Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

Kis Kornél István

BESZÉDPARAMÉTEREK ELEMZÉSE ÉS PREDIKCIÓJA MÉLY NEURÁLIS HÁLÓZATOKKAL

KONZULENSEK

Dr. Tóth Bálint Pál

Dr. Németh Géza

BUDAPEST, 2015

Tartalomjegyzék

Köszönetnyilvánítás **4**

Összefoglaló **5**

Abstract **6**

1 Bevezető **7**

1.1 A beszédképző rendszerek fejlődése 8

1.2 A beszéd elemi paraméterei 9

2 Neurális hálózatok tervezése és használata **11**

2.2 A neurális hálózat alapfogalmai és hálózati struktúrák 13

2.3 A munka során tárgyalt elrendezések 13

2.4 A neurális hálózatok tanítása 16

2.5 A tanítás hatékonyságának javítása 19

3 Kiindulási állapot – beszédparaméter adatbázisok **23**

4 Az elvégzett munka bemutatása **24**

4.1 Az alapfrekvencia kinyerése 25

4.2 Standardizálás és normalizálás 25

4.3 Interpoláció 26

4.4 A tanító adatbázis felépítése 27

4.5 A neurális hálózati modellek összeállítása 28

5 Eredmények **29**

5.1 Szigmoid (logisztikai) aktivációs függvényű architektúrák 29

5.2 Hiperbolikus tangens aktivációs függvényű architektúrák 32

5.3 Rectified Linear Unit (ReLU) aktivációs függvényű architektúrák 33

5.4 Összehasonlítás a referencia alapfrekvencia függvényekkel 35

5.5 Összehasonlítás a korábbi rendszer (HMM) által becsült alapfrekvencia menettel 37

6 A kutatómunka technikai részletei 39
7 Összefoglalás – jövőbeli tervek.....40
Forrás és irodalomjegyzék.....42
Az ábrák jegyzéke 44
Függelék 45
A felhasznált fonémakészlet 45
A neurális hálózat bemeneti és kimeneti paramétereinek listája.....45
Adatbázis fájl részlet.....47
SWIPE eredmény részlet.....47
Neurális hálózat bemeneti adat.....47

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani konzulenseimnek, Dr. Tóth Bálint Pálnak és Dr. Németh Gézának a TDK dolgozat megvalósításához nyújtott szakmai és technikai segítségükért. A dolgozat nem jöhetett volna létre a konzultációk során és a kritikus pillanatokban nyújtott odafigyelésük és segítségük nélkül.

Összefoglaló

A gépi beszédkeltés kutatásában az elmúlt évtizedben egyre nagyobb jelentőséggel bír a beszéd paramétereinek statisztika alapú modellezése. Az első jelentős eredményeket rejtett Markov modellen (Hidden Markov Model, HMM) alapuló rendszerekkel érték el. Ezen rendszereknek azonban a magas számításigény mellett más gyengeségeik is vannak: például a döntési fák nehezen modellezik a komplex környezetfüggőségeket. Jelen TDK dolgozat a rejtett Markov modellt kiváltva egy alternatív megközelítést valósít meg a beszéd gépi modellezésére a napjainkban nagy népszerűségnek örvendő mély neurális hálózatok (Deep Neural Network, DNN) segítségével.

Munkám során a szöveg fonetikus átirata és a beszédparaméterek közötti kapcsolatot DNN modellezi. Ehhez első lépésként a Budapesti Műszaki és Gazdaságtudományi Egyetem Távközlési és Médiainformatikai Tanszék Beszédtechnológia és Intelligens Interakciók Laboratóriumának korábbi rendszereiből és hang adatbázisaiból kiindulva a DNN tanításához szükséges tanító adatbázist létrehozó eljárás kidolgozására volt szükség. Ezek után a tanító adatbázist a nemzetközi irodalomra támaszkodva többféle neurális hálózat architektúrán tanítottam, különböző hiperparaméterekkel. A végső architektúra és hiperparaméter kombináció meghatározásához számos elméleti kérdésre is választ kellett adnom, az adatbázison több – néha a gépi beszédkeltés tématerületén kívül eső – módosítást is végre kellett hajtanom. Dolgozatomban az elméleti háttér áttekintése mellett bemutatom a gyakorlati megvalósítás során használt eszközöket is. Munkámban elsődlegesen a beszéd alapfrekvencia (f0) DNN-ekkel történő modellezését vizsgálom és valósítom meg kísérleti mintarendszerben, továbbá a spektrális paraméterek modellezésének a lehetőségét is elemzem. Eredményeimet összevetem korábbi gépi beszédkeltő rendszerekkel és kiértékelem a mély neurális hálózattal készített modellek pontosságát.

Jelen dolgozat rávilágít, hogy a DNN-el történő beszédparaméter modellezés – optimálishoz közeli hiperparaméterek használata esetén - érdemleges előrelépést nyújthat az eddigi megoldásokhoz képest.

Abstract

The application of statistical parametric models for Text-to-Speech (TTS) synthesis has been consistently increasing in the recent decade. The first statistical parametric method that achieved significant improvements over previous approaches was Hidden Markov Model (HMM) based speech synthesis. Unfortunately, these parametric models also suffer from several disadvantages. Apart from having a large computational cost to train, HMM-based speech synthesis systems have difficulties to model complex context dependencies. In this TDK paper, an alternative approach is used omitting Hidden Markov Models in favor of a relatively new, albeit popular modelling strategy: Deep Neural Networks (DNN).

In this approach, the relationship between the phonetic transcription of the input text and speech parameters is modelled by a DNN. The first step to build an efficient DNN model was to create a training database for the neural network. Based on previous solutions and speech corpora of the Speech Technology and Smart Interactions Laboratory, Department of Telecommunications and Media Informatics of the Budapest University of Technology and Economics , this data was used to train several state of the art neural network architectures. In the process of finding the best network architecture and hyperparameters for our purposes and transforming the training database to a representation which is best for training, several theoretical questions – sometimes from outside the field of TTS – were considered. In this TDK paper, both the theoretical and implementation elements are discussed in detail. The primary speech parameter under consideration is the fundamental frequency of voiced speech (f0), but the possibility of modelling other speech parameters are also investigated. Results are compared with previous attempts to approximate speech parameters and the error rate is objectively evaluated.

The goal of this TDK paper is to reveal that – given a near-optimal setup of hyperparameters – enhancements can be achieved over today's state-of-the-art HMM-TTS systems.

1 Bevezető

Jelen TDK dolgozat fő célja, hogy bemutasson egy mély neurális hálózatokon alapuló kísérleti rendszert, mely alkalmas az emberi beszéd paraméterfolyamjainak a becslésére. A kutatómunka végső célja – az eddigi eredményekre támaszkodva, kiegészítve a rendszert további paraméterfolyamokkal – a korábbi gépi szövegfelolvasó eljárásoknál természetesebb hangzású gépi beszéd előállítása.

A mesterséges beszédkeltésre egyre növekvő igény mutatkozik modern digitális világunkban. Az első hallásra logikusnak tűnő célcsoport, a vakok és gyengénlátók kiszolgálása mellett napjainkban az átlagember is sokféle módon hasznosíthatja a gépi beszédet generáló rendszereket (pl. IBM Watson, Google Glass, Apple Siri). A problémakör igen szerteágazó, hiszen a beszédnek érthetőnek, emberi hangzásúnak kell lennie, felhasználási helytől függően rugalmasnak, esetleg a többnyelvűség is elvárt. A tématerület kutatóit a jelenleg elérhető műszaki lehetőségek határai optimalizálási feladatok sora elé állítják. Kiemelkedő minőséget csak kötött témakör esetén lehet biztosítani (például időjárás-előrejelzés felolvasók), más területeken a rendkívül széles szókincs és nagyfokú rugalmasság biztosítása okoz kihívást a kielégítő beszédminőség mellett (jó példa ilyen alkalmazásra a vasúti pályaudvarokon használt bemondó-beszédgeneráló automata). Sokszor kompromisszumokat kell kötni a rendszer számítás- és helyigénye és a minőség között is. A későbbiekben többször említett rejtett Markov modellen [1] alapuló rendszereknek nagyobb helyigénye van a régebbi, diádos megoldásokhoz képest. A régebbi modellek egyik hibája, hogy a beszéd sokszor nem elég természetes hangzású. Ezen probléma orvoslása esetén a természetesebb gépi beszéd várhatóan tovább javítaná az ember-gép kommunikáció minőségét, mely egy újabb lépcsőfokot jelente a kognitív számítástechnika fejlődésében.

A jelen dolgozatban tárgyalt megoldás a beszédképző rendszerek legmodernebb csoportjába, a statisztikai parametrikus beszédszintetizáló rendszerek közé tartozik. A rendszer újszerűségét az adja, hogy a beszédparamétereket nem a rejtett Markov modell segítségével közelítjük, hanem a nemzetközi szinten egyre népszerűbb mély neurális hálózatokkal (Deep Neural Network, DNN) [2] modellezzük. Magyarországon a gépi beszédkeltés céljára tudomásom szerint az elsők között végzek kutatást mély neurális hálózatokkal. A DNN-ek és

azok működését biztosító kísérleti mintarendszer elemeit a dolgozat későbbi fejezeteiben részletes bemutatom. Az elvégzett munkám fő komponensei a következőek:

1. a kiindulási beszédparaméter-adatbázisból a mély neurális hálózatok számára értelmezhető tanító, validációs és teszt adatbázist létrehozó modul,
2. a neurális hálózat modelleket generáló és a tanítást vezérlő modul,
3. az eredmények kiértékelésére szolgáló modul.

Dolgozatom első fejezete megadja a munka elkezdéséhez szükséges beszédtechnológiai alapokat, valamint bemutatja a BME TMIT Beszédtechnológia és Intelligens Interakciók Laboratóriumának (a továbbiakban: TMIT Speechlab) birtokában lévő beszédparaméter- és hangadatbázis felépítését. A második fejezetben bemutatásra kerülnek a gépi tanulás legfontosabb elméleti jellegzetességei, az alapvető hálózati struktúrák, különös tekintettel a munka során használt architektúrákra és tanító algoritmusokra. A negyedik fejezet bemutatja a probléma megoldására felhasznált tanító adatbázis létrehozását, külön kiemelve az ezzel kapcsolatos – a beszédtechnológia témakörén kívül eső – problémát, ezen felül a vizsgált neurális hálózati architektúrákat az optimalizálandó hiperparamétereikkel együtt. Az ötödik fejezet beszámol az eddig elért eredményekről, összehasonlítva egymással az egyes architektúrák teljesítményét. Az utolsó fejezet a jövőbeli terveket és a jelenleg folyamatban lévő fejlesztéseket írja le.

1.1 A beszédképző rendszerek fejlődése

A beszédképző rendszerek – jelenlegi formájuk és teljesítményük eléréséig – rengeteg fejlődési lépcsőn mentek keresztül. A felhasználók részéről folyamatos igény jelentkezik az emberi beszéd analízisére, feldolgozására, és szükség esetén pedig a jó minőségű (emberire emlékeztető) gépi beszédkeltésre. Már jóval a digitális kor eljövetele előtt készültek olyan mechanikus, és később analóg elektronikus szerkezetek, amelyek képesek voltak gépi beszédkeltésre. A XX. század első felében a legsikeresebb rendszerek közé sorolható például Homer Dudley „VODER” nevű gépe (1939) [3], vagy az 40-es években Dr. Franklin S. Cooper által készített szerkezet [4], amely tulajdonképpen egy kézi paramétervezérlésű beszédkódoló

segítségével állított elő hangokat. Az első, több nyelven működő berendezés az 1980-as években készült el Dennis Klatt kutató jóvoltából [5]. Az első digitális beszédképzők rendkívül gépies hangzásúak voltak, a beszéd sokszor akadozott, és nagyon nehezen volt érthető. Az elmúlt évtizedekben a beszéd érthetősége folyamatosan javult, ez főképp a számítógépek adatfeldolgozási képességeinek és a fejlettebb matematikai modelleknek köszönhető. A tökéletesség azonban még messze van: a jelenlegi beszédképző rendszerek által szolgáltatott hangok az emberi fül számára még mindig könnyen megkülönböztethetők a valóságos emberi beszédtől általános tématerület esetén. A mai rendszerek többféle megközelítést is alkalmazhatnak, ezek például a diád- és triád alapú beszédszintézis, vagy a nagyméretű adatbázisokon alapuló korpuszos beszédszintézis. A beszédkeltő rendszerek eddigi legfrissebb, nagy technológiai ugrása a statisztikai beszédparamétereken alapuló rendszerek megjelenése volt mintegy 10 évvel ezelőtt. Az első sikereket a kutatók a rejtett Markov modelleken alapuló rendszerekkel érték el (mindmáig ezek a modellek működtették a legfejlettebb általános célú beszédkeltő alkalmazásokat) [1]. A rejtett Markov modell alapú rendszerek azonban számos nehézséggel is küzdenek: egy bizonyos pontosság elérése után csak igen sok új beszédminta bevitelével fokozható tovább a teljesítmény, a rejtett Markov modellben [2] használt döntési fák pedig nehezen modellezik a komplex környezetfüggőségeket. A neurális számítástechnika fejlődésével lehetőség nyílt egy alternatív megközelítésre, a mély neurális hálózat felhasználására [2].

1.2 A beszéd elemi paraméterei

A statisztikai parametrikus beszédszintézisben a beszédkódolókban használt megoldáshoz hasonlóan elemi paraméterekre bontják a beszédet. Ez általánosságban az alapfrekvenciát (f0) és más, az adott beszélőre jellemző spektrális paramétereket jelenti (például az un. mel-kepsztrális felbontás). Az alapfrekvencia a beszéd bonyolult, adott beszélőre jellemző nem folytonos függvénye. Az f0 a vizsgált beszélő egyedi hangmagassága mellett függ a beszélő nemétől, életkorától, sőt még olyan külső paraméterektől is, mint például a vizsgálat ideje [25]. (A reggel felvett hangfelvételeken jellemzően alacsonyabb f0 értékeket mérhetünk, mint a nap későbbi részén felvett hanganyagoknál). Ilyen bonyolult ok-okozati függés modellezése nehéz feladat

mind a hagyományosabb modellekkel, mind pedig a neurális hálózatok felhasználásával. A neurális hálózatok esetében a modellezni kívánt függvény szakadásai jelentenek elméleti szempontból leküzdendő kihívást. Az alapfrekvencia értékek nemcsak hogy nem alkotnak minden mondat esetén folytonos függvényt, hanem a zöngétlen hangkarakterek kiejtésénél nem is definiáltak, hiszen az ilyen hangkarakterek gerjesztése zajszerű, tehát nincs egy jól definiálható alapharmonikusa. Ahhoz, hogy eredményesebben modellezzük az ilyen függvényt, a tanító adatbázis elkészítése során módosításokat, például interpolációt érdemes alkalmaznunk az adatsorok egyes részein. A neurális hálózat a modellezni kívánt függvény vagy más összefüggés mintázatait numerikus értékekké alakítja, és saját, iteratív úton kialakított belső reprezentációjában tárolja. Ebből következően a tanító adatbázis minden elemének tartalmaznia kell egy definiált értéket, hiszen a „nem definiált érték” számokkal nem reprezentálható. Erről, és ehhez hasonló problémákról a tanító adatbázis elkészítésekor részletesebben lesz szó.

2 Neurális hálózatok tervezése és használata

Ez a fejezet tárgyalja a neurális hálózatok működésének megértéséhez szükséges nélkülözhetetlen alapismereteket, különös figyelmet szentelve a szerteágazó tudományterületen belül a TDK dolgozat szempontjából releváns részeknek. Szó lesz a neurális hálózatok alkalmazásának rövid történetéről, az alkalmazott technológiákról, az általam alkalmazott konkrét architektúrákról és a tanítás során alkalmazott optimalizálásokról.

2.1 A neurális hálózatok alapvető működése

A neurális számítástechnika (Neural Computation), mint tudományág mintegy 50 éves múltra tekint vissza [6]. A kezdetekkor, mint oly sok más műszaki rendszernél, a tudósok és mérnökök itt is a természetben létező biológiai folyamatok mintájára szerettek volna létrehozni tetszőleges függvény becslésére alkalmas modelleket. Ezen biológiai rendszereken alapuló hálózatok – neurális hálózatok – működése alapvetően eltér a hagyományos hálózatoktól. Közös, és egyben általános jellemzőjük, hogy a neurális rendszerek a megoldani kívánt feladatot nem valamilyen belső, tudatosan tervezett működés (pl.: állapotgép), szabályok segítségével valósítják meg, hanem a környezetből vett minták értelmezése révén. A neurális hálózat alapvető jellemzője tehát, hogy bemenetére kerülő mintázatok alapján valamilyen tanulási stratégia segítségével képes adaptálódni, a belső paramétereit a kitűzött feladat minél jobb megoldása érdekében "jó irányba" módosítani [7]. Fontos látni, hogy ez a radikálisan eltérő megközelítés teljesen másként tekint a megoldandó problémákra. A hagyományos, (nem gépi tanulással működő) számítástechnikai algoritmusok során az adott eljárás kidolgozásakor a tervező számára világos, hogy az input-output kapcsolat megteremtése során a rendszer milyen köztes állapotokon megy át. (Természetesen a rendszer működésében ettől még közrejátszhatnak heurisztikus elemek.) Egy gépi tanulással operáló algoritmus tervezése során nincs szükség az adott folyamat, vagy jelenség teljes mértékű ismeretére. Ez a megközelítés nagy segítség olyan, nehezen megfogható problémák esetén, ahol a feladat explicit leírása valamilyen okból kifolyólag nem lehetséges. Ezen okok lehetnek például a túlzott erőforrásigény, a modellezni kívánt rendszer kaotikus volta,

vagy az a tény, hogy sok esetben maga a megoldandó feladat is nehezen megfogalmazható a hagyományos paradigmák keretei között maradva. Tipikusan ilyen problémák az explicit képfelismerési feladatok (például arcok, alakok, tárgyak felismerése képekről vagy videóról). Könnyen belátható, hogy például egy alakfelismerés esetén nehezen definiálható a hagyományos módon a probléma, hiszen a bemeneti képek sokfélék lehetnek. Az ilyen tanuló rendszerek fejlesztési lépései is mások: a legfontosabb különbség, hogy az architektúra kiválasztása után a rendszer nem lesz automatikusan alkalmas az adott probléma megoldására. Ahhoz, hogy a hálózat ténylegesen el tudja végezni a kitűzött feladatot, először át kell esnie egy tanítási fázison. Ennek során a hálózatot kapcsolatba hozzuk példa bemenetekkel („unsupervised" tanulás), vagy bemenet/kimenet párokkal („supervised" tanulás). A hálózaton a példa adatsorokat végigfuttatva az a nagyszámú belső paramétereit módosítja, ekkor egy un. „belső reprezentációját" készítve el a feladatnak. Amennyiben a tanítás sikeres volt, ez a belső reprezentáció jól modellezi a feladatot, és képessé teszi a hálózatot arra, hogy valós bemenetekre is kis hibával adjon közelítést.

A neurális számítástechnika, mint önálló tudományág rendkívül aktív, folyamatosan fejlődő terület, ahol az utóbbi években is jelentős új eredmények születtek. Hinton 2006-ban megírt cikke [8] áttörést hozott a tématerületen, mert hatékony eljárást adott a sokdimenziós problémák neurális hálózatokkal való kezelésére. A fejlődést nagyban segíti, hogy elterjedtek a modern GPU-k (Graphics Processing Unit, magyarul grafikai processzor), melyek jelentős mértékben, akár nagyságrendekkel is felgyorsíthatják a számításokat. Így olyan problémák is kezelhetőek, melyekre korábban nem volt elegendő számítási kapacitás, vagy a számítás túl sokáig tartott volna. További javulást hozott a nagy mennyiségben elérhetővé vált tárkapacitás, és a nagyméretű, részletes adatbázisok megléte. A neurális hálózatokban nagy potenciál rejtőzik, különösen amiatt, mert jó eredményeket produkálnak, vagy legalábbis ígérnek tipikusan olyan területeken, ahol a hagyományos paradigmák alapján működő, korábbi rendszerek gyengén teljesítenek. A tudományterületen jelenleg azonban nagy kihívás igazán hatékony rendszert építeni, hiszen kevés az olyan kvantitatív eredmény, mely a gyakorlatban is jól használhatóan adna támpontot ilyen rendszerek ún. hiperparamétereinek a beállításához. A gyakorlatban ezért a legtöbb esetben az empirikus módszer, a már létező rendszerek vizsgálata képzi az újabb hálózatok kiindulását. Szerencsére már készült tanulmány a különböző rendszerek alkalmazhatóságát illetően, így hasonló problémákat vizsgálva támpontot kaphatunk a tervezés kiindulásánál. Matematikailag bizonyított azonban a neurális hálózat folytonos függvényekre

vonatkozó univerzális approximátor képessége. Ehhez a képességhez a hálózatnak legalább egy nemlineáris függvényt felhasználó réteggel kell rendelkeznie. A gyakorlati esetek többségében azonban a rejtett réteg(-ek) mellett a kimeneti réteget is nemlineáris neuronokkal építik fel. Ezek alapján a hálózat elméletben felhasználható tetszőleges osztályzási és regressziós problémákhoz is (például címkékkel ellátott kategóriákba sorolás, vagy folytonos függvények modellezése).

2.2 A neurális hálózat alapfogalmai és hálózati struktúrák

A neurális hálózatok – ahogy az az előző szakaszban is említtettük – a biológiai neuron, illetve az ebből felépülő neuronhálózat egy nagyon leegyszerűsített modellje alapján készültek. Egy ilyen hálózat jellemzője, hogy sok, azonos funkciójú elemi műveletvégző egység együttese alkotja. Az elemi műveletvégzők ebben az esetben egy lineáris összegzőből és egy nemlineáris függvényből állnak. Természetesen nem szükségszerűen egyféle elem alkothat hálózatot, de a legtöbb esetben az azonos feladatot ellátó elemi egységeket rétegekbe szervezik. Egy hálózat része szükségszerűen egy bemeneti „input” réteg, egy kimeneti „output” réteg, valamint tetszőleges számú opcionális rejtett „hidden” réteg. Mély neurális hálózatról jellemzően akkor beszélünk, ha hálózatunk több, a külvilággal direkt kapcsolatban nem álló rejtett réteget tartalmaz.

A neurális hálózatok egyik fő csoportja az ún. előrecsatolt hálózatok (feedforward network), amelyek gráfelméleti reprezentációja nem tartalmaz irányított kört, a másik csoport a rekurrens hálózat (recurrent network), melyben az irányított körök, visszacsatolások révén a hálózatnak időbeli memóriája is lehet. Ezen TDK dolgozatban kitűzött cél elérése során kizárólag előrecsatolt hálózatok kerültek felhasználásra.

2.3 A munka során tárgyalt elrendezések

Az előrecsatolt hálózatok egyik leggyakrabban használt alkategóriája az ún. Multi-Layer-Perceptron (MLP). Ez, mint az az elnevezésből is sejthető, a biológiai neuronhoz hasonló, azok bizonyos tulajdonságait megragadó modellnek tekinthető. MLP-t többféle elemi neuronból építhetünk, én most ebben a dolgozatban csak az egyik leggyakrabban használt elemi építőkővel,

az ún. memóriamentes Rosenblatt-perceptronnal [9] foglalkozom. A Rosenblatt-perceptron - matematikailag tekintve – tulajdonképpen két részből áll, és a következő

$$y = f(\boldsymbol{w}^T * \boldsymbol{x})$$

nemlineáris leképzést valósítja meg (ahol **w** és **x** vektorok). Az első rész egy lineáris súlyozott összegzés a bemeneti vektor (**x) és** az ún. súlyvektor (**w)** között. Az előző képletben $f(\dots)$ függvénnyel ábrázolt nemlineáris függvény az összegés után alkalmazva jelenik meg a kimeneten (*y*). A nemlineáris tag jelenléte nagyban kiterjeszti még ennek az egyszerű rendszernek a képességeit is. Az elemi neuron tehát a tetszőleges számú bemenetére érkező adatból egy nemlineáris transzformációt hajt végre az egy darab kimenetére. Ez a rendszer ebben a formában képes két (lineárisan szeparálható) tartomány szétválasztására (alapvető osztályozási feladat). Az ilyen elemi neuronokat hálózatba szervezve a képességek jelentősen javulhatnak.

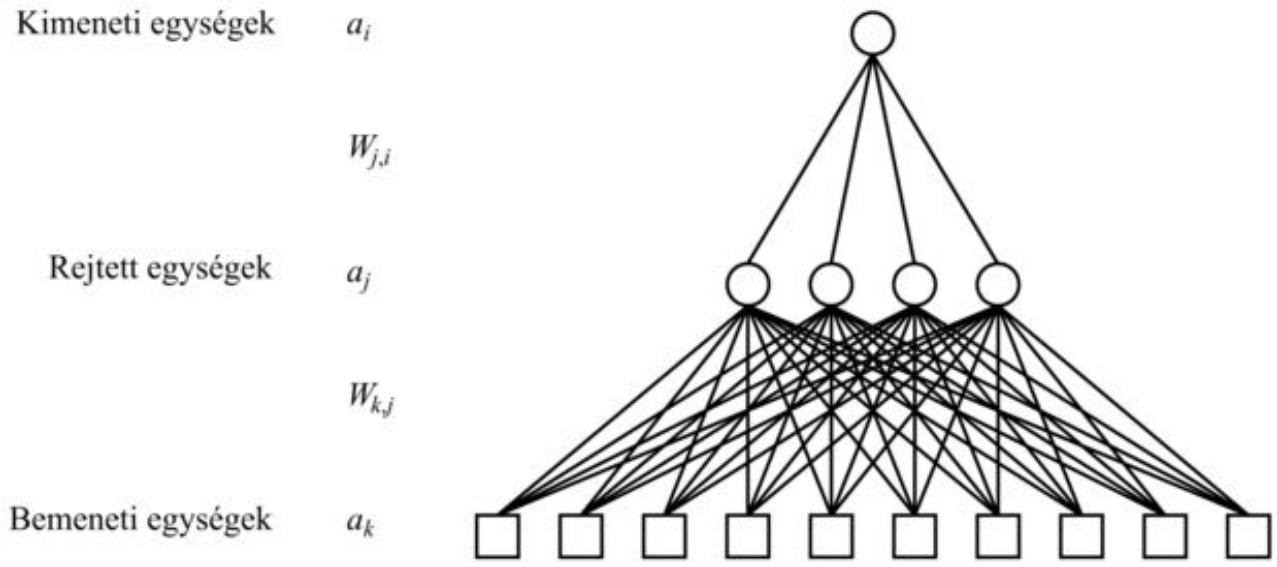
A nemlineáris függvény sokféle lehet, leggyakrabban valamilyen küszöbfüggvény jellegű nemlineáris függvényt használnak, például előjelfüggvényt ($sgn(x)$), hiperbolikus tangens függvényt ($tanh(x)$), vagy ún. szigmoid függvényt. Az elmúlt néhány évben egyre gyakoribb az ún. Rectified Linear Unit (ReLu) [10] alkalmazása is. Az aktivációs függvény mindegyikének megvan az előnye és hátránya, a ReLu például várhatóan szélesebb értékkészlete miatt alkalmasabb regressziós problémák esetén. Figyelni kell viszont arra, hogy a választott aktivációs függvény a tanító adatbázis adatait számszerűleg is befolyásolhatja. Erről bővebben a tanítás hatékonyságának javítása című alfejezetben lesz szó.

A feladathoz legmegfelelőbb aktivációs függvény megtalálása nem egyszerű feladat. A hálózati architektúra ilyen formán tekinthető szabad optimalizálandó paraméternek is, bár az kétségtelen, hogy nem minden aktivációs függvény rendelkezik olyan paraméterrel, amely a tanítás közben módosítható. Jelen alkalmazás esetén kizárólag különböző fix, előre felépített architektúrákon történt a tanítás. A jövőben, egy bonyolultabb modell összeállítása esetén az aktivációs függvény paramétereinek módosítását is esetlegesen alkalmazni lehet.

Az elemi neuronokat rétegekbe szervezve MLP-t kapunk. A rétegezésnél fontos, hogy olyan aktivációs függvényt válasszunk (ez a függvény általában rétegenként, esetleg az egész hálózatban egyforma), hogy az elemi neuron kimenete a súlyok differenciálható függvénye legyen. A tanításnál előszeretettel használt ún. hiba-visszaterjesztéses (back-propagation) algoritmus csak ekkor tud ugyanis működni.

Az MLP legegyszerűbb változata a három réteges felépítés, ahol az alapvető rétegek mindegyikéből egy van: bemeneti (input) réteg, rejtett (hidden) réteg, és kimeneti (output) réteg. Az általánosítások (Deep Neural Network- DNN) a rejtett rétegek számát növelik, ezzel a hálózat képességei még tovább nőhetnek, mivel az egyes rétegek a tanítóadatok különböző absztrakcióinak modellezésére szolgálhatnak. Bizonyítható, hogy az MLP univerzális approximátor, azaz képes tetszőleges folytonos nemlineáris függvény tetszőleges pontosságú modellezésére.

Az MLP architektúra építése során számos szabad paramétert módosíthatunk céljainknak megfelelően. Az első ilyen a hálózat mérete (ebbe a rejtett rétegek száma is beletartozik). Az egy rejtett rétegen belüli neuronok számának eldöntése alapvető fontosságú kérdés (a bemeneti és kimeneti neuronok számát meghatározza a probléma dimenziószáma, például egy 3 bemenetű, 2 kimenetű függvény közelítésekor 3 input neuron és 2 output neuron kell). Ha túl kevés rejtett rétegbeli neuront használunk, a hálózatunk nem fog megfelelően tanulni, túlzottan sok használata esetén pedig számítási kapacitást és időt pazarolunk el. Általánosságban kijelenthető, hogy bonyolultabb, illetve nagyobb mintaszámmal rendelkező problémák több rejtett rétegbeli neuront igényelnek, de ennél több konkrétum sajnos nem adható.



1. ábra – MLP hálózat felépítése. Forrás: [1]

A problémák többségénél érdemes bevezetni még egy extra, speciális egységnyi bemenetet (ún. bias bemenet) a hozzá tartozó súlyértékekkel, ez sok esetben segíti a tanulást (x irányú eltolást tesz lehetővé).

2.4 A neurális hálózatok tanítása

A tanítási eljárások többsége alapvetően két, már korábban is említett csoportba osztható. Ezek az ellenőrzött tanítás, és a nem ellenőrzött tanítás. Az MLP tanítása ellenőrzött módon történik, így a továbbiakban ezzel foglalkozom bővebben. Az ellenőrzött tanítás során a tervező rendelkezésére áll egy bemenet-kimenet párokból álló mintakészlet, amely a valós, modellezni kívánt rendszer alapján készült. A tanítás során, miután meghatároztuk a hálózat méretét, a súly értékek kezdeti értékeinek beállítása a következő lépés. Itt sok esetben véletlen inicializálást választunk, figyelve arra, hogy a nemlineáris függvény értékkészletével és az bemeneti és kimeneti adatokkal összemérhető nagyságú értékek legyenek a súlyok kezdeti értékei. Amikor a tanító mintáinkat végigfuttatjuk a hálózaton, azok kimeneteket generálnak (becsült kimenet). Ezeket a kimeneteket összehasonlítjuk az elvárt, eredeti rendszerről vett referencia kimenetekkel, és egy hibafüggvényt képzünk. A hibafüggvény megválasztása alapvetően befolyásolja a hálózat viselkedését. MLP esetén a hiba-visszaterjesztéses algoritmus sajátosságai miatt a négyzetes hibafüggvény gyakran használt, azonban az elmúlt években terjednek alternatív megoldások is. Sajnos azonban négyzetes hibafüggvény használata esetén is csak az biztos, hogy a hiba a tanítandó paraméterek folytonos, differenciálható függvénye (azaz a „hibafelület” nem kvadratikus). Ezen tulajdonság miatt megvan a veszélye annak, hogy a hibaminimalizálás során „leragadunk” egy lokális minimumhelyen, ahonnan a rendszer nem képes kimozdulni. Ennek elkerülésére többféle technika létezik, amelyek egy részéről a következő alfejezetben lesz szó.

A hiba-visszaterjesztéses algoritmus teljes bemutatása meghaladja a jelen dolgozat kereteit. A teljes leírás helyett inkább röviden összegzem a működését. Az algoritmus egy iteratív eljárás, amely minden egyes tanító minta rendszeren való átfutása után (online mód) vagy az összes tanító minta egyszeri lefuttatása után (batch update mód) a súlyfüggvény értékeinek módosításával csökkenti a hibát. A problémát az jelenti, hogy tudnunk kell, hogy a detektált kimenetben melyik súlyérték mekkora részben és milyen „irányban” vesz részt. (Erre ad választ az algoritmus). Az

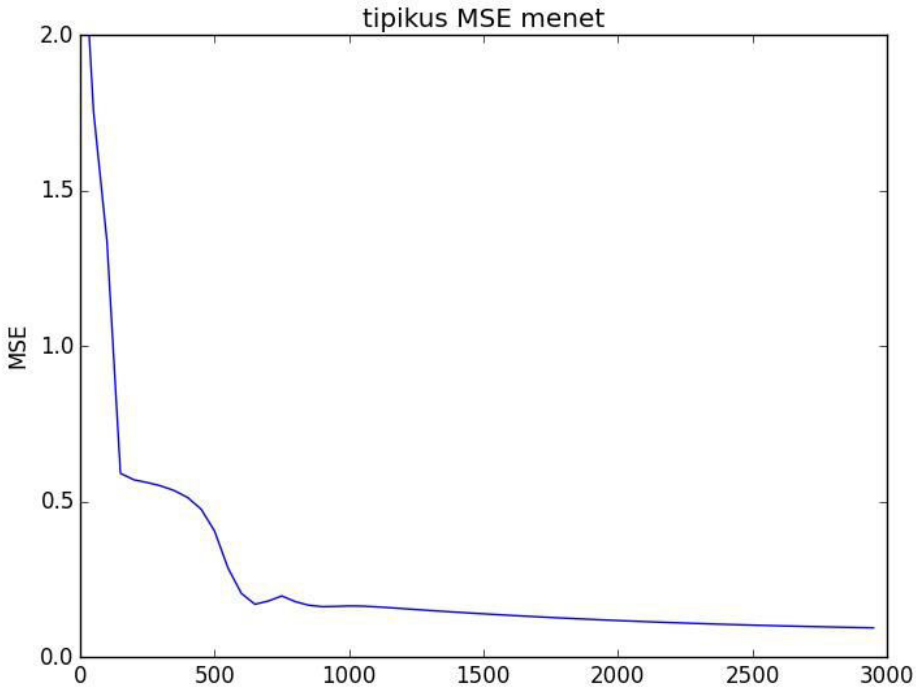
algoritmus minden iterációban végigfuttatja a tanító mintákat a rendszeren, feljegyzi a hibát, és annak megfelelően módosítja a súlyértékeket. Optimális esetben a hiba kezdetben nagyon gyorsan, majd később lassabban, de csökken az elméleti zéróig. Természetesen a gyakorlatban ez nem így valósul meg; semmi sem garantálja ugyanis, hogy a folyamat monoton, konvergens, így nem megfelelő beállítások esetén a hiba nem hogy nem csökken, de akár végtelenhez is tarthat.

Az algoritmus egy belső paramétere az ún. tanulási ráta (μ), amely egy szorzó tagként szerepel a súlymódosításkor. Vázlatosan ez a következőt jelenti:

$$W_{k+1} = W_k - \mu * \nabla f(Wk) + \eta * \Delta W_k \quad (1)$$

ahol ΔW_k a súlymódosítás értéke az adott iterációban, W_k a jelenlegi súlyérték, W_{k+1} pedig a következő iteráció súlyértéke, μ a tanulási ráta. A tanulási rátát pozitív (nulla és egy közötti valós) számként definiáltuk, ezért szükséges a negatív előjel a képletben. Az utolsó tag csak a momentum módszer (lásd később) alkalmazásakor számít, itt η a momentum módszer paramétere. A tanulási ráta egyszerűbb esetekben lehet konstans a teljes tanulás alatt, de számos adaptív tanulási ráta módosító stratégia is létezik. Nagyobb tanulási ráta gyorsabb tanulást eredményezhet, azonban ha túl nagyra választjuk, a konvergencia elvész. A hibafelület lokális minimumaiból való kimozdulást segítheti elsősorban a momentum módszer alkalmazása is, mely tehetetlenséget, és egy további szabad paramétert tesz a rendszerbe. A hiba-visszaterjesztésről részletesen [23]-ban lehet olvasni.

A gyakorlati alkalmazásoknál (különösen kevesebb neuron esetén) a hiba csökkenése kezdetben nagyon gyors, majd ez tempó lassul. Ezt szemléletesen mutatja az 2. ábra. A tanítás során általában meghatározunk egy hibafüggvény minimum értéket, amit a folyamat sikeres tanítás esetén elér. Itt érdemes leállítani az algoritmust, mert amennyiben sokáig még tovább folytatódik a tanítás, a „túltanulás” jelensége következik be, és a rendszer teljesítménye a valós adatokon az optimálishoz képest romlani fog.



2. ábra – tipikus hibafüggvényérték-menet a tanítás során. x tengely: iterációk száma, y tengely – középhiba (mean square error, MSE)

A tanítás során el kell döntenünk hogy milyen módon csoportosítjuk a rendelkezésünkre álló tanító mintákat. Egyszerűbb esetben két csoport lesz, a tanító és a validációs adatok. Bonyolultabb, gyakorlati alkalmazásoknál – jelen TDK dolgozat esetén is - egy ezektől teljesen külön kezelt, a tanítás végén végső ellenőrzésre használt ún. teszt adatbázist is használnak. A tanító halmazban lévőkön futtatjuk a hiba-visszaterjesztéses algoritmust, a maradék adatokon ellenőrizzük a sikerességet. Fontos látni, hogy ez az adat szelektáló kérdés is döntő lehet: ha túl kevés a tanító adat, nem fog megfelelően tanulni a rendszer, ha pedig túl kevés az ellenőrző adat, nem fogjuk látni a folyamat sikerességét. Ez a konfliktus különösen olyan alkalmazásoknál éles, ahol nehézkes vagy drága a mintákhoz való hozzájutás. Gyakorlatban működő szabályként általában az adatok mintegy 65-80%-át szokták a tanító halmazba, a maradékot pedig a validációs és teszt halmazba tenni. Amennyiben végső, tanítástól független vizsgálatokhoz teszt adatbázist használunk, ezt a tanító adatoktól elkülönítetten kell kezelni, így a hálózat paramétereinek végső beállítása után objektíven tudunk tesztelni.

Más, itt bővebben nem tárgyalt felépítésű hálózatok esetén a nem ellenőrzött tanulásra is lehetőségünk van. Ilyen tanulás során, mivel a bemeneti adatokhoz nem tartoznak referencia

kimenetek, hibafüggvény képzése nem lehetséges. A hálózatnak az a feladata, hogy az adatokban a (rejtett) mintázatokat megtalálja. Ezen mintázatok segítségével az adatokat csoportokba rendezheti (ez osztályozási problémáknál fordul elő). Mivel a nem ellenőrzött tanulás jelen beszédtechnológiai alkalmazásban nem került felhasználásra, a továbbiakban nem foglalkozom vele.

2.5 A tanítás hatékonyságának javítása

A neurális hálózatok tanításának egyik legfőbb nehézségét a matematikai háttér hiánya okozza [11]. Nem állnak rendelkezésre olyan tételek, amelyek a gyakorlatban is jól használhatóan adnának felső korlátot egy probléma esetén szükséges hálózatméretre, vagy tanítási paraméterre. Ebből kifolyólag a tanítás során nagy szerepe van a tervezők korábbi tapasztalatainak, illetve a hasonló problémák esetén már működő eljárások vizsgálatának. A témakörrel hosszú ideje foglalkozó szakértők sok, a gyakorlatban működő eljárást dolgoztak ki, itt ezek közül ismertetek néhányat. Ezen eljárások többségének működése a témakörben mélyebb ismeretekkel rendelkezők számára intuitív módon belátható, annak ellenére, hogy ezek nem formális matematikai precízséggel megalkotott módszerek.

Mint ahogy azt korábban már említettem, a gradiens módszer egy minimumhely kereső eljárás. Mint minden ilyen, ez is szenved a lokális minimum elérésének problémájától. A legegyszerűbb problémáktól eltekintve a sokdimenziós hibafelület változatos alakú, így ahhoz, hogy ténylegesen megállapodjunk a globális minimumhoz közel, időnként a hibafelületen a nagyobb hiba irányába („felfelé”) is el kell mozdulnunk. A gradiens módszer alap esetben erre nem képes, hanem leragadhat egy nem szükségszerűen optimális lokális minimumban. Az egyik módszer a leragadás kiküszöbölésére a momentum módszer alkalmazása. Ha a gradiens gyakran vált előjelet, a momentum módszer képes kisimítani az utat a vélelmezett minimum felé. A momentum módszer a tanulási rátához hasonló, 0 és 1 között állítható paraméterrel rendelkezik. Nagy momentum alkalmazása esetén a tapasztalatok szerint érdemes a tanulási rátát kisebbre venni, különben gyorsan túlhaladhatunk a minimumon.

A momentum módszerhez képest további optimalizálást tesz lehetővé Nesterov megoldása [12]. A Nesterov-momentumnak is nevezett módszer módosítja a (1)-es képletet. A Nesterov-momemtum alkalmazása esetén a súlyfrissítési képlet a következő:

$$W_{k+1} = W_k - \mu * \nabla \mathrm{f}(Wk + \eta * \Delta W_k) + \eta * \Delta W_k \quad (2)$$

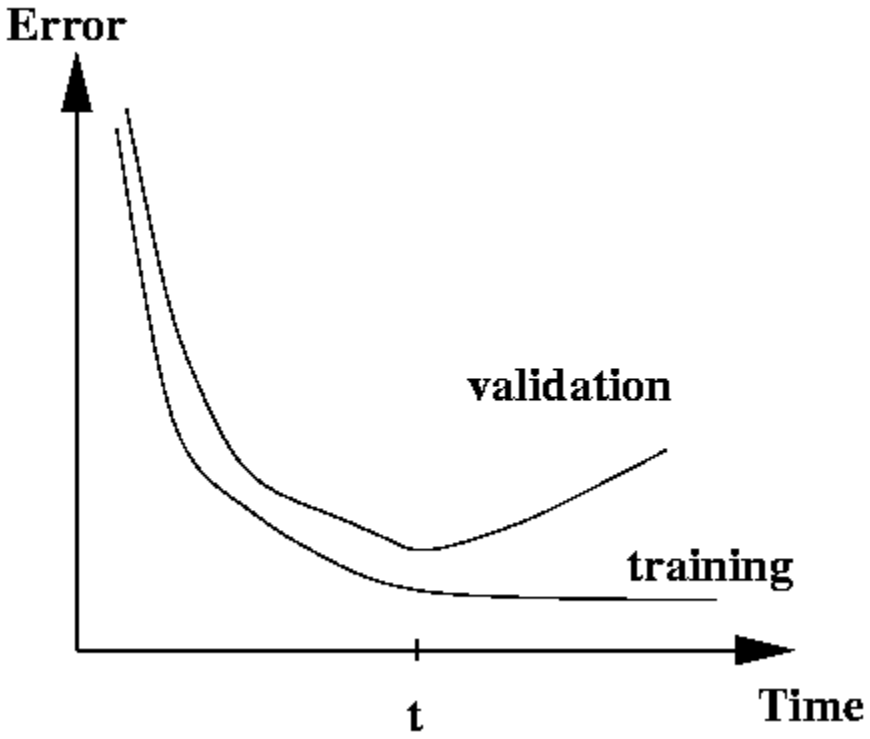
A módosítás értelmét szemléletesen úgy lehetne kifejezni, hogy amennyiben nagy (egyhez közeli) momentum paraméter használata esetén a súlyfrissítési lépésekben ez a tag túlzottan domináns lesz, azzal végső soron eltávolodunk eredeti céljainktól. A Nesterov-momentum alkalmazása ezen hatás ellen dolgozik, ami várhatóan javítja a végső eredményt.

Természetesen nem a momentum módszer az egyetlen optimalizálási módszer, bár kétségtelenül ez az egyik leginkább elterjedt. A neurális tanuló rendszerek tanításának talán legfőbb kérdése a túltanulás, illetve ennek elkerülése. A túltanulás mint probléma azt jelenti, hogy a rendszer nem a tanító adatokból próbál általános belső reprezentációt készíteni, hanem magukat a tanító adatokat tekinti a rendszer egészének. Amennyiben nem sikerül a rendszert általánosításra rávenni, az nem fog működni valós teszt adatok esetén. A tapasztalatok azt mutatják, hogy a rendszer hajlamosságát a túltanulásra a tanítandó paraméterek száma és a tanító adatok arányának hányadosa írja le jól. Ez azt jelenti, hogyha kevés adattal rendelkezünk a feladat bonyolultságához képest, vagy túl bonyolult architektúrát használunk egy egyszerűbb problémára, a rendszerünk erősebben túl fog tanulni.

A túltanulás elkerülésének egyik leghatékonyabb módszere (a hálózati architektúra adatokhoz való szabása mellett) a tanító adatbázis növelése. Azonban ennek vég nélküli fokozása sem lehet megoldás, hiszen nagyobb adatbázis nagyobb tároló és főleg számítási kapacitást kíván. Másik probléma, hogy sok valós probléma esetén költséges vagy egyenesen lehetetlen a további adatok beszerzése. Ilyen esetben más módszerekhez kell folyamodnunk a túltanulás elkerülése, vagy legalábbis késleltetése érdekében.

A túltanulás detektálása ellenőrzött tanítás során a validációs és a tanító adatbázis által számolt hibafüggvény összehasonlításával történik. Amennyiben a hiba átlaga a validációs és a tanító adatbázison közel egyező mértékben csökken, túltanulás nem lépett fel, és a tanítás biztonságosan folytatható. Ha a tanítást folytatva azt tapasztaljuk, hogy a validációs adatbázison a hiba újra növekedni kezd, miközben a tanító adatokon a hiba tovább csökken, bekövetkezett a túltanulás jelensége. Ennek elkerülésére egy logikusnak tűnő megoldás az ún. *early stopping* [13]

technika, amely megkísérli megtalálni azt a pontot, ahol a validációs halmazon a hiba még nem kezdett el újranövekedni. A következő sematikus ábrán látható az *early stopping* technika lényege.



3. ábra- Early Stopping.(validation – validációs adatbázis, training – tanító adatbázis)
Forrás: [2]

Az ábrán a *t* időpillanat a legjobb a tanítás leállítására. A valóságban a megfelelő időpont megtalálása nem könnyű, de az is igaz, hogy már az optimálishoz közeli megállással is sokat nyerhetünk.

Az *early stopping* technika helyett, vagy azzal párhuzamosan alkalmazott technológia az ún. *dropout* módszer [14]. Ez szintén elsődlegesen a túltanulás késleltetésére alkalmazható, de egészen más az elve, mint az *early stoppingnak*. Dropout alkalmazásakor tanítás során avatkozik bele a tervező a hálózatba, az egyes iterációk során a rendszer véletlenszerűen kihagy elemi neuronokat a rejtett rétegekből. Ez a technika megakadályozza a neuronok koadaptációját. A kihagyott neuronok arányát egy százalékos értékben határozzák meg. Ez általában 30 és 50% között van. A módszer során fontos, hogy a neuronokat tényleg véletlenszerűen kapcsoljuk ki-be.

A tapasztalatok szerint a *dropout* technika hatékonyan képes csökkenteni a káros túltanulást és ezáltal javítja a hálózat teljesítményét. Az alkalmazott technikákról az eredményeket tárgyaló fejezetben lesz szó.

A tanítás során el kell még döntenie a tervezőnek, hogy egy tanítási iteráció során (ezt *epoch*nak is neveik) a tanító adatbázis mekkora részét dolgozza fel a súlymódosítás céljára. A két szélsőséges eset az ún. *full-batch learning* és az *online learning*. *Full-batch* esetén a tényleges gradiens kiszámítása történik minden iterációban, az összes adat alapján. Ez sokszor szükségtelenül lassú, hiszen nehezebb párhuzamosítani a folyamatot. A másik szélsőség, az *online learning* használata esetén egy véletlenszerűen választott tanító adat alapján állítják a súlyokat. A két szélsőséges módszert viszonylag ritkán használják, gyakorlatban a kettő között található ún. *mini-batch learning* terjedt el annak nagyobb sebessége miatt. Itt a tanító adatbázis tervező által meghatározott nagyságú részeit választják ki véletlenszerűen és ezek alapján számolják a gradiens egy közelített értékét. Bár ez az érték nyilvánvalóan különbözik a valós gradienstől valamilyen mértékben, ez a bizonytalanság (zaj) a lokális minimumok elkerülése szempontjából még előnyös is lehet. Az optimális gradiens méret szintén az adott probléma függvénye. Jelen TDK dolgozat során kizárólag *mini-batch learning*-et alkalmaztam.

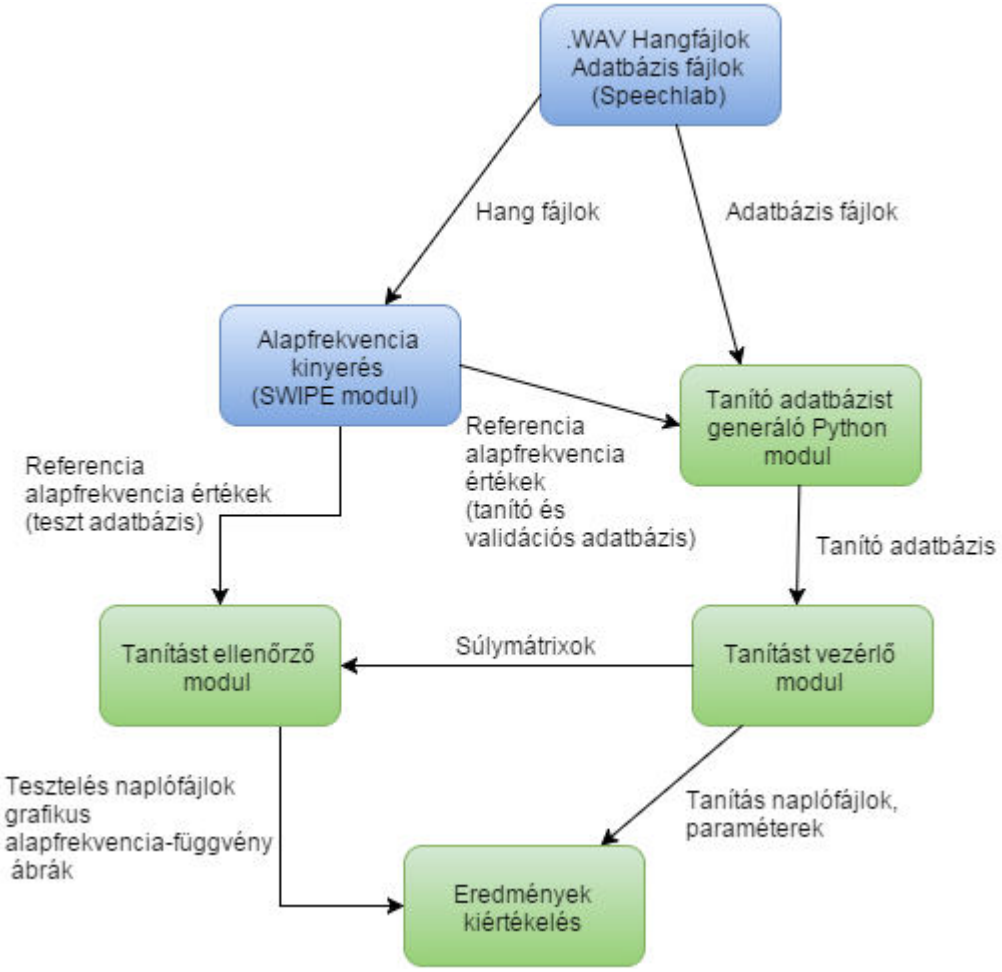
Fontos még kitérni az tanulási ráta stratégiákra is. A tanulási ráta a gradiens módszer legelemibb paramétere, helytelen beállítása esetén a hálózat nem vagy csak nagyon gyengén fog tanulni, matematikailag még a hiba növekedése is elképzelhető. A legegyszerűbb technika, a teljes tanítás alatt fixen tartott tanulási ráta paraméter korlátozottan alkalmas a gyakorlati problémák esetén. A tapasztalatok és az elméleti megfontolások alapján látható, hogy a tanítások többségénél kezdetben nagyobb, majd fokozatosan csökkenő tanulási ráta alkalmazása célszerű (ezt nevezik *learning decay* technikának [15]). Ennél is jobb megoldás, ha minden súlyértékre különböző tanulási rátát használunk, melyet akár egy negatív visszacsatolással ellátott körbe is betehetünk. Ekkor egy olyan szabályzót alkalmazunk a körben, mely oszcilláló, ugráló hiba esetén csökkenti a tanulási rátát, ha viszont nem javul a hibaérték már több iteráció után sem, akkor óvatosan növeli a paramétert. Egy ilyen rendszer természetesen jóval bonyolultabb, ezért a jelen kutatás keretében kizárólag *learning decay* került alkalmazásra.

3 Kiindulási állapot – beszédparaméter adatbázisok

A munka kezdetén a BME-TMIT SpeechLab a kutatómunka céljaira rendelkezésemre bocsátotta beszédhang adatbázisának egy részét. Az adatbázis emberi beszélőktől (stúdiókörülmények között) felvett rövid kijelentő mondatokat tartalmaz. Minden mondat egy külön .wav formátumú fájlban található. Minden hangfájlhoz egy adatbázis fájl is tartozik, mely a mondatban elhangzó fonémákat tárolja a hozzá tartozó időbélyegekkel. A teljes fonémalista a Függelékben található. Az adatbázis fájlokban ezen kívül számos más kontextus paraméter is található, mely kiválóan alkalmazható statisztikai parametrikus beszédszintetizáló rendszerek tervezésekor. Ezek a paraméterek közül néhány a fonémát a mondaton belül tágabb kontextusba helyezi (jelölve vannak a fonémát egy és kettő távolsággal megelőző és az azt követő fonémák is), valamint foglalkozik a beszéd nagyobb egységeinek a leírásával is (szótag, szó, tagmondat és mondat szinten) [1]. A kutatómunka során felhasználásra került, a fonémákat leíró paraméterek teljes listája a Függelékben megtalálható.

4 Az elvégzett munka bemutatása

Ez a fejezet részletesen ismerteti a meglévő beszédparaméter-adatbázis neurális hálózat kompatibilis tanító adatbázissá való konvertálásának lépéseit, és a tanítás folyamatát. A neurális hálózati adatbázis készítése nem csupán az egyes formátumok közötti gépies másolásból áll, hiszen figyelembe kell venni az alkalmazott neurális hálózat sajátosságait a minél sikeresebb tanítás érdekében. A következő alfejezetben bemutatásra kerülő részlépések egy összefoglalása a 4. ábrán található. Az ábrán zölddel jelöltem azokat az elemeket, melyek a TDK munkám részeként készültek.



4. ábra – a kísérleti mintarendszer folyamatábrája

4.1 Az alapfrekvencia kinyerése

Mivel az MLP hálózat ellenőrzött tanulást kíván, mindenképp szükség van egy, a hangfájlokból előállított referencia alapfrekvencia adatbázisra, amelyhez képest a neurális hálózat hibafüggvényét képezhetjük. Az alapfrekvencia hangfelvételekből való kinyerésére többféle megoldás is létezik. Jelen kutatás során az ingyenesen elérhető SWIPE (A Sawtooth Waveform Inspired Pitch Estimator) [16] szoftvert használtam, ami C nyelven íródott. A programnak a hangfájl mellé bemeneti paraméternek meg kell adni a mintavételi időt és a minimum-maximum alapfrekvencia értékeket, amelyek az adott hangfájl beszélőjére jellemzők. A program ezekből az adatokból generál egy szöveges állományt, amely a választott mintavételi időnként számol egy alapfrekvencia értéket. A generálás során az használt adatbázishoz tartozó frekvencia-szélsőértékek: 35Hz és 230Hz, a választott mintavételi idő 0.005s. Az emberi hangképzés sajátosságait, és az alapfrekvencia változásakor fellépő fiziológiai változásokat figyelembe véve ez az alapfrekvencia igen finom modellezését teszi lehetővé. (200 minta/másodperc).

4.2 Standardizálás és normalizálás

Mint ahogy azt korábban már említettem, a neurális hálózatokon a tanítás során úgy lehet sikerrel eljárni, ha a hálózatot és a tanító adatbázist kölcsönösen egymáshoz illesztjük [12]. A hálózat oldaláról - bár az elmélet szerint, ha a függvény folytonos, azt egy neurális hálózat képes közelíteni - könnyű végiggondolni, hogy az alkalmazott nemlineáris függvények értékkészletének korlátossága miatt nagy kimeneti érték előállítása problémás. Jelen kutatás során a hasonló nemzetközi próbálkozásokat tanulmányozva a bemeneti és a kimeneti adatokra különböző transzformációkat alkalmaztam. A kimeneti adatok egy hagyományos standardizáló algoritmussal skálázódtak 0.01 és 0.99 közé. Azért nem érdemes pontosan nullát és egyet választani, mert bizonyos hálózatok esetén a 0 és az 1 érték a végtelenbe tart, így itt instabillá válhat a hálózat. A bemeneti adatok esetén célszerűbb úgy áttranszformálni az adatbázist, hogy egy közel 0 várható értékű halmaz kerüljön a bemenetre minden iteráció során. Ennek megvalósítására egy alkalmas megoldás egy Gaussi standard skálázó használata. Az algoritmus nullába viszi a várható értéket,

és a szórást egységnyivé transzformálja, így megfelel az ilyen téren támasztott kritériumoknak. A skálázást globálisan, a teljes adatbázison egyszerre végeztem el, figyelve arra, hogy a validációs és teszt adatokat ne illesszük, hanem a tanító adatbázisnál meghatározott skálázási paraméterek továbbvitelével csak transzformáljuk. Ha nem így járnánk el, hanem külön illesztenénk és skáláznánk, akkor ezzel extra („jövőbeli") információhoz juttatnánk a rendszert, és ez meghamisítaná az objektív méréseket a validációs és teszt adatbázison. Az eredmények kiértékelésekor a kimeneti adatokon a vissza transzformációt is végre kell hajtani, ekkor is az eredetileg kialakított skálázó mátrixot használjuk.

4.3 Interpoláció

A neurális hálózat a külvilágból vett információit egy belső numerikus reprezentációvá alakítja át. Amennyiben a numerikus reprezentáció jól modellezi a megoldani kívánt problémát, a tanítás sikeres volt, és a hálózat a gyakorlati alkalmazás esetén is várhatóan jól fog teljesíteni. Ezen numerikus sajátosságokból adódóan azonban a nem folytonos függvények modellezése általában nagyobb problémát jelent a hálózatok számára. Az alapfrekvencia függvény modellezése során probléma, hogy a függvény nem folytonos, sőt nem is mindenhol definiált a vizsgált hangfájlokban. Az emberi beszédben előforduló hangok alapvetően két csoportra oszthatók, a zöngésekre és a zöngétlenekre. A zöngés hangoknak jól meghatározható alapfrekvenciája van, a zöngétlen fonémák gerjesztése azonban zajszerű, így ezen elemek esetén az alapfrekvencia értéke nem is definiált.

A neurális hálózat viszont csak numerikus adatokkal tud dolgozni, így amennyiben a tanítás végeztével mi egy a gyakorlatban is használható alapfrekvencia becslőt szeretnénk kapni, mindenképpen generálni kell szimulált kimenet értékeket azokhoz a mintákhoz is, amikhez a valóságban nem definiált f0 érték tartozik. Természetesen az interpoláció nem az egyetlen lehetséges megoldás a függvény folytonossá tételére, de mivel a jelen architektúra esetén ez került alkalmazásra, a továbbiakban ezzel foglalkozom.

A hálózattól elvárjuk, hogy az alapfrekvencia becslése mellett képes legyen különbséget tenni a valós és a szimulált f0 értékek között (ez tulajdonképpen önmagában tekintve egy bináris

osztályozási probléma). Ezt egy ún. zöngés/zöngétlen f0 kimenet bevezetésével oldottam meg. A hálózat a tanítás során ezt a kimenetet is optimalizálja az f0 érték mellett.

Az interpoláció során figyelnünk kell arra, hogy az f0 függvény alapvető jellegzetességeit megtartsuk. A TDK munka során csak kijelentő mondatok alkották a tanító adatbázist. A kijelentő mondatokra jellemző, hogy az f0 menet enyhén csökkenő tendenciát mutat. Egy hangfájl esetén az interpolálni szükséges mintasorozatok változó hosszúságúak, interpolált adat a hangfájl legelején és a végén is előfordulhat. Ezt az interpoláció során figyelembe kell venni. Az alkalmazott interpolációs technika a következő:

- minden interpolált szakasz lineáris, a kezdőpont a legutolsó definiált érték a szakasz előtt, a végpont a legelső definiált érték az interpolált szakasz után, ha léteznek ilyen pontok
- a lineáris interpolálás során kiszámoljuk a két adat különbségét, és az iránytól (növekvő vagy csökkenő) függően egyenletes lépésekkel alakítjuk ki a szimulált értékeket

Az interpolálás során fellépő probléma a hangfájlok elején és végén jelen lévő szünetek. Itt sincs definiált f0 érték, tehát ezt is interpolálni kell. Ekkor azonban probléma az esetleges szünetek és a mondat elején/legvégén lévő zöngétlen hangkarakterekkel való összemosás. Ezen probléma teljes feloldására a jelen TDK dolgozat során nem került sor, viszont egy köztes megoldás alkalmazására sor került a csökkenő f0 menet megtartása érdekében. Ha a hangfájl elején interpolálni szükséges sorozat van, az interpoláció a fájlban az első definiált érték 110%-ától indul (lefelé), ha a végén, akkor az utolsó definiált f0 érték 90%-áig interpolálunk (itt is lefelé). Ezzel a megoldással az interpoláció hibája egy elfogadható szintre csökkenthető. Hangsúlyozandó, hogy ez a megoldás közel sem a legoptimálisabb, többek között azért sem, mert a kijelentő mondatok esetén sem igaz az f0 függvény szigorú monotonitása.

4.4 A tanító adatbázis felépítése

A tanító adatbázist elkészítő script kimenete egy szöveges fájl, ahová soronként kerülnek be az egyes adatbázis elemek. A jelenlegi architektúrában a neurális hálózatnak összesen 363

darab bemenete és egy vagy két kimenete van, így egy sorba 364 vagy 365 db szám kerül, vesszővel elválasztva. A kimenet az f0 függvény becsült értéke az adott adatbázis elemre vonatkozóan, illetve (amennyiben használtam az adott tanítás során) a már korábban is említett bináris zöngés/zöngétlen flag. A 363 bemenet túlnyomó többsége bináris (ún. „one-hot" kódolású): 68 darab fonéma közül lehet választani, ezek mindegyike kapott egy bemenetet, így a 68 bemenet közül egy időben csak pontosan egy darab 1-es lesz. A bemenet ilyen formán történő képzése gyakori a neurális hálózatok esetén. A 68 darab fonéma egymás után ötször szerepel, mert a modellben rögzítésre kerül a korábban említetteknek megfelelően az adott fonémát megelőző, valamint az azt kettővel megelőző fonéma, valamint az azt követő és az azt kettővel követő fonéma is. Ez 68*5 = 340 bemenetet jelent. A maradék 23 bemenet numerikus, és különböző kiegészítő információkat tartalmaz az adott elemmel kapcsolatban. Ilyen bemenet például az adott adatbázis elem százalékos helyzete a fonémán belül (egy fonémához kb. 20-30 adatbázis elem tartozik, hiszen a korábban már említett SWIPE szoftver időfelbontása jóval precízebb, mint egy fonéma átlagos hosszúsága), de itt jelennek meg azok a már korábban említett paraméterek is, melyek az adott fonémát a mondaton belül tágabb környezetben helyezik el. A teljes szintaxis és az összes bemenet leírása megtalálható a Függelékben.

4.5 A neurális hálózati modellek összeállítása

A neurális hálózati modellek Python [19] programnyelv segítségével készültek, az ingyenesen elérhető Keras [21] csomag felhasználásával. A programcsomagban lehetőségünk van többféle neurális hálózati architektúra, köztük MLP felépítésére is. Bár a programcsomag még nagyon fiatal, máris jelentős felhasználói tábora van közérthetősége és a gyors prototípusfejlesztést támogató metódusai miatt. Egy MLP összeállításához definiálnunk kell egy hálózati struktúrát, majd ezután be kell vinnünk a legfontosabb paramétereket (például a tanulási ráta). A hibafüggvény és a tanító algoritmus pontos definiálása után az általunk használni kívánt tanítást segítő módszereket (amennyiben azok elérhetők) is hozzá kell adnunk a modellhez. A szoftver lehetőséget nyújt arra, hogy a tanítás végeztével az eredmény súlymátrixot elmentsük, amit teszteléskor vagy a tanulás továbbfolytatásánál újra használhatunk. A kutatómunka során

sokféle hálózati architektúra és paraméter beállítás tesztelésre került, ezekről bővebben az Eredmények fejezetben lesz szó.

5 Eredmények

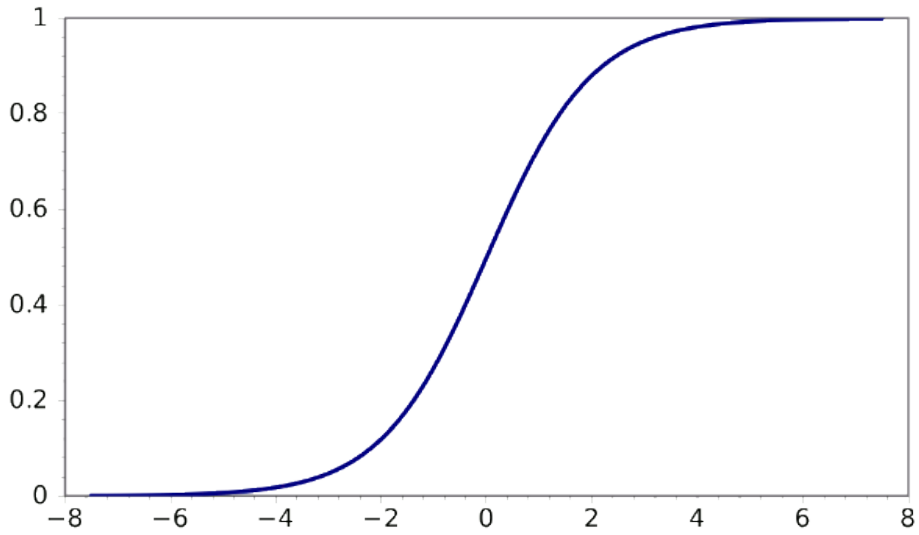
Ebben a fejezetben ismertetem a tanításhoz használt hálózati architektúrákat, a tanítás alapvető lépéseit, és az eredmények összehasonlítását a különböző architektúrák között. A következőkben felsorolt összes tanítás a *mini-batch learning* technikával történt a korábban már említett Stochastic Gradient Descent algoritmus segítségével. A *mini-batch* méret a szigmoid függvénnyel történő tanítások esetén a tanító adatbázis 2%-a volt, a $Tanh()$ és a *ReLu* függvényes tanítások esetén fix 5000 adatbázis elem volt egy *mini-batch* méret (kivéve a legutolsó, nagy neuronszámú eseteket, itt 1000 elem volt egy *mini-batch*). A kimenet minden esetben az alapfrekvencia természetes alapú logaritmusa (log f_0 és a bináris zöngés/zöngétlen flag volt). A tanítások addig folytatódtak, amíg 150 iteráción át nem csökkent a hiba. A Learning Decay technika alkalmazásra került, 10^{-6} vagy $5 * 10^{-7}$ paraméterrel. (Ez azt jelenti, hogy ezer iterációnként csökken egy ezredet (vagy fél ezredet) a tanulási ráta.) A táblázatokban szereplő adatokon kívül még sok eredmény keletkezett a munka során. A TDK dolgozatba a valamilyen szempontból érdekes eredmények kerültek be. Az eredmények minden esetben négyzetes középhiba-értéket jelentenek (Root Mean Square Error, RMSE) [17]. Bár beszédtechnológiai alkalmazásokban jelentős szerepe van a meghallgatásos és az egyéb szubjektív teszteknek, a neurális hálózatok területén nagyon elterjedt az eredmények ilyen formán történő prezentálása. Emellett az 5.4-es alfejezetben a tényleges becsült és referencia alapfrekvencia-menetek összehasonlítására is sor fog kerülni.

5.1 Szigmoid (logisztikai) aktivációs függvényű architektúrák

A neurális hálózatok egyik „klasszikus” aktivációs függvénye a szigmoid függvény. A szigmoid függvény a neurális hálózatok tudományterületén az

$$S(x) = \frac{1}{1+e^{-x}} \quad (3)$$

függvényt jelenti. Ez valójában az ún. logisztikai függvény képlete (logistic function). Ezen nemlineáris függvény grafikonja a 5. ábrán látható.



5. ábra – Szigmoid függvény grafikonja a [-8,+8] tartományon. Forrás: [3]

Bár az utóbbi években egyre ritkábban használják bonyolultabb gyakorlati feladatok esetén, érdemes megvizsgálni néhány tanítás esetére, hogy hogyan teljesít a hálózat ezzel az aktivációs függvénnyel. A szigmoid függvény, ahogy az ábráról is látható, 0 és 1 között korlátos nemlineáris függvény. A szigmoid egyik legnagyobb hátránya, hogy szenved az ún. eltűnő gradiens (*vanishing gradient*) problémától. A probléma lényege, hogy ha az aktiváció a függvény valamelyik szélső értékéhez közelít, a gradiens ezekben a régiókban már nagyon közel van a zérushoz. A közel zérus gradiens pedig a hiba-visszaterjesztéses algoritmus hatékony működését lehetetleníti el. Az ilyen aktivációs függvények esetén a tervezőnek figyelnie kell az adatok helyes inicializálására is, hiszen itt a kimenetek nem nulla várható értékűek.

A szigmoid aktivációs függvényekkel végzett tanítások eredményei:

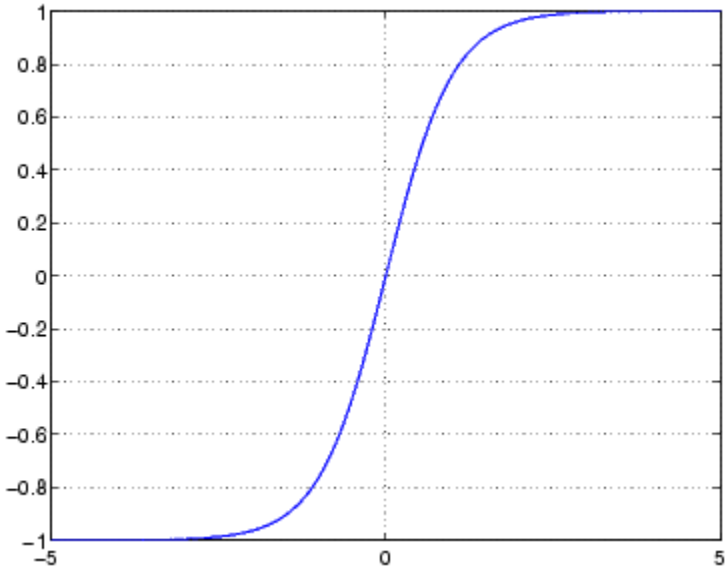
Rétegek száma	Neuron egy rétegben	Tanítási ráta	Momemtum paraméter	Dropout mérték	RMSE a teszt adatbázison
1	500	0.05	0	0	0.0555
1	1000	0.05	0	0	0.0519
1	1000	0.05	0.5	0	0.0538
2	1000	0.05	0.5(Nesterov)	0	0.0485
3	1000	0.03	0.5(Nesterov)	0	0.0478

1. táblázat – szigmoid tanítás eredmények

Az eredményeket tartalmazó táblázatban a neurális hálózatokra jellemző általános trendek megfigyelhetők. Az elméleti szemlélet alapján a neuron számok és a rétegek számának növelése (a hálózat bonyolultságának növelése) magában hordozná a hiba csökkenését (természetesen ez az összefüggés nem érvényes minden határon túl). Látható az is, hogy a momentum hatás ebben az esetben segített valamennyit, és a hálózat rétegeinek növelése is csökkentette a hibát, de egyik hatás sem volt jelentős. A hiba csökkenését szabályzó korlát itt vélhetően az aktivációs függvény maga. Mint az a következő alfejezetekben olvasható, a többi alkalmazott aktivációs függvény jobb teljesítményt nyújtott, mint a szigmoid függvényes hálózatok, így az ezzel a függvénnyel való további kísérletezés csak elméleti szempontból lenne érdekes.

5.2 Hiperbolikus tangens aktivációs függvényű architektúrák

Az elemi matematikából is jól ismert tangens hiperbolikus függvényt (illetve különböző változatait) a korai hálózatokban gyakran használták a hagyományos szigmoid függvény helyett.



6. ábra – a hiperbolikus tangens függvény. Forrás: [4]

Yann LeCun, a neurális hálózatok egy jelentős kutatója híres cikkében [11] az eredeti hiperbolikus tangens függvény (lásd 6. ábra) helyett annak egy módosítását javasolja, ami

$$f(x) = 1.7159 \tanh\left(\frac{2}{3}x\right) + ax \quad (4)$$

alakban írható le. A képletben a konstansok numerikus optimalizálási számítások eredményei. Az $+ax$ tag egy lineáris szakasz beiktatását jelenti, ez szintén a tanítás gyorsításában segíthet, azonban az a konstans meghatározására nincs konkrét ajánlás. Elméleti szempontból a hiperbolikus tangens függvény mellett szól szimmetriája, ami jobb gradienseket eredményez. Megjegyzendő azonban, hogy ez az aktivációs függvény elég számításigényes, és elméleti szempontból is egyre inkább túlhaladott. A számításigényt a függvény Taylor-soros közelítésével természetesen lehet mérsékelni, túl sok erőforrást azonban nem érdemes belefektetni, hiszen jelen feladat esetén a tapasztalatok szerint másfajta aktivációs függvénnyel jobb eredmények érhetők

el. A következő táblázatban felsorolt eredmények ezért a hagyományos $y = \tanh(x)$ függvény tanítására vonatkoznak.

Rétegek száma	Neuron egy rétegben	Tanítási ráta	Momemtum paraméter	Dropout mérték	RMSE a teszt adatbázison
1	1000	0.03	0.6	0.5	0.0347
1	2000	0.03	0.6	0.5	0.0325
2	1000	0.03	0.3	0.5	0.0364
3	1000	0.03	0.6	0.5	0.0331

2. táblázat – hiperbolikus tangens tanítási eredmények

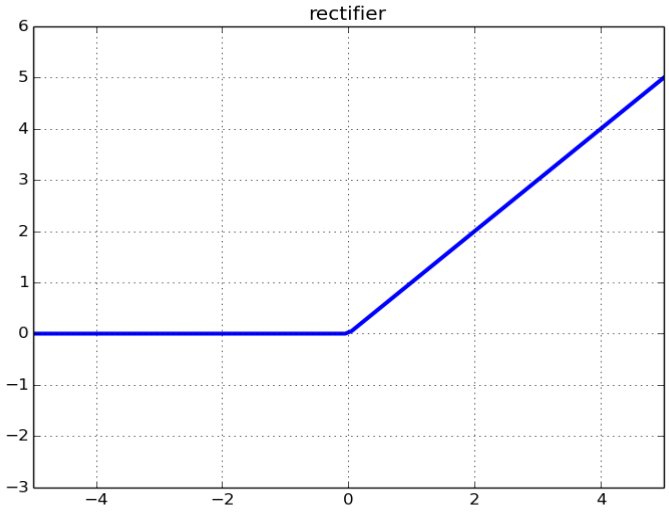
Az adatokból itt is látható, hogy a momentum módszer paraméterének csökkentése negatívan befolyásolja az eredményeket. Ez a hatást még a neuronok számának emelése is csak részben ellensúlyozza.

5.3 Rectified Linear Unit (ReLU) aktivációs függvényű architektúrák

A neurális hálózatok témakörében a „rectifier” egy ún. rámpafüggvény, ami az

$$f(x) = \max(0, x) \quad (5)$$

összefüggéssel írható le, ahol x az aktivációs függvény bemenete. A függvény grafikonja a 7. ábrán látható.



7. ábra – egy rámpafüggvény képe. Forrás: [5]

A Rectified Linear Unitnak (rövidítve ReLu-nak) nevezzük azt a neuront, amelyik valamilyen rámpafüggvényt használ aktivációs függvénynek. A legújabb biológiai kutatások és a gyakorlati tapasztalatok is igazolják az ilyen elven felépített neurális hálózatok működését. [24] cikk 2015 legnépszerűbb aktivációs függvényének nevezi a ReLu-t. Legnagyobb előnye a regressziós tanítások esetén mutatkozik meg. Az előzetes várakozásokat a ReLu-val történő tanítások igazolták: az ilyen hálózatok szignifikánsan jobban teljesítettek a szigmoidot és a hiperbolikus tangenset tartalmazó hálózati architektúrákhoz képest. Az eredmények a következő táblázatban olvashatók.

Rétegek száma	Neuron egy rétegben	Tanítási ráta	Momemtum paraméter	Dropout mérték	RMSE a teszt adatbázison
1	1000	0.05	0.8(Nesterov)	0.3	0.03024
2	1000	0.05	0.8(Nesterov)	0.3	0.02465
3	1000	0.01	0.8(Nesterov)	0.3	0.02444
3	1000	0.05	0.6(Nesterov)	0.5	0.01883
4	1000	0.05	0.6(Nesterov)	0.5	0.01836

3. táblázat – ReLu tanítás eredmények

Az adatokból látszik, hogy a ReLu-val működő hálózatok nagyon érzékenyek a Dropout mértékére, nagyobb Dropout jelentős hibacsökkenést, és ezáltal a végeredmény javulását okozta.

A ReLu-t használó hálózatokkal való további kísérletezés eddigi legjobb eredményeit a következő táblázat tartalmazza (ezen tanításoknál a batch size kisebb, 1000-es értéket kapott):

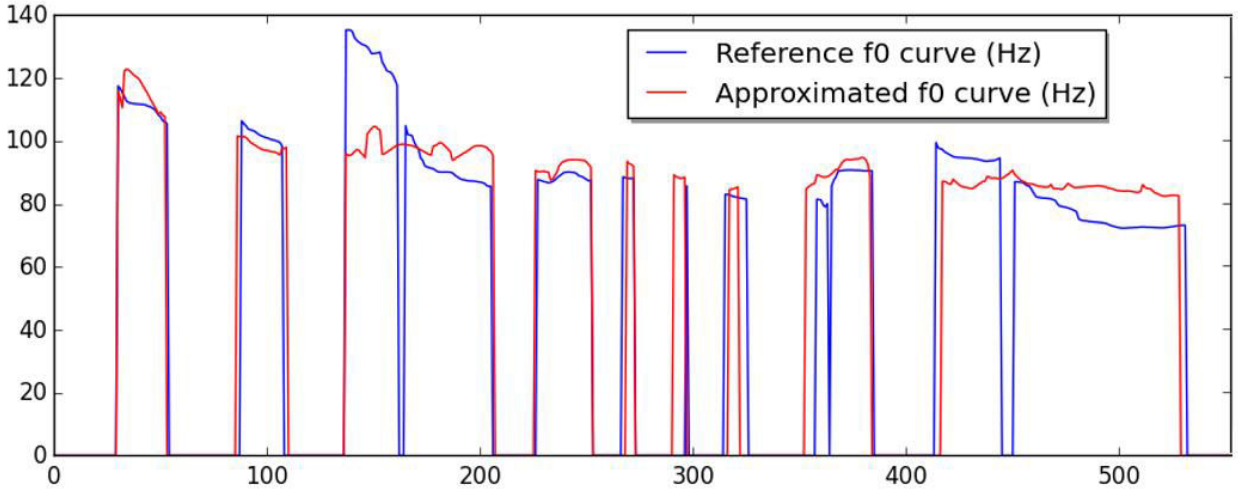
Rétegek száma	Neuron egy rétegben	Tanítási ráta	Momemtum parameter	Dropout mérték	RMSE a teszt adatbázison
2	2000	0.05	0.9(Nesterov)	0.5	0.01460
2	3000	0.08	0.8(Nesterov)	0.5	0.01351
1	4000	0.05	0.9(Nesterov)	0.5	0.01413

4. táblázat – ReLu tanítás eredmények - legjobbak

Az eredmények a nagyobb neuron számok alkalmazásával valamelyest javultak, de a kisebb batch méret miatt a számításigény és ezáltal a tanítás ideje lényegesen emelkedett.

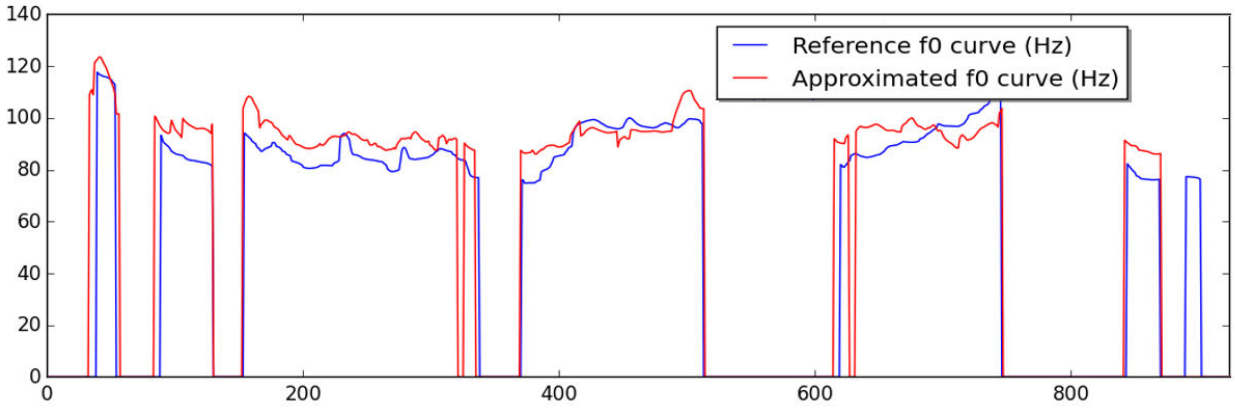
5.4 Összehasonlítás a referencia alapfrekvencia függvényekkel

Az előző alfejezetek eredmény táblázatai négyzetes középhiba (RMSE) értékeket tartalmaztak, amelyek jól összehasonlíthatók, viszont kevéssé szemléletesek, és szubjektív minőséget egyáltalán nem hordoznak. Ahhoz, hogy valósabb képet kapjunk rendszerünk teljesítményéről, mindenképpen szükséges a teljes függvénymenet megvizsgálása. Erre a célra egy kiértékelő programot készítettem, mely a teszt adatbázis referenciafüggvényeit és a tanított hálózat e függvényekre vonatkozó becsléseit egy ábrába rajzolja ki. A programot az egész teszt adatbázisra lefuttatva és az így kapott ábrákat vizsgálva pontosabb képet kaphatunk az elért eredményekről. A következőkben néhány ábra szerepel a legjobb tanítások alapján. Az ábrák kizárólag a teszt adatbázisból, azaz a tanítás során nem szereplő, a tanított hálózat számára ismeretlen mondatok alapfrekvencia-függvényeiből állnak. Az ábrákon a vízszintes tengelyen a tanító adatszám, a függőlegesen az alapfrekvencia érték szerepel, Hertz [Hz] egységben. A kék a referencia hangfájl, a piros a hálózat által becsült érték. A tanítások a táblázatban olvasható legkisebb hibát produkáló architektúra alapján készültek.



8. ábra – ismeretlen hangfájl becslése, x tengely: minták, y tengely: frekvencia érték (Hz)

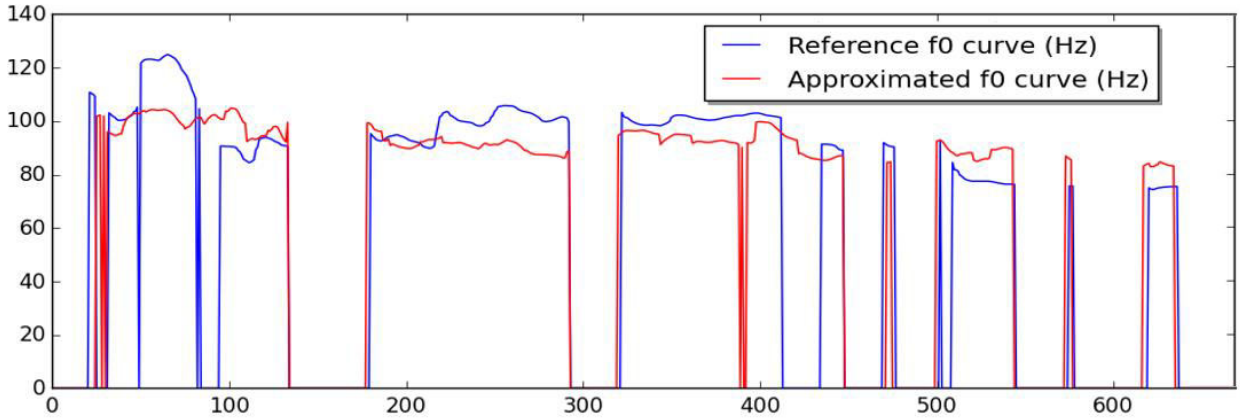
A 8. ábrán látható, hogy az egyik legfontosabb tulajdonságot, az enyhén ereszkedő tendenciát sikeresen követte le a hálózat. A szubjektív teszteken az offszet jellegű eltérések (amennyiben nem túl nagyok) várhatóan nem fognak túl nagy problémát okozni a minőségben, hiszen ez csupán azt jelenti, hogy a beszélő hangja egy kicsit magasabb, vagy alacsonyabb lesz.



9. ábra – ismeretlen hangfájl becslése 2, x tengely: minták, y tengely: frekvencia érték (Hz)

A 9. ábrán is látható – különösen a hosszabb szakaszok esetén, hogy a két függvény menete hasonló. Természetesen vannak rosszabbul sikerült szakaszok is, például az ábra végén lévő rövid rész. Azt, hogy a piros görbének hol kell nullát illetve nem nullát adnia, szintén ugyanazon tanításkor becsült bináris zöngés/zöngétlen flag bit dönti el. (Ennek a becslése sem

száz százaléking pontos, az általagos pontosság ebben az esetben 97% és 99% között volt). Ez az eltérés okozza azokat a szakaszokat, ahol a kék (referencia) görbe nulla értéket ad, a piros (becsült) nem nullát. A 9. ábrán például a 600 és 640 közötti mintáknál látható ilyen eltérés.



10. ábra – rosszabbul becsült ismeretlen hangfájl, x tengely: minták, y tengely: frekvencia érték (Hz)

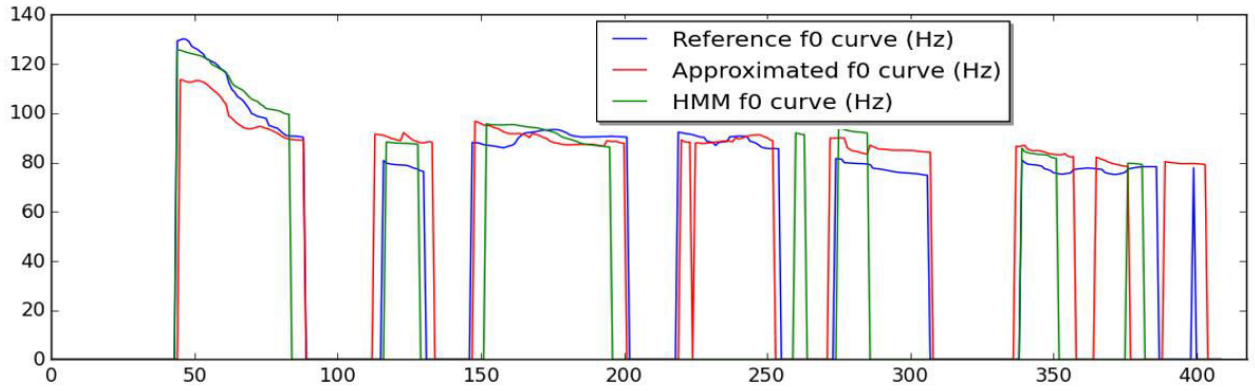
A 10. ábrán – különösen az első százötven minta esetén nagy a rossz flag becslés okozta eltérés. A szubjektív tesztek lezárultjával több információ lesz arról, hogy ez mennyiben okozza a hangminőség romlását.

5.5 Összehasonlítás a korábbi rendszer (HMM) által becsült alapfrekvencia menettel

Érdemes megvizsgálnunk azt is, hogy más modern rendszerekhez képest hogyan teljesít a DNN alapú f0 becslő. A TMIT SpeechLab-nál korábban készült HMM alapú megoldás [1] alkalmas – többek között – az f0 paraméter becslésére is. Itt azonban nincs RMSE érték, vagy más könnyen összehasonlítható mérőszám, így ismét a grafikus ábrák segítségével célszerű összehasonlítani a teljesítményt.

A HMM is ugyanazon az adatbázison lett tanítva, és ugyanúgy ismeretlen adatnak számítanak a DNN-nél használt teszt adatbázis elemei. A következőkben néhány ábrát láthatunk, amelyeken kékkel szerepel a referencia alapfrekvencia függvény, zölddel a HMM-es rendszer

által becsült, pirossal a DNN által készített görbe. A jelenlegi rendszerben az eltérő időzítések miatt a HMM-es adatok ábrázolása nem tökéletes. Így a HMM teljesítménye a valóságosnál rosszabbnak tűnhet az ábrák alapján. Szerencsére ez nem jelent olyan nagy problémát, hiszen a végső minőségi összehasonlítás nem ábrák, hanem valóságos személyek értékelése alapján fog összeállni.



11. ábra – DNN, HMM és a referencia függvény

Az 11.ábrán látható, hogy a HMM alapú rendszer inkább a függvény lényeges vonásait próbálja lekövetni, az alapfrekvencia kis ugrásait kevésbé. A HMM és a DNN is egyaránt produkál túllövéseket és elcsúszásokat is, ami arra utal, hogy egyik rendszer zöngés/zöngétlen becslője sem tökéletes, és a függvénybeli szakadások illetve az ugrások lekövetése nagy problémát okoz. A látottakból arra következtethetünk, hogy érdemes a DNN alapú rendszerekbe további erőforrásokat fektetni, mert – a DNN tanítási módszerek fejlődésével – lehetőség nyílik az alapfrekvencia függvény olyan pontosságú modellezésére, amely HMM esetén nem elérhető.

6 A kutatómunka technikai részletei

Jelen fejezet röviden összefoglalja a kutató- és fejlesztőmunkám során eddig elért eredményeket, és beszámol a konkrét munkához használt fejlesztői eszközökről.

A TDK dolgozat megvalósításához nélkülözhetetlen segítséget nyújtott a TMIT Speechlab. A szerzőnek lehetősége volt modern grafikus processzorral felszerelt számítógépeken tanítani a hálózatokat. (Nvidia GeForce GTX 970 és később Nvidia GeForce GTX Titan X típusok). A kezdeti lépések 2015 februárjától indultak a szerző önálló laboratórium projekttantárgya keretében. A GPU-s tanítások 2015 szeptember közepén kezdődtek. Az azóta eltelt időben a tesztrendszer több módosításon is átesett a hatékonyabb működés érdekében. Mivel a kutatás során használt eszközök többsége – és a működésüket megalapozó tudományos eredmények – is viszonylag újnak számítanak, a munka során figyelemmel kellett kísérni a fejlesztői eszközök gyakori módosulásait is.

Munkámhoz az ilyen jellegű kutatásoknál gyakran használt Python nyelvet választottam. A Python nyelv előnye, hogy a tudományos kutatásoknál használt többi megoldással szemben (MATLAB, R programnyelv) általánosabb célú programnyelv, így a fejlesztők lehetőségei szinte korlátlanok a felmerülő problémák megoldását illetően, ingyenessége és könnyű bővíthetősége pedig nagyfokú rugalmasságot tesz lehetővé. Az GPU-n történő alacsony szintű programozás megvalósításában nagy segítséget nyújtott a Theano [18] nevű, erre a célra kitalált Python könyvtár. A Python nyelv adatkezelését nagyban könnyítik az ingyenes Numpy és a Scipy [19] csomagok, a hatékony és könnyed adat vizualizációt pedig a Matplotlib [20] könyvtár támogatja. A neurális hálózati modellek összeállítása többek között a Theno-val kompatibilis Keras [21] osztálykönyvtárral történt.

Munkám során eddig több mint 800 sor Python kód született, és eme TDK dolgozat megírásáig 41 különböző hiperparaméter-beállítással folyt valamelyik GPU-n tanítás. A kutatás során feldolgozott adatbázis közel 2000 egyedi hangfájlból áll (ezek egyetlen beszélőtől származnak, természetesen a későbbiekben lehetőség van más beszélők vizsgálatára is), melyekből a kinyerhető tanítási adatbázis közel másfél millió elemből áll. Egy tanítás átlagosan 8-10 órányi GPU és CPU időt vesz igénybe a jelenlegi, optimalizált teszt rendszerrel.

7 Összefoglalás – jövőbeli tervek

Ez a fejezet kitér a lehetséges jövőbeli fejlesztési irányokra, valamint a munkám várható folytatására a közeljövőben.

A jelen TDK dolgozathoz kapcsolódó kutatás keretében az emberi beszéd alapfrekvenciájának becslésére alkalmas tesztrendszer készült el. Mély neurális hálózatok segítségével a kiválasztott beszélő alapfrekvencia-görbéi közelíthetők. Ez a tudás beépíthető a jövő mesterséges beszédkeltő rendszereibe a jobb beszédminőség elérése érdekében. A rendszer működése természetesen független a beszélő személyétől, így a megfelelő minta hangfájlok birtokában tetszőleges személy beszédparaméterei modellezhetővé válnak. Az eredményeket figyelembe véve látható, hogy a DNN alapú rendszer sikerrel veszi fel a versenyt a korábbi HMM-es megoldással, sőt, néhány esetben pontosabb becslésre is képes. A DNN alapú rendszerek további fejlesztésével várhatóan a beszédparaméter-modellezés egy új szintje lesz elérhető.

A kutatás a közeljövőben az alapfrekvencia mellett a többi fontos beszédparaméter (a spektrális vagy más néven mel-cepstral paraméterek) becslésével fog folytatódni. A tesztrendszer könnyen átírható ezek akár együttes tanítására is. Az ilyen további beszédparaméterek vizsgálatával – és megfelelő hardveres háttérrel - megvalósulhat akár a teljes gépi beszéd mesterséges neurális hálózatokkal történő előállítása is.

A jelenlegi tesztrendszer fejlesztése terén két jelentős fejlesztési irány látható: az egyik az adatreprezentáció további finomhangolása, fejlesztése. Itt jelenleg javítandó tényezőnek felsorolható a zöngétlen hangok és a hangfájl eleji és végi szünetek közötti összemosás csökkentése, és az interpoláció által bevitt hiba minimalizálása. A másik fő fejlesztési irány a neurális hálózati architektúrák további hangolása a megoldandó feladathoz. Itt a továbblépéshez követni kell a legújabb tudományos publikációkat is, hiszen a fiatal tudományterület gyakran produkál új, meglepő eredményeket. Jelenleg gyorsan megvalósíthatónak látszik a [22] által is emlegetett ReLu modellek továbbfejlesztéseinek (például az ún. Leaky ReLu) bevezetése. Jelenleg folyamatban van a becsült eredményekkel a hangfájlok generálása, illetve rövidesen sor fog kerülni ezek birtokában a szubjektív, emberek általi meghallgatásos tesztekre is. Ezek során

az eredeti és a korábbi, HMM-es rendszer által generált mintákkal hasonlítjuk majd össze a DNN-es megoldást.

Forrás és irodalomjegyzék

[1] Tóth, B., & Németh, G. (2008). Rejtett Markov-modell alapú mesterséges beszédkeltés magyar nyelven. *LXIII. köt*, 2-6.

[2] Zen, Heiga, Andrew Senior, and Mike Schuster. "Statistical parametric speech synthesis using deep neural networks."*International Conference on.Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE* 2-3.

[3] Dudley, H. (1940). The Vocoder—Electrical Re-Creation of Speech. *Journal of the Society of Motion Picture Engineers*, *34*(3), 272-278.

[4] Liberman, A. M., Cooper, F. S., Shankweiler, D. P., & Studdert-Kennedy, M. (1967). Perception of the speech code. *Psychological review*, *74*(6), 431.

[5] Sproat, R. W. (1997). *Multilingual text-to-speech synthesis*. KLUWER academic publishers. 29-38.

[6] Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review* 65.6 (1958): 386.

[7] Altrichter Márta, Horváth Gábor, Pataki Béla, Strausz György, Takács Gábor, Valyon József : Neurális Hálózatok. 2006 Hungarian Edition Panem Könyvkiadó Kft., Budapest, 81-112.

[8] Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504-507.

[9] Kussul, E., & Baidyk, T. (2001). Rosenblatt perceptrons for handwritten digit recognition. *International Joint Conference on Neural Networks, 2001. Proceedings. IJCNN'01.* (Vol. 2,). IEEE, 1516-1520

[10] Zeiler, M. D., Ranzato, M. A., Monga, R., Mao, M., Yang, K., Le, Q. V., ... & Hinton, G. E. (2013, May). On rectified linear units for speech processing. *International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE*. IEEE, 3517-3521.

[11] LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K. R. (2012). Efficient backprop. In *Neural networks: Tricks of the tradem* Springer Berlin Heidelberg, 9-48.

[12] Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th international conference on machine learning (ICML-13)*,1139-1147.

[13] Prechelt, L. (1998). Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks*, *11*(4), 761-767.

[14] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929-1958.

[15] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, *5*, 3.

[16] Camacho, Arturo. SWIPE: A sawtooth waveform inspired pitch estimator for speech and music. Phd. University of Florida, 2007.

[17] Zhang, G., Patuwo, B. E., & Hu, M. Y. (1998). Forecasting with artificial neural networks:: The state of the art. *International journal of forecasting*, *14*(1), 35-62.

[18] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., ... & Bengio, Y. (2010, June). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for scientific computing conference (SciPy)* (Vol. 4, p. 3).

[19] Oliphant, T. E. (2007). Python for scientific computing. *Computing in Science & Engineering*, *9*(3), 10-20.

[20] Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in science and engineering*, *9*(3), 90-95.

[21] Keras: Theano-based Deep Learning library. www.keras.io (Hozzáférés: 2015.10.20.)

[22] LeCun, Yann; Bengio, Yoshua; Hinton, Geoffrey (2015). "Deep learning". *Nature* 521: 436–444.

[23] Hecht-Nielsen, R. (1989, June). Theory of the backpropagation neural network. *1989. IJCNN., International Joint Conference on Neural Networks* , IEEE, 593-605.

[24] Olaszy Gábor: Precíziós, párhuzamos, magyar beszédadatbázis fejlesztése és szolgáltatásai 2013. 261-270.

[25] Markó Alexandra – Bóna Judit 2012. Fundamental frequency patterns: The factors of age and speech type. In: Calamai, Silvia – Celata, Chiara – Ciucci, Luca (eds.) Proceedings of 'Sociophonetics, at the crossroads of speech variation, processing and communication'. Edizioni della Normale. Pisa. 45–48.

Az ábrák jegyzéke

[1] http://project.mit.bme.hu/mi_almanach/books/aima/ch20s05 (Hozzáférés:2015.10.18.)

[2] https://www.willamette.edu/~gorr/classes/cs449/figs/earlystop.gif (Hozzáférés: 2015.10.18.)

[3] http://tikalon.com/blog/2011/sigmoid.gif (Hozzáférés: 2015.10.18.)

[4] http://eg1002.pbworks.com/f/tanh.gif (Hozzáférés: 2015.10.18.)

[5] http://danielnouri.org/media/kfkd/rectifier.png (Hozzáférés: 2015.10.18.)

Függelék

A felhasznált fonémakészlet

(68 db fonéma)

"a","a1","e","e1","i","i1","o","o1","o2","o3","u","u1","u2","u3","b","bb","c","cc","cs","cscs","d","dd","dz","dzdz","dzs","dzsdzs","f","ff","g","gg","gy","gygy","h","hh","j","jj","k","kk","l","ll","m","mm","n","nn","ny","nyny","p","pp","q","qq","r","rr","s","ss","sz","szsz","t","tt","ty","tyty","v","vv","zs","zszs","pau","ssil","esil","x"

A neurális hálózat bemeneti és kimeneti paramétereinek listája

Bináris bemenetek:

(fonémaszám (itt 68) * karakterek száma (itt 5)) darab bináris bemenet

- p_1: A kiejtés szerinti átírásban az éppen aktuális fonémát (p3) kettővel megelőző fonémát jelöli.
- p_2: A kiejtés szerinti átírásban az éppen aktuális fonémát (p3) megelőző fonémát jelöli.
- p_3: Az éppen aktuális fonémát jelöli, erre vonatkozik az összes többi azonos sorban lévő címke.
- p_4: Az aktuális fonémát (p3) követő fonémát jelöli.
- p_5: Az aktuális fonémát kettővel követő fonémát jelöli

Numerikus bemenetek:

(21 darab elemi paraméter és a speciális bemenetek)**:**

- p_6: A jelenlegi fonéma szótagon belüli pozíciója, sorszáma.
- p_7: A jelenlegi fonéma hátulról számított pozíciója a szótagon belül.
- a_3: Az előző szótagban szereplő fonémák száma.
- b_3: A jelenlegi fonémához tartozó szótagban szereplő fonémák száma.
- b_4: Az éppen aktuális fonémához tartozó szótag pozíciója, sorszáma a jelenlegi szóban.

- b_5: Az éppen aktuális fonémához tartozó szótag hátulról számított pozíciója, sorszáma a jelenlegi szóban.
- b_6: A jelenlegi szótag pozíciója, sorszáma az adott tagmondatban.
- b_7: A jelenlegi szótag hátulról számított pozíciója, sorszáma az adott tagmondatban.
- c_3: A következő szótagban lévő fonémák száma.
- d_1: Az előző szóban szereplő szótagok száma.
- e_1: A jelenlegi szóban szereplő szótagok száma.
- e_2: A jelenlegi szó tagmondaton belüli pozíciója, sorszáma.
- e_3: A jelenlegi szó tagmondaton belüli hátulról számított pozíciója, sorszáma.
- f_1: A következő szó szótagszáma.
- h_1: A jelenlegi tagmondat által tartalmazott szótagok száma.
- h_2: A jelenlegi tagmondat szavainak száma.
- h_3: A jelenlegi tagmondat mondatbeli pozíciója, sorszáma.
- h_4: A jelenlegi tagmondat hátrafelé számított mondatbeli pozíciója, sorszáma.
- j_1: A jelenlegi mondat által tartalmazott szótagok száma.
- j_2: A jelenlegi mondat által tartalmazott szavak száma.
- j_3: A jelenlegi mondat által tartalmazott tagmondatok száma.

Speciális bemenetként szerepelt az adott fonéma hossza, illetve az adott f0 érték százalékos pozíciója az adott fonémán belül.

Kimenetek:

(1 darab numerikus és 1 darab bináris, ha jelen van):

- $log(f_0)$: A becsült alapfrekvencia érték természetes alapú logaritmusa
- *zöngés/zöngétlen flag*: értéke 0, ha zöngétlen, 1, ha zöngés az adott beszédhang

Adatbázis fájl részlet

(minta)

2681716 3227280 v^i-l+a1=g@1_2/A:x_3_2/B:x-0-2@2-3&3-7#0-0$1-0!x-x;1-6|a1/C:x+0+2/D:1/E:4@2+3/F:2/G:x_x/H:9=4!1=3/I:5_2/J:22+9-3/L:0&x$x!x#x+x;x;x/M:x!x!x+

SWIPE eredmény részlet

(minta)

0.2700000 129.6627

0.2750000 129.4287

0.2800000 129.4287

Neurális hálózat bemeneti adat

(egy darab minta adatvektor, az utolsó két elem a kimenet, az előtte lévők a bemenetek)

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.12516221744836514, 0.044931489668579308, 0.40000000000000002, 0.60000000000000009, 0.60000000000000009, 0.60000000000000009, 0.5, 0.66666666666666663, 0.33333333333333331, 0.66666666666666663, 0.60000000000000009, 0.33333333333333331, 0.83333333333333326, 0.60000000000000009, 0.80000000000000004, 0.5, 0.83333333333333337, 1.0, 0.5, 1.0, 1.0, 0.75, 1.0, 4.864936463222774, 1]