
Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

# EKG alapú szívinfarktus klasszifikáció konvolúciós és rekurrens neurális hálózatokkal

**TDK dolgozat**

Készítette:

Ritter Áron

Konzulens:

dr. Szegletes Luca

2020

# Tartalomjegyzék

# Kivonat

Napjainkban a gyorsan fejlődő mesterséges intelligenciát egyre több helyen használjuk fel olyan feladatokhoz is, amelyeket korábban emberi beavatkozás nélkül megoldhatatlannak képzeltünk el. A neurális hálózatok folyamatosan kutatott területe azok orvostechnikai felhasználásának lehetőségei. Ezen a megoldásoknak a célja az orvosok munkájának megkönnyítése, a hibák valószínűségének csökkentése.

Az egyik ilyen feladat a myocardial infarction, azaz a szívinfarktusok diagnosztizálása. Ezt jelenleg az orvosok az elektrokardiogram (EKG) jel elemzésével végzik. Ez komoly specializált tudást és tapasztalatot, valamint időt igényel. Ha lenne olyan módszer, amely gyorsan megbízható osztályozást ad, azzal rengeteg forrást spórolhatnánk meg. Ezért a dolgozat célja, olyan módszerek kidolgozása, amelyek a célkitűzés teljesítését közelebb hozzák.

Munkám első részében a jelek klasszifikációjára gyakran használt konvolúciós neurális hálót (CNN) hoztam létre, melynek tanítását nehezíti az aritmia osztályozáshoz található adathalmazok kis mérete és a heterogenitása, mely a különböző szívritmuszavarok gyakoriságának különbségéből fakad. Ennek következményeként több olyan módszert is alkalmazni kell, ami segíti a háló gyors tanulását.

Munkám második részében az idősorokhoz gyakran használt rekurrens hálózatok -azokon belül is egy long short-term memory (LSTM) hálózatot - valósítok meg, majd összehasonlítom az első részben kapott eredményekkel.

# Abstract

Nowadays, the rapidly evolving artificial intelligence is used in more and more places for tasks that we previously thought were unsolvable without human intervention. An area of neural networks that is constantly being researched is their potential for medical use. The purpose of these solutions is to facilitate the work of doctors and reduce the likelihood of errors.

One such task is to diagnose myocardial infarction. This is currently done by doctors by analyzing the electrocardiogram (ECG) signal. This requires serious specialized knowledge and experience as well as time. If there were methods that would give a reliable classification quickly, we could save a lot of resources. Therefore, the aim of this work is to develop methods that can bring the possibility closer to achieve this goal.

In the first part of my work, I created a convolutional neural network (CNN), often used to classify signals, which is hampered by the small size and heterogeneity of data sets for arrhythmia classification due to differences in the frequency of different arrhythmias. As a result, several methods need to be used to help the training of neural network.

In the second part of my work, I implement the recurrent networks that are often used for time series, including a long short-term memory (LSTM) network, and then compare them with the results obtained in the first part.

# 1. fejezet

# Bevezetés

Világszerte a legmagasabb halálozási rátával a szív- és érrendszeri betegségek (cardiovascular diseases cvd) rendelkeznek, ezen belül is a legveszélyesebb az akut miokardiális infarktus (AMI), köznyelvben szívinfarktusként vagy szívrohamként ismert. Ez a szívizom súlyos vérellátási elégtelensége (myocardial ischemia) miatt bekövetkező szívizomelhalást jelöli, aminek az oka az hogy oxigénben dús vér nem tud a szívhez eljutni, mivel az artéria egy rövid időre beszűkül vagy eltömődik.

A szív strukturális redundanciájából fakadóan képes tünetmentes működésre, az elveszett szívizom aránya a szív össztömegéhez képest akár 10% is lehet, bár ez számtalan tényezőtől függ. A tünetmentesség ellenére viszont a CVD-k kockázata és súlyossága megnő, emiatt a diagnosztizálásának jelentős szerepe van. Az MI-k vezető tünete a szegycsontnál jelentkező, váratlanul megjelenő, szúró, nyomó jellegű mellkasi fájdalom, amely sokszor kisugárzik a bal karba, vállba vagy a nyakba. Emellett tünetei között tartjuk számon többek között még a sápadtság, hányinger, gyengeség, izzadás. Azonban a tünetek az idős kor vagy a cukorbetegség miatt hiányozhatnak, így a tünetek alapján történő diagnosztizálás gyakran nem használható. A pontos kórkép felállításához elektrokardiogram (EKG) vagy a vérben található biomarkerek kimutatására van szükség.

Az elektrokardiogram a szív elektromos aktivitását méri a bőrre helyezett elektródák segítségével. Ezzel a módszerrel szív- és érrendszer működésének monitorozását végzik. Azonban a kiértékelése nagymértékű szakirányú tudást illetve időt igényel. A munkám célja, hogy olyan módszert találjak, amellyel mind a kielemzéshez elvárt szaktudás, mind a szükséges idő jelentősen csökken.

Ehhez az elmúlt évtizedekben rohamosan fejlődő mesterséges intelligenciát, azon belül is a deep neurális hálózatokat használom fel. A hálók gyors előrelépését, a számítógépek számítási sebességének megnövekedéséhez és az egyre nagyobb mértékben rendelkezésre álló adat mennyiségének köszönhető. Viszont az EKG alapú szívinfarktus felismerésben is, mint a legtöbb neurális háló egészségügyi felhasználásában, nem áll nagy adatmennyiség rendelkezésre.

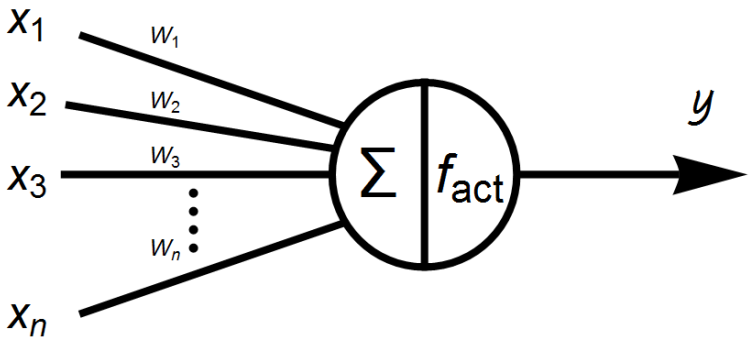
Ennek a problémának az elkerülése érdekében az idősorokhoz tervezett recurrent neural network (RNN) és azon belül is Long short-term memory (LSTM) architektúrát alkalmazom munkámban.

# 2. fejezet

# Technológia

## 2.1. Mesterséges neurális hálózatok

Mesterséges neurális hálózatok idegen névvel artificial neural networks (ANN) elméleti háttere már a késői 1940-es években megjelent[6]. Inspirációja a biológiai neuron hálózat azaz az emberi agy.




**2.1. ábra.** Elemi neuron modellje

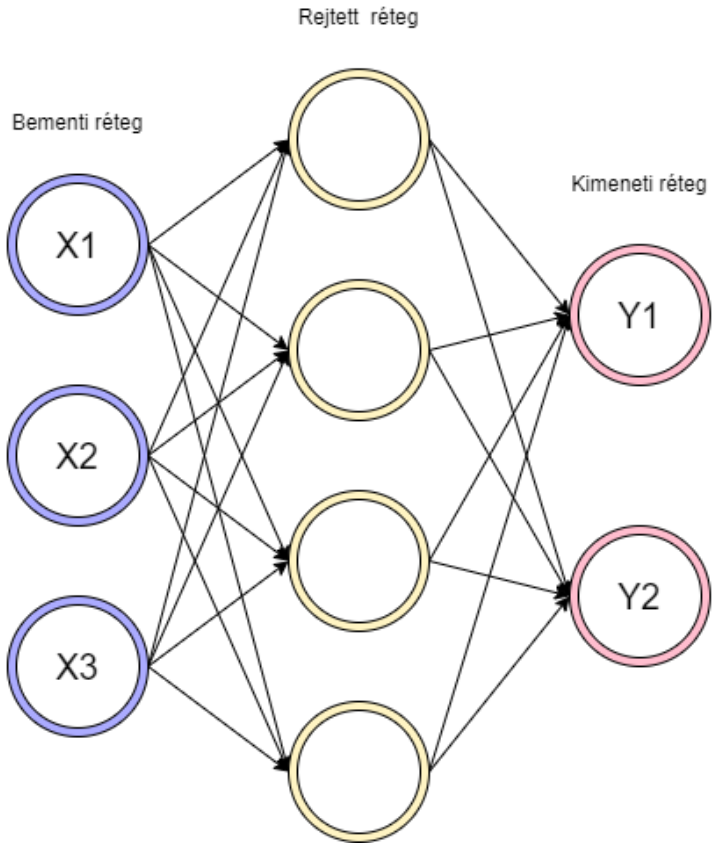
A *2.1. ábrán* megtekinthető egy neuron modellje. Amelyhez tartozó értékek a következő módon számíthatóak:

$$s = \sum_{i=1}^{n} W_i \cdot x_i, \tag{2.1}$$

$$y = f_{\text{act}}(s), \tag{2.2}$$

ahol az $s$ a neuron értéke, az $x_i$ a neuron $i$-edik bemenete, a $W_i$ az ehhez tartozó súly, az $y$ a neuron kimenete míg az $f_{act}(s)$ az aktivációs függvény.

Az elemi neuronokat, mint csúcspontokat felhasználva felépítünk egy körmentes (aciklikus) irányított gráfot. Egy ilyenre példa a 2.2. ábrán látható. Az azonos szinten álló csúcsok neve a réteg (layer), közöttük közvetlen kapcsolat nincs, viszont mind a bemeneti mind a kimeneti halmazuk általában azonos.




**2.2. ábra.** Neurális háló körmentes irányított gráfként

Hátterében a Hebbian teória áll, melyet Donald Hebb fogalmazott meg először az 1949-es The Organization of Behavior című könyvében [6], mely kimondja, hogy *a tanulás nem passzív folyamat, hanem az ideghálózatban ideiglenesen vagy véglegesen bekövetkező biokémiai és fiziológiai változások összessége – a neuroplaszticitás – mely szerint az együtt tüzelő neuronok egymás iránt fogékonyabbak lesznek.*

Mivel a neurális hálózatokban felhasznált elemi neuronok egyszerűsített modellek, ezért felépítésük viszonylag egyszerű. Három részből állnak, bemenet, érték és kimenet. Ez a 2.1. ábrán rendre a $W_i \cdot x_i$ kifejezés, $s$ érték, és a $f_act(s)$ érték. A hálót egészében nézve ezek a 2.2. ábráról a rejtett réteg egyes neuronjaihoz rendre a bemeneti réteg, a neuron saját értéke, és a kimeneti réteg. Azaz a befelé mutató él a bemenet, a csúcs az érték és a kifelé mutató él a kimenet. A csúcs értéke bemeneti értékeinek lineáris kombinációjából kiszámolható, ez a 2.1. egyenleten látható. A 2.2. egyenlet található aktivációs függvénnyel írható le a kimenet és a neuron értéke közötti összefüggés. Ez itt említett $f_act$ aktivációs függvény általában nem linearitásokat tartalmaz. Ez azért szükséges, hogy a rendszerünk ne csak lineáris kombinációkat tudjon megtanulni. Ezzel az elkészített neurális háló egy univerzális aproximátor.

Az univerzális aproximátor tétel (Universal approximation theorem) kimondja, hogy egy *megfelelően mély neurális hálózat meg tud közelíteni bármilyen tetszőlegesen komplex összefüggést a bemeneti változói között.* Továbbá belátták, hogy a ReLU aktivációs függvénnyel hálózat, amely

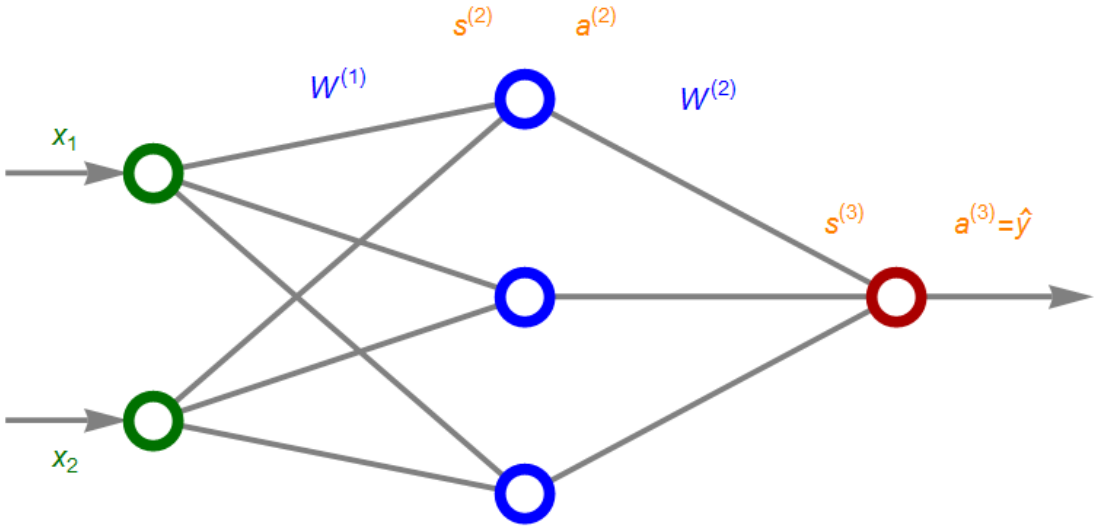
n + 1 széles képes megfelelően megközelíteni bármelyik folytonos n-dimenziós változó bemenetű függvényt [13].

Ahogy már korábban megjegyeztem, az ANN-ek neuronokból álló rétegekre bonthatóak. Megkülönböztethető sekély (shallow) és mély (deep) az ANN-eken belül, mely elnevezés nem egyértelmű. Azok a hálók melyek bemenete kézzel készített a feladat heurisztikáján alapuló jellemzőket kap sekélynek nevezzük, ezek általában kevesebb rétegből állnak, mint a mély hálózatok, melyeknél a feladat jellemzőit maguk tanulják meg.

### 2.1.1. Feed forward

A feed-forward metódus, a feed forward neural network (FFNN) alapjául szolgál. Ezekről a hálózatokról részletesen később lesz szó a 2.2. fejezetben. A kifejezés a rendszer irányításból származik, ahol a vezérló jelet gyakran feed-forward módon továbbították a rendszerben. Ez a FFNN-ekben a bemeneti adatok tovább propagálását mutatják. Ez azzal is jár, hogy ezen hálózatokban a hálózat leírására használt gráf nem tartalmazhat kört. Ezzel a módszerrel lehet a köztes értékeket kiszámolni a hálóban. Mivel a réteg összes bemenetére szükség van, hogy az értékét kiszámoljuk, így a sorrend meg van határozva. A bemeneti réteggel kezdődik és "előre terjed" a kimeneti rétegig az adat.

A *2.3. ábrán* látható egy rejtett rétegű hálózat, mely rétegei mind fully-connected (2.2.1. fejezetben részletesebben tárgyalva) rétegek. *A feed-forward metódus bemutatásának érdekében a hálózatban egyszerűsítésekkel történik. A bemeneti rétegen nincs aktiváció és nem tartozik bias a neuronokhoz, valamint a háló csak 3 rétegből áll minimális neuron számmal illetve adat mennyiséggel.*




**2.3. ábra.** Feed forward módszer minta

A 2.1. egyenletet mátrixokkal átírva a következő képen alakul.

$$\mathbf{s} = \mathbf{x} \cdot \mathbf{W}, \tag{2.3}$$

A 2.3. egyenletet a bemeneti és a rejtett réteg közé felírva:

$$\mathbf{X} = \mathbf{s}^{(1)} = \mathbf{a}^{(1)}, \tag{2.4}$$

ahol:

- $\mathbf{X}$ a bemeneti mátrix (N x m dimenziós).
- $\mathbf{W}^{(1)}$ a súly mátrix (m x k dimenziós).
- N a tanító minták száma (a példában 4).
- m a bemeneti réteg neuronjainak száma (a példában 2).
- k a rejtett réteg neuronjainak száma (a példában 3).
- $w_{\text{ij}}^{(1)}$a $\mathbf{W}^{(1)}$ súly mátrix i-edik neuronjának és a rejtett réteg j-edik neuronját összekötő él súlyát jelöli.

Behelyettesítve az előző egyenletbe, megkaphatjuk az $\mathbf{s}^{(2)}$ paraméteres alakját.

$$\mathbf{s}^{(2)} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \\ x_1^{(3)} & x_2^{(3)} \\ x_1^{(4)} & x_2^{(4)} \end{bmatrix} \cdot \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{bmatrix} =$$

$$= \begin{bmatrix} x_1^{(1)} \cdot w_{11}^{(1)} + x_2^{(1)} \cdot w_{21}^{(1)} & x_1^{(1)} \cdot w_{12}^{(1)} + x_2^{(1)} \cdot w_{22}^{(1)} & x_1^{(1)} \cdot w_{13}^{(1)} + x_2^{(1)} \cdot w_{23}^{(1)} \\ x_1^{(2)} \cdot w_{11}^{(1)} + x_2^{(2)} \cdot w_{21}^{(1)} & x_1^{(2)} \cdot w_{12}^{(1)} + x_2^{(2)} \cdot w_{22}^{(1)} & x_1^{(2)} \cdot w_{13}^{(1)} + x_2^{(2)} \cdot w_{23}^{(1)} \\ x_1^{(3)} \cdot w_{11}^{(1)} + x_2^{(3)} \cdot w_{21}^{(1)} & x_1^{(3)} \cdot w_{12}^{(1)} + x_2^{(3)} \cdot w_{22}^{(1)} & x_1^{(3)} \cdot w_{13}^{(1)} + x_2^{(3)} \cdot w_{23}^{(1)} \\ x_1^{(4)} \cdot w_{11}^{(1)} + x_2^{(4)} \cdot w_{21}^{(1)} & x_1^{(4)} \cdot w_{12}^{(1)} + x_2^{(4)} \cdot w_{22}^{(1)} & x_1^{(4)} \cdot w_{13}^{(1)} + x_2^{(4)} \cdot w_{23}^{(1)} \end{bmatrix} \tag{2.5}$$

Innen látható, hogy az $\mathbf{s}^{(2)}$ egy N x k (4 x 3) dimenziós mátrix. Amelyből rejtett réteg kimenete az $\mathbf{a}^{(2)}$ megkapható a 2.2 egyenlet alapján.

$$\mathbf{a}^{(2)} = f_{act}(\mathbf{s}^{(2)}) = \begin{bmatrix} a_{11}^{(2)} & a_{12}^{(2)} & a_{13}^{(2)} \\ a_{21}^{(2)} & a_{22}^{(2)} & a_{23}^{(2)} \\ a_{31}^{(2)} & a_{32}^{(2)} & a_{33}^{(2)} \\ a_{41}^{(2)} & a_{42}^{(2)} & a_{43}^{(2)} \end{bmatrix}, \tag{2.6}$$

ahol az $a_{ij}^{(2)}$ megegyezik az $f_{act}(s_{ij}^{(2)})$. Folytatva a számolást eljutunk a kimeneti réteghez:

$$\mathbf{s}^{(3)} = \begin{bmatrix} a_{11}^{(2)} & a_{12}^{(2)} & a_{13}^{(2)} \\ a_{21}^{(2)} & a_{22}^{(2)} & a_{23}^{(2)} \\ a_{31}^{(2)} & a_{32}^{(2)} & a_{33}^{(2)} \\ a_{41}^{(2)} & a_{42}^{(2)} & a_{43}^{(2)} \end{bmatrix} \cdot \begin{bmatrix} w_{11}^{(2)} \\ w_{21}^{(2)} \\ w_{31}^{(2)} \end{bmatrix} = \begin{bmatrix} a_{11}^{(2)} \cdot w_{11}^{(2)} + a_{12}^{(2)} \cdot w_{21}^{(2)} + a_{13}^{(2)} \cdot w_{31}^{(2)} \\ a_{21}^{(2)} \cdot w_{11}^{(2)} + a_{22}^{(2)} \cdot w_{21}^{(2)} + a_{23}^{(2)} \cdot w_{31}^{(2)} \\ a_{31}^{(2)} \cdot w_{11}^{(2)} + a_{32}^{(2)} \cdot w_{21}^{(2)} + a_{33}^{(2)} \cdot w_{31}^{(2)} \\ a_{41}^{(2)} \cdot w_{11}^{(2)} + a_{42}^{(2)} \cdot w_{21}^{(2)} + a_{43}^{(2)} \cdot w_{31}^{(2)} \end{bmatrix} \tag{2.7}$$

A kimeneti réteg értékeihez hozzájutunk, ha hattatjuk az $f_{act}$ aktivációs függvényt.

$$\mathbf{a}^{(3)} = f_{act}(\mathbf{s}^{(3)}) = \begin{bmatrix} a_{11}^{(3)} \\ a_{21}^{(3)} \\ a_{31}^{(3)} \\ a_{41}^{(3)} \end{bmatrix} = \hat{y}m \tag{2.8}$$

, ahol az $a_{ij}^{(3)}$ megegyezik az $f_{act}(s_{ij}^{(3)})$. Ezzel eljutottunk a hálózat bemeneti értékekhez tartozó predikciójához.

### 2.1.2. Backpropagation

A gépi tanulásban, a backpropagation széles körben használt algoritmus a FFNN-k (feedforward neural network) supervised tanítására. Viszont egyéb ANN-ekhez és függvényekhez is van generalizált backpropagation. Az algoritmus kiszámolja a cost function gradiensét a hálózat súlyaihoz viszonyítva az egyetlen bemeneti-kimeneti példánál. Ennek hatékonysága, a korábban felvetett súlyonként külön-külön számolt gradienshez képest tette lehetővé a többrétegű (mély) hálózatok tanítását, a súlyok frissítésével a cost function minimalizálása érdekében.

Az algoritmus a 80-as évek felé terjedt el a "Learning representations by back-propagating errors" című Rumelhart David, Hinton Geoffrey és Williams Ronald[15] publikáció után. Bár a módszert sokszor függetlenül felfedezték. Habár a módszer a 60-as évekbeli kutatásból származik, és már néhány publikációban a 70-es években is használták, ennek ellenére az akkori korban zsák utcának sorolták. Hiszen egy egyszerű XOR függvényt megtanuló hálóhoz is több rétegű struktúra szükséges, amelynek a tanítása az akkori számítógépekkel nehézkes és idő igényes volt.

Az backpropagation algoritmus a gradiens kiszámítására utal, nem pedig arra, hogy a már kiszámolt gradienst, hogy használjuk fel a súlyok frissítésére. Azt a 2.1.3. fejezetben kifejtett optimizer-ek feladata.

### 2.1.3. Optimizers

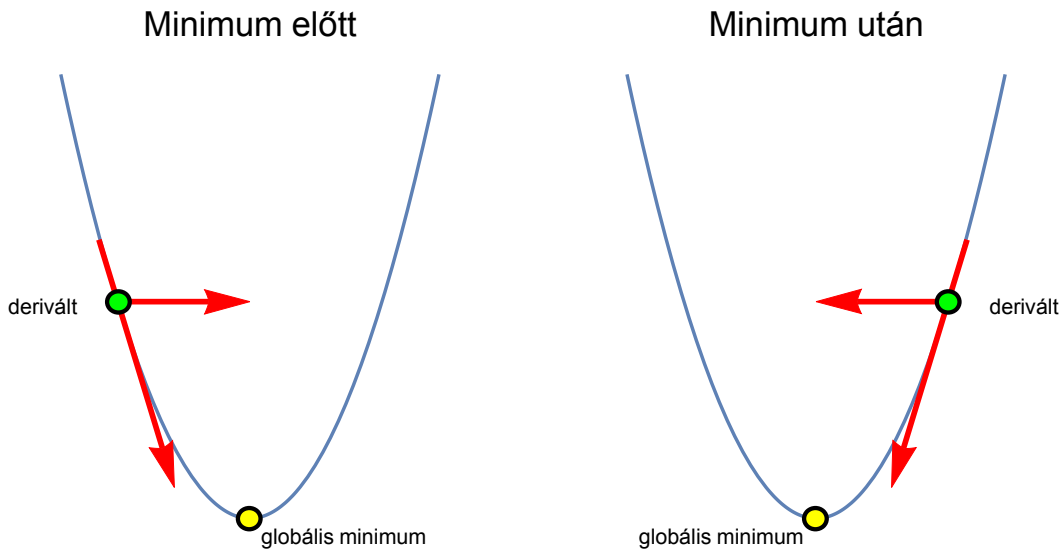
#### 2.1.3.1. Gradiens ereszkedés (Gradient descent)

A gradient descent-et eredetileg Cauchy javasolta 1847-ben. Ezzel az első rendű iteratív optimalizációs algoritmussal, a lokális minimumok megtalálhatóak. Amennyiben a lokális maximumot keressük gradient ascent-nek hívjük. Ezekben az algoritmusokban a szélsőérték megtalálásához a gradienssel vagy a gradiens közelítő értékével negatívan arányos lépést tesz.

A 2.4. ábrán amikor a függvény érték nagyobb a minimumnál és a derivált negatív, ha kivonjuk $\Theta_0$-ból az érték növekedni fog és emiatt közelebb kerülünk a minimumhoz. Ugyan így, amennyiben a derivált pozitív az érték csökkenni fog emiatt szintén közelebb kerülünk a minimumhoz.

A 2.3. ábrán bemutatott hálón egy gradiens ereszkedés, amennyiben a költség függvény a négyzetes költség felhasználva 2.3 és 2.6. egyenletet a költség értéke a következő:

$$C = \sum \left[ \frac{1}{2} \left( \mathbf{Y} - f_{\text{act}} \left( f_{\text{act}} \left( \mathbf{X} \cdot \mathbf{W}^{(1)} \right) \cdot \mathbf{W}^{(2)} \right) \right)^2 \right] \tag{2.9}$$




**2.4. ábra.** Gradiens ereszkedés elve

Felhasználva a lánc szabályt és a deriválás azonosságait:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial z} \tag{2.10}$$

$$\frac{d(u+v)}{\mathrm{dx}} = \frac{\mathrm{du}}{\mathrm{dx}} + \frac{\mathrm{dv}}{\mathrm{dx}} \tag{2.11}$$

A kimeneti réteg és a rejtett réteg közötti súly mátrix $\mathbf{W^{(2)}}$ gradiense a következő:

$$\begin{aligned} \frac{\partial C}{\partial \mathbf{W}^{(2)}} &= \frac{\partial \sum \left[\frac{1}{2}\left(\mathbf{Y}-\hat{\mathbf{Y}}\right)^2\right]}{\partial \mathbf{W}^{(2)}} = \sum \frac{\partial \frac{1}{2}\left(\mathbf{Y}-\hat{\mathbf{Y}}\right)^2}{\partial \mathbf{W}^{(2)}} = \\ &= -\left(\mathbf{Y}-\hat{\mathbf{Y}}\right) \cdot \frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{W}^{(2)}} = -\left(Y-\hat{\mathbf{Y}}\right) \cdot \frac{\partial \hat{Y}}{\partial s^{(3)}} \cdot \frac{\partial s^{(3)}}{\partial \mathbf{W}^{(2)}} = \\ &= -\left(\mathbf{Y}-\hat{\mathbf{Y}}\right) \cdot f'_{\mathrm{act}} s^{(3)} \cdot \frac{\partial a^{(2)} \cdot \partial \mathbf{W}^{(2)}}{\partial \mathbf{W}^{(2)}} = \\ &= -\left(\mathbf{Y}-\hat{Y\mathbf{Y}}\right) \cdot a^{(2)} \cdot f'_{\mathrm{act}}\left(s^{(3)}\right) = \delta^{(3)} \cdot a^{(2)} \end{aligned} \tag{2.12}$$

Tehát a számítás 4 darab tanuló adatra mátrixos formában a következő:

$$\frac{\partial C}{\partial \mathbf{W}^{(2)}} = \frac{\partial \sum \left[\frac{1}{2}\left(\mathbf{Y}-\hat{\mathbf{Y}}\right)^2\right]}{\partial \mathbf{W}^{(2)}} = \left(a^{(2)}\right)^T \cdot \delta^{(3)} = \tag{2.13}$$

$$= \begin{bmatrix} a_{11}^{(2)} & a_{21}^{(2)} & a_{31}^{(2)} & a_{41}^{(2)} \\ a_{12}^{(2)} & a_{22}^{(2)} & a_{32}^{(2)} & a_{42}^{(2)} \\ a_{13}^{(2)} & a_{23}^{(2)} & a_{33}^{(2)} & a_{43}^{(2)} \end{bmatrix} \cdot \begin{bmatrix} \delta_1^{(3)} \\ \delta_2^{(3)} \\ \delta_3^{(3)} \\ \delta_4^{(3)} \end{bmatrix} = \begin{bmatrix} a_{11}^{(2)} \cdot \delta_1^{(3)} + a_{21}^{(2)} \cdot \delta_2^{(3)} + a_{31}^{(2)} \cdot \delta_3^{(3)} + a_{41}^{(2)} \cdot \delta_4^{(3)} \\ a_{12}^{(2)} \cdot \delta_1^{(3)} + a_{22}^{(2)} \cdot \delta_2^{(3)} + a_{32}^{(2)} \cdot \delta_3^{(3)} + a_{42}^{(2)} \cdot \delta_4^{(3)} \\ a_{13}^{(2)} \cdot \delta_1^{(3)} + a_{23}^{(2)} \cdot \delta_2^{(3)} + a_{33}^{(2)} \cdot \delta_3^{(3)} + a_{42}^{(2)} \cdot \delta_4^{(3)} \end{bmatrix}$$

Ugyanígy folytatva a bemeneti réteg és a rejtett réteg közötti súly mátrix $W^{(1)}$ gradiense a következő:

$$\begin{aligned}
\frac{\partial C}{\partial W^{(1)}} = \frac{\partial \sum \left[\frac{1}{2}\left(Y-\hat{Y}\right)^2\right]}{\partial W^{(1)}} = \sum \frac{\partial \frac{1}{2}\left(Y-\hat{Y}\right)^2}{\partial W^{(1)}} = \\
= -\left(Y-\hat{Y}\right)\cdot\frac{\partial \hat{Y}}{\partial W^{(1)}} = -\left(Y-\hat{Y}\right)\cdot\frac{\partial \hat{Y}}{\partial s^{(3)}}\cdot\frac{\partial s^{(3)}}{\partial W^{(1)}} = \\
= -\left(Y-\hat{Y}\right)\cdot f'_{\text{act}}s^{(3)}\cdot\frac{\partial s^{(3)}}{\partial W^{(1)}} = \delta^{(3)}\cdot\frac{\partial s^{(3)}}{\partial W^{(1)}} = \delta^{(3)}\cdot\frac{\partial s^{(3)}}{\partial a^{(2)}}\cdot\frac{\partial a^{(2)}}{\partial W^{(1)}} = \\
= \delta^{(3)}\cdot\left(W^{(2)}\right)^T\cdot\frac{\partial a^{(2)}}{\partial s^{(2)}}\cdot\frac{\partial s^{(2)}}{\partial W^{(1)}} = \delta^{(3)}\cdot\left(W^{(2)}\right)^T\cdot f'_{\text{act}}s^{(2)}\cdot\frac{\partial s^{(2)}}{\partial W^{(1)}} = \\
= X^{\mathrm{T}}\cdot\delta^{(2)}
\end{aligned} \tag{2.14}$$

Tehát a számítás 4 darab tanuló adatra mátrixos formában a következő:

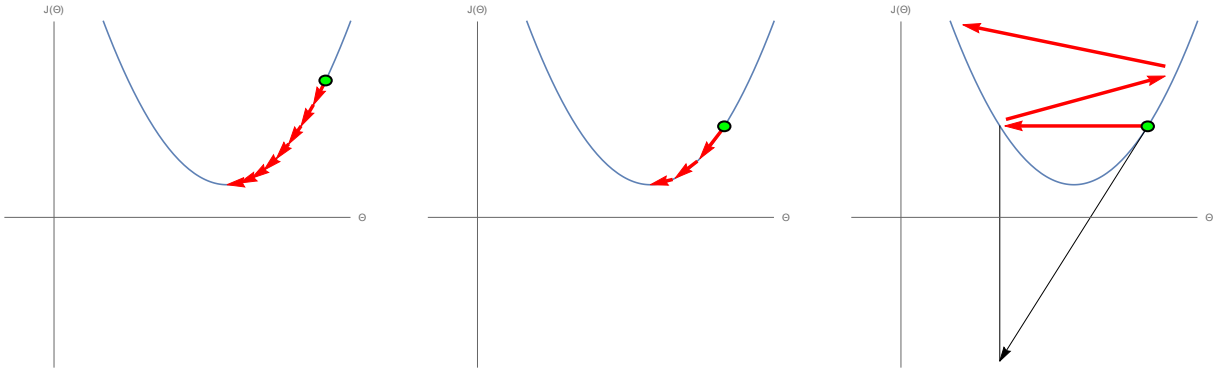
$$\begin{aligned}
\frac{\partial C}{\partial W^{(1)}} = \frac{\partial \sum \left[\frac{1}{2}\left(Y-\hat{Y}\right)^2\right]}{\partial W^{(1)}} = X^T\cdot\delta^{(2)} = \\
= \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} & x_1^{(4)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} & x_2^{(4)} \end{bmatrix}\cdot\begin{bmatrix} \delta_{11}^{(2)} & \delta_{12}^{(2)} & \delta_{13}^{(2)} \\ \delta_{21}^{(2)} & \delta_{22}^{(2)} & \delta_{23}^{(2)} \\ \delta_{31}^{(2)} & \delta_{32}^{(2)} & \delta_{33}^{(2)} \\ \delta_{41}^{(2)} & \delta_{42}^{(2)} & \delta_{43}^{(2)} \end{bmatrix} = \begin{bmatrix} \hat{W}_{11}^{(1)} & \hat{W}_{12}^{(1)} & \hat{W}_{13}^{(1)} \\ \hat{W}_{21}^{(1)} & \hat{W}_{22}^{(1)} & \hat{W}_{23}^{(1)} \end{bmatrix}
\end{aligned} \tag{2.15}$$

A már kiszámolt súlymátrixok gradiensével arányosan módosítjuk a mátrixokat, a következő módon:

$$W^{(1)*} = W^{(1)} - \mu\frac{\partial C}{\partial W^{(1)}} \tag{2.16}$$

$$W^{(2)*} = W^{(2)} - \mu\frac{\partial C}{\partial W^{(2)}} \tag{2.17}$$

Ahol a $\mu$ a tanulási ráta paraméter és a $W^{(\mathrm{i})*}$ a következő iteráció i-edik súlymátrixa. A $\mu$ paraméter megválasztása kifejezetten fontos, ugyanis ahogy a 2.5. ábrán látható a túl alacsonyra választott érték esetén a konvergencia nagyon lassan következik be. Míg a túl nagy tanulási ráta akár arra is képes, hogy a rendszerbe divergenciát vigyen be.
Ezeknek elkerüléséhez decaying tanulási rátát alkalmazunk, amely a kezdeti nagy értéke miatt gyorsan konvergál a szélsőértékbe, majd az iterációk során folytonos csökkenéssel a túllövést és a divergenciát elkerüli.




**(a)** Túl alacsony tanulási ráta **(b)** Optimális tanulási ráta **(c)** Túl nagy tanulási ráta

**2.5. ábra.** Tanulási ráta különböző értékei

A már korábban felmerült problémákon kívül észrevehető a gradiens ereszkedés egyik problémája. Ez a módszer nem képes a globális és a lokális szélső értékek megkülönböztetésére. Ennek a problémának a megoldására különböző módszereket használunk. Az egyik ilyen megoldás a Momentum módszer, ahol az előző értékek alapján inerciája is van az iterációnak. Ezáltal megfelelő paraméter választással gyakran lehet a lokális szélsőértékből kimozdítani a globális szélsőérték felé.

#### 2.1.3.2. Adam

Az Adam optimizer a jelenleg leggyakrabban használt optimalizálási algoritmus. A sztochasztikus gradient descent egy speciális megvalósítása. Az algoritmust Diederik Kingma és Jimmy Ba prezentálta az Adam: A Method for Stochastic Oprimization[9] című cikkükben.
Az algoritmus a következő előnyökkel rendelkezik:

- Egyszerű megvalósítás.
- Számítási szempontból hatékony.
- Alacsony memória igény.
- A gradiens diagonális átméretezésére invariáns.
- Alkalmas a nagyon zajos/ vagy ritka gradienssel rendelkező problémákhoz.
- A hiperparaméterek értelmezése intuitív, és általában kevés hangolást igényel.

Az Adam algoritmus a következő hiperparaméterekkel rendelkezik:

- **alpha**: A tanulási ráta, általános értéke 0.001.
- **beta1**: Az exponenciális decay rátája az első becslésnek, általános értéke 0.9.
- **beta2**: Az exponenciális decay rátája a második becslésnek, általános értéke 0.999.
- **epsilon**: A nullával való osztás elkerülésére egy alacsony szám, általában $10^{-8}$

### 2.1.4. Cost function

A 2.1.3. fejezetben említett optimezer-eknek szükségük van cost function-re, hogy tudják, mit kell minimalizálni a súlyok frissítésével. Ez az érték a 2.1.1. fejezetben definiált feed

forward módszer által megkapott kimeneti érték és a valós kimeneti érték közötti valamilyen szempontbeli eltérést nevezzük a hibának. Ehhez a szemponthoz szükségünk van cost functionre, mely bemenete általában a valódi és a predikált érték, vissza térési értéke pedig a hiba mértéke. Ettől az általános megoldástól bizonyos esetekben eltérnek például ha nem supervised tanítást folytatnak a nem ismert valódi érték miatt kénytelenek más bemeneti értéket adni.

A költségfüggvénynek a következő követelményeknek kell eleget tenniük:

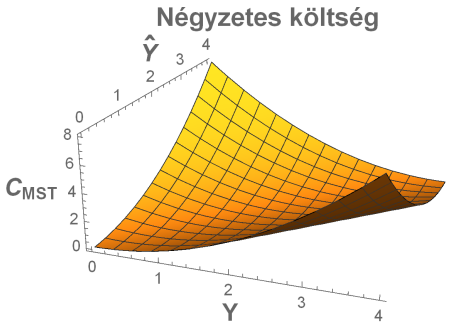
1. A C költségfüggvényt a $C_x$ költségfüggvényekhez viszonyítva, ahol x az egyedi tanító érték, például

   $C = \frac{\sum_n C_x}{n}$

   , különben a modell csak egy példával tanítható egyszerre.

2. A C költségfüggvény nem függhet csak az utolsó réteg kimenetétől. Ez a 2.1.2. fejezetben említett backpropagation algoritmus működéséhez szükséges.

#### 2.1.4.1. Quadratic cost




**2.6. ábra.** Quadratic cost illusztráció

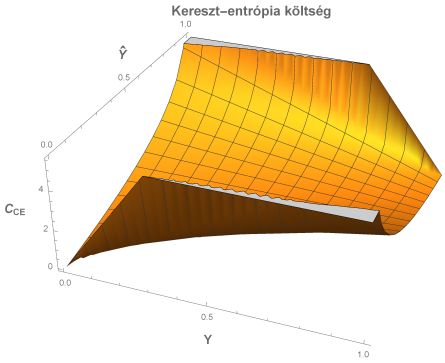
A quadratic cost magyarul négyzetes hiba és négyzet összeg hibaként is ismert, ez a következő módon van definiálva:

$$C_{\mathrm{MST}}(Y, \hat{Y}) = \frac{1}{2} \sum_n \left(Y_n - \hat{Y}_n\right)^2 \tag{2.18}$$

A költségfüggvény gradiense a neurális hálózat kimenete szempontjából:

$$\frac{\delta \mathrm{C}_{\mathrm{MST}}}{\delta \hat{Y}} = \left(Y - \hat{Y}\right) \tag{2.19}$$

#### 2.1.4.2. Cross-entropy cost/ Binarz cross-entropy cost




**2.7. ábra.** Kereszt-entrópia költség illusztráció

Más néven Bernoulli negatív log-valószínűség és Bináris Kereszt-Entrópia is ismert

$$C_{\mathrm{CE}}(Y,\hat{Y}) = -\sum_{n}\left[\ln\left(1-Y_n\right)\left(1-\hat{Y}_n\right)+\ln Y_n\hat{Y}_n\right] \tag{2.20}$$

A költségfüggvény gradiense a neurális hálózat kimenete szempontjából:

$$\frac{\delta \mathrm{C}_{\mathrm{CE}}}{\delta\hat{Y}} = \frac{\hat{Y}-Y}{\left(1-\hat{Y}_n\right)\hat{Y}_n} \tag{2.21}$$

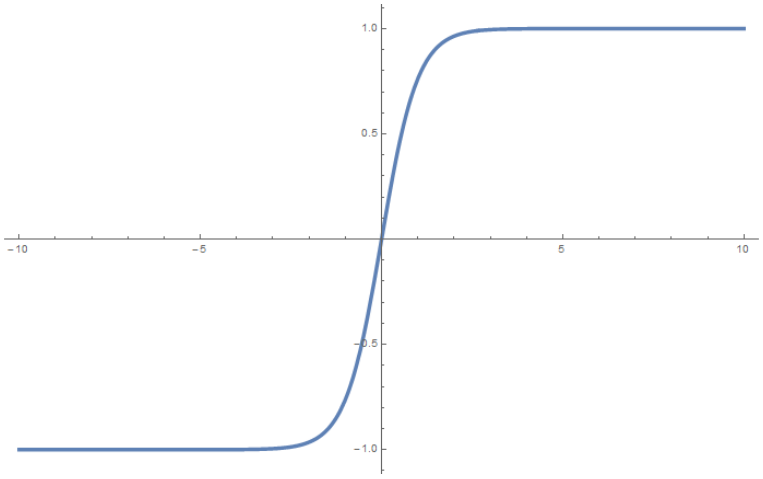
### 2.1.5. Aktivációs függvények

Aktivációs függvények összehasonlítása:

- Non-linearitás (nonlinear) - Amikor az aktivációs függvény nem lineáris, a kettő rétegű neurális hálózat bebizonyítottan universal function approximator [13]. Amennyiben ez a tulajdonság nem teljesül a két réteg között azok össze vonhatóak.
- Érték készlet (range) - Amikor az aktivációs függvény érték készlete véges a gradiens alapú tanító metódusok általában stabilabbak, mivel a minta megjelenések csak korlátozott súlyokat befolyásolnak. Amikor az érték készlet végtelen, a tanulás általánosan hatékonyabb, mivel a minta megjelenés nagy mértékben befolyásolja a legtöbb súlyt. Emiatt általában kisebb tanulási ráta szükséges.
- Folyamatosan differenciálható (continously differentiable) - Ez a tulajdonság kívánatos a gradiens alapú optimalizációs metódusok lehetővé tételéhez.
- Monoton (monotonic) - Amikor az aktivációs függvény monoton, az egyrétegű modellhez kapcsolódó hibafelület garantáltan konvex.
- Megközelíti az identitást a származási hely közelében - Amikor az aktivációs függvénynek ezen tulajdonsága teljesül, a neurális hálózat hatékonyan megtanulja, amennyiben ez megtanulható, a bemenet és a kimenet kapcsolatát, amikor a háló súlyai kicsi véletlen számokkal vannak inicializálva. Ezzel ha a tulajdonság nem teljesül külön figyelmet kell fordítani a súlyok inicializálására.

#### 2.1.5.1. Tanh

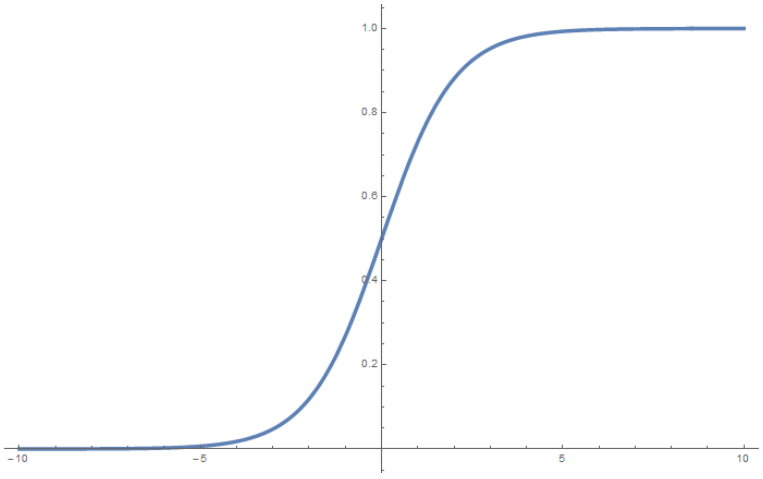
$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.22}$$




**2.8. ábra.** Tanh függvény

#### 2.1.5.2. Sigmoid

$$f(x) = \frac{1}{e^{-x} + 1} \tag{2.23}$$



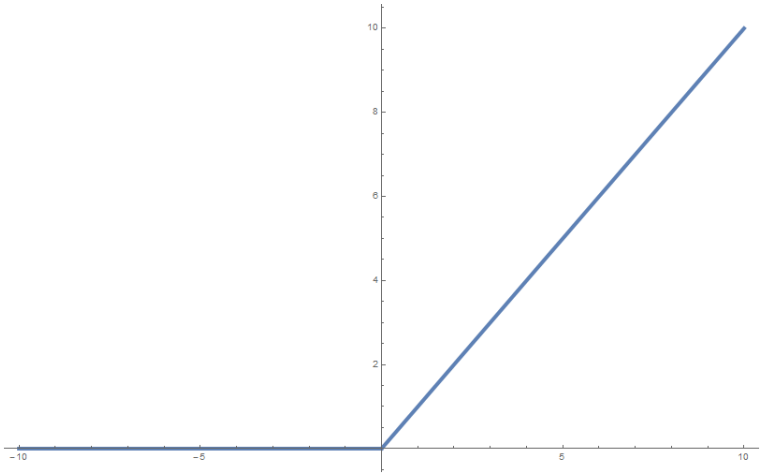

**2.9. ábra.** Sigmoid függvény

#### 2.1.5.3. Egyenirányító (Rectifier)

Az egyenirányító aktivációs függvény a független változó pozitív értékeként van definiálva:

$$f(x) = x^+ = \max(0, x) \tag{2.24}$$

Ahol $x$ a neuron bemeneti értéke.




**2.10. ábra.** Rectifier Linear Unit

Előnyös volt a korábban használt aktivációs függvényekkel szemben, hogy biológia szempontból elfogadható az egyirányúságából fakadóan. Ritka aktiválás hiszen a véletlenszerűen inicializált értékek közül csak 50% aktiválódik (nem 0 kimenet). Hatékony számítás mivel csak összehasonlítás, összeadás és szorzásból áll valamint léptékinvariáns hiszen $\max(0, ax) = a \cdot \max(0, x)$ minden $a \geq 0$ esetén.

Valamint a korábbi leggyakoribb sigmoid aktivációs függvényhez képest kevesebb vanishing gradiens probléma fordul elő.

#### 2.1.5.4. Softmax

A softmax függvény $\sigma : \mathbb{R}^K \to \mathbb{R}^K$ hozzá rendelés amely a következő módon van definiálva

$$\sigma(\mathbf{z})_i = \frac{e^{Z_i}}{\sum_{j=1}^{K} e^{Z_j}}, \quad \text{ahol} \quad i = 1, ..., K \quad \text{és} \quad \mathbf{z} = (z_1, ..., z_K) \in \mathbb{R} \tag{2.25}$$

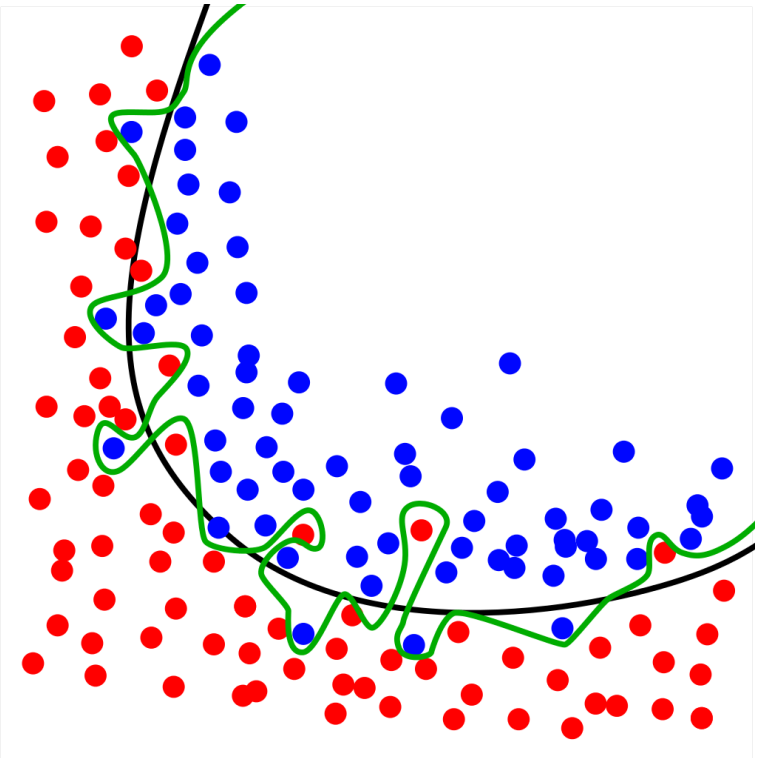
Azaz a softmax függvény bemenete K valós szám, és azt normalizálja egy valószínűségeloszlásba, amely a bemenet számok exponenciájával arányos K valószínűségből áll. Tehát a softmax előtt a bemenetek lehetnek negatívak, nagyobbak mint egy és az összegüknek sem kell egynek lennie. Viszont a függvény aktiválása után minden komponens a $(0, 1)$ intervallumban lesz és az összegük egy. Emiatt, ha jól használják, azaz a klasszifikációs osztályok egymást kizárják (nem lehet több osztály tagja) a függvény visszaadja a bemenet egyes osztályaiba való tartozás valószínűségét. A softmax függvény különböző többosztályú osztályozási (multiclass classification) módszerben használják például a multinomális logisztikus regresszióban[3].

### 2.1.6. Regularization

A regularizációnak a fő célja, hogy a háló túl-tanulását, optimális esetben megakadályozza, de legalábbis csökkentse annak hatását. Ahogy azt Neumann János mondta:

*With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.*

Azaz a túl-tanulás vagyis az overfitting abból a problémából származik, hogy bármilyen komplex adatszett megfelelően nagy számú paraméter esetén végtelenül közelíthető, viszont ezen esetben az adatok kapcsolata, helyett az adatokat tanulja meg a háló, ezáltal az eredeti feladatra használhatatlan lesz. Hiszen habár a háló a tanító adatokat megtanulta, és emiatt gyengül a prediktív képessége.



**2.11. ábra.** Túl tanulás illusztráció[1]

A regularizáció a regresszió egy formája, amely korlátozza vagy nullára csökkenti az együttható becsléseit. A lineáris regresszió a következő módon nézz ki.

$$\hat{y} \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p \tag{2.26}$$

Ahol $Y$ reprezentálja a tanult kapcsolatot míg a $\beta$ reprezentálja a becsült együtthatóit a különböző változóknak és előrejelzőknek ($X$). Az eddig cost function-nek hívott függvényt az $Error()$ függvénnyel jelezzük, ami miatt az új költség függvényünk a következően néz kiregularizáció hiányában. Az együtthatók úgy vannak kiválasztva, hogy minimalizálják a költség függvényünket.

$$\text{Loss} = \text{Error}\,(y, \hat{y}) \tag{2.27}$$

Amennyiben a tanító adatkészlet zajjal van szennyezve, a becsült együtthatókkal a jövőbeli adatok nem predikálhatóak. A regularizáció lényege ezen esetekben látható, hiszen a célja, hogy a predikciót rontó becsléseket nullára csökkenti, azaz regularizálja.

[1] `https://en.wikipedia.org/wiki/File:Overfitting.svg`

A leggyakrabban használt regularizációk az L1 és az L2, melyek neve a vektor normáiból jön. A vektor normák a következő módon számíthatóak

$$||W||_1 = |w_1| + |w_2| + \ldots + |w_N| \tag{2.28}$$

$$||W||_2 = (w_1^2 + w_2^2 + \ldots + w_N^2)^{\frac{1}{2}} \tag{2.29}$$

$$||W||_p = (w_1^p + w_2^p + \ldots + w_N^p)^{\frac{1}{p}} \tag{2.30}$$

#### 2.1.6.1. L1 regularizáció (Lasso regression)

Amennyiben a lineáris regresszió L1 normát implementál regularizációra a költség függvénye a következő módon változik:

$$\text{Loss} = \text{Error}\,(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i| \tag{2.31}$$

Az L1 regularizáció célja, hogy a becsült súlyok értékét a nulla közelében tartja. Emiatt a kialakuló súlymátrixok ritkák (sparse-ek) lesznek. Aminek az előnye, hogy a tanult jellemzők közül csak az adatkészlethez legjobban szükséges jellemzők maradnak meg. A bemutatásához 2.26. egyenletet egyszerűsítem, azzal hogy csak egy független változót használok.

$$\hat{y} = \beta_0 + w \cdot x \tag{2.32}$$

Az 2.28. egyenletbe behelyettesítve ezt:

$$Loss = (b + w \cdot x - y)^2 + \lambda\,|w| \tag{2.33}$$

Ahol a $\lambda > 0$ egy manuálisan beállított paraméter. Illetve fontos megjegyezni, hogy a $|w|$ differenciálható mindenhol a $w = 0$-n kívül.

$$\frac{d\,|w|}{\text{dw}} = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \end{cases} \tag{2.34}$$

Gradiens alapú optimalizációs algoritmust alkalmazzuk:

$$w_{\text{new}} = w - \eta \frac{\partial \text{Loss}}{\partial w} = w - \eta \cdot \left(2 \cdot H + \lambda \cdot \frac{d\,|w|}{\text{dw}}\right) = \begin{cases} w - \eta \cdot (H + \lambda) & w > 0 \\ w - \eta \cdot (H - \lambda) & w < 0 \end{cases} \tag{2.35}$$

Ahol:

$$H = 2 \cdot x \cdot (b + w \cdot x - y) \tag{2.36}$$

Illetve egyszerűbb olvashatóság miatt legyen:

$$\eta = 1 \tag{2.37}$$

$$w_{\text{new}} = \begin{cases} (w - H) - \lambda & w > 0 \\ (w - H) + \lambda & w < 0 \end{cases} \tag{2.38}$$

Amiből megfigyelhető, hogy a $\lambda$ mindig csökkenti a $w$ értékét.

#### 2.1.6.2. L2 regularizáció (Ridge regression)

Amennyiben a lineáris regresszió L2 normát implementál regularizációra a költség függvénye a következő módon változik:

$$\text{Loss} = \text{Error}\,(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2 \tag{2.39}$$

Az L1 regularizációnál felírt számításokat kiterjesztve az L2-re:

$$w_{\text{new}} = (w - 2 \cdot \lambda \cdot w) - H \tag{2.40}$$

Amiből megfigyelhető, hogy a megfelelően választott $\lambda$ esetén a súlymátrix elemei nullához közeliek lesznek, viszont nem lesz ritka a mátrix mivel az értékek nem nullák. Hátránya, hogy nem stabil ugyanis a négyzetes tag miatt nagy lesz a hiba különbség. Emiatt az L2 regularizáció, akkor képes hatékonyan működni, hogy ha minden jellemző benne van a kimenetben és közel azonos súllyal.

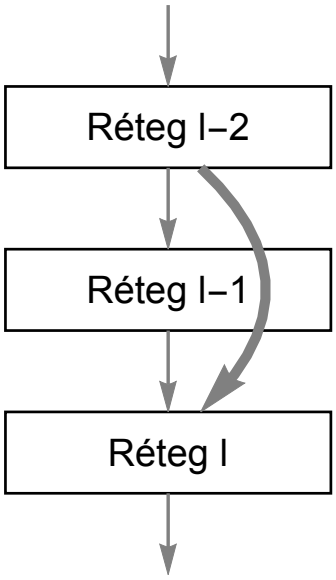
#### 2.1.6.3. Batch normalization

A batch normalization egy olyan metódus, amellyel az ANN-ek gyorsabbakká és stabilabbakká tehetőek, a bemeneti réteg normalizálásával, úgy hogy a réteget újra méretezzük és központosítjuk.

Habár a módszer hatékonysága tagadhatatlan, a hatékonyság oka viszont jelenleg is vita tárgya. Az egyik ilyen a belső kovariáns eltolódás probléma enyhítése hozza fel magyarázatnak, míg másik tudósok a függvény simítása miatt javítja a hatékonyságot.

### 2.1.7. Residual Network

A residual network az ANN azon fajtája, amelynek inspirációja az agykéregben (cerebral cortex) található piramissejtekből (pyramidal neuron) fakad. A residual network-k skip-connections vagy más néven short-cuts-okat használnak, hogy a rétegeket hagyjanak ki. Ez nem csak egy réteg kihagyására használható, sőt általában ezen hálózatok tartalmaznak dupla vagy tripla ugrást is. Hogy ennek értelme legyen a rétegek között non-lineáris aktivációs függvénynek kell lennie, valamint gyakran batch normalization is történik a rétegek között.



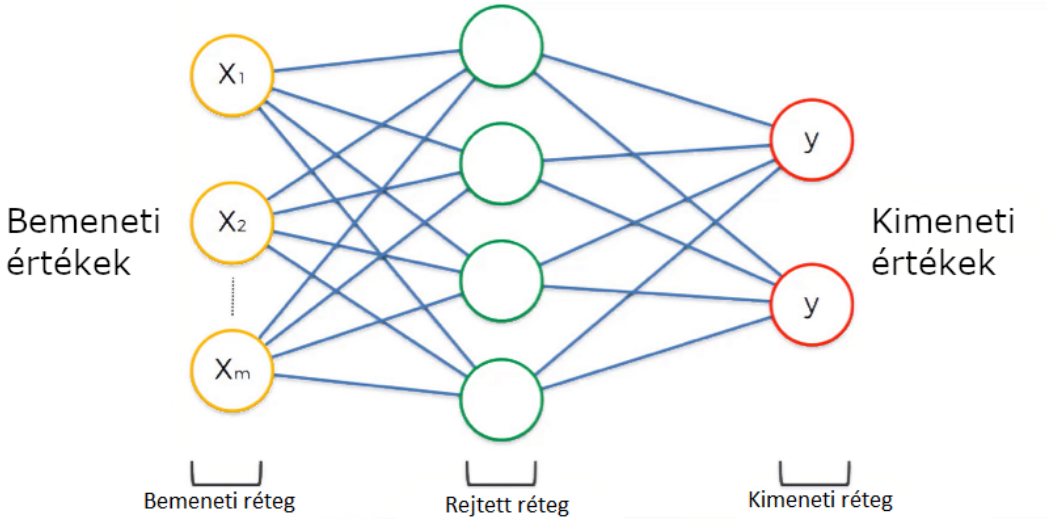

**2.12. ábra.** Residual network struktúrája

A hálózatban a réteg kihagyás egyik motivációja a vanishing gradiens probléma elkerülése. Ez a sok réteget tartalmazó hálózatokban a réteg lépések miatt történő gradiens nagyságrendekkel való csökkenését jelenti, ami miatt a mélyebb rétegek súlya szinte nem változik, ezzel a tanulást szinte teljesen blokkolja.

A problémát azzal javítja, hogy ameddig meg nem tanulja a súlyokat, addig felhasználja az előző (vagy több réteggel előtte lévő) réteg kimenetét. Azaz tanulás közben kihagy rétegeket, ezzel sekélyebbé téve a hálót, és gyorsítva a tanulást. A tanítás közben a hálózat folyamatosan vissza állítja a rétegeket, ahogy a benne lévő rétegek egyre jobban közelítenek a megoldásukhoz. A tanulás folyamán a funkció tér folyton bővül, ahogy a rétegek vissza kapcsolódnak, emiatt a skip-connection nélküli hálónál gyorsabban tanul, de könnyebben elhagyja a megoldás teret a zavarások hatására, ami miatt több adatra van szüksége.

## 2.2. Mesterséges neurális rétegek

### 2.2.1. Fully connected layer




**2.13. ábra.** Fully Connected Layer

Az *2.13. ábrán* látható egy példa a dense vagy más fully connected layer-re (FC) a hálózatban. Az ábráról látható, hogy az FC a FFNN-ek közé tartozik, ugyanis nincs kör a gráf reprezentációjában. A neuron értéke meghatározható az előtte való réteg neuronjainak lineáris kombinációjából. Ez felírható mátrixos alakban

$$\mathbf{x} \cdot \mathbf{W} = \mathbf{s}, \tag{2.41}$$

ahol az $\mathbf{x}$ egy n elemszámú vektor, ahol az értékek az előző réteg neuronjainak a kimenetele, $\mathbf{s}$ a réteg összegzett értéke, amelyen az aktivációs függvény még nem volt hattatva, és a $\mathbf{W}$ a súlymátrix n x m alakú, ahol n az előző réteg neuron száma és m a réteg neuron száma. Ebből jól látható, hogy egy sűrű rétegek esetében a két réteg neuron számának a szorzata adja meg a tanítható paraméterek számát, ez viszont veszélyes az adat túl tanítása miatt. Ez a legegyszerűbb hálózat, amelyet klasszifikációs problémákhoz még most is gyakran használnak.

### 2.2.2. Convolutional connected layer

A convolutional layers (CL) és a belőle álló convolutional neural network (CNN) az ANN-ek egyik gyakran használt osztálya. Ez a korábban említett FC rétegekhez hasonlóan eleget tesz a FFNN feltételeinek (nem tartalmaz kört a gráf reprezentációja). Leggyakrabban a kép, videó és természetes nyelv feldolgozására (NLP) használják, viszont orvosi kép analízisre, valamint idősorok és javaslati rendszerekben is megtalálható. Eltolásos invariáns (shift invariant) vagy tér invariáns (space invariant) neurális hálózatként is ismert (SIANN), a megosztott-súly szerkezet és a fordítási invariancia karakterisztika miatt. Az elnevezése az általános mátrix szorzás helyett használt matematikai műveletből a konvolúcióból alakult ki.
A matematikában a konvolúció egy olyan művelet, ami függvényeken és disztribúciókon van értelmezve. Két függvény (f és g) konvolúciója az $f * g$. Az integrálja a szorzatnak, ahol az egyik

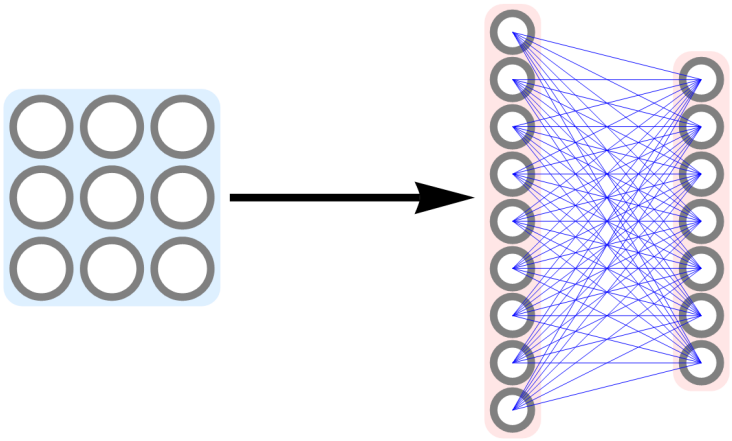
függvény eltolt és fordított:

$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(t-\tau)g(\tau)\,d\tau = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau \tag{2.42}$$

Ebből fakad, hogy a CNN-ek felhasználják a bemeneti adatban a térbeli elhelyezkedést is, ami miatt jól használhatóak, minta felismerésre. Felépítésük rétegenként egyre bonyolultabb minták felismerését teszi lehetővé.

#### 2.2.2.1. 2D Convolutional

Amikor a fully connected neural network-t képek klasszifikációjára akarták használni a hálózat hátrányai előtérbe helyeződtek.



**2.14. ábra.** Kép átalakítása Dense Neural Network bemenetté

A 2.14. ábrán megfigyelhető, az hogy a bemeneti réteg neuron száma és emiatt a háló súlyainak a száma nagy mértékben nő a kép átméretezésével (3x3 9, 4x4 16, 1024x768 786432).
Emiatt a modern felbontású képekhez a gyakorlatban használhatatlanok.

További probléma, hogy az átalakítási folyamat során elveszítjük a kép struktúra információját, hiszen a réteg neuronjai között megszűnt a térbeli helyzetük felhasználásának lehetősége. A következő neuron értékek számításakor a réteg minden neuronjának azonos hatása van, amiből következik, hogy a hálónak kell megtanulnia a térbeli elhelyezkedésből fakadó kapcsolatot is. Sőt, különbséget tesz a képen különböző helyen megjelenő tárgyak között is. Ez akár az intuíciónk alapján is beláthatóan hátrányos a képfelismerés szempontjából.

Hiszen általában a feladat nem az adott objektum felismerése a kép adott területén, hanem a kép bármely részén felkel ismerni az objektumot, illetve a hálózatnak nem csak a tanítási adatban talált specifikus objektumokat kell képesnek lennie felismerni hanem az objektumok olyan példányát is amelyek nem pontos mása, azaz asszociációra is képesnek kell lennie. Ezzel szemben a 2.14. ábrán különböző helyen elhelyezkedő objektumok eltérő súlyokon keresztül aktiválódnak a következő réteg neuronjaiban, emiatt viszont a hiba vissza terjesztésénél csak azok a súlyok változnak jó irányba, ahol éppen volt a tárgy, ami nagyságrendekkel növeli mind a tanításhoz szükséges adatkészletet, mind a hozzá szükséges időt a nagyságrendekkel nagyobb paraméter számon kívül is.

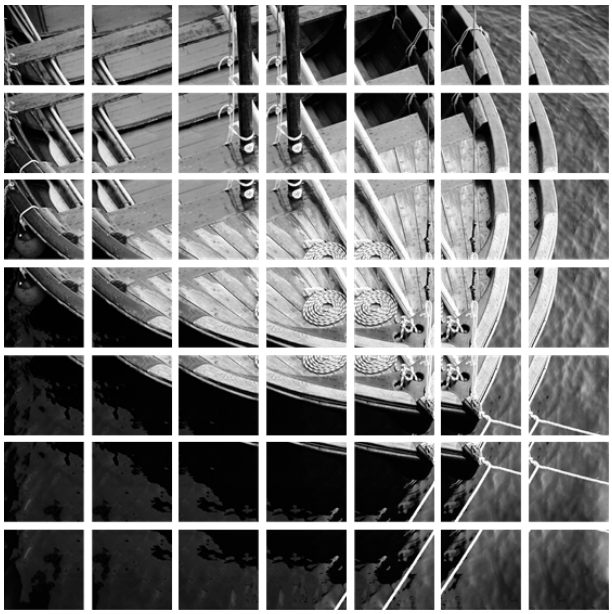
Tehát a fully connected rétegeknek három nagy hibája van:

- Normális kép méret esetén túl nagy paraméter szám.
- Elveszik a strukturális információ a bemeneti adatból (ez akár az idő tengely is lehet)
- Adatot ismer fel nem mintát

Ezen problémák által ihletve a következő megoldást alkalmazzák a convolutional layer-ek. A képet felbontjuk kisebb részekre, hiszen az adathalmaznak csak a térben egymáshoz közeli részei relevánsak számunkra. Az alrészek azonos nagyságúak és elcsúsztatva vannak a képből kivágva.

Ennek a paraméterei:

- Szűrő (filter) - Kettő dimenziós érték (2D konvolúció esetén), a felbontott alegység szélességét és magasságát tartalmazza általában $3x3$ vagy $5x5$. (Irodalomban gyakran Kernel néven is használják.)
- Lépés köz (stride) - Szintén két dimenziós érték, a csúsztatás lépését adja meg. Általában $1x1$ vagy $2x2$. A $2x2$ esetben a réteg hasonlóan viselkedik, mint a 2.2.4. szakaszban kifejtett pooling réteg.
- Szűrő mélysége/száma (filter depth) - Minden szűrő különböző jellemző felismerését képes megtanulni, emiatt általában több szűrőt teszünk egymás mögé, a különböző jellemzők megtanulását a véletlenül való inicializálás biztosítja.
- Zero padding - Kép széleit nullákkal párnázzuk ki, hogy a kép mérete más milyen módon vagy ne változzon. Hogy azonos méretű kimenetet kapjunk: $zeropadding_* = (filter_* - 1)/2$ ahol a $*$ a dimenziót jelöli (x, y).



**2.15. ábra.** Minta a képek alegységekre bontására

Ezzel a megoldással a térbeli elhelyezkedést bevezettük a hálózatunkba, hiszen a következő réteg értékét az alegységek értékeiből számolódik konvolúcióval. Viszont a súlyok száma nem

csökkent, hanem még több lett ezt egy $64x64$ méretű képen mutatom be, $3x3$ lépés közzel, illetve $1x1$ zeropadding-gal. Így az alegységek száma $64x64$ lesz.

Hiszen a kép lapítva:
$64 \cdot 64 = 4096$, míg az alegységei lapítva
$3 \cdot 3 \cdot (64 \cdot 64) = 36864$ darab bemeneti neuront jelentenének.

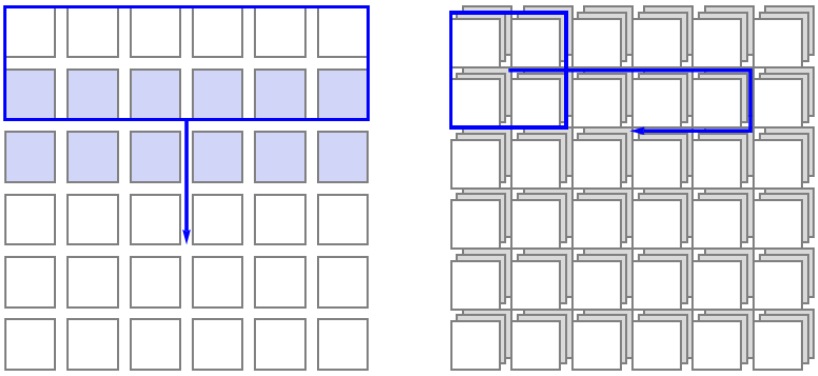
Az objektum elhelyezkedésétől függetlenül való meghatározására a következő megoldás ezt a problémát is megszünteti. Hiszen egyszerű intuícióból adódik, hogy a kép minden alegységében az azonos formákat/funkciókat keressük. Ezt úgy valósítjuk meg, hogy az alegységek szűrői között a súlyokat azonosak. Így a bemenetünk csak $3 \cdot 3 = 9$ lenne lelapítva.
Az összes súly gradiensét szeletenként összeadva kapjuk meg a backpropagationt.

#### 2.2.2.2. 1D Convolutional

Az 1D convolutional rétegek nagy hatékonyságot mutatnak a különböző szenzorok idősorainak analízisére. Kezdetben a NLP-ben (természetes nyelv feldolgozásban) is használták, azonban ott ma már a LSTM (long short-term memory) az elterjedtebb, ugyanis a szavak közelsége nem feltétlenül jó indikátor a tanulható mintára.

Különbségek az 1D és a 2D CNN között:

A konvolúciós hálózatok ugyanazt a karakterisztikával és megközelítéssel rendelkeznek a dimenziójuktól függetlenül (1D, 2D, 3D). A lényeges különbség a bemenet adatának dimenziója és a szűrő bejárása az adatnak, ami jól látható a 2.16. ábrán.



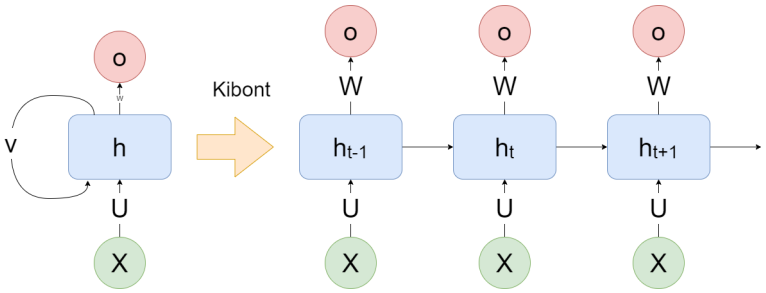
**(a)** 1D convolution **(b)** 2D convolution

**2.16. ábra.** 1D 2D CNN szűrő bejárás

Ezen kívül, megjegyzendő különbség, hogy 1D convolutional rétegeknél lehet nagyobb szűrő méretett használni, hiszen amíg a 2D convolutional rétegben a szűrő és az általa generált paraméterek száma négyzetes, addig itt csak lineáris. Ezáltal a nagyobb ablak méret nem okoz csak lineáris növekedést a paraméter számban.

### 2.2.3. Recurrent neural network

A recurrent neural network (RNN) az ANN azon osztálya, ahol a csomópontok közötti kapcsolat alakítja ki az irányított gráfot az időbeli szekvenciák mentén.



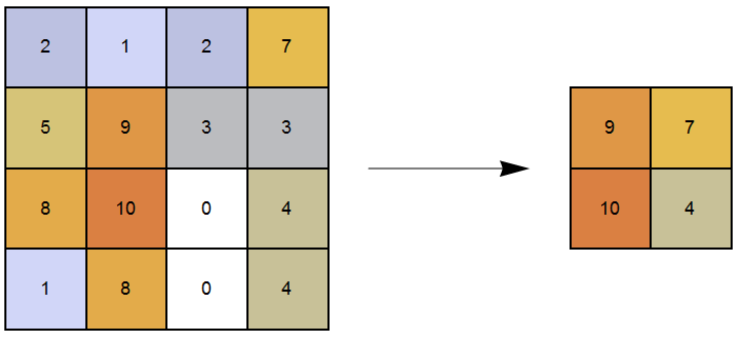

**2.17. ábra.** Alap recurrent neural network kibontása

Emiatt képes a időbeli dinamikus viselkedésre. Az FFNN-ekből leszármaztatott RNN-ek képesek a belső állapotukat (memóriájukat) arra felhasználni, hogy különböző hosszúságú bemeneteket feldolgozzanak.

### 2.2.4. Pooling layer

A pooling layer általában a konvolúciós hálózatokban van használva, ahol az adatok dimenziójának csökkentésére használják őket, oly módon, hogy egy réteg kimeneti neuron csoportját a következő réteg egy neuronjába kombinálják. Ezzel egyszerűsítve a számításokat. Az ehhez hasonló dimenzió csökkenés érhető el, hogy ha növeljük a konvolúciós réteg stride paraméterét.

- Max pooling - Az előző réteg neuron csoportjának a legnagyobb értékét rendeli a következő réteg neuronjához.
- Average pooling - Az előző neuron csoport átlag értékét rendeli a következő réteg neuronjához.




**2.18. ábra.** Max Pooling példa

# 3. fejezet

# Implementáció

## 3.1. Adat szett

A dolgozatban használt adatokhoz a PhysioNet MIT-BIH Arrhytmia és a PTB Diagnostic ECG adatbázisok szolgálnak alapul. Ezek az adatok címkézett EKG rekordok.

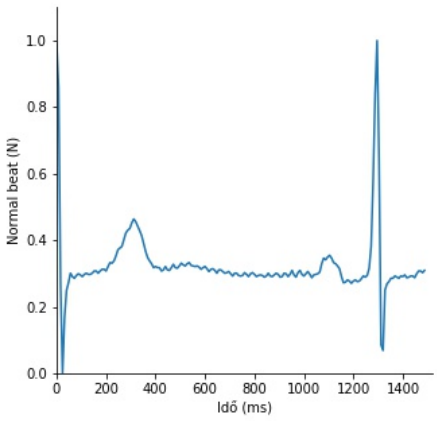
### 3.1.1. PhysioNet MIT-BIH Arrhytmia adatbázis

Ez az első általánosan elérhető szabványos vizsgálati adatkészlet a szívritmus zavar (arrhytmia) felismerés értékeléséhez. Az adatbázisban az EKG adatok forrása, a több mint 4000 hosszú távú Holter-felvételek, amelyeket a Beth Izrael Aritmia Laboratóriuma (Beth Israel Hospital Arrhytmia Laboratory) szerzett meg 1975 és 1979 között ezek közül 23 rekord lett véletlenszerűen választva és 25 úgy kiválasztva, hogy az adatbázis tartalmazzon ritka de klinikai szempontból fontos jelenségeket, amelyek a kis számú Holter-felvétel mintából nem lennének reprezentálva. Az összes rekord kicsivel hosszabb mint 30 perc.

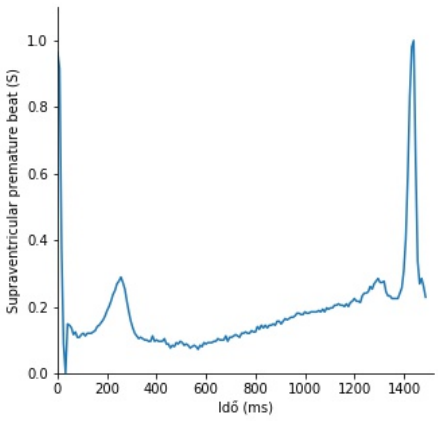
Az első csoport reprezentálja azokat a hullámformákat, amelyekkel a szívritmus zavar detektor rutin klinikai használat esetén találkozhat. Azokban az esetekben amelyekben a szakértők számára nem volt elemzésre megfelelő minőségű EKG jel elvetették. A második csoportot úgy választották ki, hogy tartalmazzanak komplex kamrai, junctionális és supraventrikuláris aritmiákat és vezetőképességi rendellenességeket is.

A mérésekben használt EKG vezeték konfiguráció (ECG lead configuration) a felső jelnél egy módosított végtagi vezeték II (MLII), melyet az elektródák mellkasra való elhelyezéséből kapunk. Míg az alsó jelnél általában egy módosított vezeték V1 (esetenként V2 vagy V5) a használt konfiguráció. Ez a BIH Aritmia Laboratórium rutin konfigurációja.

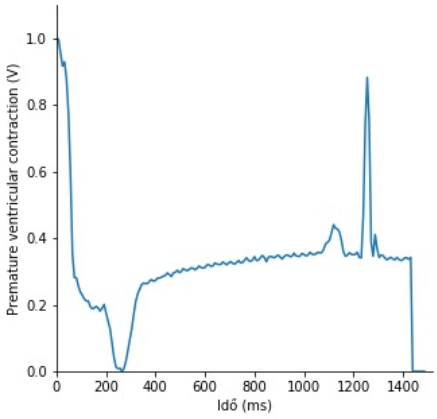
Az eredeti rekordok analóg értékek voltak, melyeket digitalizáció során szűrtek az analóg-digitális konverter (ADC) telítettségének korlátozására illetve az élsimítására. Mindehhez sávszűrőt használtak, amely a valós időhöz viszonyítva (néhány felvétel gyorsabban volt vissza játszva) a 0.1-100 Hz -es átviteli sávot használta, meghaladva a legalacsonyabb és a legmagasabb frekvenciákat, amelyek a felvételekből helyreállíthatóak. A sávszűrt jelet 360 Hz-es mintavételi frekvenciával mintavételezték, és unipoláris ADC-ket használtak, 11-bit felbontással a -5mV és a +5mV tartományon.



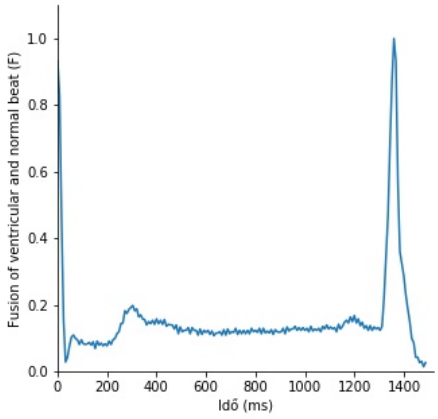

**(a)** Normal beat (N)



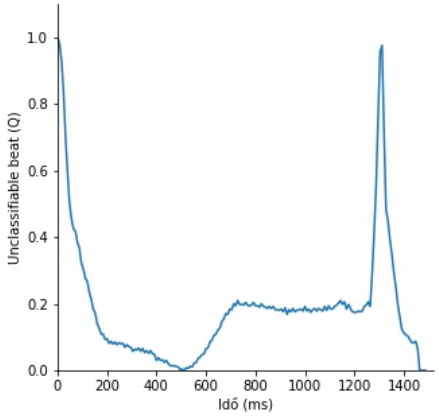

**(b)** Supraventicular premature beat (S)




**(c)** Premature Venticular contraction(V)




**(d)** Fusion beat (F)




**(e)** Unclassifiable (Q)

**3.1. ábra.** Egy-egy szívverés kép a használt csoportokból.

Az eredeti szívverés-címkézést egy egyszerű, lefutó él érzékeny QRS detektorral történt, amely minden eseményt szívverésnek érzékelt. Ezt később két kardiológus ellenőrizte, illetve javította ott, ahol hiányzott az észlelés, vagy ahol hibásan észlelt a detektor. Ezen kívül az abnormális szívveréseket is megcímkézték.

Az eredeti címkézést Acharya *és társai* [2] alapján az ANSI/AAMI EC54 2012 szabvány alkalmazásával 5 csoportba sorolom, a *3.1. ábrákon* ezen csoportokból látható egy-egy szívverés. A csoportok a következő módon alakulnak ki:

| **Csoport** | **Címke** |
|---|---|
| N | • Normális (Normal)<br>• Bal/Jobb Tawara-szárblokk (Left/Right bundle branch block)<br>• Pitvari elszabadult ütés (Atrial escape beat)<br>• Nodális elszabadult ütés (Nodal escape beat) |
| S | • Korai pitvari ütés (Atrial premature beat)<br>• Aberrált korai pitvari ütés (Aberrated atrial premature beat)<br>• Korai nodális ütés (Nodal premature beat)<br>• Korai szupraventrikuláris ütés (Supra-venticular premature beat) |
| V | • Korai kamrai ütés (Premature venticular contraction)<br>• Kamrai elszabadult ütés (Venticular escape beat) |
| F | • Kamrai és normál ütés fúziója (Fusion of venticular and normal beat) |
| Q | • Gyors ütés (Paced beat)<br>• Gyors és normál ütés fúziója (Fusion of paced and normal beat)<br>• Osztályozhatatlan (Unclassifiable) |

### 3.1.2. PTB Diagnostic EKG adatbázis

Physikalisch-Technische Bundesanstalt (PTB), a Német Nemzeti Metrológiai Intézet által felvett és digitalizált adatkészlet.

Az összegyűjtött EKG-k egy nem kereskedelmi, PTB prototípusú felvevővel készültek, a következő előírásokkal:

- 16 bemeneti csatorna (14 EKG, 1 légzés, 1 hálózati feszültség)
- Bemeneti feszültség: $\pm\,16\,mV$, a kompenzált eltolt feszültség $\pm\,300\,mV$-ig.
- Bemeneti ellenállás: $100\,\Omega$ (DC)
- Felbontás: 16 bit
- Sávszélesség: $0\,-\,1\,kHz$ (szinkron mintavételezés minden csatornára)
- Zajszint-felvétel a jelfeldolgozás alatt.

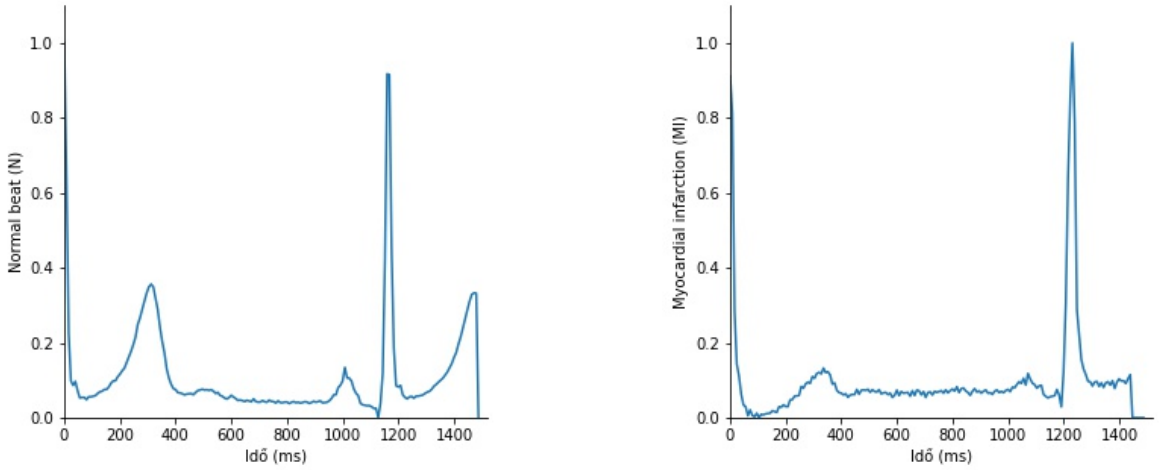
Az adatbázis 290 egyed (17-87 éves, 209 férfi, 81 nő) 590 rekordjából áll. Minden rekord tartalmaz 15 darab egyidejűleg mért jelet. A hagyományos 12 vezetéket (I, II, III, AVR, AVL, AVF, V1, V2, V3, V4, V5, V6) illetve a 3 Frank vezetékes EKG-t (VX, VY, VZ). Minden jel 1000 Hz-en van digitalizálva, 16 bites felbontásban a $\pm 16.384\,mV$ tartományban.

Az adatok különböző szív- és érrendszeri betegséggel (Cardiovascular disease) diagnosztizált páciensektől származnak.

| **Diagnosztikai osztály** | **Alanyok száma** |
|---|---|
| Szívinfarktus (Myocardial infarction) | 148 |
| Szív elégtelenség (Cardiomyopathy) | 18 |
| Tawara-szárblokk (Left bundle branch block) | 15 |
| Szívritmuszavar (Dysrhythmia) | 14 |
| Kamrai hipertrófia (Myocardial hypertrophy) | 7 |
| Szívbillentyűhibák (Valvular heart disease) | 6 |
| Szívizomgyulladás (Myocarditis) | 4 |
| Vegyes | 4 |
| Egészséges | 52 |
| Nem elérhető | 22 |

**3.1. táblázat.** PTB adatbázis adatainak csoportosítása

A feladatnak megfelelően ezekből csak az egészséges és a szívinfarktus (MI) adatokat használtam fel. Ezeknek az illusztrációja látható a *3.2. ábrán.*



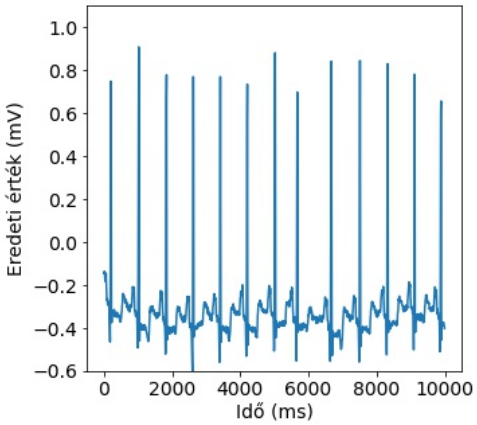

**(a)** Egészséges szívverés **(b)** Szívinfarktus

**3.2. ábra.** Egy-egy szívverés kép a használt csoportokból.

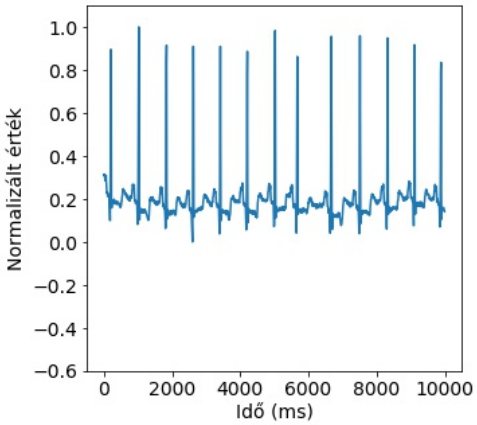
### 3.1.3. Adatok előkészítése

Szívverés kinyerése az adatokból a következő módon történik:

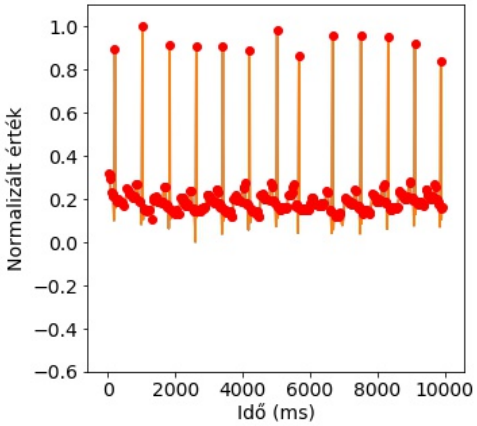
- Csatorna választás: - Hogy a csökkentsük a háló paramétereinek a számát, csak egy csatornát használunk (maximum 2-t használhatnánk hiszen van olyan adatbázisunk ahol csak 2 csatorna van). Illetve, hogy ne legyen a tanító adatok között kapcsolat a többi csatornát tanításra sem használjuk fel. A II csatornát választottuk.
- Újra mintavételezés: - A különböző gépek/adatbázisok különböző mintavételi frekvenciát alkalmaztak. Ezeket egységesíteni kell, figyelve arra, hogy a közös frekvencia kisebb legyen, mint az adatbázisokban használt legkisebb. A választott mintavételezési frekvencia a 125 Hz.
- Darabolás: - A rekordokat 10 másodperces ablakokra vágjuk, egy ilyen ablaknyi adat látszik a *3.3 a al-ábrán.*
- Normalizálás: - Az ablak amplitúdó értéket 0 és 1 közötti tartományba alakítjuk. Ezzel elkerülve az embereknél illetve gépeknél létező amplitúdó különbségeket. Ez a *3.3 b al-ábrán* megfigyelhető.
- Lokális maximum keresés: - Az ablakban található összes lokális maximum megkeresése az első derivált 0 metszésével. Ez azért fontos, mert az R hullámcsúcs mindig lokális maximummal jár. Ezek a *3.3 c al-ábrán* pirossal jelölt pontok.
- Küszöb alkalmazása (threshold): - Az EKG R hullámcsúcs jelöltek megkeresés 0.8-as küszöb alkalmazásával a normalizált lokális maximumokra. Ez a *3.3 d al-ábrán* megfigyelhető.
- Névleges szívverés: - Az R-R idő intervallumuk mediánját az ablak névleges szívverésének választjuk (R).
- Szívverés kinyerése: - Minden R-csúcshoz kiválasztunk a jelből $1.2T$ hosszú részt. ez tartalmazza a szívverést.
- Padding: - A kapott jelet adott hosszra (187 diszkrét időpillanat, ami a 125 Hz mintavételezési frekvencia miatt 1.496 másodpercet jelent.) való kiegészítése nullákkal.



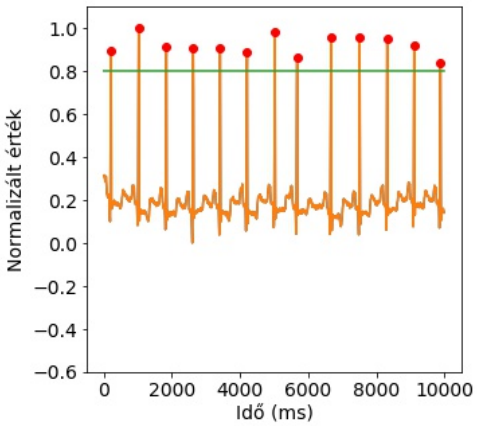

**(a)** 10 másodperces ablak



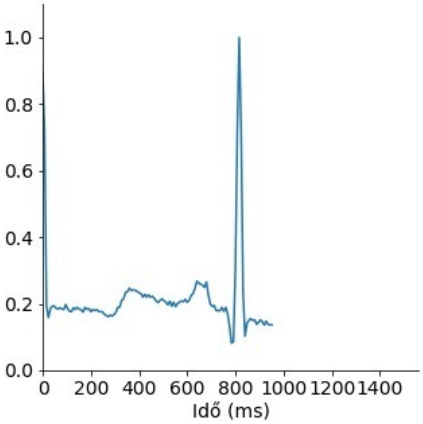

**(b)** Normalizálás



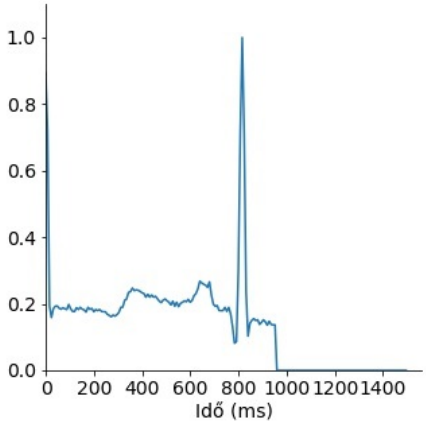

**(c)** Lokális maximum meghatározása




**(d)** Threshold alkalmazásával R csúcsok megkeresése




**(e)** Kinyert szívverés




**(f)** Párnázott szívverés

**3.3. ábra.** Az adat előkészítés illusztrációja

## 3.2. 1D convolutional architektúra

A CNN hálózat alapjául a Kachuee *és társai* [7] által javasolt hálózati architektúrát implementáltam, melyben az 1D convolutional rétegeken residual blokkokba szerveződnek. Ez a felépítés figyelhető meg a F.1.1. ábrán.

A modell két részre bontható a rétegek típusai és a szerepük szerint:

1. Konvolúciós rész, melynek szerepe az EKG jellemzők megtanulása.
2. Fully connected rész, melynek szerepe az EKG jellemzőkből az eredeti jel besorolásának predikálása.

A convolutional blokk tartalmazza a bemeneti réteget, ahol a korábban taglalt előkészített adatottal feedlejük. Ez a tömb (N, 187, 1) alakú, ahol az N a batch méret. A convolutional rétegek 32 szűrőt tartalmaznak, melyek kernel mérete 5 és lépés köze 1. Ezen rétegek bemenete az első kivételével (N, 187, 32) méretű tömbök, tehát a tanítható paraméterek (súlyok) száma rétegenként $32 \cdot 32 \cdot 5 + 32 = 5152$. Az első konvolúciós réteg bemenet pedig (N, 187, 1), azaz a súlyainak a száma $32 \cdot 1 \cdot 5 + 32 = 192$. A konvolúciós rétegek aktivációs függvénye a ReLU.

A két al-modell közül a CNN rész mélyebb, ezért a gyorsabb tanulás érdekében residual skip kapcsolat van egyes rétegei között. Ezek alapján a jelölés egyszerűsítése miatt kettő konvolúciós rétegből, egy residual skip kapcsolatból és egy pooling rétegből úgy nevezetet residual blokkokat hozunk létre. A F.1.1. ábrán látható, hogy a residual skip kapcsolat megvalósításához szükség van concatenate rétegre, mely a második tengely mentén összefűzi bemeneteit, így a kimenete egy (N, 187+187, 32) tömb. Hogy a konvolúciós rétegek mérete ne növekedjen exponenciális mértékben használnunk kell a korábban már említett max pooling-ot. A pooling réteg kernel mérete 5 és lépés köze 2, illetve zero padding-ot alkalmaz, hogy a kimenete (N, 187, 32) legyen.

A FC blokk négy réteget tartalmaz. A klasszifikáció ért felelős három FC réteg, melyekre rendre 32,32 és 5 neuronból állnak, és a CNN blokkal történő összekötéséhez szükséges flatten réteg, amely a convolutional réteg 3 dimenziós kimenetét a FC réteg által értelmezhető 2 dimenziós bemenetre alakítja át. Az első két FC réteg 32 neuronból áll és ReLU az aktivációs függvényük. Míg az utolsó réteg a csoportok számával megegyező 5 neuront tartalmaz és a többosztályú osztályozás miatt softmax az aktivációs függvénye.

Mivel a két adatbázisban nem azonos a csoportok száma, így ha átszeretnénk vinni a háló tudását a struktúráján változtatni kell. A változás annyiból áll, hogy az utolsó rétegben, amely egy FC réteg, a benne szereplő 5 neuron helyett 2 neuron lesz. Mivel a modell minden másban megegyezik ezért külön nincs illusztrációja. Ugyanezt a módszert alkalmazzuk a másik CNN modellre is.

A 3.6. alfejezetben lesz a háló tanulási paraméterei, illetve a tanítás menete részletesebben kifejtve.

## 3.3. 1D convolutional architektúra autó-enkóder

Az 3.4. alfejezetben tárgyalt hálót kiegészítjük egy autó-enkóder struktúrával ez látható a F.1.2. ábrán.

Ez csak az implementált modell CNN blokkjánra van hatással, ahol az egyes convolutional rétegek paraméterei változnak, az előző architektúrában definiált residual blokkok az eddigi kettő helyet három convolutional réteget tartalmaznak. Az első három residual blokk első convolutional rétegének a stride-ja 2. Így amennyiben a bemenetük $(N, m, k)$ méretű a kimenetük $(N, [\frac{m}{2}], k)$, ez az enkóder rész.

Az utolsó három residual blokknak (mivel 5 van ezért a középső mind két csoportba bele tartozik) max pooling rétege nincs, így a residual skip kapcsolat a valósítja meg az upsamplinget. A modell ezen része hasonlítható a dekóder részhez.

A hálózat autó-enkóder struktúrája tehát felbontható az enkóder, a dekóder és a szűk keresztmetszet. A hálózatban az enkóder rész 3 residual blokkot használ, ugyanúgy mint a dekóder rész. Ezek szinte szimmetrikusak, az eltérést a bemenet 2-vel való osztási maradékából adódik, amely az enkóder részekben lefelé van kerekítve, míg a dekóder részekben nincs. Az átfedés az enkóder és a dekóder között a szűk keresztmetszet (bottleneck) itt a konvolúciós rétegek dimenziója (N, 24, 32).

Fontos megjegyezni, hogy a modell csak strukturálisan követi az AE példáját, hiszen az alapvető esetben a felügyelet nélküli tanuláshoz alkalmazható. A javasolt modellben az AE szerepe a gyorsabb tanulás elősegítése, azzal hogy a legfontosabb jellemzők tanulódnak csak meg a szűk keresztmetszet miatt. Ezzel a túl tanulás problémáját is enyhítjük.

## 3.4. LSTM I. architektúra

Az elkészített architektúra topológiája a F.1.3. ábrán látható. Ez a mérete miatt az appendixben kapott helyet. Mivel a célom EKG osztályozás volt LSTM-et tartalmazó neurális háló segítségével, így a modell két részre bontható szét a rétegek típusa és szerepe szerint.

1. LSTM rész, melynek szerepe az EKG jellemzők megtanulása.
2. Fully connected rész, melynek szerepe az EKG jellemzőkből az eredeti jel besorolásának predikálása.

Az architektúra LSTM blokkjának kiindulási pontját Smirnov *és társai*[18] cikkben elkészített háló-összehasonlítás adta meg. Ebben a cikkben több különböző paraméterrel hasonlították össze az LSTM-ek tanulását. Az eredményeik alapján két LSTM réteget használtam, melyre megmutatták, hogy az egy rétegnél jobb eredményt ad. Az LSTM rétegeket szekvenciálisan sorba kötöttem, figyelve arra, hogy dropout és batch-normalizálás legyen a rétegek között. Valamint, habár a cikkben 128 volt az LSTM unit száma, a kisebb adatkészletre tekintettel lévén a munkámban ezt 64-re csökkentettem. Mind a két LSTM réteg visszaadja a szekvenciáját (return_sequence = true), hogy a klasszifikációs blokk az egészet értékelje, ugyanis az utolsó értékek 0-val való feltöltése könnyen elrontja azokat.

Emiatt viszont ahhoz, hogy megfelelő legyen az adat dimenziója, flatten réteget alkalmazunk a két blokk között.

A fully-connected block, a *Özal*[20] cikkében is használt 2 rétegből áll. Viszont a kisebb osztály szám (csak 2 MI és normal), valamint az alacsonyabb adatmennyiségből fakadó over-fitting veszélye miatt, csak 64 neuront alkalmaztam az első és 1 neuront a második rétegben, valamint rendre ReLU aktivációs függvényt és Sigmoid aktivációs függvényt.

## 3.5. LSTM II. architektúra

A II. LSTM modell az első kiegészítése skip-connectionnel. Ez a F.1.4. ábrán található, szintén az appendixben a mérete miatt. Itt a klasszifikációs blokk bemenetébe az első LSTM réteg kimeneti is közvetlenül megjelenik. Ezen kívül az architektúrája megegyezik az előző fejezetben tárgyalttal.

## 3.6. Tanítás és Hiperparaméterek

Az adatbázist 3 részre bontottam, train, validation és test. Az adatbázis használt adatát 100%-nak tekintve rendre 70%, 25% és 5% a részek mérete.

| **Adatbázis** | **Train** | **Valid** | **Test** |
|---|---|---|---|
| MIT-BIH | 76606 | 27360 | 5478 |
| PTB | 10184 | 3637 | 729 |

A CNN modellekben alkalmazott transfer learning, miatt az implementált modellek tanítása a következő:

1. Adatok előkészítése: - 3.1.3. szakaszban definiáltak alapján.
2. Adatok train, validation és test halmazba hasítása: - Fontos, hogy ezek a halmazok között ne legyen semmilyen áttekintés.
3. Standardizálás: - Train adat standardizálása, majd a test és a validation adatok standardizálása a train átlagával és szórásával.
4. Háló tanítása: - Szívritmuszavar adatokkal.
5. Háló átalakítás: - Utolsó FC réteg neuron számának megváltoztatása, a CNN részmodell súlyainak változtatása nélkül.
6. Súlyok fagyasztása: - A FC blokk súlyain kívül minden réteg súlyát fagyasztjuk.
7. Háló tanítás: - Szívinfarktus adatok.

Az LSTM modelleknél a tanítás hasonlít a CNN-ekéhez. A különbség, az hogy a 4-6. pontokat nem valósítjuk meg, ezáltal ezek az implementált hálózatok nem alkalmaznak transfer learninget. A dolgozat alatt a Keras modellszintű könyvtárat használtam TensorFlow háttérrendszerrel a modellek tanításra és értékelésre A tanítás a Google által ingyen biztosított Colab környezetben történt.

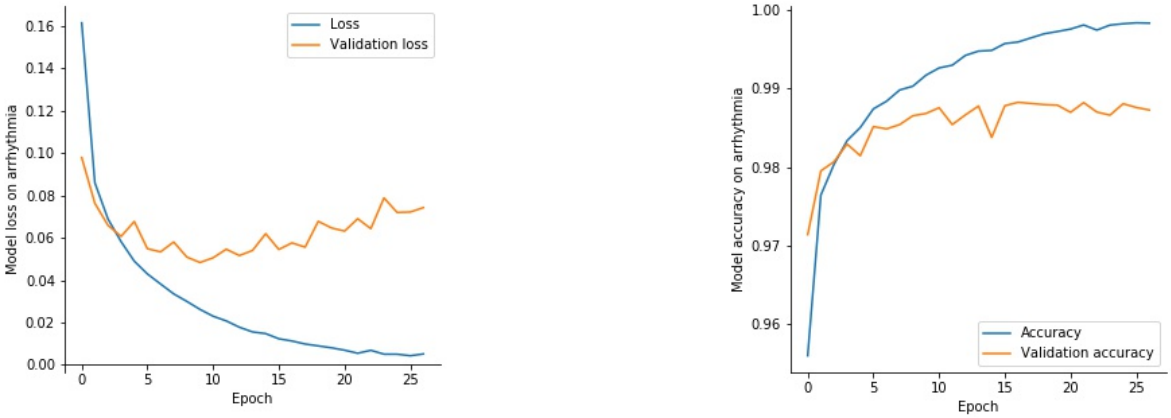
A hálózatokban a Kereszt-entrópia költség függvényt használtuk a sigmoid és a softmax kimeneteken. A túl tanulás ellen early stopping van beállítva, 10-es patience-el és a legjobb validációs pontossághoz. Optimalizáló metódusnak az Adam módszert alkalmaztam[8]. Melynek a paraméterei a 2.1.3.2. szekcióban definiált default paramétereit használtam. Minden implementált modellben a tanítás során 32 volt a batchekben az adatok száma.

Az architektúrákban az utolsó rétegek kivételével l2 regularizáció van alkalmazva 0.01 paraméterrel ez a *Lipton és társai*[12] cikkben használt érték, és a dropout paramétere 0.2 volt.

# 4. fejezet

# Eredmények

## 4.1. 1D konvolúciós neurális háló autó-enkóder nélkül

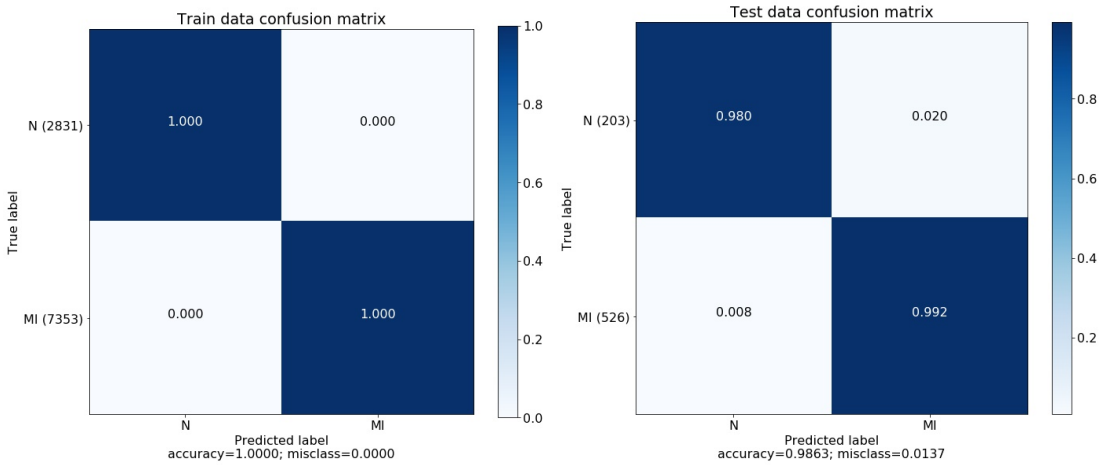



**(a)** Modell tanítás veszteség értékei

**(b)** Modell tanítás pontosság értékei

**4.1. ábra.** MI adatokon a modell tanítás tulajdonságai

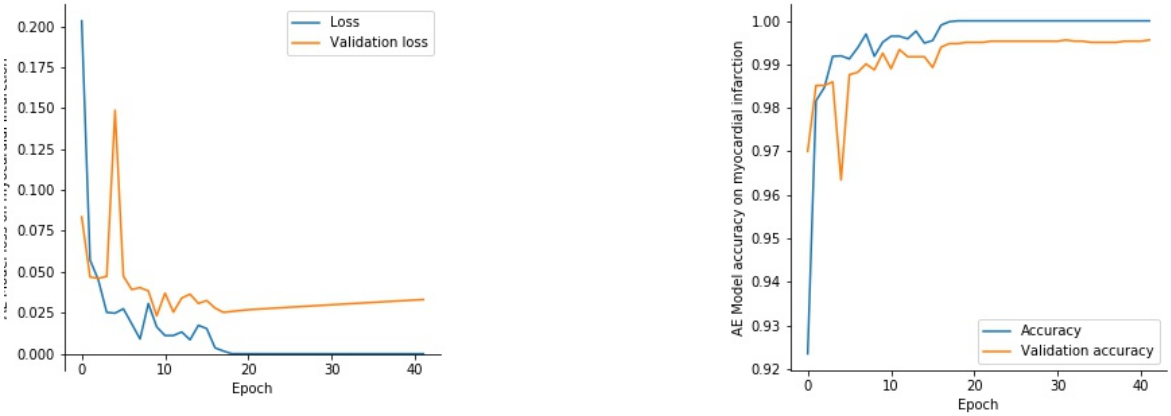
A *4.1. ábra* a modell tanításának tulajdonságait mutatja. Látható, hogy a tanító adatokon a pontosság elérte a 100%-ot. Ezt a a *4.2 a ábrán* látható igazságmátrix megerősíti ezt, míg a test adatokon 98.63% a pontosság és kiszámolható a hozzá tartozó precizitás és a felidézés, melyek értékei rendre 98% és 99.2%.




**(a)** Train adatok igazságmátrixa **(b)** Test adatok igazságmátrixa

**4.2. ábra.** MI adatokon igazságmátrix

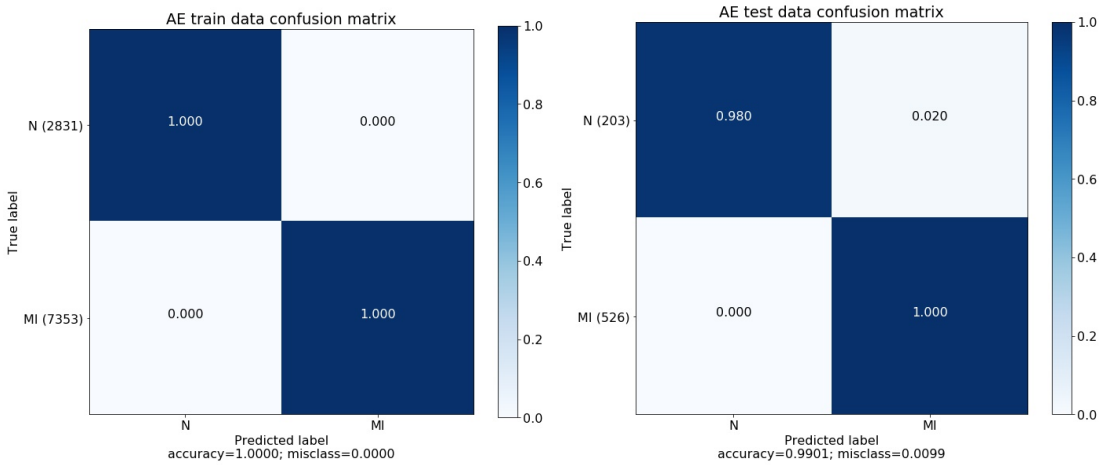
## 4.2. 1D konvolúciós neurális háló autó-enkóderrel




**(a)** Modell tanítás veszteség értékei **(b)** Modell tanítás pontosság értékei

**4.3. ábra.** MI adatokon a modell tanítás tulajdonságai

A *4.3. ábráról* leolvasható, hogy az autó-enkóder struktúrát tartalmazó modell a tanító adatokra elérte a 100% pontosságot. A *4.6. ábrán* látható, hogy a test adatokon a pontosság 99% és kiszámolható a hozzá tartozó precizitás és a felidézés, melyek értékei rendre 98% és 100%.

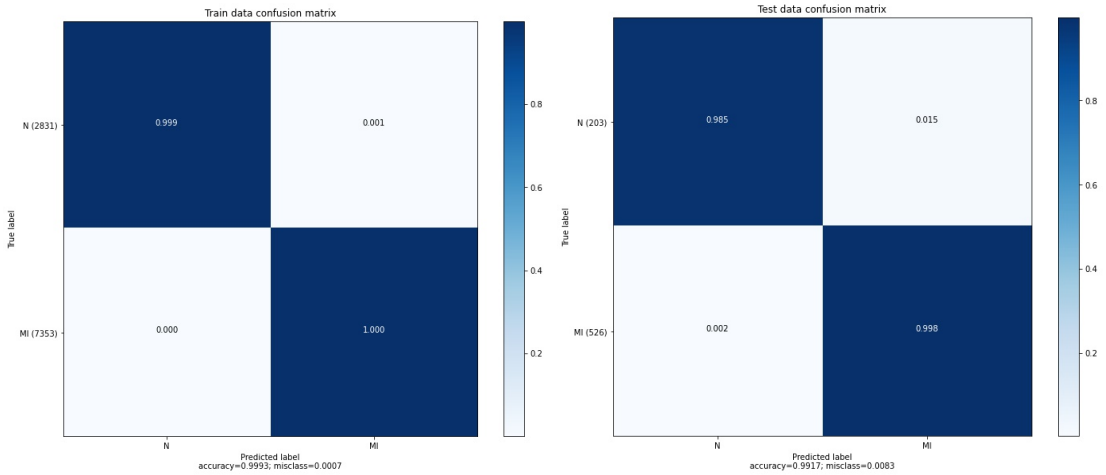



**(a)** Train adatok igazságmátrixa **(b)** Test adatok igazságmátrixa

**4.4. ábra.** MI adatokon igazságmátrix

## 4.3. LSTM I. architektúra

A *4.5. ábrán* látható igazságmátrix mutatja a tanító és a teszt adatokon az eredményt. Míg a tanító adatokn 99.93% pontosságot tudott elérni addig a test adatokon 99.17% a pontosság és kiszámolható a hozzá tartozó precizitás és a felidézés, melyek értékei rendre 98.5% és 99.6%. És ezekből az $F_1 = 99.05\%$ számítható.

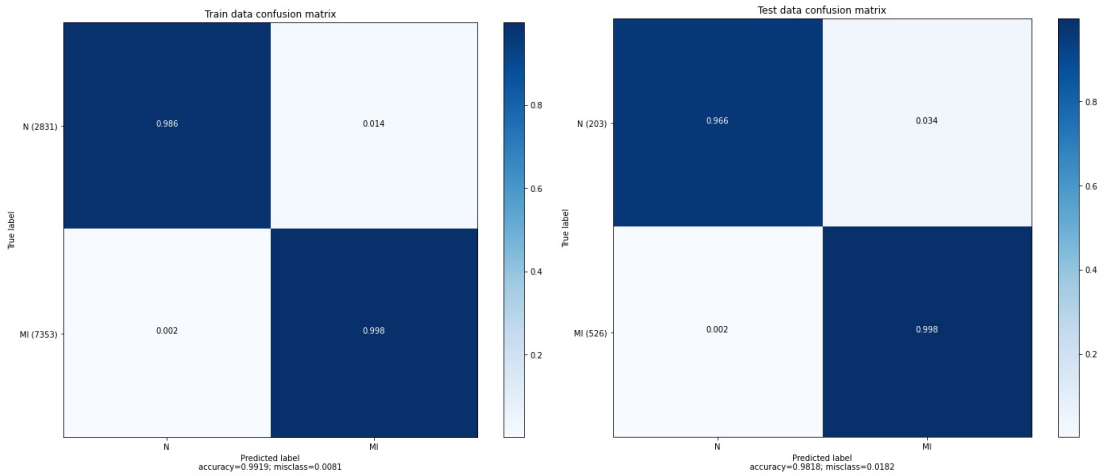



**(a)** Train adatok igazságmátrixa **(b)** Test adatok igazságmátrixa

**4.5. ábra.** MI adatokon igazságmátrix

## 4.4. LSTM II. architektúra

A *4.5. ábrán* látható igazságmátrix mutatja a tanító és a teszt adatokon az eredményt. Míg a tanító adatokn 99.19% pontosságot tudott elérni addig a test adatokon 98.18% a pontosság és kiszámolható a hozzá tartozó precizitás és a felidézés, melyek értékei rendre 96.6% és 99.8%. És ezekből az $F_1 = 98.17\%$ számítható.




**(a)** Train adatok igazságmátrixa **(b)** Test adatok igazságmátrixa

**4.6. ábra.** MI adatokon igazságmátrix

## 4.5. Összehasonlítás

| **Modellek** | **Pontosság (%)** | **Precizitás (%)** | **Felidézés (%)** |
|---|---|---|---|
| **LSTM I. architektúra** | **99.17** | **98.5** | **99.6** |
| LSTM II. architektúra | 98.18 | 96.6 | 99.8 |
| Convolutional architektúra | 98.6 | 98 | 99.2 |
| *Convolutional architektúra with AE* | *99* | *98* | *100* |
| Kachuee és társai[7] | 95.9 | 95.5 | 95.1 |
| Acharya és társai[1] | 93.5 | 92.8 | 93.7 |
| Safdarian és társai[16] | 94.7 | – | – |
| Kojuri és társai[10] | 95.6 | 97.9 | 93.3 |
| Sun és társai[19] | – | 82.4 | 92.6 |
| Liu és társai[11] | 96 | 99 | 93 |
| Remya és társai[4] | 86 | – | 83 |
| Feng és társai[5] | 95.4 | – | 98.2 |
| Sharma és társai[17] | – | – | 98.7 |

**4.1. táblázat.** Szívinfarktus osztályozás eredmények összehasonlítása

A *4.1. táblázatban* látható a pontosság (accuracy), precizitás (precision) és a felidézés (recall) alapján az összehasonlítása a szívinfarktus osztályozásra javasolt modelleknek és az irodalomban található megoldásoknak. Fél kövérrel kiemeltem a legjobban teljesítő LSTM modellt, valamint dőlt-el a legjobban teljesítő konvolúciós modellt. Ebből látszik, hogy a javasolt modellek teljesítménye meghaladja az irodalomban találtakat. Észrevehető, hogy az LSTM struktúrák között a residual skip-connection-t tartalmazó II. modell rosszabbul teljesített mint a modell alapja. Ez valószínűleg a residual network nagyobb adat szükségletének és a háló nem megfelelő mélységének kombinációjából fakad.

# 5. fejezet

# Összefoglalás

A dolgozat fő célja az EKG alapú szívinfarktus klasszifikáció volt rekurrens és azon belül is LSTM hálózattal, valamint konvolúciós hálózattal. A dolgozat első része a deep learning eszköztár alapjainak megismeréséről szól, kitérve a használt eszközökre illetve azok megértéséhez szükséges módszerekre.
Ezután a célkitűzésben felvetett módszerekkel történő háló implementációja történt. Emellett a dolgozatban használt adatbázisok valamint az adatok előkészítésének lépéseit mutattam be.

Ezeken az adatokon tanítottam az implementált hálókat, és a javasolt modellek eredményét vetjük össze a irodalomban fellelhető eredményekkel. Az implementált LSTM modellek pontossága rendre *99.17%* és 98.18%. Ezek az eredmények az irodalomban fellelhető előzményekhez képest kimagaslónak számítanak. Az implementált CNN-ek is hasonlóan jó, bár végeredményként rosszabb eredményeket értek el. Ez a base CNN modellnél 98.6% pontosságot, míg az autó-enkóder struktúrát alkalmazó modellnél 99% pontosságot jelent. Ezzel mutatva, hogy az autó-enkóder struktúra előnyös alkalmazni, míg a LSTM modellek mutatják, hogy ilyen kis méretű adat mennyiség esetén a residual network használata nem ajánlott.

## 5.1. Jövőbeli tervek

Mivel a jelen munkában csak külön külön használtam a rekurrens és a konvolúciós hálókat ezért a jövőben tervezem a jelenleg is fejlődő, ezen módszerek össze olvadásából álló konvolúciós-rekurrens háló elkészítését. Valamint tervezem, további hiperparaméterekkel kipróbálni, és ezzel finomra hangolni az implementált hálókat.

Budapest, 2020. október 28.

*Ritter Áron*

# Irodalomjegyzék


[1] U Rajendra Acharya – Hamido Fujita – Shu Lih Oh – Yuki Hagiwara – Jen Hong Tan – Muhammad Adam: Application of deep convolutional neural network for automated detection of myocardial infarction using ecg signals. *Information Sciences*, 415. évf. (2017), 190–198. p.

[2] U Rajendra Acharya – Shu Lih Oh – Yuki Hagiwara – Jen Hong Tan – Muhammad Adam – Arkadiusz Gertych – Ru San Tan: A deep convolutional neural network model to classify heartbeats. *Computers in biology and medicine*, 89. évf. (2017), 389–396. p.

[3] Christopher M Bishop: *Pattern recognition and machine learning*. 2006, springer, 206–209. p.

[4] Ammarah Farooq – SyedMuhammad Anwar – Muhammad Awais – Saad Rehman: A deep cnn based multi-class classification of alzheimer's disease using mri. In *2017 IEEE International Conference on Imaging systems and techniques (IST)* (konferenciaanyag). 2017, IEEE, 1–6. p.

[5] Kai Feng – Xitian Pi – Hongying Liu – Kai Sun: Myocardial infarction classification based on convolutional neural network and recurrent neural network. *Applied Sciences*, 9. évf. (2019) 9. sz., 1879. p.

[6] Donald Olding Hebb: *Organization of Behavior*. 1949.

[7] Mohammad Kachuee – Shayan Fazeli – Majid Sarrafzadeh: Ecg heartbeat classification: A deep transferable representation. In *2018 IEEE International Conference on Healthcare Informatics (ICHI)* (konferenciaanyag). 2018, IEEE, 443–444. p.

[8] Diederik P Kingma – Jimmy Ba: Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] Diederik P. Kingma – Jimmy Ba: Adam: A method for stochastic optimization, 2017.

[10] Javad Kojuri – Reza Boostani – Pooyan Dehghani – Farzad Nowroozipour – Nasrin Saki: Prediction of acute myocardial infarction with artificial neural networks in patients with non-diagnostic electrocardiogram. *Journal of Cardiovascular Disease Research*, 6. évf. (2015) 2. sz., 51–59. p.

[11] Taiyong Li – Min Zhou: Ecg classification using wavelet packet entropy and random forests. *Entropy*, 18. évf. (2016) 8. sz., 285. p.

[12] Zachary C. Lipton – David C. Kale – Charles Elkan – Randall Wetzel: Learning to diagnose with lstm recurrent neural networks, 2017.

[13] Zhou Lu – Hongming Pu – Feicheng Wang – Zhiqiang Hu – Liwei Wang: The expressive power of neural networks: A view from the width. In *Advances in neural information processing systems* (konferenciaanyag). 2017, 6231–6239. p.

[14] Roshan Joy Martis – U Rajendra Acharya – Choo Min Lim – KM Mandana – Ajoy K Ray – Chandan Chakraborty: Application of higher order cumulant features for cardiac health diagnosis using ecg signals. *International journal of neural systems*, 23. évf. (2013) 04. sz., 1350014. p.

[15] David E Rumelhart – Geoffrey E Hinton – Ronald J Williams: Learning internal representations by error propagation. Jelentés, 1985, California Univ San Diego La Jolla Inst for Cognitive Science.

[16] Naser Safdarian – Nader Jafarnia Dabanloo – Gholamreza Attarodi: A new pattern recognition method for detection and localization of myocardial infarction using t-wave integral and total integral as extracted features from one cycle of ecg signal. *Journal of Biomedical Science and Engineering*, 07. évf. (2014. 01), 818–824. p.

[17] Lakhan Dev Sharma – Ramesh Kumar Sunkaria: Inferior myocardial infarction detection using stationary wavelet transform and machine learning approach. *Signal, Image and Video Processing*, 12. évf. (2018) 2. sz., 199–206. p.

[18] Denis Smirnov – Engelbert Mephu Nguifo: Time series classification with recurrent neural networks, 2018. URL `pp.1\T1\textendash8.url:https://project.inria.fr/aaldt18/files/2018/08/oral.pdf`..

[19] Li Sun – Yanping Lu – Kaitao Yang – Shaozi Li: Ecg analysis using multiple instance learning for myocardial infarction detection. *IEEE transactions on biomedical engineering*, 59. évf. (2012) 12. sz., 3348–3356. p.

[20] Özal Yildirim: A novel wavelet sequence based on deep bidirectional lstm network model for ecg signal classification. *Computers in Biology and Medicine*, 96. évf. (2018), 189 – 202. p. ISSN 0010-4825.
URL `http://www.sciencedirect.com/science/article/pii/S0010482518300738`.

# Summary

The main purpose of my dissertation was the classification of myocardial infarction using recurrent neural network (LSTM) and convolutional neural network. The paper consists of 3 main parts. In the first part I present the basis of some deep learning methods, including the used techniques and the needed informations in order to understand the mentioned methods. Afterwards the second part is about the implementation of the networks using these mechanisms and the presentation of the used databases and the preprocessing steps of the data. These are the data we train our proposed networks, and we compare the suggested system's result to results in the literature. We can declare that the proposed LSTM model's accuracy is 99,17% and 98,18%. These results are outstanding compared to earlier researches.

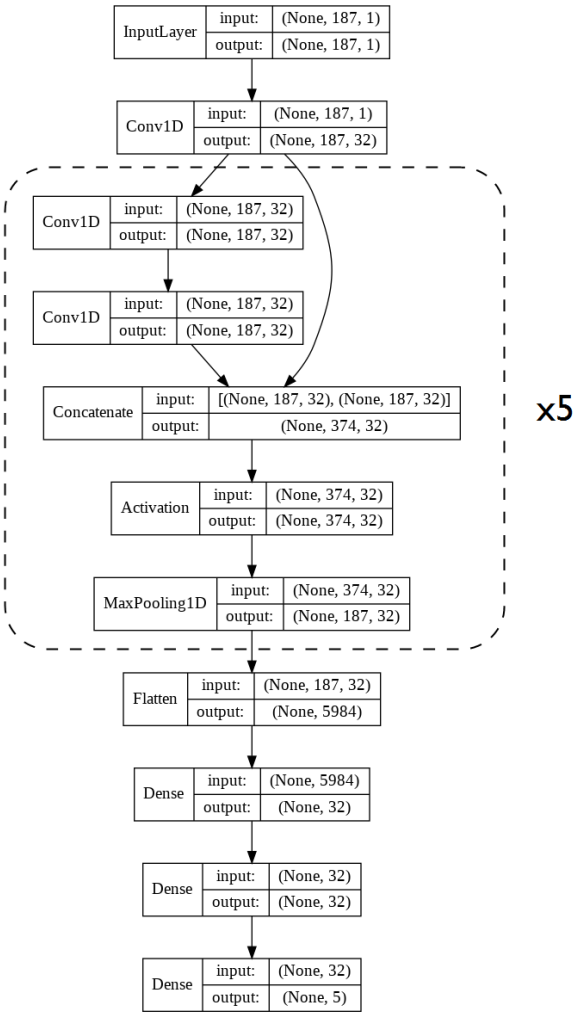
## 5.2. Future work

Since I only used separately the recurrent and convolutional neural network in this project, my plan for the future is to make a convolutional-recurrent neural network. In which the convolutional and recurrent systems emerge. As well as I plan to try out more hiperparameters to make the implemented systems more accurate. Keywords: *myocardial infarction, PTB, convolutional*

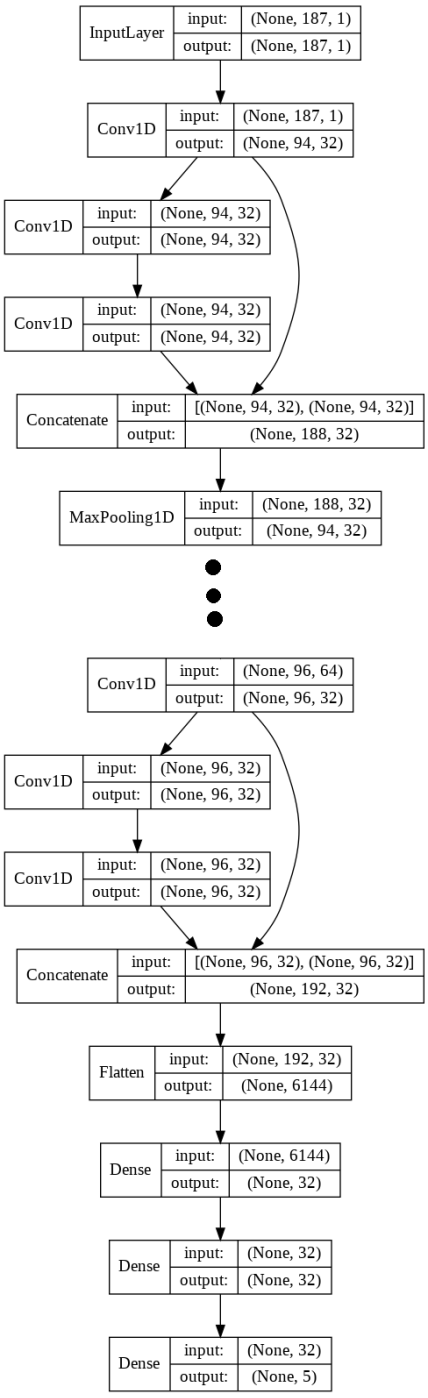
*neural network, transfer learning, residual network*

# Függelék

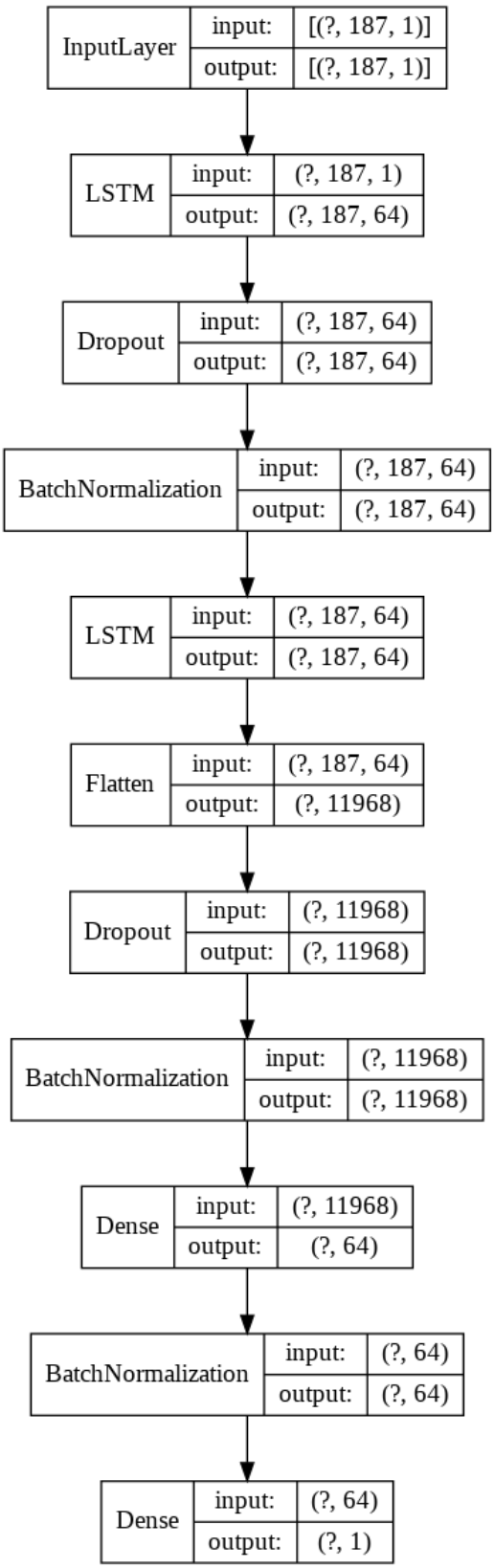
## F.1. Az implementált architektúrák topológiája



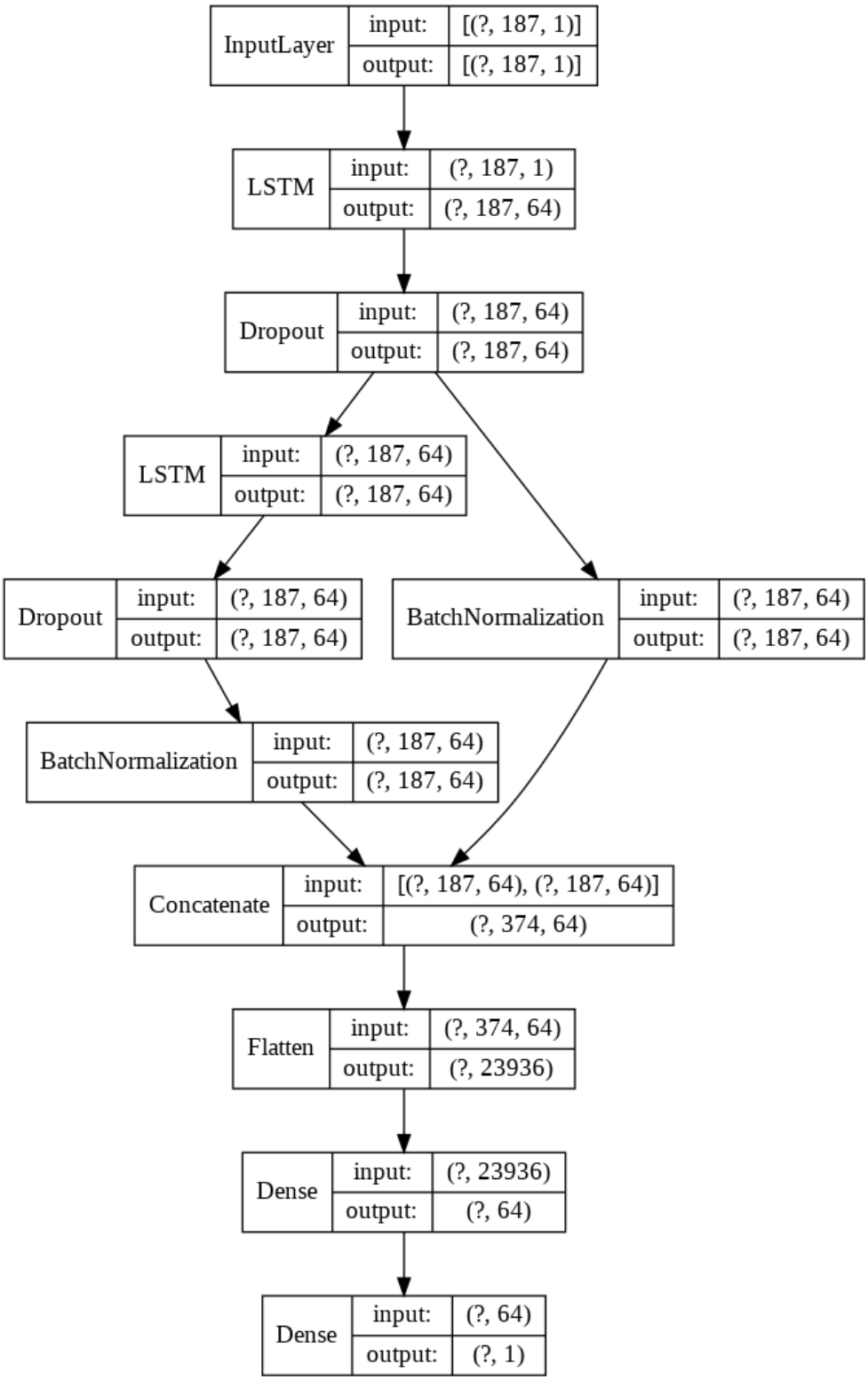

**F.1.1. ábra.** Convolutional architektúra aritmia esetén.




**F.1.2. ábra.** Convolutional architektúra aritmia esetén, autó-enkóder struktúrával.




**F.1.3. ábra.** `LSTM I. architektúra.`




**F.1.4. ábra.** LSTM II. architektúra.