv no trīs veida slāņiem. Konvolūcijas slāņa (no angļ. convolution layer), apvienošanas slāņa (no angļ. pooling layer) un pilnībā savienotā slāņa (no angļ. fully connected layer). Var apskatīt attēlu 2.1., kur tiek parādīta CNN arhitektūra ar daudz avotu sensoru signāliem, kā ievad datiem. Konvolūcijas tiek veiktas sensoru signāliem gar laika asi. Tā kā sensora signāli ir viendimensijas, tiek veikta 1D konvolūcija nevis 2D. Atšķirību var apskatīt

attēlā 2.2.. Pēc tam katra signāla avota apgūtās īpašības tiek apvienotas un izmantotas kā ieejas dati pilnībā savienotajam slānim. Tālāk tiks dziļāk apskatīts katrs no slāņiem.

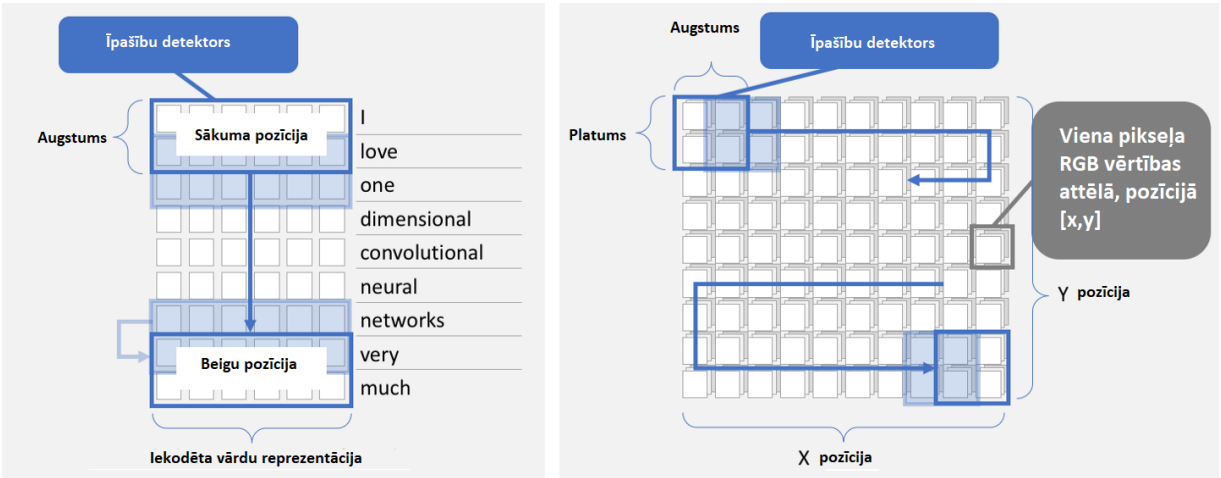


Att. 2.1. Konvolūciju neironu tīkls ar daudz avotu senosoru ieejas datiem, kas sastāv no konvolūcijas, apvienošanas un pilnībā savienotajiem slāņiem.

[17]

Konvolūcijas slānis

Kad dati nokļūst konvolūcijas slānī, slānis bīda katru kodolu (filtru) pāri ieejas datu telpiskajai dimensijai, lai izveidotu aktivācijas karti. Šajā gadījumā tiek pielietota nevis 2D konvolūcija, kā attēlu gadījumā, bet gan 1D konvolūcija. Galvenā atšķirība ir ievades datu dimensija un tas, kā filtrs bīdās pāri datiem un to var redzēt attēlā 2.2..



Att. 2.2. Pa labi 2D konvolūcijas piemērs ,kur 2x2 filtrs tiek bīdīts pa divdimensionāliem datiem un pa kreisi 1D konvolūcijas piemērs, kur filtrs iet cauri vārdu vektoriem ar soli 2

[18]

Īpašību kartes(no angļ. Feature maps) jeb aktivācijas karte (no angļ. Activation maps) ir izejas dati, kas tiek iegūti, kad viens filtrs tiek pielietots iepriekšējam slānim. Dotais filtrs tiek

bīdīts pa visu iepriekšējo slāni, vienlaicīgi pārvietojoties pa vienam pikselim. Katra pozīcija izraisa neirona aktivāciju un izejas dati tiek apkopoti aktivizācijas kartē. Kā piemēru var apskatīt 2.3., kur filtru ar izmēru 3 tiek piemērots 1x6 ievades matricai, lai iegūtu izejas datus ar izmēru 1x4.

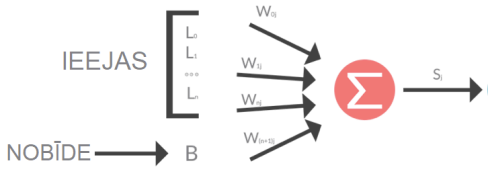
Att. 2.3. 1D konvolūcija.
[19]

Konkrētais filtrs ar lielumu 3 tiek izmantots, lai aprēķinātu vienu izejas vērtību. Šajā gadījumā, tiek aprēķināts ievades datu un filtra skalārais reizinājums, kas ir tikai attiecīgo elementu reizinājums, kam seko kopējās summas aprēķināšana, lai iegūtu vienu vērtību. Matemātiski tas izskatās šādi 2.1..

$$(1 * 2) + (3 * 0) + (3 * 1) = 5 \tag{2.1.}$$

Lai iegūtu pilnīgu izejas datu masīvu $(5, 6, 7, 2)$, izmanto to pašu operāciju, kur izvēlēto kodolu bīda pāri ievades datiem ar soli 1. Atšķirībā no 2D konvolūcijas, kur filtru virza divos virzienos, 1D konvolūcijas gadījumā filtrs tiek bīdīts vienā virzienā - pa kreisi vai pa labi.

Kad filtru bīda pāri ievades datiem, tiek izveidots skalārs produkts, kas tiek aprēķināts katrai filtra vērtībai. Un beigās rezultātam tiek pievienota nobīde. Nobīdes vienība ir papildus neirons, kas pievienots katram pirms izejas slānim un nobīde ir konstanta vērtība 1. Nobīdes vienības nav savienotas ar iepriekšējo slāni un tādējādi neatspoguļo patiesu tīkla aktivitāti. Šīs struktūras mērķis ir noteikt, ka, ja visas neirona ieejas ir 0, tad pēc aktivizācijas funkcijas pielietošanas neironam joprojām būs kaut kādas vērtības.



Att. 2.4. Neirons ar ievade datiem un saviem svariem, kā arī īpašs ieejas elements - nobīde, kuras vērtība vienmēr ir 1

[20]

Nobīdes vienība tiek pievienota ievades un katra slēptā slāņa sākumā vai beigās, un tas neietekmē iepriekšējā slāņa vērtības. Citiem vārdiem sakot, šiem neironiem nav nekādu ienākošo savienojumu. Nobīdes svaru pievienošanu apraksta ar vienādojumu 2.2., kur izvades dati Y, ievades dati X, svari - W, W_b nobīde vienības svari..

$$Y = W * X + W_b \tag{2.2.}$$

No šīs informācijas tīkls iemācīsies, kuri filtri tiek aktivizēti, kad filtram atrodoties noteiktā pozīcijā ievades datos tiek atrastas attiecīgās īpašības. Šo procesu sauc par aktivizāciju. Katram filtram ir sava atbilstošā aktivizēšanas karte, kura tiek sakārtota atkarībā no dziļuma dimensijas. Katra īpašību karte meklē kaut ko citu. Viena īpašību karte var meklēt taisnas līnijas, otra - līknes. Tādā veidā tiek izveidota pilnīga izvades datu informācija no konvolūcijas slāņa.

Aktivizācijas funkcijas

Neironu tīkls bez aktivācijas funkcijas būtībā ir tikai lineārs regresijas modelis. Aktivizācijas funkcija veic ievades datu nelineāru transformāciju, padarot modeli spējīgu mācīties un veikt sarežģītākus uzdevumus. Aktivizācijas funkcija ņem vērā mijiedarbību starp dažādiem parametriem un veic datu transformāciju, lai pēc tam izlemtu kuras neironu vērtības pāriet uz nākamo slāni. Eksistē vairākas aktivizācijas funkcijas:

- **ReLU** - Pielāgotā lineārā vienība ir visizplatītākā aktivizēšanas funkcija dziļās mācīšanās modeļos. Funkcija atgriež 0, ja tā saņem negatīvus datus, bet ja ir pozitīvas vērtības, tad atgriež šo vērtību x atpakaļ. To var rakstīt ar vienādojumu 2.3..

$$f(x) = max(0, x) \tag{2.3.}$$

  Dēļ negatīvām x vērtībām, gradients var tiekties uz 0. Gradients būs 0 un dēļ tā svari netiks atjaunoti. Tas nozīmē, ka tie neironi, kas nonāk šajā stāvoklī, pārtrauks reaģēt uz kļūdas vai ievades datu izmaiņām. Šo problēmu sauc par mirstošo ReLu.

- **Softmax** - Softmax ir ļoti interesanta aktivizēšanas funkcija, jo tā ne tikai ierobežo izejas datu vērtības no datu kopas $X = x_1, ...x_n$ diapazonā [0,1], bet arī pārveido izvades datus tā, lai kopējā summa būtu 1. Softmax funkcijas 2.4. izejas dati atbilst kategoriskajam varbūtību sadalījumam, kas norāda to kāda varbūtība ir katrai klasei.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{Z_k}} \tag{2.4.}$$

  z ir izejas slāņa ievades vektors, j indeksē izejas vienības, tāpēc j = 1, 2, ..., K.

- **Sigmoid** - Šīs aktivizācijas funkcijas 2.5. priekšrocība, atšķirībā no lineārās funkcijas, ir tas, ka aktivizēšanas funkcijas izejas dati no padotās datu kopas $X = x_1, ...x_n$ būs diapazonā (0,1). Tāpēc tas ir īpaši piemērots modeļiem, kuros ir jāparedz, ka izejas dati būs varbūtības.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.5.}$$

  Virzoties uz abiem sigmoid funkcijas galiem, Y vērtības mēdz reaģēt ļoti minimāli uz X izmaiņām. Un tas var izraisīt to, ka neironu tīkls iestrēgst apmācības laikā. Šī reģiona gradients būs neliels un tas rada izzūdošā gradienta problēmas.

- **than** - tanh ir līdzīgs loģistiskajai sigmoidai. Tanh funkcijas 2.6. gadījumā dati no datu kopas $X = x_1, ...x_n$ tiek ierobežoti diapazonā no -1 līdz 1. Priekšrocība ir tā, ka negatīvās ieejas vērtības tiks piesaistītas stipri negatīvām vērtībām un nulles ieejas vērtības būs tuvu nullei. Bet, tāpat kā sigmoid funkcijai, tanh ir izzūdošā gradienta problēmas.

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.6.}$$

Hiperparamteri

Hipersparametru pielāgošana dziļajam neironu tīklam ir sarežģīts process. Hiperparamteri var būt šādi:

- **Mācīšanās ātrums** - Mācīšanās ātrums nosaka, cik daudz optimizācijas algoritmā nepieciešams atjaunot svarus. Var izmantot fiksētu mācību ātrumu, pakāpeniski samazināt mācīšanās ātrumu, paātrinātās metodes vai adaptīvu mācīšanās ātrumu.

- **Epohu skaits** - Epohu skaits nosaka to, cik reižu visa treniņu datu kopa iziet cauri neironu tīklam.

- **Partijas lielums** - Partijas lielums nosaka paraugu skaitu, kas tiks izplatīti caur tīklu. Minimālais partijas skaits parasti ir diapazonā no 16 līdz 128.

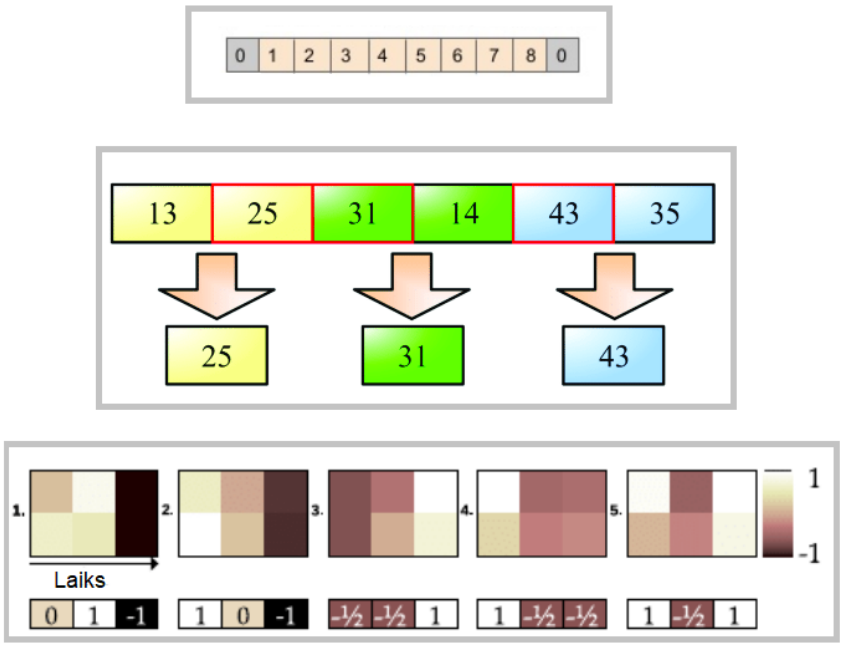
- **Svara inicializācija** - Lai novērstu to, ka neironi mirst, vajadzētu inicializēt svarus ar nelieliem nejaušiem skaitļiem, bet ne pārāk maziem, lai izvairītos no nulles gradienta. Vienmērīgs sadalījums parasti darbojas labi.

- **Izslēgšanas metode priekš tīkla regulēšanas** - Kā labu regulēšanas metodi izmanto izslēgšanu, lai izvairītos no tā, ka dziļie neironu tīkli pārmācās. Metode izslēdz no neironu tīkla gadījuma vienības atbilstoši izvēlētajai varbūtībai. Kā sākuma vērtība var izvēlēties 0,5.

- **Kodola / filtra lielums** - Filtrs ir svaru matrica, kura tiek bīdīta parī ievades datiem. Konvolūcijas filtrs ir veids, kā atspoguļot īpašības, kuras satur konkrētie ievades dati. Filtru piemērus iespējams apskatīt attēlā 2.5.. Filtra matricas svari tiek iegūti, apmācot datus. Mazāki filtri iemācās īpašības, kas ir pēc iespējas lokālākas un lielāki filtri ir vairāk globālām, augsta līmeņa un reprezentatīvai īpašībām. Filtra izmērs parasti ir nepāra.

- **Papildināšana** - Papildināšanai (no angļ. padding) parasti izmanto, lai pievienotu papildus kolonnu un rindu ar nulles vērtībām, lai pēc konvolūcijas saglabātu nemainīgus telpiskos izmērus, tādējādi uzlabojot veiktspēju, jo tādā veidā tiek saglabāta informāciju pie robežām. 1D gadījumā nulles var pievienot vektora galos, kā attēlots 2.5.. Izejas datu izmērs ir vienāds ar ieejas datu lielumu, to panāk pievienojot papildus datus vienmērīgi pa kreisi un pa labi, bet, ja pievienojamo kolonnu skaits ir nepāra, tas jāpievieno papildus kolonnu pa labi.

- **Soļi** - Soļi (no angļ. Stride) parasti ir to pikseļu skaits, kurus vēlaties izlaist, notiekot konvolūcijai, kur ievades datu svaru elementus reizina ar filtra elementiem jeb kamēr filtru bīda pāri ievades datiem horizontāli un vertikāli. To lieto, lai ievērojami samazinātu ieejas datu izmēru. 1D gadījumā filtrs iet cauri vektoram 2.5..



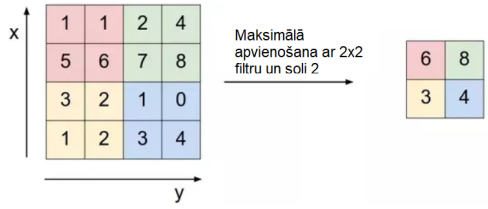
Att. 2.5. Pa kreisi augšējā bildē parādīts, kā izskatās nulles vērtības iestatīšana, pa vidu parādīta 1D maksimālā apvienošana, izmantojot soli 2 un apakšējā bildē ir attēloti daži filtri

[21] [22] [23]

Apvienošanas slānis

Apvienošanas slāņi tika izstrādāti, lai samazinātu parametru skaitu, kas nepieciešami, lai aprakstītu slāņus, kas ir dziļāk tīklā. Apvienošana arī samazina aprēķinu skaitu, kas vajadzīgs, lai apmācītu tīklu, vai arī, lai vienkārši izpildītu klasifikācijas uzdevuma ne pārāk ilgā laikā.

Apvienošanas slāņi ir vērsti uz to, lai pakāpeniski samazinātu dimensiju skaitu, tādējādi vēl vairāk samazinot parametru skaitu un modeļa skaitļošanas sarežģītību. Apvienošanas slānis darbojas ar katru ievades datu aktivizēšanas karti un veic dimensiju mērogošanu, izmantojot, piemēram, "Max" funkciju. Varam apskatīt attēlā 2.6. aktivizēšanas karti, kurai pielieto maksimālo apvienošanu izmantojot filtru ar dimensionalitāti 2x2.



Att. 2.6. Maksimālās apvienošanas operācija
[24]

Šo slāni sauc par maksimālais apvienošanas slāni un šajā gadījumā tiek izvēlēta maksimālā vērtība no katras iepriekšējā slāņa neironu kopas. Lielākajai daļai CNN tiek izmantoti maksimālās apvienošanas slāņi ar kodoliem, kuru dimensionalitāte ir 2x2, kam tiek pielietoti 2 soļi gar ieejas datu telpisko dimensiju. Kā arī notiek aktivizēšanas kartes mērogošana līdz 25% no sākotnējā datu lieluma. Apvienošanas slāņa destruktīvā rakstura dēļ ir tikai divas maksimālās apvienošanas metodes, kuras izmanto.

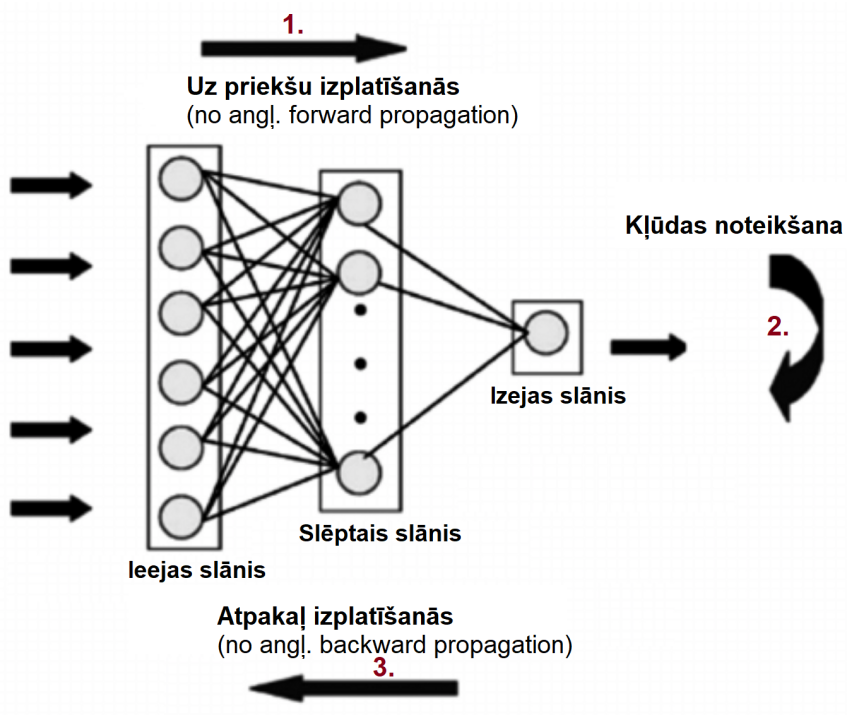
Parasti soļi un kodoli tiek iestatīti kā 2×2, kas ļaus slānim izstiepties caur ieejas datu telpisko dimensiju. Turklāt var tikt izmantota dublēšanās, kur solis ir iestatīts uz 2, un kodola izmērs ir 3. Destruktīvā rakstura dēļ, ja kodola lielums pārsniedz 3, tas parasti ievērojami samazinās modeļa veiktspēja [25].

Pilnībā savienotais slānis

Tīkla galā tiek pievienots pilnībā savienotais slānis. Pilnībā savienotais slānis satur visus savienojumus ar visām iepriekšējā slāņa aktivācijām, kā redzams regulārajos neironu tīklos. Veids, kādā pilnībā savienotais slānis darbojas, ir tas, ka tas skatās uz iepriekšējā slāņa izvadi un nosaka, kuras īpašības visvairāk atbilst konkrētai klasei un pēc tam tiek izvadītas klašu varbūtības. Pēc tam tiek pielietots atpakaļ izplatīšanās algoritms, lai atjaunotu svarus līdz brīdim, kad tīkls sasniedz kādu noteiktu mazu kļūdas slieksni vai izdara visas iterācijas un apstājas.

2.1.2. Apmācība

Galvenais apmācību mērķis ir samazināt kļūdu starp prognozētajiem un faktiskajiem datiem. Lai samazinātu kļūdu starp faktiskajiem un prognozētajiem datiem, kļūda tiek atpakaļ izplatīta (no angļ. backpropagate) visos slāņos jeb tiek atjaunināti svari un kad tas ir izdarīts viss process atkārtojas. Shematiski to var redzēt attēlā 2.7..



Att. 2.7. Tīkla apmācības shēma
[26]

Atpakaļ izplatīšanās ir paņēmiens, ko izmanto, lai atjauninātu svarus, izmantojot krītošo gradientu (no angļ. gradient descent). Krītošā gradienta metode ir iteratīvs optimizācijas algoritms, lai atrastu minimālo funkciju. Šajā gadījumā ir jāsamazina kļūdas funkcija (no angļ. loss function), lai atrastu funkcijas lokālo minimumu. Tiek aprēķināts kļūdas funkcijas gradients, kas atkarīgs no neironu tīkla svariem un šis aprēķins turpinās atpakaļ pa visu tīklu. Svari tiek atjaunināti, izmantojot nelielu soli alfa. Vairāk par šo procesu var uzzināt [27].

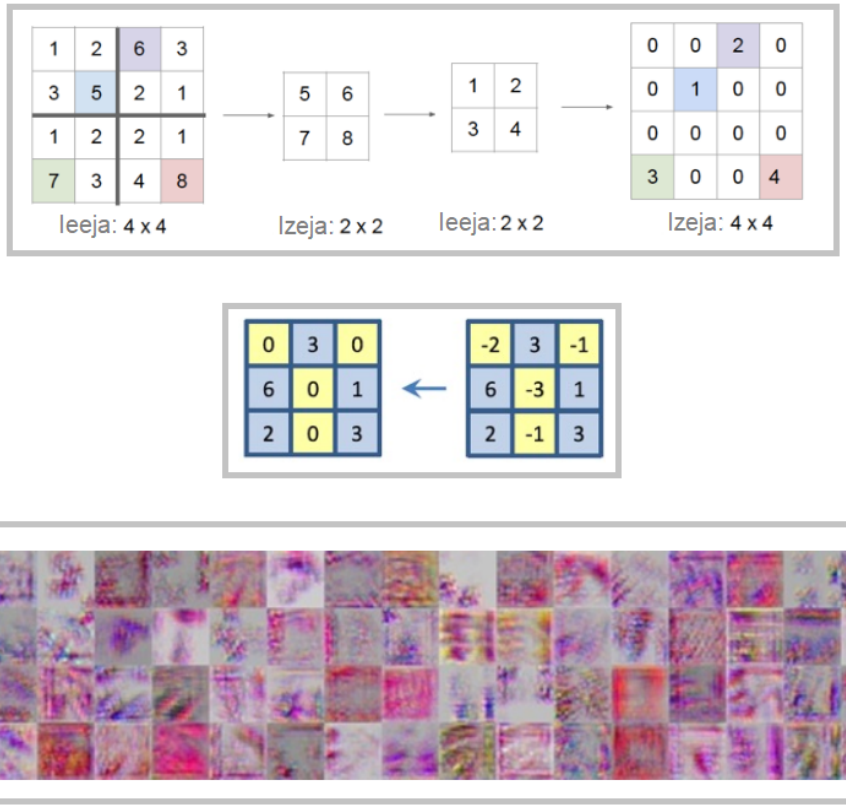
CNN gadījumā, kad tiek veikta atpakaļ izplatīšanās veidojas papildus grūtības, kas rodas, piemēram, kad nepieciešams veikt atpakaļ izplatīšanos caur apvienošanas slāni.

Veicot uz priekš izplatīšanos (no angļ. forward propogate) ir nepieciešams saglabāt informāciju par pozīcijām, kurās maksimālajā apvienošanas slānī aktivizācijas kartēm tika noteiktas maksimālā vertības 2.8.. Atpakaļ izplatīšanās gadījumā visas vietas, kurās neatrodas maksimālā vērtība, tiek iestatītas kā 0, bet maksimālās vērtības vietās tiek iestatīta vērtība 1. Tiek iegūta matrica ar 0 un 1. [26]

Aktivizācijas funkcijas gadījumā arī ir nepieciešams pielietot atpakaļ izplatīšanos. Aktivizācijas karte tiek padota aktivizācijas funkcijai, kas ir, piemēram, ReLU. Tiek iegūta aktivizācijas karte, kur visas negatīvās vērtības tiek aizstātas ar nullēm un to var apskatīt attēlā 2.8.. Izmantojot ReLU nelinearitātes (no angļ. non-linearities), kas veic aktivizācijas kartes pārveidošanu, tiek nodrošināts tas, ka aktivizācijas kartes vienmēr ir pozitīvas.

Tālāk pēc atpakaļ izplatīšanās caur maksimālo apvienošanas slāni un aktivizācijas funkcijas, abu iegūto matricu attiecīgie elementi tiek sareizināti savā starpa (no angļ. element wise multiplication) iegūstot pārveidotu aktivizācijas karti.

Konvolūcijas slānī, veicot atpakaļ izplatīšanos, CNN izmanto filtrus, kurus tas iemācījās, lai konvolvētu (no angļ. convolve) aktivizācijas kartes no iepriekšējā slāņa. Lai veiktu iznvertāciju, tiek izmantotas to pašu filtru transponētās versijas, bet šie filtri tiek pielietoti iegūtajām pārveidotajām aktivizācijas kartēm nevis zemāk esošā slāņa izeja datiem. Iegūtie rekonstruētie dati parāda visnozīmīgāko informāciju, kas tiek izmantota, lai aktivizētu neironu, kā piemēram, parādīts attēlā 2.8.. [28]



Att. 2.8. Augšējā bilde attēlo atpakaļ izplatīšanos, izmantojot apvienošanas slāņus, vidējā bilde attēlo atpakaļ izplatīšanos, izmantojot ReLU nelinearitātes, apakšējā bilde attēlo iegūtās aktivizācijas kartes pēc atpakaļ izplatīšanās, izmantojot konvolūciju

[29] [30]

Kļūdas funkcijas

Kļūdas funkcija ir svarīga mākslīgo neironu tīklu daļa, ko izmanto, lai novērtētu neatbilstību starp prognozētajiem izejas datiem un faktiskajām klašu etiķetēm. Kļūdas funkcija ir ne negatīva vērtība, kur modeļa veiktspēja palielinās līdz ar kļūdas funkcijas vērtības samazināšanos. Dažas no šīm funkcijām ir:

- **logcosh**. Šī funkcija ir prognozētās kļūdas logaritms no hiperboliskā kosinusa [31]. Un šo funkciju apraksta ar vienādojumu 2.7., kur Y apzīmē vektoru ar patiesajām n vērtībām un $\hat{Y}$ apzīmē vektoru ar prognozētajām n vērtībām. $logcosh$ darbojas galvenokārt kā vidējā kvadrāta kļūda, bet to tik stipri neietekmē gadījumi kad ļoti bieži tiek veikta nepareiza izejas datu prognozēšana.

$$L(Y, \hat{Y}) = \sum_{i=1}^{n} log(cosh(\hat{Y}_i - Y_i)) \tag{2.7.}$$

- **Kategoriskā starpentropijas kļūda (angļ. val. categorical crossentropy)** To sauc arī par Softmax kļūdu. Šajā gadījumā lieto $Softmax$ aktivizācijas funkciju, kas apvienota ar starpentropijas kļūdu. Šo funkciju izmanto, kad nepieciešams klasificēt vairākas klases. Galvenais atcerēties, ka patiesajiem datiem jābūt kategoriskā formātā.

- **Vidējā kvadrāta kļūda** Vidējā kvadrāta kļūda ir visbiežāk izmantotā regresijas kļūdas funkcija. MSE ir kvadrātiskā attālumu summa starp mūsu patiesajiem datiem un prognozētajiem datiem. Un MSE aprēķina pēc formulas 2.8., kur Y_i ir vektors, kas apzīmē n patieso vērtību skaitu un $\hat{Y}_i$ ir vektors, kas apzīmē n prognožu skaitu.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 \tag{2.8.}$$

2.2. Rekurentie Neironu tīkli

Rekurentie neironu tīkli (angļ. Recurrent Neural Networks) ir mākslīgie neironu tīkli, ko parasti izmanto runas atpazīšanā un dabiskās valodas apstrādē (angļ. Natural Language Processing).

RNN ir uzraudzīto mašīnmācīšanās modeļu klase, kas izgatavota no mākslīgiem neironiem ar vienu vai vairākām atgriezeniskās saitēm. Šos tīklus sauc par rekurentajiem, jo tie veic vienu un to pašu uzdevumu katram elementam, un izejas dati ir atkarīga no iepriekšējiem aprēķiniem.

RNN ideja ir izmantot informāciju, kas ir secīga, lai šie tīkli varētu atpazīt datu secīgās īpašības un izmantot šīs īpašības, lai prognozētu nākamo iespējamo scenāriju [32].

RNN tiek izmantoti dziļā apmācībā un modeļu izstrādē, kas simulē neironu darbību cilvēka smadzenēs. Tie ir īpaši spēcīgi gadījumos, kad kontekstam ir izšķiroša nozīme, lai prognozētu rezultātus. Tie atšķiras no citiem mākslīgo neironu tīkliem ar to, ka tie izmanto atgriezeniskās saites cilpas, lai secīgi apstrādātu datus.

Šīs atgriezeniskās saites cilpas ļauj saglabāt informāciju un šādu efektu bieži raksturo arī kā atmiņu [33]. Tāpēc var domāt, ka šiem tīkliem ir sava $atmia$, kurā ir ietverta informācija par līdz šim aprēķināto, tāpēc teorētiski tos var izmantot, lai saglabātu informāciju par patvaļīgi garu virkni. Taču praksē šie tīkli spēj apskatīt tikai informāciju, kas ir bijusi dažus soļus atpakaļ.

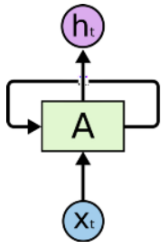
Piemēram, ja ir jāprecizē nākamais vārds noteiktā teikumā, visu iepriekšējo vārdu savstarpējās attiecības palīdz prognozēt labāku gala rezultātu. RNN atceras visas šīs attiecības apmācības laikā.

2.2.1. Arhitektūra

Vienkāršam RNN ir trīs slāņi - ievades, rekurentais slēptais un izejas slānis. Diagrammā 2.9. vienkāršā veidā ir attēlots, kā rekurentais neironu tīkls A apskata ievades datus x_t un atgriež izejas datus h_t.

Tradicionālie neironu tīkli pieņem, ka visi ieejas un izejas dati ir savstarpēji neatkarīgi [32]. Bet, ja nepieciešams prognozēt nākamo vārdu teikumā, ir nepieciešams zināt kādi vārdi bija izmantoti pirms tam. Pieņemot lēmumu tiek ņemti vērā pašreizējie ievades dati, kā arī tā informācija, kuru tīkls ir iemācījies no iepriekš saņemtajiem ievades datiem.

RNN gadījumā cilpa ļauj informāciju pārsūtīt no viena tīkla posma uz otru. Šīs cilpas ir tas, kas padara rekurentos neironu tīklus īpašus. Bet kopumā tie īpaši neatšķiras no parastajiem neironu tīkliem. Rekurento neironu tīklu var uzskatīt par viena un tā paša tīkla vairākām kopijām, kur katra no šīm kopijām nosūta ziņojumu nākamajam pēctecim.



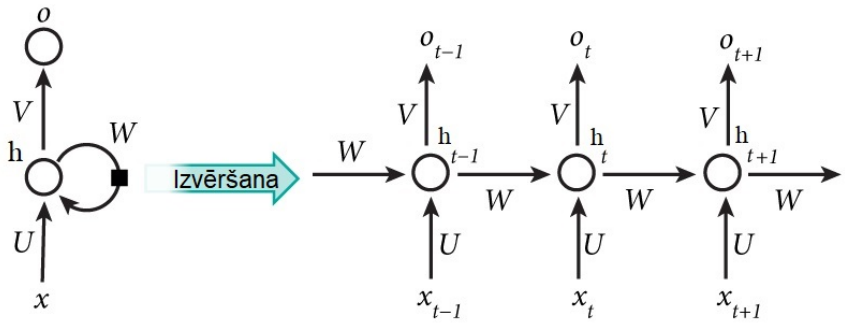
Att. 2.9. Rekurento neironu tīklu vienkārša reprezentācija
[34]

RNN kļūdas var izplatīt vairāk nekā divos slāņos, lai iegūtu informāciju, kas ir bijusi daudz senāk. Šo procesu parasti sauc par izvēršanu. Diagrammā 2.10. varam apskatīt rekurentā neironu tīklu A izvēršanu. Kā piemēru varam apskatīt situāciju, ja mūsu virkne ir 5 vārdu teikums. Šajā

gadījumā tiks izveidots neironu tīkls ar 5 slāņiem, kur viens slāni saturēs informāciju par vienu vārdu.

Diagrammā 2.10. h_t ir slēpta stāvoklis laika solī t. Slēptajam slānim h ir M slēptās vienības $h_t = (h_1, h_2, ..., h_M)$, kas ir savstarpēji saistītas laikā ar rekurentiem savienojumiem.

Slēpto vienību inicializēšanai izmanto mazus nulles elementus, kas var uzlabot vispārējo veiktspēju un tīkla stabilitāti. Varam iedomāties slēpto stāvokli h_t kā tīkla atmiņu. h_t ir tas elements, kas saglabā informāciju par to, kas noticis visos iepriekšējos posmos, tāpēc to arī dēvē par tīkla $atmiu$.



Att. 2.10. Rekurento neironu tīklu trīs laika posmu $izvrana$
[32]

1. Tīklam tiek padoti viena laika soļa ieejas dati jeb tīklam tiek padots ieejas datu mainīgais x_t laika solī t. Piemēram, x_1 varētu būt vektors, kas atbilst teikuma otrajam vārdam.

2. Tālāk aprēķina pašreizējo slēpto stāvokli, izmantojot pašreizējos ieejas datus un iepriekšējo stāvokli. Citiem vārdiem sakot, notiek h_t aprēķins pēc formulas 2.9..

$$h_t = f(U_{x_t} + W_{h_{t-1}}) \tag{2.9.}$$

   Funkcija f parasti ir nelineāra, piemēram, tanh vai ReLU. h_{t-1} kas ir nepieciešams, lai aprēķinātu pirmo slēpto stāvokli, parasti tiek inicializēts tā, ka visas sākuma vērtības ir nulles.

   U, W, V ir parametru matricas. Atšķirībā no tradicionālajiem dziļajiem neironu tīkliem, kur katrā slānī tiek izmantoti dažādi parametri, RNN visos posmos izmanto vienādus parametrus (U, W, V). Tas atspoguļo faktu par to, ka veic vienu un to pašu uzdevumu katrā posmā ar dažādiem ievades datiem. Tas ievērojami samazina kopējo parametru skaitu, ko nepieciešam iemācīties.

3. Pašreizējais h_t kļūst par h_{t-1} nākamajā apstrādes posmā. Varam veikt nepieciešamos aprēķinus tik daudz laika posmiem, cik pieprasa konkrētā problēma. Pēc tam varam apvienot informāciju no visiem iepriekšējiem stāvokļiem.

4. Kad visu laika posmu aprēķini ir pabeigti, pēdējo pašreizējo iegūto stāvokli izmanto, lai aprēķinātu izvades datus o_t. o_t ir izejas dati laika solī t un tie tiek aprēķināti, pamatojoties tikai uz atmiņu laikā t. Praksē tas ir nedaudz sarežģītāk, jo h_t parasti nevar iegūt informāciju no soļiem, kas tikuši izpildīti pārāk sen atpakaļ.

   Ja apskata vienkāršu piemēru tad, ja grib paredzēt nākamo vārdu teikumā, tad izejas dati būtu visas dotās vārdnīcas varbūtību vektors 2.10..

$$o_t = softmax(V_{h_t}) \tag{2.10.}$$

   Matemātiski $softmax$ funkcija paņem nenormalizētu vektoru un normalizē to, lai iegūtu varbūtības sadalījumu.

   Diagrammā 2.10. ir parādīti izvades dati katrā laika posmā, bet atkarībā no uzdevuma, tas var arī būt nevajadzīgs. Piemēram, prognozējot teikumu mūs interesē tikai gala rezultāts, nevis katra individuāla vārda nozīme. Tāpat arī nav nepieciešamība pēc ievades datiem katrā laika posmā. RNN galvenā iezīme ir tieši slēptais stāvoklis, kurā tiek uztverta daļa informācijas no virknes [32].

5. Pēc tam izvades datus salīdzina ar patieso rezultātu, un pēc tam tiek aprēķināta kļūda izmantojot kļūdas funkciju.

6. Tad tiek pielietota atpakaļ izplatīšanās algoritms, lai atjaunotu svarus līdz brīdim, kad tīkls sasniedz kādu noteiktu mazu kļūdas slieksni vai izdara visas iterācijas un apstājas.

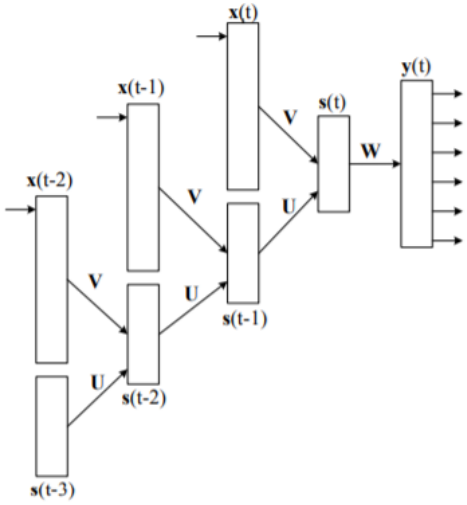
2.2.2. Atpakaļ izplatīšanās laikā

Uz priekšu izplatīšanās gadījumā ievaddati tiek padoti un pārvietojas uz priekšu katrā laika posmā. Bet gadījumā ar atpakaļ izplatīšanos figurāli iet atpakaļ laikā, lai mainītu svarus. Šo procesu sauc par atpakaļ izplatīšanos laikā.

Atpakaļ izplatīšanās algoritms rekurentajiem neironu tīkliem patiesībā ir līdzīgs parastajam neironu tīklam, tikai šai gadījumā apvienojam kļūdas gradientus visos laika posmos.

Lai saprastu un vizualizētu atpakaļ izplatīšanos, nepieciešams apskatīt tīklu visos laika posmos. Jāatceras arī, ka RNN var vai nevar būt izejas dati katrā laika posmā un svari ir vienādi katrā laika posmā.

Atpakaļ izplatīšanās laikā pamatprincips ir „izvēršana" [35]. Visi rekurentie svari var tikt dublēti telpiski patvaļīgam laika posmu skaitam. Tas tiek apzīmēts, kā τ. Līdz ar to katram mezglam, kas sūta aktivāciju tiešā vai netiešā veidā pa rekurentu savienojumu ir vismaz τ kopiju skaits, ko var redzēt attēlā 2.11..



Att. 2.11. Tīkla izvēršanas($\tau = 3$)
[35]

Kopējās kļūdas funkcija ir vienkārša standarta kļūdas funkcijas summa katrā laika posmā, ja tīkla apmācība sākas laika brīdi t_0 un beidzas laika brīdi t_1.

Vispirms tiek aprēķināta starpentropijas kļūda, izmantojot pašreizējos izvades datus un faktiskos izvades datus. RNN gadījumā, ja y_t ir paredzamā vērtība un $\overline{y_t}$ ir faktiskā vērtība, kļūdu aprēķina kā savstarpēju entropijas zudumu [36]. To aprēķina konkrētā laika momentā t izmantojot vienādojumus 2.11. un kopējo izmantojot vienādojumu 2.12.:

$$E_t(\overline{y_t}, y_t) = -\overline{y_t} log(y_t) \tag{2.11.}$$

$$E(\overline{y}, y) = -\sum \overline{y_t} log(y_t) \tag{2.12.}$$

Krītošā gradienta metode katrā laika solī veic svaru vērtību pārrēķinu (atjaunošanu). Paredzētajam tīklam gradientu aprēķina katram laika posmam atkarībā no svara parametriem.

Kad svari ir vienādi visos laika posmos, ņem gradientu no visiem laika posmiem un apvieno. Pēc tam svari tiek atjaunoti gan rekurentajiem neironiem, gan pilnībā savienotajiem slāņiem.

Kā piemēru varam apskatīt gadījumu, kad apstrādā pilnu virkni kā vienu apmācības piemēru, piemēram, ja nepieciešams izveidot vārdu. Šajā gadījumā kopējo kļūdu aprēķina, summējot kļūdu katrā laika posmā jeb kļūdu katrai rakstzīmei.

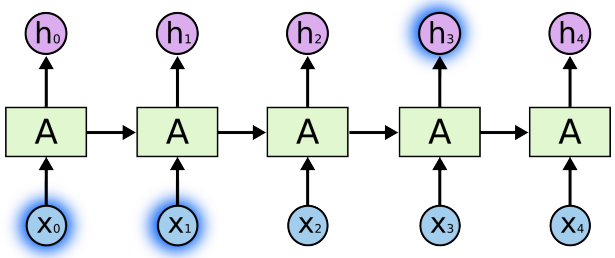
2.2.3. Ilgtermiņa atkarību problēma

Teorētiski RNN būtu jāvar saistīt iepriekšējo informāciju ar pašreizējo informāciju. Piemēram, izmantojot iepriekšējos vārdus, būtu jāvar saprast, kāds būs nākošais vārds.

Dažreiz ir nepieciešams tikai apskatīt jaunāko informāciju, lai veiktu nepieciešamo uzdevumu. Kā piemēru var apskatīt valodas modeli, kur vajag paredzēt nākamo vārdu,

pamatojoties uz iepriekšējiem vārdiem. Ja cenšas prognozēt pēdējo vārdu teikumā *"Mākoņi ir ..."*, nav nepieciešamības pēc papildu konteksta, lai zinātu, ka nākamais vārds visticamāk būs *"debesīs"*.

Šādos gadījumos, ja starpība ir maza starp attiecīgajiem datiem un vietu, kur šie dati ir nepieciešami, tad ir iespējams apmācīt RNN izmantot iepriekšējo informāciju X_0, X_1, lai prognozētu pašreizējo informāciju h_3. Shematiski to var aplūkot attēlā 2.12..

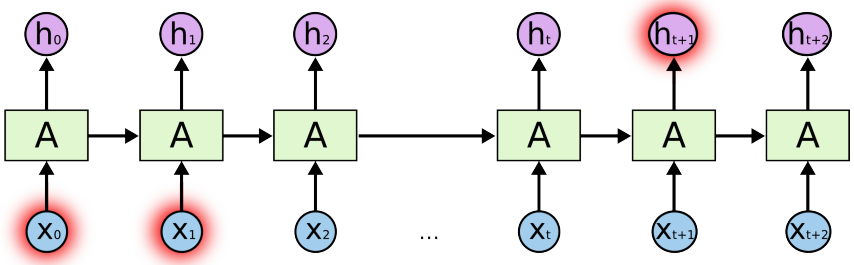


Att. 2.12. Atkārotot neirunu tīklu īstermiņa atkarība, kur $X_0, X_1, X_2, X_3 un X_4$ apzīmē ieejas datus konkrētā laika brīdī, h_0, h_1, h_2, h_3, h_4 izejas dati konkrētā laika brīdī un A reprezentē rekurento neironu tīklu.

[34]

Bet ir arī gadījumi, kad nepieciešams vairāk konteksta. Piemēram ja vēlamies prognozēt pēdējo vārdu tekstā: *"Es uzaugu Francijā ... es brīvi runāju ... valodā."* Visticamāk nākamajam vārdam, iespējams, ir jābūt valodas nosaukumam, bet ja vēlas sašaurināt valodas izvēli, ir nepieciešams zināt iepriekšējo kontekstu, par kuru tika runāts tālā pagātnē.

Šādā gadījumā starpība starp attiecīgo informāciju un vietu, kur šī informācija ir nepieciešama, kļūst ļoti liela. Attēlā 2.13. Jo vairāk šī atšķirība pieaug, jo grūtāk tīklam ir iespējams iemācīties savienot informāciju X_0, X_1, kas ir bijusi pagātnē, lai prognozētu pašreizējo informāciju h_{t+1}.



Att. 2.13. Rekurento neironu tīklu ilgtermiņa atkarība, kur X_t apzīmē ieejas datus konkrētā laika brīdī, h_t izejas dati konkrētā laika brīdī un A reprezentē rekurento neironu tīklu.

[34]

2.3. Ilgi-īslaicīgā atmiņa (Long Short-Term Memory)

LSTM ir jauna rekurentā tīkla arhitektūra, kas izmanto atbilstošu uz gradienta balstītu mācību algoritmu. LSTM tīklam ievērojama ietekme ir bijusi valodu modelēšanā, runā-tekstā transkripcijā, mašīntulkojumā un citas lietojumprogrammās.

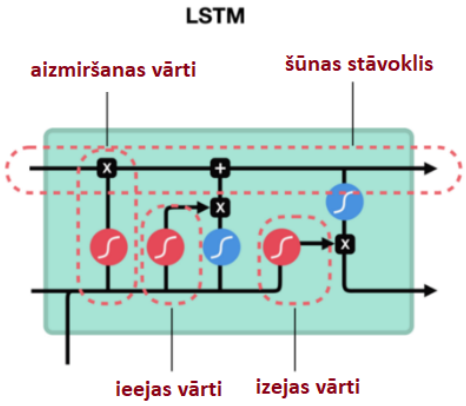
LSTM ir līdzīga kontroles plūsma kā rekurentajam neironu tīklam. Tas apstrādā datus, kas pārsūta informāciju, kad tā tiek izplatīta uz priekšu. LSTM ir izstrādāts, lai pārvarētu kļūdas atpakaļ izplatīšanās problēmas, kas rodas RNN tīklā [37].

Atšķirības ir darbībās, ko veic LSTM šūnas. LSTM paplašina RNN ar atmiņas šūnām, nevis periodiskām vienībām, lai saglabātu un izvadītu informāciju ilgā laika periodā, atvieglojot īstermiņa apmācības procesu.

LSTM arī izmanto vārtus (no angļ. gates), kas ir mehānisms, kas balstīts uz komponenciālo ievades datu reizināšanu, kas nosaka katras atsevišķas atmiņas šūnas uzvedību. LSTM atjauno savu šūnu stāvokli, balstoties uz vārtu aktivizēšanas procesa.

Ievades dati tiek padoti dažādiem vārtiem, kas kontrolē to, kādas darbības tiek veiktas šūnu atmiņā. Tas nozīme, ka šis tīkls var iemācīties savienot laika intervālus pat, ja tie pārsniedz 1000 soļus un ja rodas trokšņaini, nesavienojami ieejas dati [33]. Tas tiek panākts, izmantojot piemērotu gradienta algoritmu tīkla arhitektūrā, kas nodrošina vienmērīgu kļūdas plūsmu caur īpašu vienību iekšējo stāvokli.

LSTM rekurentais slēptais slānis satur īpašas vienības, ko sauc par atmiņas blokiem. Atmiņas bloku var apskatīt attēlā 2.14.. Atmiņas bloki satur atmiņu šūnas, kas satur pašsavienojumus, kas uzglabā īslaicīgu tīkla stāvokli, papildus īpašajām multiplikatīvām vienībām, ko sauc par vārtiem, kas nepieciešami, lai kontrolē informācijas plūsmu.



Att. 2.14. LSTM atmiņas bloka uzbūve
[38]

Katrs atmiņas bloks oriģinālajā tīkla arhitektūras modelī satur ieejas vārtus un izejas vārtus. Ievades vārti kontrolē ieejas aktivizēšanas plūsmu atmiņas šūnā. Izvades vārti kontrolē šūnas izejas plūsmas aktivizēšanu pārējā tīklā.

Vēlāk tīkla arhitektūrai tika mainīta un atmiņas blokam tika pievienoti aizmiršanas vārti. Tas tika izdarīts, lai samazinātu LSTM modeļa problēmu, kas neļauj tiem veikt nepārtrauktu ievades datu plūsmas apstrādi, ja dati nav segmentēti pa apakškopām.

Turklāt mūsdienu LSTM arhitektūrā ir pievienots ķēdes savienojums no iekšējām šūnām uz vārtiem, kas atrodas tai pašā šūnā, lai būtu iespējams noteikt precīzu izejas laiku [34].

Bet tomēr izmantojot LSTM jārēķinās ar to, ka šis tīkls cieš no augstas sarežģītības slēptajos slāņos. Tipiskam LSTM ir aptuveni četras reizes vairāk parametru nekā vienkāršam RNN. Kaut arī LSTM ir parādījis uzlabotas spējas iemācīties ilgtermiņa laika atkarības, tomēr tiek apgalvots, ka tīkla aizture mehānismiem nav iespējas visaptveroši diskriminēt būtiskāko un nesvarīgāko informāciju [39]. Tāpēc LSTM grūti ir uzdevumi, kā darbības atzīšana, kur sekvences bieži var saturēt daudzus nesvarīgus kadrus.

2.3.1. Arhitektūra

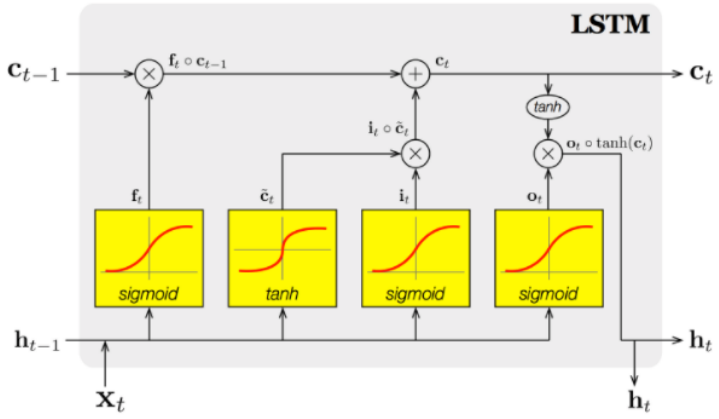
Aizmiršanas vārti

Pirmkārt, ir aizmiršanas vārti. Šie vārti izlemj, kāda informācija ir jāizmet un ko nepieciešams saglabāt. Informācija no iepriekšējā slēpta stāvokļa un informācija no pašreizējās ievades tiek nodota caur sigmoidas funkciju.

Tiek atgrieztas vērtības no 0 līdz 1. Jo tuvāk 0 vērtībai nozīmē, ka informāciju var aizmirst, un jo tuvāk 1 nozīmē, ka informāciju jāsaglabā. Aizmiršanas vārti mēra šūnas iekšējo stāvokli pirms pievienot to tā kā ieejas datus šūnai, izmantojot šūnas pašregulācijas savienojumu, tādējādi adaptīvi aizmirstot vai atiestatot šūnas atmiņu [38].

Ieejas vārti

Lai atjauninātu šūnas stāvokli, ir ieejas vārti. Pirmkārt, tiek nodots iepriekšējais slēptais stāvoklis un pašreizējā ievade sigmoidas funkcijai. Tas izlemj, kuras vērtības tiks atjauninātas, pārveidojot vērtības no 0 līdz 1. 0 nozīmē, ka tās nav svarīgas, un 1 ir svarīgas. Arī tiek nodots slēptais stāvoklis un pašreizējā ievade $tanh$ funkcijai, lai saspiestu vērtības starp -1 un 1, tas palīdz regulēt tīklu. Tad sareizina $tanh$ izvades datus ar sigmoidas izvades datiem. Sigmoidas funkcija izlems, kura informācija ir svarīga no $tanh$ funkcijas atgrieztās informācijas. Šīs funkcijas var redzēt attēlā 2.15..



Att. 2.15. LSTM Atmiņas šūnas elementi
[40]

Šūnu stāvoklis

Tagad vajadzētu būt pietiekamai informācijai, lai aprēķinātu jauno šūnas stāvokli. Pirmkārt, šūnas stāvoklis tiek reizināts ar aizmiršanas vektoru. Šai gadījuma ir iespēja izmest konkrētas vērtības šūnas stāvoklī, ja šūnas stāvokli reizināta ar vērtībām, kas ir tuvu pie 0. Pēc tam ņem izejas datus no ievades vārtiem un veic saskaitīšanu, šādā veidā atjauno šūnas stāvokli ar jaunām vērtībām, kuras neironu tīkls uzskata par atbilstošām. Beigās tiek iegūts jauns šūnu stāvoklis.

Izejas vārti

Pēdējais posms ir izvades vārti. Izvades vārti nosaka, kādam jābūt nākamajam slēptajam stāvoklim. Slēptais stāvoklis satur informāciju par iepriekšējiem ievades datiem. Slēpto stāvokli izmanto arī prognozēm.

Pirmkārt, iepriekšējo slēpto stāvokli un pašreizējos ievades datus padod sigmoidas funkcijai. Tad nodod jauni veidoto šūnu stāvokli $tanh$ funkcijai. Reizina $tanh$ funkcijas izvades datus ar sigmoidas izvades datiem, lai izlemtu, kādai informācijai jābūt slēptā stāvoklī. Šīs funkcijas var redzēt attēlā 2.15.. Un no tā veidojas jaunie izvades dati, kas ir slēptais stāvoklis. Jaunais šūnu stāvoklis un jaunais slēptais stāvoklis tiek pārnests uz nākamo laika soli.

Ķēdes savienojumi

Viena LSTM variācija ir tāda, ka šūnai tiek pievienoti ķēdes savienojumi. Tas nozīmē, ka vārti var aplūkot šūnu stāvokli. Bieži vien tiek veidoti ķēdes savienojumi, kas ļauj vārtiem būt atkarīgiem ne tikai no iepriekšējā slēptā stāvokļa st − 1, bet arī no iepriekšējā iekšējā stāvokļa ct − 1. Ķēdes savienojumi palīdz iemācīties precīzus notikumu laikus. LSTM, ko papildina ar ķēdes savienojumiem spēj no iekšējām šūnām līdz multiplikatīvajiem vārtiem iemācīties smalka atšķirība starp pīķu secībām, kas izvietotas piemēram 50 vai 49 laika posmos attālumā.

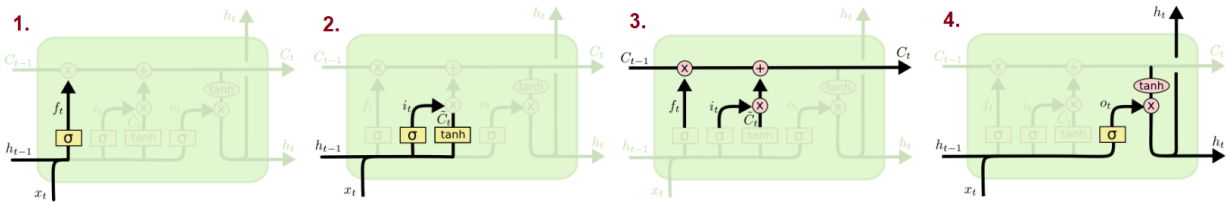
Savienotie aizmiršanas un ievades vārti

Vēl viena variācija ir savienot aizmiršanas un ievades vārtus. Tā vietā, lai atsevišķi nolemtu, ko aizmirst un kādu jaunu informāciju vajadzētu pievienot, šis lēmums tiek pieņemts kopā. Tātad sūnā aizmirst tikai tad kad ir informācija ar kuru var aizvietot iepriekšējo informāciju un pretēji.

Slēgtās Rekurentās Vienības

Slēgtās Rekurentās Vienības (angļ. val. Gated Recurrent Unit) vai GRU ir LSTM variācija. Šajā modelī tiek apvienots šūnu stāvoklis un slēptais stāvoklis un notiek vēl citas izmaiņas. Atšķirībā no LSTM, GRU ir vienkāršāka arhitektūra un prasa mazāk aprēķinu. Vairāk var par šo tīklu var uzzināt [34].

2.3.2. Šūnas un slēptā stāvokļa atjaunošana



Att. 2.16. Tīkla vienības aktivizēšanās shēma
[34]

LSTM tīkls izveido ieejas sekvences kartēšanas secību [41], kuru var apskatīt attēlā 2.16., no $x = (x_1, ..., x_T)$ līdz izejas virknei $y = (y_1, ..., y_T)$. To dara tā, ka aprēķina tīkla vienības aktivizēšanos.

Lai aprēķinātu tīkla vienības aktivizāciju, atmiņas bloka elementiem tiek izmantoti dažādi vienādojumi, darbības tiek veiktas iteratīvi no $t = 1$ līdz T.

Aizmiršanas vārtu vienādojumu apraksta ar formulu 2.13.. Ievades vārtu vienādojumu apraksta ar formulu 2.14.. Ieejas modulācijas vārtu vienādojumu apraksta ar formulu 2.15.. Šūnas stāvokļa vienādojumu apraksta ar formulu 2.16.. Izvades vārtu vienādojumu apraksta ar formulu 2.17.. Slēptā stāvokļa vienādojumu apraksta ar formulu 2.18.. Izvades datu vienādojumu apraksta ar formulu 2.19..

$$f_t = \sigma(W_{fx}x_t + W_{fh}h_{t-1} + W_{fc}c_{t-1} + b_f) \tag{2.13.}$$

$$i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1} + W_{ic}c_{t-1} + b_i) \tag{2.14.}$$

$$\bar{c} = g(W_{cx}x_t + W_{ch}h_{t-1} + b_c) \tag{2.15.}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \overline{c} \tag{2.16.}$$

$$o_t = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + W_{oc}c_t + b_o) \tag{2.17.}$$

$$h_t = o_t \odot m(c_t) \tag{2.18.}$$

$$y_t = \phi(W_{yt}ht + b_y) \tag{2.19.}$$

W termini apzīmē svara matricas (piemēram, W_{ix} ir svaru matrica no ieejas vārtiem līdz ievadam),W_{ic}, W_{fc}, W_{oc} ir diagonālās svara matricas ķēdes savienojumiem. b termini apzīmē nobīdes (angļ. bias)vektorus (b_i ir ieejas vārtu nobīdes vektors). σ ir loģistikas sigmoidas funkcija. i, f, o un c ir attiecīgi ievades vārti, aizmiršanas vārti, izejas vārti un šūnu aktivācijas vektori, visi šie lielumi ir tādi paši kā šūnas izvades aktivizēšana vektors h, h šūnas aktivizēšana vektors. c_{t-1} iepriekšējā šūnas aktivizācijas vektors, $\odot$ ir vektoru elementi, g un m ir šūnas ievades un šūnu izvades aktivizācijas funkcijas, parasti $tanh$. ϕ ir tīkla izvades datu aktivizācijas funkcija piemēram, $softmax$.

1. Vispirms tiek apvienots iepriekšējais slēptais stāvoklis un pašreizējā ievade. Šo procesu sauc par apvienošanu. Apvienotie dati tiek padoti aizmiršanas vārtiem. Šajā slānī tiek izdalīti dati, kuri netiks izmantoti tālāk. Tiek apskatīts h_{t-1} un x_t, un katram c_{t-1} elementam tiem izdots skaitli no 0 līdz 1, lai noteiktu ko aizmirst. Kā vizuāli, tas ir attēlots var redzēt attēla 2.16. pirmajā solī.

2. Kandidāta slānis tiek izveidots, izmantojot apvienošanas datus. Kandidāts satur iespējamās vērtības, kuras var pievienot šūnas stāvoklim. Tālāk apvienotie dati arī tiek padoti ievades slānim. Šis slānis nosaka, kādus kandidāta datus vajadzētu pievienot jaunajam šūnu stāvoklim. Vispirms sigmoids slānis, ko sauc par ieejas vārtu slāni, nolemj, kuras vērtības tik atjauninātas. Tālāk $tanh$ slānis rada jaunu kandidātu vērtību vektoru $\overline{c_t}$, kas varētu būt pievienots stāvoklim. Trešajā posmā apvieno kandidātu vektoru un ieejas vārtu slāni, lai izveidotu stāvokļa atjauninājumu. Kā vizuāli, tas ir attēlots var redzēt attēla 2.16. otrajā solī.

3. Pēc aizmiršanas slāņa, kandidāta slāņa un ievades slāņa aprēķināšanas tiek aprēķināts šūnas stāvoklis. Tiek izmantoti visi šie vektori un iepriekšējais šūnas stāvoklis. Tiek reizināts vecais stāvoklis ar f_t, aizmirstot informāciju, kuru nolēma aizmirst agrākajā fāzē. Tad pievieno $i_t \odot \overline{c_t}$. Tās ir jaunās kandidāt vērtības, kas ir mērogotas atkarībā no tā cik daudz ir nolemts atjaunināt katru šūnas stāvokļa vērtību. Kā vizuāli, tas ir attēlots var redzēt attēla 2.16. trešajā solī.

4. Tiek aprēķināti izejas dati. Vispirms izmanto sigmoidu slāni, kas izlemj, kādas šūnu stāvokļa daļas tiks izvadītas. Šūnu stāvoklis iet caur $tanh$, lai mērogotu vērtības no -1 līdz 1. Pēc tam šie dati tiek reizināti ar sigmoid vārtu izejas datiem, lai tiktu izvadīta tikai informācija, kuru nolēmām atcerēties iepriekš. Sareizinot izejas datus un jauno šūnu stāvoklis tiek iegūts jaunais slēptais stāvoklis. Kā vizuāli, tas ir attēlots var redzēt attēla 2.16. ceturtajā solī.

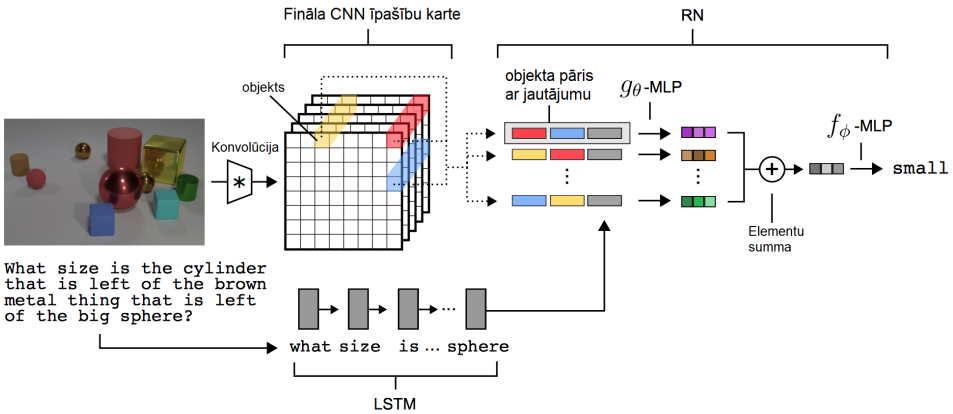
2.4. Apvienotā arhitektūra (CNN + LSTM)

CNN - LSTM arhitektūra atšķiras no tradicionāli tīras CNN vai tīras LSTM arhitektūras. Pirmā puse ir CNN, un tā tiek izmantota īpašību iegūšanai. Pēdējā daļa ir LSTM, kas veic prognozēšanu, izmantojot īpašības, ko ieguva CNN, lai analizētu un novērtētu kādi būs izejas dati nākamajā laika posmā.

Turklāt eksperimenti liecina, ka LSTM un CNN iemācās dažādas īpašības, kad tiek apmācīti pie tiem pašiem datiem. Šī atšķirība rada iespēju apvienot īpašības, kuras iemācījušies gan CNN un LSTM, vienā modelī. Tas dod iespēju iegūt labāku precizitāti nekā tad, kad tiek izmantos katrs modelis patstāvīgi [42].

CNN - LSTM parasti tiek izmantots, ja ievades datiem ir telpiska struktūra, piemēram, 2D struktūra vai pikseļi attēlā 2.17. vai arī 1D vārdu struktūra teikumā, rindkopā vai dokumentā. Kā arī ievades dati satur laika struktūra, piemēram, secīgi attēli, kas veido video, vai vārdi tekstā vai arī, ja nepieciešams ģenerēt izejas datus, kam piemīt laika struktūra, piemēram, aprakstošus teikumus [43].

Tātad CNN - LSTM arhitektūra ietver CNN slāņu izmantošanu, lai veiktu ievades datu aprakstošo īpašību izgūšanu un apvienojot ar LSTM tīkls var veikt virknes datu prognozēšanu.

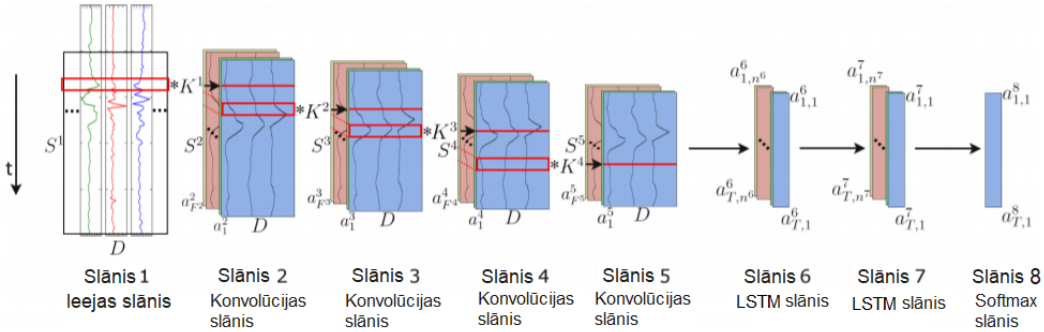


Att. 2.17. Modelim ir jāaplūko dažādu formu, izmēru, krāsu objekti un jāspēj atbildēt uz jautājumiem, kas saistīti ar vairākiem šādiem objektiem.

[44]

CNN un LSTM kombinācija vienotā sistēmā jau ir piedāvājusi jaunākos rezultātus runas atpazīšanas domēnā, kur nepieciešama modelēšana, kas prasa laiku informāciju [45]. Šāda veida arhitektūra spēj uztvert laika atkarību no īpašībām, kas iegūtās izmantojot konvolūcijas operāciju.

Kā piemēru varam apskatīt $DeepConvLSTM$ struktūras arhitektūru 2.18., kas tik izmantota avotā [46], lai atpazītu aktivitātes.



Att. 2.18. DeepConvLSTM arhitektūru
[46]

Attēlā 2.18. no kreisās puses signāli, kas nāk no valkājamajiem sensoriem, tiek apstrādāti ar četriem konvolūcijas slāņiem, kas ļauj mācīties no datiem. Pēc tam divi pilnībā savienotie slāņi veic nelineāru transformāciju, kas dod klasifikācijas rezultātu izejas slānī pa labi ar $softmax$ loģistisko regresiju.

1. slāņa ievade dati atbilst sensora datiem, kuru izmērs ir DxS^1. D apzīmē numuru sensoru kanālu un S^l īpašības karšu garuma konkrētā slānī l.

2. – 5. Slāņi ir konvolūcijas slāņi. K^l apzīmē kodolus konkrētā l slānī (attēloti kā sarkani kvadrāti). F^l apzīmē īpašību karšu skaitu konkrētā l slānī. Konvolūcijas slāņos a_i^l apzīmē aktivāciju, kas nosaka īpašību karti i slānī l.

6. un 7. slānis ir LSTM slāņi. LSTM slāņos: $a_{t,i}^l$ apzīmē vienības i aktivizēšanu slēptajā slānī l laikā posmā t. Laika ass ir vertikāla.

Šajā modelī netiek lietoti apvienošanas slāņi, jo tiek uzskatīts, ka apvienošanas slāņu izmantošana pēc konvolūcijas slāņiem traucē konvolūcijas slāņu spējai iemācīties samazināt neapstrādāta sensora datus.

Apkopojumā modelim ieejas dati tiek padoti CNN slānim, kur notiek datu transformācijas, kas aprakstītas iepriekš nodaļā par Konvolūcijas neironu tīkliem, un galā tiek iegūta ieejas datu reprezentācija īpašību telpā. Pēc tam pēdējā CNN slāņa iegūtās īpašības tiek izmantotas LSTM slāņos, lai iegūtu secīgu laika informāciju. Pēdējā LSTM slāņa pēdējo slēpto vektoru izmanto $softmax$ klasifikators, lai ģenerētu klašu etiķetes, kas tiek salīdzināta ar patieso klašu etiķeti, tādā veidā nosakot tīkla precizitāti.

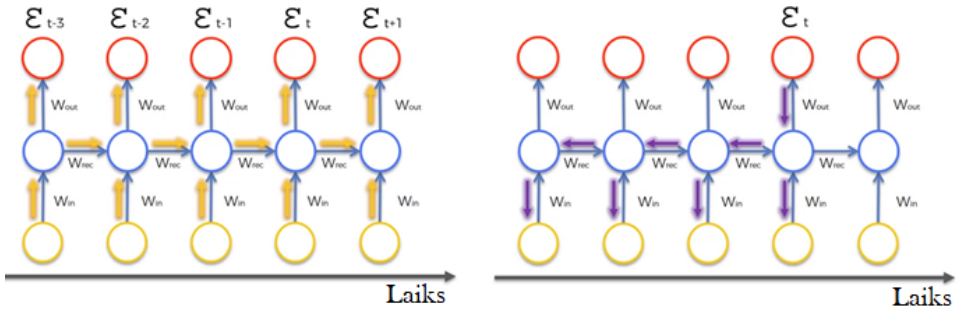
2.5. Izzūdošais un eksplodējošais gradients

Stohastiskais krītošais gradients (angļ. stochastic gradient descent) šķiet pietiekami vienkāršs process, bet daudzos tīklos var pamanīt, ka svari, kas ir tuvāk tīkla beigām, mainās daudz vairāk nekā sākumā [47]. Un jo dziļāks tīkls, jo mazāk un mazāk sākuma slāņi mainās.

Problēma rodas, jo svari tiek inicializēts nejauši. Ja tie gandrīz vai nemainās, tad tie nekad nespēs sasniegt pareizās vērtības vai arī apmācība aizņems gadus.

RNN apmācības laikā aprēķina izeju datus ar vēlamajiem izejas datiem. Un apmācības laikā kļūdas funkcija salīdzina iegūtos rezultātus, kas attēlā 2.19. ir attēloti ar sarkaniem apļiem, ar vēlamo rezultātu. Beigās iegūst šīs vērtības visā laika rindā, katram atsevišķam sarkanajam aplim.

Pēc tam kļūdas funkcijas (no angļ. cost function) vērtība tiek atpakaļ izplatītas tīklā atkarībā no tā, cik daudz laika soļi tikuši veikti, to var redzēt pa labi attēlā 2.19..



Att. 2.19. Gradienta aprēķināšana gaita. Pa kreisi parādīts, kā tiek aprēķināta kļūda katrā laika posmā un pa labu tiek parādīts, kā notiek kļūdas atpakaļ izplatīšana
[48]

Kā piemēru, varam apskatīt vienu kļūdas posmu e_t kādā laika brīdī t.

Pirmkārt, tiek aprēķināta kļūdas funkcija (cost function) e_t, un pēc tam nepieciešams izplatīt kļūdas funkciju atpakaļ visā tīklā, jo ir nepieciešams atjaunināt svarus. Būtībā ikvienam neironam, kas piedalījās izejas datu aprēķināšanā un ir saistīts ar kļūdas funkciju, nepieciešams veikt svaru atjaunošanu, lai mazinātu kļūdu.

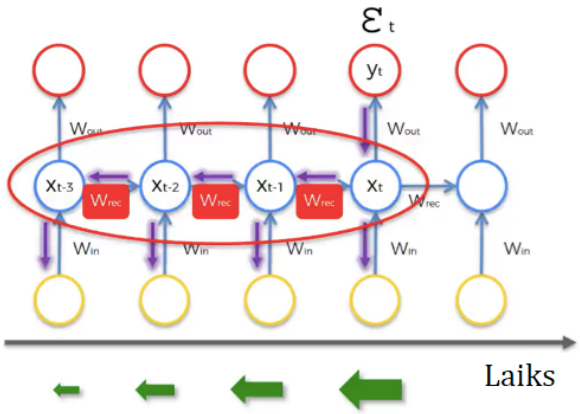
Izmantojot RNN ir jāatceras, ka ne tikai neironiem, kas darbojas tieši zem konkrētā izvades slāņa, bet visiem neironi, kas atrodas tālu atpakaļ laikā, nepieciešams atjaunot svarus. Tas nozīme, ka ir nepieciešams doties atpakaļ laikā līdz šiem neironiem.

Problēma ir tieši saistīta ar W_{rec} atjaunināšanu. W_{rec} ir svari, kas tiek izmantoti, lai savienotu slēptos slāņus pašus ar sevi izkliedētā laika cilpā un tos sauc par rekurentajiem svariem. Piemēram, lai nokļūtu no x_{t-3} līdz x_{t-2} ir jāreizina x_{t-3} ar W_{rec}. Tad, lai nokļūtu no x_{t-2} uz x_{t-1}, atkal notiek reizināšana x_{t-2} ar W_{rec}.

Tātad, rezultātā reizina ar vieniem un tiem pašiem svariem vairākas reizes, un tieši šeit rodas problēma. Ja reizina kaut ko ar mazu skaitli iegūstam to, ka vērtība samazinās ļoti ātri.

Svari sākumā tiek izinicializēti ar gadījuma vērtībām, kas ir tuvu nullei, un no turienes tīkls sāk pielāgot svarus apmācības procesā. Bet, ja sākt ar W_{rec}, kura vērtības ir tuvu nullei un pēc tam reizināt $x_t, x_{t-1}, x_{t-2}, x_{t-3}, ...$ ar šīm vērtībām, gradients ar katru reizinājumu kļūst arvien mazāks un mazāks. Un jo mazāks ir gradients, jo grūtāk tīklā atjaunināt svarus un ilgāk, lai sasniegtu gala rezultātu [48].

Apkopojumā, ja W_{rec} ir mazs, tad ir izzūdošā gradienta problēma, un, ja W_{rec} ir liels, rodas eksplodējošā gradienta problēma 2.20.. Izzūdošā gradienta problēmas gadījumā, jo tālāk ejam cauri tīklam, jo zemāks ir gradients, un tas noved pie tā, ka kļūst grūtāk apmācīt svarus. Tas ir kā domino efektu, kas ietekmē visus turpmākos svarus visā tīklā.



Att. 2.20. Gradienta izzušanas un eksplodēšanas problēmas attēlojums [?]

Eksplodējošā gradienta gadījumā varam:

- Pārtraukt atpakaļ izplatīšanu pēc noteikta punkta, bet tas parasti nav optimāls, jo ne visi svari tiek atjaunināti;
- Sodīt vai mākslīgi samazināt gradientu;
- Noteikt maksimālo gradientu robežu.

Izzūdošā gradienta gadījumā varam:

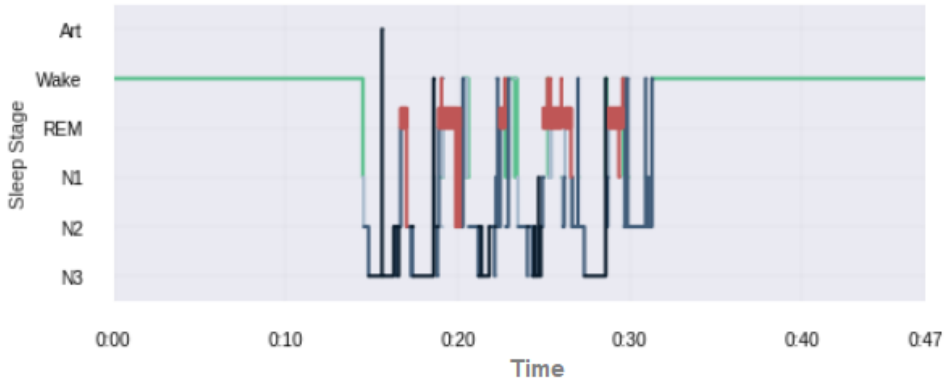
- Inicializēt svarus tā, lai samazinātu izzušanas gradienta potenciālu;
- Izmantot Atbalsta stāvokļa tīklus (angļ. Echo State Networks), kas paredzēti izzūdošā gradienta problēmas risināšanai;
- Izmantot Ilgi-īslaicīgos atmiņas tīklus (angļ. Long Short-Term Memory Networks).

3. Prakstiskā daļa

3.1. Datu sagatavošana

Datu kopa

Tika apskatītas dažādas miega EEG datu bāzes. Un tika izlemts lietot datubāzi [49], lai veiktu tīklu apmācību un testēšanu. Šī datubāze satur 61 polisomnogrammu kolekciju ar pavadošām hipnogrammām. Šajā datu kopā visi EDF galvenes lauki atbilst arī EDF + specifikācijām, kas ļauj izmantojot Visbrain pieejamo funkcionalitāti, lai nolasītu hipnogrammas datus. Hipnogrammā 3.1. uz y ass tiek attēlota informācija par miega fāzēm un uz x ass tiek attēlots laiks, kuru cilvēks pavada miegā. Šajā gadījumā visas nakts ieraksta laiks tiek apzīmēts sekundēs, sākot ar 0:00 līdz 0:47. Datubāzes EDF faili ir pilnas nakts polisnogrāfiski miega ieraksti, kas satur EEG, EOG (Elektrookulogrāfija),deguna gaisa plūsmu, taisnās zarnas temperatūru, submentālo EMG (Elektromogramma) un notikumu marķieri. Daļa EDF failu bieži vien satur arī informāciju par gaisa nazālo elpošanu un rektālo ķermeņa temperatūru. Tīkla apmācībai tiek izmantoti dati no diviem kanāliem EEG Fpz-Cz un EEG Pz-Oz.



Att. 3.1. Hipnogrammas faila SC4001EC-Hypnogram.edf grafisks attēlojums

Apmācību un testa dati

Balstoties uz hipnogrammu un to attiecīgo EFD failu datiem tika paņemti EEG Fpz-Cz un EEG Pz-Oz signāli, kas tika sadalīti pa apakšsignāliem. Apakšsignālu garums ir 3000x2, jo hipnogrammas iztveršanas frekvence ir 0.03 un katram šādam intervālam pēc hipnogrammas datiem tiek piešķirta konkrēta miega fāze. Iegūtie signāli tika sadalīti pa attiecīgajām klasēm, kur katra klase atbilst konkrētai miega fāzei.

Balstoties uz dotās informācijas no hipnogrammam, par miega fāžu sadalījumu, tīkla apmācīšanai tikai ņemti 500 EEG signālu paraugi ar izmēru 3000x2, kas atbilst konktētām miega fāzēm atšķirīgos laika momentos. Šie paraugi tika ņemti nejaušā secībā no septiņiem EDF failiem. Pēc tam šie paraugi tika sadalīti intervālos ar garumu 500 vai 1000 vai atstāti ar orģinālo

garumu 3000. Tas ir atkarīgs kādus gadījumus vēlamies testēt. Ja ņemam ievades datu formu 1000x2, tad tīklam kopā būs 7500 apmācības datu. Kad datu forma ir 500x2 tad apmācības datu kopā būs 15000. Kad ieejas datu forma ir 3000x2 tad kopējie apmācības dati ir 2500.

Tīkla validācijai no atlikušajiem datiem tika ņemti 100 EEG signālu paraugi no katras miega fāzes ar izmēru 3000x2. Arī šie signāli tika sadalīti tādos pašos intervālos, kā norādīts ieejas datu izmērā. Tātad, ja ieejas dati ir 3000x2, tad kopējais validācijas datu skaits ir 500, 1000x2 gadījumā 1500 un 500x2 gadījumā 3000 datu. Apmācības un validācijas datu sadalījumu var apskatīt tabulā 3.1..

Testēšanai tiek izmantots viss hipnogrammas fails SC4001EC-Hypnogram.edf un attiecīgais EDF fails SC4001E0-PSG.edf. Abi EEG signāli no EDF faila tiek sadalīti līdzīgos paraugu garumos kā apmācības gadījumā, kur ieejas datu formāts ir 1000x2, 3000x2 un 500x2. Koda piemēru var apskatīt pielikumā 5..

Tabula 3.1.

Apmācības un validācijas datu sadalījums

Ievades datu forma	Apmācības dati	Validācijas dati
500 x 2	15000	3000
1000 x 2	7500	1500
3000 x 2	2500	500

Datu sagatavošanas metodes

Turpmākā sadaļā tiks aprakstītas sekojošas funkcijas, kuras tiek izmantotas datu kopas sagatavošanai:

- retrieveHypnogramData_2v
- seperateSignalsFromEDF
- divideSignalInSleepStages
- getStageTrainingAndValidationData

Pirmā metode $retrieveHypnogramData_2v$, kas atrodama pielikumā 6., ir paredzēta hipnogrammas datu izgūšanai. Kā ieejas datus nepieciešams padot mainīgo *path_to_hypno*, kas ir hipnogrammas faila atrašanās vieta un mainīgo *path_to_datafile*, kas ir attiecīgās hinpogrammas EDF fails, kas satur EEG signālu informāciju. Metode *read_hypno*, kas ņemta no Visbrain Sleep bibliotēkas, atgriež mainīgo data, kas satur vērtības no 0 līdz 5, kuras atbilst konkrētai miega fāze (0-"Nomoda", 1 -"N1", 2-"N2", 3-"N3", 4-"REM", 5-"Art"). 5 klase, kas šajā gadījumā ir artefakti, netiek ņemta vērā tīkla apmācīšanā un testēšanā. Un mainīgais sf, kas

tiek atgriezts, ir iztveršanas frekvence, tā ir svarīga, lai būtu zināms, cik liels ir viena parauga izmērs.

Otrā metode $seperateSignalsFromEDF$, kas atrodama pielikumā 6., atgriež datu struktūru ar vienu kolonu $Signal$ un divām rindām, pirmā satur Fpz-Cz un otrā satur Pz-Oz signālu. Kā ieejas datus arī nepieciešams padot manīgo $datafile$, kas ir EDF faila, kas satur EEG signālu informāciju. Mainīgais n tiek nodefinēts kā 2, jo no faila nepieciešams nolasīt tikai Fpz-Cz un EEG Pz-Oz signālus. Lai ielasītu informāciju no EDF faila tiek izmantota $pyedflib$ bibliotēkas metode $EdfReader$. Tālāk tiek izveidota jauna datu struktūra $eegSignals$ ar vienu kolonu, kur glabās EEG signālu informāciju. Nākošais solis ir izgūt signālus no ielasītā faila. Ciklā no faila tiek nolasīta attiecīgā signāla informācija izmantojot readSignal metodi un šī metode atgriež masīvu ar attiecīgā signāla vērtībām. Šo masīvu saglabā izveidotajā datu struktūrā. Kad tiek iegūti abu signālu dati, fails tiek aizvērts un mainīgais d tiek izdzēsts. $eegSignals$ tiek atgriezts.

Trešajai metodei $divideSignalInSleepStages$, atrodama pielikumā 6., tiek padots mainīgais *sleep_stages*, kas satur hipnogrammas datus un mainīgais *sleep_signal*, kas satur EEG signāla datus. Pirmais solis ir iegūt parauga izmēru, mainīgo $chunks$ aprēķina kopējo signāla garumu dalot ar hipnogrammas kopējo elementu skaitu. Mainīgais $step$ ir sākuma pozīcija un tas tiek inicializēts kā 0. Mainīgais $nextStep$ ir beigu pozīcija. Tālāk nodefinē datu struktūru *data_set*, ar divām kolonām. $Signal$ kolona saturēs apakšsignālu informācija un $Class$ kolona saturēs informāciju par to, kurai miega stqadījai pieder konkrētais apakšsignāls. Tālāk ciklā tiek iets cauri hipnogrammas datiem. Mainīgais $subsignal$ saturēs apakšsignālu, kurš tiek ņemts no kopējā signāla, pēc noteikta intervāla. Intervāla sākumu nosaka manīgais step un beigu pozīciju nosaka mainīgais $nextStep$. Sākuma pozīcija tagad ir beigu pozīcija un jaunā beigu pozīcija ir sākuma pozīcija, kam pieskaitīts mainīgais $chunks$. Mainīgais $signal_with_stage$ saturēs masīvu ar apakšsignāla vērtībām un tālāk šis masīvs tiek saglabāts datu struktūrā *data_set*, kā arī hipnogrammas elements, kas apzīmē miega stadīju, arī tiek saglabāts šajā datu struktūrā. Kad tiek iziets cauri visiem hipnogrammas elementiem, tiek atgriezta datu struktūra *data_set*, kas satur apakšsignālu un to miega fāžu informāciju.

Ceturtā metode $getStageTrainingAndValidationData$, kas atrodama pielikumā 6., ir paredzēta testa, apmācības un validācijas datu kopu sagatavošanai. *signal_data1* mainīgais ir pirmā kanāla EEG signāla dati, $signal_data2$ mainīgais ir otrā kanāla EEG signāla dati. $classOfSamples$ mainīgais apzīmē, kuras miega fāzes datus vēlamies sadalīt . $traingSize$ mainīgais apzīmē to cik daudz datu tiks izmantots tīkla apmācībai, $testSize$ mainīgais apzīmē to cik daudz datu tiks izmantots tīkla testēšanai un $validationSize$ mainīgais apzīmē to cik daudz datu tiks izmantots tīkla validācijai. Sākumā tiek nodefinētas datu struktūras *eeg_test*, kas saturēs testa datus,*eeg_val*, kas saturēs validācijas datus un *eeg_train*, kas saturēs apmācības datus. Tiek arī nodefinēta datu struktūra $temp$, kas saturēs konkrētu signālu datus. Ciklā tiek iets cauri visiem pirmā kanāla signāliem. Tiek izgūta miega fāze un abu kanālu signālu dati, kas pēc tam tiek saglabāti datu struktūrā $temp$. Un pēc tam tiek veikta *eeg_test*,*eeg_val* un *eeg_train*

izmēru pārbaude, lai zinātu vai datu struktūrām var pievienot vēl vienu rindu ar datiem. Pēc tam, kad ir iziets cauri visiem kanāla signāliem, tiek atgrieztas šīs datu struktūras, kuras pēc tam tiks padotas izveidotajam partiju (no angļ. batch) ģenerātoram.

Tīkla apmācīšana

Balstoties uz pamācību [50] tika izveidots, partiju (no angļ. batch) ģenerātors. Šis ģenerātors tiek lietots, lai ģenerētu datu kopu uz vairākiem kodoliem reālajā laikā un izmantojot šo ģenerātoru ir iespējams padot izveidoto datu kopu uzreiz izveidotajam tīkla modelim, tādā veidā samazinot atmiņas patēriņu. Turpmākā sadaļā tiks aprakstītas sekojošas funkcijas, kuras tiek izmantotas tīkla apmācībā:

- __data_generation
- reshape
- to_categorical
- MyDatGen
- ModelCheckpoint
- model.fit_generator

Partiju ģenerātora izejas kodā bija nepieciešams pārveidot *__data_generation* metodi, lai šī metode spētu izveido datu partijas no sagatavotās datu kopas signāliem, kas tiks padotas tīklam. Pielāgoto metodi iespējams apskatīt pielikumā 6..

Kā ieejas datus nepieciešams padot sarakstu ar indeksiem *list_indexes_temp*, kas apzīmē to, kuras rindas dati tiks ņemti no kopējās datu struktūras $dataSet$. Mainīgais y, saturēs miega fāzes jeb klases, sākumā tas ir tukšs masīvs. Mainīgais $chunk$ nosaka cik liels būs apakšsignāls. Mainīgais $parts$ nosaka to cik daļās nepieciešams būs dalīt kopējo signālu. k ir skaitītājs, kas būs nepieciešams, lai sekotu līdz cik daudz apakšsignālu tiek pievienots konkrētajai partijai. a ir mainīgais, kas saturēs iegūtos apakšsignālus no divu kanālu signālu datiem.

Tālāk tiek ciklā iets cauri indeksiem, lai noteiktu, kurus signālus nepieciešams sadalīt pa apakšsignāliem un iekļaut partijā. $stepOne$ ir sākuma pozīcija, ko inicializē ar 0. Tiek izgūta konkrēto signālu miega fāze un saglabāta mainīgajā cl, kā arī no $dataSet$ tiek izgūti signāli no abiem kanāliem un saglabāti mainīgajā sig.

Tālāk šie signāli tiek izdalīti atsevišķi, $sign$ satur informāciju par pirmā kanāla signālu un $sign2$ satur informāciju par otrā kanāla signālu.

Nākošais solis ir ciklā iet cauri mainīgajam $parts$, kas nosaka cik daļās jādala šie signāli. Vispirms ir pārbaude vai nav sasniegts partijas atļautais izmērs un pēc tam abu kanālu signāli

tiek sadalīti apakšsignālos un saglabāti mainīgajā a. Mainīgajā y saglabā attiecīgo signālu miega fāzi.

Pēc tam nepieciešams iegūto datu kopu pārveidot, lai tā saturētu pareizu dimensiju skaitu. To dara izmantojot metodi $reshape$. *a_new* forma pēc tam būs šāda (signālu skaits, garums, kanāli). *expand_dim* mainīgais nosaka to vai nepieciešams pievienot papildus dimensiju, jo ir tīkla modeļi, kuriem nepieciešams padot datus, kas satur 4 dimensijas nevis trīs. Ja ir nepieciešams pievienot papildus dimensiju, tad izmanto *expand_dims* metodi. *a_new* forma tad izskatītos šādi - (signālu skaits, platums, garums, kanāli).

Tiek atgriezti iegūtie apakšsignāli un to klases, kas tiek pārveidotas kategoriskā formātā izmantojot metodi *to_categorical*.

Kad dati ir sagatavoti, tie vēl ir jāpadod tīklam, lai varētu sākt apmācīt tīklu. Realizāciju kodā var apskatīt pielikumā 6..

Vispirms tiek ģenerēts saraksts priekš apmācības datu kopas, kas reprezentē katras rindas indeksu un satur skaitļus no 0 līdz datu kopas izmēram . Nākošais solis ir konvertēt datu struktūras $dataSet$ kolonu, kas satur miega fāzes jeb signālu klases, par sarakstu. To pašu atkārto arī validācijas datu kopai.

Nākošais solis ir izveidot partiju ģeneratoru priekš apmācības un validācijas, ko izmantos apmācot tīklu, izmatojot metodi $MyDatGen$. *list_of_Indexes* ir mainīgais , kas satur katras datu kopas rindas indeksu. *batch_size* mainīgais nosaka partijas lielumu. $dataSet$ ir mainīgais, kas satur datu kopu.

Tiek izveidoti divi partiju ģeneratori. Viens priekš apmācības datiem un otrs priekš validācijas datiem. Parametri, kas jāpadod ir $labels$, kas ir mainīgais, kas satur miega fāzes jeb klases. *expand_dim* ir mainīgais, kas nosaka vai ir nepieciešams ieejas datiem pievienot papildus dimensiju. dim ir mainīgais, kādas būs ieejas datu dimensijas - signālu skaits, garums, kanāli.

Pirms apmācīšanas tiek nodefinēts $ModelCheckpoint$, kur parametrs *save_best_only* ir $False$, kas nozīmē, ka netiek saglabāti labākie svari tīkla apmācības laikā, $filepath$ mainīgais nosaka, kurus svarus tīkls izmantos. Pēc tam tiek izsaukts *model.fit_generator*, $model$ ir tīkla arhitektūra, kas tiek izveidota iepriekš, par tīkla arhitektūrām, kuras tiek izmantotas tiks apskatītas tālāk darbā un *fit_generator* metode nosaka to, ka tīkla apmācīšanā tiks izmantoti izveidotie ģeneratori - *tra_generator*, *val_generator*. Metodes mainīgais $epochs$ nosaka cik daudz epohu tiks apmācīts, $generator$ mainīgais nosaka kāds būs apmācības datu ģenerators, *validation_data* mainīgais nosaka kādu validācijas ģeneratori izmantos.

Tīkla apmācīšanai tikai ņemti 500 EEG signālu paraugi no katras miega fāzes ar izmēru 3000x2. Šie paraugi tika ņemti nejaušā secībā no septiņiem EDF failiem. Pēc tam šie paraugi tika sadalīti intervālos ar garumu 500 vai 1000 vai atstāti ar orģinālo garumu 3000. Tas ir atkarīgs kādus gadījumus vēlamies testēt. Ja ņemam ievades datu formu 1000x2, tad tīklam kopā būs

7500 apmācības datu. Kad datu forma ir 500x2 tad apmācības datu kopā būs 15000. Kad ieejas datu forma ir 3000x2 tad kopējie apmācības dati ir 2500.

Tīkla validācijai no atlikušajiem datiem tika ņemti 100 EEG signālu paraugi no katras miega fāzes ar izmēru 3000x2. Arī šie signāli tika sadalīti tādos pašos intervālos, kā norādīts ieejas datu izmērā. Tātad, ja ieejas dati ir 3000x2, tad kopējais validācijas datu skaits ir 500, 1000x2 gadījumā 1500 un 500x2 gadījumā 3000 datu.

Testēšanai tiek izmantots viss hipnogrammas fails SC4001EC-Hypnogram.edf un attiecīgais EDF fails SC4001E0-PSG.edf. Abi EEG signāli no EDF faila tiek sadalīti līdzīgos paraugu garumos kā apmācības gadījumā, kur ieejas datu formāts ir 1000x2, 3000x2 un 500x2. Koda piemēru var apskatīt pielikumā 5..

3.2. Apmācības modeļi

Darbā tiek apskatītas dažādas tīkla arhitektūras - LSTM 3.2., CNN 3.3., CNN ar 2 konvolūcijas slāņiem, kas apvienots ar LSTM 3.4. un CNN ar 8 slāņiem, kas apvienots ar LSTM 3.5.. Gadījumi kad tiek definēti CNN slāņi, izmantojot *Keras* bibliotēku, tiek izmantoti 1D konvolūcijas slāņi. CNN ir vienādas iezīmes un ievēro to pašu pieeju, neatkarīgi no tā, vai tas ir 1D, 2D vai 3D.

CNN-2 - LSTM un CNN-8 - LSTM ievades dati ir 3D tenzors ar formu: (partijas izmērs, parauga lielums, kanāli). Šajā gadījumā partijai būs vērtība 1, parauga lielums būs piemēram 500 un tiks izmantoti 2 kanālu dati. CNN un LSTM gadījumā ievades dati būs 2D (parauga lielums, kanāli), piemēram, 2 kanālu dati, kur parauga izmērs ir 500. Katrs šāds laika intervāls satur informāciju par pacientu EEG datiem konkrētā miega fāzē.

Izmantojot hibrīda CNN-LSTM modeli, katru paraugu sadala tālākās apakšgrupās. CNN modelis izgūs īpašības no katras apakšgrupas, un LSTM apkopos īpašības no šīm apakšgrupām. Šajā gadījumā viss CNN modelis tiek iesaiņots Laikā Sadalītos (angļ. TimeDistributed) slāņos tā, lai to varētu piemērot katrai parauga apakšgrupai. Katrs ievades datu laika intervāls tiek iesaiņots un pēc tam LSTM slānis apkopo rezultātus, pirms modelis atgriež prognozētos klašu datus.

CNN, CNN-2 - LSTM un CNN-8 - LSTM modeļiem konvolūcijas pirmā slāņa filtra augstums tīkliem ir 3. To sauc arī par kodola lielumu. Ja tiktu definēts tikai viens filtrs, neironu tīkls pirmajā slānī varētu apgūt vienu īpašību no datiem. Tas var nebūt pietiekams, tāpēc jānodefinē lielāks filtru skaits, piemēram, 16 filtri. Tas ļauj pirmajā tīkla slānī apmācīt 16 dažādas īpašības. Pirmā neironu tīkla slāņa izejas dati ir 498x16 neironu matrica. Katra izejas datu matricas kolonna satur viena filtra svarus. Ar nodefinēto kodola lielumu un, ņemot vērā ievades matricas garumu, katrs filtrs satur 498 svarus. Un kā aktivizēšanas funkcija tiek izmantota ReLU.

Lai samazinātu izejas datu sarežģītību un novērstu to, ka tīkls pārmācās. Pēc CNN slāņa tiks izmantots apvienošanas slānis. Šajā gadījumā tiks izmantoti maksimālās apvienošanas slāņi

ar filtriem, kuru dimensionalitāte ir 2x2. Pēc konvolūcijas slāņa ir iespējams arī pievienot partijas normalizēšanu, kas mazina dažādu slāņu ieeju ietekmi. Normalizējot neironu izvadi, aktivizēšanas funkcija saņems tikai ieejas datus, kas ir tuvu nullei. Tas ir arī risina izzūdošā gradienta problēmu.

Tīkliem izmantos gan Izslēgšanas (no angļ. Dropout) un Telpiskās izslēgšanas (no angļ. Spatial Droput) slāņus. Izslēgšanas ir regulēšanas metode, ko patentē Google, lai samazinātu neironu tīklu pārmācīšanos, novēršot to, ka rodas sarežģīta savstarpējā pielāgošanās apmācības datos. Tas ir ļoti efektīvs veids, kā ar neironu tīkliem veikt modeļa vidējošanu (no angļ. averaging). Izslēgšanas slānis piešķir nejauši izvēlētiem svariem 0 vērtību.

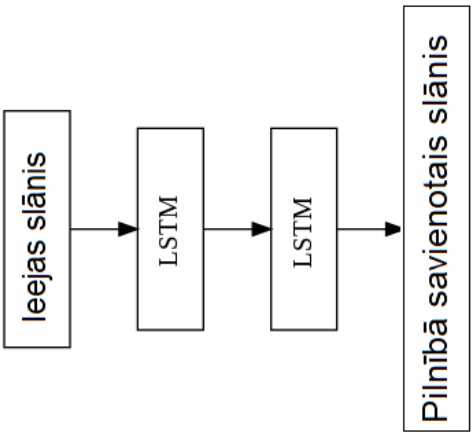
Telpiskās izslēgšanas slānis veic tādu pašu funkciju kā Izslēgšanas, tomēr tā vietā, lai samazinātu atsevišķu elementu svarus tiek atmesta visa 1D īpašību karte. Izmantojot šo darbību, tīkls kļūst mazāk jutīgs pret mazām datu izmaiņām. Ja īpašību kartēs blakus esošie kadri ir cieši saistīti, tad izslēgšana neregulēs aktivizāciju bet tikai izraisīs efektīvu mācīšanās ātruma samazināšanos.

Kā arī tīkla arhitektūras tiks izmantots izlīdzināšanas slānis. Izlīdzināšana nozīmē to ,ka tiks samazinātas dimensijas tā, lai paliktu tikai viena dimensija. Izlīdzināšanas slānis pārveido tenzoru tā lai izveidotos tāda formu, kas ir vienāda ar tenzora elementu skaitu jeb viendimensionāls masīvs ar visiem elementiem.

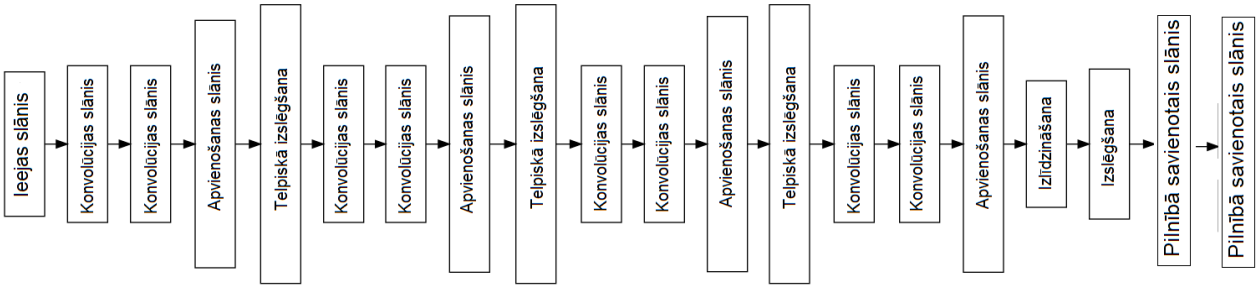
LSTM modelim pirmais LSTM slānis izmanto parametru ”return_sequences” ar vērtību ”True”, kas nosaka to vai izejas dati būs tādā pat formātā kā pēdējie ieejas dati vai arī tiks atgriezts 1D vektors.

Katra tīkla beigās ir pilnībā savienotais slānis, kas veic klasifikāciju pēc īpašībām, kuras tiek izgūtas no konvolūcijas slāņiem un samazinātas (angļ. val. downsampled) izmantojot apvienošanas slāņus. Pilnībā savienotajā slānī katrs slāņa neirons ir savienots ar katru iepriekšējā slāņa neironu.

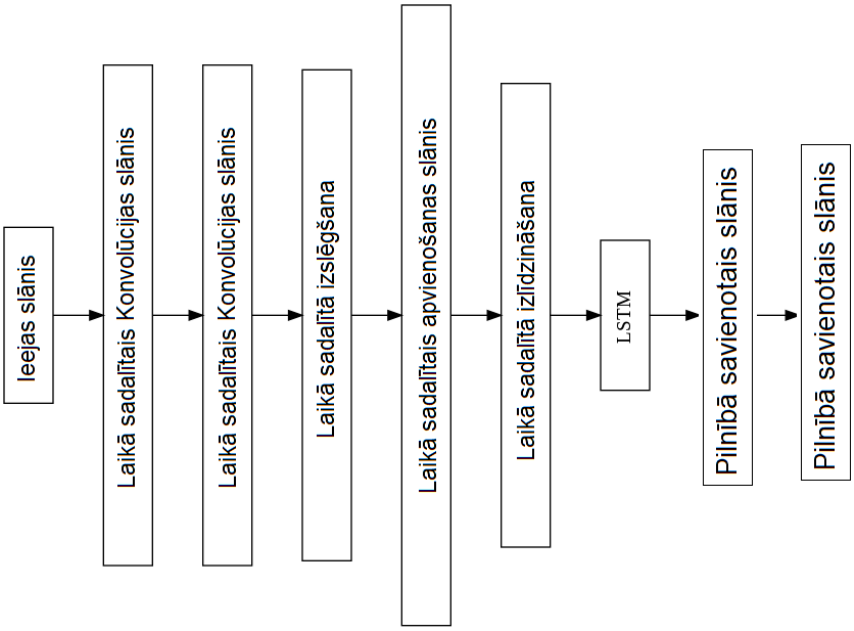
Pēdējais slānis samazina ieejas vektora garumu uz piecu elementu vektoru, jo izejas dati ir piecas klases, kuras vēlas paredzēt (”Wake”, ”REM”, ”N1”, ”N2”, ”N3”). Sigmoid tiek izmantots kā aktivizēšanas funkcija. Izejas dati tiek pārveidoti tā, lai sasummējot visu piecu klašu vērtības kopā gala rezultāts būt viens. Tādējādi izejas vērtība atspoguļo varbūtību katrai no piecām klasēm, kas šajā gadījumā ir miega fāzes.



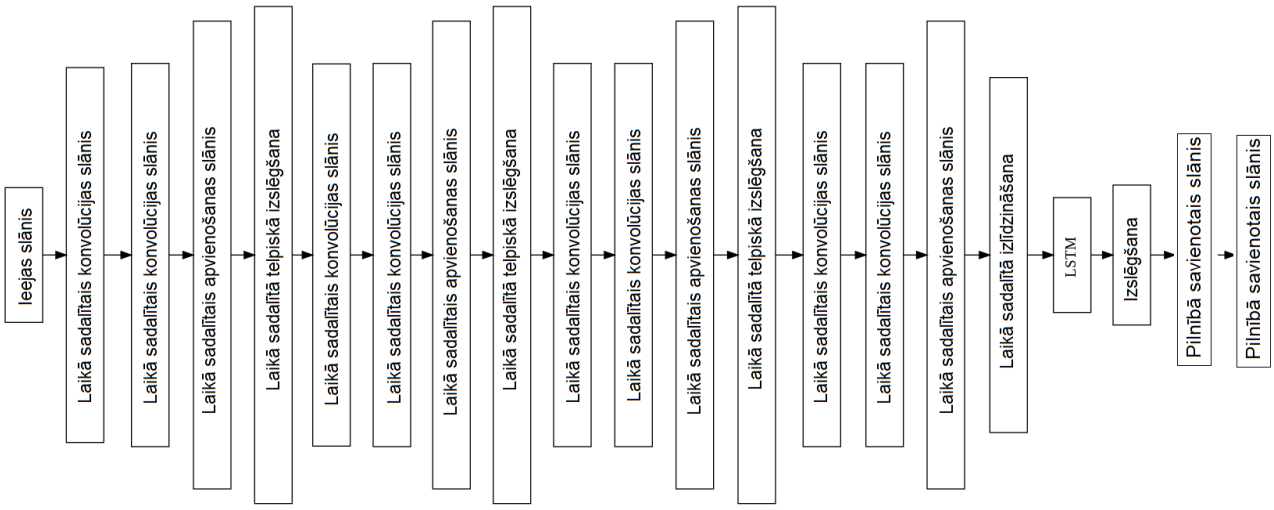
Att. 3.2. LSTM tīkla arhitektūra



Att. 3.3. CNN tīkla arhitektūra



Att. 3.4. CNN-2 - LSTM tīkla arhitektūra



Att. 3.5. CNN-8 - LSTM tīkla arhitektūra

3.3. Rezultātu novērtēšanas metrikas

Korelācijas koeficients ir skaitlisks mērījums kādam korelācijas veidam, kas nozīmē statistisko sakarību starp diviem mainīgajiem. Mainīgie lielumi var būt divas kolonas no novērojumu datu kopas, ko bieži sauc par paraugiem, vai divas daudzfaktoru gadījumu izlases mainīgās komponentes ar zināmu sadalījumu. Ja attiecība starp mainīgajiem nav lineāra, tad korelācijas koeficients nepietiekami atspoguļo attiecību starp mainīgajiem lielumiem. Pozitīva korelācija nozīmē to, ka vienam manīgajam palielinoties arī otrs palielinās. Negatīva korelācija nozīmē to, ka vienam mainīgajam palielinoties otrs samazinās. Ja nav korelācijas tad arī nav saistības starp abu mainīgo izmaiņām. Korelācijas spēks nozīmē to, ka, jo tuvāk absolūtā vērtība -1 vai 1, jo spocīgāka ir korelācija. Saikni starp korelācijas koeficienta matricu corr, kovariācijas matricu C un standartnovirzi σ datu kopai $\hat{y}$ un y nosaka vienādojums 3.1..

$$corr = \frac{C(Y, \hat{Y})}{\sigma_y \sigma_{\hat{y}}} \tag{3.1.}$$

Atkarībā no $corr$ vērtības tiek noteiks cik stipra ir korelācija starp divām datu kopām. Ja $0 = |corr|$ tad korelācijas neeksistē, ja $0 < |corr| < 0.2$ tad pastāv ļoti vāja korelācija. Ja $0.2 \leq |corr| < 0.4$ tad pastāv vāja korelācija. Ja $0.4 \leq |corr| < 0.6$ tad pastāv mēreni spēcīga korelācija. Ja $0.6 \leq |corr| \leq 0.8$ tad pastāv spēcīga korelācija Kad $0.8 \leq |corr| < 1.0$ tad ir ļoti spēcīga korelācija un ja $1.0 = |corr|$ tad ir ideāla korelācija.

Vidējā kvadrāta kļūda mēra prognozes vidējo kvadrāta kļūdu. Katram punktam tā aprēķina kvadrātisko starpību starp prognozētajiem un patiesajiem datiem un pēc tam iegūst vidējo vērtību starp abiem lielumiem. MSE aprēķina pēc formulas 2.8.. Jo augstāka šī vērtība,

jo sliktāks ir modelis. Šī kļūda nekad nepieņem negatīvu vērtību, jo pirms summēšanas individuālās prognozēšanas kļūdas tiek celtas kvadrātā. Perfektam modelim šī vērtība būs nulle.

Pārpratuma matrica ir tabula 3.6., ko bieži izmanto, lai aprakstītu klasifikācijas modeļa veiktspēju un tā satur 4 dažādas paredzamo un faktisko vērtību kombinācijas.

n=20	Prognozēts: Nē	Prognozēts: Jā
Patiess: Nē	TN = 5	FP = 10
Patiess: Jā	FN = 2	TP = 3

Att. 3.6. Pārpratuma matrica
[35]

- TP - Pareiza atbilsme ir rezultāts, kurā modelis pareizi prognozē pozitīvo klasi.
- FP - Kļūdaina atbilsme ir rezultāts, kurā modelis nepareizi prognozē pozitīvo klasi.
- TN - Pareiza neatbilsme ir rezultāts, kurā modelis pareizi prognozē negatīvo klasi.
- FN - Kļūdaina neatbilsme ir rezultāts, kurā modelis nepareizi prognozē negatīvo klasi.

Precizitāte ir viena metrika klasifikācijas modeļu novērtēšanai. Precizitāte ir tā daļa no prognozētajiem datiem, kuru modelim bija pareizi atpazinis. Precizitāte palīdz veikt datu analīzi, lai iegūtu vairāk ieskatu par modeļa veiktspēju. Formāli precizitātes formula ir 3.2., kur n ir kopējais paraugu skaits, $\hat{y}$ ir prognozētie dati un y ir patiesie dati.

$$acc = \frac{1}{n}\sum_{i=0}^{n-1} 1(\hat{y}_i = y_i) \tag{3.2.}$$

F_1 **novērtējums (no angļ.** F_1 **score)** nosaka tīkla precizitāti kad tiek veikta testēšana. Lai aprēķinātu šo novērtējumu izmanto formulu 3.3.. Lai aprēķinātu rezultātu, tiek ņemta vērā gan precizitāte p, gan testa atsauksme r. Precizitāte ir pareizo atbilsmju skaits, kas dalīts ar kopējo pareizo atbilsmju un kļūdaino atbilsmju skaitu. Atsauksme ir pareizo atbilsmju skaits, kas dalīts ar kopējo pareizo atbilsmju skaitu un kļūdaino neatbilsmju skaitu. Vērtība 1 apzīmē labāko un 0 apzīmē sliktāko gadījumu.

$$F_1 = 2 * \frac{p * r}{p + r} \tag{3.3.}$$

4. Rezultāti

Klasifikācija izmantojot 2 klases

Veicot detalizētāku datu kopas izpēti redzam, ka nomodas dati proporcionāli ir daudz vairāk, nekā miega dati. Autore ir ieinteresēta tieši miega fāžu noteikšanai. Šī iemesla dēļ vispirms tiks veikta nomodas un kopējās miega fāzes klasifikācija. Tas tika darīts, lai būtu iespējams izvērtēt vai modeļi vispār spēj atpazīt momentus, kad cilvēks ir nomodā vai arī cilvēks ir aizmidzis. Apmācībai dati tika ņemti no viena EDF faila SC4001E0-PSG.edf un tā pavadošās hipnogrammas SC4001EC-Hypnogram.edf.

Iegūtās pārpratuma matricas 4.1., 4.2. procentuāli apzīmē, cik pareizi un nepareizi tika klasificētas miega un momoda fāzes, kad datu izmērs ir 500x2 un 50 epohām. No Nomoda fāzes klasifikācijai tika paņemti 3258 paraugi un no miega fāzes tika ņemti 3309 paraugi.

CNN modeļa gadījumā var redzēt, ka tikai 6% (202) no visiem miega fāzes paraugiem tika klasificēti kā Nomoda fāzes paraugi. Līdzīgi arī ir CNN-2 - LSTM modelim Miega paraugu klasifikācijas gadījumā. CNN-8 - LSTM par 1% (3136) labāk izdevās klasificēt miega fāzes paraugus, bet LSTM modeļa kļūda ir 7% (228).

Apskatot Nomoda klasifikāciju, situācija ir mazliet sliktāka. CNN pareizi izdevās klasificēt 71% (2073) no visiem Nomoda fāzes paraugiem, CNN-2 - LSTM gadījumā 72% (2078) izdevās klasificēt korekti. Labākais rezultāts ir CNN-8 - LSTM, kam izdevās klasificēt 74% (2083) korekti, bet LSTM modelim tikai 63% (2068) izdevās klasificēt korekti, kas ir par apmēram 10 % sliktāk nekā pārējiem 3 modeļiem.

Tabula 4.1.

Pārpratuma matricas

CNN

	Nomoda	Miega
Nomoda	2073	1185
Miega	202	3107

CNN-2 - LSTM

	Nomoda	Miega
Nomoda	2078	1180
Miega	183	3126

CNN-8 - LSTM

	Nomoda	Miega
Nomoda	2083	1175
Miega	173	3136

Tabula 4.2.

LSTM

	Nomoda	Miega
Nomoda	2068	1180
Miega	228	3030

Pārpratuma matricas 4.3., 4.4. procentuāli apzīmē, cik pareizi un nepareizi tika klasificētas miega un Nomoda fāzes, kad datu izmērs ir 1000x2. No Nomoda fāzes klasifikācijai tika paņemti 2066 paraugi un no miega fāzes tika ņemti 1956 paraugi.

Salīdzinājumā ar iepriekš apskatītājām pārpratuma matricām 4.1., **??**, precizitāte pie datu izmēra 1000x2 uzlabojas. Šajā gadījumā CNN miega fāzea paraugus atpazīst ar 98% (1926) precizitāti. Līdzīgi kā iepriekš, CNN-2 - LSTM iegūst par vienu 1% (1940) labāku rezultātu, bet pats labākais rezultāts ir CNN-8 - LSTM, kas spēja 100%(1955) noteikt miega fāzes no Nomoda fāzi. LSTM spēja atpazīt korekti 88% (1727) no miega fāzes paraugiem.

Apskatot Nomoda fāzes paraugus, CNN spēja pareizi klasificēt 85% (1756), CNN-2 - LSTM 82% (1686). CNN-8 - LSTM, līdzīgi kā CNN, korekti klasificēja 85% (1754), bet, kā redzams LSTM gadījumā, modelis spēja pareizi noteikt 65% no visiem paraugiem.

Tabula 4.3.

Pārpratuma matricas

CNN

	Nomoda	Miega
Nomoda	1756	310
Miega	33	1926

CNN-2 - LSTM

	Nomoda	Miega
Nomoda	1686	380
Miega	19	1940

CNN-8 - LSTM

	Nomoda	Miega
Nomoda	1754	312
Miega	4	1955

Tabula 4.4.

LSTM

	Nomoda	Miega
Nomoda	1335	731
Miega	222	1727

Klasifikācija izmantojot 5 klases

Balstoties uz labajiem klasifikācijas rezultātiem, kas tika iegūti klasificējot miega un nomodas fāzes, tika nolemts apskatīt datu kopas klasifikāciju ar visām klasēm (Nomoda, N1, N2, N3 un REM). Kā piemērs, tika apskatīti [3], [4], [51].

Pārpratumu matricām 4.5., 4.6. ar dzelteno krāsu tiek atzīmēts, cik paraugu pareizi tiek atpazīti kā attiecīgā miega fāze. Virs matricas ir modeļa nosaukums un pati pārpratuma matricas attēlo, to kā tiek atpazīti EEG signāli no katras miega fāzes. Vertikāli tiek attēlotas patiesās miega fāzes un horizontāli tiek attēlotas prognozētās miega fāzes. Attiecīgās rindas un kolonas vērtības matricā atspoguļo to, pie kuras miega fāzes tiek klasificēti attiecīgi EEG signāli. Piemēram, ja apskata CNN matricu 4.5. Nomoda fāzi, var redzēt to, ka 215 Nomoda fāzes signāli tika

klasificēta, ka tie pieder Nomoda fāzei, 165 Nomoda fāzes signāli tika klasificēti, ka tie pieder N1 fāzei, 38 tika klasificēti kā N2, 12 kā N3 un 104 kā REM miega fāzei piederoši signāli.

Kopumā testēšanai no hipnogrammas SC4001EC-Hypnogram.edf un attiecīgais EDF fails SC4001E0-PSG.edf tika ņemti 543 Nomoda signāli, 348 N1 fāzes signāli, 1500 N2 fāzes signāli, 1320 N3 fāzes signāli, 740 REM fāzes signāli, kad ieejas datu izmērs ir 500x2 un epohas ir 50.

LSTM modeļa gadījumā, kad ieejas dati ir 500, var novērot to, ka no visiem paraugiem, kas pieder Nomoda fāzei 39% (210) tiek klasificēti kā piederoši N2 fāzei un 24% (130) kā piederoši REM fāzei, bet tikai 11% (62) tiek klasificēti kā Nomoda fāze.

No N1 fāzes paraugiem 31% (108) tiek uzskatīts par REM fāzes paraugiem, bet tikai 15% (51) tiek uzskatīti par N1, kā arī N2 fāzes paraugi 29% (432) gadījumu tiek klasificēti kā REM fāzes paraugi un 20% (293) gadījumu kā N1, bet tikai 26% (393) gadījumu kā N2. N3 fāzes paraugi tiek uzskatīti par piederošiem 26% (341) N1 un 25% (326) REM fāzei, bet pašai N3 fāzei 17% (226) no visiem N3 fāzes paraugiem. REM fāzes gadījumā, 31% (226) paraugi tiek uzskatīti par REM un 31% (226) N2 fāzes paraugiem.

CNN gadījumā, var novērot to, ka Nomoda fāze pareizi klasificē 40% (215) bet 30% (165) tiek klasificēti kā N1 fāzei un 19% (104) kā REM fāzei piederoši. N1 fāzes paraugi lielākoties tiek klasificēti, kā piederoši N2 fāzei ar 46% (160), bet tikai 18% (64) tiek klasificēti kā N1 fāze. N2 fāzes paraugus modelis 79% (1188) klasificē korekti un 15% (225) paraugu klasificē kā N3 fāzes un kā var redzēt N3 fāzes gadījumā paraugi 27% (353) tiek uzskatīti par N2 fāzei piederošiem, bet pašai N3 fāzei 64% (845). REM fāzes paraugi 44% (326) gadījumu tiek uzskatīti par N2 fāzes paraugiem, bet kā REM tikai 33% (247).

CNN-2 - LSTM gadījumā no Nomoda fāzes paraugiem 43% (234) tiek klasificēti kā REM fāzes paraugi, bet 42% (227) kā Nomoda. Šajā gadījumā no visiem N1 fāzes paraugiem tikai 1% (4) tika klasificēts korekti, pārējie tiek klasificēti kā citu klašu paraugi, lielākoties 48% (167) kā REM un 36% (125) kā N2. Līdzīgi kā CNN gadījumā N2 fāzes paraugi bieži tiek klasificēti kā N3 fāzes paraugi, šajā gadījumā 30% (443). 60% (893) tika klasificēti korekti kā N2 fāze. N3 fāzes paraugi tiek 9% (117) gadījumu pieskaitīti pie N2, bet 88% (1162) tiek klasificēti korekti. No REM fāzes paraugiem 33% (241) tiek uzskatīti par N2 fāzes signāliem un 63% (466) tiek uzskatīti korekti par REM.

CNN-8 - LSTM gadījumā no Nomoda fāzes paraugiem 41% (221) tiek klasificēti kā REM fāzes paraugi, bet 43% (235) kā Nomoda. Šajā gadījumā no visiem N1 fāzes paraugiem 29% (100) tika klasificēts korekti un 44% (154) kā REM fāzes paraugi. Līdzīgi kā CNN gadījumā N2 fāzes paraugi bieži tiek klasificēti kā N3 fāzes paraugi, šajā gadījumā 13% (190). 71% (1070) tika klasificēti korekti kā N2 fāze. N3 fāzes paraugi tiek 28% (376) gadījumu pieskaitīti pie N2 un 67% (896) tiek klasificēti korekti kā N3. No REM fāzes paraugiem 21% (155) tiek uzskatīti par N1 fāzei un 67% (495) tiek uzskatīti korekti par REM.

Tabula 4.5.

Pārpratuma matricas

LSTM

	Nomoda	N1	N2	N3	REM
Nomoda	62	58	210	83	130
N1	51	51	88	50	108
N2	179	293	393	203	432
N3	158	341	269	226	326
REM	93	83	226	122	226

CNN

	Nomoda	N1	N2	N3	REM
Nomoda	215	165	38	12	104
N1	19	64	160	21	84
N2	13	40	1188	225	34
N3	9	5	353	845	8
REM	17	155	326	5	247

Tabula 4.6.

CNN-2 - LSTM

	Nomoda	N1	N2	N3	REM
Nomoda	227	5	60	8	234
N1	32	4	125	20	167
N2	21	0	893	443	143
N3	35	1	117	1162	95
REM	33	3	241	7	466

CNN-8 - LSTM

	Nomoda	N1	N2	N3	REM
Nomoda	235	71	12	5	221
N1	25	100	61	8	154
N2	16	112	1070	190	112
N3	14	16	376	896	18
REM	24	155	75	1	495

Pārpatuma matricām 4.7.un 4.8. kopumā testēšanai no hipnogrammas SC4001EC-Hypnogram.edf un attiecīgais EDF fails SC4001E0-PSG.edf tika ņemti 808 Nomoda signāli, 174 N1 fāzes signāli, 750 N2 fāzes signāli, 660 N3 fāzes signāli, 375 REM fāzes signāli, kad ieejas datu izmērs ir 1000x2.

Kad datu izmērs ir 1000x2 tiek apskatīti trīs modeļi CNN, CNN-2 - LSTM un CNN-8 - LSTM. LSTM tīrais modelis netiek apskatīts pietiekami neprecīzo klasifikācijas datu dēļ, jo salīdzinājumā ar pārējiem modeļiem LSTM uzrādīja daudz zemāku precizitāti miega fāžu un nomodas atpazīšanā.

CNN gadījumā, var novērot to, ka Nomoda fāze pareizi klasificē 76% (613) bet 9% (74) tiek klasificēti kā N1 fāzei un 10% (79) kā REM. N1 fāzes paraugi tiek klasificēti, kā piederoši N2 fāzei ar 20% (35), REM 25% (44), bet tikai 33% (58) tiek klasificēti kā N1 fāze. N2 fāzes paraugus modelis 50%(376) klasificē korekti un 35% (262) paraugu klasificē kā N3 fāzes un kā var redzēt N3 fāzes gadījumā paraugi 7% (47) tiek uzskatīti par N2 fāzei piederošiem, bet pašai N3 fāzei 91% (599). REM fāzes paraugi 23% (87) gadījumu tiek uzskatīti par N1 fāzes paraugiem, bet kā REM 58% (217).

CNN-2 - LSTM gadījumā no Nomoda fāzes paraugiem 18% (147) tiek klasificēti kā N1 fāzes paraugi, bet 71% (576) kā Nomoda. Šajā gadījumā no visiem N1 fāzes paraugiem 56% (97) tika

klasificēts korekti, pārējie tiek klasificēti kā citu klašu paraugi, kur 28% (49) kā tika klasificēti kā N2. N2 fāzes paraugi 54% (406) tika klasificēti korekti kā N2 fāze, 24% (180) N1 un 15% (113) N3. N3 fāzes paraugi tiek 16% (104) gadījumu pieskaitīti pie N2, bet 80% (527) tiek klasificēti korekti. No REM fāzes paraugiem 68% (256) tiek uzskatīti par N1 fāzes signāliem un tikai 5% (19) tiek uzskatīti korekti par REM.

CNN-8 - LSTM gadījumā no Nomoda fāzes paraugiem 13% (107) tiek klasificēti kā REM fāzes paraugi, 14% (115) N1, bet 71% (574) kā Nomoda. Šajā gadījumā no visiem N1 fāzes paraugiem 25% (44) tika klasificēts korekti, 45% (78) kā REM fāzes paraugi un 18% (32) N2. Līdzīgi kā CNN gadījumā N2 fāzes paraugi bieži tiek klasificēti kā N3 fāzes paraugi, šajā gadījumā 15% (114). 64% (483) tika klasificēti korekti kā N2 fāze. N3 fāzes paraugi tiek 14% (93) gadījumu pieskaitīti pie N2 un 82% (544) tiek klasificēti korekti kā N3. No REM fāzes paraugiem 20% (76) tiek uzskatīti par N1 fāzei un 77% (288) tiek uzskatīti korekti par REM.

Tabula 4.7.

Pārpratuma matricas

CNN

	Nomoda	N1	N2	N3	REM
Nomoda	613	74	12	30	79
N1	23	58	35	14	44
N2	10	43	376	262	59
N3	9	1	47	599	4
REM	18	87	53	0	217

CNN-2 - LSTM

	Nomoda	N1	N2	N3	REM
Nomoda	576	147	54	11	20
N1	15	97	49	2	11
N2	5	180	406	113	46
N3	7	20	104	527	2
REM	10	256	88	2	19

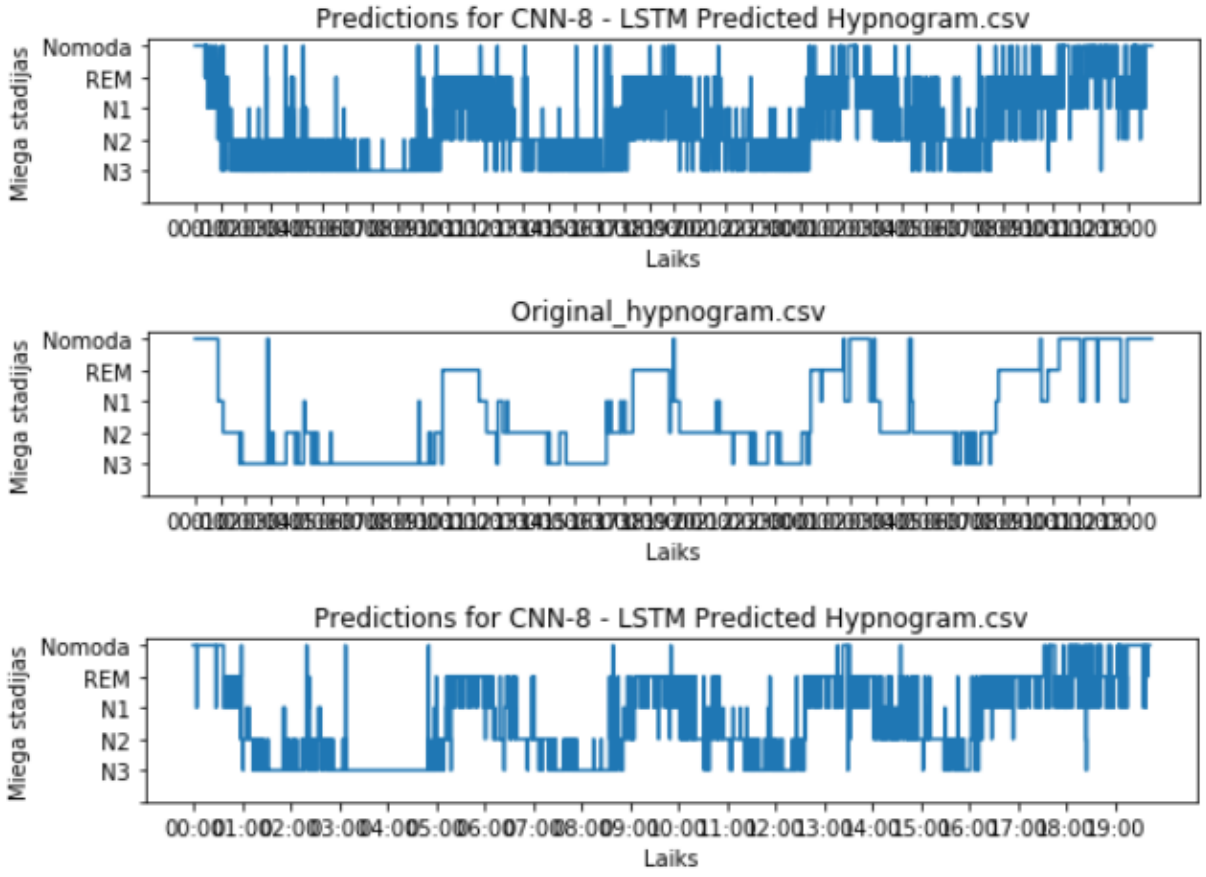
Tabula 4.8.

CNN-8 - LSTM

	Nomoda	N1	N2	N3	REM
Nomoda	574	115	4	8	107
N1	15	44	32	5	78
N2	4	71	483	114	78
N3	1	14	93	544	8
REM	6	76	5	0	288

Tika veikts grafiskais attēlojums labākā modeļa iegūtajiem rezultātiem. Pirmajā grafikā 4.1.tiek attēloti CNN-8 - LSTM modeļa prognozētie hipnogrammas dati, apmēram, tajā laika intervālā, kad cilvēks ir aizmidzis, pie datu izmēra 500x2, kad epohu skaits ir 50. Pa vidu ir orģinālais hipnogrammas faila grafiskais attēlojums arī miega intervālā. Trešais grafiks zem orģinālās hipnogrammas attēlo CNN-8 - LSTM modeļa prognozētos hipnogrammas datus miega

intervālā pie datu izmēra 1000x2, kad epohu skaits ir 50. Prognozēto datu grafiski atšķiras no orģinālās hipnogrammas, bet kopumā ir redzami iezīmes, kas liecina par to, ka šie grafiki attēlo konkrēto hipnogrammu. Kad datu izmērs ir 1000x2 tas ir vairāk izteikts, nekā tad, kad datu izmērs ir 500x2.



Att. 4.1. Hipnogrammas

Pārskats

Darba laikā tika:

- Veikta miega fāžu izpēte, lai uzzinātu ar ko miega periodā katra fāze atšķiras un kādi viļņu paterni tiek novēroti.
- Apskatīti jau eksistējošie miega EEG signālu klasifikācijas projekti.
- Izpētīti dziļās apmācības algoritmi. Vispirms CNN tad RNN un LSTM. Beigās tika apskatīts arī apvienotais CNN un LSTM modelis.
- Veikta pieejamo miega EEG datubāžu izpēte, lai atrastu derīgas datubāzes, kas satur gan EDF failus gan to pavadošās hipnogrammas.
- Izpētīti risinājumi, kā veikt datu izguvi no EDF failiem un to pavadošajām hipnogrammām.
- Apskatītas iespējamās izstrādes vides un modeļu izstrādes iespējas. Kā izstrādes vide tika izvēlēta Google Colab un kā programmēšanas valoda Python. Modeļu izstrādei tika izmantota Keras, kas ir atvērtā koda neironu tīkla bibliotēka, kas rakstīta Python.
- Veikta datu kopas sagatavošana, kur no EDF failiem tika ņemti divu kanālu EEG signāli un pēc tam sadalīti ap apkašsignāliem, balstoties uz hipnogrammas miega fāžu sadalījumu.
- Izveidots partijas ģenerators, kas tika izmantots tīkla apmācībā un validācijā.
- Apmācītas četri modeļi uz divām klasēm (Nomoda un Miega) un pēc tam testēti. Iegūtie testēšanas rezultāti tika apvienoti kopējās tabulās un salīdzināti.
- Apmācītas četri modeļi uz 5 klasēm (Nomoda, N1,N2,N3 REM) un pēc tam testēti. Iegūtie testēšanas rezultāti tika apvienoti kopējās tabulās un salīdzināti.

Secinājumi

- Dziļās apmācības metodes veiksmīgi spēj klasificēt Nomoda fāzi no miega fāzēm.
- Grūtības sagādā Nomoda fāzes atšķiršana no REM, kur REM varētu papildus noteikt, izmantojot elektromogrāfiju un acu artefaktu klātbūtnes noteikšanu, izmantojot papildus EEG sensorus.
- LSTM nespēj veiksmīgi klasificēt datus, pārpratumu matricā uzrādot vāju klasificēšanas veiktspēju.
- Nākošais solis būtu sagatavot datu kopu, kas saturētu tikai miega fāzes bez Nomoda fāzes, bet priekš tā dziļajām apmācības metodēm, ir nepieciešama daudz lielāka datu kopa.
- Uz izvēlēto datu kopu, CNN apvienojot ar LSTM, izdevās iegūt labākus rezultātus nekā izmantojot katru tīklu arhitektūru atsevišķi.
- 2 klašu gadījumā, kad datu izmērs ir 1000x2 modeļi spēj ļoti precīzi noteikt momentu, kad cilvēks ir aizmidzis.
- 2 klašu gadījumā Nomoda fāzi atšķirt no miega fāzes modeļiem sagādā lielāka grūtības, nekā atšķirt miega fāzes no Nomoda fāzes.
- N3 miega fāze visbiežāk tiek jaukta ar N2 fāzi, kā arī var secināt to, ka Nomoda fāze bieži tiek, uzskatīta par REM vai N1 staiju, jo tas ir moments, kas cilvēks vēl nav pilnībā aizmidzis.
- N3 un N2 ir fāzes, kuras modeļi spēj labāk atšķirt no citām fāzēm, jo šajās fāzēs cilvēks jau ir aizmidzis un viļņu paterni, kas rodas šo fāžu laikā ļoti atšķiras no REM, Nomoda un N1 fāzes.
- Darba laikā apskatītie projekti arī izmantoja to pašu datubāzi, kuru izvēlējās autore, jo ir piejamas, samēra maz miega EEG datu bāzes, kas iekļautu arī pavadošās hipnogrammas, kā arī datubāzes satur failus, kas nevienmēr ir EDF formātā.

Izmantotās literatūras un avotu saraksts

[1] M. M. Rahman, M. I. H. Bhuiyan, and A. R. Hassan, “Sleep stage classification using single-channel eog,” Computers in biology and medicine, vol. 102, pp. 211–220, 2018.

[2] J. Shi, X. Liu, Y. Li, Q. Zhang, Y. Li, and S. Ying, “Multi-channel eeg-based sleep stage classification with joint collaborative representation and multiple kernel learning,” Journal of neuroscience methods, vol. 254, pp. 94–101, 2015.

[3] S. J. Kern, “Automatic sleep stage classi

cation using convolutional neural networks with long short-term memory,” p. 48, August 18, 2017.

[4] A. Supratak, H. Dong, C. Wu, and Y. Guo, “Deepsleepnet: A model for automatic sleep stage scoring based on raw single-channel eeg,” IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 25, no. 11, pp. 1998–2008, 2017.

[5] J. Li, B. M. Sadler, and M. Viberg, “Sensor array and multichannel signal processing,” IEEE Signal Processing Magazine, vol. 28, no. 5, pp. 157–158, 2011.

[6] Sandwell and W. B. H. N. Trust, “What is a sleep eeg? information and advice for patients,” Information and advice for patients, pp. 1–4, October, 2014.

[7] U. Seneviratne, M. J. Cook, and W. J. D’Souza, “electroencephalography in the diagnosis of genetic generalized epilepsy syndromes,” Frontiers in neurology, vol. 8, p. 499, 2017.

[8] “Sleep arhitecture,” Normal Sleep, Feb 4, 2014.

[9] “Stages of sleep and sleep cycles,” How Sleep Works, DECEMBER 19, 2018.

[10] A. Y. A. Raman K. Malhotra, “Introduction to sleep stage scoring,” Sleep Stages and Scoring Technique, vol. 3, pp. 77 – 99.

[11] B. J. Swihart, B. Caffo, K. Bandeen-Roche, and N. M. Punjabi, “Characterizing sleep structure using the hypnogram,” Journal of Clinical Sleep Medicine, vol. 4, no. 04, pp. 349–355, 2008.

[12] R. Cabiddu, S. Cerutti, S. Werner, G. Viardot, and A. M. Bianchi, “Modulation of the sympatho-vagal balance during sleep: frequency domain study of heart rate variability and respiration,” Frontiers in physiology, vol. 3, p. 45, 2012.

[13] S. B. Robert J. Thomas, Sudhansu Chokroverty, “Hypnogram analysis,” vol. 6, p. 127.

[14] M. Pettit, “The importance of sleep cycles on productivity (+ tips to improve yours),” August 6, 2018.

[15] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” nature, vol. 521, no. 7553, p. 436, 2015.

[16] H. Chen, O. Engkvist, Y. Wang, M. Olivecrona, and T. Blaschke, “The rise of deep learning in drug discovery,” Drug discovery today, 2018.

[17] D. Ghosh, A. Olewnik, and K. Lewis, “Product “in-use” context identification using feature learning methods,” 08 2016, p. V01BT02A020.

[18] H. Lim, J. Park, and Y. Han, “Rare sound event detection using 1d convolutional recurrent neural networks,” in Proceedings of the Detection and Classification of Acoustic Scenes and Events 2017 Workshop, 2017, pp. 80–84.

[19] T. Lane, “1d and 3d convolutions explained with… ms excel!” Oct 17, 2018.

[20] L. Aragno, “Machine learning on javascript, part i: Neural networks,” DECEMBER 18, 2017.

[21] P.-H. Kuo and C.-J. Huang, “A green energy application in energy management systems by an artificial intelligence-based solar radiation forecasting model,” Energies, vol. 11, p. 819, 04 2018.

[22] C. Manenti, T. Pellegrini, and J. Pinquier, “Cnn-based phone segmentation experiments in a less-represented language,” 09 2016, pp. 3549–3553.

[23] S. Pascual, “Recurrent neural networks ii (d2l3 deep learning for speech and language upc 2017),” p. 13, Jan 25, 2017.

[24] Prabhu, “Understanding of convolutional neural network (cnn)—deep learning,” Mar 4, 2018.

[25] A. Karpathy, “Convolutional neural networks: Architectures, convolution / pooling layers,” CS231n Convolutional Neural Networks for Visual Recognition, Spring, 2018.

[26] D. Sood, “Backpropagation concept explained in 5 levels of difficulty,” Jul 22, 2018.

[27] N. Cui, “Applying gradient descent in convolutional neural networks,” in Journal of Physics: Conference Series, vol. 1004, no. 1. IOP Publishing, 2018, p. 012027.

[28] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in European conference on computer vision. Springer, 2014, pp. 818–833.

[29] S. D. Subir Varma, “Chapter 12 convolutional neural networks,” in Deep Learning, 2018-09-27.

[30] K. Frans, “Visualizing features from a convolutional neural network,” 2019.05.17.

[31] P. Grover, “5 regression loss functions all machine learners should know,” Jun 5, 2018.

[32] D. BRITZ, “Recurrent neural networks tutorial, part 1 – introduction to rnns,” WILDML Artificial Intelligence, Deep Learning, and NLP, SEPTEMBER 17, 2015.

[33] N. Donges, “Recurrent neural networks and lstm,” Feb 26, 2018.

[34] C. Olah, “Understanding lstm networks,” 2015.

[35] J. Guo, “Backpropagation through time,” Unpubl. ms., Harbin Institute of Technology, 2013.

[36] D. GUPTA, “Fundamentals of deep learning – introduction to recurrent neural networks,” December 7, 2017.

[37] A. Sherstinsky, “Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network,” CoRR, vol. abs/1808.03314, 2018.

[38] M. Nguyen, “Illustrated guide to lstm’s and gru’s: A step by step explanation,” Sep 24, 2018.

[39] H. Salehinejad, J. Baarbe, S. Sankar, J. Barfett, E. Colak, and S. Valaee, “Recent advances in recurrent neural networks,” arXiv preprint arXiv:1801.01078, 2017.

[40] E. Kang, “Long short-term memory (lstm): Concept,” Sep 2, 2017.

[41] H. Sak, A. Senior, and F. Beaufays, “Long short-term memory recurrent neural network architectures for large scale acoustic modeling,” in Fifteenth annual conference of the international speech communication association, 2014.

[42] K. J. Geras, A.-r. Mohamed, R. Caruana, G. Urban, S. Wang, O. Aslan, M. Philipose, M. Richardson, and C. Sutton, “Blending lstms into cnns,” arXiv preprint arXiv:1511.06433, 2015.

[43] J. Brownlee, “Cnn long short-term memory networks,” arXiv preprint arXiv:1511.08458, August 21, 2017.

[44] H. Gupta, “Deepmind’s relational reasoning networks—demystified,” Jul 3, 2017.

[45] F. Javier Ordóñez and D. Roggen, “Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition,” Sensors, vol. 16, p. 115, 01 2016.

[46] F. J. Ordóñez and D. Roggen, “Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition,” Sensors, vol. 16, no. 1, 2016.

[47] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” IEEE transactions on neural networks, vol. 5, no. 2, pp. 157–166, 1994.

[48] S. MONCADA, “The vanishing gradient problem,” RECURRENT NEURAL NETWORKS (RNN) – THE VANISHING GRADIENT PROBLEM, AUGUST 23, 2018.

[49] A. L. Goldberger, Amaral et al., “Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals,” Circulation, vol. 101, no. 23, pp. e215–e220, 2000. [Online]. Available: https://physionet.org/pn4/sleep-edfx/

[50] A. Amidi and S. Amidi, “A detailed example of how to use data generators with keras,” August, 2017.

[51] jaindeepali, “Eeg signal classification,” Feb 8, 2016.

Pielikumi

1. CNN-2 - LSTM iegūtie svari

Darknet ietvarā iegūtie svari atrodas `CNN2LSTM_weights` mapē.

2. CNN-8 - LSTM iegūtie svari

Visi mxnet-ssd ietvarā iegūtie svari atrodas `CNN8LSTM_weights` mapē.

3. CNN iegūtie svari

Visi mxnet-ssd ietvarā iegūtie svari atrodas `CNN_weights` mapē.

4. LSTM iegūtie svari

Visi mxnet-ssd ietvarā iegūtie svari atrodas `LSTM_weights` mapē.

5. Izmantotās apmācības un validācijas datu kopas

Izmantotās apmācības un validācijas datu kopas atrodas `sleep_edf` mapē.

6. Python piezīmju grāmatas fails SleepEEGClassification.ipynb

Izejas koda fails atrodas `SleepEEG` mapē.

Galvojums

Ar šo es, Linda Kalašņikova, galvoju, ka maģistra darbs ir izpildīts patstāvīgi, konsultējoties ar darba vadītāju. No svešiem pirmavotiem ņemtā informācija ir norādīta ar atsaucēm, dati un definējumi ir uzrādīti darbā. Šis darbs tādā vai citādā veidā nav nekad iesniegts nevienai citai pārbaudījumu komisijai.

Esmu informēts, ka mans maģistra darbs tiks ievietots un apstrādāts Vienotajā datorizētajā plaģiāta kontroles sistēmā plāģiāta kontroles nolūkos.

2019.gada _____ _____ _____
(paraksts)

Es, Linda Kalašņikova, atļauju Ventspils Augstskolai savu maģistra darbu bez atlīdzības ievietot un uzglabāt Latvijas Nacionālās bibliotēkas pārvaldītā datortīklā Academia (www.academia.lndb.lv), kurā tie ir pieejami gan bibliotēkas lietotājiem, gan globālajā tīmeklī tādā veidā, ka ikviens tiem var piekļūt individuāli izraudzītā laikā, individuāli izraudzītā vietā.

Piekrītu _____

Nepiekrītu _____

2019.gada _____ _____