UNIVERZA V LJUBLJANI | FMF Fakulteta za matematiko in fiziko

Matematično-fizikalni praktikum

Dvanajsta naloga: *Strojno učenje*

Simon Bukovšek, 28211067

Škofja Loka, 29. junij 2024

Profesor: prof. dr. Borut Paul Kerševan

Naloga: Strojno učenje

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 tipičnih 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).

1 Uvod

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 (ne-trivialnih funkcijskih odvisnosti itd.).

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 $\mathcal{D} = \{(\mathbf{x}_k, \mathbf{y}_k)\}_{k=1.N}$, 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 $\mathbf{h} : \mathbb{R}^Q \to \mathbb{R}$, 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.

Osnovni gradnik odločitvenih dreves je tako kar stopničasta funkcija $H(x_i - t_i)$, 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 $\mathbf{a}_i$, 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

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

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

kjer je $\mathbf{X}$ nabor vhodnih vrednosti, $\mathbf{w}$ vektor vrednosti uteži, s katerimi tvorimo uteženo vsoto ter b dodatni konstanti 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.

2 Higgs dataset in razlaga spremenljivk

Preden se slepo podamo v uporabo algoritmov strojnega učenja, je smiselno, da se seznanimo s podatki, ki jih bomo uporabili. Podatki so bili simulirani s pomočjo Monte Carlo simulacij trkov protonov pri invariantni masi $\sqrt{s} = 8\,\mathrm{TeV}$. Vsebujejo 28 spremenljivk, ki so podane in razložene v Tabeli 1. Podatki so dostopni v [1], vendar je tam dokumentacija precej pomanjkljiva. Za boljše razumevanje tega poglavja priporočam, da se bralec posvetuje z [2] in kako specializirano literaturo o fiziki delcev, na primer [3].

Veliko uporabljenih izrazov v tej tabeli zahteva dodatno pojasnilo. Za vse nadaljnje izračune uporabljamo $c = \hbar = 1$.

2.1 Invariantna masa

Invariantna masa za dva delca ($\sqrt{s}$) se izračuna kot

$$s = (E_1 + E_2)^2 - \|\mathbf{p}_1 + \mathbf{p_2}\|^2 = m_1 + m_2 - 2(E_1E_2 + \mathbf{p_1} \cdot \mathbf{p}_2).$$

Zakaj ravno črka s in zakaj koren? Kaj ni to nekoliko nepotrebna komplikacija? Oznaka pride iz računanja sipalnih presekov v kvantni teoriji polja. To je ena izmed Mandelstamovih spremenljivk in je enaka kvadratu vektorja četverca vsote gibalnih količin dveh vstopnih delcev. Ker je kvadrat štiri-vektorja Lorentzovo invarianten, je to pri računanju sipalnih presekov dosti bolj ugodna količina. Zatorej moramo pri navajanju energije na trkalniku vedno pisati $\sqrt{s}$.

Pri energijah, kot jih proizvede Veliki hadronski trkalnik (LHC), smemo mase delcev zanemariti. Tako je invariantna masa dveh delcev enaka

$$s = 2p_1p_2(1 - \cos\theta),$$

kjer je θ kot med delcema. Za več delcev je invariantna masa definirana analogno:

$$s = \left(\sum_{i=1}^{n} E_i\right)^2 - \left\|\sum_{i=1}^{n} \mathbf{p}_i\right\|^2.$$

ime spremenljivke	tip	opis spremenljivke
lepton_pT	zvezna	p_T^ℓ leptona
lepton_eta	zvezna	η_ℓ leptona
lepton_phi	uniformna	ϕ_ℓ leptona
missing_energy	zvezna	E^{miss} energija nastalih nevtrinov
missing_energy_phi	uniformna	ϕ^{miss} smeri nastalih nevtrinov
jet1_pt	zvezna	p_T prvega pljuska
jet1_eta	zvezna	η prvega pljuska
jet1_phi	uniformna	ϕ prvega pljuska
jet1_b-tag	diskretna	b-tag prvega pljuska
jet2_pt	zvezna	p_T drugega pljuska
jet2_eta	zvezna	η drugega pljuska
jet2_phi	uniformna	ϕ drugega pljuska
jet2_b-tag	diskretna	b-tag drugega pljuska
jet3_pt	zvezna	p_T tretjega pljuska
jet3_eta	zvezna	η tretjega pljuska
jet3_phi	uniformna	ϕ tretjega pljuska
jet3_b-tag	diskretna	b-tag tretjega pljuska
jet4_pt	zvezna	p_T četrtega pljuska
jet4_eta	zvezna	η četrtega pljuska
jet4_phi	uniformna	ϕ četrtega pljuska
jet4_b-tag	diskretna	b-tag četrtega pljuska
m_jj	zvezna	inv. masa m_{jj} dveh pljuskov
m_jjj	zvezna	inv. masa m_{jjj} treh pljuskov
m_lv	zvezna	inv. masa $m_{\ell\nu}$ leptona in nevtrina
m_jlv	zvezna	inv. masa $m_{j\ell\nu}$ pljuska, leptona in nevtrina
m_bb	zvezna	inv. masa m_{bb} dveh kvarkov dno
m_wbb	zvezna	inv. masa m_{Wbb} bozona $W^\pm$ in dveh b-kvarkov
m_wwbb	zvezna	inv. masa m_{WWbb} dveh bozonov $W^\pm$ in dveh b-kvarkov

Tabela 1: Razlaga spremenljivk v Higgs datasetu.

2.2 Koordinatni sistem

Za razlago kotov si je potrebno ogledati zgradbo eksperimenta ATLAS (glej sliko 1), ki je sestavljen iz več plasti detektorjev, ki merijo različne lastnosti delcev. Sama vsebina detektorja za naše potrebe ni zelo pomembna, pomembno je le, da je cilindrične oblike. Ta valj tvori koordinatni sistem, kjer je os z vzporedna smeri trka (torej osi valja), osi x in y pa sta pravokotni na os z. Kot θ je kot med osjo z in smerjo delca, kot ϕ pa je kot med osjo x in projekcijo smeri delca na ravnino xy. Kot θ nam sicer pove, v katero smer leti delec, vendar pa ni invarianten na Lorentzove *booste* (relativistične spremembe hitrosti). Zato uporabljamo količino η, imenovano *psevdo-rapidnost* (glej sliko 2), ki je definirana kot

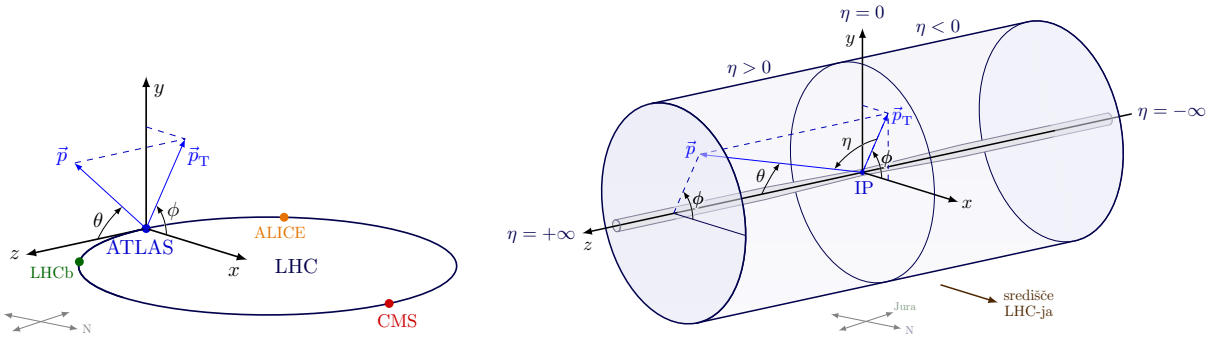
$$\eta = -\ln\left(\tan\left(\frac{\theta}{2}\right)\right).$$

Za brezmasne delce je ta količina kar enaka normalni rapidnosti:

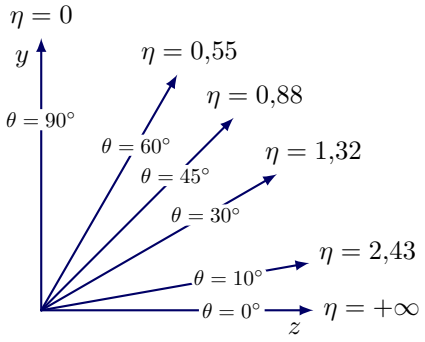
$$y = \frac{1}{2}\ln\left(\frac{E+p_z}{E-p_z}\right) = \operatorname{artanh}\left(\frac{p_z}{E}\right) = \operatorname{artanh}(v_z).$$

Količina η je aditivna pri Lorentzovih *boostih* vzdolž osi trka. Naslednja uporabljena količina je transverzalna gibalna količina p_T, kar je samo projekcija gibalne količine na ravnino pravokotno na os trka. Celotno tri-gibalno količino delca lahko zapišemo kot

$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = p_T \begin{bmatrix} \cos\phi \\ \sin\phi \\ \sinh\eta \end{bmatrix}.$$



Slika 1: Koordinatni sistem eksperimenta ATLAS.



Slika 2: Pretvorba med sferičnim kotom θ in psevdo-rapidnostjo η.

Velikost gibalne količine pa je kar $p = p_T \cosh \eta$. Pri relativističnih trkih je mogoče zanemariti maso delcev, zato je energija delca enaka $E = p$. Tako nam količine η, ϕ in p_T povedo vse o kinematiki delca (ali pljuska). Tako izražene količine nam omogočijo še lažjo izražavo invariantne mase:

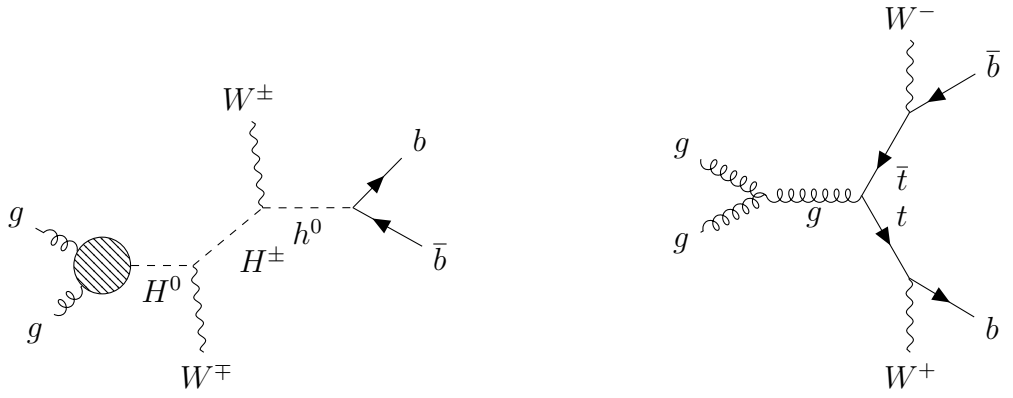
$$s = M^2 = 2p_{1T}p_{2T}(\cosh(\eta_1 - \eta_2) - \cos(\phi_1 - \phi_2)).$$

Senzorji so okoli točke trka postavljeni v čebulasto strukturo za čim natančnejšo detekcijo različnih vrst delcev (oziroma njihovih razpadnih produktov – fotonov, hadronov, elektronov, mionov) in njihovih gibalnih količin. Praktično edina izjema so nevtrini, ki jih je popolnoma nepraktično meriti. Vseeno pa lahko izračunajo, kolikšna je energija in celo povprečna smer nastalih nevtronov. To ugotovijo tako, da izmerijo energije vseh zaznanih delcev in to odštejejo od začetne energije protonov. Razlika se imenuje *manjkajoča masa* (ali energija) E^{miss}; njena prečna komponenta je shranjena pod spremenljivko `missing_energy_magnitude`. Smer manjkajoče energije pa je shranjena pod spremenljivko `missing_energy_phi`:

$$\mathbf{E}_T^{\text{miss}} = \begin{bmatrix} E_T^{\text{miss}} \cos \phi^{\text{miss}} \\ E_T^{\text{miss}} \sin \phi^{\text{miss}} \end{bmatrix}.$$

2.3 Pljuski

Po trku tipično nastanejo leptoni in nevtrini ter kvarki, ki se hadronizirajo v pljuske (jets). Teh je lahko med dva in štiri. Na mestu senzorja lahko posamezen pljusk sestavlja na stotine delcev. Specializirana programska oprema zazna, kateri delci so pripadali skupnemu pljusku, in sešteje njihove energije. Tako lahko dobimo podatek o skupni energiji in smeri začetnega kvarka ali gluona, iz katerega je pljusk nastal. Še več – lahko se izračuna, ali je pljusk nastal iz kvarka dno b. Vsakemu pljusku se zato poleg smeri in velikosti gibalne količine pripiše diskreten (binaren) podatek o vsebnosti b kvarka. Temu označevanju rečemo b-tagging. Za vsakega od štirih pljuskov imamo torej štiri podatke. Prvim 21 spremenljivkam bomo rekli *osnovne spremenljivke*.



(a) Signal. **(b)** Ozadje.

Slika 3: Feynamanova diagrama za signal in ozadje.

2.4 Signal in ozadje

Za razlago zadnjih sedem spremenljivk si moramo malo natančneje ogledati, kaj je sploh namen Higgs dataseta. Osredotočimo se na dva simulirana procesa: signal (slika 3a) in ozadje (slika 3b). Pri signalu gre za nastanek hipotetičnih *težkih* Higgsovih bozonov H^0 in $H^{\pm}$ ter navadnega Higgsovega bozona h^0. Proces ozadja vsebuje iste začetne in končne delce kot signal, le da gre mehanizem po poteh, ki jih Standardni model predvideva.

V datasetu so simulirane meritve dogodkov signala in ozadja. Naša naloga je, da nevronsko mrežo naučimo (oziroma jo prisilimo, da se sama nauči) razločevati med dogodki signala in ozadja na podlagi 21 osnovnih spremenljivk. Na žalost pa se nevronska mreža ne zaveda fizikalnih zakonitosti v ozadju, zato bo (verjetno) samo z osnovnimi spremenljivkami precej težko ločevala. Na srečo smo fiziki opremljeni z nekoliko več vpogleda v fizikalno ozadje, zato lahko modelu dodamo nove spremenljivke, ki odražajo fizikalne procese in ki lahko olajšajo razlikovanje med signalom in ozadjem.

2.5 Izpeljane spremenljivke

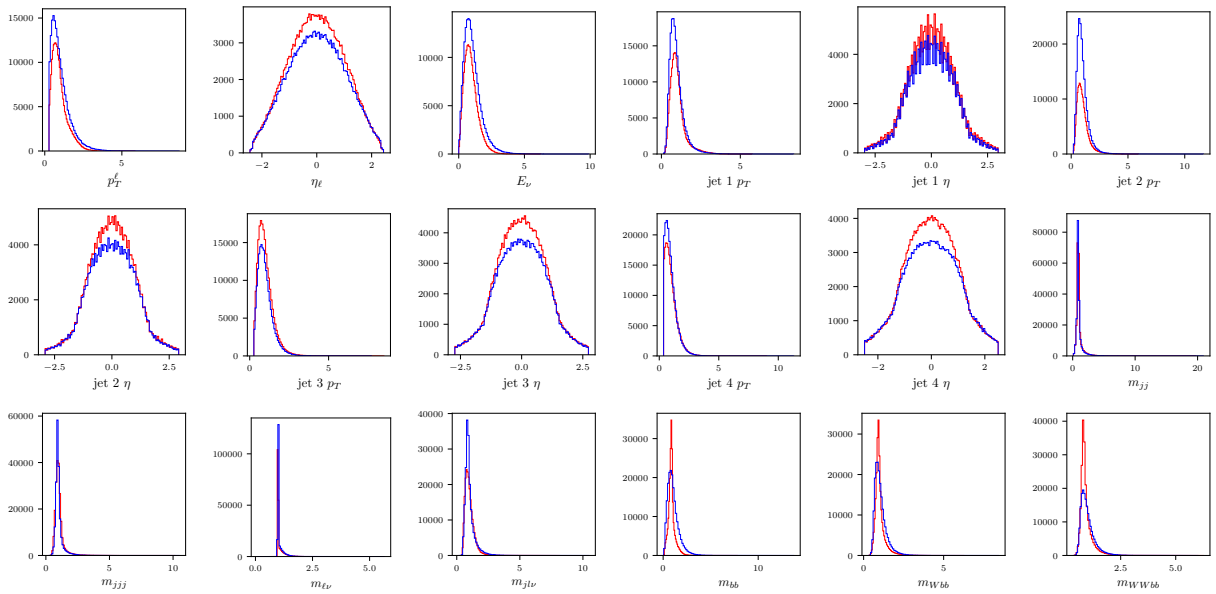
Zadnjih sedem spremenljivk je nekoliko posebnih – povejo namreč nekaj več o samem procesu trka in ne samo o končnih produktih. Najprej upoštevamo, da W bozona ne moremo zaznati kot končen produkt, saj prehitro razpade. Dva glavna procesa razpada W bozona sta na lepton in nevtrino $\ell\nu$ in na dva kvarka, ki tvorita pljuske jj. Zato je smiselno uporabiti formule iz odstavka o invariantni masi in za vsak trk izračunati naslednje invariantne mase. Več o tem v [2].

- Maso dveh pljuskov m_{jj} in invariantno maso nevtrina in najbolj energetičnega leptona $m_{\ell\nu}$. Obe porazdelitvi bi morali dati vrh v bližini mase W bozona m_W (to velja tako za ozadje, kot za signal).
- Maso treh pljuskov m_{jjj} ali pljuska, leptona in nevtrina $m_{j\ell\nu}$. V primeru ozadja dobimo vrh pri masi t kvarka m_t, v primeru signala pa ne.
- Če izračunamo invariantno maso dveh pljuskov z b-tagom (m_{bb}), bi v primeru signala ta porazdelitev dala vrh pri masi navadnega Higgsovega bozona m_h, v primeru ozadja pa ne.
- Invariantno maso m_{Wbb} bozona W in dveh b-kvarkov. V primeru signala bi ta porazdelitev dala vrh pri masi težkega Higgsovega bozona $m_{H^{\pm}}$, v primeru ozadja pa ne.
- Invariantno maso m_{WWbb} dveh bozonov W in dveh b-kvarkov. V primeru signala bi ta porazdelitev dala vrh pri masi težkega Higgsovega bozona m_{H^0}, v primeru ozadja pa ne.

2.6 Podatki

Spremenljivke so razdeljene v tri kategorije: zvezne, uniformne in diskretne. Diskretne spremenljivke so samo b-tagi, ki vsebujejo samo podatek, ali pljusk vsebuje b-kvark ali ne. Uniformne spremenljivke so koti ϕ, saj pri trku ni nobene preferenčne radialne smeri. Vse ostale spremenljivke so zvezne. Diskretnih spremenljivk ne bomo uporabljali, saj so za njihovo uporabo v strojnem učenju potrebni posebni prijemi. Uniformne spremenljivke nam same po sebi nič ne povedo, vendar njihove razlike niso nujno nepomembne. Kljub temu bomo uporabili samo zvezne spremenljivke (med drugim tudi iz razloga, ker se mi ob uporabi celotnega nabora spremenljivk računalnik sesuje).

Neobdelani podatki so prikazani na histogramih na Sliki 4.



Slika 4: Histogrami porazdelitev neobdelanih podatkov. V rdečem so prikazani podatki za signal, v modrem pa za ozadje.

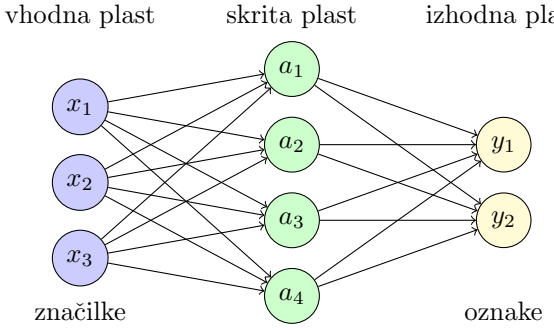
Vidimo, da na oko ne moremo ločiti signalov in ozadij.

3 Globoke nevronske mreže (DNN)

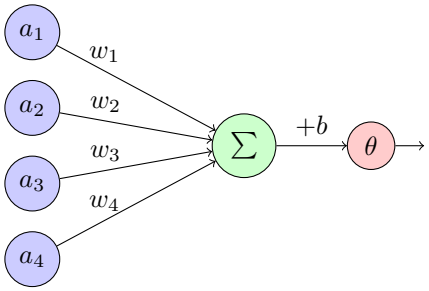
3.1 Zgradba

Za začetek si bomo pogledali splošno delovanje nevronskih mrež in razjasnili nekaj pojmov, ki jih bomo uporabljali v nadaljevanju. Za podrobnejšo razlago priporočam [4]. Kot rečeno že v uvodu, je naša nevronska mreža funkcija $h : \mathbb{R}^n \to \mathbb{R}$. Vhodni podatki se imenujejo *features* (možen slovenski prevod je "značilke") in so podane kot n-dimenzionalen vektor $\mathbf{x}$, izhodni podatki se imenujejo *labels* (oznake) in so predstavljene z vektorjem $\mathbf{y}$. V našem primeru je oznaka samo ena (torej lahko y označujemo kot skalar) in predstavlja, ali je dogodek signal ali ozadje. V Higgs datasetu je približno 11 milijonov simulacij trkov, nekaj več kot polovica jih je po mehanizmu signala, preostanek pa po mehanizmu ozadja. Vsak dogodek sestavlja 28 podatkov (features, našteti v tabeli 1) in oznaka, ki pove, ali je dogodek del signala ali ozadja. Cilj je naučiti nevronsko mrežo, da iz 28 podatkov napove, ali je dogodek signal ali ozadje.

Mrežo sestavljajo sloji nevronov, delovanje pa si lahko predstavljamo s pomočjo aktivacij posameznih nevronov. Vsak nevron v posameznem sloju je povezan z vsemi nevroni v prejšnjem sloju in z vsemi nevroni v naslednjem sloju. Začetni podatki aktivirajo prvi sloj, ta pa preko uteži aktivira naslednji sloj in tako naprej. Aktivacija zadnjega sloja (ki ga v našem primeru sestavlja en sam nevron) je napoved modela (glej sliko 5a).



(a) Primer enostavne nevronske mreže. Mreža vsebuje tri značilke in dve oznaki ter ima en skriti sloj z dimenzijo 4. Vsaka puščica predstavlja utež. Značilke lahko zberemo v vektor $\mathbf{x} = (x_1, x_2, x_3)$, oznake pa v vektor $\mathbf{y} = (y_1, y_2)$.



(b) Prikaz delovanja enega nevrona. Vsaka povezava predstavlja utež w, utežene aktivacije prejšnjega sloja se seštejejo, doda se pristranskost b in se nato prenese skozi aktivacijsko funkcijo θ.

Slika 5: Dva prikaza komponent nevronske mreže.

Naj vektor $\mathbf{a}_i$ predstavlja aktivacije nevronov v nekem sloju, matrika W pa uteži, ki povezujejo ta sloj z naslednjim. Potem lahko aktivacije v naslednjem sloju $\mathbf{a}_{i+1}$ izračunamo kot

$$\mathbf{a}_{i+1} = \theta(\mathsf{W}\mathbf{a}_i + \mathbf{b}).$$

Tukaj je θ aktivacijska funkcija, ki poskrbi za nelinearnost mreže, $\mathbf{b}$ pa nek vektor pristranskosti (bias, glej sliko 5b). Tri najbolj tipične aktivacijske funkcije so:

- sigmoid:
$$\theta(x) = \frac{1}{1 + e^{-x}};$$
- ReLU (rectified linear unit):
$$\theta(x) = \max(0, x);$$
- ELU (exponential linear unit):
$$\theta(x) = \max(x, e^x - 1).$$

Podatki, ki jih izmerijo na detektorju imajo lahko različne enote in se raztezajo čez več velikostnih redov. Zato je smiselno podatke normalizirati, da so vse spremenljivke približno enakega reda velikosti. Spet obstaja več načinov, kako to doseči, naštel bom dva:

- razteg in premik podatkov tako, da je povprečje 0 in standardni odklon 1;
- razteg in premik podatkov, tako da so na intervalu $[0, 1]$.

3.2 Učenje

Na začetku so vse uteži in vse pristranskosti postavljene naključno, zato nevronska mreža še ne zna ničesar napovedati. Učenje poteka tako, da mreži dajemo vhodne podatke $\mathbf{x}$, gledamo njene napovedi $\hat{y}$ in jih primerjamo z dejanskimi oznakami y. Želimo spremeniti uteži in pristranskosti tako, da bo napaka čim manjša. Definiramo *funkcijo izgube* L, ki je odvisna od vrednosti uteži in nam pove, kako daleč je napoved modela od oznake. Napoved je vrednost med 0 in 1, oznaka pa je bodisi 0 (ozadje), bodisi 1 (signal). Iščemo optimalne vrednosti uteži in pristranskosti, ki minimizirajo funkcijo izgube. Zaradi računske stabilnosti in optimizacije pošljemo modelu več dogodkov zapovrstjo in šele potem popravimo uteži. Dogodke sestavimo v *batche* (skupine) velikosti m, po vsakem batchu pogledamo, kako se naše napovedi razlikuje od oznak in na podlagi tega s pomočjo *optimizacijskega algoritma* popravimo vrednosti uteži. Najenostavnejši optimizacijski algoritem je kar spust po gradientu – preprosto izračunamo gradiente funkcije izgube in se z majhnim korakom premaknemo v nasprotni smeri. To ponavljamo, dokler ne dosežemo približnega

minimuma. Ker uteži popravljamo šele po vsakem batchu, rečemo temu *stochastic gradient descent* (SGD). Obstajajo naprednejši optimizacijski algoritmi, ki upoštevajo tudi prejšnje korake in tako pospešijo učenje. Najbolj uporabljeni so *adam* in njegove izpeljave (več v [5]). Funkcij izgube je prav tako cela plejada, mi bomo uporabljali *binary cross-entropy* (BCE) in *mean squared error* (MSE, več je razloženo v uvodu).

Podatke, ki so nam na voljo tipično razdelimo na množico za učenje in množico za testiranje. Ko želimo oceniti uspešnost modela, moramo to namreč preveriti na podatkih, ki jih model še ni videl. Hkrati pa to pomeni, da lahko podatke, ki jih imamo na voljo za učenje uporabimo večkrat. Ko modelu podamo vse podatke, ki smo jih namenili za učenje, temu rečemo *epoha*. Podatke potem lahko zmešamo in jih v drugačnih kombinacijah batchov spet podamo modelu za učenje. Izkaže se, da s tem lahko dosežemo boljše učenje z isto količino podatkov. Čisto na koncu učenja skozi model pošljemo še nabor *validacijskih podatkov*, ki jih model ni videl še nikoli prej, niti med sprotnim testiranjem. Ti rezultati nam podajo dejansko uspešnost modela. Obenem pa je treba biti pazljiv, da se model ne specializira preveč na učne podatke – temu rečemo *overtraining* in ga lahko zaznamo, če se napaka na testnih podatkih začne povečevati, medtem ko se napaka na učnih podatkih še zmanjšuje.

3.3 Uporaba

Sedaj smo razložili vse relevantne pojme v zvezi z delovanjem nevronskih mrež, zato lahko testno poženemo model na danih podatkih. Začetne nastavitve (tudi imenovani hiperparametri) so:

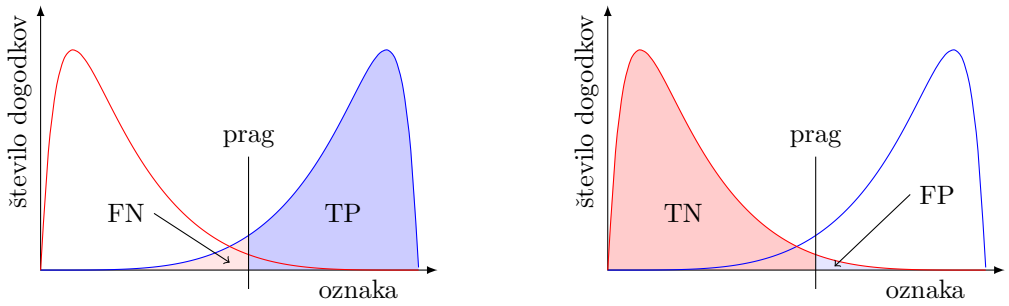
- normalizacija: na interval $[0, 1]$;
- velikost *batchov*: 1000;
- odstotek podatkov, ki se uporabi za testiranje po vsaki epohi: 10 %;
- nevronska mreža z dvema skritima plastema s po 50 nevroni;
- aktivacijska funkcija: ReLU;
- aktivacijska funkcija na koncu: *sigmoid*;
- optimizacijski algoritem: *adam*;
- funkcija izgube: *binary crossentropy*;
- število epoh: 200.

Skozi naučen model smo poslali še 100 000 preostalih, validacijskih podatkov – 52 876 dogodkov signala in 47 124 dogodkov ozadja. Za vsak trk (to je za vsak nabor 18 featurov) nam je model podal številko med 0 in 1 (to je zaloga vrednosti sigmoide, ki jo uporabimo na koncu nevronske mreže). Trke smo ločili po oznaki in jih nanesli na histogram (glej sliko 7a). Razvidno je, da se je model približno naučil razlikovati med signalom in ozadjem – večina dogodkov ozadja ima nižjo predvideno oznako kot dogodki signala. Vse kar bi zdaj še morali storiti, je določiti prag, pri katerem bomo rekli, da je dogodek signal. Ko si izberemo neko vrednost praga, dobimo štiri vrste primerov (glej 6):

- *pravilno pozitivni* (TP): model je pravilno napovedal prisotnost bozona,
- *lažno pozitivni* (FP): model je napovedal prisotnost bozona, v resnici pa ga ni bilo,
- *pravilno negativni* (TN): model je pravilno napovedal odsotnost bozona,
- *lažno negativni* (FN): model je napovedal odsotnost bozona, v resnici pa je bil.

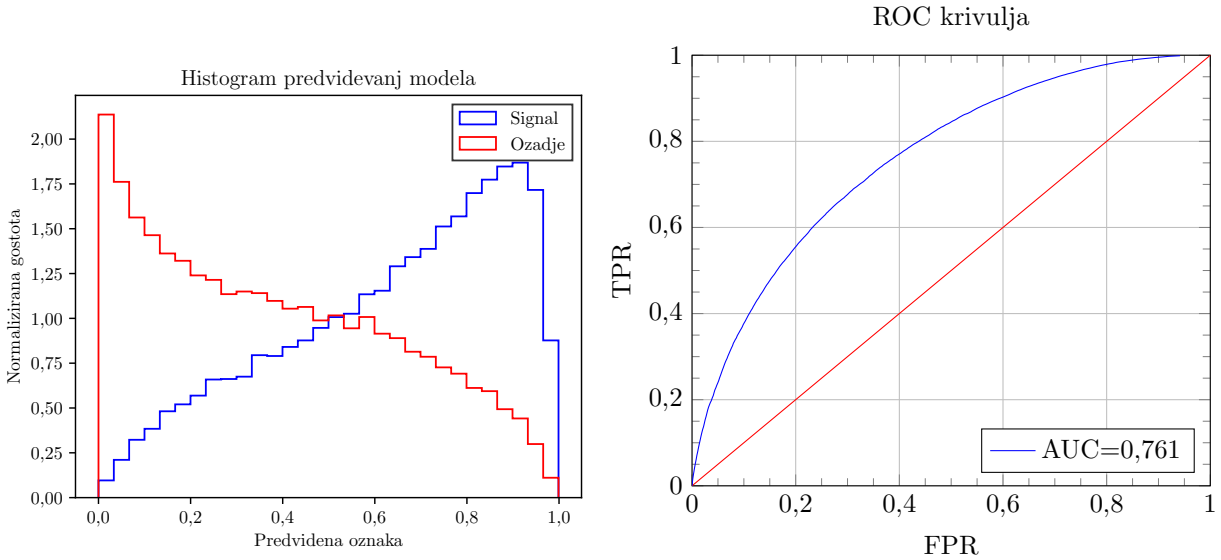
Pri določanju uspešnosti modela nas predvsem zanima, kako so te štiri količine odvisne od izbire meje. V ta namen vpeljemo dve novi količini, imenovani *delež pravilno pozitivnih primerov* (angl. true positive rate, TPR) in *delež lažno pozitivnih primerov* (angl. false positive rate, FPR). Definiramo ju kot

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}\,, \quad \mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}\,. \tag{1}$$



Slika 6: Prikaz pravilno pozitivnih in negativnih ter lažno pozitivnih in negativnih primerov. Rdeča krivulja predstavlja porazdelitev negativnih primerov po oznaki, modra pa pozitivnih. Pokončna črna črta označuje izbran prag.

Sedaj lahko narišemo odvisnost TPR od FPR pri vseh možnih vrednostih praga. Tako dobimo *ROC* (receiver operating characteristic) krivuljo. Boljši modeli imajo krivuljo, ki se čim bolj približa zgornjemu levemu kotu grafa (glej sliko 7b). Če želimo uspešnost modela povzeti z eno številko, tipično uporabimo vrednost površine pod ROC krivuljo. Tej vrednosti pravimo AUC (area under curve) – bližje kot je številki 1, boljši je model, bližje kot je številki 0,5, slabši je model. V našem primeru je AUC enak 0,76.



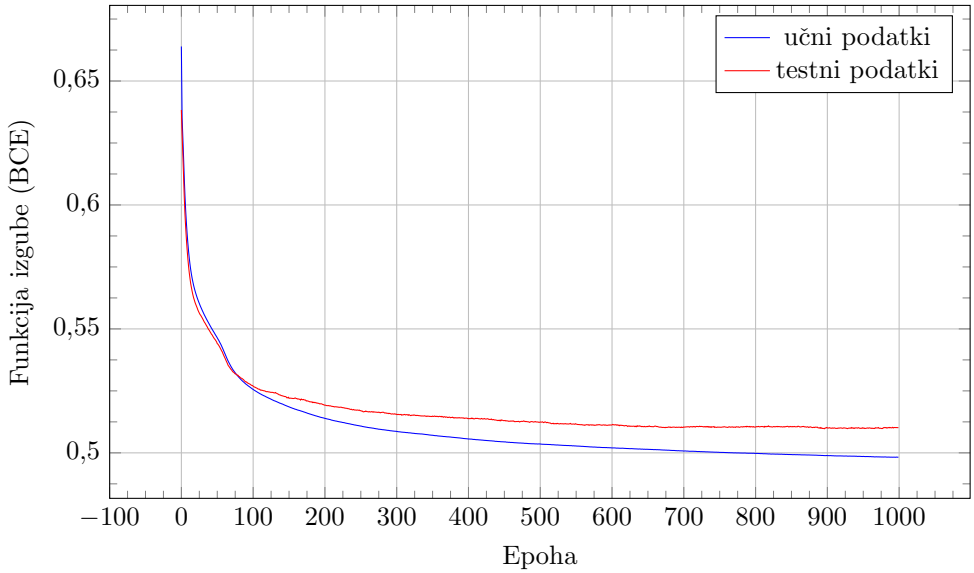
(a) Histogram napovedi za signal in ozadje.

(b) ROC krivulja.

Slika 7: Rezultati osnovnega modela.

3.4 Pretreniranje

Kot že omenjeno v prejšnjih podpoglavjih, se model lahko preveč specializira na učne podatke – se pretrenira (overtraining). To se zgodi pri prevelikem številu epoh. Na Sliki 8 je prikazan primer vrednosti funkcije izgube (BCE, binary cross-entropy) za testne in učne podatke po vsaki epohi. Vidimo, da se sprva obe vrednosti močno zmanjšujeta, kar pomeni, da se model uči. Čez čas pa se vrednost funkcije izgube za testne podatke neha zmanjševati, torej nadaljnje treniranje ni več smiselno. Če bi pustil učenje do 10 000 epoh, bi videli, da se začne testna funkcija izgube celo večati. Da bi to preprečili uporabimo tako imenovan *early stopping*, kar pomeni, da prenehamo z učenjem, ko se funkcija izgube za testne podatke neha zmanjševati. V nadaljevanju bom uporabil pogoj, da program neha z učenjem, če se funkcija izgube za testne podatke ne zmanjšuje več 5 epoh.



Slika 8: Funkcija izgube za testne in učne podatke po vsaki epohi.

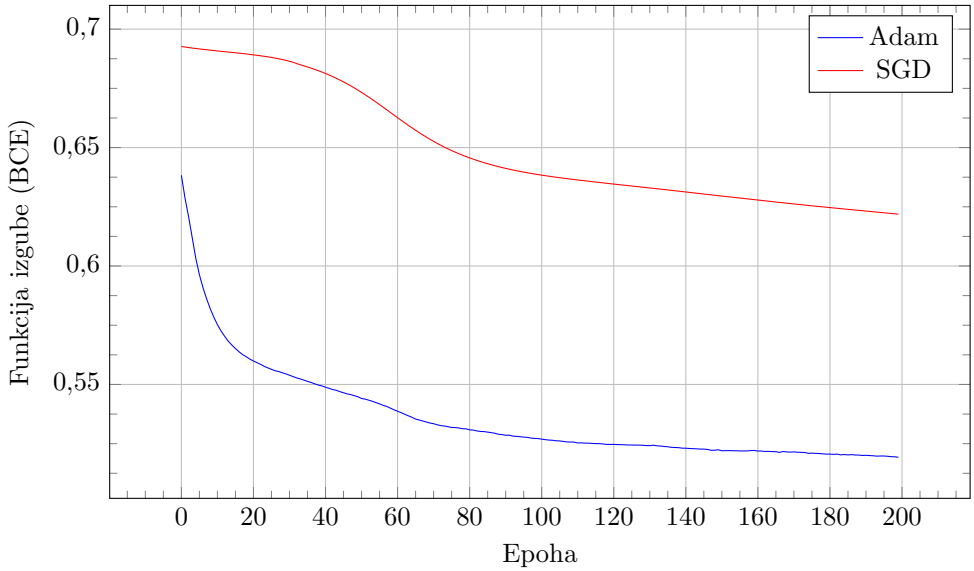
3.5 Hiperparametri

V tem delu bomo spreminjali hiperparametre, enega po enega, in opazovali, če se AUC kaj bistveno spremeni. Začetni hiperparametri so navedeni v podpoglavju 3.3, brez early stoppinga. Rezultati so zbrani v Tabeli 2. Najprej lahko opazimo, da pri večini sprememb ne pride do znatnih odstopanj vrednosti AUC.

sprememba hiperparametra	AUC
začetni hiperparametri	0,7610
velikost batcha: 100	0,7593
velikost batcha: 10 000	0,7453
število epoh: 100	0,7507
število epoh: 300	0,7610
dele podatkov za testiranje: 20 %	0,7599
aktivacijska funkcija: sigmoid	0,7125
aktivacijska funkcija: ELU	0,7510
optimizator: SGD	0,7025
optimizator: Adamax	0,7481
skaliranje na $\mu = 0$ in $\sigma = 1$	0,8235
funkcija izgube: MSE	0,7578
skrita sloja velikosti 25	0,7501
skrita sloja velikosti 100	0,7681
brez skritih slojev	0,6819
štirje skriti sloji	0,7516

Tabela 2: Vpliv spremembe hiperparametrov na AUC.

Komentiral pa bi tiste spremembe, ki so vplivale na AUC več kot za 0,02. Če smo aktivacijsko funkcijo skritih slojev zamenjali iz ReLU na sigmoido se je AUC zmanjšal za 0,04. To je posledica počasnejšega učenja in se funkcija izgube v 200 epohah še ni ustalila. Pri več epohah bi najbrž dosegli podoben AUC kot v referenčnem primeru. Ob spremembi velikosti batcha se v obe smeri AUC zmanjša, vendar je tudi hitrost učenja ob enakem številu epoh obratno sorazmerna z velikostjo batcha – pri velikosti 10 000 smo torej dosegli podoben AUC v desetini časa. Bistveno poslabšanje uspešnosti modela opazimo pri menjavi optimizatorja iz Adama na najosnovnejšo varianto: spust po gradientu (SGD). Razlog je spet ta, da je učenje preprosto počasnejše in v 200 epohah funkcija izgube še ne doseže minimuma. Za demonstracijo



Slika 9: Funkcija izgube za testne podatke po vsaki epohi za optimizatorja Adam in SGD.

sem narisal kako se funkcija izgube spreminja z epohami za oba optimizatorja (glej sliko 9). Ni čudno, da je Adam deležen toliko pozornosti.

Rahlo izboljšavo dobimo, če velikost obeh skritih slojev povečamo na 100. To je posledica večjega števila prostostnih stopenj, ki jih ima model na voljo. Če sloje zmanjšamo na 25, se AUC zmanjša za približno enako vrednost. Če odstranimo skrite sloje, ostane modelu le še 19 prostostnih stopenj, zato AUC pričakovano pade (na histogramu zgleda še slabše). Pri dodanih dveh skritih slojih pride po 200 epohah že skoraj do pretreniranja. V tako konfiguraciji zelo dobro zaznava dogodke ozadja, medtem ko konfiguracija z MSE zelo dobro zaznava dogodke signala. Presenetljivo se izkaže, da je krepko najboljša izboljšava sprememba normalizacije vhodnih podatkov na $\mu = 0$ in $\sigma = 1$. Korak, ki je tipično deležen najmanj pozornosti, se izkaže za najbolj pomembnega.

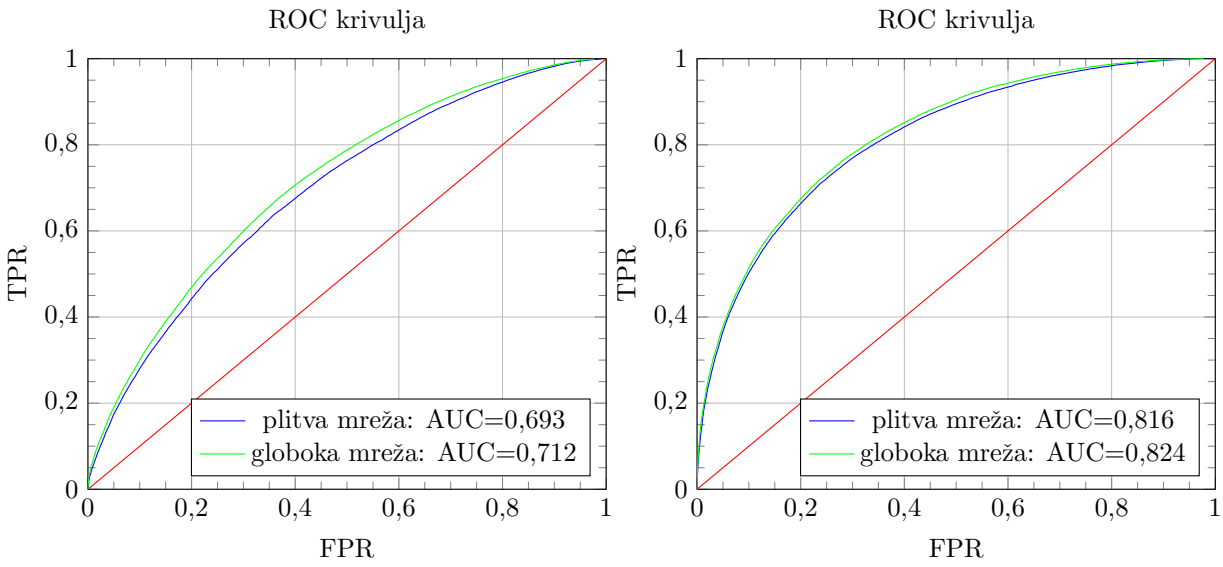
3.6 Osnovne in izpeljane spremenljivke v plitvih in globokih mrežah

V tem delu se bomo vprašali isto vprašanje kot v [2] in poskusili približno ponoviti njihove rezultate. Primerjali bomo uspešnost plitvih in globokih mrež v odvisnosti od izbire vhodnih spremenljivk. Plitva mreža bo imela en skriti sloj velikosti 100, globoka pa pet takih slojev. Uporabili bomo dva nabora spremenljivk: samo osnovne zvezne in vse zvezne (glej poglavje 2). Osnovnih zveznih spremenljivk je 11, vseh zveznih pa 18. V [2] so sicer uporabili tudi diskretne in binarne spremenljivke, vendar se bomo mi omejili le na zvezne. Ostali hiperparametri bodo nastavljeni na vrednosti iz podpoglavja 3.3, edina sprememba bo normalizacija na $\mu = 0$ in $\sigma = 1$, saj je to najbolj pozitivno vplivalo na AUC. Za vsak slučaj bomo dodali še early stopping – če se funkcija izgube v petih epohah ne izboljša, program avtomatsko zaključi z učenjem. Rezultati so prikazani v Tabeli 3 in na slikah 10.

	Površina pod ROC krivuljo (AUC)	
Mreža	Osnovne	Vse vhodne spremenljivke
Plitva	0,6933	0,8155
Globoka	0,7117	0,8236

Tabela 3: Vrednosti AUC pri uporabi različnih vrst vhodnih spremenljivk za plitve (en skriti sloj) in globoke (pet skritih slojev) nevronske mreže.

Globoki mreži sta se izredno hitro ustavili, že po 15 epohah. Plitvi mreži sta se učili dlje časa in se ustavili po okoli 100 epohah. Zaključek v članku [2] je, da so globoke mreže boljše od plitvih, in



(a) ROC krivulja za osnovne spremenljivke.

(b) ROC krivulja za vse spremenljivke.

Slika 10: ROC krivulje za plitve in globoke mreže za osnovne in vse spremenljivke.

da je uspešnost globokih mrež neodvisna od uporabljenih izpeljanih spremenljiv. Pri meni je dodatek izpeljanih spremenljivk bistveno povečal uspešnost obeh vrst mrež (plitve in globoke), vseeno pa se kaže, da pri osnovnih spremenljivkah globoka nevronska mreža povzroči boljši napredek uspešnosti kot pri vseh spremenljivkah.

4 Pospešena odločitvena drevesa (BDT)

O tej temi sem nekoliko manj podučen, zato ne bo tako dolgega uvoda. Nekaj pregledne razlage se lahko najde v [6]. Osnovna ideja odločitvenih dreves je sestaviti zaporedje vprašanj, ki nas usmerijo proti predvideni vrednosti oznake. Podobno kot pri vprašalnikih za izbiro poklica, le da je vsako nadaljnje vprašanje odvisno od prejšnjega odgovora. Pri vsakem vprašanju primerjamo velikost ene izmed vhodnih spremenljivk glede na nek prag in se glede na to odločimo, ali gremo v levo ali desno vejo. Veja se lahko zaključi z listom (leaf), ki nam poda predvideno oznako ($\hat{y}$) ali pa vsebuje naslednje vprašanje (razvejitev). Pri drevesu lahko omejimo število listov ali število nivojev. Prednost takega pristopa je, da lahko končno strukturo vizualiziramo in razumemo, kako model deluje (bela škatla napram nevronski mreži, ki je črna škatla). Glede na to da podatke tipično primerjamo med sabo samo pri isti spremenljivki, je tak pristop tudi neodvisen od začetne normalizacije podatkov, ki je bila pri nevronskih mrežah še kako pomembna. Slabost odločitvenih dreves je, da se zelo hitro pretrenirajo.

Proces učenja dreves je pravzaprav določanje "vprašanj" na razvejiščih, oziroma določanje pragov za vsako spremenljivko. Optimalno izbiro pragu določa *funkcija nečistosti* (impurity function), ki nam pove, kako dobro smo z nekim pragom razdelili podatke. Več o tem v poglavju 3.3 v [6]. Pogoste izbire za funkcijo nečistosti so cross-entropy, Ginijev indeks in misclassification error.

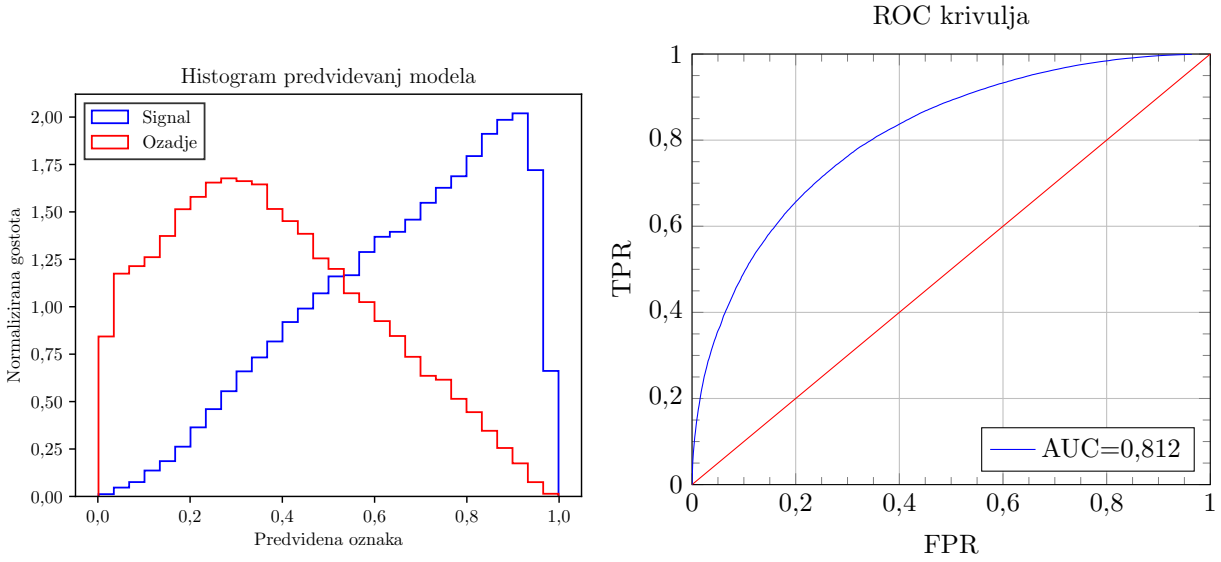
Odločitvena drevesa so prisotna že od osemdesetih let prejšnjega stoletja. V devetdesetih pa so se že pojavile izpeljane metode, ki so združevale več odločitvenih dreves v en model. Eden izmed teh je *boosted decision trees* (BDT). Ideja je, da se vzame uteženo povprečje več odločitvenih dreves, ki se učijo iz napak drug drugega. Same uteži določa optimizacijski algoritem, dandanes sta med priljubljenejšimi AdaBoost in Gradient boosting. Mi bomo uporabili CatBoost, ki uporablja gradient boosting.

4.1 Uporaba

Za začetek sem pognal BDT na osnovnih hiperparametrih, ki so:

- brez normalizacije;
- maksimalno število dreves: 100;
- funkcija nečistosti: LogLoss;
- maksimalna globina dreves: 6.

Rezultati so prikazani na sliki 11.

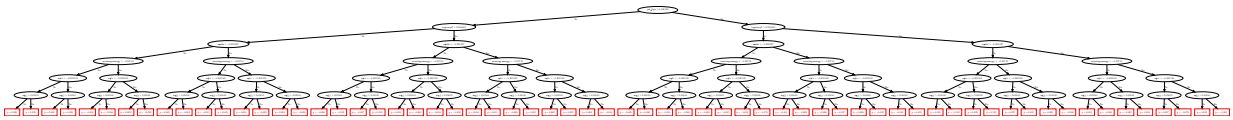


(a) Histogram napovedi za signal in ozadje.

(b) ROC krivulja.

Slika 11: Rezultati osnovnega modela BDT.

Kot omenjeno, lahko odločitvena drevesa vizualiziramo. Na sliki 12 je prikazano eno izmed dreves (najuspešnejše), ki ga je model sestavil. Razvidno je, da je model uporabil naslednjih 6 spremenljivk: `jet_2-pt`, `lepton-pT`, `m_wbb`, `missing-energy`, `m_lv` in `m_jj`. Večina izmed njih je izpeljanih.



Slika 12: Vizualizacija enega izmed dreves, ki ga je sestavil model. Priporočam povečavo.

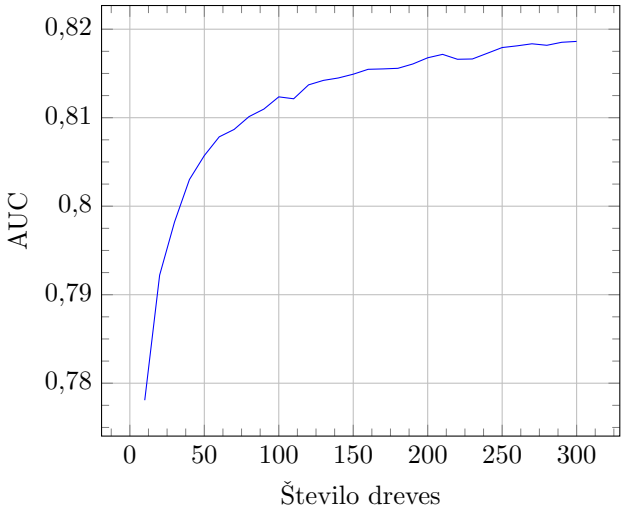
4.2 Hiperparametri

Enako kot z nevronskimi mrežami sem spreminjal nekaj hiperparametrov in opazoval, kako se spremeni AUC. Rezultati so zbrani v Tabeli 4.

Sprememba normalizacije očitno ne vpliva na uspešnost, kot je pričakovano. Večanje števila dreves vpliva pozitivno na uspešnost, prav tako maksimalna globina dreves (čeprav je sprememba minimalna). Menjava funkcije nečistosti na cross-entropy je zmanjšala uspešnost. Primerjava med številom dreves in AUC je prikazana na grafu 13. Zgleda, kot da med številom dreves in AUC velja približno logaritemska zveza.

sprememba hiperparametra	AUC
začetni hiperparametri	0,8123
normalizacija na $\mu = 0$ in $\sigma = 1$	0,8123
maksimalno število dreves: 30	0,7982
maksimalno število dreves: 300	0,8186
funkcija nečistosti: Cross-entropy	0,7738
maksimalna globina: 3	0,7960
maksimalna globina: 12	0,8158

Tabela 4: Vpliv spremembe hiperparametrov na AUC.



Slika 13: Vpliv števila dreves na AUC.

4.3 Osnovne in izpeljane spremenljivke

Sedaj bomo tako kot v prejšnje poglavju primerjali uspešnost modelov, če uporabimo samo osnovne spremenljivke. Namesto globokih in plitvih mrež bomo opazovali nizka in visoka drevesa (maksimalna globina 3 in 12). Vse ostale nastavitve so enake kot v prejšnjem podpoglavju. Rezultati so zbrani v Tabeli 5 in na sliki 14.

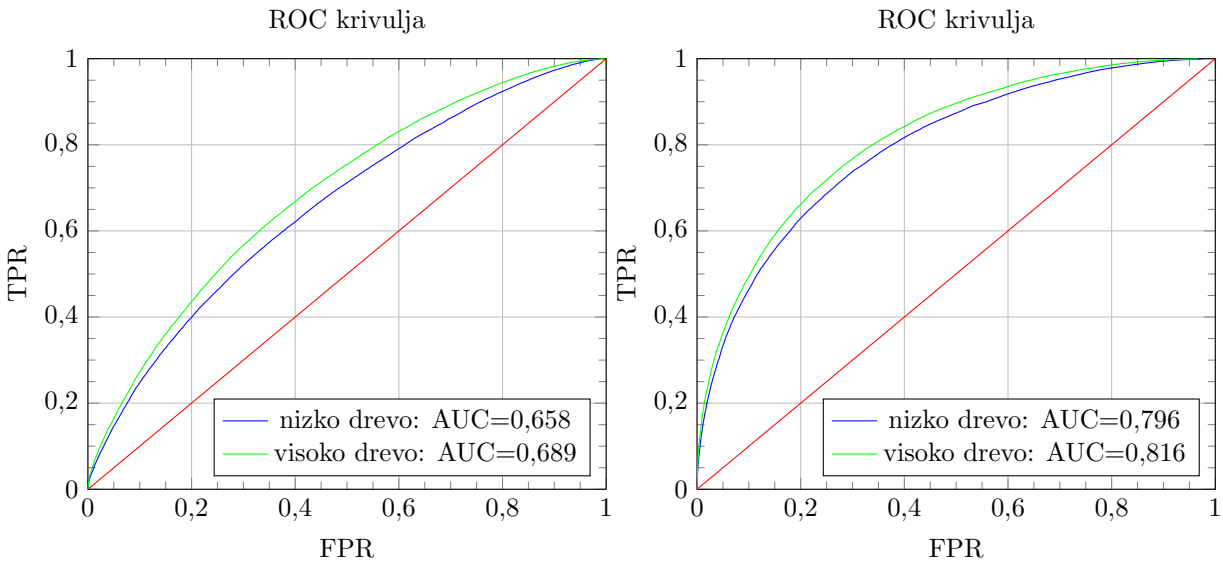
	Površina pod ROC krivuljo (AUC)	
Drevo	Osnovne	Vse vhodne spremenjivke
Nizko	0,6584	0,7960
Visoko	0,6888	0,8158

Tabela 5: Vrednosti AUC pri uporabi različnih vrst vhodnih spremenljivk za nizka (globina 3) in visoka (globina 12) odločitvena drevesa.

Spet lahko sklenemo, da dodatek izpeljanih spremenljivk poveča uspešnost modela ne glede na višino drevesa. Pri uporabi samo osnovnih spremenljivk je izboljšava pri povečanju višine drevesa večja kot pri uporabi vseh spremenljivk.

5 Dodatna naloga

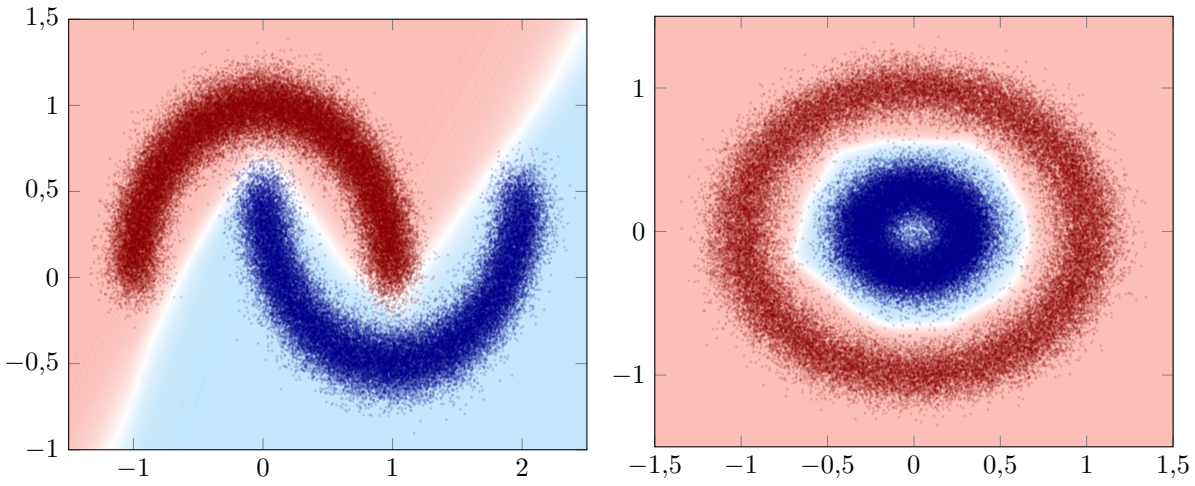
Za zaključek smo implementirali DNN in BDT še za dva *playground* dataseta iz SciKit-Learn. Prvi je `make_moons`, ki generira dva polmeseca, drugi pa `make_circles`, ki generira dva kroga. Iz obeh datasetov



(a) ROC krivulja za osnovne spremenljivke. **(b)** ROC krivulja za vse spremenljivke.

Slika 14: ROC krivulje za visoka in nizka drevesa za osnovne in vse spremenljivke.

smo generirali 100 000 podatkov. Nevronsko mrežo smo sestavili iz dveh skritih slojev velikosti 10 in jih učili 15 epoh. V obeh primerih je ti zadoščalo, da smo dosegli AUC = 1. Rezultati so prikazani na slikah 15.

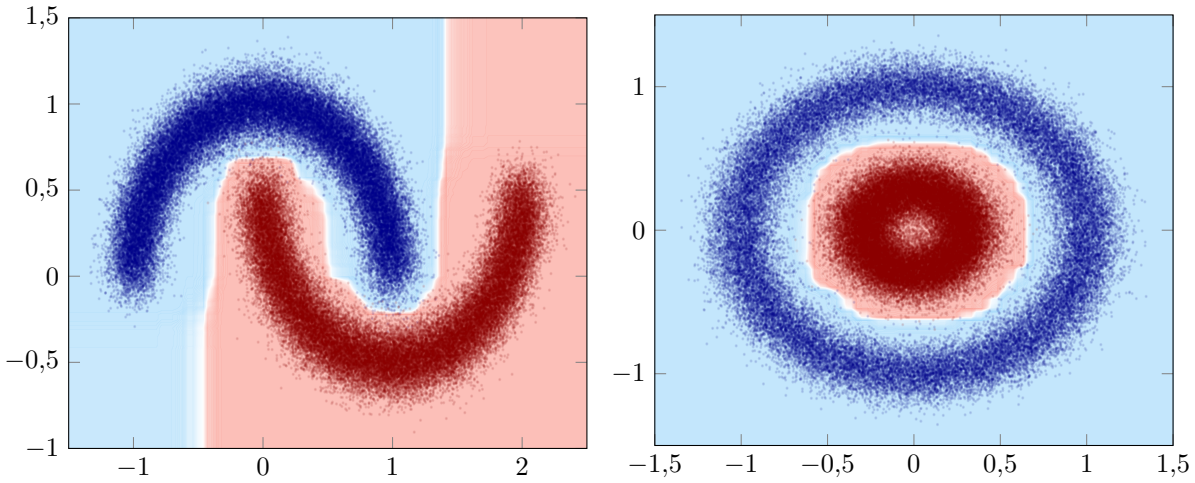


(a) Lunice. **(b)** Krogi.

Slika 15: Prikaza, kako nevronska mreža razloča podatke, na podlagi učenja za `make_moons` in `make_circles`.

Iste vzorce smo testirali tudi na odločitvenih drevesih. Uporabili smo iste hiperparametre kot v prejšnjem podpoglavju. Rezultati so prikazani na sliki 16. Takšna konfiguracija je omogočala zelo dobro razločevanje podatkov, saj smo dosegli AUC = 1 v obeh primerih. Pri podatkih bi sicer lahko povečal razpršenost, tako da bi se nekateri prekrivali in bi bilo tudi v principu nemogoče doseči popolno razločevanje, vendar sem se odločil, da pustim parameter `noise` na vrednosti 0,1, saj so vizualizacije preglednejše.

Opazimo lahko nekaj razlik v načinu, kako BDT in DNN vlečeta mejo med skupinami podatkov. DNN poišče ekstremne točke posameznih skupin in med njimi povleče tako rekoč ravne črte, hkrati pa nima težav niti z ugotavljanjem krivih linij. Odločitvena drevesa tvorijo meje iz kratkih ravnih odsekov, ki



(a) Lunice. **(b)** Krogi.

Slika 16: Prikaza, kako odločitvena drevesa razločajo podatke, na podlagi učenja za `make_moons` in `make_circles`.

kažejo večinoma v glavnih in stranskih smereh neba.

6 Zaključek

V tej nalogi smo se spoznali z nevronskimi mrežami in odločitvenimi drevesi. V zvezi z nevronskimi mrežami ni nisem spoznal nič posebej novega, imel sem le lepo priložnost, da sem preizkusil vpliv vseh hiperparametrov na rezultat. Najbolj me je presenetila pomembnost normalizacije na uspešnost. Odločitvena drevesa so bila zame popolna novost, zato sem se moral najprej podučiti o njihovi osnovni strukturi in delovanju. Presenetljivo je, da kljub relativno enostavnejši strukturi odločitvena drevesa dosežejo podobno uspešnost kot nevronske mreže.

Čisto za konec bi komentiral še predmet Matematično-fizikalni praktikum kot celoto. Glavni cilj je jasno spoznavanje različnih računalniških metod za reševanje raznovrstnih fizikalnih problemov – vrlina, ki jo dandanes potrebuje skoraj vsak fizik. Primarna metoda učenja pa je – *učenje skozi lastne napake*. Pri tem predmetu je skoraj nemogoče, da bi se študent izognil lastnim napakam, in ravno reševanje iz jame, ki si jo študent sam nakoplje, je tisto, od česar se največ nauči. Obenem od študenta zahteva določeno mero discipliniranosti ali pa neprespanih noči, saj je tempo precej intenziven.

Še pregled statističnih podatkov vseh dvanajstih poročil:

- število grafov: 202, to je 17 na poročilo;
- število besed: 25 629, to je 2136 na poročilo;
- število vrstic kode: 4342 (brez risanja grafov), to je 362 na poročilo;
- število animacij: 6;
- število ur vloženega dela: več, kot sem pripravljen priznati.

Literatura

[1] D. Whiteson. *HIGGS*. UCI Machine Learning Repository. 2014. DOI: `https://doi.org/10.24432/C5V312`.

[2] P. Baldi, P. Sadowski in D. Whiteson. »Searching for exotic particles in high-energy physics with deep learning«. V: *Nature Communications* 5.1 (jul. 2014). ISSN: 2041-1723. DOI: 10.1038/ncomms5308. URL: http://dx.doi.org/10.1038/ncomms5308.

[3] D. J. Griffiths. *Introduction to elementary particles; 2nd rev. version*. Physics textbook. New York, NY: Wiley, 2008. ISBN: 9783527406012. URL: https://cds.cern.ch/record/111880.

[4] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. Ur. R. Roumeliotis in N. Tache. 2. izd. O'Reilly Media, Inc., 2019. ISBN: 9781492032649.

[5] D. P. Kingma in J. Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[6] Y. Coadou. »Boosted Decision Trees«. V: *Artificial Intelligence for High Energy Physics*. WORLD SCIENTIFIC, feb. 2022, str. 9–58. ISBN: 9789811234033. DOI: 10.1142/9789811234033_0002. URL: http://dx.doi.org/10.1142/9789811234033_0002.