Házi feladat dokumentáció

Ipari képfeldolgozás és megjelenítés

Készítette: Bujtor Bálint

Contents

A feladat ismertetése:

A házi feladatom három főbb részből áll. Első részként egy paraméterezhető konvolúciós neurális hálót kellett elkészítenem, amely osztályozásra képes. A feladatom második része ennek a neurális hálónak a tanítása volt, egy 55 osztályból álló közlekedési táblákat tartalmazó adatbázison. A tanítást végző függvénynek is paraméterezhetőnek kell lennie. A tanítás során validálni is kellett és a futás végén a legjobb validációs pontosságot kell visszaadni. A feladat harmadik, egyben utolsó részfeladata az előző két feladat hiperparamétereinek az optimalizálása a Bayesian optimization python könyvtár segítségével.

Első feladat:

A házi feladat első része tehát a paraméterezhető konvolúciós neurális háló megalkotása volt.

A háló paraméterei a következők:

- nC: Az osztályok száma [55]
- nFeat: Az első réteg kimeneti csatornaszáma. Az ezt követő rétegek be- és kimeneti csatornaszáma egyezzen ezzel meg, majd minden leskálázó (strided konvolúciós) réteg duplázza ezt meg. [4,8,16,32]
- nLevels: A háló szintjeinek száma. Egy szintnek az azonos térbeli kiterjedésű tenzorokon operáló rétegeket nevezzük (2 leskálázás közt). [2-5]
- dropOut: Az osztályozó réteget közvetlenül megelőző dropout réteg bemeneti valószínűsége [0-0.5]
- layersPerLevel: Az egy szinten található konvolúciós rétegek száma. [1-5]
- residual: Bináris változó, True érték esetén minden konvolúciós rétegben (kivéve a leskálázó rétegeket) valósíts meg egy reziduális kapcsolatot a szint bemenete és a szint végén megjelenő leskálázó réteg bemenete közt. [True, False]
- kernelSize: A konvolúciós rétegek mérete. [3,5,7]
- nLinType: Kategorikus változó, amellyel a nemlinearitás típusát állíthatja [Categorical]
- bNorm: Bináris változó, amellyel állítható, hogy a konvolúciós rétegekbe teszünk-e BatchNormot. [True, False]

A feladatot több kisebb részfeladatra osztottam. A teljes neurális háló szintekből épül fel, az egyes szintek pedig rétegekből épülnek fel, ezért kifizetődő, hogy ha az alsóbb szintektől kezdve, felfelé építkezve valósítjuk meg a neurális hálót. A megoldás során először egy adott réteget megvalósító osztályt hoztam létre, majd egy szintet, ami rétegekből épül fel. Végül pedig a teljes hálót készítettem el, a már létrehozott szintekből.

Layer class:

A neurális hálóm legkisebb építőegysége a Layer class. Az osztály inicializálásakor számos paramétert meg tudunk adni. Az inplanes paraméterrel a bemeneti csatornaszám adható meg, ennek értéke az első, bemeneti rétegnél tipikusan 3 (színes képek esetén) vagy 1 (szürkeárnyalatos képek esetében). Az ezt követő rétegek bemenete már az nfeat paraméter által meghatározott érték lehet. Az outplanes a kimeneti csatornaszámot adja meg. Megadható tovább a kernelméret, a stride értéke (erre a paraméterre a strided konvolúciós rétegeknél lesz majd szükségünk). További paraméter a nemlinearitás típusa, amely egy kategorikus változó. Én három aktivációs függvényt adtam meg, ezek a ReLU, a LeakyReLu illetve a Sigmoid függvények, alapértelmezett a ReLU függvény. Ezen kívül a bnorm paraméterrel tudjuk beállítani,

hogy kívánunk egy batch normalizálást végrehajtani az adott rétegben. Ha szeretnénk, akkor az osztály 2 dimenziós batchnormot alkalmazok, illetve a konvolúció bias paraméterét hamisra állítja (hiszen ezt a batchnorm eliminálja). Ellenkező esetben a biast igazra, a bnorm paraméter értéke ebben pedig egy identitás blokkal lesz egyenlő, amely a beérkező tensort változatlanul adja ki a kimenetére. Végül azt is lehetőségünk van kiválasztani, hogy reziduális réteget szeretnénk-e vagy sem. A választásunkat az osztály res változójában tároljuk és a forward függvényben fogjuk majd figyelembe venni.

Az inicializálás során létrehozunk egy 2 dimenziós konvolúciót, amelynek paramétereit az osztály megfelelő paramétereivel tesszük egyenlővé. Érdemes megemlíteni, hogy paddinget is alkalmazok, melyre azért van szükség, hogy a képméret csökkenését elkerüljem. Továbbá elmentjük a batchnorm és aktivációs függvények típusát az osztály erre létrehozott saját változóiba.

Minden nn.Module-ból leszármaztatott osztály implementálása során feladatunk a forward() metódus felüldefiniálása, ezt ebben az esetben sem hagyhatjuk ki. A beérkező x változón sorban végrehajtjuk a konvolúciót, a batchnormalizálást és az aktivációt, majd az így kapott eredmény lesz a visszatérési érték. Ha reziduális réteget választottunk, akkor a bemeneti x változót eltároljuk, majd az aktivációt megelőzően hozzáadjuk az aktuális tensorhoz.

Level class:

Következő, eggyel magasabb szintet képviselő építőelemünk a Level osztály. Ez az osztály több rétegből épül fel. Az egyes réteget meghatározó paramétereken felül további paraméter az egy szinten lévő rétegek száma.

Az inicializáló során az egyes szinteket egy listában tárolom, amelyet majd egy nn.Sequential osztálynak fogok átadni paraméterként. A Sequential osztály előnye, hogy segítségével a fordító el tudja érni az egyes elemek parameters() függvényét, hogy el tudja végezni a backpropagationt, amit egy sima listával nem tudnánk megtenni. További előnye, hogy van saját forward függvénye is, ami sorban meghívja a listában lévő elemek forward függvényét, ezzel jelentősen egyszerűsítve az osztályomat.

A listát elsőként a bemeneti réteggel bővítem ki, amelynek bemeneti és kimeneti csatornaszáma nem egyezik, hiszen ez a háló legelső rétegénél például különbözik. Ezt követően összesen layerperlevel – 1 alkalommal bővítem ki a listát egy-egy réteggel. Ezek a rétegek megegyeznek egymással, bemeneti és kimeneti csatornaszámuk azonos. A listát utolsó eleme a kimeneti, strided konvolúciós réteg, amely kétszeres stride és csatornaszámában különbözik a középső rétegektől. A tervezés során úgy döntöttem, hogy az egy szinten lévő rétegek számába nem veszem be a strided konvolúciós réteget. Az inicializáció utolsó lépéseként a Sequential osztály visszatérési értékét elmentjük egy saját változóba. Ennek segítségével fogjuk tudni meghívni sorban az egyes rétegek forward() függvényét.

A Layer osztályhoz hasonlóan itt is felül kell definiálni a forward függvényt, ami rendkívül egyszerű lesz, ennek oka, amint azt már említettem, hogy a Sequential osztálynak van forward függvénye, így tulajdonképpen a Sequential osztály forward függvénye fut le.

Network class:

A network osztályban építem fel a végleges neurális hálót az eddig létrehozott osztályok segítségével. A szintet és a réteget meghatározó paraméterek mellett további paramétereket is meg kell adnunk az inicializálás során. Az nfeat paraméter az első réteg kimeneti csatornaszámát adja meg, az nc paraméter

az osztályaink számát. További paraméter a szintek számát meghatározó nLevels változó és az osztályozó réteget közvetlenül megelőző dropout réteg bemeneti valószínűsége.

Az inicializálás elejét hasonlóképpen végeztem el, mint a Level osztályban, csak itt eggyel magasabb szinten. A bemeneti réteget a megfelelő paraméterekkel hozom létre, majd egy ciklus használatával listába gyűjtöm a középső rétegeket, amit ugyanúgy átadok egy Sequential osztálynak, mint a Level osztálynál, a Sequential osztály által visszaadott függvényt pedig egy saját változóban tárolom. Itt érdemes lehet némi magyarázatot fordítani a ciklushatár értékeire. Mint az a kódban látható 1 és nLevels + 1 között fut a ciklus. Erre azért van szükség, hogy a szintek létrehozásakor fel tudjam használni a ciklusváltozó értékét a csatornaszám meghatározásához. Mint az a specifikációban szerepel, az egyes szintek csatornaszámát az azt követő szintnek mindig dupláznia kell, a szinten belül ez az érték állandó. A ciklusváltozó felhasználásával ez a feltétel biztosítható: $planes = 2^i * nFeat$.

A középső rétegeket követi a dropout réteg létrehozása, ami a dropout valószínűségét kapja meg paraméterként. Ezt követően egy adaptív kétdimenziós poolingot is végrehajtok, melyre azért van szükség, hogy a különböző szint, ezáltal csatornaszám és tensorméret ne okozzon problémát. A neurális háló utolsó eleme pedig az osztályozást megvalósító lineáris réteg. Ennek a rétegnek a bemeneti dimenzióját a középső rétegek csatornaszáma alapján határoztam meg, kimeneti dimenziója az osztályok számával egyezik meg.

A forward függvény során sorban meghívom az egyes függvényeket majd a kapott eredménnyel térek vissza.

Második feladat:

A feladat második részeként a létrehozott neurális hálóhoz kellett létrehoznom egy paraméterezhető neurális hálót, amelyben mind a háló, mind a tanulásra vonatkozó hiperparamétereket tudjuk módosítani. A tanító algoritmus további paraméterei:

- bSize: A bacth méret [4,8,16,32,64,128,256]
- lr: a tanulási ráta [1e-5,1e-1]
- lr_ratio: a tanulási ráta ütemező eta_min paramétere és a kezdeti tanulási ráta hányadosa [0.01,1]
- numEpoch: az epochok száma [10-30]
- decay: a weight_decay paraméter értéke [1e-3,1e-7]

A feladat elején inicializáltam a pseudo random generátorokat a 42-es értékkel. Erre azért van szükség, hogy különböző futtatások esetén ugyanazokat a random értékeket adja értékül az egyes véletlenszerű értékekkel inicializált változóknak paramétereknek. Ezzel ki tudjuk küszöbölni azt, hogy az egyes futtatások során a különböző véletlen értékekből adódóan más eredményeket kapjunk.

Listdataset class:

Ezt követően szükségem volt egy saját Dataset osztályra, amelyben eltárolhatom mind a train, mind a validációs adatokat. Egy saját Dataset osztály létrehozásakor az ős Dataset osztálynak három függvényét kell felüldefiniálnunk, hogy használni tudjuk a saját osztályunkat. A három függvény az init(), a getitem() és a len() függvények.

Elsőként az init() függvényt implementáltam, mely paraméterként megkapja az adatbázis gyökérkönyvtárát, illetve opcionálisan a képméretet. Első lépésként egy listában elmentem az

alkönyvtárak neveit, hogy később ezeken keresztül el tudjam érni a képfileokat. Továbbá létrehozok egy-egy listát a képeknek, valamint a label-eknek.

A képfileokat a létrehozott listába illesztem egy beágyazott for ciklus segitségével. A külső ciklus az almappákon halad végig, míg a belső ciklus az almappában található képeken iterál végig. Minden egyes ilyen képfile-t beillesztek a listába. A labelek feldolgozása hasonlóképpen történik, azzal a különbséggel, hogy az adott label értéke az aktuális almappa indexe lesz (0-54 között).

Az inicializáció utolsó lépéseként elmentem a képnek a méretét, továbbá egy transzformációt is, amellyel a képfile-t tensorrá lehet alakítani.

A következőnek a getitem függvényt programoztam le. A függvény a paraméterként megkapott indexű képpel tér vissza. Ehhez elsőként az index alapján meghatározom a kép aktuális almappáját, a képfilet RGB formátumba konvertálom, majd végrehajtom rajta a transzformációt. Végül visszatérek a kép elérési útjával, az átkonvertált képpel és annak labeljével.

Utolsó feladatként a len függvényt hoztam lére, amely egyszerűen visszatér a lista hosszával.

Train függvény:

A második feladat érdemi része a train függvényben van megvalósítva, amely a neurális hálót meghatározó paramétereken túl rendelkezik a tanításra vonatkozó paraméterekkel is, valamint egy visu logikai változóval is, amellyel azt állíthatja be a felhasználó, hogy szeretné-e a tanitás során megjeleníteni az egyes változókat (a loss, valamint accuracy változását, mind a tanító, mind a validációs adatokon, továbbá a legjobb pontosságot és losst.

Elsőként szükség van arra, hogy az adatokat a gyorsabb feldolgozás érdekében egy iterálható objektumba tegyem, amely egyszerre egy batch-nyi adatot tud feldolgozni. E célból létre kell hozni egy-egy dataloadert mind a tanító, mind a validációs adatok számára, amelyeket a megfelelő értékekkel inicializálok. Fontos, hogy a tanító adatokon a train paraméter igaz értékű legyen.

Ezt követően létrehozok a saját neurális háló osztályomból egy objektumot a megfelelő paraméterekkel. Továbbá definiálom a loss függvényt, az optimalizáló algoritmust, valamint a tanulási ráta ütemezőt a feladat specifikációinak megfelelően. Ezután a megjelenítendő adatoknak létrehozom a szükséges változókat, adatstruktúrákat.

Következőnek jön a tanításhoz szükséges ciklus, amelyen belül a gradiens számítást, backpropagationt, és a validálást elvégzem. Nullázom a szükséges változókat, a hálómat train módba kapcsolom, majd minden egyes epochban végigiterálok a tanitó adatok struktúráján. A kinyert képeket és átadom a hálónak, a háló kimenete alapján veszteséget, gradienst számolok, majd az optimalizáló függvényemet léptetem. Közben kiszámítom a megjelenítendő változóimat is.

Ezt követően, az epochon belül validálni kell a validációs adatokon. A program felépítése hasonló, mint a tanítás során, azonban pár lényeges különbséget érdemes kiemelni. Fontos, hogy validálás során másik adatsoron dolgozunk, illetve, hogy nem számítunk gradienst. Emiatt nem kell a loss alapján gradienst számolnunk és az optimalizáló függvényt sem kell léptetni. A megjelenítendő adatokat itt is elmentjük, illetve ha az aktuális loss és accuracy jobb mint az eddigi legjobb, akkor a modell paramétereit kimentjük egy külön file-ba, hogy ezt később is el tudjuk érni, anélkül hogy újra be kelljen tanítani a modellt. Az epoch végén nem szabad elfeledkezni a tanulási ráta ütemező függvény léptetéséről sem.

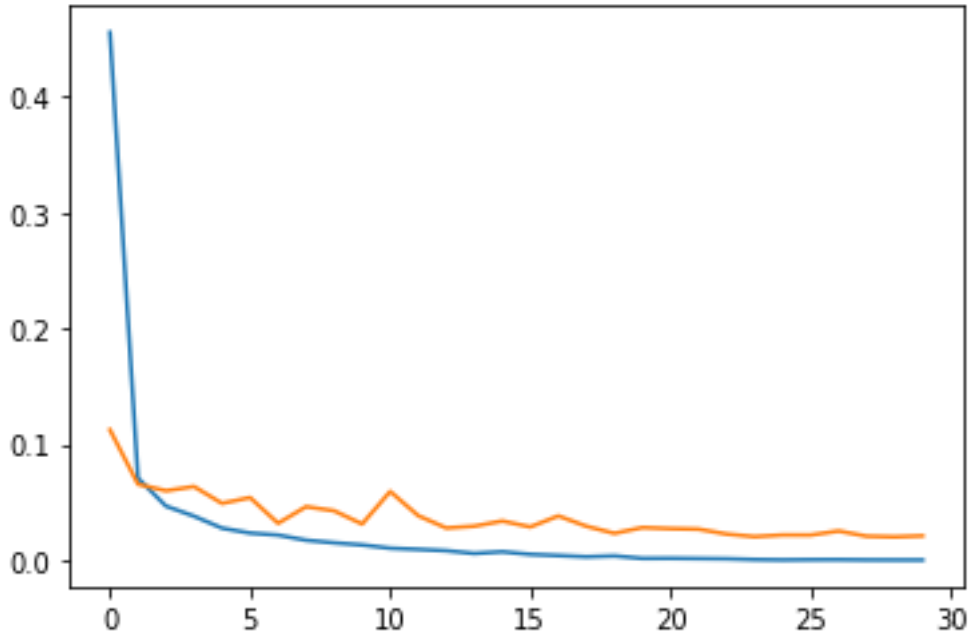
A train függvény végén, utolsó feladatunk, hogy ha szükségünk van az adatok megjelenítésére, akkor ezt megtegyük, továbbá, hogy visszatérjünk a tanítás legjobb pontosságával. A Bayesian Optimization során ezt a változót fogjuk maximalizálni.

Eredmények (nem optimalizált eset):

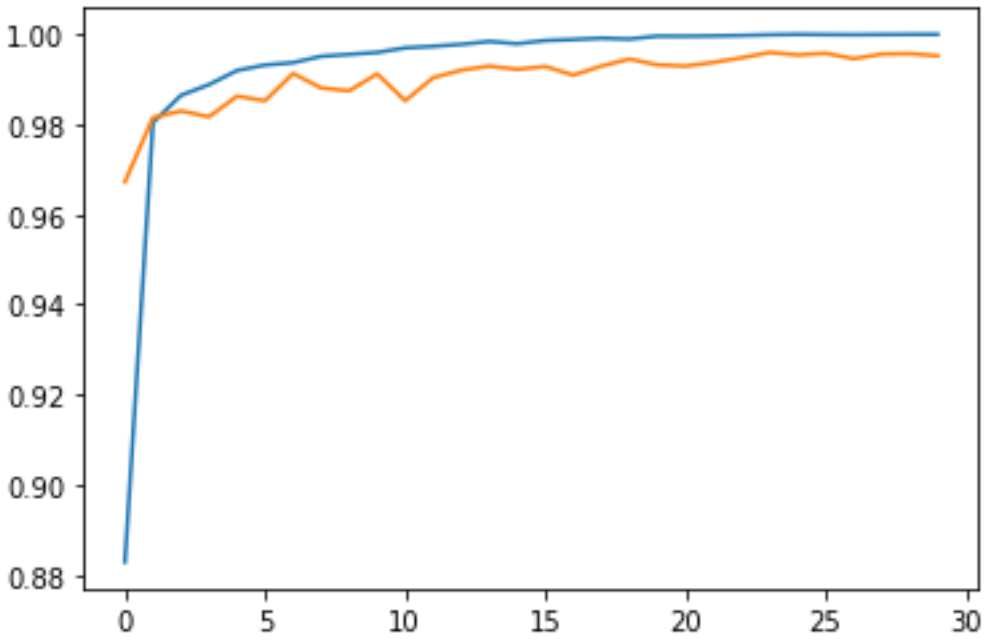
Miután teszteltem az eddig megírt programok helyességét elvégeztem a tanítást egy számomra megfelelőnek gondolt, de még nem optimalizált paraméterkombinációra. A kapott eredmények alább láthatóak, a paraméterek pontos értéke:

```
# running train function with default values before optimization
train(inplanes=3, nfeat=8, nc=55, nlevels=3, dropout=0.2, layersperlevel=3, kernel=5,
      nlin=1, bnorm=True, residual=True,
      bsize=64, lr=1e-3, lr_ratio=0.1, numepoch=30, decay=1e-5, visu=True)
```

Az eredményeken jól látható, hogy megfelelően történik a tanítás, a veszteség az epochok során csökken, amelyet a validációs veszteség követ, nem tapasztalható overfitting. Ugyanez elmondható a pontosságról is.



Tanító és validációs veszteség az epochok függvényében (30 epoch). Kékkel a tanítási, sárgával a validációs veszteség van jelölve.



Tanító és validációs pontosság az epochok függvényében (30 epoch). Kékkel a tanítási, sárgával a validációs pontosság van jelölve.

A függvényünk a tanítás során megjeleníti a legjobb validációs veszteség és pontosság értékeket is. Ezen számértékek mögöttes tartalmára érdemes további figyelmet fordítani.

Legjobb validációs veszteség: 0.020

Legjobb validációs pontosság: 0.996

A két adat közül a validációs pontosság a beszédesebb, ezért ezt fogom részletesebben elemezni. A validációs pontosság megmutatja, hogy olyan adatokon, amelyekkel a tanítás során nem találkozott a neurális háló (tehát nem látta még az adott képet), milyen pontossággal ismeri fel. A tanítási adatbázisom közlekedési táblákat tartalmaz a feladat során cél, hogy ezeket a közlekedési táblákat minél jobban felismerje a neurális háló.

Az egyes adatsorokon kapott validációs pontosság azonban önmagában nem értelmezhető, tudnunk kell azt is, hogy az adott adatsor mennyire könnyen tanulható, milyen könnyen lehet elérni rajta magas validációs pontosságot. A grafikon ábráján láthatjuk, hogy ez az adatsor viszonylag könnyen tanulható, hiszen a neurális háló már az első epoch után is 97% körüli pontossággal rendelkezett. Ezek alapján megállapíthatjuk, hogy a kapott 99.6%-os eredmény egyáltalán nem kiemelkedő ezen az adatsoron.

Gyakorlati alkalmazás lehet például, hogy egy önvezető autó, amely kamerákkal van felszerelve az úton haladva a szembejövő táblákat felismeri és sebességvektorán a táblából kinyert információ alapján változtat. A validációs pontosság 99.6%, ez azt jelenti, hogy 1000 szembejövő táblából 996-ot fel tud ismerni a neurális háló. Ez elsőre bőven megfelelő pontosságnak tűnhet azonban egy olyan biztonságkritikus és felelősségteljes környezetben, mint a közlekedés, bizony az a négy hibás felismerés is komoly következményekhez vezethet. Gondoljunk csak arra, hogy mi történhet akkor, ha a háló például

egy gyalogosátkelőhelyre figyelmeztető táblát összetéveszt egy 70 km/h-s sebességkorlátozó táblával. Nem elégedhetünk tehát meg ezzel a pontossággal sem.

Harmadik feladat:

Hogyan tudnánk az adatsor pontosságán tovább javítani? Egy neurális háló pontosságát nem csak az egyes neuronokat összekötő súlyok és offszet értékek (bias) határozza meg. A neurális hálóknak vannak úgynevezett hiperparaméterei, melyek szintúgy hatással vannak a háló pontosságára, mint a súlyok és offszetek. Ahhoz, hogy még jobb pontosságot érjünk el, ezeket a hiperparamétereket is optimalizálni kell. Ezek a hiperparaméterek a házi feladatom esetében a tanító függvény paraméterek. A házi feladat utolsó részében ezeket a paramétereket kell optimalizálni, melyet a Bayesian Optimization könyvtár segítségével fogok megtenni.

Bayesian optimization:

A bayesian optimization könyvtár segítségével fogom a hiperparamétereimet optimalizálni. Az optimalizáció a következőképpen foglalható össze röviden. A könyvtár egy általunk megadott függvény kimenetét maximalizálja a paramétereinek hangolásával. Ezen paraméterek határait nekünk kell megadni, észszerű választások alapján. Fontos megjegyezni, hogy a maximalizáló függvény csak folytonos paramétereket alkalmaz, ezért bármilyen nem folytonos értéket nekünk kell átkonvertálni. Az optimalizáció folytonos értékeket ad át a mi függvényünknek ezért, ha diszkrét paraméterek is vannak a függvényben, akkor azt a megfelelő módon kell folytonos tartományból diszkrétre leképezni.

Az optimalizáció során megadhatjuk, hogy hány random lépést hajtson végre mielőtt elkezdené optimalizálni az értékeket (legyen ennek a száma x). Az optimalizálandó paraméterek száma legyen n; ez az n paraméter meghatároz egy n változós függvényt, amelynek a maximumpontját igyekszik meghatározni. Az iterálás kezdetén x alkalommal ennek az n paraméternek véletlen értékeket ad a függvény. A függvény minden egyes paraméterkombinációra ad egy kimenetet, ezekre az optimalizáló algoritmus illeszt egy közelítő görbét. A következő iteráció során az illesztett görbe maximuma, valamint az ismeretlen kombinációk (pontok, tartományok) alapján határozza meg a következő paraméterkombinációt, amelyet be fog illeszteni a függvénybe. Az n ismétlés elvégzésével visszaadja azt a paraméterkombinációt, amellyel a legnagyobb kimenetet érte el. N iteráció után tehát ezek az értékek fogják a legjobb pontosságot adni, nekünk ezeket kell behelyettesíteni a függvényünkbe. Természetesen a diszkrét paramétereket konvertálni kell.

A feladat implementálása:

A feladat első lépéseként létrehoztam egy függvényt, ami az optimalizáló függvény folytonos paramétereit a megfelelő módon diszkrét értékké konvertálja. Általános esetben a folytonos értékeket egész számra kerekítettem, majd ezeket az egész számokat integer típussá konvertáltam. Ez alól kivétel az nFeat és batchSize paraméter, ahol kettő hatványára emelem a paramétereket, ezzel egyszerűsítve, pontosítva az algoritmust (kisebb tartományon könnyebben tud optimalizálni), valamint ezáltal a feladat specifikációjának is biztosan megfelelek. A paraméterek konvertálása után meghívom az eredeti tanító függvényt.

Következő lépésként minden egyes paraméterhez megadom azokat a határokat, amelyeken belül azok mozoghatnak. A maximalizáló függvény ezeket is meg fogja kapni. Ezután már csak az optimalizálás meghívására van szükség, valamint a program sikeres futtatására.

A futtatás sajnos sokszor problémába ütközött, az internetkapcsolat megszakadása miatt azelőtt megszakadt a kapcsolat, hogy a maximalizáló függvény le tudott volna futni. A probléma kiküszöbölésének érdekében az algoritmust ciklikus futásúvá alakítottam, kisebb ismétlésszámmal operálva az optimalizálás során. A könyvtár lehetőséget biztosít arra, hogy elmentsük az egyes iterációs lépések során kapott eredményeket, hogy azokat később felhasználhassuk és onnan folytathassuk az optimalizálást, ahol abbahagytuk. Ezt a futás során én is igénybe vettem. Minden ciklus iteráció elején betöltöttem az eddigi eredményeket, majd a ciklus végén elmentettem azokat.

Eredmények és értékelés:

Az optimalizálás futtatása után legjobb eredmény paraméterei az alábbiak:

Target	Batchsize	Decay	Dropout	Kernel	Layersperlevel	LR	LR_ratio	Nfeat	Nlvls	Res
0.98	7.24	1.0E-07	0.39	0.52	2.93	0.02	0.03	4.51	2.52	0.84

Folytonos értékek

Target	Batchsize	Decay	Dropout	Kernel	Layersperlevel	LR	LR_ratio	Nfeat	Nlvls	Res
0.98	128.00	1.0E-07	0.39	3.00	3.00	0.02	0.03	32.00	3.00	1.00

Konvertált értékek

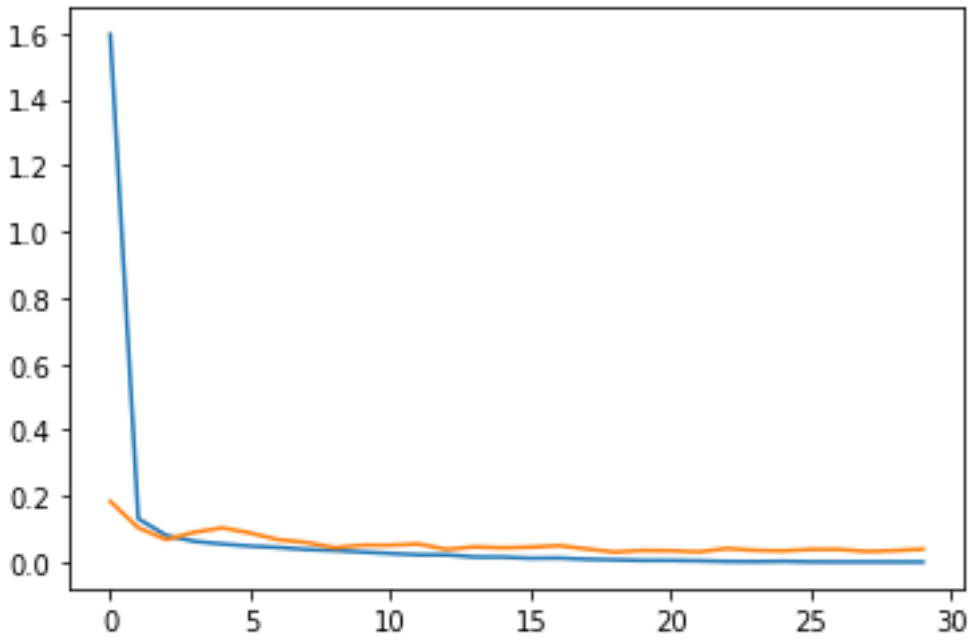
Ezen paraméterek mellett futtatott tanítás után a validációs pontosság 98%-os lett. Ezen eredmény mellett is érdemes elidőzni egy kicsit. Ugyan ez az érték kisebb, mint az optimalizálatlan tanítás során kapott 99.6%-os validációs pontosság ugyanakkor érdemes megjegyezni, hogy az optimalizálás során csak az adatbázis 20%-a volt felhasználva, valamint 30 epoch helyett csak 20 epochot végeztünk el (bár kisebb epochszám nem feltétlen jelent rosszabb eredményt). Ezen körülmények között tehát a 98%-os pontosság kiemelkedőnek számít.

Optimalizált tanítás:

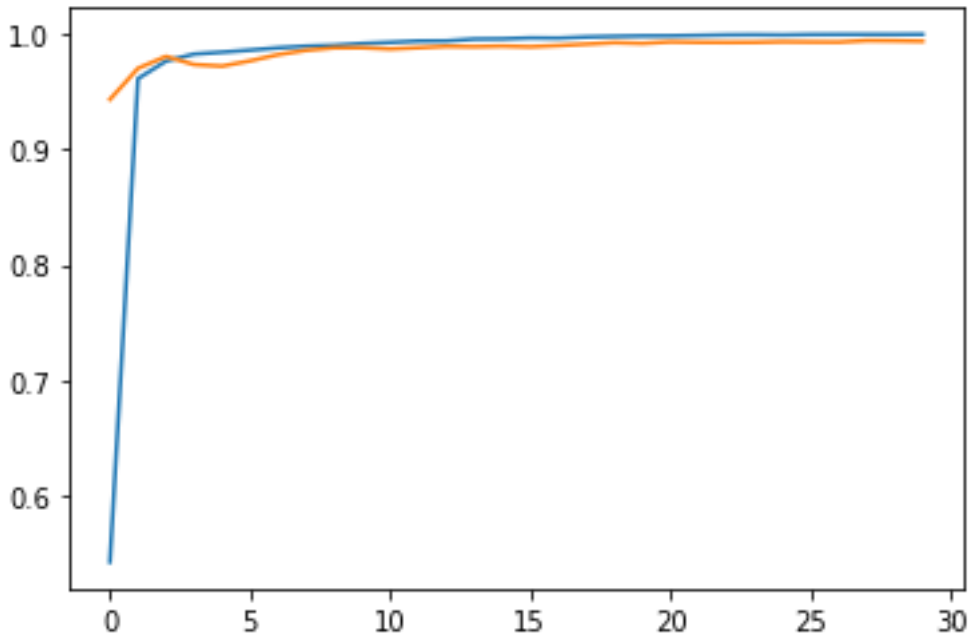
A házi feladat utolsó lépéseként megvizsgáltam, hogy a Bayesian Optimization alkalmazásával kapott hiperparaméter értékek segítségével milyen pontosságot tud elérni a neurális háló. Az optimalizált és nem optimalizált eredmények egyszerre készültek, azonos random seedek mellett. Az eredmények alább láthatóak.

Legjobb validációs veszteség optimalizált hiperparaméterek esetén: 0.030

Legjobb validációs pontosság optimalizált hiperparaméterek esetén: 0.995



Tanító és validációs veszteség optimalizált hiperparaméterek mellett, az epochok függvényében (30 epoch). Kékkel a tanítási, sárgával a validációs veszteség van jelölve.



Tanító és validációs pontosság optimalizált hiperparaméterek mellett, az epochok függvényében (30 epoch). Kékkel a tanítási, sárgával a validációs pontosság van jelölve.

Az eredmények elsőre meghökkentőnek tűnhetnek, hiszen az optimalizált hiperparaméterek esetén azt várnánk el a neurális hálótól, hogy nagyobb pontossággal operáljon, mint alapesetben. A jelenség oka az adatsorokban keresendő. Az optimalizálás során nem az adatsor egészét dolgozta fel a rendszer, hanem

annak egy véletlenszerűen kiválasztott 20%-val dolgozott, hogy gyorsabban lefusson az optimalizálás. (Megjegyzés: az optimalizálást Google Colaboratory-ban végeztem, a Google e célból fenntartott GPU egységein. Az optimalizálást sajnos nem tudtam volna elvégezni a teljes adatsoron, mivel a szolgáltatás egyszerre 12 óra folytonos GPU használatot engedélyez és a csökkentett adatsoron való optimalizálás is 8 óráig futott.) Ennek az információnak a birtokában már érthető, hogy miért nem kaptunk jobb eredményt, mint nem optimalizált esetben. A neurális háló pontossága tovább lehet javítható, ha a Bayesian Optimization maximalizáló függvényében, a hangolni kívánt paramétereket több körben szűkítjük, valamint, ha az egész adatsort használjuk, egy olyan GPU-n, ami ezzel a feladattal belátható időn belül végez.

Felhasznált irodalom, források:

A házi feladatom az alábbi linken elérhető:

https://colab.research.google.com/drive/15ULx1N3z6iz7rZQS1JtoIcifiFWgRFDD?usp=sharing

A házi feladat megoldása során az alábbi forrásokat, irodalmat használtam fel:

- https://github.com/fmfn/BayesianOptimization
- https://pytorch.org/docs/stable/index.html
- https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py
- https://arxiv.org/pdf/1512.03385.pdf
- https://cs231n.github.io/convolutional-networks/

Függelék:

Az optimalizálás során kapott eredmények:


```
{'target': 0.984375, 'params': {'bsize': 7.237021869453782, 'decay': 1e-07, 'dout':
0.39425003781945933, 'krnl': 0.5151446914476931, 'layersplvl': 2.931889301051508, 'lr':
0.01785396782364612, 'lr_ratio': 0.034282373093213095, 'nf': 4.509489481011639, 'nlvls':
2.5159813604018852, 'resid': 0.842958891338791}}
Iteration 0:
	{'target': 0.11580882352941177, 'params': {'bsize': 4.502132028215444, 'decay':
0.0002797475390071862, 'dout': 5.718740867244332e-05, 'krnl': 0.6046651452636795, 'layersplvl':
1.293511781634226, 'lr': 0.009242936090932092, 'lr_ratio': 0.1943976092638942, 'nf':
3.036682181129143, 'nlvls': 3.19030242269201, 'resid': 0.538816734003357}}
Iteration 1:
	{'target': 0.03125, 'params': {'bsize': 4.515167086419769, 'decay': 0.0003148490215532802,
'dout': 0.10222612486575872, 'krnl': 1.7562348727818908, 'layersplvl': 1.0547751863958523, 'lr':
0.06705004634273844, 'lr_ratio': 0.4231317543434557, 'nf': 3.676069485337255, 'nlvls':
2.421160815785701, 'resid': 0.1981014890848788}}
Iteration 2:
	{'target': 0.9126838235294118, 'params': {'bsize': 6.80446741205322, 'decay':
3.183525043817445e-05, 'dout': 0.15671208907962142, 'krnl': 1.3846452313386282, 'layersplvl':
2.7527783045920766, 'lr': 0.0894617202837497, 'lr_ratio': 0.09419376925608013, 'nf':
2.117164349698647, 'nlvls': 2.5094912586937066, 'resid': 0.8781425034294131}}
Iteration 3:
	{'target': 0.023809523809523808, 'params': {'bsize': 2.590081002998301, 'decay':
0.0005789344857574484, 'dout': 0.47894476507525097, 'krnl': 1.0663305699460341, 'layersplvl':
2.3837542279009467, 'lr': 0.03155840794429624, 'lr_ratio': 0.6896359184047678, 'nf':
4.503877015692119, 'nlvls': 2.0548648320325755, 'resid': 0.7501443149449675}}
Iteration 4:
	{'target': 0.03759765625, 'params': {'bsize': 7.9331665334389685, 'decay':
0.00025190916218559864, 'dout': 0.1402219960322026, 'krnl': 1.578558656902977, 'layersplvl':
1.206452013155284, 'lr': 0.04479487368232876, 'lr_ratio': 0.9095095480621646, 'nf':
2.8808424451210386, 'nlvls': 2.8633260157590463, 'resid': 0.13002857211827767}}
Iteration 5:
```

{'target': 0.0234375, 'params': {'bsize': 5.0455872979581855, 'decay': 1e-07, 'dout': 0.4134291932644383, 'krnl': 1.9583845573841177, 'layersplvl': 2.911953109266925, 'lr': 0.03750325440786954, 'lr_ratio': 0.10189853247813654, 'nf': 2.0548674369382423, 'nlvls': 4.72676720945217, 'resid': 0.8001448584298431}}
Iteration 6:
{'target': 0.984375, 'params': {'bsize': 7.237021869453782, 'decay': 1e-07, 'dout': 0.39425003781945933, 'krnl': 0.5151446914476931, 'layersplvl': 2.931889301051508, 'lr': 0.01785396782364612, 'lr_ratio': 0.034282373093213095, 'nf': 4.509489481011639, 'nlvls': 2.5159813604018852, 'resid': 0.842958891338791}}
Iteration 7:
{'target': 0.9404296875, 'params': {'bsize': 7.722469589435457, 'decay': 1e-07, 'dout': 0.3131423784548188, 'krnl': 0.1741158638412137, 'layersplvl': 2.6526478118513825, 'lr': 0.045857526947038256, 'lr_ratio': 0.8345261866940961, 'nf': 2.251222235706435, 'nlvls': 2.041184249006989, 'resid': 0.9551726101720973}}
Iteration 8:
{'target': 0.9439338235294118, 'params': {'bsize': 5.671156630540395, 'decay': 1e-07, 'dout': 0.33181984176154034, 'krnl': 0.05557245307263581, 'layersplvl': 2.9963810781724316, 'lr': 0.04930784827735077, 'lr_ratio': 0.6314422950460323, 'nf': 2.129710982051293, 'nlvls': 2.012897824927774, 'resid': 0.1907097404299405}}
Iteration 9:
{'target': 0.97216796875, 'params': {'bsize': 7.9612142211893655, 'decay': 1e-07, 'dout': 0.2899138556794879, 'krnl': 0.08438254543233725, 'layersplvl': 2.948423003382014, 'lr': 0.08013215977248024, 'lr_ratio': 0.06139199341510931, 'nf': 2.7005826080072666, 'nlvls': 2.1107131647234496, 'resid': 0.5844934380968575}}
Iteration 10:
{'target': 0.6936813186813187, 'params': {'bsize': 2.1162017472217824, 'decay': 0.0003212323506134031, 'dout': 0.10581405800002952, 'krnl': 0.5310933187444524, 'layersplvl': 1.9831463185606766, 'lr': 0.005345720886256867, 'lr_ratio': 0.578376429437093, 'nf': 2.4401857247174306, 'nlvls': 3.767916610709853, 'resid': 0.6997583600209312}}
Iteration 11:
{'target': 0.023809523809523808, 'params': {'bsize': 2.614006572966955, 'decay': 0.0005859854177792137, 'dout': 0.34720007886387255, 'krnl': 0.8283585390538053, 'layersplvl': 1.0999069178921743, 'lr': 0.05359428162749201, 'lr_ratio': 0.667156698767591, 'nf': 3.5446673361749257, 'nlvls': 4.83378426797244, 'resid': 0.5865550405019929}}
Iteration 12:
{'target': 0.8203125, 'params': {'bsize': 7.420411491727301, 'decay': 0.0008625390433241771, 'dout': 0.06963817362537927, 'krnl': 1.6147825774190476, 'layersplvl': 1.7953536739710672, 'lr': 0.01654376616972211, 'lr_ratio': 0.9282334945920736, 'nf': 3.0432975792365196, 'nlvls': 4.252436309408466, 'resid': 0.7259979853504515}}
Iteration 13:
{'target': 0.6590073529411765, 'params': {'bsize': 7.299836547234859, 'decay': 0.0003763901601650967, 'dout': 0.3754712170136686, 'krnl': 0.6977966839556851, 'layersplvl': 1.5398557835300521, 'lr': 0.08958966295742472, 'lr_ratio': 0.433810277972582, 'nf': 4.894520141445157, 'nlvls': 3.9903244934553443, 'resid': 0.6216957202091218}}
Iteration 14:
{'target': 0.8566849816849816, 'params': {'bsize': 2.688475837720251, 'decay': 5.060569021879955e-05, 'dout': 0.22495606673997026, 'krnl': 1.1567792287742635, 'layersplvl': 1.8162736055225623, 'lr': 0.023710327754500344, 'lr_ratio': 0.9043457253566313, 'nf': 3.7210384600168576, 'nlvls': 2.0086109810934767, 'resid': 0.6171449136207239}}
Iteration 15:
{'target': 0.023351648351648352, 'params': {'bsize': 2.321827270676101, 'decay': 1e-07, 'dout': 0.3950368852904198, 'krnl': 1.9249552667930523, 'layersplvl': 1.2963017800241448, 'lr': 0.06789199074748424, 'lr_ratio': 0.9379670957279277, 'nf': 2.0157739473681713, 'nlvls': 2.305646656566389, 'resid': 0.8557602787689411}}
Iteration 16:
{'target': 0.97216796875, 'params': {'bsize': 7.7544029348816945, 'decay': 1e-07, 'dout': 0.3897691348056893, 'krnl': 0.18001829477167397, 'layersplvl': 2.9911252513920377, 'lr': 0.02700206557260425, 'lr_ratio': 0.9161196686401734, 'nf': 3.7710810186735158, 'nlvls': 4.938586495626724, 'resid': 0.9029131328836629}}
Iteration 17:
{'target': 0.091796875, 'params': {'bsize': 7.870142474764983, 'decay': 1e-07, 'dout': 0.1828553347877898, 'krnl': 1.9660362793333541, 'layersplvl': 2.8192481936023555, 'lr': 0.04960592811729158, 'lr_ratio': 0.77958011587244, 'nf': 4.653123791825708, 'nlvls': 4.693631894136592, 'resid': 0.3627550562989196}}
Iteration 18:
{'target': 0.0205078125, 'params': {'bsize': 7.756340941980184, 'decay': 1e-07, 'dout': 0.494348636157412, 'krnl': 0.007117774551240955, 'layersplvl': 1.3253051767251154, 'lr': 0.09957192224354619, 'lr_ratio': 0.5352148229423522, 'nf': 2.1290185005386357, 'nlvls': 4.941173044617394, 'resid': 0.6368889052457971}}
Iteration 19:

{'target': 0.02197802197802198, 'params': {'bsize': 2.23806347801632, 'decay': 1e-07, 'dout': 0.08716563272933187, 'krnl': 0.06566626325025005, 'layersplvl': 2.8833145957897024, 'lr': 0.03087720447786347, 'lr_ratio': 0.5536426386628067, 'nf': 2.1284269464687453, 'nlvls': 2.0654213628093805, 'resid': 0.6170986871735926}}
Iteration 20:
{'target': 0.02619485294117647, 'params': {'bsize': 3.959869410632577, 'decay': 0.00047299460355261654, 'dout': 0.44297104965538725, 'krnl': 0.7145395200049995, 'layersplvl': 2.8170703018395984, 'lr': 0.062339777978022363, 'lr_ratio': 0.025663030418090718, 'nf': 4.788311701231284, 'nlvls': 4.072690752550772, 'resid': 0.9973228504514805}}
Iteration 21:
{'target': 0.023351648351648352, 'params': {'bsize': 3.034043050071971, 'decay': 0.0008628779639460851, 'dout': 0.4662977315185818, 'krnl': 1.3936363229798003, 'layersplvl': 1.132000345444125, 'lr': 0.07554875062972062, 'lr_ratio': 0.7563374265766339, 'nf': 4.76907360663945, 'nlvls': 4.134574275885415, 'resid': 0.12427096197216 47}}
Iteration 22:
{'target': 0.022435897435897436, 'params': {'bsize': 2.1192808030387735, 'decay': 0.0009737916342209685, 'dout': 0.014153244010397303, 'krnl': 0.4924221352060918, 'layersplvl': 2.720055897365776, 'lr': 0.053887718123521874, 'lr_ratio': 0.5572937588989082, 'nf': 4.526092677078817, 'nlvls': 2.372519945359733, 'resid': 0.2791836790111395}}
Iteration 23:
{'target': 0.9673713235294118, 'params': {'bsize': 5.5145556287497275, 'decay': 3.0501211255157517e-05, 'dout': 0.280515109627855, 'krnl': 0.03729457874588604, 'layersplvl': 2.6012653453612327, 'lr': 0.023305097641363636, 'lr_ratio': 0.8090341436625913, 'nf': 3.1635819321925154, 'nlvls': 4.590625563678286, 'resid': 0.7471216427371846}}
Iteration 24:
{'target': 0.03768382352941176, 'params': {'bsize': 5.337441403942513, 'decay': 0.0008635584198618811, 'dout': 0.02995884475610583, 'krnl': 0.2426869114814747, 'layersplvl': 1.0891037570895234, 'lr': 0.01075833796931823, 'lr_ratio': 0.23345224522177616, 'nf': 4.1389669411480305, 'nlvls': 3.6791509461624274, 'resid': 0.012555980159115854}}
Iteration 25:
{'target': 0.9770220588235294, 'params': {'bsize': 6.94401966595299, 'decay': 1e-07, 'dout': 0.035107600519242, 'krnl': 0.3291071324935888, 'layersplvl': 2.934571796805402, 'lr': 0.07688707726768979, 'lr_ratio': 0.9114363219811024, 'nf': 3.4767752230660984, 'nlvls': 3.0069347128810864, 'resid': 0.8686229842171758}}
Iteration 26:
{'target': 0.9641544117647058, 'params': {'bsize': 6.618411733977497, 'decay': 1e-07, 'dout': 0.380480305814726, 'krnl': 0.08447312119133144, 'layersplvl': 2.985674415532273, 'lr': 0.04458908376289840 6, 'lr_ratio': 0.6844365144783187, 'nf': 4.934562860225051, 'nlvls': 4.749059723974707, 'resid': 0.9361967595119015}}
Iteration 27:
{'target': 0.15384615384615385, 'params': {'bsize': 2.5087692327613436, 'decay': 1e-07, 'dout': 0.03548530808830325, 'krnl': 0.2924746336327939, 'layersplvl': 1.0802133489409196, 'lr': 0.027127001228904574, 'lr_ratio': 0.9793852268804712, 'nf': 3.4929852197052376, 'nlvls': 2.0263093722986545, 'resid': 0.14638026181615715}}
Iteration 28:
{'target': 0.0260989010989011, 'params': {'bsize': 2.4482413778846213, 'decay': 1e-07, 'dout': 0.14561129648430954, 'krnl': 1.1234688340345522, 'layersplvl': 2.848865312356356, 'lr': 0.07995520462098911, 'lr_ratio': 0.8692775419700687, 'nf': 2.0872134288300948, 'nlvls': 4.814323378479228, 'resid': 0.0033828058462329613}}
Iteration 29:
{'target': 0.13720703125, 'params': {'bsize': 7.691636382948783, 'decay': 1e-07, 'dout': 0.036808027354822026, 'krnl': 1.984304116843852, 'layersplvl': 2.969838297521097, 'lr': 0.09285952063311798, 'lr_ratio': 0.03614146752816978, 'nf': 2.084886871042621, 'nlvls': 4.1504554574145365, 'resid': 0.6390278506733504}}
Iteration 30:
{'target': 0.2403846153846154, 'params': {'bsize': 2.431845678136921, 'decay': 3.2820397632728096e-05, 'dout': 0.28405023095997106, 'krnl': 0.40658646932198095, 'layersplvl': 1.5046514891406468, 'lr': 0.07438514714896854, 'lr_ratio': 0.2034751862982256, 'nf': 3.744076781819773, 'nlvls': 4.910059967264937, 'resid': 0.8468288014900353}}
Iteration 31:
{'target': 0.027472527472527472, 'params': {'bsize': 3.4390865548855167, 'decay': 0.000506279662702697, 'dout': 0.3099778591906899, 'krnl': 1.6579617991003575, 'layersplvl': 1.3135827892921685, 'lr': 0.001867434455719178, 'lr_ratio': 0.0793219222820301, 'nf': 3.459035332811095, 'nlvls': 3.818988384959991, 'resid': 0.5688514370864813}}
Iteration 32:
{'target': 0.024356617647058824, 'params': {'bsize': 3.9041744559329645, 'decay': 1.1482707202992506e-05, 'dout': 0.28987260962289846, 'krnl': 0.7602823452471008, 'layersplvl': 2.1018964382357934, 'lr': 0.07453598974634115, 'lr_ratio': 0.6725405645186527, 'nf': 2.7947586729884284, 'nlvls': 2.199004503285325, 'resid': 0.3700841979141063}}
Iteration 33:

{'target': 0.7008272058823529, 'params': {'bsize': 5.778305042129387, 'decay': 0.0007898470074861519, 'dout': 0.37637777686940693, 'krnl': 0.13307296270822988, 'layersplvl': 1.520630197157082, 'lr': 0.08047740882869711, 'lr_ratio': 0.20149993979709446, 'nf': 3.9183826426398203, 'nlvls': 3.574010927371201, 'resid': 0.9248079703993507}}
Iteration 34:
{'target': 0.02389705882352941, 'params': {'bsize': 3.579780622922666, 'decay': 0.0009340455054250446, 'dout': 0.36753298164433473, 'krnl': 1.5443560590864935, 'layersplvl': 2.815631705007048, 'lr': 0.09319788719899176, 'lr_ratio': 0.023812057245841046, 'nf': 2.7030862583642614, 'nlvls': 3.850335071004973, 'resid': 0.9490163206876164}}
Iteration 35:
{'target': 0.21507352941176472, 'params': {'bsize': 7.352493760177435, 'decay': 1e-07, 'dout': 0.02327464623278125, 'krnl': 1.7823916816773122, 'layersplvl': 1.355921449608196, 'lr': 0.014736070328874386, 'lr_ratio': 0.04598857647256503, 'nf': 4.982511968101191, 'nlvls': 2.1013910166472036, 'resid': 0.9249875553498398}}
Iteration 36:
{'target': 0.974609375, 'params': {'bsize': 7.972386571207242, 'decay': 1e-07, 'dout': 0.2677736555061968, 'krnl': 0.025619447964010833, 'layersplvl': 2.6400538629711363, 'lr': 0.08447392434825744, 'lr_ratio': 0.746166582383823, 'nf': 4.72202869616796, 'nlvls': 2.039088027846848, 'resid': 0.9336619664981434}}
Iteration 37:
{'target': 0.9825367647058824, 'params': {'bsize': 6.6596394096101665, 'decay': 1e-07, 'dout': 0.0037314231501740514, 'krnl': 0.044635112069614014, 'layersplvl': 2.631706484937782, 'lr': 0.015363230088383516, 'lr_ratio': 0.2702482170245912, 'nf': 2.41593802960119, 'nlvls': 3.911740819679805, 'resid': 0.9739263804918097}}
Iteration 38:
{'target': 0.021599264705882353, 'params': {'bsize': 6.340985483243448, 'decay': 1e-07, 'dout': 0.2760354021221828, 'krnl': 1.4249461817602984, 'layersplvl': 1.0638417996114897, 'lr': 0.054092278599585625, 'lr_ratio': 0.9276983632894047, 'nf': 3.9453766012770393, 'nlvls': 4.9522848738359215, 'resid': 0.9787217582228636}}
Iteration 39:
{'target': 0.0695970695970696, 'params': {'bsize': 2.245725414057207, 'decay': 1e-07, 'dout': 0.007245710437668973, 'krnl': 1.7907022592761221, 'layersplvl': 1.0831334154358918, 'lr': 0.015220990106435978, 'lr_ratio': 0.6310820683396945, 'nf': 4.156487307829557, 'nlvls': 2.0267078848693396, 'resid': 0.407369227356754 6}}
New optimizer is now aware of 40 points.