Eötvös Loránd Tudományegyetem
Természettudományi Kar

Aleksziev Rita Antónia
Alkalmazott matematikus MSc
Számítástudomány szakirány

Transzformációtanulás 2 dimenziós képeken mély autoenkóder hálózatokkal

Szakdolgozat

Témavezető: Dr. Kondacs Attila
SZTAKI

Belső konzulens: ifj. Dr. Benczúr András
Operációkutatási Tanszék

Budapest, 2017.

Köszönetnyilvánítás

Köszönettel tartozom témavezetőmnek, Kondacs Attilának, amiért elvállalta a konzulensi teendőket. A készülő munka többszöri figyelmes átolvasásával, hasznos formai és tartalmi tanácsaival nagy segítséget nyújtott a szakdolgozat írásában. Köszönöm továbbá belső témavezetőmnek, Benczúr Andrásnak, hogy észrevételeivel segítette a munkámat.

Tartalomjegyzék

Előszó

A mesterséges neurális hálózatok a tudomány egyre több területén oldanak meg különféle problémákat. Ezen feladatok nagy része invariáns a bemenet bizonyos transzformációira. Jelen dolgozat célja, hogy bemutassa a mesterséges neurális hálózatok alapjait, majd tárgyaljon olyan példákat, amelyek az említett invarianciákat kihasználják, vagy az azokat adó transzformációk paramétereit kinyerik az adatból. A dolgozat végén bemutatok két saját hálózatot, amely különböző mértékű eltolásokkal transzformált kézzel írott számjegyet ábrázoló képeket kezel. Feladatuk a számjegyfelismerés, a minta centralizálása, és az eltolás paramétereinek visszaadása.

1. Bevezetés

A gépi tanulás a mesterséges intelligencia egyik ága, amely olyan rendszerekkel foglalkozik, melyek tanulni képesek, azaz tapasztalatok alapján fejlődik a működésük. A hagyományos programozásban lebontjuk a feladatokat kisebb, precízen definiált részfeladatokra, és utasításokkal "megmondjuk" a számítógépnek, hogy hogyan oldja meg őket. Ezzel szemben gépi tanulás esetén a megoldás pontos lépéseinek megadása helyett megfigyelt adatokat, példákat biztosítunk a számítógépnek, amelyből az generál egy modellt. Tanulás alatt a következő, Mitchell ([16]) által adott fogalmat értjük.

1.1. Definíció. *Adott $\mathcal{A}$ számítógépes program, $\mathcal{E}$ és $\mathcal{T}$ halmazok, és egy P valós értékű függvény. $\mathcal{E}$ elemeit tapasztalatoknak, $\mathcal{T}$ elemeit feladatoknak nevezzük. Ekkor $\mathcal{A}$ **tanul** feladatoknak egy $\mathcal{T}$ halmazára nézve $\mathcal{E}$ tapasztalatokból, ha a $\mathcal{T}$-n nyújtott, P-vel mért teljesítménye $\mathcal{E}$ hatására nő.*

A gyakorlatban ez azt jelenti, hogy a rendszer tanító adatok, minták alapján képes (önállóan vagy emberi segítséggel) szabályszerűségeket felismerni. A fenti definíció megengedi bármilyen $\mathcal{E}$ és $\mathcal{T}$ halmazok, illetve P függvény használatát, még P értelmezési tartománya sem rögzített. Amikor egy konkrét probléma megoldására keresünk tanuló algoritmust, akkor ezeket kell pontosan megadnunk. Egy olyan $\mathcal{A}$ program esetén például, amelytől azt várjuk, hogy tanuljon meg sakkozni, $\mathcal{T}$ egyetlen eleme a sakkozás, a tapasztalatok halmaza emberek elleni játszmákból áll, P pedig a megnyert játszmák aránya az összeshez képest. Egy önvezető autó vezérléséhez használt algoritmus feladathalmaza a különböző útvonalakon való szabályos közlekedés, tapasztalatait képsorok (és más szenzorok bemenetei), illetve hozzájuk tartozó kezelési parancsok adják, teljesítményét pedig a hibátlanul megtett út átlagos hosszával mérjük.

Általában a program teljesítményét nem, vagy nem csak $\mathcal{E}$ elemein mérjük, erre egy másik, $\mathcal{E}'$ tesztelő halmaz szolgál. Két különböző halmaz fenntartása azért indokolt, hogy az esetleges túltanulást észrevegyük. Azaz nemcsak azt várjuk a programtól, hogy az általa már ismert adatokra illetve helyzetekben megfelelően működjön, hanem azt is, hogy új bemenetre jól tudjon általánosítani: egy új stratégiával játszó ellenfél ellen is nyerjen sakkban, vagy ismeretlen utakon is tudjon közlekedni. Jellemző még, hogy nem az összes tapasztalat hatásának bekövetkezése után végzünk először mérést, hanem felváltva látjuk el $\mathcal{A}$-t tapasztalatokkal és mérjük az addig nyújtott teljesítményt $\mathcal{E}$-n vagy $\mathcal{E}'$-n. Ekkor tanulásnak tekintjük, ha az utolsó teljesítményérték az elsőnél (számottevően) nagyobb, nem várjuk el, hogy minden egyes iterációban az előzőhöz képest nőjön.

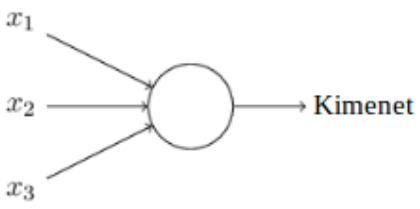
A **mesterséges neurális hálózat** egy tanításhoz használatos modell, amelynek biológiai inspirációja az élőlények idegrendszere, és bemenő értékek együtteséből kimenő értéket állít elő. Meghatározott sorrendben egymáshoz kapcsolt feldolgozó (számoló) egységekből áll, ezek felelnek meg a biológiai neuronoknak, és együtt képesek tanulásra a köztük lévő kapcsolatok változtatásával. A modell kiemelkedő eredményeket hozott olyan területeken, mint a számítógépes látás, a beszédfelismerés és a nyelvfeldolgozás, ahol a megoldandó problémák az emberi agy számára egyszerűek, azonban nehéz őket formálisan leírni. Ilyen például az arcok vagy a kézzel írott karakterek felismerése, mozgó tárgyak követése, hallott szöveg leírása, a röntgen- és ultrahangos felvételek elemzése, vagy akár egy jármű irányítása forgalomban.

2. Neurális hálózatok és mély tanulás

2.1. A neurális hálózatokról általában

A mesterséges neuronháló alapvető építőkövei a **mesterséges neuronok**, amelyek bemenő értékekből kimenő értéket számoló egységek. Ezek a **hálózati topológia** által meghatározott módon kapcsolódnak egymáshoz. Ennek reprezentációjára általában irányított gráfot szokás használni, amelynek a neuronok a csúcsain helyezkednek el. A gráf minden uv irányított élére teljesül, hogy az u-nak megfelelő neuron kimenetét bemenetként megkapja a v-n álló neuron. Bizonyos neuronok bemenete a hálózat bemenetének egy része, mások kimeneteit pedig hálózat kimenetének tekintjük. A neuronok számítási képleteit (az őket összekötő élek súlyát) meghatározó eljárás a hálózat **tanulási szabály**a.

A neuronok legegyszerűbb fajtája a **perceptron**, amelyet Frank Rosenblatt fejlesztett ki az 1950-es évek végén. Ez az egyszerű egység néhány bináris bemeneti értékből számít egyetlen bináris kimenetet.



1. ábra. Perceptron x_1, x_2, x_3 bemenetekkel

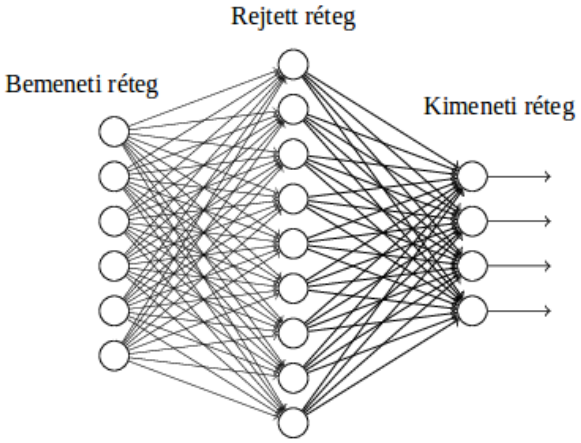
A bejövő $x_1, x_2, ...$ értékeket valós $w_1, w_2, ...$ számokkal súlyozza, a kimenetet pedig az alapján határozza meg, hogy az így kapott összeg elér-e egy előre meghatározott k küszöbértéket. A súlyok és a küszöbérték a neuron saját paraméterei. Formálisan tehát a perceptron kimenetét a

$$kimenet = \begin{cases} 0, & \text{ha } \sum_{i=1}^{n} w_i x_i \leq k \\ 1, & \text{ha } \sum_{i=1}^{n} w_i x_i > k \end{cases}$$

formula adja. Az egyszerűség kedvéért a $\sum_{i=1}^{n} x_i w_i$ kifejezést vektorok szorzataként, azaz $\underline{w}\underline{x}$ alakban szokás használni, illetve bevezethető egy $b = -k$ eltolásérték (bias). Ezekkel a jelölésekkel tehát a neuron a $\underline{w}\underline{x} + b$ értéket számítja ki, és ennek függvényében határozza meg a kimenetét. Szemléletes a $\underline{w}$ vektor elemeire élsúlyokként tekinteni: w_i azon él súlya, amelyen x_i érkezik. A továbbiakban tehát a "neuron paraméterei" és az "élsúlyok" fogalmakat felcserélhetőnek tekintjük, és a kontextustól függ, hogy beleértjük-e az eltolásértékeket is.

Neuronok bármilyen függvényt megvalósító számoló egységek lehetnek (amennyiben az értékkészlet egy számhalmaz). A gyakorlatban sokféle probléma megoldására használatos olyan hálózat, amely neuronjai leírhatóak a Rosenblatt-féle perceptron általánosított változataként. Ez a fajta neuron az eredetinek egy f aktivációs függvénnyel való kiegészítése. Az **általánosított perceptron** kimenete az $f(\underline{w}\underline{x} + + b)$ érték. Az aktivációs függvény intuitív jelentése, hogy eldönti, hogy a neuron mennyire reagál az őt ért impulzusokra. A legegyszerűbb esetben f kimenete bináris, tehát a bemenő információ alapján vagy továbbít ingert, vagy nem. A leggyakrabban használt aktivációs függvények monoton növekvőek, folytonosan differenciálhatóak, nemlineárisak, és értékkészletük a $(0, 1)$ vagy a $(-1,1)$ intervallum. Az ilyen – 'S' alakú grafikonnal rendelkező – függvényeket gyűjtőnéven szigmoid-függvényeknek nevezzük. Az elnevezés néha a logisztikai függvény speciális esetére utal, amelyet az $f(x) = \frac{1}{1+e^{-x}}$ képlet definiál. Ezen kívül még sűrűn találkozhatunk aktivációs függvényként a hiperbolikus tangens, az arcus tangens és a softsign ($f(x) = \frac{x}{1+|x|}$) függvényekkel.

A hálózati topológia két fő fajtája a hurokmentes vagy **előrecsatolt** (feed-forward), és a visszacsatolt vagy **rekurrens**. Az előrecsatolt hálók rendszerint rétegekbe (layer) rendezve tartalmazzák a neuronokat oly módon, hogy minden egyes egység csak a közvetlenül megelőző réteg egységeitől kap bemeneti jelet. Így az információ az első (bemeneti) szinttől a közbülső (rejtett) szinteken át rétegről rétegre az utolsó (kimeneti) szint felé áramlik. rekurrens háló rétegei között ezzel szemben nincs ilyen rendezés, azaz a hálózati topológiát leíró gráf köröket tartalmaz. [20]



2. ábra. Rétegekből álló előrecsatolt neurális hálózat [18]

A mesterséges neurális hálózatokat bemenet-kimenet párok segítségével tréningezzük, azaz rendelkezésre állnak lehetséges bemenetek, és azok egy részéhez a kívánt kimenet (címke). A tanítás célja, hogy a hálózat megtanuljon egy olyan leképezést végrehajtani, ami illeszkedik az általa látott mintákra (azaz az ismert bemenetekre a hozzájuk tartozó kimenetet adja), és elég általános ahhoz, hogy új bemenetekre is alkalmazható legyen. A rendelkezésre álló tanító adattól függően a tréningezés három fajtáját különböztetjük meg. **Felügyelt tanulás**nak nevezzük, amikor az adott bemenetek mindegyikéhez ismert a kívánt kimenet, és a tréningezéshez az így kapott párokat használjuk. **Felügyelet nélküli** tanulás esetén a várt kimenetre nincs szükség, csupán az ismert bemenetek közötti összefüggéseket tanulja a hálózat. Azokat a tanító algoritmusokat, amik a fenti két módszert ötvözik, **félig felügyelt**nek nevezzük, ezek azokban az esetekben használatosak, amikor a mintabemenetek egy részéhez rendelkezésünkre áll a hozzájuk tartozó kimenet, de nem mindegyikhez. Klasszikus példa felügyelt tanulásra a kézzel írott számjegyek felismerése, ahol a bemenet egy számjegyet ábrázoló kép, a kimenet pedig egy 0 és 9 közötti érték. A megtanulandó függvény természetesen az, ami minden képet abba a számjegybe képez, amelyiket ábrázolja. Abban az esetben, ha a rendelkezésre álló tanító adathalmaz mérete nem elég nagy, mesterségesen megnövelhetjük azt (augmentáció). Ennek a két leggyakoribb módja a zaj hozzáadása a már meglévő adatokhoz, illetve valamilyen transzformáció alkalmazása.

A neuronokat összekötő élek súlya kezdetben rendszerint valamilyen eloszlás szerinti véletlen, a tanulási folyamat pedig ezen élsúlyok fokozatos változtatásából áll. A módosítások pontos menetét a hálózat **tanulási szabály**a határozza meg. Annak számszerűsítésére, hogy mennyire járunk közel a célhoz, bevezetünk egy C hiba-

vagy **költségfüggvény**t, amely legtöbbször a hálózat kimenete és a kívánt kimenet közti eltérést méri, mint például a négyzetes távolság. A tanulás közvetlen célja ezen függvény értékének minimalizálása. A leggyakrabban használt tanulási szabályok gradiens-alapú iteratív módszerek, amelyek az élsúlyokra és az eltolásokra vonatkozó parciális deriváltakra épülnek. Az aktivációs függvényeket emiatt célszerű úgy választani, hogy azok "könnyen" differenciálhatóak legyenek. Ilyen például a már említett $f(x) = \frac{1}{1+e^{-x}}$ logisztikai függvény, amelynek az x helyen vett deriváltja $f'(x) = f(x)(1 - f(x))$.

A tanító algoritmusok nagy részének egy ciklusa egy "előre"-lépésből (a neuronok kimeneteinek kiszámítása valahány tanító mintára rétegről rétegre), a deriváltak meghatározásából, és az élsúlyok illetve az eltolásértékek kis mértékű módosításából áll. Ebben tehát a mesterséges neurális hálózat nem követi a biológiai neuronháló működését, mert diszkrét, egymást követő lépések sorozatát hajtja végre ellenben az idegek decentralizált, központi órajel nélküli folytonos működésével. Az egész mintahalmazon való egyszeri végigfutást **epoch**nak nevezzük. Azt, hogy az iterációt meddig ismételjük, a **megállási szabály** határozza meg. Ez lehet például egy előre megadott ismétlési szám, de jellemző, hogy megáll az algoritmus, ha a hibafüggvény egy bizonyos küszöbérték alá csökken, vagy ha a paraméterek változtatásának mértéke már nagyon kicsi.

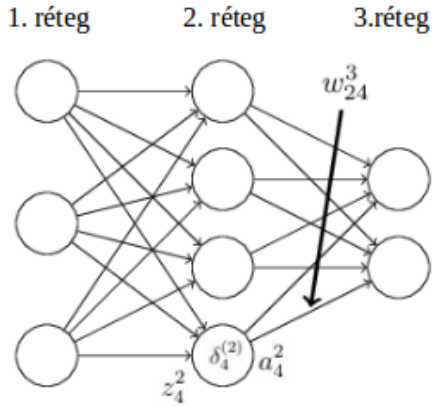
A deriváltak kiszámításához a gyakorlatban általában a **hiba-visszaterjesztési algoritmust** (backpropagation, [18]) használják, amelynek menete a következő: a kimeneti réteg minden neuronjára kiszámolunk egy lokális hibát, majd rétegenként, hátulról előre haladva számoljuk a belső neuronok hibáit. Ezekből a hibákból ezután következtetünk a hibafüggvény parciális deriváltjaira.

A hiba-visszaterjesztési algoritmus leírásához szükséges bevezetni a következő jelöléseket. Az $(l-1)$-edik réteg k-adik neuronjából az l-edik réteg j-edik neuronjába vezető él súlya legyen w^l_{jk}. Az l-edik réteg neuronjának eltolásértékét jelöljük b^l_j-vel, kimenetét pedig a^l_j-vel, azaz $a^l_j = f\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right)$, ahol f a neuron aktivációs függvénye, az összeg pedig az $(l-1)$-edik réteg neuronjainak indexein fut végig. Az l-edik réteg j-edik neuronjába érkező adatok élsúlyokkal súlyozott és eltolásértékkel eltolt összegét z^l_j-vel jelöljük, tehát $z^l_j = \sum_k w^l_{jk} a^{l-1}_k + b^l_j$.

A gradiens-módszer a C költségfüggvény $\frac{\partial C}{\partial w}$ és $\frac{\partial C}{\partial b}$ parciális deriváltjaira támaszkodik, a hálózat minden w élsúlyára és b eltolásértékére. A költségfüggvénynek többek között a következő két fő feltételt kell teljesítenie ahhoz, hogy a hálózat hibavisszaterjesztési algoritmussal tréningezhető legyen. Az egyik, hogy C kifejezhető legyen az egyes x tanító adatokra vonatkozó C_x függvények $C = \frac{1}{n}\sum_x C_x$ átlagaként. Ez azért fontos, mert a tanítás során valójában a $\frac{\partial C_x}{\partial w}$ és a $\frac{\partial C_x}{\partial b}$ deriváltakat tudjuk kiszámolni, és ezekből a tanító adathalmazon való átlagolással állítjuk elő

$\frac{\partial C}{\partial w}$-t és $\frac{\partial C}{\partial b}$-t. A másik feltétel, hogy C felírható legyen a hálózat kimeneteinek függvényeként. Természetesen C-nek más változói is vannak, mint például a hálózat többi élsúlya, vagy a kívánt output, de ezeket ebben a felírásban rögzített paraméternek tekintjük.

A hiba-visszaterjesztési algoritmus először az utolsó szintről indulva minden neuronra kiszámol egy **neuronhibá**t. Az l-edik szint j-edik neuronjának hibáját δ_j^l-vel jelöljük, képlete pedig $\frac{\partial C}{\partial z_j^l}$. Az elnevezés arra utal, hogy a neuron bemeneteinek súlyozott összegét, z_j^l-t egy kis Δz_j^l értékkel megváltoztatva a hálózat hibafüggvényének értéke $\delta_j^l \Delta z_j^l$-vel változik. A hibaérték tehát azt hivatott kifejezni, hogy az adott neuron mennyire járul hozzá az egész hálózat hibájához.



3. ábra. Néhány példa a használt jelölésekre egy 3 rétegű neurális hálózatban. A 2. réteg 4. neuronjába érkező érték z_4^2, a belőle kimenő a_4^2. A közte és a 3. réteg 2. neuronja közt futó él súlya w_{24}^3 , a neuron hibája pedig δ_4^2. [18]

Az utolsó (L-edik) szint neuronjainak hibája a deriváltra vonatkozó láncszabály alapján $\delta_j^L = \frac{\partial C}{\partial a_j^L} f'(z_j^L)$. Az algoritmus általános lépésének leírásához érdemes áttérnünk mátrixos ábrázolásra: jelölje δ^l az l-edik réteg hibáinak vektorát, azaz azt a vektort, amelynek j-edik eleme δ_j^l, ezzel analóg módon értelmezzük az a^l és a b^l vektorokat. Az $l-1$-edik szintről az l-edik szintre vezető élek súlyaiból alkotott mátrix w^l, amely j-edik sorának k-adik eleme w_{jk}^l. Ezekkel a jelölésekkel tehát az l-edik réteg hibáit az $l+1$-edik szintre kapott értékekből a láncszabály rekurzív alkalmazásával kapott $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot f'(z^l)$ formula adja, ahol $\odot$ az elemenkénti szorzást jelöli, és f-et is elemenként alkalmazzuk a vektorra.

Az így kapott hibákat használja fel a tanító algoritmus a gradiens-módszerhez szükséges parciális deriváltak kiszámítására. A b eltolásokra vonatkozó deriváltakkal

könnyű dolgunk van, ugyanis a b_j^l-hez tartozó $\frac{\partial C}{\partial b_j^l}$ éppen megegyezik δ_j^l-vel. A w élsúlyokra vonatkozó parciális deriválthoz szükség van a hibákon kívül a kiindulási réteg aktivációira is, erre a $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$ egyenlet vonatkozik.

Egy lehetséges tanulási szabály a széles körben használatos **gradient descent** (”legmeredekebb lejtő”) módszer. Ez az algoritmus úgy keresi egy többváltozós függvény minimum helyét, hogy egy véletlenszerűen (vagy más módszerrel) választott helyről indulva a gradiens -1-szeresével arányos lépésekkel választ új helyet addig, amíg a függvény értéke kisebb, mint az előző vizsgált pontban. A helyváltoztatás mértékét az algoritmus paramétere, az η **tanulási ráta** határozza meg. Ez egy közelítő módszer, nem mindig ad pontos eredményt, ráadásul az általa visszaadott érték egy lokális minimumot közelít, amiről általában semmi nem garantálja, hogy globális minimum is. Egy C függvény értelmezési tartományának v eleméből tett általános lépés tehát $\Delta v = -\eta \nabla C$, ahol ∇C a gradiens vektor, azaz a v után következőként a $v - \eta \nabla C$ pontot vesszük. Fontos, hogy η-t elég kicsinek válasszuk ahhoz, hogy a kapott eredmény jó közelítés legyen a minimum helyre, de elég nagynak ahhoz, hogy az algoritmus ne legyen túl lassú. A backpropagation eredményeként kapott deriváltak segítségével tehát a következőképpen frissítjük a neurális hálózat w_{jk} élsúlyait és b_l eltolásértékeit:

$$_{jk} \to w'_{jk} = w_{jk} - \eta \frac{\partial C}{\partial w_{jk}}$$
$$b_l \to b'_l = b_l - \eta \frac{\partial C}{\partial b_l}$$

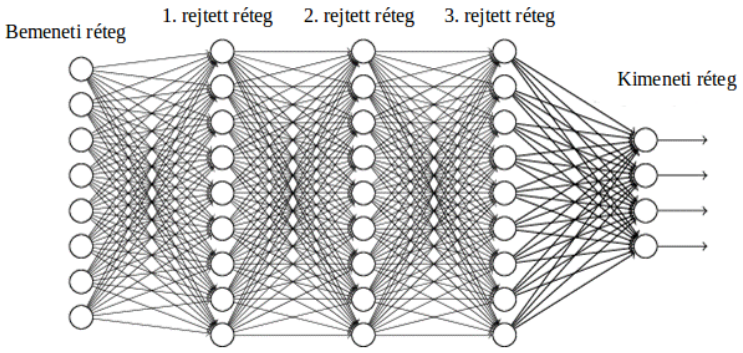
Ahhoz, hogy ∇C-t pontosan meghatározzuk, minden x tanító mintára ki kell számolnunk a rá vonatkozó C_x költségfüggvény ∇C_x gradiensét, és ezeket átlagolnunk kell. Ha nagy az adathalmaz, akkor ez rengeteg ideig tarthat. Erre a problémára nyújt megoldást az algoritmus módosított változata, a **stochastic gradient descent** módszer, amely egy iterációban a mintahalmaznak csak egy részhalmazát használja arra, hogy becslést adjon a gradiensre. Ez az algoritmus kiválaszt $X_1, X_2, \cdots X_m$ elemeket a tanító halmazból, ezekre kiszámolja a ∇C_{X_j} gradienseket, majd ezek $\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m}$ átlagát használja ∇C helyett, egyfajta becslésként. Az egyszerre kiválasztott m elem halmazát egy **köteg**nek (batch) nevezzük. Ha a kötegek mérete elég nagy, akkor ez a véletlenített algoritmus jól közelíti a költségfüggvény minimum helyét.

2.2. A mély tanulásról általában

Az utóbbi évtizedekben a rendelkezésre álló számítógépes infrastruktúra fejlődése egyre bonyolultabb hálózatok használatát tette lehetővé. Ahogy idővel a tanító adatok száma is gyarapodott, egyre hasznosabbnak bizonyult az összetettebb modellek használata.

2.1. Definíció. *Azokat az előrecsatolt neurális hálózatokat, amelyekben a neuronok legalább három szintbe rendezve helyezkednek el,* ***mély hálózatok****nak nevezzük.*

A mély hálózatok fejlesztése az 1980-as években kezdődött, és azóta folyamatosan egyre jobb és jobb eredményeket érnek el egyre több alkalmazási területen.



4. ábra. Mély hálózat három rejtett réteggel [18]

A legelső mély modelleket pontosan körülvágott, kis méretű képeken lévő alakzatok felismerésére tudták megtanítani. [24] A mai képfelismerő algoritmusok már nagy felbontású képeken is képesek a felismerésre, függetlenül a minta elhelyezkedésétől. Hasonló fejlődést mutat, hogy amíg a korai verziók csak kétféle tárgyat tudtak felismerni (esetleg egy tárgy jelenlétét vagy hiányát megállapítani), addig a modern hálózatok már 1000 különböző alakzatot is képesek lokalizálni. A képfelismerésen kívül nagyon hatékonynak bizonyultak például a beszédfelismerésben, illetve olyan játékokban, mint a sakk vagy a Go.

2.3. A mély tanulás előnyei

Annak ellenére, hogy átütő sikereket értek el számos területen, a mély tanuló algoritmusok elmélete még sincs részletesen kidolgozva, emiatt keveset lehet tudni arról,

hogy miért is működnek ilyen hatékonyan. Bizonyos hálózati paraméterek meghatározása csupán kísérletek eredménye.

A mély neurális hálózatok sikerének egyik lehetséges oka az, hogy a rétegek magas száma lehetővé teszi, hogy a tanulandó fogalmak reprezentációjának egyfajta hierarchiája jöjjön létre. Így egy bonyolult feladat megoldását egyszerűbb részfeladatmegoldásokból fejezi ki a hálózat. Egy szemléletes példa erre a képfelismerés, hiszen a függvény, ami a pixelekből álló képet a rajta szereplő tárgy(ak) kódjára képezi le, nagyon bonyolult. Egy megfelelően tréningezett mély hálózat a kívánt leképezést egyszerűbb függvényekre bontja, melyeket egy-egy réteg ír le. A rétegek sorozata egyre absztraktabb tulajdonságait írja le a bemeneti képnek. Az első réteg könnyedén megtalálja a képen az éleket szomszédos pixelek összehasonlításával. Az innen továbbított adatok segítségével a második réteg felismeri a sarkokat, mint élek speciális együttesét. Hasonló elven, a tanító adatoktól függően harmadik szint már akár bizonyos formák (testrészek, állatok, stb.) konkrét részeit is észlelheti, és így tovább. Végül az utolsó szint a részek intenzitásából és egymáshoz viszonyított helyzetéből következtet egész tárgyak jelenlétére a képen. [10]

2.4. A mély tanulás hátrányai

Mély hálózat tréningezése során gyakran előfordul, hogy a költségfüggvény által felvett érték csökkenése néhány tanító iteráció után lelassul, vagy eleve nem olyan gyors, mint egy sekély hálózat esetén. Ezt gyakran a gradiens értékek instabilitása okozza: a túl kicsi és a túl nagy deriváltak is ellehetetlenítik a tanulást.

A gyakorlatban használt aktivációs függvények nagy részének deriváltja -1 és 1 közé esik, ez vezet az **eltűnő gradiensek** problémájához (vanishing gradient problem). Mivel a hiba-visszaterjesztési algoritmus láncszabállyal számítja a deriváltakat, ezért minden réteg esetén annyi $[-1,1]$-beli számot kell összeszoroznia, amennyire messze van az adott réteg a kimenettől. Emiatt a hiba a rétegek számában exponenciális mértékben csökken a bemenet felé, tehát az első néhány szintet alkotó neuronok tanulása (paramétereik változása) nagyon lelassul.

A gradiens instabilitásán az sem segít, ha olyan aktivációs függvényt választunk, amelynek 1-nél nagyobb abszolút értékű is lehet a deriváltja. Egy ilyen választás ugyanis egy másik, nagyon hasonló problémához, a **felrobbanó gradiensek**hez (exploding gradient problem) vezet. Ebben az esetben a bemenethez közeli szintek deriváltjai túl nagyok lesznek, így a későbbi szintek tanulása lesz aránytalanul lassú. A gradiensek instabilitásának ez a fajtája megelőzhető úgy, hogy egy előre meghatározott értéknél nem engedjük nagyobbra nőni a deriváltakat. Például ha egy rétegen a deriváltak valamilyen norma szerint meghaladják az általunk megadott

határt, akkor lenormálhatjuk őket. A deriváltak ilyen jellegű mesterséges lekicsinyítését gradiensvágásnak (gradient clipping) nevezzük.

Az eltűnő gradiens probléma kiküszöbölése speciális hálózati elemek alkalmazásával lehetséges. Ilyen például az LSTM (Long Short-Term Memory) [1], amely egy memóriaként működő rekurrens hálózat. Egy bemeneti kapu, egy saját magára kapcsolt neuron, egy felejtő kapu és egy kimeneti kapu alkotja. Az önrekurrens kapcsolat biztosítja, hogy az LSTM belső állapota változatlan maradhasson, ha erre szükség van. Maga az elem úgy van felépítve, hogy a hozzá tartozó derivált mindig 1 vagy ahhoz közeli érték, ezért nem csökkenti és nem is növeli nagy mértékben a gradienst.

Hátránya még a mély tanulásnak, hogy sokszor több adatra és több időre lenne szükség a megfelelő eredmény eléréséhez, mint amennyi rendelkezésre áll. Ha viszont túl sokáig tréningezzük a hálózatot, vagy rosszul állítjuk be a paramétereket, akkor könnyen előfordulhat a túltanulás jelensége: habár a tanító adathalmazon sikeresen lecsökken a hibafüggvény értéke, a kapott modell nem elég általános ahhoz, hogy korábban nem látott bemenetre is jól teljesítsen.

Minél több réteg és neuron van egy hálózatban, annál több úton juthat el adat a bemenetből a kimenetbe. Ez már néhány szint esetén is rengeteg irány jelent, a továbbadott értékeken túl ezáltal is lehetséges lenne információt kódolni. Ezt azonban a jelenleg használt neurális hálózatok túlnyomó része egyáltalán nem használja ki, az adatfolyam-kiválasztás a tervező számára átláthatatlan.

2.5. Néhány példa

2.5.1. Többrétegű perceptron

A neurális hálózatok egyik legrégebbi és legegyszerűbb fajtája a többrétegű perceptron modell.

2.2. Definíció. *A **többrétegű perceptron** egy olyan előrecsatolt hálózat, amelynek minden neuronja egy-egy általánosított perceptron. Hálózati topológiája **teljesen összekötött** rétegekből álló irányított gráffal írható le, ami azt jelenti, hogy az utolsó kivételével minden szint minden eleméből irányított él indul a következő szint minden elemébe. Rétegeinek száma legalább három: egy bemeneti, egy kimeneti, és legalább egy rejtett szint alkotja.*

Bizonyos feladatok megoldására már ez az architektúra is megtanítható, mint például valamilyen minta felismerése egy képen. Egy felügyelt tanítással tanított háromrétegű perceptron hálózat 98%-os pontossággal képes kézzel írott számjegyek

kategorizálására [18]. Hátránya, hogy nem veszi figyelembe a vizsgált képek tartalmának térbeli elrendeződését, például az egymástól távol lévő pixeleket ugyanúgy kezeli, mint az egymáshoz közelieket.

2.5.2. Konvolúciós hálók

A gyakorlatban képfeldolgozásra (és minden, háló-szerű topológiával rendelkező adat feldolgozására) a többrétegű perceptron modellnél valamivel bonyolultabb **konvolúciós hálózat (CNN, [18])** bizonyult a leghatékonyabbnak. A konvolúciós hálók biológiai inspirációja bizonyos állatok, többek között majmok agyának látókérge. Itt megfigyelhető az a jelenség, hogy egy-egy neuron a látómező egy-egy részére reagál aktivitásával. Ha a majom szemei nem mozognak, akkor az egy neuronhoz tartozó terület az állat látóteréhez viszonyítva állandó helyen található, ez a terület a neuron receptív mezője. A konvolúciós hálózat definiálásához be kell vezetnünk a konvolúció és az összevonó (pooling) réteg fogalmát.

A konvolúció művelete széles körben használatos; értelmezhető tömbökre, függvényekre és eloszlásokra is, létezik folytonos és diszkrét, illetve egy- és többdimenziós változata is. A statikus képfeldolgozásban az adatokat általában 2 dimenziós tömbökkel (mátrixokkal) ábrázoljuk, ezért most csak a kétdimenziós diszkrét konvolúció definíciójára lesz szükség.

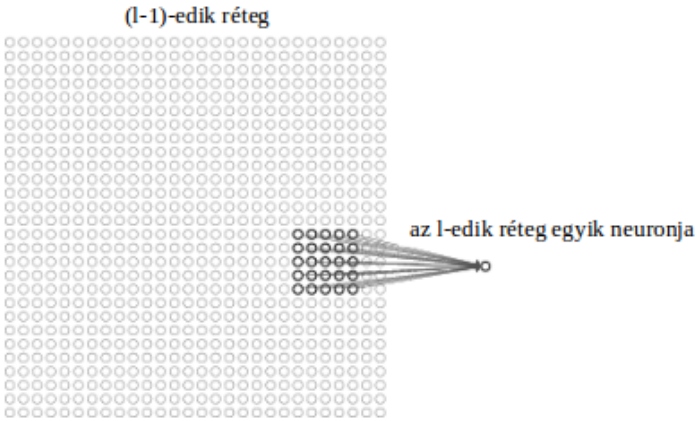
2.3. Definíció. *Legyen A egy $m_A \times n_A$ alakú, B pedig egy $m_B \times n_B$ alakú mátrix. Az indextartományokon kívül eső elemeket definiáljuk nullának. Ekkor A és B 2 dimenziós diszkrét konvolúciójának eredménye az a C mátrix, melyben az i-edik sor j-edik eleme*

$$C(i,j) = \sum_{m=0}^{m_A+m_B-1} \sum_{n=0}^{n_A+n_B-1} A(m,n)B(i-m,j-n).$$

A konvolúciós hálózatok terminológiájában az első argumentumra a „bemenet”, a másodikra a „szűrő” vagy „kernel”, az eredményre pedig a „tulajdonságtérkép” vagy „tulajdonságsík” (feature map) elnevezés használatos. A konvolúció szemléletes leírása a következő: a kernelt vízszintesen és függőlegesen is tükrözzük, majd ”végigtoljuk” a bemeneten, minden lépésben elvégezve a kernel és a bemenet éppen fedett része közötti skaláris szorzást.

A neurális hálózatok nyelvére lefordítva a konvolúció művelete egy olyan, perceptronokból álló alrétegnek felel meg, amelynek elemei az előző rétegből $m_B \cdot n_B$ darab neuronnal vannak összekötve, továbbá közös súlyokkal és eltolásértékekkel

rendelkeznek. Mint ahogyan azt később látni fogjuk, a konvolúciós hálózat egy rétegét több ilyen alréteg alkotja. Az alkalmazásokban a kernel mérete általában jóval kisebb, mint a bemeneté, így minden neuron az előző rétegnek csak egy kis tartományából gyűjt információt. Ez az $m_B \times n_B$ méretű, téglalap alakú terület felel meg a receptív (vagy más néven felfogó) mezőnek.



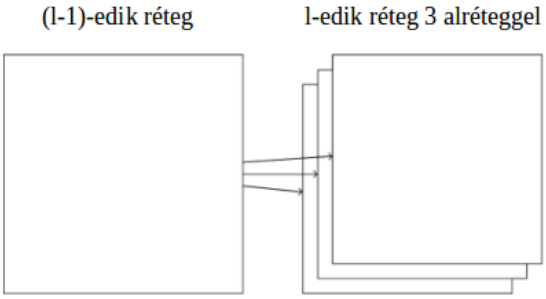
5. ábra. Egy 28×28-as bemeneten elvégzett konvolúciós alréteg egy neuronja és annak receptív mezeje 5×5-ös szűrőmérettel. [18]

Előfordul, hogy a neurális hálózat a konvolúciót valamilyen változtatással alkalmazza. Jellemző például, hogy a szűrőt nem egy pixeles lépésekkel toljuk végig a képen, azaz nem mindenhol, hanem például csak egy 3x4-es téglalap alakú rács pontjaiban végzünk skaláris szorzást. Egy másik változtatási lehetőség, hogy csak a kép széléig, vagy onnan csak valamilyen előre meghatározott távolságra megyünk el a szűrővel. Az alkalmazott lépésköz (stride), a szűrő mérete, és az, hogy meddig csúsztatjuk a szűrőt (padding), a hálózat (azon belül a konvolúciót megvalósító alréteg) rögzített paraméterei, ezek megválasztása az adott problémától függ.

A konvolúcióban alkalmazott szűrő értékeire pixelként tekintve maga a szűrő is valamilyen kisméretű mintának felel meg. Amikor végigcsúsztatjuk a bemeneti képen, akkor a skaláris szorzások eredménye – a tulajdonságtérkép – arról hordoz információt, hogy a szűrő által reprezentált alakhoz hasonlóak szerepelnek-e a képen, és ha igen, akkor hol. Ha még egy réteget alkalmazunk, azaz a tulajdonságtérképet konvolváljuk valamilyen szűrővel, akkor az előző szinten megszerzett adatok közötti összefüggéseket vizsgáljuk, tehát az első szint szűrői által ábrázolt képrészek egymáshoz viszonyított helyzetéről hordoz a következő szint információt.

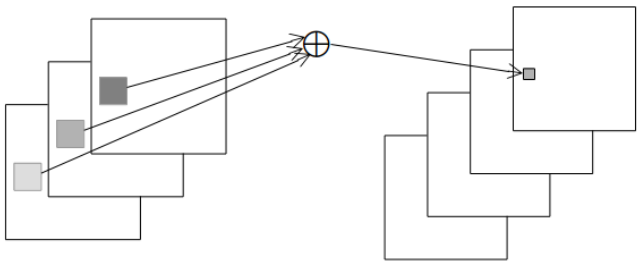
A bemenő élek súlya és az eltolásérték egy alrétegen belül neurononként megegyezik, hiszen a kernel nem változik a konvolúció folyamán. Ezáltal a neuronok kényszerítve vannak, hogy ugyanazokat a tulajdonságokat reprezentálják a kép különböző részein. A **megosztott súlyok** alkalmazása nagy mértékben csökkenti a tanítandó szabad paraméterek számát a teljes összeköttetéshez képest, illetve a súlyok által elfoglalt memóriaterület is kevesebb lesz. [3] A konvolúciós alréteg egy olyan reprezentációját adja a képnek, amely ekvivariáns az eltolásra: egy, a bemeneti képen lévő tárgy eltolása ugyanolyan mértékű eltolást eredményez a kimenetben. A forgatásra, skálázásra és egyéb transzfomációkra ez nem teljesül. Miután a hálózat detektál egy tulajdonságot, annak a pontos pozíciója már nem kerül továbbításra, csak a többi tulajdonsághoz viszonyított helyzete hordozza az információt.

Egyetlen szűrő csak egyfajta tulajdonságot detektál, a képfelismeréshez azonban minden szinten sok ilyen szükséges. Egy konvolúciós rétegen belül emiatt több alréteg kap helyet, amelyek mind külön tulajdonságtérképeket hoznak létre. Ha egy több alrétegből álló konvolúciós réteget alkalmazunk, akkor a 2 dimenziós adatból 3 dimenziós adat jön létre, úgynevezett csatornákat alkotva: a harmadik dimenzió mérete a csatornák száma.



6. ábra. Egy 3 alrétegből álló konvolúciós réteg sematikus rajza [18]

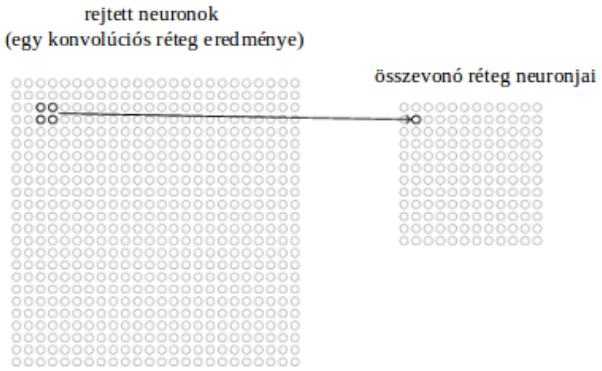
Színes képek feldolgozásánál gyakori, hogy már a bemenő adat is 3 csatornás, ugyanis a piros, a zöld és a kék színkomponenseket külön képeken szokás megadni a hálózatnak. Abban az esetben, amikor a konvolúció bemenete is több (c_1 darab) csatornából áll, a kimenet minden egyes csatornája c_1 tulajdonságtérkép összege. A bemeneti csatornák nem osztják meg a szűrőiket, minden (bemeneti csatorna, kimeneti csatorna) párra különböző filter vonatkozik. A kimeneti csatornák számát c_2-vel, a bemeneti csatornákat $x = (x_1, \dots x_{c_1})$-gyel, a kimeneti csatornákat $y = = (y_1, \cdots y_{c_2})$-vel jelölve tehát minden (x_i, y_j) párnak megfelel egy $f_{i,j}$ szűrő, és a j-edik kimeneti csatorna értékeit az $y_j = \sum_{i=1}^{c_2} x_i * f_{i,j}$ képlet határozza meg ($i = = 1 \dots c_1, j = 1 \dots c_2$), ahol $*$ a konvolúció műveletét jelöli.



7. ábra. Egy 4 alrétegből álló konvolúciós réteg egyetlen neuronjának számítási menete 3 csatornás bemenetre alkalmazva. A neuron receptív mezőjére a 3 csatornán külön-külön szűrőket alkalmazunk (ezeket az ábrán különböző árnyalatú szürke négyzetek jelölik), majd ezek eredményeit összeadjuk ($\oplus$), így kapjuk a kimenet egy csatornájának egy elemét.

A konvolúciós hálózatok a konvolúciós rétegen kívül **összevonó** (pooling) vagy almintavételező (subsampling) rétegeket is tartalmaznak, amelyek fő funkciója az információ leegyszerűsítése (a réteg méretének csökkentése) és az adatfolyam irányának kiválasztása.

Az összevonó réteg felépítése nagyon hasonló a konvolúciós rétegéhez; egy-egy neuronnak itt is megfeleltethető az előző rétegből egy-egy téglalap alakú terület, amellyel össze van kötve.



8. ábra. Egy összevonó réteg 2×2-es receptív mezővel [18]

Ezek a receptív mezők egyforma méretűek, valamilyen állandó lépésközzel követik egymást, és előre meghatározott mértékben lóghatnak le az előző réteg aktivitásai által definiált kép vagy tulajdonságtérkép széléről.

Egy általános pooling réteg neuronjai valamilyen összefoglaló statisztikát készítenek a receptív mezőjük kimeneteiből. Egy max pooling réteg például egy-egy téglalap alakú területről csak a legnagyobb értéket továbbítja, az L_2 pooling pedig az aktivitások négyzetösszegének négyzetgyökét számítja ki. Ha egy olyan konvolúciós réteg után alkalmazunk összevonást, amely több tulajdonságtérképet generált, akkor a tulajdonságtérképekre külön-külön értendő a művelet, ezért a csatornák számán ez a réteg nem változtat.

Ha a receptív mezők közti lépésköz 1-nél nagyobb, akkor a kimeneti méret kisebb lesz a bemenetinél, így ezek az összevonó rétegek azokban az esetekben is hasznosak, amikor különböző méretű bemeneteket kell kezelnie a hálózatnak. Előfordulhat például, hogy képfelismerésre szeretnénk tréningezni egy modellt, de a tanító adathalmaz elemei nem azonos méretű képek. A bemeneti réteg alakja bármilyen hálózati topológia esetén rögzített, így ilyenkor egy lehetséges előfeldolgozási lépés összevonó rétegekkel egységes méretűre hozni a képeket.

A felügyelten tanított konvolúciós neurális hálózatok az eddig bemutatott elemekből általában a következő módon épülnek fel. Az első réteg egy konvolúciós réteg, ezt valamilyen nemlineáris aktivációs függvény követi, majd egy összevonó réteg. Ez a felépítés ismétlődik néhányszor, majd az utolsó összevonó réteget teljes összeköttetéssel, azaz perceptron-réteggel kapcsoljuk a kimenetre.

A fent leírt konvolúciós hálózatra ad általánosabb alternatívát a **G-konvolúciós hálózat (G-CNN)** [7], amely rétegei az eltolás helyett, vagy azon felül további transzformációkra is ekvivariánsak. Ebben a modellben egy K csatornás képet $\mathbb{Z}^2$-ből $\mathbb{R}^K$-ba menő (véges tartójú) függvényként kezelünk. Egy objektum *szimmetriájá*nak nevezünk minden olyan transzformációt, amely az objektumot önmagába viszi. Bármely g és h szimmetriára a gh kompozíció és a g^{-1} inverz is szimmetria, továbbá g és g^{-1} kompozíciója az identitás. Egy adott objektum szimmetriái tehát csoportot alkotnak. Tekintsük $\mathbb{Z}^2$ szimmetriáinak egy G csoportját, ilyen például a $p4$ nevű csoport, amelynek elemei az eltolások, a 90°-os középpontos forgatások bármilyen középpont körül, illetve ezek kompozíciói. G minden g elemére definiálhatunk egy L_g transzformációt képeken a következőképpen:

$$[L_g f](x) = f(g^{-1}(x)),$$

ahol f a bemeneti kép. A transzformáció eredménye szintén egy kép, tehát $\mathbb{Z}^2$-ből $\mathbb{R}^K$-ba menő leképezés.

Ezen fogalmak segítségével írhatók le a G-konvolúciós hálózat rétegei: a G-konvolúció, a G-pooling és a nemlinearitás, amelyek mind ekvivariánsak G elemeire. A **G-konvolúció** művelete nagyon hasonló a hagyományos konvolúcióhoz, annyiban tér el tőle, hogy a szűrőt nem tüközzük, és eltolás helyett egy G-beli transzformációt alkalmazunk rá. A K csatornából álló $f = (f_1, \dots f_K)$ bemeneti képre a $\psi : \mathbb{Z}^2 \to \mathbb{R}$ szűrővel való konvolúció eredményét az

$$[f * \psi](x) = \sum_{y \in \mathbb{Z}^2} \sum_{k=1}^{K} f_k(y)\psi(x - y)$$

képlet adja, míg a G-konvolúció eredménye

$$[f \star \psi](g) = \sum_{y \in \mathbb{Z}^2} \sum_{k=1}^{K} f_k(y)\psi(g^{-1}y).$$

A G-konvolúció tehát nem egy $\mathbb{Z}^2$-n értelmezett függvényt ad, a keletkező tulajdonságtérkép értelmezési tartománya G. Az elsőtől különböző rétegeken emiatt a szűrők is G-n értelmezett függvények lesznek, a konvolúció képlete pedig a következőre módosul:

$$[f \star \psi](g) = \sum_{h \in G} \sum_{k=1}^{K} f_k(h)\psi(g^{-1}h).$$

A G-konvolúciós réteget egy **G-pooling** réteg követi. Ebben az esetben valamilyen $f : G \to \mathbb{R}$ tulajdonságtérképen belül a receptív mezők G részhalmazai, a P összevonó operátor hatása egy rögzített U receptív mezőre pedig a

$$Pf(g) = \max_{k \in gU} f(k)$$

formulával írható le, ahol $gU = \{gu | u \in U\}$. A receptív mező eltolásának az felel meg, hogy U helyett G-nek egy H részcsoportját választjuk, és azon végezzük P-t.

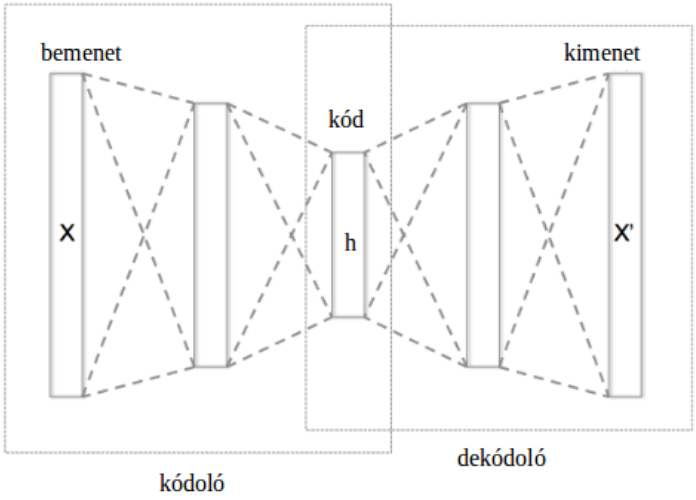
Egy $\nu : \mathbb{R} \to \mathbb{R}$ nemlineáris aktivációs függvényt úgy építhetünk be a G-konvolúciós hálózatba, hogy bevezetjük a C_ν **kompozíciós operátor**t, amelyre $C_\nu = \nu(f(g))$ teljesül f tulajdonságtérkép esetén.

A G-konvolúcióra belátható, hogy ekvivariáns G elemeire, azaz minden $u \in G$-re, illetve minden f tulajdonságtérképre és ϕ szűrőre $[[L_u f] * \psi](g) = [L_u[f * \psi]](g)$ teljesül. A G-pooling csak annak a H részcsoportnak az elemeire invariáns, amelyet receptív mezőnek választottunk.

2.5.3. Autoenkóderek

A felügyelet nélküli tanítás egyik eszköze az **autoenkóder** neurális hálózat [10]. Ez egy speciális alakú előrecsatolt hálózat, amelyet az identitás függvényre tanítunk, azaz a megfelelően tréningezett autoenkóder a bemenetét a kimenetre másolja. A költségfüggvény egyszerűen a kimenet és a bemenet eltérését méri, a tanítás ezt minimalizálja. Egy szimpla másoló hálózat persze önmagában nem hasznos, a végső cél az, hogy egy kiemelt szerepű rejtett réteg h aktivációi a bemenő adat lényeges tulajdonságait reprezentálják a tanítás után.

Az autoenkóder hálózati struktúra két részből áll: x-szel jelölt bemenet esetén egy kódoló (encoder) f függvényt megvalósító hálózatrész adja a $h = f(x)$ kódot, és egy dekódoló g függvénynek megfelelő hálózatrész készíti el ebből az $x' = g(f(x))$ rekonstrukciót.



9. ábra. Autoenkóder hálózati struktúra

A kódoló és a dekódoló általában ugyanannyi rétegből áll, és egymásnak valamilyen értelemben vett tükörképei. Azt, hogy h egy megfelelően kifejező kódját adja a bemenetnek, úgy érhetjük el, hogy a h-t adó réteg méretét kisebbre választjuk x méreténél. Ez a mesterséges dimenziócsökkentés rákényszeríti a hálózatot, hogy csak a meghatározó tulajdonságokat közvetítse, és azokból állítsa elő a bemenet mását. A tömörítésnek ezen formája olyan bemenetek kezelésekor jön jól, amelyeket változékony, zajos, sok dimenziós adatként kap meg a hálózat, de jól leírhatók kevés paraméterrel. Például az emberi hang is magas dimenziójú, nagyon variábilis,

mert ahány ember, annyiféle hangképző szerv létezik. Azt viszont, hogy mit mond a beszélő, néhány testrész állapotából (a nyelvnek a szájpadláshoz viszonyított helyéből, a száj nyitottságából, a lágy szájpadlás állásából, stb.) el lehet dönteni. A meghatározó tulajdonságok kiválasztásának másik alkalmazása a zajszűrés. Ebben a feladatban egy olyan $\tilde{x}$ inputot kap a hálózat, amely valamilyen x adatból ismeretlen zaj hozzáadásával született, a feladat pedig a "tiszta" x meghatározása, azaz az eredeti adat visszaállítása. Az autoenkóder hálózatoktól tehát valójában nem várjuk el, hogy a rekonstrukció tökéletesen megegyezzen az inputtal.

Egy példa autoenkóder hálózatok használatára félig felügyelt tanításban az **egymásra kapcsolt autoenkóderek rendszere** (stacked autoencoder, [2]), amely mély neurális hálózatok előtréningezésére alkalmas. Ha egy L rétegből álló előrecsatolt mély hálózatot tréningezünk, akkor a modell L darab sorba rendezett autoenkóderből áll, melyek kódolói a mély hálózat rétegeinek felelnek meg, azaz az i-edik kódoló a mély hálózat i-edik rétege, az i-edik dekódoló pedig annak valamilyen értelemben vett tükörképe. Az első autoenkóder bemenete a mély hálózat bemenete, a többié pedig az őt megelőző autoenkóder kódolójának kimenete. A mély hálózat előtréningezése rétegenként történik. Először az első autoenkódert tréningezzük arra, hogy minél jobban visszaadja a hálózat tanító adathalmazának elemeit. Ezután a már betréningezett első kódoló kimeneteinek rekonstrukciójára tréningezzük a másodikat, és így tovább. A mély hálózatot a felügyelt tanítás előtt az így kapott súlyokkal és eltolásértékekkel inicializáljuk.

Olyan autoenkóderben, ahol a kódoló minden rétegének egyértelműen megfelel a dekódoló egy rétege, a tanítási folyamatot gyorsíthatja, ha az így kapott párok tagjai között is futnak hálózatrészek. Az ilyen, keresztbe menő összeköttetésekkel kiegészített autoenkódert **létrahálózatnak (ladder network, [26])** nevezzük. Előnye, hogy a bemenettől és a kimenettől egyaránt távol eső középső rétegeknek nem kell az adat minden részletét leírniuk, ehelyett absztrakt tulajdonságok reprezentációját tanulják meg. Például egy képfeldolgozó létrahálózat középső rétegei csak azt kódolják el, hogy milyen formákat ábrázol a kép, és azok egymáshoz képest hogyan helyezkednek el, a formák pontos részleteit csak a hálózat első és utolsó néhány rétege kódolja. A létrahálózatot az egyszerű autoenkódertől az is megkülönbözteti, hogy minden kódoló-dekódoló összeköttetéssel egy-egy új tag kerül a hálózat költségfüggvényébe, méghozzá az érintett rétegek kimeneteinek valamilyen mérték szerinti eltérése.

A képfeldolgozásban használt egyik speciális autoenkóder struktúra a **konvolúciós autoenkóder**, amelynek kódolója egy konvolúciós hálózat. A dekódoló hálózatrész úgy épül fel, hogy minden kódolóbeli rétegnek van itt egy párja; a rétegek számát L-lel jelölve a kódoló (és egyben a hálózat) i-edik rétegének párja az $L - i +$

+ 1-edik dekódoló réteg. Habár a konvolúció és az összevonás nem invertálhatóak, a hálózat inicializálásakor és tanításakor törekszünk arra, hogy minden szint által megvalósított művelet hatása a párjáéval ellentétes legyen. A konvolúciós rétegek párjait **dekonvolúciós réteg**eknek nevezzük. A konvolúció pontos invertálásához az $y_j = \sum_{i=1}^{c_1} x_i * f_{i,j}$ $(j = 1 \dots c_1)$ egyenletrendszert kellene megoldanunk az y_j értékek ismeretében az $f_{i,j}$ és x_i ismeretlenekre. Általában ez egy alulhatározott rendszer, így közelítő módszerrel keresünk egy megoldást. Ehhez definiálunk egy költségfüggvényt a következőképpen: $C(y) = \frac{\lambda}{2} \sum_{j=1}^{c_2} \| \sum_{i=1}^{c_1} x_i * f_{i,j} - y_j \|_2^2 + \sum_{i=1}^{c_1} |x_i|^p$, ahol p és λ előre meghatározott konstansok, általában $p = 1$. A költségfüggvény második tagja azért felel, hogy x_i megfelelően ritka mátrix legyen, mivel az alakfelismerésben ilyen adatok a jellemzőek. A λ érték határozza meg az összeg tagjainak relatív hozzájárulását a költségfüggvényhez. Az autoenkóder hálózat tréningezése során a dekonvolúciós réteg által inputként megkapott $Y = y^1, \dots, y^I$ tulajdonságtérképeket felhasználva keressük azon x_i képeket és $f_{i,j}$ szűrőket, amelyek minimalizálják a költségfüggvény értékét. A dekonvolúciós réteg kimenete tehát az így kapott (minden y-ra közös) $x = (x_1, \dots, x_{c_0})$ c_0 csatornából álló kép. [31]

Az összevonó rétegek párjait **unpooling** rétegeknek nevezzük. Ezekre szintén igaz, hogy bemeneti és kimeneti méretük a párjukéval ellentétes. Az unpooling réteg az őt megelőző réteghez úgy kapcsolódik, hogy az előző réteg minden neuronja csak az unpooling réteg egy téglalap alakú részén található neuronokkal van összekötve. Ezek a területek a réteg párja által meghatározott receptív mezők. Az unpooling réteg neuronjai által megvalósított művelet többféle is lehet, függ például attól is, hogy a párja milyen összevonást végez. Ha max pooling szerepel a kódolóban, akkor a program minden receptív mezőre megjegyzi, hogy azon belül melyik helyen volt a maximum (ezt az információt switch-nek nevezzük), így az unpooling ezeken a helyeken álló neuronjai változtatás nélkül továbbítják a bemeneteiket, a többi pedig nullát ad kimenetül. L_2 (vagy más átlagoló) pooling esetén az unpooling réteg minden neuronja a bemenetét adja tovább. [32]

Az autoenkóderek közé tartozik még a **variációs autoenkóder (VAE)** [8], melynek népszerű alkalmazási területe az adatgenerálás, azaz egy adathalmaz segítségével annak elemeihez hasonló objektumok létrehozása. A modell megfelelő tréningezés után képes például kézzel írott számjegyek vagy fikcionális arcképek generálására. A variációs autoenkóderek működése azon a megfigyelésen alapszik, hogy bármilyen d dimenziós eloszlás előállítható úgy, hogy egy megfelelően bonyolult függvényt alkalmazunk d darab normális eloszlású változóra. Ezért a kódolót rákényszerítjük a tanulás során, hogy a kimeneti rétegének elemei közel normális eloszlást kövessenek. A dekódolóra tekinthetünk úgy, mint egy olyan hálózatra, amely tréningezés után az első néhány szintjével a bemeneti, normális eloszlású adatokból rejtett változókat állít elő, amelyek leírják a generálni kívánt adatot, majd a további

rétegeivel elkészíti a kimenetet.

10. ábra. Képzeletbeli híresség arcképei variációs autoenkóderrel generálva [22]

A variációs autoenkóder dekódolójának a következő feladatot kell megoldania. Legyen adott néhány X elem valamilyen $\mathcal{X}$ vektortérből, ezek a rendelkezésre álló adatpontok. Adott ezen kívül egy $\mathcal{Z}$ vektortér, amelyből valamilyen P eloszlás szerint mintavételezhetünk, és egy $\mathcal{X}$-be képező $f_\theta(z)$ determinisztikus függvény, ahol θ paraméter. Rögzített θ-ra $f_\theta(z)$ tehát egy valószínűségi változó $\mathcal{X}$ fölött, ennek segítségével definiáljuk a $P_\theta(X|z) = \mathcal{N}(f_\theta(z), \sigma^2 \cdot I)$, azaz $f_\theta(z)$ várható értékű, $\sigma \cdot I$ szórású eloszlást, ahol σ előre adott konstans, I pedig az identitásmátrix. Ekkor egy olyan optimális θ értéket keresünk, amellyel $f_\theta(z)$ nagy valószínűséggel hasonló lesz az előre adott adatmintákhoz, azaz a

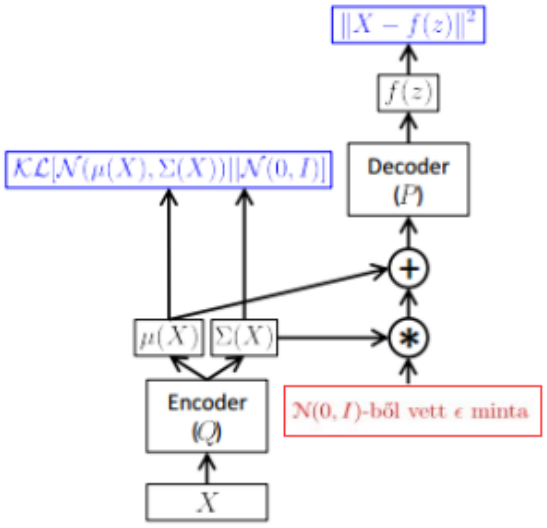
$$P(X) = \int P_\theta(X|z)P(z)dz$$

egyenletet oldjuk meg, azzal a feltételezéssel élve, hogy ha a modell nagy valószínűséggel generálja az ismert adatokat, akkor azokhoz hasonlóakat is nagy eséllyel ad. Egy neurális hálózat esetén f-et a dekódoló hálózati topológiája határozza meg, θ-t pedig a súlyok és eltolásértékek.

A kódoló szerepe, hogy olyan z vektorokat adjon kimenetnént, amelyek nagy valószínűséggel az ismert X adatpontokat generálják a dekódolón keresztül. Tehát valamilyen $Q(z|X)$ eloszlást valósít meg, amellyel $P(z|X)$-et közelíti. A tréningezés során feltesszük, hogy a valódi $P(z|X)$ normális eloszlást követ diagonális kovarianciamátrixszal, ezért $Q(z|X)$-et is ilyen formában keressük, azaz $Q(z|X) = = \mathcal{N}(\mu_\theta(X), \Sigma_\theta(X))$, ahol μ és Σ determinisztikus függvények, és a θ paraméter

értéke változik a kódoló tanítása során. A dekódoló bemenetének a kódoló kimenete által adott eloszlásból vett mintának kell lennie, de a véletlen mintavételezés nem deriválható, így a hiba-visszaterjesztő algoritmust nem tudnánk használni, ha a hálózatban lenne ilyen réteg. Ezért a kódoló után beépített mintavételezés helyett a hálózat bemenetét kiegészítjük egy $\mathcal{N}(0, I)$-ből vett ϵ vektorral, és a kódoló $\mu(X)$, $\Sigma(X)$ kimeneteivel skálázva számítjuk belőle a dekódoló bemenetét: $z = \mu(X) + + \Sigma^{\frac{1}{2}}(X)\epsilon$.

A variációs autoenkóder költségfüggvénye két tagból áll. Az egyik cél, hogy a kódoló kimenetéből kapott normális eloszlás minél közelebb legyen a standard normális eloszláshoz, ezt Kullback–Leibler divergencia [28] használatával mérjük. A másik minimalizálandó mennyiség a hálózat bemenetének és kimenetének négyzetes eltérése.



11. ábra. Variációs autoenkóder struktúra. A kék téglalapok a költségfüggvény tényezőit tartalmazzák. [8]

A betréningezett variációs autoenkóder jól használható adatgenerálásra. Ehhez a kódolóra már nincs szükség, egy $\mathcal{N}(0, I)$-ből vett mintán futtatjuk a dekódolót.

3. Transzformációtanulás

A mesterséges neurális hálózatok számos alkalmazási területén jellemző, hogy az adat különböző jellemzői egymástól függetlenül változnak. Ezért kétféle adatot kell

kezelnie a modellnek. Az egyik fajta adat az, ami a konkrét feladat megoldásához szükséges, a másik pedig azon tulajdonságokat írja le, amelyek változására invariáns a probléma. Ezek keveredése nagyban megnehezíti a hálózat feladatát.

Tekintsünk például egy olyan alkalmazást, ahol egy repülő és saját tengelye körül forgó tárgyat ábrázoló videófelvétel elemzése a cél. Ekkor a bemenet egy nagyon magas dimenziós tér eleme, a dimenziószám az egy képkockát alkotó pixelek száma szorozva a videó hosszával másodpercekben mérve és a másodpercenként megjelenő képek számával. Ha szét tudnánk választani a rendszert leíró lineáris faktorokat (mint például a hely és a mozgás koordinátái: sebesség, szögsebesség; a nézőpont; a megvilágítás iránya; stb.) a többitől (pl. maga a tárgy és a háttér kinézete), akkor a bemenet leírásához jóval kevesebb dimenzió is elegendő lenne, azaz egyszerűsödne a probléma.

Egy másik példa lehet a képfeldolgozás, azon belül a kézzel írott számjegyek felismerése. Az, hogy melyik jegyet tartalmazza a kép, invariáns az eltolásra, forgatásra, nagyításra, tükrözésre, de még a kézírás stílusára is. Hasonló példákat találhatunk a hangfeldolgozás területén: az emberi beszéd is tartalmaz olyan elemeket, amelyek nem szükségesek a mondandó megértéséhez (hangmagasság, hangerő, stb.).

A feladat, amelyről a továbbiakban szó lesz, olyan neurális hálózat készítése, amely minél több, egy adott problémára invariáns faktort tud leválasztani a bemenő adatból. Elsősorban a képelemzésen lesz a hangsúly, azaz olyan modelleken, amelyek el tudják választani a bemeneti kép tartalmát leíró információt az invarianciáktól. Például számjegyek esetén definiáljuk normalizált helyzetnek azt, ha a pixelek súlypontja a kép középpontjával egybeesik, és nincs elforgatva a szokásos íráshoz képest. Ekkor ha egy olyan számjegyet ábrázol a bemeneti kép, amely egy normalizált számjegyre alkalmazott $T_1, T_2, \ldots T_k$ transzformációkkal kapható meg, akkor a hálózat kívánt működése, hogy olyan reprezentációját adja a bemenetnek, amelyben külön jelennek meg az ezen transzformációkat leíró paraméterek, és külön azok, amelyek azt kódolják, hogy melyik számjegy van a képen.

3.1. Kapcsolódó eredmények

3.1.1. Forgatás-, skálázás-, és eltolás-invariáns mintafelismerő rendszer

Ha egy adott feladatra irreleváns tulajdonságokat külön tudjuk választani a többitől, az számos alkalmazásban hasznos lehet. Valódi adatból egymás után vett minták sorozatára jellemző, hogy egyszerre csak egy tulajdonság változik, mint például egy emberi arcról készült videófelvétel egymást követő képkockáin az arckifejezés. Az invariáns faktorok leválasztását előtréningezési lépésként használva tehermentesíteni

tudjuk a végső feladatot megoldó hálózatot, hiszen olyan reprezentációját állíthatjuk elő a bemenetnek, amiben már nem szerepelnek a fölösleges paraméterek. Ez az ötlet áll Yüceer és Oflazer forgatás-, skálázás-, és eltolás-invariáns mintafelismerő rendszere mögött is [30]. Ők nem neurális hálózattal végezték a kép előfeldolgozását, hanem egyszerű, előre definiált függvényeket alkalmaztak a bemenetre. Az általuk bemutatott normalizáló előfeldolgozás három lépésből áll. A *T-blokk* felel az eltolásinvarianciáért: kiszámolja a pixelek súlypontját, és olyan eltolást alkalmaz, hogy a súlypont a kép közepére kerüljön. Ezután az *S-blokk* úgy skálázza a képet a középpontból, hogy az 1 értékű pixelek átlagos sugara a kép szélességének negyede legyen. Végül a forgatásért felelős *R-blokk* Karhunen–Loève transzformáció [25] segítségével úgy forgatja a képet, hogy az 1 értékű pixelek legnagyobb szórásának iránya egybeessen az x-tengellyel. Az így kapott képet adták aztán inputként egy egyszerű alakzatok klasszifikációjára tréningezett hálózatnak.

3.1.2. Rekurrens Figyelemmodell

Egy másik alkalmazás lehet például egy probléma részekre bontása: ha egy kép több számjegyet is tartalmaz, akkor ezek helyzetének meghatározása után elég egy-egy olyan hálózatot futtatnunk, amely egyetlen számjegy felismerésére alkalmas, nincs szükség ennél bonyolultabb struktúrára vagy külön tanításra. Az ilyen jellegű eljárásokra úgy is tekinthetünk, mint a modell figyelmének a megfelelő helyekre való irányítására. Ez a képelemzési módszer az emberi látáshoz is közel áll, ugyanis mi sem egyszerre dolgozzuk fel az egész látóteret. Ehelyett a látórendszerünk egy felszínes elemzés alapján választ ki érdekesnek talált területeket, és csak azokra fókuszálva végez részletes vizsgálatot, majd ezekből állítja elő a látottak egy belső reprezentációját. [23]

Hasonló megközelítéssel működik a **Rekurrens Figyelemmodell** (Recurrent Attention Model, RAM) [17], amely egy rekurrens képfeldolgozó neurális hálózat. Szekvenciálisan vizsgálja a bemeneti kép egy-egy területét, minden lépésben az addig szerzett információ és a konkrét megoldandó probléma alapján választja ki a következő feldolgozandó képrészletet. Az állóképen való alakfelismerésen kívül a modell videójátékok játékosának vezérlésére is alkalmas a képernyőn megjelenő mozgókép alapján. Egyetlen feldolgozási fázis (a t-edik) a következő lépésekből áll. A *szenzor*nak nevezett ρ hálózatrész megkapja az x_t bemeneti képet és annak l_{t-1} koordinátákkal meghatározott helyét, ezekből elkészít egy $\rho(x_t, l_{t-1})$ reprezentációt. Ennek egy lehetséges módja, hogy a kijelölt hely körüli valamekkora régiót részletesen elkódolja, a kép többi részét pedig az l_{t-1}-től vett távolságuk függvényében egyre kisebb részletességgel. Az így kapott reprezentációból az f_g-vel jelölt, $\theta_g = \{\theta_g^0, \theta_g^2, \theta_g^2\}$ paraméterekkel ellátott hálózatrész állít elő egy $g_t = f_g(x_t, l_{t-1}; \theta_g)$ tulajdonságvektort. A

modell folyamatosan számon tart egy *belső állapot*nak nevezett vektort, amely minden időpillanatban az addig gyűjtött információt foglalja össze. Ezen belső állapot frissítése a következő lépés, amelyet az f_h-val jelölt, θ_h-val paraméterezett hálózatrész végez: $h_t = f_h(h_{t-1}, g_t; \theta_h)$. A tulajdonságvektor és a belső állapot alapján eldönti a modell, hogy hova irányítsa a szenzort a következő lépésben (azaz milyen l_t koordinátákat adjon neki), és elvégez egy *környezeti lépés*t. Az l_t helyet sztochasztikusan választja az f_l-lel jelölt, θ_l-lel paraméterezett hálózatrész által parametrizált eloszlásból: $l_t \sim p(\cdot|f_l(h_t; \theta_l))$. Az, hogy mi a környezeti lépés, attól a konkrét feladattól függ, amire a modellt alkalmazzuk, de ezt is egy paraméteres eloszlásból választjuk, amelynek paraméterét egy másik hálózatrész, az f_a adja: $a_t \sim p(\cdot|f_a(h_t; \theta_a))$, ahol θ_a az f_a paramétere. Minden fázis után a hálózat kap egy r_{t+1} jutalmat, a cél ezek összegének, azaz az $R = \sum_{t=1}^{T} r_t$ érték maximalizálása. Képfelismerés esetén például $r_t = 1$, ha t fázis után már helyesen felismeri a tárgyat a képen, és 0, ha nem. Egy ilyen, vagy ehhez hasonló hálózatban a kép kiválasztott helyeinek sorozatát meghatározó rendszer működését tudná megkönnyíteni a helyparaméterek elválasztása a többi tulajdonságtól.

3.1.3. TAG

Egy másik, paraméter-elválasztást használó struktúra a **TAG** [11] hálózat, amely iteratív módon bontja koherens részekre, csoportokra a bemenetet és a belső reprezentációt is. A modell autoenkóder-alapú, és zajszűrésre tréningezzük, azaz a rendelkezésre álló adatokat zajjal terhelve adjuk bemenetül a hálózatnak, a kívánt kimenet pedig az eredeti adat, pontosabban egy eloszlás a lehetséges kimeneteken, ahol az eredeti adathoz nagy valószínűség tartozik. A tiszta bemenetet x-szel, zajjal terhelt változatát $\tilde{x}$-mal jelöljük. A rendszer T iterációt tesz, ahol T előre meghatározott konstans, a hálózat bemenete. Az i-edik iteráció kimenete $q_i(x)$, ami a $p(x|\tilde{x})$ posteriort kívánja közelíteni.

A hálózat a bemenet minden x_j elemére bevezet előre meghatározott számú (K darab) rejtett bináris változót, amelyek közül a k-adik, azaz $g_{k,j}$ értéke 1, ha az adott elemet a k-adik csoport generálta. A $g_{k,j}$ értékek által alkotott vektort g_k-val jelöljük, és az így adódó vektorok segítségével a $q_i(x)$ előáll K-tényezős összegként $q_i(x) = \sum_{k=1}^{K} q_i(x|g_k)q_i(g_k)$ alakban. A $q_i(x|g_k)$ valószínűségeket normális eloszlással modellezzük, z_k^i várható értékkel és v szórással, $q_i(g_k)$-t pedig egy diszkrét eloszlás adja, amelyhez tartozó valószínűségeket m_k^i jelöli. Ezeket a z_k^i és m_k^i paramétereket tréningezi a hálózat, v pedig előre meghatározott konstans. A minimalizálandó költségfüggvény $\mathrm{C(x)} = -\sum_{i=1} \log q_i(x)$.

A tréningezést egy különálló hálózati elem, a Tagger végzi, amely egy aszimmetrikus létrahálózat. Minden iterációban megkapja az előzőben kapott paramétereket

(i = 1-re az inicializációs értéket), illetve az ezekből számított $\delta z_k^i = \frac{\partial C(\tilde{x})}{\partial z_k^i}$ hibát és $L(m_k^i) = \frac{q_i(\tilde{x}|g_k)}{\sum_{h=1}^{k} q_i(\tilde{x}|g_h)}$ likelihood arányt. A TAG tréningezése során tehát közvetlenül ennek a hálózatnak a súlyai és eltolásértékei változnak. A megfelelően tanított hálózat a zajszűrésen kívül szegmentációra is alkalmas, mert egy képen szereplő különböző tárgyakat külön csoportnak felelteti meg, illetve bizonyos esetekben képes elválasztani a hátteret az előtértől.

3.1.4. AIR

Szintén külön kezeli jelentésük szerint a bemenetet leíró változókat az **Attend-Infer-Repeat (AIR, [9])** hálózat, amely képeken végez a variációs autoenkóderekhez hasonló inferenciát iteratív módon. A cél egy adott X bemenetre és θ-val parametrizált adott $P_\theta(X|z)P_\theta(z)$ modellre az adatot leíró z vektor meghatározása a $P(z|X) = \frac{P_\theta(X|z)P(z)}{P_\theta(X)}$ posterioron keresztül. A modell kihasználja, hogy a képek tartalma általában különálló objektumokból áll, így a z vektort z^i csoportokra bontva keresi, ahol minden csoport egyetlen tárgy tulajdonságait írja le.

Mivel a képen lévő minták száma képről képre változik, ezért az X-et generáló modellről feltesszük, hogy $p_\theta(X) = \sum_{n=1}^{N} P_N(n) \int P_\theta(z|n)P_\theta(x|z)dz$ alakú, azaz valamilyen rögzített, legfeljebb N értékű eloszlás szerint meghatározza az objektumok n számát, majd egy $P_\theta(.|n)$ eloszlásból mintavételezi a $z = (z^1, \dots z^n)$ változókat, amelyekből előállítja az $X \sim P_\theta(.|z)$ adatot. Az itt használt eloszlások és az N érték előre adottak, pontos megadásuk a konkrét feladattól függ. A z^i változócsoportok kétféle változót tartalmaznak: a z^i_{what} részhalmaz elemei írják le az adott tárgy alakját, identitását, a többi változó pedig z^i_{where}-t alkotja és a képen belüli helyzetet (orientációt, helyet, stb.) reprezentálja.

A valódi posteriort egy $q_\phi(z, n|X)$ eloszlással közelítjük, ahol ϕ a tanulandó paraméter, amely a $KL[q_\phi(z, n|X)||P_\theta(z, n|X)]$ Kullback–Leibler divergenciát minimalizálja. Hogy az objektumok számától függő szekvenciális feldolgozást egyszerűbbé tegyük, n-et egy változó hosszúságú z_{pres} vektorral helyettesítjük, amely n darab 1-est és egy lezáró 0 karaktert tartalmaz. A közelítő posterior így a következő formában írható fel:

$$q_\phi(z, z_{pres}|X) = q_\phi(z^{n+1}_{pres} = 0|z^{1:n}, X) \prod_{i=1}^{n} q_\phi(z^i, z^i_{pres} = 1|X, z^{1:i-1}).$$

q_ϕ ebben az esetben egy neurális hálózat, amely iteratív módon működik, és minden lépésben visszaadja a kapott (leíró változók terén értelmezett) eloszlás paramétereit, például normális eloszlás esetén a várható értéket és a szórásnégyzetet.

Az iteráció befejezésének pillanatát z_{pres} szabályozza. Ha a hálózat kimenetében $z_{pres} = 1$, akkor még legalább egy objektumot le kell írnia, ha $z_{pres} = 0$, akkor az iteráció leáll. Minden iterációban az aktuális z^i érték függ az előzőektől, ezáltal kizárjuk annak a lehetőségét, hogy a képnek ugyanazt az elemét több csoport is leírja. A rendszer a θ és ϕ paramétereket egyszerre tréningezi a $P_\theta(X)$-et alulról korlátozó $\mathcal{L}(\theta, \phi) = E_{q_\phi}[\log \frac{P_\theta(X,z,n)}{q_\phi(z,n|x)}]$ függvény maximalizálásával.

3.1.5. Transzformáló Autoenkóder

Az információ-szétválasztás felhasználható olyan neurális hálózat működtetéséhez is, amely nem csak két összetettségi szintet (neuronok és rétegek) tartalmaz, mint a hagyományos modellek. A neuronokat csoportosító, hierarchikusan egymásból felépülő **kapszulák** ötletét használja a Geoffrey Hinton által bevezetett Transforming Autoencoder [13] struktúra. Az itt bemutatott képfelismerő hálózat minden kapszulája valamilyen entitást detektál a képen, és kétféle információt továbbít: a hozzá tartozó entitás jelenlétének valószínűségét (intenzitását) és az általánosított pozíciót, amivel az aktuálisan megjelenik. Entitás lehet például egy tárgy egy bizonyos része, ennek általánosított pozíciója pedig az orientációja, mérete, színe, stb. A hierarchia legalsó szintjét képező kapszulák a hálózat bemenetéből számítják a kimenetüket, a többi pedig az alacsonyabb szintekről kap jóslatokat (tippeket) az általánosított pozíciójára vonatkozóan, és ezek között keres egybeeséseket.

A bemeneti kép feldolgozása a konvolúciós réteg receptív mezőihez hasonlóan elhelyezkedő területek elemzésével kezdődik. Minden területen ugyanazt a műveletsort hajtjuk végre külön-külön. Először egy nemlineáris aktivációs függvényekkel rendelkező perceptron-réteggel kötjük össze. Ezt a réteget arra tréningezzük, hogy előre meghatározott számú különböző elemi kapszula által reprezentált entitások általánosított pozícióit adja kimenetül egy-egy p hosszú valós vektorként, kiegészítve az entitás jelenlétének valószínűségével. Az, hogy melyik kapszulához milyen entitás tartozik, a tanítás során alakul ki. Mivel a bemeneti területek megosztják egymás között az élsúlyokat, ezért a kép minden része pontosan ugyanazokon a feldolgozási lépéseken esik át, elhelyezkedéstől függetlenül.

Azokat a kapszulákat, amelyek nem a hierarchia legalsó szintjén állnak, mátrixszorzást megvalósító rétegek kötik össze az eggyel alacsonyabb szintű entitásokkal. Legyenek $i_1 \dots i_k$ azok a kapszulák, ahonnan az l-edik (magasabb szintű) kapszula bemenetet kap, $t_{i_1} \dots t_{i_k}$ az aktuális általánosított pozíciókat kódoló vektorok, $p_{i_1} \cdots p_{i_k}$ pedig a hozzájuk tartozó valószínűségek. Ekkor az i_j-ből l-be menő, jóslatért felelős hálózatrész a (tanítás eredményeként kapott) $p \times p$ méretű $T_{i_j l}$ valós mátrixszal való szorzást valósítja meg. Az így kapott $v_{i_1} = T_{i_1 l} t_{i_1}, \dots, v_{i_k} = T_{i_k l} t_{i_k}$

vektorok első két koordinátájához még hozzáadjuk annak a felfogó mezőnek a koordinátáit, ahonnan az aktuális számítás indult. A t_l pozícióvektorra vonatkozó jóslatok az így kapott $v'_{i_1} \dots v'_{i_k}$ vektorok $p_{i_1} \cdots p_{i_k}$ értékekkel súlyozva. Ezek között az l-edik kapszula egy expectation–maximization (EM, [27]) módszert használó klaszterezési algoritmussal keres egybeeséseket, hasonlóságokat, és számítja ki az entitás jelenlétének bizonyosságát. Ha egy kapszula egy szűk klasztert talál a bemenetei között, akkor azt az információt továbbítja, hogy az entitás nagy valószínűséggel jelen van, általánosított pozíciója pedig a klaszter elemeinek (valamilyen előre meghatározott módszerrel számított) középpontja. Mivel a klaszteren kívüli bemeneteket egyszerűen figyelmen kívül hagyjuk, ezért egy ilyen kapszula jól szűri a zajt.

Az eddig bemutatott struktúrák nem explicit módon választják el a valós ekvivariáns faktorokat a többi paramétertől, hanem erős tanuló algoritmusokat használnak erre. Ennek az a hátránya, hogy jelen formájukban nincs egyszerű kiterjesztésük bonyolultabb vagy több invarianciára. Léteznek olyan neurális hálózatok, amelyek explicit paraméter-szétválasztást valósítanak meg különböző értelmezési tartományokon. Jelen fejezet további része két ilyen struktúrát mutat be, melyek közül mindkettő autoenkóder-alapú.

3.1.6. DCIGN

A **mély konvolúciós inverz grafikai hálózat (DCIGN)** [15] képeknek egy olyan interpretációját képes megtanulni, amelyben 3 dimenziós struktúrát és nézőpont-transzformációt kódoló paraméterek vannak különválasztva a tartalmat leíróktól. A modell a konvolúciós autoenkóderekéhez hasonló kódoló és dekódoló részhálózatból áll, és a tréningezés a kódoló z kimenetének egyes részeit kényszeríti arra, hogy bizonyos transzformációkat reprezentáljanak. Egy ilyen speciális reprezentációból a dekódoló képes újragenerálni a képet, így egyetlen érték változtatásával új képek kaphatóak, amelyek csak az adott tulajdonságban térnek el egymástól.

Az egyik példa, amelyen a hálózat működését bemutatták, egy arcokból álló adathalmaz, a kezelendő transzformációk pedig a függőleges tengely körüli forgatás (jobbra-balra bólogatás), az emelkedés (fel-le bólogatás), illetve a fény beesési szöge. Ezeknek a kitüntetett tulajdonságokat leíró z_1, z_2 illetve z_3 változókat külső változóknak, a többit, amik például az arc alakjának, az arckifejezésnek, stb. felelnek meg, belső változó nevezzük. A hálózat a tanítás során egyszerre mindig egy olyan adatköteget (batch) kapott bemenetként, amelynek elemei vagy csak egy külső változóban tértek el egymástól, vagy az összes külső változóban megegyeztek, és a többi tulajdonságban tértek el. A tréningezés egy ciklusa a következő lépésekből áll. Először véletlenszerűen választunk a {függőleges forgatás, emelkedés, fény szöge, belső tulajdonságok} halmazból egy elemet 1:1:1:10 arányú valószínűségekkel, ennek fog

megfelelni a z_{train} változó. Ezután azokból a kötegekből választunk véletlenszerűen egyet, amelyekben csak a választott tulajdonság változik. A köteg minden elemére futtatjuk a hálózat kódoló részét, és kiszámoljuk a kapott reprezentációvektorok átlagát. Mielőtt ezeket továbbadnánk, a köteg minden elemére kicseréljük a z_{train}-n kívüli vektorelemeket az átlagvektor megfelelő indexű elemeire. Ezután az így kapott vektorokkal futtatjuk a dekódolót, kiszámítjuk a hibát, és hiba-visszaterjesztési algoritmus segítségével a dekódoló rész gradienseit. A z_{train}-hez tartozó derivált változatlanul marad, a többi, vele egy szinten lévő neuronét viszont az átlagtól való eltéréssel helyettesítjük. Ezekkel az új gradiensekkel folytatódik a hiba-visszaterjesztés a kódoló hálózatrész rétegeiben.

Ez a fajta tréningezés a kódolót és a dekódolót is speciális viselkedésre ösztönzi. Az, hogy egy kötegen belül egyetlen kivétellel minden változót egységesítünk, arra kényszeríti a dekódolót, hogy a kötegen belüli eltéréseket a maradék egy változó alapján kreálja újra. A gradiensek rögzítése pedig arra tréningezi a kódolót, hogy az eltérésekről minden információt a kijelölt neuronba gyűjtsön. Az így betréningezett hálózat kódolójának kimenetéből már egyszerűen megkaphatóak a valódi transzformációt leíró értékek.

3.1.7. XCov

A [6] cikkben bemutatott hálózat alapja szintén egy autoenkóder, azonban itt a struktúrán kívül egy speciális költségfüggvény is segíti az adat változékonyságát adó tulajdonságok különválasztását. Az előző bemutatott megoldáshoz hasonlóan a kódoló kimeneti értékei két diszjunkt halmazra oszthatók. Az egyik halmaz elemei a rekonstrukcióban és a tulajdonságok megkülönböztetésében is szerepet játszanak, ezek az $\hat{y}$ vektort alkotó *megfigyelt változók*. Az ezen kívülieket csak a rekonstrukcióhoz használja a hálózat, ezeket *látens változóknak* nevezzük, és a z vektor tartalmazza őket. Adott D adathalmazbeli x bemenetre és y címkére (azaz kívánt megfigyelt változókra) az F kódolóból és G dekódolóból álló hálózat a következő számítási lépéseket teszi:

$$\{\hat{y}, z\} = F(x, \theta);$$

$$\hat{x} = G(y, z, \Phi),$$

ahol θ a kódoló, Φ pedig a dekódoló paramétere. A költségfüggvénynek a szokásos rekonstrukciós költségen kívül két másik tagja is van. Az egyik a megfigyelt változókon értelmezett felügyelt költség, a kívánt és a kapott megfigyelt változók kereszt-entrópiája. A másik a megfigyelt és a látens változók közötti (tehát felügyelet nélküli) kereszt-kovariancia, amelyet **XCov**nak nevezünk. Ez a tag választja el a megfigyelt és a látens változókat egymástól azáltal, hogy megakadályozza, hogy a

látens változók az adatköteg olyan különbözőségeit kódolják, amik a címkék eltérésének tulajdoníthatóak. A hálózat tréningezése során tehát azokat a $\hat{\theta}$, $\hat{\Phi}$ paramétereket keressük, amelyekre

$$\hat{\theta}, \hat{\Phi} = \underset{\theta,\Phi}{argmin} \sum_{x,y \in D} \|x - \hat{x}\| + \beta \sum_{i} y_i \log(\hat{y}_i) + \gamma C$$

teljesül, ahol C az XCov tagot jelöli, azaz N-es kötegméret esetén a kötegbeli sorszámot felső indexszel, a kötegen vett átlagot pedig felülvonással jelölve

$$C(\hat{y}^{1\ldots N}, z^{1\ldots N}) = \frac{1}{2} \sum_{i,j} [\frac{1}{N} \sum_{n} (\hat{y}_i^n - \bar{\hat{y}_i})(z_j^n - \bar{\hat{z}_j})]^2.$$

A β és γ értékek meghatározása a konkrét feladattól függ. Minél nagyobbra választjuk γ-hoz képest β-t, annál jobban fog teljesíteni a hálózat klasszifikációban, és minél kisebbre, annál jobban el tudja választani a megfigyelt változókat a látens változóktól.

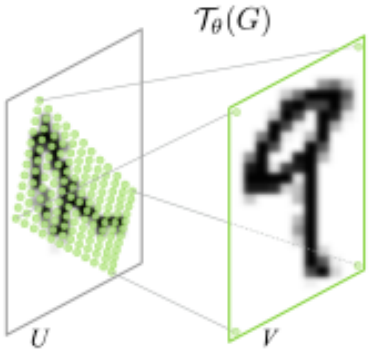
3.1.8. Spatial Transformer modul

Jelen alfejezet egy olyan hálózati egységet mutat be, amely bármilyen képfeldolgozó neurális hálózatba beépíthető, és jól használható a bemutatottakhoz hasonló invarianciatanuló struktúrákban.

Abban az esetben, amikor egy alakfelismerésre tréningezett, normalizált bemeneten működő H hálózat futtatása előtt végzett előfeldolgozási lépésként egy H' hálózatból megkapjuk a képen szereplő formák paramétereit, felmerül a kérdés, hogy H' kimenetéből pontosan hogyan állítjuk elő H bemenetét. Erre nyújt megoldást a **Spatial Transformer** nevű deriválható modul [14], amely egy hálózatba beépítve képek bizonyos síkbeli geometriai transzformációinak lehetőségét teremti meg. Így tehát a kép fontos régióinak kiválasztása után egyetlen lépésben kanonikus alakra is tudjuk hozni azokat. A Spatial Transformer bármilyen paraméterezett transzformációt meg tud tanulni hiba-visszaterjesztéssel és gradiens-módszerrel, amennyiben az a paraméterei szerint deriválható.

Egy adott transzformációra a mechanizmus három részből áll, melyek mindegyike deriválható a paraméterei szerint. Első lépésben a *lokalizáló hálózat* néhány rejtett rétegével az U bemenetből kiszámítja a képre alkalmazandó T_θ transzformáció paramétereit. Az így kapott $\theta = f_{loc}(U)$ vektor mérete a transzformáció típusától függ. A lokalizáló elem lehet például egy egyszerű, néhány rétegből álló konvolúciós hálózat. Második lépésben a paraméterek ismeretében a *rácsgeneráló* hálózatrész előállít egy mintavételezési rácsot, amely a bemeneti kép azon (x_i^s, y_i^s) koordinátáinak halmaza,

ahonnan a transzformált kimenet elkészítéséhez szükséges mintát venni. A rácsgeneráló ezt annak függvényében állítja elő, hogy a kimeneti pixeleknek hova kell esniük. Ezek az előre meghatározott $G_i = (x_i^t, y_i^t)$ koordináták alkotják a célrácsot, amely általában egy hagyományos állású G négyzetrács. A mintavételezési rács pontjainak koordinátáit az $(x_i^s, y_i^s) = T_\theta(G_i)$ képlet adja.



12. ábra. $T_\theta(G)$ mintavételezési rács alkalmazása az U képre V kimenettel [14]

A rácsgenerálás után következik a harmadik fázis, amely maga a transzformáció. Minden kimeneti pixelt úgy kapunk, hogy egy mintavételező kernelt alkalmazunk a bemenet $T_\theta(G)$ által meghatározott helyén. A mintavételező kernel pontos működése az adott feladattól függ. A Spatial Transformer modul többcsatornás bemenetre is alkalmazható, ilyen esetben ugyanazokat a műveleteket végzi el az összes csatornán.

4. Kísérletek

A kutatómunkám során olyan mesterséges neurális hálózat létrehozását kíséreltem meg, amely egy kétdimenziós képen lévő alakzat helyparamétereit választja el a többi tulajdonságot leíró változóktól, és egy centralizált változatát adja vissza a bemeneti képnek. Ebben a fejezetben két hálózat teszteredményeit mutatom be. Az egyik egy konvolúciós autoenkóder struktúra, amely a DCIGN modellhez hasonló transzformációkat végez a kódoló kimenetén, a másik pedig ugyanennek egy olyan változata, ahol a tanulást a létrahálózatéhoz hasonló keresztkapcsolatok segítik.

4.1. A hálózatok felépítése és tréningezése

A két említett hálózat autoenkóder struktúrája megegyezik, csak a keresztkapcsolatokban térnek el egymástól. Mindkét kódoló öt konvolúciós rétegből áll, mindet

egy-egy nemlineáris aktivációs függvény követi. Az utolsó három konvolúciós réteg után egy-egy max pooling is helyet kap. A legutolsó összevonó réteg kimenete 10 elemű neuronréteghez kapcsolódik, ez adja a kódoló kimenetének 12 értékéből az első tízet. A maradék kettő kimenetet úgy kapjuk, hogy a kódoló az első két konvolúció és az összes max pooling réteg kimenetéből információt továbbít a magas aktivitású neuronok helyéről. Ezt úgy teszi, hogy a neuronokhoz koordinátákat rendel, és ezeket minden szinten a neuron-aktivitásokkal súlyozza az összes neuronra, majd ezeket normalizálja egy képen belül, végül összeadja a köteg elemeire. Az így kapott értékeket perceptron-réteg köti össze a kimenettel. Ez az extra információtovábbítás azért szükséges, mert a konvolúció nem továbbít információt a szűrők elhelyezkedéséről. A dekódoló öt dekonvolúciós rétegből és a kódolóra szimmetrikusan három unpooling rétegből áll.

Az első hálózat, amelyről ez a fejezet szól, egyáltalán nem tartalmaz közvetlen kapcsolatokat a kódoló és a dekódoló alsóbb szintjei között. A második hálózat keresztkapcsolatai közül az egyik, c_1, a kódoló első konvolúciós szintjének h_1 kimenetét köti össze a dekódoló utolsó dekonvolúciós szintjének d_1 bemenetével, és ezekből állítja elő a d'_1 aktivitást, amit végül az utolsó dekonvolúciónak bemenetül adunk. A másik keresztkapcsolat, c_2, egy réteggel fentebb helyezkedik el: a második konvolúció kimenetét, h_2-t köti össze az utolsó előtti dekonvolúció d_2 bemenetével, és belőlük állítja elő d'_2-t. A keresztkapcsolatok a [21] cikk által leírt vanilla combinator-t valósítják meg. A keresztkapcsolattal ellátott hálózat sematikus rajza a fejezet végén található.

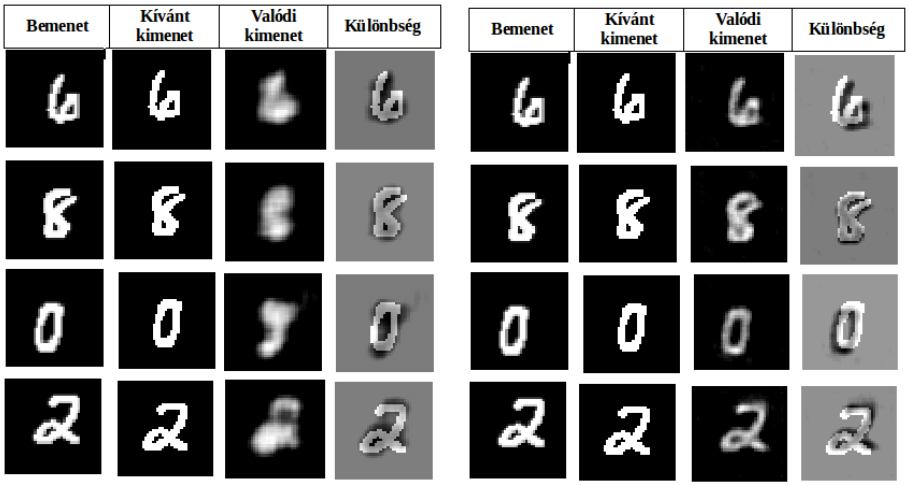
A hálózatokat az MNIST [4] kézzel írott számjegyeket tartalmazó adatbázis segítségével tréningeztem. A tanító algoritmus az eredeti adathalmaz elemeiből minden iterációban háromféle adatköteg egyikét állította elő eltolással. Egy "x" típusú batch minden eleme ugyanabból a MNIST-képből kapható meg, és az y tengely mentén való eltolásuk is megegyezik, tehát csak x tengellyel párhuzamos eltolásokkal térnek el egymástól. Az "y" típusú batchek hasonlóan jönnek létre, egy ilyen adatköteg tagjai csak az y tengellyel párhuzamos eltolásokkal különböznek egymástól. A harmadik típus neve "többi", egy ilyen batch elemei azonos eltolással jönnek létre különböző MNIST-képekből. Az eltolások minden esetben legfeljebb 3 pixel értékűek voltak minden irányban. A kódoló kimenetének 12 eleméből az utolsó hivatott az y irányú eltolást kódolni, az utolsó előtti az x irányút, a maradék 10 pedig az adat további változékonyságáért felel. A tréningezés során minden köteg esetén úgy adtuk a kódoló kimenetét a dekódolónak, hogy azon a változón kívül, amelyik a köteg típusának felel meg, az összes többit egyenként átlagoltuk a batch elemeire.

Mindkét hálózat esetén az volt a cél, hogy egy középről eltolt képet adva bemenetként annak centralizált változatát kapjuk vissza, és a kódoló kimenetében az

általunk kijelölt helyeken jelenjenek meg a helyet leíró paraméterek. Centralizált vagy középre igazított helyzeten minden esetben az MNIST-beli elhelyezkedését értjük a számjegyeknek. A költségfüggvény az első hálózat esetében két tagból áll. Az egyik a rekonstrukciós költség, amely a bemenet középre igazított változata és a kimenet négyzetes eltérését méri. A másik arra ösztönzi a kódolót, hogy csak a kijelölt változót használja a kötegen belüli változások leírására. Ez a tag a kódoló kimenetének és az előző bekezdésben leírt átlagolás eredményének a négyzetes eltérése. Ezek a költségek a másik hálózat esetén is jelen vannak, és kiegészülnek két további taggal. Mindkét keresztkapcsolatra hozzáadjuk a költségfüggvényhez a kódoló és a dekódoló kimenetének eltérését adott szinten, azaz h_i és d'_i négyzetes hibáját. A felsorolt hibatagok nem ugyanolyan súllyal szerepelnek a költségfüggvényben. A rekonstrukciós hiba együtthatóját választottuk a legnagyobbnak, ennek ötödével súlyozzuk a keresztkapcsolatok hibáját, és annak felével a kódoló kimeneténél mért hibát.

4.2. Tesztek

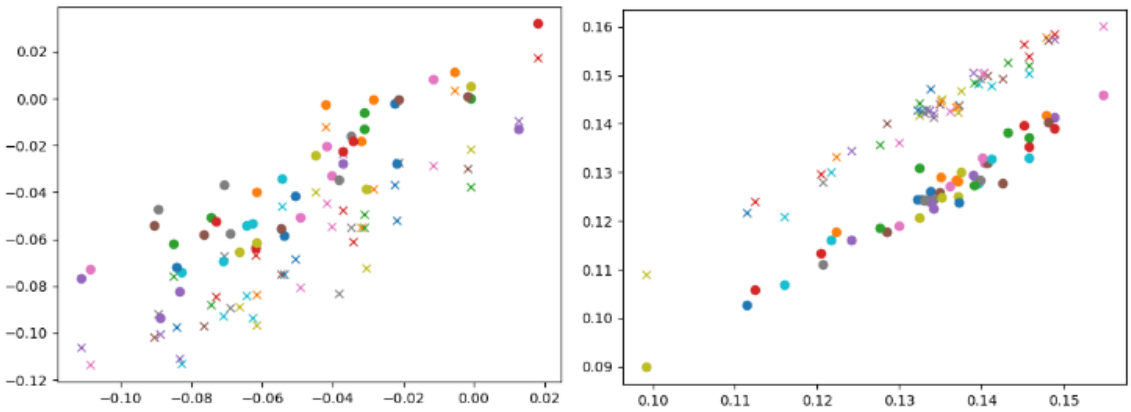
A hálózatok tesztelése során az MNIST adathalmazból eltolással kapott képeket adtunk bemenetként, és a teljesítményt két szempontból vizsgáltuk. Az egyik a centralizált rekonstrukció minősége, azaz a hálózat kimenetének a középre igazított képtől (azaz az eredeti MNIST-elemtől) való eltérése. A másik elvárás az volt, hogy a kódoló kimenetének általunk kijelölt részei megfelelően reprezentálják az eltolás mértékét. A centralizált rekonstrukció minőségét a két hálózat esetén ugyanannyi tréningezés után, ugyanazokon a bemeneteken és kívánt kimeneteken szemlélteti a következő ábra.



13. ábra. Négy példa a két hálózat által adott kimenetekre.

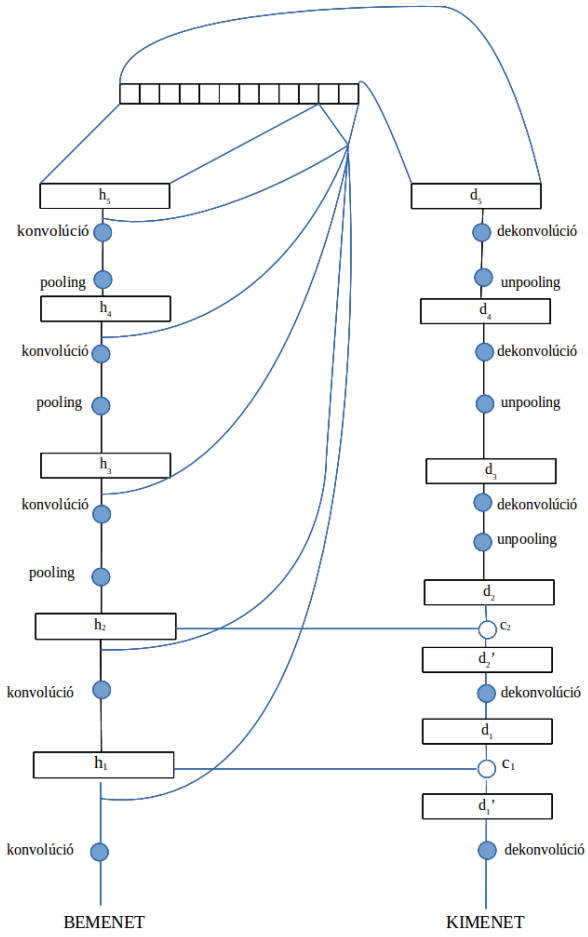
A bal oldali táblázatban az első hálózat, a jobb oldaliban pedig a második, keresztkapcsolatokkal segített hálózat eredményei láthatóak. Mindkét táblázat bal szélső oszlopában eltolt MNIST-beli képek szerepelnek, ezeket kapja a hálózat inputként. A második oszlopban az első oszlop elemeinek centralizált változata áll, ezeket várjuk kimenetként. A harmadik oszlop tartalmazza a hálózat által adott kimenetet, a negyedik pedig a kívánt kimenet és a valódi kimenet pixelenkénti különbségét. A bemeneti képek eltolásainak mértéke két tizedes jegyre kerekítve fentről lefelé rendre $(1.88, 2.62)$, $(1.11, 0.12)$, $(2.73, -2.43)$ és $(-2.20, 1.82)$. Az eltolást leíró számpár első eleme az y tengellyel párhuzamos eltolás mértéke, a második pedig az x tengellyel párhuzamos eltolásé, az értékek pixelben értendők. Az ábrán jól látszik, hogy a második hálózat sokkal pontosabban adja vissza a bemeneti mintát, mint az első, azonban a középre igazításban elmarad tőle. Ennek az az oka, hogy a hozzáadott keresztkapcsolatok megtanulták az identitás függvényt, így valójában a rekonstrukció nagy részét nem az autoenkóder végezte.

A kódoló kimenetét csak felügyelet nélkül tréningeztük, ezért a tesztelésben sem a valódi eltolásértékekkel vetjük össze. Ehelyett azt ellenőrizzük, hogy a megfelelő neuronok aktivitásai ekvivariánsak-e a bemenet eltolására. Tehát azt szeretnénk, hogy ha bemenetként adunk a hálózatnak egy képet, majd ugyanannak a képnek egy eltoltját, akkor az x tengellyel párhuzamos eltolásnak megfelelő neuronaktivitás az x irányú eltolás mértékével arányosan változzon, és ugyanez teljesüljön az y dimenzióra is. Ennek eredményeit a két hálózatra a következő grafikonok szemléltetik.



14. ábra. Minden centralizált képhez egy kör és egy kereszt tartozik a grafikonon. A vízszintes tengely jelöli az eltolás nélkül beadott képek esetén kapott x-nek megfelelő neuron-aktivitásokat. A függőleges tengely mentén kereszt jelöli a +3-mal eltolt képre kapott aktivitást, és kör a -3-mal eltoltra kapottat.

A bal oldali grafikon az első hálózat kódolójának kimeneti aktivitásaihoz tartozik, a jobb oldali pedig a második, keresztkapcsolatokkal segített hálózatéhoz. Minden bemeneti képet átadtunk a hálózatnak eltolás nélkül, +3, és -3 pixelnyi x irányú eltolással. A grafikonokról az olvasható le, hogy habár mindkét esetben korrelál az x iránynak megfeleltetett neuron aktivitása a valódi x irányú eltolással, a második hálózat esetében egy sokkal szorosabb összefüggés jön létre. Megfigyelhető továbbá, hogy a második grafikonon a körök vannak alul, azaz pozitív értékű eltolás esetén a neuron kimenete is pozitív irányba változik, ellentétben az első hálózattal.



15. ábra. Az általam használt hálózat sematikus rajza. Az aktivációs és normalizációs lépések az egyszerűség kedvéért nem szerepelniek az ábrán.

5. Összefoglalás

A dolgozatban bemutattam a mesterséges neurális hálózatok illetve a mély tanulás alapjait, és tárgyaltam néhány alapvető példát. Felvázoltam a transzformációtanulás feladatát, és annak jelentőségét. Leírtam 7 olyan autoenkóder-alapú hálózati struktúra működését, amelyek vagy explicit paraméterszétválasztást valósítanak meg, vagy implicit módon kihasználják az adat invarianciáit. Ezen kívül említettem egy transzformáló hálózati modult, amely bármilyen képfeldolgozó hálózatba beilleszthető. Az utolsó fejezetben leírtam a saját kísérleti eredményeimet: bemutattam egy speciálisan tréningezett konvolúciós autoenkódert, és annak egy kiegészített változatát.

Hivatkozások

[1] *Lstm networks for sentiment analysis*, http://deeplearning.net/tutorial/lstm.html, Accessed: 2017-05-05.

[2] *Stacked autoencoders*, http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders.

[3] *Számtábla felismerés konvolúciós neurális hálóval*, https://bois083.wordpress.com/szamtabla−felismeres−konvolucios−neuralis−−haloval.

[4] *The MNIST database of handwritten digits*, http://yann.lecun.com/exdb/mnist/.

[5] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent, *Unsupervised feature learning and deep learning: A review and new perspectives*, CoRR **abs/1206.5538** (2012).

[6] Brian Cheung, Jesse A. Livezey, Arjun K. Bansal, and Bruno A. Olshausen, *Discovering hidden factors of variation in deep networks*, CoRR **abs/1412.6583** (2014).

[7] Taco S Cohen and Max Welling, *Group equivariant convolutional networks*, arXiv preprint arXiv:1602.07576 (2016).

[8] Carl Doersch, *Tutorial on variational autoencoders*, arXiv preprint arXiv:1606.05908 (2016).

[9] S. M. Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, Koray Kavukcuoglu, and Geoffrey E. Hinton, *Attend, infer, repeat: Fast scene understanding with generative models*, CoRR **abs/1603.08575** (2016).

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

[11] Klaus Greff, Antti Rasmus, Mathias Berglund, Tele Hao, Harri Valpola, and Juergen Schmidhuber, *Tagger: Deep unsupervised perceptual grouping*, Advances in Neural Information Processing Systems, 2016, pp. 4484–4492.

[12] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra, *Draw: A recurrent neural network for image generation*, arXiv preprint arXiv:1502.04623 (2015).

[13] Geoffrey Hinton, Alex Krizhevsky, and Sida Wang, *Transforming auto-encoders*, Artificial Neural Networks and Machine Learning–ICANN 2011 (2011), 44–51.

[14] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu, *Spatial transformer networks*, CoRR **abs/1506.02025** (2015).

[15] Tejas D. Kulkarni, Will Whitney, Pushmeet Kohli, and Joshua B. Tenenbaum, *Deep convolutional inverse graphics network*, CoRR **abs/1503.03167** (2015).

[16] Tom M Mitchell, *Machine learning. 1997*, Burr Ridge, IL: McGraw Hill **45** (1997), no. 37, 870–877.

[17] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al., *Recurrent models of visual attention*, Advances in neural information processing systems, 2014, pp. 2204–2212.

[18] Michael A. Nielsen, *Neural networks and deep learning*, Determination Press, 2015, `http://neuralnetworksanddeeplearning.com/`.

[19] A. L. Nigrin, *Sonnet: a self-organizing neural network that classifies multiple patterns simultaneously*, 1990 IJCNN International Joint Conference on Neural Networks, June 1990, pp. 313–318 vol.2.

[20] Stuart Russell, Peter Norvig, *Neurális hálók*, Mesterséges intelligencia, Panem Kft., 2002.

[21] Mohammad Pezeshki, Linxi Fan, Philemon Brakel, Aaron Courville, and Yoshua Bengio, *Deconstructing the ladder network architecture*, International Conference on Machine Learning, 2016, pp. 2368–2376.

[22] Alec Radford, *Face manifold from conv/deconv variational autoencoder*, `https://youtu.be/XNZIN7Jh3Sg`, Jan 2015.

[23] Ronald A Rensink, *The dynamic representation of scenes*, Visual cognition **7** (2000), no. 1-3, 17–42.

[24] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams, *Neurocomputing: Foundations of research*, MIT Press, Cambridge, MA, USA, 1988, pp. 696–699.

[25] Tania Stathaki, *The Karhunen-Loeve transform (KLT) in image processing*, [Online; accessed 25-May-2017].

[26] Harri Valpola, *From neural pca to deep unsupervised learning*, Advances in Independent Component Analysis and Learning Machines (2015), 143–171.

[27] Wikipedia, *Expectation–maximization algorithm — Wikipedia, the free encyclopedia*, 2017, [Online; accessed 28-May-2017].

[28] ———, *Kullback–Leibler divergence — wikipedia, the free encyclopedia*, 2017, [Online; accessed 30-May-2017].

[29] Saul A. Teukolsky William T. Vetterling William H. Press, Brian P. Flannery, *Numerical recipes in fortran 77: the art of scientific computing*, 2 ed., Cambridge University Press, 1992.

[30] Cem Yuceer and Kemal Oflazer, *A rotation, scaling, and translation invariant pattern classification system*, Pattern Recognition **26** (1993), 687–710.

[31] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, *Deconvolutional networks*, 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, June 2010, pp. 2528–2535.

[32] M. D. Zeiler, G. W. Taylor, and R. Fergus, *Adaptive deconvolutional networks for mid and high level feature learning*, 2011 International Conference on Computer Vision, Nov 2011, pp. 2018–2025.

[33] J. Zhao, M. Mathieu, R. Goroshin, and Y. LeCun, *Stacked What-Where Auto-encoders*, ArXiv e-prints (2015).