Matematično-fizikalni praktikum

12. naloga: Strojno učenje (machine learning)

Gregor Žunič

Dandanes je uporaba različnih algoritmov strojnega učenja (Machine Learning, ML) v znanosti že rutinsko opravilo. Poznamo tri osnovne vrste stojnega učenja:

- Nadzorovano učenje (Supervised learning):
  - Klasifikacija (Classification): sortiranje v različne kategorije.
  - Regresija (Regression): modeliranje oz. 'fitanje' napovedi.
- Nenadzorovano učenje (npr. sam najdi kategorije).
- Stimulirano učenje (Artificial Intelligence v ožjem pomenu besede).

V fiziki (in tej nalogi), se tipično ukvarjamo s prvo kategorijo, bodisi za identifikacijo novih pojavov/delcev/... ali pa za ekstrakcijo napovedi (netrivialnih funkcijskih odvisnosti etc).

ML algoritmi imajo prednost pred klasičnim pristopom, da lahko učinkovito razdrobijo kompleksen problem na enostavne elemente in ga ustrezno opišejo:

- pomisli na primer, kako bi bilo težko kar predpostaviti/uganiti pravo analitično funkcijo v več dimenzijah (in je npr. uporaba zlepkov (spline interpolacija) mnogo lažja in boljša).
- Pri izbiri/filtriranju velike količine podatkov z mnogo lastnostmi (npr dogodki pri trkih na LHC) je zelo težko najti količine, ki optimalno ločijo signal od ozadnja, upoštevati vse korelacije in najti optimalno kombinacijo le-teh...

Če dodamo malce matematičnega formalizma strojnega učenja: Predpostavi, da imamo na voljo nabor primerov $\mathcal{D} = \{(\mathbf{x}_k, y_k)\}_{k=1.N}$, kjer je $\mathbf{x}_k = \left(x_k^1, \ldots, x_k^M\right)$ naključno izbrani vektor M lastnosti (karakteristik) in je $\mathbf{y}_k = \left(y_k^1, \ldots, y_k^Q\right)$ vektor Q ciljnih vrednosti, ki so lahko bodisi binarne ali pa realna števila[1]. Vrednosti $(\mathbf{x}_k, \mathbf{y}_k)$ so neodvisne in porazdeljene po neki neznani porazdelitvi $P(\cdot, \cdot)$. Cilj ML metode je določiti (priučiti) funkcijo $h : \mathbb{R}^Q \to \mathbb{R}$, ki minimizira pričakovano vrednost *funkcije izgube (expected loss)*

$$\mathcal{L}(h) = \mathbb{E}\, L(\mathbf{y}, \mathbf{h}(\mathbf{x})) = \frac{1}{N} \sum_{k=1}^{N} L(\mathbf{y_k}, \mathbf{h}(\mathbf{x_k})).$$

Tu je $L(\cdot, \cdot)$ gladka funkcija, ki opisuje oceno za kvaliteto napovedi, pri čemer so vrednosti $(\mathbf{x}, \mathbf{y})$ neodvisno vzorčene iz nabora $\mathcal{D}$ po porazdelitvi P. Po koncu učenja imamo torej na voljo funkcijo $\mathbf{h}(\mathbf{x})$, ki nam za nek vhodni nabor vrednosti $\hat{\mathbf{x}}$ poda napoved $\hat{\mathbf{y}} = \mathbf{h}(\hat{\mathbf{x}})$, ki ustrezno kategorizira ta nabor vrednosti.

Funkcije $\mathbf{h}$ so v praksi sestavljene iz (množice) preprostih funkcij z (nekaj) prostimi parametri, kar na koncu seveda pomeni velik skupni nabor neznanih parametrov in zahteven postopek minimizacije funkcije izgube.

[1] ...ali pa še kaj, prevedljive na te možnosti, npr barve...

Osnovni gradnik odločitvenih dreves je tako kar stopničasta funkcija $H(x_i - t_i) = 0, 1$, ki je enaka ena za $x_i > t_i$ in nič drugače in kjer je x_i ena izmed karakteristik in t_i neznani parameter. Iz skupine takšnih funkcij, ki predstavljajo binarne odločitve lahko skonstruiramo končno uteženo funkcijo

$$\mathbf{h}(\mathbf{x}) = \sum_{i=1}^{J} \mathbf{a}_i \, H(x_i - t_i),$$

kjer so $\mathbf{a}_i$ vektorji neznanih uteži. Tako t_i kot $\mathbf{b}_i$, lahko določimo v procesu učenja. Nadgradnjo predstavljajo nato *pospešena* odločitvena drevesa (BDT), kjer nadomestimo napoved enega drevesa z uteženo množico le-teh, tipično dobljeno v ustreznih iterativnih postopkih (npr. AdaBoost, Gradient Boost ipd.).

Pri nevronskih mrežah je osnovni gradnik t.i. *perceptron*, ki ga opisuje preprosta funkcija

$$h_{w,b}(\mathbf{X}) = \theta \left(\mathbf{w}^T \cdot \mathbf{X} + b \right),$$

kjer je $\mathbf{X}$ nabor vhodnih vrednosti, $\mathbf{w}$ vektor vrednosti uteži, s katerimi tvorimo uteženo vsoto ter b dodatni konstatni premik (bias). Funkcija θ je preprosta gladka funkcija (npr. arctan), ki lahko vpelje nelinearnost v odzivu perceptrona. Nevronska mreža je nato sestavljena iz (poljubne) topologije takšnih perceptronov, ki na začetku sprejme karakteristiko dogodka $\mathbf{x}$ v končni fazi rezultirajo v napovedi $\hat{\mathbf{y}}$, ki mora seveda biti čim bližje ciljni vrednosti $\mathbf{y}$. Z uporabo ustrezne funkcije izgube (npr MSE: $\mathcal{L}(h) = \mathbb{E}\,||\mathbf{y} - \hat{\mathbf{y}}||^2$), se problem znova prevede na minimizacijo, kjer iščemo optimalne vrednosti (velikega) nabora uteži $\mathbf{w}_i$ ter b_i za vse perceptrone v mreži. Globoke nevronske mreže (DNN) niso nič drugega, kot velike nevronske mreže ali skupine le-teh.

Že namizni računalniki so dovolj močni za osnovne računske naloge, obstajajo pa tudi že zelo uporabniku prijazni vmesniki v jeziku Python, na primer:

- Scikit-Learn (*scikit-learn.org*): odprtokodni paket za strojno učenje,
- TensorFlow (*tensorflow.org*): odprtokodni Google-ov sistem za ML, s poudarkom na globokih nevronskih mrežah (Deep Neural Networks, DNN) z uporabo vmesnika Keras. Prilagojen za delo na GPU in TPU.
- Catboost: (*Catboost.ai*) : odprtokodna knjižnica za uporabo pospešenih odločitvenih dreves (Boosted Decision Trees, BDT). Prilagojena za delo na GPU.

Za potrebe naloge lahko uporabimo tudi spletni vmesnik Google Collab (*colab.research.google.com*), ki dopušča omejen dostop do večjih računskih zmogljivosti.

Naloga: Na spletni učilnici je na voljo material (koda, vzorci) za ločevanje dogodkov Higgsovega bozona od ostalih procesov ozadja. V naboru simuliranih dogodkov je 18 karakteristik (zveznih kinematičnih lastnosti), katerih vsaka posamezno zelo slabo loči 'signal' od ozadja, z uporabo BDT ali (D)NN, pa lahko tu dosežemo zelo dober uspeh. Na predavanjih smo si ogledali glavne aspekte pomembne pri implementaciji ML, kot so uporaba ustreznih spremenljivk (GIGO), učenje in prekomerno učenje (training/overtraining), vrednotenje uspeha metode kot razmerje med učinkovitostjo (efficiency) in čistostjo (precision) vzorca (Receiver Operating Characteristic, ROC). Določi uspešnost obeh metod (in nariši ROC) za nekaj tipičih konfiguracij BDT in DNN, pri čemer:

- Študiraj vpliv uporabljenih vhodnih spremenljivk - kaj, če vzamemo le nekatere?
- Študiraj BDT in NN in vrednoti uspešnost različnih nastavitev, če spreminjaš nekaj konfiguracijskih parametrov (npr. število perceptronov in plasti nevronskih mrež pri DNN in število dreves pri BDT).

Dodatna Naloga: Implementiraj distribucije iz 'playground' zgleda v BDT (lahko tudi RandomForests) in DNN, te distribucije so na voljo v vseh popularnih ML paketih (npr. Scikit...).

Lahko si izbereš/izmisliš še kakšnega - npr. v vseh paketih je že na voljo standardni 'moons' dataset.

1 Reševanje

V duhu mafijskega praktikuma sem se odločil da je potrebno nevronsko mrežo zares dobro razumeti, zato sem NN sprogramiral sam[2]. Moj program podpira dodajanje poljubno število layerjev in podpira nekaj različnih aktivacijskih funkcij. Žal je ADAM algoritem zelo zakompliciran za implementacijo v obliki, kot sem sestavil mrežo, zato se mreža uči s pomočjo back-propagation-a.

Glede same implementacije ne morem povedati veliko več, kot je napisano v navodilu, ker se že iz samega navodila da zgraditi popolnoma delujočo mrežo.

2 Primerjava moje implementacije in DNN Tensorflowa

Pri obeh primerih sem najprej naredil mrežo z enakimi osnovnimi parametri, ki vsebujeta 2 skrite plasti s 50 nevroni in ReLu aktivacijsko funkcijo, ter sigmoidno funkcijo za izhodno plast. Tudi velikost batchov in hitrost učenja sta bila nastavljena na enako vrednost, vendar je še vedno Tensorflow implementacija dosegla precej boljše rezultate. Edina razlika nastopi v fazi učenja, ker Tensorflow implementira sam po sebi še dosti optimizacijskih metod (eden od primerov je dropout) ter podpira utilizacijo vseh jeder oziroma GPUja (če je ta na voljo). Tega seveda nebom napisal sam, ker je GPU programiranje zelo zahtevno in ni smiselno. Zdaj lahko primerjam moj model z modelom Tensorflowa, razlike v performanci so vidne na sliki 1.

Accuracy sem preprosto implementiral vsi pravilni deljeni z vsemi meritvami, oziroma malce bolj formalno se glasi $A = \frac{TP+TN}{TOTAL}$. AUC (integral po ROC) je malo težje na roke povedati, zato sem ta del prepustil Tensorflowu.

Iz slike 1 je očitno, da ima Tensorflow pri enakih parametrih precej bolj učinkovito, in boljše učenje. Tudi odvod accuracy po epochih ne konvergira, kar pomeni, da ima Tensorflow še precej maneverskega prostora!.

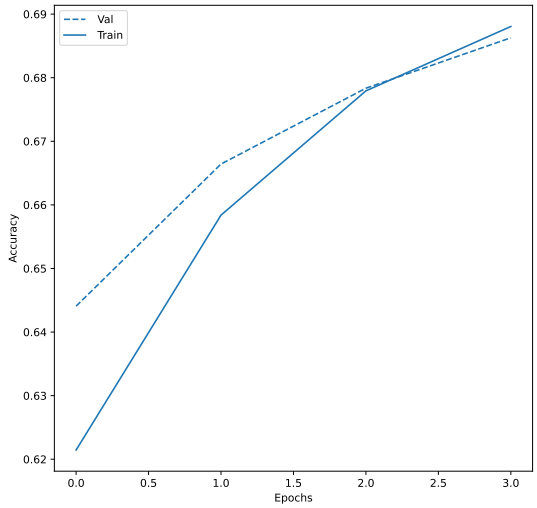
Zanimivo pa je, da je training čas moje mreže (40 sekund, za že prej navedene parametre) bil precej boljši, kot treniranje s Tensorflowom (120 sekund). Verjetno je bottleneck v treniranju hitrost prenašanja podatkov iz CPU v GPU, ker za našo mrežo (ki ni zelo potratna) ne potrebujemo močnih mašin. Za to analizo nisem šel preveč v podrobnosti zaradi veliko razlogov. Najbolj pomemben razlog je, da je moj računalnik zelo močen na prostoru CPUja (m1 pro procesor), GPU pa postane superioren šele pri velikih dimenzijah (miljoni, miljarde parametrov +). To pomeni, da podatek časa učenja ni najboljši, ker je v njem preveč faktorjev. Malo iz teme, zanimiv podatek je, da je moj GPU skoraj tako hiter, kot Nvidia Tesla T4, ki je zelo sposobna grafična kartica. To je posledica novih Applovih tensor jeder v grafični kartici. Je pa lepo videti, da je računalnik skoraj enako hiter kot grafična, ki sama po sebi stane več, kot cel MacBook Pro.

Nepresenetljiv sklep je, da je Tensorflow precej boljši, vendar se tudi moja mreža očitno zna učiti!.

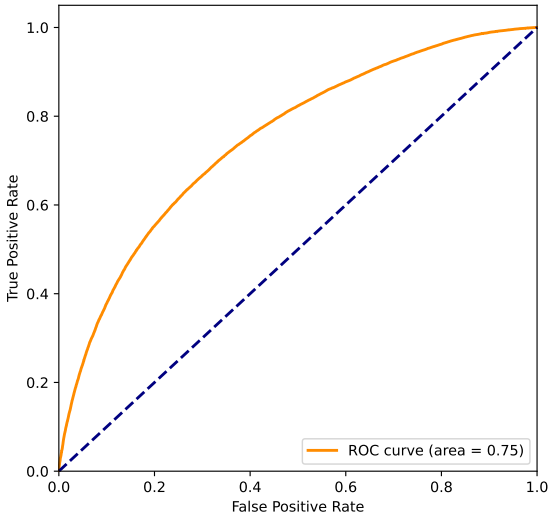
3 Analiza vhodnih dimenzij

Katere dimenzije so za uspešno klasifikacijo najbolj pomembne? Lažje je seveda predstaviti, katere dimenzije popolnoma zmedejo nasprotnika. To so dimenzije, ki naredijo največjo razliko na loss function, če jih pokvarimo.

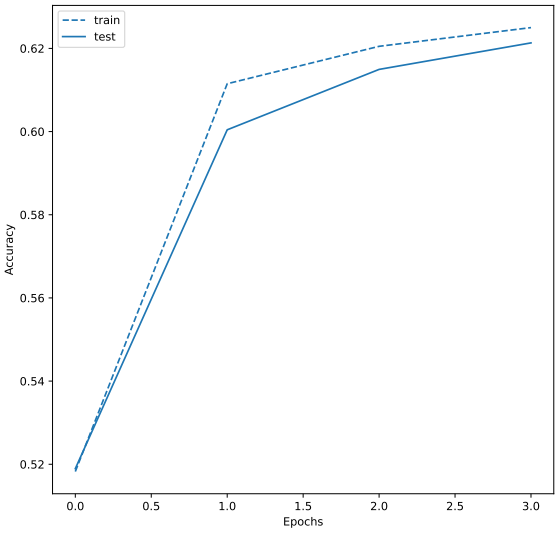
[2]program za nevronsko mrežo sem v Pythonu implementiral že lansko leto pri predmetu Računalništvo. Sem pa mrežo zares napisal čisto iz ničle samo s pomočjo Numpy, ker je bilo drugače množenje in obračanje matrik daleč prepočasno. Dostopen je na linku https://github.com/gregpr07/NeuralNetwork.



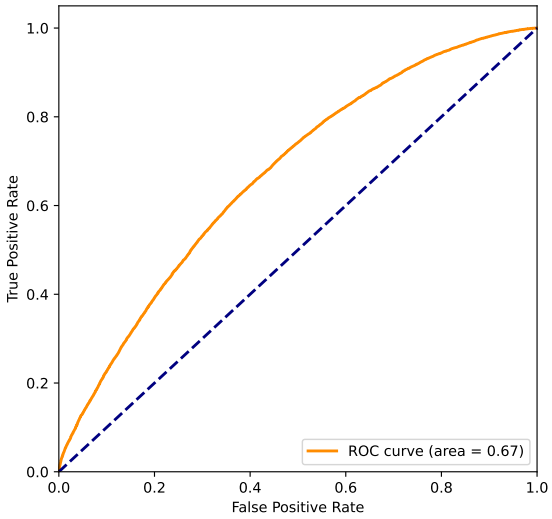
(a) Accuracy z Tensorflow NN.



(b) ROC z Tensorflow NN. $AUC = 0.74$



(c) Accuracy mojega modela



(d) ROC mojega modela. $AUC = 0.67$

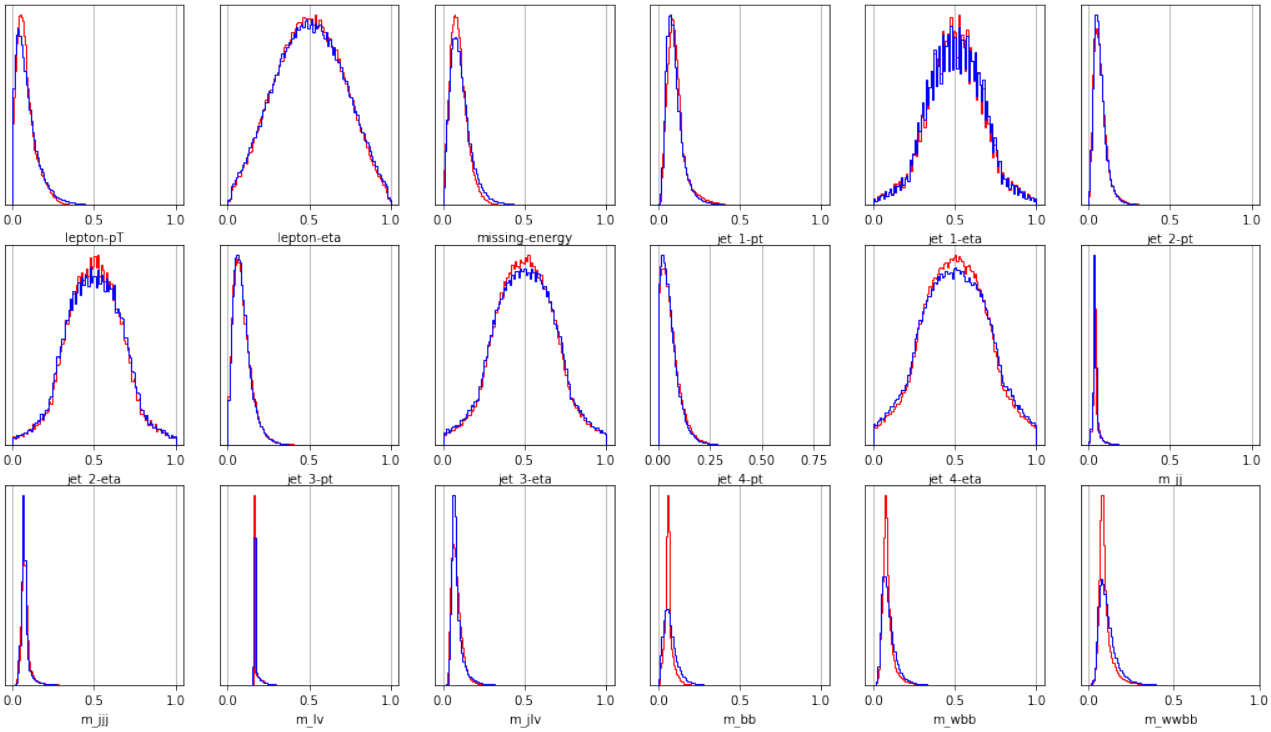
Slika 1: Primerjava performance moje NN in Tensorflowa.

3.1 Opis lastnosti

Vhodni podatki (labels) so čisto navadni vektorji posameznih lastnosti, output pa je zgolj skalar oziroma boolean, če so input vektorji higsov bozon ali ne. Ker so input podatki precej raztroseni, je hiter in zelo učinkovit način procesiranja le teh, da jih pretvorimo v distribucijo - oziroma povemo koliko je posamezna dimenzija vhoda daleč od povprečja. Če te distribucije narišemo, se povsod vidi Gausovka (graf 2).

3.2 Kako pokvariti dimenzijo?

Možnih rešitev in implementacij tega problema je zares veliko. Naprimer, najlažji način bi bil, da celotno dimenzijo postavimo na 0. To je vredu način, vendar se zna zgoditi, da premalo zmede



Slika 2: Vse lastnosti predstavljene kot distribucije.

algoritem, kar bo pomenilo, da podatki ne bodo dobri.

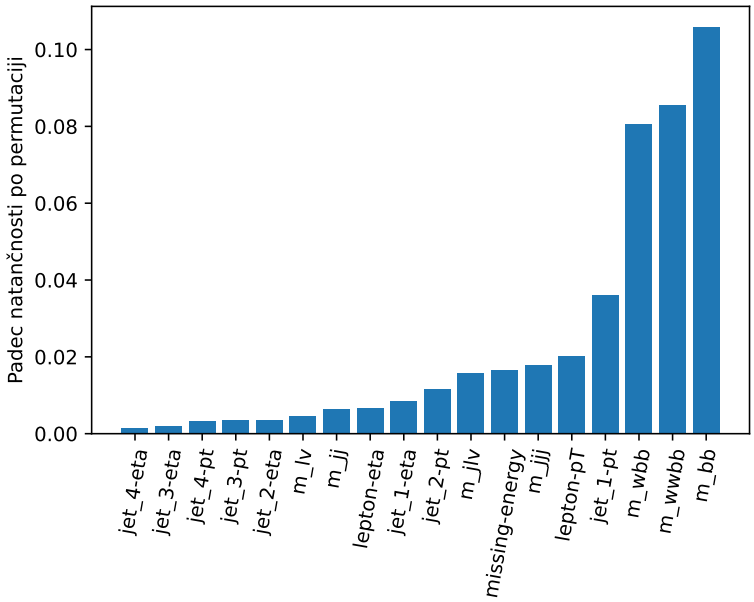
Precej priročna in zelo hitra za implementacijo je permutacijska metoda[1, 2], ki je zelo hitra za izračunati. Zelo na hitro povedano, kar storimo je, da v enem stolpcu "pokvarimo" značilnosti podatkov tako, da jih permutiramo. To pomeni, da bi sedaj morale biti napovedi manj natančna. Koliko manj natančne, pa nam pove sprememba funkcija izgube. Če to sedaj naredimo za vse stolpce lahko dobimo relativno pomembnost stolpcov (značilnosti) podatkov. Tudi implementacija je zelo hitra, saj lahko zgolj uporabim kakšen že napisan program, naprimer `PermutationImportance` iz modula `eli5.sklearn`.

Ko vsak stolpec permutiramo, pogledamo kateri ima največji padec natančnosti - torej kateri stolpec najbolj zmede klasifikacijo. Iz tega lahko narišem dokaj lep sliko 3. Iz njega je lepo vidno, katere lastnosti so najbolj pomembne za detekcijo Higsovih bozonov. Kaj zares pomenijo labele, je dobro napisano na tej spletni strani. Iz moje mreže lahko sklepam, da je najbolj pomembno, da pravilno izmerimo Psevdohitrost eta n-te skupine curkov.

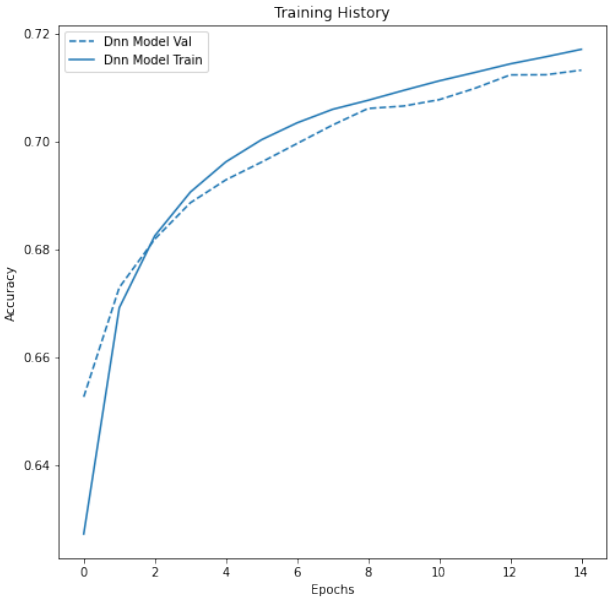
4 Optimizacija DNN

Zanimalo me je tudi, kako daleč zares lahko grem - kaj je največja natančnost, ki jo lahko doseže model. Torej najprej lahko preklopim na ADAM algoritem, za izgubno funkcijo izberem binary_crossentropy in povečam število epochov. Empirično sem ugotovil, da povečanje nevronov v vsakem skriti plasti (če je več kot 50) zares skoraj nima vpliva. Tudi dodajanje skritih plasti nima smisla, ker je velikost podatkov majhna[3]. Torej grem lahko na kavo ter natreniram mrežo z 15 epochi. Rezultati so precej solidni, slika 4.

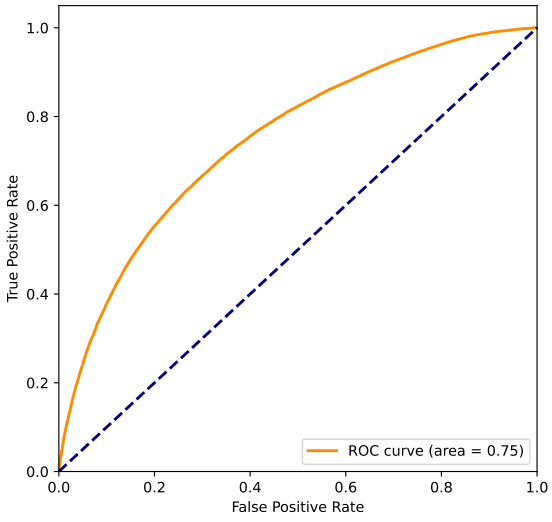
[3]To je zelo relativno. Vendar, če primerjamo število podatkov z drugimi je to res majhno



Slika 3: Pomembnost posameznih label. Vidimo, da so najbolj pomembne za detekcijo bozonov m_{bb}, m_{wwbb} ter m_{wbb}.



(a) Accuracy z dobro natreniranim Tensorflow NN.



(b) ROC z dobro natreniranim Tensorflow NN. $AUC = 0.76$

Slika 4: Performanca najboljše nevronske mreže.

5 Zaključek

Rezultati so dokaj dobri. Je pa tudi res, da se nevronska mreža niti približno nauči tako dobro, kot bi se za kakšen bolj simplističen dataset (naprimer MNIST, kjer so natančnosti zares blizu 100%). Tudi moja mreža je dokaj primerljiva Tensorflow implementaciji, kar pomeni, da sem jo verjetno napisal pravilno.

Literatura

[1] André Altmann, Laura Toloşi, Oliver Sander, and Thomas Lengauer. Permutation importance: a corrected feature importance measure. *Bioinformatics*, 26:1340–1347, 04 2010.

[2] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

Literatura

[1] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow : concepts, tools, and techniques to build intelligent systems.* O'Reilly Media, Inc, Sebastopol, CA, 2019.

[2] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features, 2017.

[3] Tianqi Chen and Carlos Guestrin. Xgboost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 2016.

[4] P. Baldi, P. Sadowski and D. Whiteson, *Searching for Exotic Particles in High-Energy Physics with Deep Learning,* Nature Commun. **5** (2014) 4308 doi:10.1038/ncomms5308 [arXiv:1402.4735 [hep-ph]].