Opis raspodjele kvarkova i gluona u protonu pomoću gausijanskih procesa

Radočaj, Andrija

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:474705

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-24**

Repository / Repozitorij:

Repository of the Faculty of Science - University of Zagreb

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

Andrija Radočaj

OPIS RASPODJELE KVARKOVA I GLUONA U PROTONU POMOĆU GAUSIJANSKIH PROCESA

Diplomski rad

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

INTEGRIRANI PREDDIPLOMSKI I DIPLOMSKI SVEUČILIŠNI STUDIJ
FIZIKA; SMJER ISTRAŽIVAČKI

Andrija Radočaj

Diplomski rad

Opis raspodjele kvarkova i gluona u protonu pomoću gausijanskih procesa

Voditelj diplomskog rada: prof. dr. sc. Krešimir Kumerički

Ocjena diplomskog rada: _____

Povjerenstvo: 1. _____

2. _____

3. _____

Datum polaganja: _____
Zagreb, 2024.

Veliko hvala mentoru prof. dr. sc. Krešimiru Kumeričkom na vodstvu i pomoći pri izradi ovog diplomskog rada.

Hvala svim mojim prijateljima na opuštenim druženjima i lijepim trenutcima tijekom fakultetskih dana.
Posebno hvala Lini, Raphaeli i Dunji.

Hvala mojoj obitelji na podršci kroz cijelo školovanje.
Posebno hvala tati.

Sažetak

Partonske distribucijske funkcije protona opisuju raspodjelu impulsa njegovih sastavnica te stoga nose važnu informaciju o njegovoj kvarkovsko-gluonskoj dinamici. S obzirom na to da su fundamentalno neperturbativna veličina kvantne kromodinamike, određuju se posredno prilagodbom modela na eksperimentalne podatke. Diferencijalni udarni presjeci dubokih neelastičnih raspršenja ovise upravo o njima te se mogu smisliti razne metode ekstrakcije tih funkcija iz eksperimentalnih podataka različitih procesa. U ovom radu ispituje se metoda strojnog učenja za ekstrakciju spomenutih distribucijskih funkcija njihovim modeliranjem kao gausijanskih procesa te Monte Carlo dropout metoda neuronskih mreža za usporedbu. U svrhu ispitivanja metode uvedena su određena nefizikalna pojednostavljenja. Također, s istim ciljem zadane su nefizikalne partonske distribucijske funkcije koje služe za simulaciju podataka diferencijalnih udarnih presjeka nefizikalnih procesa. Distribucijske funkcije ekstrahirane iz simuliranih podataka mogu se tada usporediti s onim zadanim kako bi se na taj način mogla procijeniti uspješnost metode kao i potencijalno zaključiti nešto o njenoj primjenjivosti na stvarne eksperimentalne podatke. Također, određeni zaključci doneseni su usporedbom rezultata metode gausijanskih procesa s rezultatima Monte Carlo dropout metode neuronskih mreža.

Ključne riječi: Duboko neelastično raspršenje, Partonske distribucijske funkcije, Strojno učenje, Gausijanski procesi, Neuronske mreže, Monte Carlo dropout

Description of distribution of quarks and gluons in proton using Gaussian processes

Abstract

Parton distribution functions of proton describe the distribution of momenta of its components and thus provide an important information about its quark-gluon dynamics. Considering they are fundamentally non-perturbative quantity of quantum chromodynamics, they are determined indirectly by fitting the model to experimental data. Differential cross sections of deep inelastic scatterings depend on those functions and various methods of their extraction can be devised from experimental data of different processes. This thesis examines the machine learning method for extraction of the mentioned distribution functions by modeling them as Gaussian processes and the Monte Carlo dropout method of neural networks for comparison. For testing purposes of the method certain non-physical simplifications were introduced. Likewise, with the same goal there are given non-physical parton distribution functions which serve to simulate differential cross section data of different non-physical processes. Distribution functions extracted from the simulated data can then be compared with the given ones in order to evaluate the efficacy of the method and potentially conclude something on its applicability to real experimental data. Also, certain conclusions were drawn by comparing the results of the Gaussian processes method with the results from Monte Carlo dropout method of neural networks.

Keywords: Deep inelastic scattering, Parton distribution functions, Machine learning, Gaussian processes, Neural networks, Monte Carlo dropout

Sadržaj

1 Uvod **1**

2 Elektron-proton raspršenje **3**

2.1 Elastično raspršenje 3

2.2 Duboko neelastično raspršenje 6

3 Strojno učenje **9**

3.1 Funkcija gubitka 9

3.2 Optimizacija modela 11

3.3 Odabir modela i postavki učenja 13

4 Gausijanski procesi **17**

4.1 Regresija gausijanskih procesa - vektorska slika 18

4.2 Regresija gausijanskih procesa - funkcijska slika 20

4.3 Granična vjerojatnost 21

4.4 Ilustracija ekstrakcije PDF-a 22

5 Neuronske mreže **26**

5.1 Povijesni uvod 26

5.2 Neuron kao računalna jedinica 26

5.3 Potpuno povezana aciklična neuronska mreža 28

5.4 Algoritam propagacije unatrag 30

5.5 Stohastički gradijentni spust 31

5.6 Napredni optimizacijski algoritmi 35

5.7 Funkcija gubitka pri ekstrakciji PDF-a 38

6 Ekstrakcija PDF-a **40**

6.1 Metoda gausijanskih procesa 41

6.2 Metoda neuronskih mreža 42

6.3 Unakrsna provjera 43

6.4 Rezultati 44

7 Zaključak **47**

Dodaci **48**

A gaussian.ipynb **48**

B ffnn.ipynb **66**

Literatura **95**

1 Uvod

Početkom 19. stoljeća John Dalton postavio je prve postulate o postojanju atoma [1]. Joseph John Thomson je gotovo stotinu godina kasnije zajedno sa svojim suradnicima proveo eksperiment kojim je dokazano da su katodne zrake posebne čestice čija svojstva ne ovise o katodnom materijalu i koje su potom nazvane elektroni. Daljnjim razvojem fizike Ernest Rutherford i njegovi suradnici svojim su eksperimentom desetak godina nakon otkrića elektrona otkrili do tada skrivenu podstrukturu atoma. Raspršenjem alfa-čestica na tankim listićima zlata uvidjeli su da mali broj tih čestica skrene svoju putanju za više od 90°. Dobiveni rezultati doveli su ih do zaključka da se atomi sastoje od pozitivno nabijene jezgre koja sadrži većinu njihove mase i koju okružuju negativno nabijeni elektroni. Raspršenja su na taj način postala važan alat u fizici pri otkrivanju i proučavanju podstrukture materije kao i pri pronalasku novih čestica.

Danas se napretkom znanosti sve postojeće elementarne čestice dijele na čestice spina 1/2 ili fermione te prijenosnike sila spina 1 ili baždarne bozone. Higgsov bozon spina 0 posljednje je otkrivena elementarna čestica te se može svrstati sama u svoju kategoriju. Temeljnih fermiona u standardnom modelu ima 12 te svaki fermion ima i svoju pripadajuću antičesticu. Fermioni se nadalje dijele na šest leptona i šest kvarkova. Svi leptoni osim tri neutrina posjeduju električni naboj kao svojstvo te time sudjeluju u elektromagnetskim međudjelovanjima koja su opisana kvantnom elektrodinamikom. Kvarkovi osim električnog naboja posjeduju i naboj boje te tako osim u elektromagnetskim međudjelovanjima sudjeluju i u jakim nuklearnim. Za razliku od leptona, kvarkovi u svemiru opažaju se samo u vezanim sustavima, takozvanim hadronima, gdje su baždarni bozoni zvani gluoni prijenosnici jake nuklearne sile među njima. Teorija koja opisuje kvarkovsko-gluonske sustave naziva se kvantna kromodinamika te je njen najveći izazov neperturbativnost teorije pri niskim energijama.

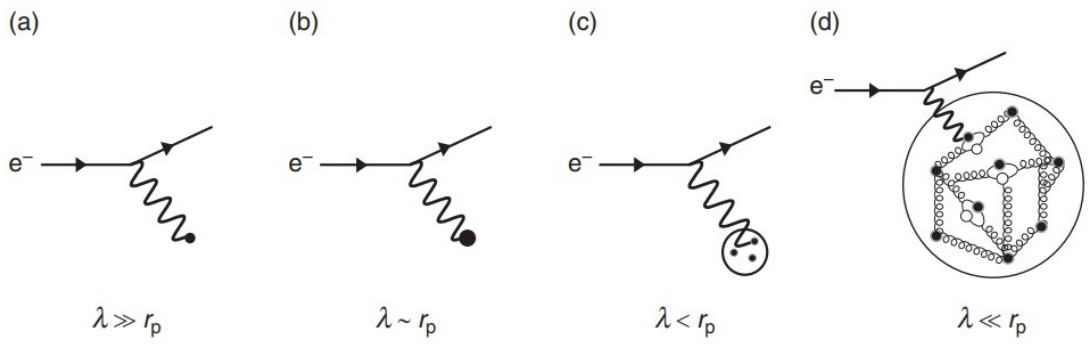
Dinamika interagirajućeg kvarkovsko-gluonskog sustava rezultira raspodjelama momenata, takozvanim partonskim distribucijskim funkcijama (engl. *Parton Distribution Function*, PDF), tih čestica unutar hadrona. Budući da je kvantna kromodinamika pri niskim energijama i udaljenostima relevantnim za neki hadron neperturbativna teorija, spomenute kinematičke raspodjele određuju se posredno nizom eksperimenata. Jedan od skupova procesa uz pomoć kojih se može dobiti uvid u tu složenu

dinamiku je duboko neelastično raspršenje (engl. *Deep Inelastic Scattering*, DIS) na hadronu o kojemu će biti više riječi u sljedećem poglavlju. Diferencijalni udarni presjek veličina je od presudne važnosti pri proučavanju bilo kakvog raspršenja u fizici te je njenu teorijsku poveznicu s veličinom od interesa ključno poznavati.

Poznavanjem poveznice između diferencijalnih udarnih presjeka odabranih procesa u kojima sudjeluje ispitivani hadron i njegovih partonskih distribucijskih funkcija mogu se smisliti razne metode ekstrakcije tih funkcija iz dostupnih eksperimentalnih podataka. Obje metode koje će se ovdje proučiti i usporediti spadaju u domenu strojnog učenja. Pristup koji ima veliki neistraženi potencijal ekstrakcije spomenutih distribucijskih funkcija je putem njihovog modeliranja kao gausijanskih procesa te će o njemu biti više riječi u četvrtom poglavlju. Regresija specifična za gausijanske procese otvara vrata za metodu ekstrakcije čiji je način rada ilustriran na kraju spomenutog četvrtog poglavlja i čiji će se rezultati na kraju usporediti s rezultatima dobivenim Monte Carlo dropout metodom neuronskih mreža. Matematička formulacija neuronskih mreža opisana je detaljno u petom poglavlju. Rad dviju metoda usporedit će se zadavanjem devet nefizikalnih partonskih distribucijskih funkcija po izboru koje će se nastojati replicirati iz simuliranih eksperimentalnih podataka diferencijalnih udarnih presjeka jednakog broja nefizikalnih procesa.

2 Elektron-proton raspršenje

Izračuni diferencijalnih udarnih presjeka za sudare koji uključuju protone moraju uzeti u obzir njegovu podstrukturu. Primjerice, u sudaru elektrona i protona pri niskim energijama dominantan proces je elastično raspršenje koje je opisano međudjelovanjem virtualnog fotona s protonom u cijelosti te stoga pruža mogućnost istrage globalnih svojstva protona kao na primjer njegovog nabojnog polumjera [2]. Povećanjem energije dominantan proces postaje duboko neelastično raspršenje gdje se elektron sada elastično raspršuje na kvarkovima i gluonima u protonu, a ne više na protonu u cijelosti. Priroda elektron-proton raspršenja ovisi o odnosu valne duljine virtualnog fotona i polumjera protona te se prema tome razaznaju četiri glavna režima ovog sudara ilustrirana na Slici 2.1 ispod.



Slika 2.1: Priroda elektron-proton raspršenja prema odnosu valne duljine virtualnog fotona i polumjera protona [2]

2.1 Elastično raspršenje

Dobro poznati režimi elastičnog raspršenja elektrona na protonu su Rutherfordovo i Mottovo raspršenje [2]. U oba slučaja energija elektrona je dovoljno mala da je kinetička energija odbijenog protona zanemariva u odnosu na njegovu energiju mirovanja. Proton se u izvodu spomenutih raspršenja aproksimira točkastom česticom što u slučaju velike valne duljine virtualnog fotona u odnosu na radijus protona prema Slici 2.1 ima smisla. Rutherfordovo raspršenje se obično izvodi iz prvog reda računa smetnje promatranjem raspršenja nerelativističkog elektrona u statičnom Coulombovom potencijalu protona $V(r) = \alpha/r$ te se za diferencijalni udarni presjek po prostornom kutu tada dobije:

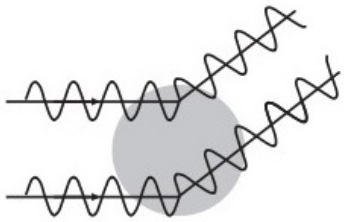
$$\left(\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega}\right)_{\text{Rutherford}} = \frac{\alpha^2}{16E_K^2\sin^4(\theta/2)}, \tag{2.1}$$

gdje je α snaga sprezanja elektromagnetskog međudjelovanja, a $E_K = \mathrm{p}^2/2m_e$ energija nerelativističkog elektrona mase m_e s tro-impulsom norme p. S druge strane, Mottovo raspršenje je granica elektron-proton raspršenja gdje je elektron sada relativistički, no odskok protona je i dalje zanemariv. Mottovo raspršenje može se izvesti promatrajući raspršenje relativističkog elektrona u Coulombovom potencijalu jezgre bez spina te tada diferencijalni udarni presjek po prostornom kutu ima oblik:

$$\left(\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega}\right)_{\text{Mott}} = \frac{\alpha^2}{4E^2\sin^4(\theta/2)}\cos^2\frac{\theta}{2}, \tag{2.2}$$

gdje je $E = \gamma_e m_e$ energija relativističkog elektrona s Lorentzovim relativističkim faktorom γ_e.

Kako bi se uzeo u obzir konačan opseg raspodjele naboja protona uvodi se takozvana funkcija strukture (engl. *form factor*). Naime, kao što ilustrira Slika 2.2 ispod, funkcija strukture uzima u obzir fazne razlike između doprinosa raspršenom valu od različitih točaka raspodjele naboja [2]. Ako je valna duljina virtualnog fotona puno veća od polumjera protona doprinosi raspršenom valu od pojedinih točaka raspodjele naboja bit će u fazi te stoga interferirati konstruktivno. S druge strane, ako je valna duljina virtualnog fotona manja od polumjera protona faze raspršenih valova jako će ovisiti o položaju dijela raspodjele naboja odgovornim za raspršenje. U tom slučaju, destruktivne interferencije smanjuju ukupnu amplitudu.



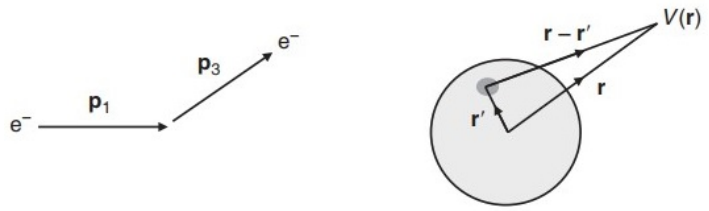
Slika 2.2: Ilustracija podrijetla faktora oblika pri elastičnom raspršenju [2]

Gustoća naboja može se označiti kao $Q\rho(\mathbf{r'})$, gdje je Q ukupan naboj, a $\rho(\mathbf{r'})$ raspodjela naboja normalizirana na jedinicu:

$$\int \rho(\mathbf{r'})\mathrm{d}^3\mathbf{r'} = 1. \tag{2.3}$$

Potencijal raspodjele naboja u točki na udaljenosti $\mathbf{r}$ od ishodišta s oznakama uvedenim na Slici 2.3 ima oblik:

$$V(\mathbf{r}) = \int \frac{Q\rho(\mathbf{r'})}{4\pi|\mathbf{r} - \mathbf{r'}|} \mathrm{d}^3\mathbf{r'}. \tag{2.4}$$



Slika 2.3: Potencijal konačne raspodjele naboja [2]

U Bornovoj aproksimaciji valne funkcije ulaznog i izlaznog elektrona izražene su kao ravni valovi, $\psi_i = e^{i(\mathbf{p}_1 \cdot r - Et)}$ i $\psi_f = e^{i(\mathbf{p}_3 \cdot r - Et)}$ uz oznake sa Slike 2.3, te je tada matrični element najnižeg reda računa smetnje za raspršenje na protonu jednak:

$$\mathcal{M}_{fi} = \langle \psi_f | V(\mathbf{r}) | \psi_i \rangle = \int e^{-i\mathbf{p}_3 \cdot \mathbf{r}} V(\mathbf{r}) e^{i\mathbf{p}_1 \cdot \mathbf{r}} \mathrm{d}^3\mathbf{r}. \tag{2.5}$$

Uvođenjem oznake za tro-impuls virtualnog fotona $\mathbf{q} = (\mathbf{p}_1 - \mathbf{p}_3)$ i uvrštavanjem potencijala iz izraza (2.4) slijedi:

$$\mathcal{M}_{fi} = \int e^{i\mathbf{q}\cdot\mathbf{R}} \frac{Q}{4\pi|\mathbf{R}|} \mathrm{d}^3\mathbf{R} \int \rho(\mathbf{r'}) e^{i\mathbf{q}\cdot\mathbf{r'}} \mathrm{d}^3\mathbf{r'}, \tag{2.6}$$

gdje je dodatno uveden vektor $\mathbf{R} = \mathbf{r} - \mathbf{r'}$ kao pokrata. Može se primijetiti kako je integral preko $\mathrm{d}^3\mathbf{R}$ ekvivalentan izrazu za matrični element raspršenja na potencijalu točkastog naboja. Matrični element za raspodjelu naboja može se stoga napisati kao:

$$\mathcal{M}_{fi} = \mathcal{M}_{fi}^{pt} F(\mathbf{q}^2), \tag{2.7}$$

gdje je $\mathcal{M}_{fi}^{pt}$ ekvivalent matričnom elementu za točkasti proton, dok je funkcija strukture $F(\mathbf{q}^2)$ dana kao:

$$F(\mathbf{q}^2) = \int \rho(\mathbf{r}) e^{i\mathbf{q}\cdot\mathbf{r}} \mathrm{d}^3\mathbf{r}. \tag{2.8}$$

Izraz za Mottovo raspršenje sada prema Fermijevom zlatnom pravilu poprima oblik:

$$\left(\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega}\right)_{\text{Mott}} \rightarrow \frac{\alpha^2}{4E^2 \sin^4(\theta/2)} \cos^2\frac{\theta}{2} |F(\mathbf{q}^2)|^2. \tag{2.9}$$

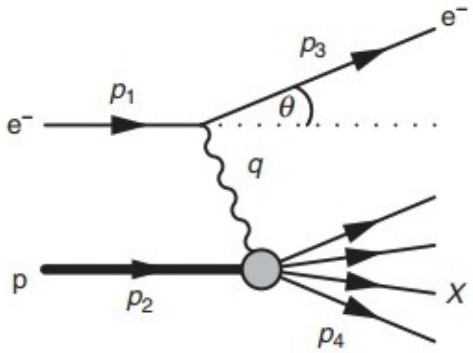
Rutherfordovo i Mottovo raspršenje mogu se dakle generalizirati na konačnu raspodjelu naboja uz pomoć funkcije strukture, no i dalje se trzaj protona zanemaruje. Može se pokazati kako je najopćenitiji Lorentz-invarijantan izraz za elektron-proton raspršenje izmijenom jednog virtualnog fotona dan takozvanom Rosenbluthovom formulom:

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega} = \frac{\alpha^2}{4E_1^2 \sin^4(\theta/2)} \frac{E_3}{E_1} \left(\frac{G_E^2 + \tau G_M^2}{1+\tau} \cos^2(\theta/2) + 2\tau G_M^2 \sin^2 \frac{\theta}{2} \right) \tag{2.10}$$

uz oznaku $\tau = Q^2/4m_p^2$, gdje je $Q^2 = -q^2$ negativna vrijednost kvadriranog četvero-impulsa q virtualnog fotona, a m_p masa protona. Funkcije strukture $G_E(Q^2)$ i $G_M(Q^2)$ ovdje uvedene ovise o četvero-impulsu virtualnog fotona te opisuju nabojne i magnetske raspodjele protona. Ekstrakcija spomenutih funkcija strukture iz eksperimentalnih podataka diferencijalnog udarnog presjeka može dati određeni uvid u strukturu protona.

2.2 Duboko neelastično raspršenje

Zbog konačne veličine protona udarni presjek za elektron-proton elastično raspršenje brzo opada s energijom [2]. Za razliku od elastičnog elektron-proton raspršenja koje se može opisati samo s jednom kinematičkom varijablom, kod neelastičnog raspršenja $e^- p \rightarrow e^- X$ prikazanog na Slici 2.4 postoji dodatni stupanj slobode. Dodatni stupanj slobode dolazi od invarijantne mase W nastalog hadronskog stanja X koja sada ovisi o četvero-impulsu virtualnog fotona, $W^2 = p_4^2 = (p_2 + q)^2$ uz oznake sa Slike 2.4, te stoga može poprimiti razne vrijednosti za razliku od elastičnog raspršenja gdje je invarijantna masa završnog stanja uvijek jednaka masi protona.



Slika 2.4: Neelastično raspršenje elektrona na protonu [2]

Neelastično raspršenje stoga se opisuje s dvije nezavisne kinematičke varijable. Neke od kinematičkih varijabli koje se koriste su Bjorkenov x te varijable Q^2, y i ν:

$$Q^2 = -q^2 = -(p_1 - p_3)^2, \tag{2.11}$$

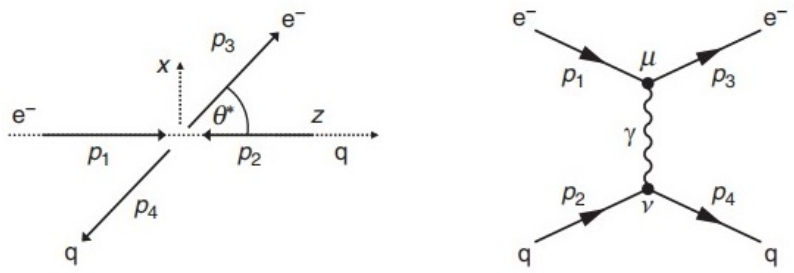
$$x = \frac{Q^2}{2p_2 \cdot q} \quad (0 \leq x \leq 1), \tag{2.12}$$

$$y = \frac{p_2 \cdot q}{p_2 \cdot p_1} \quad (0 \leq y \leq 1), \tag{2.13}$$

$$\nu = \frac{p_2 \cdot q}{m_p}, \tag{2.14}$$

gdje su p_1, p_2, p_3 te q četvero-impulsi označeni na Slici 2.4, a m_p je masa protona.

Opis dubokog neelastičnog elektron-proton raspršenja može se započeti promatranjem elastičnog raspršenja elektrona na kvarku prikazanog na Slici 2.5 ispod.



Slika 2.5: Elastično raspršenje elektrona na kvarku [2]

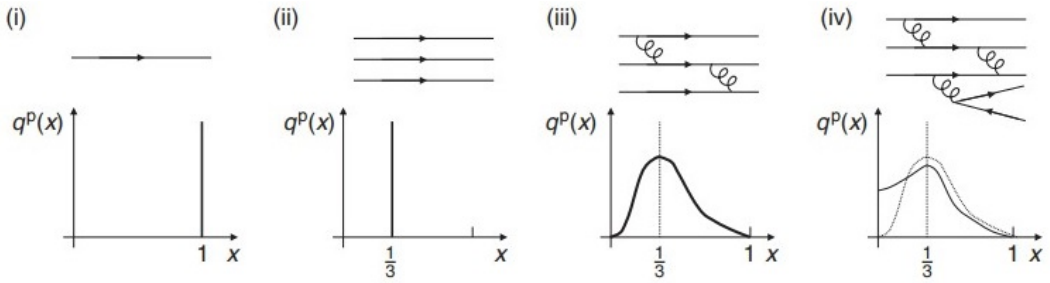
U granici visokih energija gdje se mase elektrona i kvarka mogu zanemariti izraz za diferencijalni udarni presjek elastičnog elektron-kvark raspršenja u prvom redu računa smetnje jednak je:

$$\frac{d\sigma}{dQ^2} = \frac{4\pi\alpha^2 Q_i^2}{Q^4}\left[(1-y) + \frac{y^2}{2}\right], \tag{2.15}$$

gdje je Q_i naboj kvarka okusa i [2].

Kvarkovi unutar protona naravno međudjeluju izmjenom gluona te dinamika tog međudjelujućeg sustava rezultira njihovom impulsnom raspodjelom [2]. Spomenute raspodjele izražene su preko partonskih distribucijskih funkcija (engl. *Parton Distribution Function*, PDF). Partonska distribucijska funkcija $q^p(x)$ za kvark određenog

okusa i definira se tako da $q_i^p(x)\delta x$ predstavlja broj kvarkova okusa i kojima je udio longitudinalnog impulsa u protonu između x i $x + \delta x$.



Slika 2.6: Četiri moguća oblika PDF-a: (i) jedna točkasta čestica; (ii) tri statična kvarka svaki s 1/3 impulsa protona; (iii) tri međudjelujuća kvarka koji mogu izmjenjivati impulse; (iv) međudjelujući kvarkovi uključujući više redove računa smetnje [2]

Nadalje, sada se prema izrazu (2.15) može zaključiti da diferencijalni udarni presjek za elastično raspršenje elektrona na kvarku određenog okusa i s udijelom impulsa između x i $x + \delta x$ ima oblik:

$$\frac{d\sigma}{dQ^2} = \frac{4\pi\alpha^2}{Q^4}\left[(1-y) + \frac{y^2}{2}\right] \times Q_i^2 q_i^p(x)\delta x. \tag{2.16}$$

Izraz dobiven iznad potom se podijeli sa δx te se zbroje svi okusi kvarkova koji sudjeluju pri raspršenju pa slijedi:

$$\frac{d^2\sigma}{dxdQ^2} = \frac{4\pi\alpha^2}{Q^4}\left[(1-y) + \frac{y^2}{2}\right] \sum_i Q_i^2 q_i^p(x). \tag{2.17}$$

Kao što je dobro poznato, u statičkom se modelu proton sastoji od dva up-kvarka i jednog down-kvarka. Ipak, proton je u stvarnosti dinamičan sustav u kojem jako međudjelujući kvarkovi neprestano izmjenjuju virtualne gluone koji mogu fluktuirati u kvark-antikvark parove [2]. Gluoni s velikim impulsom potisnuti su gluonskim propagatorom oblika $1/q^2$ te je stoga more kvarkova i antikvarkova sklono nastajanju pri malim vrijednostima x.

Poznavanje podataka o diferencijalnim udarnim presjecima više različitih procesa dubokog neelastičnog raspršenja na protonu (primjerice neutrino-proton DIS, mion-proton DIS) kao i nekih drugih procesa u kojima sudjeluje proton može poslužiti pri ekstrakciji PDF-a koje onda otkrivaju neke detalje o podstrukturi protona.

3 Strojno učenje

Ljudi su se još prije ostvarenja ideje o računalima počeli pitati mogu li takvi strojevi postati pametni [3]. Danas je umjetna inteligencija područje s puno praktičnih primjena i aktivnih istraživačkih tema koje uspješno napreduje. Umjetna inteligencija velikim je dijelom inspirirana onom koju nalazimo u živim bićima. Naime, sustavi umjetne inteligencije za svoje funkcioniranje trebaju na neki način imati sposobnost stjecanja vlastitog znanja izvlačenjem određenih obrazaca iz neobrađenih podataka. Ta sposobnost umjetne inteligencije naziva se strojno učenje (engl. *machine learning*).

Strojno učenje može se opisati kao razvijanje algoritama koji uče iz podataka kako bi prepoznali obrasce i donosili odluke. Prvi korak u strojnom učenju je odabir modela (engl. *model selection*) koji će nastojati riješiti zadani problem. Model je skup različito parametriziranih hipoteza odnosno funkcija te se proces učenja odnosi na dobar odabir jedne od njih. S obzirom na prirodu problema strojno učenje može se podijeliti u dvije osnovne kategorije: učenje pod nadzorom (engl. *supervised learning*) i učenje bez nadzora (engl. *unsupervised learning*) [4].

Strojnom učenju pod nadzorom svojstven je skup za učenje u kojem svaki primjer ima svoju oznaku (engl. *label*). Neki parametri odabranog modela ostavljaju se slobodni s ciljem da u idućem koraku njihovom optimizacijom model bude u mogućnosti što bolje opisati dostupne podatke te na taj način i potencijalno dobro predvidjeti oznake onih nedostupnih. U ovaj oblik strojnog učenja spadaju problemi regresije i klasifikacije.

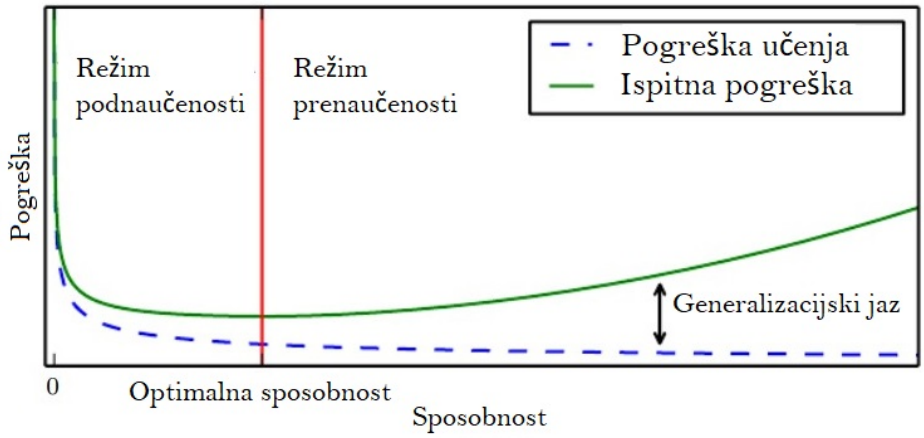
S druge strane, cilj strojnog učenja bez nadzora može biti otkrivanje anomalija u podatcima ili grupiranje podataka sličnih karakteristika u određene skupine [4]. Skup za učenje ovog oblika strojnog učenja je neoznačen. Jedan od primjera takvog učenja je grupiranje nadolazeće elektroničke pošte u skupine poželjne i nepoželjne pošte na temelju njenog sadržaja.

3.1 Funkcija gubitka

Kako bi optimizacija modela bila moguća potrebno je poznavati neku funkciju gubitka (engl. *loss function*) primjerenu za problem koji se nastoji riješiti. Optimizacija modela vrši se tada minimizacijom jedne takve funkcije izvrijednjene na skupu za

učenje (engl. *training set*) uz pomoć odabranog algoritma. Osim na skupu za učenje funkcija gubitka može se izvrijedniti i na ispitnom skupu (engl. *validation set*) koji nakon procesa učenja na taj način može poslužiti kao test kvalitete generalizacije optimiziranog modela na nedostupne podatke.

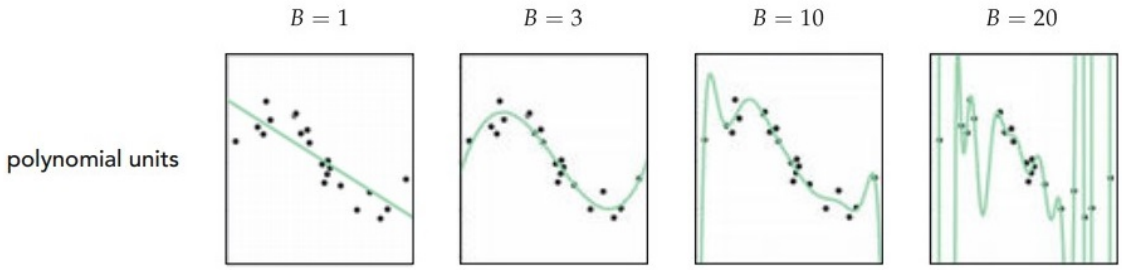
Konačni cilj strojnog učenja je dobar cjeloviti opis problema, a to uključuje i generalizaciju na podatke koji računalu trenutno nisu dostupni. Najbolji model stoga prema Slici 3.1 nije onaj koji minimizira funkciju gubitka izvrijednjenu na skupu za učenje, već onaj koji ju minimizira izvrijednjenu na ispitnom skupu. S obzirom na odnos pogreške učenja i ispitne pogreške postoje dva ekstremna režima naučenosti - podnaučenost (engl. *underfitting*) i prenaučenost (engl. *overfitting*).



Slika 3.1: Funkcija gubitka izvrijednjena na skupu za učenje i na ispitnom skupu u ovisnosti o sposobnosti modela da opiše veliki broj problema [3]

Režimu podnaučenosti svojstvena je velika pogreška učenja, a kao posljedica toga i velika ispitna pogreška. Drugim riječima, model tada ne uspijeva dobro opisati skup za učenje, a onda niti ispitni skup. Podnaučenost može biti posljedica odabira prejednostavnog modela koji nije u mogućnosti dohvatiti složenost zadanog problema ili posljedica preranog prekida u učenju. S druge strane, režim prenaučenosti predstavlja situaciju u kojoj model svojom složenošću jako precizno opisuje skup za učenje, ali ne uspijeva kvalitetno predvidjeti ponašanje podataka izvan spomenutog skupa. Zorna ilustracija posljedice podnaučenosti i prenaučenosti može se vidjeti na Slici 3.2 gdje prilagodba polinoma različitih stupnjeva na podatke daje rezultate različitih valjanosti. Naime, u priloženom primjeru na Slici 3.2 polinom prvog stupnja ne dohvaća složenost dostupnih podataka, polinom drugog stupnja daje dobar opis problema uz mogućnost kvalitetne generalizacije, dok polinomi viših stupnjeva sve bolje

opisuju dani skup za učenje time narušavajući mogućnost dobrog cjelovitog opisa problema.



Slika 3.2: Prilagodba polinoma različitih stupnjeva na dostupne podatke [5]

Odabirom modela visoke sposobnosti da opiše veliki broj različitih problema dolazi se dakle do opasnosti od prenaučenosti. Tom problemu može se doskočiti na razne načine ovisno o metodi strojnog učenja koja se provodi. Skupan naziv za metode izbjegavanja režima prenaučenosti je regularizacija [3]. Jedna od široko primjenjivih metoda je transformacija funkcije gubitka na način:

$$\text{funkcija gubitka} \leftarrow \text{funkcija gubitka} + \lambda \cdot (\text{složenost hipoteze}), \tag{3.1}$$

gdje je λ vrijednost koja u procesu optimizacije parametara određuje odnos kažnjavanja pogreške učenja i složenosti hipoteze. U jednom od slučaja sa Slike 3.2 složenost hipoteze može se opisati kao zbroj apsolutnih vrijednosti koeficijenata polinoma (L1 regularizacija) ili kao zbroj kvadrata koeficijenata polinoma (L2 regularizacija).

3.2 Optimizacija modela

Postoje razni algoritmi za minimizaciju funkcije gubitka te se svi na neki način u osnovi temelje na takozvanom gradijentnom spustu. Njihov krajnji cilj je optimizacijom parametara modela dostići globalni minimum funkcije gubitka izvrijednjene na skupu za učenje i po mogućnosti to učiniti što brže i efikasnije. Neki od njih su standardni gradijentni spust, stohastički gradijentni spust, njegova implementacija s različitim oblicima takozvanog impulsa, AdaGrad, RMSprop, Adam i tako dalje [3]. Pojednostavljena shema algoritma gradijentnog spusta nalazi se na Slici 3.3 ispod te se temelji na činjenici da općenita funkcija najbrže opada u smjeru obrnutom od smjera svog gradijenta. Pojedina iteracija u **while** petlji sa Slike 3.3 naziva se epoha

učenja te se na kraju svake epohe parametri modela ažuriraju na način da algoritam nastoji doći korak bliže jednom od minimuma funkcije gubitka.

Ulaz: skup za učenje (X, Y), diferencijabilna funkcija gubitka $\mathcal{L}(X, Y, \mathbf{w})$, stopa učenja $\eta^{(i)}$, početna parametarska točka $\mathbf{w}^{(0)} = \left(\mathrm{w}_1^{(0)}, ..., \mathrm{w}_n^{(0)}\right)$

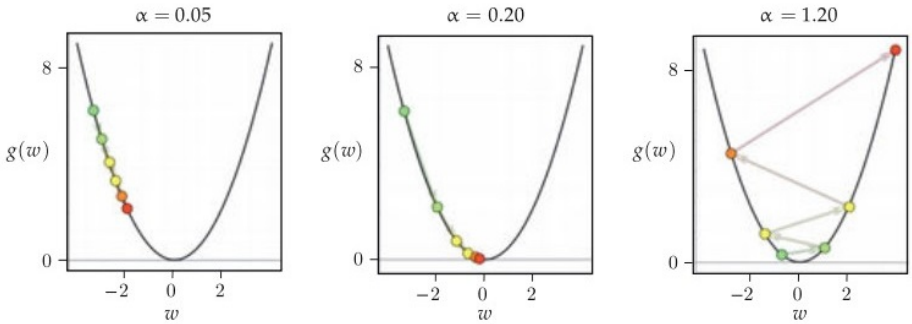
1. $i \leftarrow 0$
2. **while** uvjet zaustavljanja nije ispunjen **do**
3. Računanje $\nabla_{\mathbf{w}} \mathcal{L}(X, Y, \mathbf{w}^{(i)})$
4. Ažuriranje $\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \eta^{(i)} \nabla_{\mathbf{w}} \mathcal{L}(X, Y, \mathbf{w}^{(i)})$
5. $i \leftarrow i + 1$
6. **end while**.

Slika 3.3: Shematski prikaz algoritma gradijentnog spusta

Mogući uvjeti zaustavljanja **while** petlje sa Slike 3.3 iznad su:

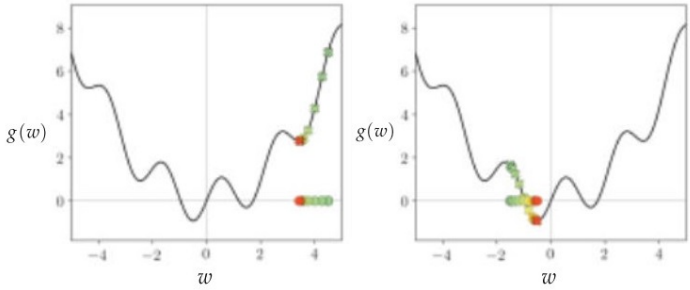
- $||\nabla_{\mathbf{w}} \mathcal{L}(X, Y, \mathbf{w}^{(i)})||_2 \approx 0 \rightarrow$ mala norma gradijenta,
- $||\mathbf{w}^{(i+1)} - \mathbf{w}^{(i)}||_2 \approx 0 \rightarrow$ mala promjena parametara,
- $|\mathcal{L}(X, Y, \mathbf{w}^{(i+1)}) - \mathcal{L}(X, Y, \mathbf{w}^{(i)})| \approx 0 \rightarrow$ mala promjena funkcije gubitka,
- indeks i postiže zadani broj epoha.

Stopa učenja bilo kojeg algoritma minimizacije mora biti pomno izabrana. Na Slici 3.4 jasno se vidi da premala stopa učenja dovodi do spore konvergencije, dok prevelika stopa može biti uzrok nemogućnosti konvergencije.



Slika 3.4: Ilustrativni prikaz rada algoritma standardnog gradijentnog spusta po epohama za tri stope učenja - od lijeva na desno: premala stopa, optimalna stopa, prevelika stopa [5]

Također, na Slici 3.5 zorno je prikazano kako odabir početne parametarske točke može utjecati na krajnji rezultat algoritma na način da algoritam tako završi u različitim minimumima funkcije gubitka.



Slika 3.5: Ilustrativni prikaz rada običnog algoritma gradijentnog spusta po epohama za dvije različite početne parametarske točke [5]

Algoritmi u širokoj uporabi poput Adam puno su sofisticiraniji od standardnog gradijentnog spusta te tijekom optimizacije na temelju gradijenata iz prethodnih epoha prilagođavaju stopu učenja iz epohe u epohu svakom pojedinom parametru zasebno [3]. Također, upotrebom raznih metoda (poput raznih oblika impulsa) takvi algoritmi nastoje izbjeći zaustavljanje u lokalnim minimumima funkcije gubitka na koje naiđu putem kako bi potencijalno stigli do onog globalnog.

3.3 Odabir modela i postavki učenja

Prva stavka u pokušaju dobrog opisa nekog problema strojnim učenjem je odabir modela odnosno skupa parametriziranih hipoteza. Primjer korištenja različitih modela za opis istog problema je prilagodba različitih stupnjeva polinoma na podatke sa Slike 3.2. Parametri modela u tom su slučaju koeficijenti polinoma odabranog stupnja te se nad njima vrši optimizacija. Također, skup iz kojeg se bira jedan optimalan model može biti parametriziran kao što su parametrizirane i hipoteze modela, no na toj razini govori se o takozvanim hiperparametrima. Hiperparametri modela međuostalim mogu na primjer biti stopa učenja algoritma minimizacije η ili prethodno uveden faktor regularizacije funkcije gubitka λ [3]. Hiperparametri se stoga mogu smatrati postavkama jednog modela uz pomoć kojih se može upravljati ponašanjem procesa učenja. Postupak dakle kreće s biranjem modela $\mathcal{H}_i$ (primjer: regresija polinomima ili regresija neuronskim mrežama), potom se biraju postavke modela odnosno njegovi hiperparametri $\boldsymbol{\theta}$ (primjer: stopa učenja) te se naposlijetku tako definiranom modelu optimiziraju parametri $\mathbf{w}$ (primjer: koeficijenti

polinoma). Treba primijetiti da se vrijednosti hiperparametara biraju prije početka procesa učenja te algoritam učenja ne prilagođava njihove vrijednosti, već samo vrijednosti parametara. Ipak, postoje algoritmi koji mogu poslužiti za dobar odabir modela i njegovih hiperparametara kojem bi se potom standardnim procesom učenja optimizirali njegovi parametri.

Jedna od metoda odabira modela ili njegovih hiperparametara je optimizacijom svih kandidata na istom skupu za učenje. Ako je skup dostupnih podataka dovoljno velik njegova podjela na veći skup za učenje i manji ispitni skup neće onemogućiti dobar opis cjelokupnog zadanog problema. Svaki kandidat optimizira se tada uz pomoć skupa za učenje te se potom njegova izvedba ocjenjuje izvrjednjavanjanjem funkcije gubitka na ispitnom skupu [4]. Optimalan kandidat tada je onaj s najmanjom ispitnom greškom.

Ipak, ponekad skup dostupnih podataka nije dovoljno velik te stoga njegova podjela na skup za učenje i ispitni skup nije povoljna opcija jer bi se na taj način izgubilo previše dragocjene informacije korisne za proces učenja [4]. Problemu odabira modela ili njegovih hiperparametara u slučaju malog skupa podataka tada se priskače takozvanim algoritmom unakrsne provjere (engl. *cross validation*) shematski prikazanim na Slici 3.6 ispod.

Ulaz: skup za učenje $\mathrm{S} = \{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_N, \mathbf{y}_N)\}$, funkcija gubitka $\mathcal{L}$,
skup hiperparametara Θ, algoritam za učenje A, prirodni broj k

1. podjela skupa S na k disjunktnih podskupova S_1,S_2,...,S_k
2. $\forall\ \theta \in \Theta$:
3. for $i = 1, ..., k$
4. $h_{i,\theta} = \mathrm{A}(\mathrm{S} \backslash \mathrm{S}_i; \theta)$
5. $\mathrm{error}(\theta) = \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}_{\mathrm{S}_i}(h_{i,\theta})$

Izlaz:
$\theta^* = \mathrm{argmin}_\theta[\mathrm{error}(\theta)]$
$h_{\theta^*} = \mathrm{A}(\mathrm{S}; \theta^*)$

Slika 3.6: Shematski prikaz unakrsne provjere za odabir postavki modela [4]

Naime, skup dostupnih podataka podijeli se na k disjunktnih podskupova te se svaki kandidat optimizira k puta na način da se u svakoj iteraciji izdvoji jedan podskup koji služi kao ispitni skup. Srednja vrijednost ispitnih pogrešaka jednog kandidata tada

predstavlja njegovu ocjenu valjanosti. Kandidat s najmanjom srednjom vrijednošću ispitnih pogrešaka odabire se kao optimalan te se potom vrši optimizacija njegovih parametara standardnim algoritmom učenja gdje se pritom koriste svi dostupni podatci.

Osim prethodno opisanih metoda odabira modela, njegovih postavki i parametara ističe se i metoda implementacije Bayesovog teorema koji u matematičkom obliku glasi:

$$P(A_k|B) = \frac{P(B|A_k)P(A_k)}{P(B)} = \frac{P(B|A_k)P(A_k)}{\sum_i P(B|A_i)P(A_i)}, \tag{3.2}$$

gdje su A_i i B neki događaji s određenom vjerojatnošću pojavljivanja P.
Izraz (3.2) iznad u kontekstu odabira parametara $\mathbf{w}$ modela $\mathcal{H}_i$ s hiperparametrima $\boldsymbol{\theta}$ poprima oblik:

$$p(\mathbf{w}|Y, X, \boldsymbol{\theta}, \mathcal{H}_i) = \frac{p(Y|X, \mathbf{w}, \mathcal{H}_i)p(\mathbf{w}|\boldsymbol{\theta}, \mathcal{H}_i)}{p(Y|X, \boldsymbol{\theta}, \mathcal{H}_i)}, \tag{3.3}$$

gdje $p(Y|X, \mathbf{w}, \mathcal{H}_i)$ označuje vjerojatnost da određeni skup parametara $\mathbf{w}$ opisuje skup za učenje, dok se $p(\mathbf{w}|\boldsymbol{\theta}, \mathcal{H}_i)$ odnosi na znanje o skupu parametara prije poznavanja podataka [6]. Krajnji rezultat vjerojatnosti određenog skupa parametara $\mathbf{w}$ dakle uzima u obzir prvotno znanje o parametrima i stečeno znanje o podatcima. Normalizirajuća konstanta $p(Y|X, \boldsymbol{\theta}, \mathcal{H}_i)$ u nazivniku posljednjeg izraza ne ovisi o parametrima jer se odnosi na graničnu vjerojatnost (engl. *marginal likelihood*) da određeni model s odabranim postavkama opisuje skup za učenje te je dana kao:

$$p(Y|X, \boldsymbol{\theta}, \mathcal{H}_i) = \int p(Y|X, \mathbf{w}, \mathcal{H}_i)p(\mathbf{w}|\boldsymbol{\theta}, \mathcal{H}_i)\mathrm{d}\mathbf{w} \tag{3.4}$$

Nadalje, prijelazom na višu razinu parametrizacije dolazi se do hiperparametara te se analogno dobije izraz:

$$p(\boldsymbol{\theta}|Y, X, \mathcal{H}_i) = \frac{p(Y|X, \boldsymbol{\theta}, \mathcal{H}_i)p(\boldsymbol{\theta}|\mathcal{H}_i)}{p(Y|X, \mathcal{H}_i)}, \tag{3.5}$$

gdje sada $p(\boldsymbol{\theta}|\mathcal{H}_i)$ označava znanje o hiperparametrima prije pogleda na podatke. Normalizirajuća konstanta $p(Y|X, \mathcal{H}_i)$ dana je kao:

$$p(Y|X, \mathcal{H}_i) = \int p(Y|X, \boldsymbol{\theta}, \mathcal{H}_i)p(\boldsymbol{\theta}|\mathcal{H}_i)\mathrm{d}\boldsymbol{\theta}. \tag{3.6}$$

Na završnoj razini govori se dakako o odabiru samog modela:

$$p(\mathcal{H}_i|Y,X) = \frac{p(Y|X,\mathcal{H}_i)p(\mathcal{H}_i)}{p(Y|X)}, \tag{3.7}$$

gdje je $p(Y|X) = \sum_i p(Y|X,\mathcal{H}_i)p(\mathcal{H}_i)$. Sada se može primijetiti kako implementacija cjelokupne metode iziskuje računanje nekoliko integrala. Ovisno o detaljima modela, spomenute integrale može biti nemoguće analitički riješiti te se stoga općenito pribjegava analitičkim aproksimacijama ili raznim numeričkim metodama. U praksi se pogotovo integral u izrazu (3.6) teško rješava te se stoga u svrhu pronalaska optimalnih postavki modela kao aproksimacija cjelokupne metode samo maksimizira granična vjerojatnost iz izraza (3.4) s obzirom na skup hiperparametara $\boldsymbol{\theta}$. Pristup maksimizacije granične vjerojatnosti koristi se u metodi regresije specifičnoj za gausijanske procese o kojoj više govori sljedeće poglavlje.

4 Gausijanski procesi

Gausijanski proces skup je nasumičnih varijabli od kojih svaki njihov konačan broj čini multivarijantnu Gaussovu raspodjelu, poopćenje Gaussove raspodjele na više dimenzija [6]. Multivarijantna Gaussova raspodjela za N-dimenzionalni nasumični vektor $\mathbf{f}$ oblika

$$\mathbf{f} = (f_1, ..., f_N)^T = (f(\mathbf{x}_1), ..., f(\mathbf{x}_N))^T \tag{4.1}$$

označuje se kao:

$$\mathbf{f} \sim \mathcal{N}_N(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{4.2}$$

gdje je

$$\begin{aligned} \boldsymbol{\mu} = E[\mathbf{f}] &= (E(f_1), ..., E(f_N))^T \\ &= (E[f(\mathbf{x}_1)], ..., E[f(\mathbf{x}_N)])^T \equiv (\mu(\mathbf{x}_1), ..., \mu(\mathbf{x}_N))^T \end{aligned} \tag{4.3}$$

srednja vrijednost i

$$\begin{aligned} \Sigma_{i,j} &= E[(f_i - \mu_i)(f_j - \mu_j)] \\ &= E[(f(\mathbf{x}_i) - \mu_i)(f(\mathbf{x}_j) - \mu_j)] = cov[f(\mathbf{x}_i), f(\mathbf{x}_j)] \equiv k(\mathbf{x}_i, \mathbf{x}_j) \end{aligned} \tag{4.4}$$

funkcija kovarijance. Gustoća multivarijantne Gaussove raspodjele uz prethodno uvedene oznake ima sljedeći oblik [7]:

$$\mathcal{N}_N(\mathbf{f}|\mu, \Sigma) = \frac{1}{(2\pi)^{N/2}} \frac{1}{|\Sigma|^{1/2}} \cdot \exp\left[-\frac{1}{2}(\mathbf{f} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{f} - \boldsymbol{\mu}) \right]. \tag{4.5}$$

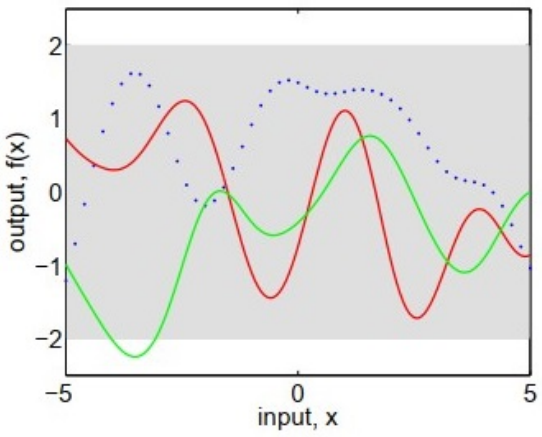
Gausijanski proces je dakle u potpunosti određen svojom srednjom vrijednošću i funkcijom kovarijance te se uz prethodno uvedene oznake piše kao:

$$f(\mathbf{x}) \sim \mathcal{GP}(\mu(\mathbf{x}), k(\mathbf{x}, \mathbf{x'})). \tag{4.6}$$

Funkcija kovarijance koja se često koristi kod gausijanskih procesa je takozvana kvadrirana eksponencijalna funkcija (engl. *squared-exponential*):

$$k(\mathbf{x}, \mathbf{x'}) = \texttt{exp.quad}(\mathbf{x}, \mathbf{x'}; l) = \exp\Big(-\frac{1}{2l^2} |\mathbf{x} - \mathbf{x'}|^2 \Big). \tag{4.7}$$

Na primjer, generiranjem nasumičnog Gaussovog vektora koristeći prethodno spomenutu funkciju kovarijance sa skalom $l = 1$ te vizualizacijom generiranih vrijednosti kao funkcije ulaznih vrijednosti dobije se Slika 4.1 ispod.



Slika 4.1: Nasumične funkcije gausijanskog procesa srednje vrijednosti $\mu(\mathrm{x}) = 0$ i funkcije kovarijance $k(\mathrm{x}, \mathrm{x'}) = \texttt{exp.quad}(\mathrm{x}, \mathrm{x'}; l = 1)$ [6]

U slučaju funkcije kovarijance $\texttt{exp.quad}(\mathbf{x}, \mathbf{x'}; l)$ skala l nosi informaciju o tome koliko moraju biti odmaknute dvije vrijednosti funkcije da bi one postale nekorelirane [6].

4.1 Regresija gausijanskih procesa - vektorska slika

Ovdje će se proučiti bajesijanska analiza modela regresije s gausijanskim šumom:

$$f(\mathbf{x}) = \phi(\mathbf{x})^T \mathbf{w} \qquad\qquad y = f(\mathbf{x}) + \varepsilon, \tag{4.8}$$

gdje $\phi(\mathbf{x})$ označava transformaciju ulaznog vektora $\mathbf{x}$ dimenzije d na prostor značajki dimenzije d_ϕ na kojem je problem linearan [6]. Dakako, pristranost odnosno odsječak na y-osi podrazumijeva se dodavanjem jedinice kao prve komponente vektora $\phi(\mathbf{x})$. Naravno, u slučaju izvorno linearnog problema transformacija je identitet. Skup dostupnih podataka oblika je $\{(\mathbf{x}_n, y_n) | n = 1, ..., N\}$, gdje je N broj primjera. Nadalje, pretpostavit će se da šum prati nezavisnu, identički raspodijeljenu Gaussovu raspodjelu srednje vrijednosti 0 i varijance σ_N^2:

$$\varepsilon \sim \mathcal{N}(0, \sigma_N^2) \tag{4.9}$$

Pretpostavka o takvom šumu sada dovodi do izraza za gustoću vjerojatnosti zapažanja ako su dani parametri spremljeni u vektoru $\mathbf{w}$:

$$p(\mathbf{y}|X, \mathbf{w}) = \prod_{n=1}^{N} p(y_n|\phi(\mathbf{x}_n), \mathbf{w}) = \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi}\sigma_N} \cdot \exp\left(-\frac{y_n - \phi(\mathbf{x}_n)^T \mathbf{w}}{2\sigma_N^2}\right) = \tag{4.10}$$

$$= \frac{1}{(2\pi\sigma_N^2)^{(N/2)}} \cdot \exp(-\frac{1}{2\sigma_N^2} \left|\mathbf{y} - \phi(X)^T\mathbf{w}\right|^2) = \mathcal{N}(\phi(X)^T\mathbf{w}, \sigma_N^2 I).$$

U bajesijanskom formalizmu potrebno je uvesti takozvani prior za parametre koji izražava znanje o parametrima prije pogleda na zapažanja. Ovdje će se za parametre postaviti gausijanski prior srednje vrijednosti nula s matricom kovarijance Σ_p:

$$\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \Sigma_p). \tag{4.11}$$

Zaključivanje u ovom formalizmu temelji se na takozvanoj posteriornoj raspodjeli parametara iz izraza (3.3) koji se ovdje može zapisati u jednostavnijem obliku:

$$p(\mathbf{w}|\mathbf{y}, X) = \frac{p(\mathbf{y}|X, \mathbf{w})p(\mathbf{w})}{p(\mathbf{y}|X)}, \tag{4.12}$$

gdje normalizirajuća konstanta kao i prije predstavlja graničnu vjerojatnost iz izraza (3.4) koja ne ovisi o parametrima te se također jednostavnije može zapisati kao:

$$p(\mathbf{y}|X) = \int p(\mathbf{y}|X, \mathbf{w})p(\mathbf{w})\mathrm{d}\mathbf{w}. \tag{4.13}$$

Sada se u svrhu predviđanja ispitnih primjera uprosječuje preko svih mogućih vrijednosti parametara s posteriornom vjerojatnošću kao težinom uprosječivanja. S druge strane, pristupi regresiji koji nisu bajesijanskog tipa određenim kriterijem odabiru jedan skup parametara. Raspodjela predviđanja $f_* \triangleq f(\mathbf{x}_*)$ u točki $\mathbf{x}_*$ tada je dana kao:

$$\begin{aligned} p(f_*|\mathbf{x}_*, X, \mathbf{y}) &= \int p(f_*|\mathbf{x}_*, \mathbf{w})p(\mathbf{w}|X, \mathbf{y})\mathrm{d}\mathbf{w} = \\ &= \mathcal{N}\left(\frac{1}{\sigma_N^2}\phi(\mathbf{x}_*)^T A^{-1}\Phi\mathbf{y}, \phi(\mathbf{x}_*)^T A^{-1}\phi(\mathbf{x}_*)\right), \end{aligned} \tag{4.14}$$

gdje je $\Phi = \Phi(X)$ matrica dimenzije $(d_\phi \times N)$ čiji su stupci vektori $\phi(\mathbf{x}_n)$, dok je druga novouvedena matrica $A = \sigma_N^{-2}\Phi\Phi^T + \Sigma_p^{-1}$. Sada se nakon nekoliko koraka uz dobro poznavanje linearne algebre jednažba iznad može napisati na sljedeći način:

$$\begin{aligned} f_*|\mathbf{x}_*, X, \mathbf{y} \sim \mathcal{N}\big(&\phi_*^T\Sigma_p\Phi(K + \sigma_N^2 I)^{-1}\mathbf{y}, \\ &\phi_*^T\Sigma_p\phi_* - \phi_*^T\Sigma_p\Phi(K + \sigma_N^2 I)^{-1}\Phi^T\Sigma_p\phi_*\big), \end{aligned} \tag{4.15}$$

gdje je uvedena oznaka $\phi(\mathbf{x}_*) = \phi_*$ i definirana matrica $K = \Phi^T \Sigma_p \Phi$. U funkcijskoj slici pristupa postat će jasno da prethodno uvedena matrica K predstavlja matricu kovarijance, dok se funkcija kovarijance može onda iščitati kao $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \Sigma_p \phi(\mathbf{x}')$.

4.2 Regresija gausijanskih procesa - funkcijska slika

Kada se neki proces modelira kao gausijanski u svrhu izvršenja regresije svojstvene jednom takvom procesu srednja vrijednost mu se pretpostavlja $\boldsymbol{\mu} = \mathbf{0}$ te mu se odabire primjerena funkcija kovarijance k(**x**,**x'**) koja ga treba opisati. Pretpostavka o trivijalnoj srednjoj vrijednosti pojednostavljuje daljnje izraze te proces regresije čini manje sklonim prenaučenosti. Kao i prije, za tip regresije koji se ovdje razmatra dostupni podatci modeliranog procesa su oblika $\{(\mathbf{x}_n, y_n) | n = 1, ..., N\}$, gdje se $y_n = f(\mathbf{x}_n) + \varepsilon$ smatra nasumično dobivenom vrijednošću gausijanskog procesa f u točki $\mathbf{x}_n$ [6]. Nadalje, pretpostavkom o aditivnom nezavisno raspodijeljenom gausijanskom šumu ε s varijancom σ_N^2 tada slijede izrazi ispod:

$$cov(y_p, y_q) = k(\mathbf{x}_p, \mathbf{x}_q) + \sigma_N^2 \delta_{pq} \qquad \text{ili} \qquad K_{\mathbf{y}} = K(X, X) + \sigma_N^2 I. \tag{4.16}$$

Multivarijantna Gaussova raspodjela srednjih vrijednosti mjerenja **y** u točkama X i funkcijskih vrijednosti $\mathbf{f}_*$ u interpolacijskim i ekstrapolacijskim točkama X_* tada ima oblik:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X, X) + \sigma_N^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix} \right). \tag{4.17}$$

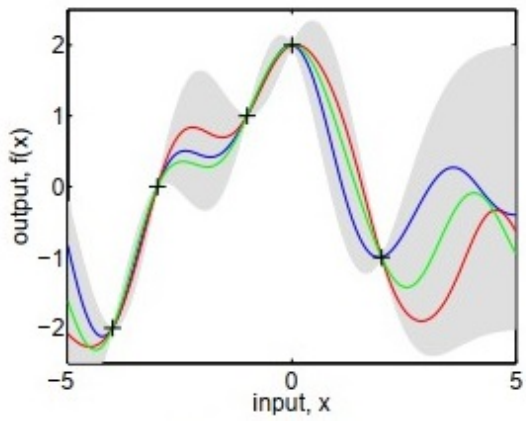
Izvodom uvjetne raspodjele iz jednadžbe iznad dolazi se do ključnih jednadžbi predviđanja za regresiju gausijanskih procesa:

$$\mathbf{f}_* | X, \mathbf{y}, X_* \sim \mathcal{N}(\bar{\mathbf{f}}_*, K_{\mathbf{f}_*}), \quad \text{gdje je} \tag{4.18}$$

$$\bar{\mathbf{f}}_* = K(X_*, X)[K(X, X) + \sigma_N^2 I]^{-1} \mathbf{y} \quad \text{te} \tag{4.19}$$

$$K_{\mathbf{f}_*} = K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma_N^2 I]^{-1} K(X, X_*). \tag{4.20}$$

Naravno, postupak za mjerenja **y** bez šuma dobije se postavljanjem $\sigma_N^2 = 0$ te se tada uz funkciju kovarijance `exp.quad`$(\mathbf{x}, \mathbf{x'}; l = 1)$ dobiju primjeri funkcija sa Slike 4.2 ispod.



Slika 4.2: Nasumične funkcije gausijanskog procesa srednje vrijednosti $\mu(\mathrm{x}) = 0$ i funkcije kovarijance $k(\mathrm{x}, \mathrm{x'}) = \texttt{exp.quad}(\mathrm{x}, \mathrm{x'}; l = 1)$ koje prolaze zadanim točkama [6]

Pri regresiji je cilj što bolje opisati mjerene podatke biranjem najboljih kandidata za vrijednosti parametara izabrane funkcije kovarijance. Prilagodba parametara funkcije kovarijance na mjerene podatke postiže se maksimizacijom već spomenute granične vjerojatnosti.

4.3 Granična vjerojatnost

Kao što je već spomenuto, kada se neki proces modelira kao gausijanski srednja vrijednost mu se postavlja $\boldsymbol{\mu} = \mathbf{0}$ te mu se odabire neka funkcija kovarijance $k(\mathbf{x},\mathbf{x'};\boldsymbol{\theta})$ s pripadajućim hiperparametrom modela $\boldsymbol{\theta}$. Cilj je izračunati graničnu vjerojatnost da tako postavljeni model opisuje skup za učenje. Pri računanju te vjerojatnosti treba uzeti u obzir sve funkcijske vektore s $\mathbf{x}_n$ koordinatama skupa za učenje koji se mogu izgenerirati pretpostavljenim modelom, $\mathbf{f}|X \sim \mathcal{N}(\mathbf{0}_N, K)$ [6]. Umnoškom vjerojatnosti za ostvarenje pojedinog funkcijskog vektora i vjerojatnosti da on opisuje proces od interesa te najavljenom sumacijom po svim vektorima modela, za graničnu vjerojatnost slijedi izraz:

$$p(\mathbf{y}|X) = \int p(\mathbf{y}|\mathbf{f}, X)p(\mathbf{f}|X)\mathrm{d}\mathbf{f}. \tag{4.21}$$

Izraz za prirodni logaritam granične vjerojatnosti da zadani model opisuje skup za učenje tada je uz $\mathbf{y}|\mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma_N^2 I)$ i prethodno uvedenu oznaku $K_\mathbf{y}$ jednak:

$$\ln[p(\mathbf{y}|X, \boldsymbol{\theta})] = -\frac{1}{2} \cdot \mathbf{y}^T K_\mathbf{y}^{-1}\mathbf{y} - \frac{1}{2} \cdot \ln|K_\mathbf{y}| - \frac{N}{2} \cdot \ln(2\pi). \tag{4.22}$$

Izraz (4.22) iznad cilj je optimizacijom hiperparametra $\boldsymbol{\theta}$ maksimizirati što bi značilo da se njegov negativni oblik nastoji minimizirati. Minimizacija se može provesti gradijentnim spustom te je u tu svrhu potrebno poznavati gradijent logaritma granične vjerojatnosti koji se računa uz pomoć izraza:

$$\frac{\partial}{\partial \theta_j} \ln[p(\mathbf{y}|X, \boldsymbol{\theta})] = \frac{1}{2} \cdot \mathbf{y}^T K_{\mathbf{y}}^{-1} \frac{\partial K_{\mathbf{y}}}{\partial \theta_j} K_{\mathbf{y}}^{-1} \mathbf{y} - \frac{1}{2} \cdot \mathrm{tr}\left(K_{\mathbf{y}}^{-1} \frac{\partial K_{\mathbf{y}}}{\partial \theta_j} \right). \tag{4.23}$$

4.4 Ilustracija ekstrakcije PDF-a

Jedna od metoda ekstrakcije PDF-a iz podataka diferencijalnog udarnog presjeka određenih procesa je modeliranje tih funkcija kao nezavisnih gausijanskih procesa. Poznavanjem teorijskih izraza ovisnosti diferencijalnih udarnih presjeka tih procesa o impulsnim raspodjelama u priči tada se isti također modeliraju kao gausijanski procesi čije značajke ovise o značajkama već modeliranih distribucijskih funkcija. Opisanom metodom regresije gausijanskih procesa moguće je tada napraviti prilagodbu nad mjerenim podatcima diferencijalnog udarnog presjeka za odabrane procese raspršenja na protonu. U nastavku ovog rada proučavat će se slučaj gdje su poznati podatci (n+1) procesa u kojima uz doprinose od gluona prevladavaju doprinosi n kvarkova. Naime, u višim redovima računa smetnje uz partonske distribucijske funkcije n kvarkova ističe se i jedna gluonska koja određuje udio impulsa protona kojeg nose svi gluoni. Prilagodba na podatke diferencijalnih udarnih presjeka tada vodi do sustava jednadžbi koji daje jedinstveno rješenje distribucijskih funkcija.

Za ilustraciju rada ove metode ekstrakcije mogu se uvesti određena pojednostavljenja. Prvo pojednostavljenje je zadržavanje na samo dvije distribucijske funkcije koje najviše doprinose u mjerenjima diferencijalnih udarnih presjeka dvaju proučavanih procesa. Drugo pojednostavljenje je smanjenje broja kinematičkih varijabli na jednu te će se ono zadržati kroz cijeli rad. Za potrebe ove ilustracije dva proučavana procesa raspršenja opisana su dakle pojednostavljenim izrazima za diferencijalni udarni presjek te su njihove ovisnosti o partonskim distribucijskim funkcijama zadane izrazima ispod:

$$\frac{d\sigma_1}{d\mathrm{x}} = h_1(\mathrm{x})[0] \cdot \mathtt{pdf}_1(\mathrm{x}) + h_1(\mathrm{x})[1] \cdot \mathtt{pdf}_2(\mathrm{x}), \tag{4.24}$$

$$\frac{d\sigma_2}{d\mathrm{x}} = h_2(\mathrm{x})[0] \cdot \mathtt{pdf}_1(\mathrm{x}) + h_2(\mathrm{x})[1] \cdot \mathtt{pdf}_2(\mathrm{x}), \tag{4.25}$$

gdje $h_1(\mathrm{x})$ i $h_2(\mathrm{x})$ predstavljaju teorijski određene amplitude tvrdog raspršenja (engl. *hard scattering amplitude*) za jedan i drugi proces, dok $\mathtt{pdf}_1(\mathrm{x})$ i $\mathtt{pdf}_2(\mathrm{x})$ distribucijske funkcije koje je cilj ekstrahirati.

Pri izračunu logaritma granične vjerojatnosti zahtijeva se poznavanje srednjih vrijednosti mjerenih podataka $\mathbf{y}$ i varijance šuma σ_N^2 te matrice kovarijance $K(X, X; \boldsymbol{\theta})$ kojom je cilj opisati zadani proces. U pojednostavljenom problemu ovog potpoglavlja diferencijalni udarni presjeci dvaju procesa raspršenja su različite transformacije dviju partonskih distribucijskih funkcija. Pri modeliranju tih funkcija kao nezavisnih gausijanskih procesa srednje vrijednosti $\boldsymbol{\mu} = \mathbf{0}$ potrebno je svakom pojedinom odabrati funkciju kovarijance koja će ga predstavljati. Diferencijalni udarni presjek za svaki od procesa raspršenja tada isto postaje gausijanski proces s funkcijom kovarijance koja ovisi o funkcijama kovarijance modeliranih distribucijskih funkcija. Naime, ako je odabrana funkcija kovarijance jedne modelirane funkcije $\mathtt{exp.quad}(\mathrm{x}, \mathrm{x'}; l_1)$ i druge $\mathtt{exp.quad}(\mathrm{x}, \mathrm{x'}; l_2)$ slijedi da je funkcija kovarijance za prvi proces raspršenja:

$$\begin{aligned} k_1(\mathrm{x}, \mathrm{x'}) = {} & h_1(\mathrm{x})[0] \cdot h_1(\mathrm{x'})[0] \cdot \mathtt{exp.quad}(\mathrm{x}, \mathrm{x'}; l_1) \\ & + h_1(\mathrm{x})[1] \cdot h_1(\mathrm{x'})[1] \cdot \mathtt{exp.quad}(\mathrm{x}, \mathrm{x'}; l_2), \end{aligned} \tag{4.26}$$

dok za drugi proces raspršenja ona ima oblik:

$$\begin{aligned} k_2(\mathrm{x}, \mathrm{x'}) = {} & h_2(\mathrm{x})[0] \cdot h_2(\mathrm{x'})[0] \cdot \mathtt{exp.quad}(\mathrm{x}, \mathrm{x'}; l_1) \\ & + h_2(\mathrm{x})[1] \cdot h_2(\mathrm{x'})[1] \cdot \mathtt{exp.quad}(\mathrm{x}, \mathrm{x'}; l_2). \end{aligned} \tag{4.27}$$

S obzirom na to da ovise o istim distribucijskim funkcijama, procesi raspršenja od interesa međusobno su zavisni te stoga postoji i funkcija kovarijance koja opisuje njihov odnos:

$$\begin{aligned} k_{12}(\mathrm{x}, \mathrm{x'}) = {} & h_1(\mathrm{x})[0] \cdot h_2(\mathrm{x'})[0] \cdot \mathtt{exp.quad}(\mathrm{x}, \mathrm{x'}; l_1) \\ & + h_1(\mathrm{x})[1] \cdot h_2(\mathrm{x'})[1] \cdot \mathtt{exp.quad}(\mathrm{x}, \mathrm{x'}; l_2), \end{aligned} \tag{4.28}$$

$$\begin{aligned} k_{21}(\mathrm{x}, \mathrm{x'}) = {} & h_2(\mathrm{x})[0] \cdot h_1(\mathrm{x'})[0] \cdot \mathtt{exp.quad}(\mathrm{x}, \mathrm{x'}; l_1) \\ & + h_2(\mathrm{x})[1] \cdot h_1(\mathrm{x'})[1] \cdot \mathtt{exp.quad}(\mathrm{x}, \mathrm{x'}; l_2), \end{aligned} \tag{4.29}$$

gdje je uzeta u obzir pretpostavljena nezavisnost distribucijskih funkcija kao gausijanskih procesa. Srednje vrijednosti mjerenih podataka prvog procesa $\mathbf{y}_1$ i drugog

procesa $\mathbf{y}_2$ mogu se zajedno staviti u matricu $\mathbf{y}$ koja predstavlja srednje vrijednosti svih mjerenih podataka:

$$\mathbf{y} = \begin{bmatrix} \mathbf{y}_1(\mathbf{x}_1) \\ ... \\ \mathbf{y}_1(\mathbf{x}_N) \\ \mathbf{y}_2(\mathbf{x}_1) \\ ... \\ \mathbf{y}_2(\mathbf{x}_N) \end{bmatrix}. \tag{4.30}$$

Pripadna matrica funkcije kovarijance tada ima sljedeći oblik:

$$K = \begin{bmatrix} k_1(\mathbf{x}_1, \mathbf{x}_1) & ... & k_1(\mathbf{x}_1, \mathbf{x}_N) & k_{12}(\mathbf{x}_1, \mathbf{x}_1) & ... & k_{12}(\mathbf{x}_1, \mathbf{x}_N) \\ .... & ... & ... & ... & ... & ... \\ k_1(\mathbf{x}_N, \mathbf{x}_1) & ... & k_1(\mathbf{x}_N, \mathbf{x}_N) & k_{12}(\mathbf{x}_N, \mathbf{x}_1) & ... & k_{12}(\mathbf{x}_N, \mathbf{x}_N) \\ k_{21}(\mathbf{x}_1, \mathbf{x}_1) & ... & k_{21}(\mathbf{x}_1, \mathbf{x}_N) & k_2(\mathbf{x}_1, \mathbf{x}_1) & ... & k_2(\mathbf{x}_1, \mathbf{x}_N) \\ ... & ... & ... & ... & ... & ... \\ k_{21}(\mathbf{x}_N, \mathbf{x}_1) & ... & k_{21}(\mathbf{x}_N, \mathbf{x}_N) & k_2(\mathbf{x}_N, \mathbf{x}_1) & ... & k_2(x_N, x_N) \end{bmatrix} \tag{4.31}$$

te ovisi o hiperparametrima modela l_1 i l_2, odnosno o vektoru $\boldsymbol{\theta}$ koji se postavlja na način:

$$\boldsymbol{\theta} = \begin{bmatrix} l_1 \\ l_2 \end{bmatrix}. \tag{4.32}$$

U ovom trenutku moguće je uz pomoć danih $\mathbf{y}$ i K izračunati logaritam granične vjerojatnosti iz izraza (4.22). Određenim algoritmom minimizacije njegove negativne vrijednosti tada se optimiziraju slobodni hiperparametri l_1 i l_2 zastupljeni u matrici K što vodi do najboljeg opisa podataka dvaju procesa raspršenja od interesa kojeg je moguće dobiti opisanom metodom regresije uz korištenje odabranih funkcija kovarijance. Algoritmom unakrsne provjere mogu se ocjeniti modeli s različitim funkcijama kovarijance te usporediti njihov učinak na krajnji rezultat.

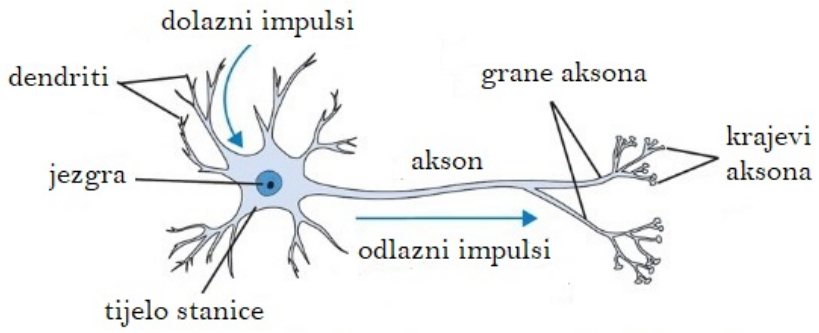
Opisana metoda ekstrakcije odiše relativnom jednostavnošću, no odabir funkcije kovarijance u cijelu metodu unosi određenu pristranost. Isti problem ekstrakcije može se pokušati riješiti Monte Carlo dropout metodom neuronskih mreža. Neuronske mreže metoda su strojnog učenja koja svojim velikim brojem parametara te svojom fleksibilnošću u postavkama može služiti u rješavanju raznih vrsta pro-

blema. S druge strane, fleksibilnost metode posljedica je njene kompleksnosti iz koje izviru brojni izazovi. Čak i pri rješavanju relativno jednostavnih problema neuronskim mrežama potrebno je dosta znanja za potpuno razumijevanje metode. Također, kao i u slučaju metode gausijanskih procesa, ni metoda neuronskih mreža nije otporna na pristranost koju unosi odabir njene arhitekture i brojnih hiperparametara. Više o neuronskim mrežama donosi sljedeće poglavlje.

5 Neuronske mreže

5.1 Povijesni uvod

Neuronske mreže posebna su grana strojnog učenja čije metode inspirirane biološkim mozgom nastoje naučiti prepoznati uzorke iz podataka s ciljem rješavanja širokog spektra problema. Živčana stanica ili neuron prikazana na Slici 5.1 osnovna je gradivna jedinica živčanog sustava u živim bićima. Neuron putem dendrita prima, potom obrađuje i na poslijetku prenosi podatke prema njemu susjednom neuronu putem aksona. Upravo je ta ideja o isprepletenoj mreži na isti način povezanih računalnih jedinica za obradu podataka ukomponirana u strojno učenje neuronskim mrežama.



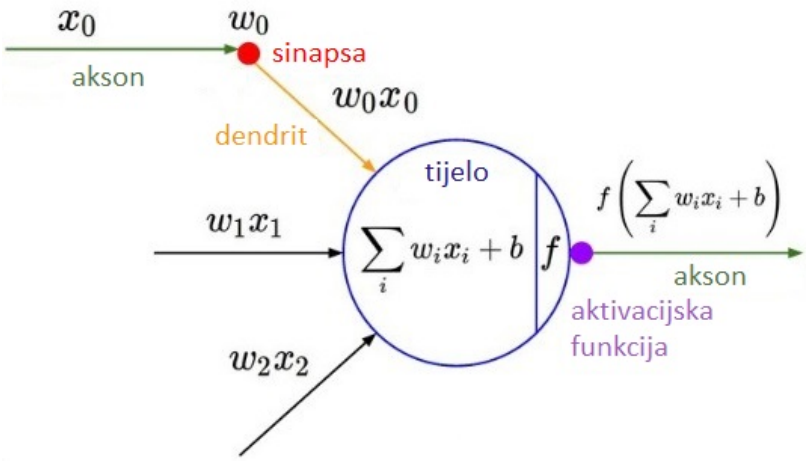
Slika 5.1: Živčana stanica [8]

Neki od prvih algoritama učenja imali su namjeru biti računalni modeli načina na koji uči mozak [3]. Iako se vrste neuronskih mreža koje se danas koriste za strojno učenje ponekad koriste i u razumijevanju funkcioniranja mozga, one općenito nisu dizajnirane da budu realistični modeli njegove biološke funkcije. S druge strane, danas se neuroznanost i dalje smatra važnim izvorom inspiracije za istraživanja u polju dobokog učenja, ali ne više i kao dominantan vodič njegovog daljnjeg razvoja.

5.2 Neuron kao računalna jedinica

Svaka računalna jedinica u neuronskoj mreži funkcionira na isti način prikazan na Slici 5.2. Kao ulazne podatke x_i prima obrađene izlazne podatke od prethodnih neurona u mreži množeći ih pritom težinom w_i (engl. *weight*). Potom se sve te vrijednosti zbroje te se tom iznosu dodaje pristranost b (engl. *bias*). Na kraju

se na dobivenu vrijednost primijeni odabrana takozvana aktivacijska funkcija (engl. *activation function*).



Slika 5.2: Računalna jedinica neuronske mreže [8]

Primjer najjednostavnijeg algoritma postignutog s neuronskim mrežama je takozvani perceptron (Rosenblatt 1958.) kojeg čini samo jedna računalna jedinica s funkcijom predznaka kao aktivacijskom funkcijom [4]. Naime, algoritam postavljen na prethodno opisani način uz prikladno odabranu funkciju gubitka koristi se za probleme binarne klasifikacije oblika $\mathbb{R}^d \rightarrow$\{-1,1\}. Skup za učenje oblika je S $= \{(\mathbf{x}_1, \mathrm{t}_1), ..., (\mathbf{x}_N, \mathrm{t}_N)\}$ te algoritam svoje predviđanje iskazuje kao y $= \mathtt{sig}(\langle \mathbf{w}, \mathbf{x} \rangle + b)$, gdje je `sig` spomenuta funkcija predznaka. Kako bi se sve skupa moglo lakše zapisati pristranost se može dodati u vektor težine na način $\mathbf{w} = (w_1, ..., w_d, b)^T$ te tada uz $\mathbf{x} = (x_1, ..., x_d, 1)^T$ predviđanje algoritma postaje y $= \mathtt{sig}(\langle \mathbf{w}, \mathbf{x} \rangle)$. Treba primijetiti da za točke koje su dobro klasificirane vrijedi $\langle \mathbf{w}, \mathbf{x}_n \rangle \mathrm{t}_n > 0$ dok za one pogrešno klasificirane vrijedi $\langle \mathbf{w}, \mathbf{x}_n \rangle \mathrm{t}_n) \leq 0$. U slučaju da u iteraciji i još uvijek postoji točka $\mathbf{x}_n$ koju algoritam pogrešno klasificira tada dakle vrijedi $\langle \mathbf{w}^{(i)}, \mathbf{x}_n \rangle \mathrm{t}_n) \leq 0$ te ispravak težinskog vektora u tom slučaju poprima oblik:

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}^{(i)}, \mathrm{S}) = \{\text{ako} : \langle \mathbf{w}^{(i)}, \mathbf{x}_n \rangle \mathrm{t}_n) \leq 0\} = \mathbf{w}^{(i)} + \mathrm{t}_n \mathbf{x}_n. \tag{5.1}$$

Utjecaj opisanog ispravka težinskog vektora na bolju klasifikaciju te točke može se lako vidjeti u sljedećoj jednadžbi:

$$\mathrm{t}_n \langle \mathbf{w}^{(i+1)}, \mathbf{x}_n \rangle = \mathrm{t}_n \langle \mathbf{w}^{(i)} + \mathrm{t}_n \mathbf{x}_n, \mathbf{x}_n \rangle = \mathrm{t}_n \langle \mathbf{w}^{(i)}, \mathbf{x}_n \rangle + \|\mathbf{x}_n\|^2, \tag{5.2}$$

gdje je uzeto u obzir da t_n pripada skupu \{-1,1\}. Ovaj jednostavan algoritam poka-

zuje moć samo jedne računalne jedinice te služi kao dobra motivacija za uvođenje cijele mreže istih. Shema rada cijelog algoritma zorno je prikazana na Slici 5.3 ispod.

Ulaz: skup za učenje $S = \{(\mathbf{x}_1, t_1), ..., (\mathbf{x}_N, t_N)\}$

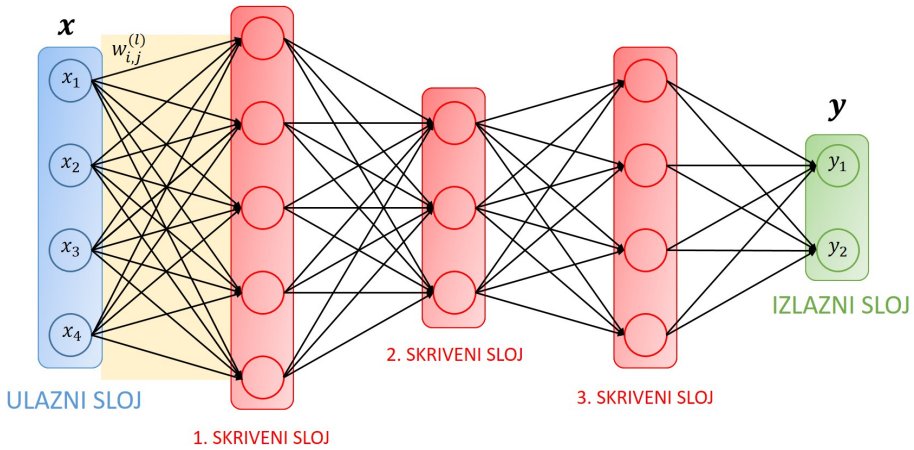
1. **inicijalizacija**: $\mathbf{w}^{(0)} = (0, ..., 0)$
2. for $i = 0, 1, 2, ...$
3. if ($\exists\, n$ takav da $\langle \mathbf{w}^{(i)}, \mathbf{x}_n \rangle t_n \leq 0$) then:
4. $\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + t_n \mathbf{x}_n$
5. **else**:
6. **Izlaz**: $\mathbf{w}^{(i)}$

Slika 5.3: Perceptron algoritam [4]

Neuronske mreže sastoje se od više slojeva prethodno opisanih računalnih jedinica. S obzirom na način povezanosti neurona i njihovu organizaciju u mreži razlikuju se primjerice aciklične neuronske mreže, mreže s povratnom vezom te lateralno povezane mreže. Neke od primjera acikličkih arhitektura mreža su potpuno povezane mreže, konvolucijske mreže, autoenkoder i generativne sukobljene mreže. Upravo potpuno povezana mreža glavni je fokus nastavka ovog poglavlja.

5.3 *Potpuno povezana aciklična neuronska mreža*

U slučaju potpuno povezane neuronske mreže svaki neuron iz jednog sloja povezan je sa svakim neuronom iz susjednih slojeva. Slika 5.4 ispod jasno dočarava izgled jedne takve mreže. Ulazni sloj naziva se onaj gdje se vrši unos podataka u mrežu na



Slika 5.4: Primjer potpuno povezane aciklične neuronske mreže s tri skrivena sloja

obradu. Svaki član $\mathbf{x}_n$ iz skupa za učenje $\mathrm{S} = \{(\mathbf{x}_1, \mathbf{t}_1), ..., (\mathbf{x}_N, \mathbf{t}_N)\}$ pojedinačno ulazi u mrežu te svaka njegova značajka odnosno komponenta ima sama svoj ulaz. Nakon ulaznog sloja slijedi odabrani broj takozvanih skrivenih slojeva gdje svaka računalna jedinica primljene podatke obrađuje i šalje računalnim jedinicama u idućem sloju na daljnju obradu na način opisan u prethodnom potpoglavlju. Na kraju mreže slijedi njeno konačno predviđanje za pojedinu točku $\mathbf{x}_n$ zbog čega broj računalnih jedinica u izlaznom sloju odgovara dimenziji ciljnih vrijednosti $\mathbf{t}_n$.

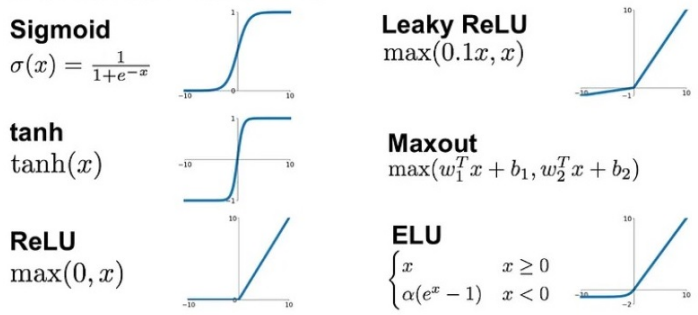
Radi matematičke formulacije neuronskih mreža uvode se neke oznake [7]. Težina $w_{i,j}^{(l)}$ je ona koja povezuje i-ti neuron l-tog sloja s j-tim neuronom $(l-1)$-tog sloja dok pristranost $b_i^{(l)}$ pripada i-tom neuronu l-tog sloja. Parametri l-tog sloja tada su matrica $\boldsymbol{W}^{(l)}$ i vektor $b^{(l)}$. Uvedena notacija omogućava zapis izlaza i-tog neurona u l-tom sloju na način:

$$\mathrm{o}_i^{(l)} = f\left(\mathrm{a}_i^{(l)} = \sum_j w_{i,j}^{(l)} \mathrm{o}_j^{(l-1)} + b_i^{(l)}\right) = f\left(\mathrm{a}_i^{(l)} = \left(\boldsymbol{W}^{(l)} \mathrm{o}^{(l-1)}\right)_i + b_i^{(l)}\right), \tag{5.3}$$

gdje je f odabrana aktivacijska funkcija. Izlaz svih neurona u l-tom sloju može se tada kompaktno zapisati kao:

$$\mathbf{o}^{(l)} = f\left(\mathbf{a}^{(l)} = \boldsymbol{W}^{(l)} \mathrm{o}^{(l-1)} + \boldsymbol{b}^{(l)}\right). \tag{5.4}$$

Pri rješavanju nekog problema neuronskim mrežama važno je međuostalim dobro odabrati broj slojeva, broj neurona u pojedinom sloju, aktivacijsku funkciju te konačno i broj epoha optimizacijskog algoritma. Neke od aktivacijskih funkcija koje se najčešće koriste prikazane su na Slici 5.5. Varijable poput prethodno nabrojanih hiperparametri su modela što jasno znači da se odabiru prije konačne optimizacije parametara odnosno težina mreže.



Slika 5.5: Aktivacijske funkcije [9]

Odabir gore nabrojanih hiperparametara uvelike odlučuje o kvaliteti ishoda procesa učenja, odnosno hoće li doći do podnaučenosti ili prenaučenosti. Jedna od metoda regularizacije neuronskih mreža za izbjegavanje prenaučenosti je upotreba takozvanog sloja isključivanja (engl. *dropout*) koji se može dodati nakon svakog standardnog gustog sloja neurona čime se u svakoj iteraciji optimizacijskog algoritma pri procesu učenja nasumično isključi odabrani postotak neurona u pojedinom sloju. Pri konačnom predviđanju mreže slojevi isključivanja obično se uklanjaju te se na kraju dobije jedinstveni rezultat. Ipak, ponekad korištenje spomenutih slojeva pri zaključivanju može dati više ili manje dobru procjenu neodređenosti samog modela jer se njihovom nasumičnošću u isključivanju neurona višestrukim zaključivanjem dobije raspodjela različitih rezultata [10]. Opisana metoda procjene neodređenosti aciklične gusto povezane neuronske mreže poznata je kao Monte Carlo dropout metoda.

5.4 Algoritam propagacije unatrag

Algoritmom propagacije unatrag naziva se postupak računanja gradijenta funkcije gubitka s obzirom na parametre neuronske mreže, sve njene težine i pristranosti [7]. Pravilo lančanog deriviranja ključno je u njegovom izvodu. Uzeći u obzir prethodno uvedenu notaciju, parcijalna derivacija funkcije gubitka po nekoj težini iz mreže ima oblik:

$$\frac{\partial \mathcal{L}}{\partial w_{i,j}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathsf{a}_i^{(l)}} \frac{\mathsf{a}_i^{(l)}}{\partial w_{i,j}^{(l)}} = \delta_i^{(l)} \mathsf{o}_j^{(l-1)}, \tag{5.5}$$

gdje je radi lakšeg daljnjeg računa uvedena nova oznaka:

$$\delta_i^{(l)} \triangleq \frac{\partial \mathcal{L}}{\partial \mathsf{a}_i^{(l)}}. \tag{5.6}$$

S druge strane, za parcijalnu derivaciju po pristranostima vrijedi:

$$\frac{\partial \mathcal{L}}{\partial b_i^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathsf{a}_i^{(l)}} \frac{\partial \mathsf{a}_i^{(l)}}{\partial b_i^{(l)}} = \delta_i^{(l)} \cdot 1. \tag{5.7}$$

Ponovnom primjenom pravila lančanog deriviranja slijedi:

$$\delta_i^{(l)} \triangleq \frac{\partial \mathcal{L}}{\partial \mathsf{a}_i^{(l)}} = \sum_k \frac{\partial \mathcal{L}}{\partial \mathsf{a}_k^{(l+1)}} \frac{\partial \mathsf{a}_k^{(l+1)}}{\partial \mathsf{a}_i^{(l)}} = \sum_k \delta_k^{(l+1)} \frac{\partial \mathsf{a}_k^{(l+1)}}{\partial \mathsf{a}_i^{(l)}}, \tag{5.8}$$

gdje se posljednja preostala parcijalna derivacija u izrazu izvrijedni na sljedeći način:

$$\frac{\partial \mathrm{a}_k^{(l+1)}}{\partial \mathrm{a}_i^{(l)}} = \frac{\partial \mathrm{a}_k^{(l+1)}}{\partial \mathrm{o}_i^{(l)}} \frac{\partial \mathrm{o}_i^{(l)}}{\partial \mathrm{a}_i^{(l)}} = w_{k,i}^{(l+1)} f'(\mathrm{a}_i^{(l)}). \tag{5.9}$$

Konačno, izraz (5.8) sada poprima oblik:

$$\delta_i^{(l)} = f'(\mathrm{a}_i^{(l)}) \sum_k \delta_k^{(l+1)} w_{k,i}^{(l+1)}. \tag{5.10}$$

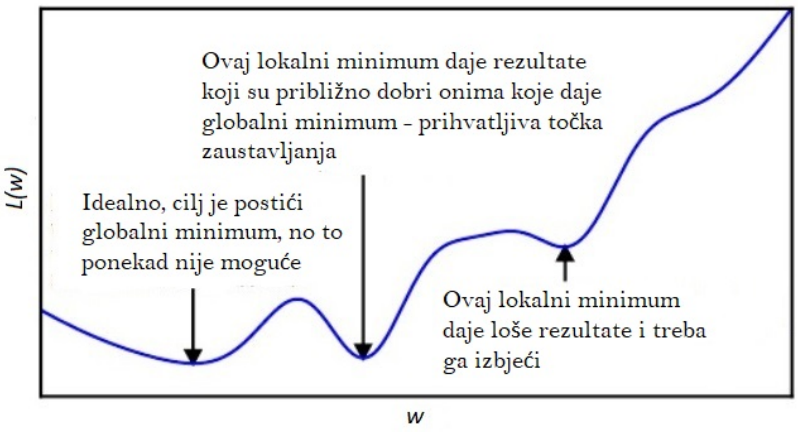
Korisno je još posebno istaknuti posljednji sloj L od kojeg cijeli algoritam propagacije unatrag počinje:

$$\delta_i^{(L)} = \frac{\partial \mathcal{L}}{\partial \mathrm{a}_i^L} = \frac{\partial \mathcal{L}}{\partial y_i} \frac{\partial y_i}{\partial \mathrm{a}_i^L} = \frac{\partial \mathcal{L}}{\partial y_i} g'(\mathrm{a}_i^{(L)}), \tag{5.11}$$

gdje je y_i i-ta komponenta predviđanja mreže te g aktivacijska funkcija izlaznih neurona. Na poslijetku, gradijenti funkcije gubitka s obzirom na težine $w_{i,j}^{(l)}$ i pristranosti $b_i^{(l)}$ u mreži dobivaju se uvrštavanjem izraza (5.10) u izraze (5.5) i (5.7) iznad.

5.5 Stohastički gradijentni spust

Osim već upoznatog standardnog gradijentnog spusta postoje napredniji algoritmi optimizacije parametara čija je upotreba neizbježna pri optimizaciji kompleksnijih modela strojnog učenja poput neuronskih mreža. Sofisticiraniji algoritmi minimizacije funkcije gubitka uvedeni su kako bi se izbjeglo zapinjanje algoritma u nezadovoljavajućim lokalnim minimumima, osigurala dobra konvergencija te skratilo vrijeme optimizacije.



Slika 5.6: Ilustracija odnosa različitih minimuma funkcije gubitka [3]

Zadovoljavajući lokalni minimum funkcije gubitka je prema Slici 5.6 na prethodnoj stranici onaj čija je vrijednost bliska vrijednosti globalnog minimuma. Cilj algoritma je izbjeći zapinjanje u lokalnim minimumima čija vrijednost puno odstupa od vrijednosti globalnog minimuma.

Prva varijacija običnog gradijentnog spusta koja se uvodi je stohastički gradijentni spust (engl. *Stochastic Gradient Descent,* SGD) shematski prikazan na Slici 5.7 ispod.

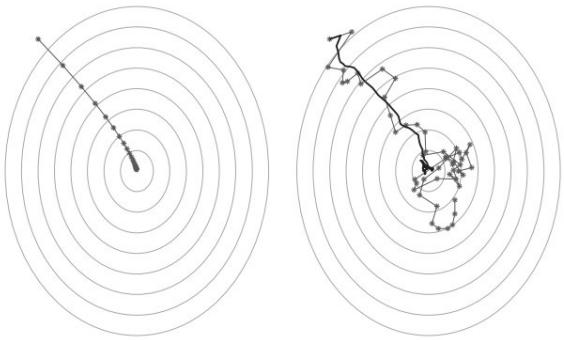
Ulaz: skup za učenje $\mathrm{S} = \{(\mathbf{x}_1, \mathbf{t}_1), ..., (\mathbf{x}_N, \mathbf{t}_N)\}$, diferencijabilna aditivna funkcija gubitka $\mathcal{L}(\mathrm{S}, \mathbf{w})$, stopa učenja $\eta^{(i)}$, početna parametarska točka $\mathbf{w}$

1. $i \leftarrow 0$
2. **while** uvjet zaustavljanja nije ispunjen **do**
3. Uzorkovanje skupa za učenje S:
   podskup od m elemenata $\{(\mathbf{x}^{(1)}, \mathbf{t}^{(1)}), ..., (\mathbf{x}^{(m)}, \mathbf{t}^{(m)})\}$
4. Računanje $\mathbf{g} \leftarrow \frac{1}{m} \sum_{k=1}^{m} \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}^{(k)}, \mathbf{t}^{(k)}; \mathbf{w})$
5. Ažuriranje $\mathbf{w} \leftarrow \mathbf{w} - \eta^{(i)} \mathbf{g}$
6. $i \leftarrow i + 1$
7. **end while**.

Slika 5.7: Stohastički gradijentni spust [3]

Za razliku od običnog gradijentnog spusta koji parametre ažurira uz pomoć cijelog skupa za učenje, stohastički gradijentni spust to čini uz pomoć nasumičnog uzorka skupa za učenje uzorkovanog u svakoj epohi iznova te se stoga primjenjuje pod uvjetom da je funkcija gubitka problema aditivna u sljedećem smislu:

$$\mathcal{L}(\mathrm{S}, \mathbf{w}^{(i)}) = \sum_{n=1}^{N} \mathcal{L}(\mathbf{x}_n, \mathbf{t}_n; \mathbf{w}^{(i)}). \tag{5.12}$$



Slika 5.8: Ilustracija usporedbe rada standardnog gradijentnog spusta (lijevo) i stohastičkog gradijentnog spusta (desno) [4]

Veličinu uzorka iz skupa za učenje važno je pažljivo odabrati jer njen izbor može utjecati na kvalitetu i brzinu konvergencije. Prednosti implementacije stohastičkog gradijentnog spusta u odnosu na standardni gradijentni spust je njegova nasumičnost ilustrirana na Slici 5.8 koja mu pomaže kod bijega iz nezadovoljavajućeg lokalnog minimuma te ga istovremeno čini manje sklonim dostizanju režima prenaučenosti.

Stohastički gradijentni spust popularan je izbor optimizacijskog algoritma, no učenje uz njega može biti sporo [3]. Metoda impulsa shematski prikazana na Slici 5.9 ispod konstruirana je stoga s ciljem da se ubrza proces optimizacije - posebno u uvjetima velike zakrivljenosti, malih ali dosljednih gradijenata ili gradijenata s prisutnim šumom. Algoritam s implementiranim impulsom inspiriran je drugim Newtonovim zakonom za jediničnu masu. Naime, za $\alpha = 1$ peti korak u shemi algoritma sa Slike 5.9 ispod postaje:

$$\mathbf{v}^{(i+1)} - \mathbf{v}^{(i)} = -\eta^{(i)} \frac{1}{m} \sum_{k=1}^{m} \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}^{(k)}, \mathbf{t}^{(k)}; \mathbf{w}^{(i)}), \tag{5.13}$$

što je u slučaju jedinične mase analogno diskretnoj verziji drugog Newtonovog zakona u sljedećem obliku:

$$\frac{d\mathbf{p}}{dt} = -\nabla E_p \tag{5.14}$$

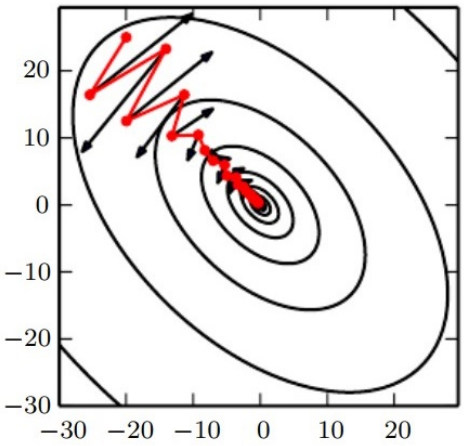
Ulaz: skup za učenje S $= \{(\mathbf{x}_1, \mathbf{t}_1), ..., (\mathbf{x}_N, \mathbf{t}_N)\}$, diferencijabilna aditivna funkcija gubitka $\mathcal{L}(\mathrm{S}, \mathbf{w})$, stopa učenja $\eta^{(i)}$, parametar impulsa α, početna parametarska točka $\mathbf{w}$, početna brzina $\mathbf{v}$

1. $i \leftarrow 0$
2. **while** uvjet zaustavljanja nije ispunjen **do**
3. Uzorkovanje skupa za učenje S:
   podskup od m elemenata $\{(\mathbf{x}^{(1)}, \mathbf{t}^{(1)}), ..., (\mathbf{x}^{(m)}, \mathbf{t}^{(m)})\}$
4. Računanje $\mathbf{g} \leftarrow \frac{1}{m} \sum_{k=1}^{m} \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}^{(k)}, \mathbf{t}^{(k)}; \mathbf{w})$
5. Ažuriranje $\mathbf{v} \leftarrow \alpha \mathbf{v} - \eta^{(i)} \mathbf{g}$
6. Ažuriranje $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{v}$
7. $i \leftarrow i + 1$
8. **end while**.

Slika 5.9: Stohastički gradijentni spust s implementiranim impulsom [3]

Hiperparametar $\alpha \in [0, 1)$ određuje koliko brzo će doprinosi prethodnih gradijenata

eksponencijalno padati pri učenju. Slika 5.10 ispod ilustrira prednost implementacije impulsa u algoritam. U danoj ilustraciji impuls vodi putanju po dužini kanjona funkcije gubitka dok se algoritam bez impulsa kreće naprijed-nazad po uskoj osi kanjona tako gubeći vrijeme.



Slika 5.10: Putanja po funkciji gubitka korištenjem SGD-a s implementiranim impulsom (crveno) - u svakom koraku po putu ucrtan je i smjer kojeg bi u tom trenutku odabrao standardni GD (crno) [3]

Osim implementacije standardnog impulsa ističe se i implementacija takozvanog Nesterovljevog impulsa shematski prikazana na Slici 5.11 ispod [3]. Jedina razlika među njima je parametarska točka u kojoj se izvrjednjuje gradijent - on se sada izvrje-

Ulaz: skup za učenje S $= \{(\mathbf{x}_1, \mathbf{t}_1), ..., (\mathbf{x}_N, \mathbf{t}_N)\}$, diferencijabilna aditivna funkcija gubitka $\mathcal{L}(\mathrm{S}, \mathbf{w})$, stopa učenja $\eta^{(i)}$, parametar impulsa α, početna parametarska točka $\mathbf{w}$, početna brzina $\mathbf{v}$

1. $i \leftarrow 0$
2. **while** uvjet zaustavljanja nije ispunjen **do**
3. Uzorkovanje skupa za učenje S:
   podskup od m elemenata $\{(\mathbf{x}^{(1)}, \mathbf{t}^{(1)}), ..., (\mathbf{x}^{(m)}, \mathbf{t}^{(m)})\}$
4. Privremeno ažuriranje $\widetilde{\mathbf{w}} \leftarrow \mathbf{w} + \alpha\mathbf{v}$
5. Računanje $\mathbf{g} \leftarrow \frac{1}{m}\sum_{k=1}^{m} \nabla_{\mathbf{w}}\mathcal{L}(\mathbf{x}^{(k)}, \mathbf{t}^{(k)}; \widetilde{\mathbf{w}})$
6. Ažuriranje $\mathbf{v} \leftarrow \alpha\mathbf{v} - \eta^{(i)}\mathbf{g}$
7. Ažuriranje $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{v}$
8. $i \leftarrow i + 1$
9. **end while**.

Slika 5.11: Stohastički gradijentni spust s implementiranim Nesterovljevim impulsom [3]

dnjuje nakon primijene trenutne brzine. Nesterovljev impuls može se stoga interpretirati kao pokušaj dodavanja korekcijskog člana standardnom impulsu. U slučaju standardnog gradijentnog spusta na konveksnoj funkciji gubitka, algoritam s implementiranim Nesterovljevim impulsom dovodi stopu konvergencije preostalog gubitka $\mathcal{L}(\mathbf{w}) - \min_{\mathbf{w}} \mathcal{L}(\mathbf{w})$ s prethodne $O(1/i)$ (nakon i epoha) na $O(1/i^2)$. Nažalost, s druge strane, u slučaju stohastičkog gradijentnog spusta Nesterovljev impuls ne poboljšava istu stopu konvergencije u usporedbi s njegovim ne korištenjem.

5.6 Napredni optimizacijski algoritmi

Dobro je poznato kako je stopa učenja jedan od hiperparametara kojeg je najteže odrediti zbog toga što njen odabir značajno utječe na izvedbu modela [3]. Dosad spomenuti algoritmi u osnovi imaju globalnu konstantnu stopu učenja iz epohe u epohu kojoj je moguće dodati komponentu promjenjivosti. Sloboda u tom pogledu ostavljena je oznakom stope učenja s indeksom epohe. S druge strane, napredniji algoritmi imaju integriranu promjenjivost stope učenja kroz epohe te mogu postići bolje rezultate prilagođavajući se uvjetima na koje algoritam nailazi. Štoviše, za raz-

Ulaz: skup za učenje $S = \{(\mathbf{x}_1, \mathbf{t}_1), ..., (\mathbf{x}_N, \mathbf{t}_N)\}$, diferencijabilna aditivna funkcija gubitka $\mathcal{L}(S, \mathbf{w})$, globalna stopa učenja η, početna parametarska točka $\mathbf{w}$, mala konstanta δ za numeričku stabilnost (možda 10^{-7})

1. $i \leftarrow 0$
2. inicijalizacija varijable nakupljenih gradijenata $\mathbf{r} = \mathbf{0}$
3. **while** uvjet zaustavljanja nije ispunjen **do**
4. Uzorkovanje skupa za učenje S:
   podskup od m elemenata $\{(\mathbf{x}^{(1)}, \mathbf{t}^{(1)}), ..., (\mathbf{x}^{(m)}, \mathbf{t}^{(m)})\}$
5. Računanje $\mathbf{g} \leftarrow \frac{1}{m} \sum_{k=1}^{m} \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}^{(k)}, \mathbf{t}^{(k)}; \mathbf{w})$
6. Nakupljanje kvadriranih gradijenata $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g}$
7. Računanje $\Delta\mathbf{w} \leftarrow -\frac{\eta}{\delta + \sqrt{\mathbf{r}}} \odot \mathbf{g}$
   (dijeljenje i korjenovanje primjenjuju se element po element)
8. Ažuriranje $\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$
9. $i \leftarrow i + 1$
10. **end while**.

Slika 5.12: AdaGrad algoritam [3]

liku od svih prethodno spomenutih algoritama takvi sofisticiraniji algoritmi prilagođavaju stopu učenja svakog pojedinog parametra zasebno.

Prvi od naprednijih algoritama optimizacije čiji će se način rada ovdje opisati je AdaGrad, shematski prikazan na Slici 5.12 iznad, koji prilagođava stopu učenja pojedinog parametra skalirajući ju obrnuto proporcionalno korijenom zbroja kvadrata svih prethodnih parcijalnih derivacija funkcije gubitka po tom parametru. Parametri s najvećim parcijalnim derivacijama funkcije gubitka prema tome imaju brzo opadajuću stopu učenja, dok onim parametrima s malim iznosima parcijalne derivacije stopa učenja relativno sporo opada. Ukupni učinak tada je putanja po parametarskom prostoru funkcije gubitka u smjerovima s blažim nagibom [3]. Ipak, nakupljanje svih kvadriranih gradijenata kroz cijeli proces učenja može dovesti do preranog i pretjeranog smanjenja stopa učenja.

Opisani nedostatak algoritma AdaGrad nastoji popraviti algoritam RMSprop shematski prikazan na Slici 5.13 ispod [3]. Naime, pri nakupljanju prethodnih gradijenata RMSprop s vremenom zaboravlja one od prije mnogo epoha na način da se

Ulaz: skup za učenje S $= \{(\mathbf{x}_1, \mathbf{t}_1), ..., (\mathbf{x}_N, \mathbf{t}_N)\}$, početna parametarska točka $\mathbf{w}$, diferencijabilna aditivna funkcija gubitka $\mathcal{L}(\mathrm{S}, \mathbf{w})$, globalna stopa učenja η, stopa opadanja ρ, mala konstanta δ za numeričku stabilnost (obično 10^{-6})

1. $i \leftarrow 0$
2. inicijalizacija varijable nakupljanja kvadriranih gradijenata $\mathbf{r} = \mathbf{0}$
3. **while** uvjet zaustavljanja nije ispunjen **do**
4. Uzorkovanje skupa za učenje S:
   podskup od m elemenata $\{(\mathbf{x}^{(1)}, \mathbf{t}^{(1)}), ..., (\mathbf{x}^{(m)}, \mathbf{t}^{(m)})\}$
5. Računanje $\mathbf{g} \leftarrow \frac{1}{m} \sum_{k=1}^{m} \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}^{(k)}, \mathbf{t}^{(k)}; \mathbf{w})$
6. Nakupljanje kvadriranih gradijenata $\mathbf{r} \leftarrow \rho\mathbf{r} + (1-\rho)\mathbf{g} \odot \mathbf{g}$
7. Računanje $\Delta\mathbf{w} \leftarrow -\frac{\eta}{\sqrt{\delta+\mathbf{r}}} \odot \mathbf{g}$
   ($\frac{1}{\sqrt{\delta+\mathbf{r}}}$ primjenjuje se element po element)
8. Ažuriranje $\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$
9. $i \leftarrow i + 1$
10. **end while**.

Slika 5.13: RMSprop algoritam [3]

nakupljanje gradijenata sada odvija na način:

$$\mathbf{r}^{(i)} = \rho\mathbf{r}^{(i)} + (1-\rho)\mathbf{g}^{(i)} \odot \mathbf{g}^{(i)}, \tag{5.15}$$

gdje je $\rho \in [0,1)$ takozvana stopa opadanja. Daljnjim razvijanjem veličine $\mathbf{r}^{(i-1)}$ u izrazu (5.15) jasno se vidi željeni učinak zaboravljanja starih gradijenata:

$$\begin{aligned} \mathbf{r}^{(i)} &= \rho\mathbf{r}^{(i-1)} + (1-\rho)\mathbf{g}^{(i)} \odot \mathbf{g}^{(i)} = \\ &= \rho^2\mathbf{r}^{(i-1)} + \rho(1-\rho)\mathbf{g}^{(i-1)} \odot \mathbf{g}^{(i-1)} + (1-\rho)\mathbf{g}^{(i)} \odot \mathbf{g}^{(i)} \\ &= \text{...i iteracija} = \sum_{k=1}^{i} \rho^{(i-k)}(1-\rho)\mathbf{g}^{(k)} \odot \mathbf{g}^{(k)}, \end{aligned} \tag{5.16}$$

gdje je uzeto u obzir da je $\mathbf{r}^{(0)} = \mathbf{0}$. Naime, krajnji rezultat izraza (5.16) iznad pokazuje da se kvadrati gradijenata novijih epoha množe s manjom potencijom stope opadanja $\rho \in [0,1)$, dok se oni starijih epoha množe s većom potencijom iste veličine. Ukupni učinak je postupno zaboravljanje starih gradijenata.

Posljednji algoritam koji će se ovdje proučiti je Adam shematski prikazan na Slici 5.14 ispod [3]. Za razliku od RMSprop koji nakuplja samo kvadrirane gradijente i postupno ih zaboravlja, Adam uz pomoć još jedne varijable nakuplja i same gradijente te ih također postupno zaboravlja na isti način kao kvadrirane. Osim toga, Adam ispravlja i pristranost koja je posljedica incijalizacije dviju varijabli koje mu služe za nakupljanje spomenutih veličina kao nul-vektora. Naime, utjecaj pristranosti na očekivanu vrijednost vektora $\mathbf{r}$ sa Slike 5.14 može se lako vidjeti pretpostavi li se da je gradijent niz epohe stacionaran proces odnosno da mu se u procesu učenja ne mijenjaju srednja vrijednost i varijanca.

$$\begin{aligned} E\left[\mathbf{r}^{(i)}\right] &= E\left[\sum_{k=1}^{i} \rho_2^{(i-k)}\left(1-\rho_2\right)\mathbf{g}^{(k)} \odot \mathbf{g}^{(k)}\right] = \\ &= E\left[\mathbf{g}^{(i)} \odot \mathbf{g}^{(i)}\right]\left(1-\rho_2\right)\sum_{k=1}^{i} \rho_2^{(i-k)} + \zeta \\ &= E\left[\mathbf{g}^{(i)} \odot \mathbf{g}^{(i)}\right]\rho_2^{i}\left(1-\rho_2\right)\sum_{k=1}^{i} \frac{1}{\rho_2^{k}} + \zeta \\ &= \ldots = E\left[\mathbf{g}^{(i)} \odot \mathbf{g}^{(i)}\right]\left(1-\rho_2^{i}\right) + \zeta, \end{aligned} \tag{5.17}$$

gdje je greška $\zeta = 0$ ako je pretpostavka o stacionarnosti gradijenta valjana [11]. Popravke pristranosti sa Slike 5.14 su sada opravdane.

Ulaz: skup za učenje S $= \{(\mathbf{x}_1, \mathbf{t}_1), ..., (\mathbf{x}_N, \mathbf{t}_N)\}$, početna parametarska točka $\mathbf{w}$, diferencijabilna aditivna funkcija gubitka $\mathcal{L}(\mathrm{S}, \mathbf{w})$, globalna stopa učenja η (predloženi iznos: 0.001), stope opadanja ρ_1,$\rho_2 \in [0, 1)$ (predloženi iznosi: 0.9 i 0.999 respektivno), mala konstanta δ za numeričku stabilnost (predloženi iznos: 10^{-8})

1. $i \leftarrow 0$
2. inicijalizacija varijable nakupljanja gradijenata $\mathbf{s} = \mathbf{0}$
3. inicijalizacija varijable nakupljanja kvadriranih gradijenata $\mathbf{r} = \mathbf{0}$
4. **while** uvjet zaustavljanja nije ispunjen **do**
5. Uzorkovanje skupa za učenje S:
   podskup od m elemenata $\{(\mathbf{x}^{(1)}, \mathbf{t}^{(1)}), ..., (\mathbf{x}^{(m)}, \mathbf{t}^{(m)})\}$
6. Računanje $\mathbf{g} \leftarrow \frac{1}{m} \sum_{k=1}^{m} \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}^{(k)}, \mathbf{t}^{(k)}; \mathbf{w})$
7. $i \leftarrow i + 1$
8. Nakupljanje gradijenata $\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1 - \rho_1)\mathbf{g}$
9. Nakupljanje kvadriranih gradijenata $\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1 - \rho_2)\mathbf{g} \odot \mathbf{g}$
10. Popravka pristranosti $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1-\rho_1^i}$
11. Popravka pristranosti $\hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1-\rho_2^i}$
12. Računanje $\Delta\mathbf{w} \leftarrow -\eta \frac{\hat{\mathbf{s}}}{\delta + \sqrt{\hat{\mathbf{r}}}}$
    (operacije se primjenjuju element po element)
13. Ažuriranje $\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$
14. **end while**.

Slika 5.14: Adam algoritam [3]

Postoje još neki popularni algoritmi optimizacije poput RMSprop s implementiranim impulsom i AdaDelta [3]. Nažalost, ipak se trenutno nijedan algoritam optimizacije nije istaknuo kao najbolji te njegov izbor stoga uvelike ovisi o korisnikovom poznavanju pojedinog.

5.7 Funkcija gubitka pri ekstrakciji PDF-a

Za ilustraciju izgleda funkcije gubitka koja će se koristiti pri ekstrakciji PDF-a neuronskom mrežom zadržava se pojednostavljenje uvedeno u četvrtom poglavlju gdje u dva proučavana procesa prevladavaju doprinosi samo dvije distribucijske funkcije.

Također, diferencijalni udarni presjeci i dalje se pojednostavljuju na način da ovise o jednoj kinematičkoj varijabli. Na temelju izraza (4.24) i (4.25) se uz oznaku:

$$\left(\frac{d\sigma_i}{d\mathrm{x}}\right)_{prediction} = h_i(\mathrm{x})[0] \cdot \mathtt{pdf}_{1,prediction}(\mathrm{x}) + h_i(\mathrm{x})[1] \cdot \mathtt{pdf}_{2,prediction}(\mathrm{x}) \tag{5.18}$$

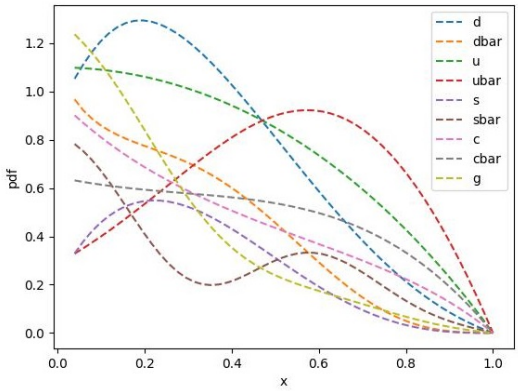
odabire funkcija gubitka oblika:

$$\mathcal{L} = \frac{1}{N_1 + N_2} \sum_{i=1}^{2} \sum_{n=1}^{N_i} \left[\frac{\left(\frac{d\sigma_i}{d\mathrm{x}}\right)_{prediction} \Big|_{\mathrm{x}=\mathrm{x}_n} - \left(\frac{d\sigma_i}{d\mathrm{x}}\right)_{data_mean} \Big|_{\mathrm{x}=\mathrm{x}_n}}{\left(\frac{d\sigma_i}{d\mathrm{x}}\right)_{data_error} \Big|_{\mathrm{x}=\mathrm{x}_n}} \right]^2, \tag{5.19}$$

gdje prva sumacija po indeksu i uzima u obzir oba procesa, dok ona po indeksu n prelazi preko skupa za učenje svakog pojedinog procesa.

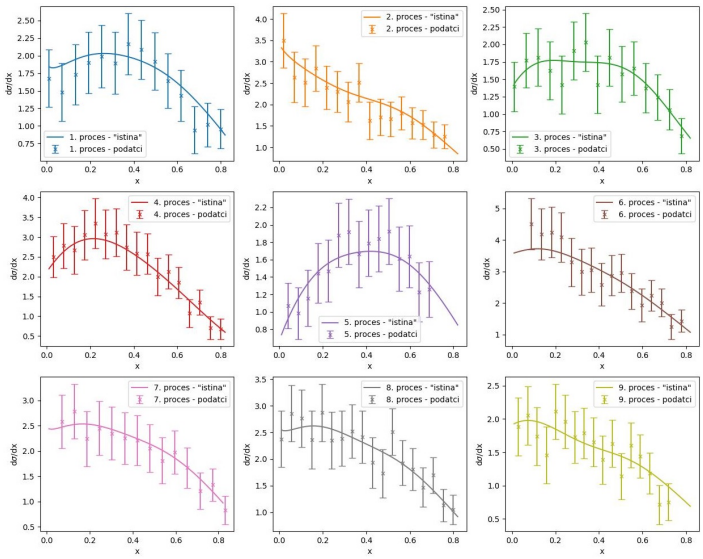
6 Ekstrakcija PDF-a

S ciljem ispitivanja metode ekstrakcije gausijanskim procesima i one neuronskom mrežom, zadano je devet nefizikalnih partonskih distribucijskih funkcija za pokušaj njihove replikacije. Funkcije su zadane kao razni umnošci polinoma do najviše drugog stupnja, trigonometrijskih funkcija, hiperboličnih funkcija te eksponencijalnih funkcija. Njihov se vizualni oblik može vidjeti na Slici 6.1 ispod.



Slika 6.1: Nefizikalne partonske distribucijske funkcije zadane za replikaciju

Za potrebe ekstrakcije distribucijskih funkcija zadano je devet nefizikalnih procesa čije su amplitude tvrdog raspršenja u izrazima za diferencijalne udarne presjeke umnošci i količnici različitih polinoma do najviše trećeg stupnja. Simulirani podatci diferencijalnih udarnih presjeka svih devet procesa prikazani su na Slici 6.2 ispod uz uvedeno nefizikalno pojednostavljenje ovisnosti o samo jednoj kinematičkoj varijabli.



Slika 6.2: Simulirani podatci diferencijalnih udarnih presjeka nefizikalnih procesa s intervalom greške $\pm 1.96\sigma$

Relevantni interval $\pm 1.96\sigma$ predstavlja interval pouzdanosti od 95%, što znači da se s vjerojatnošću od 95% očekuje da prava vrijednost funkcije leži unutar tog raspona.

6.1 Metoda gausijanskih procesa

Ekstrakcija partonskih distribucijskih funkcija metodom gausijanskih procesa obavljena je u programskom jeziku Python uz pomoć `lsqfitgp` paketa koji nudi razne mogućnosti pri modeliranju gausijanskih procesa te cijelu metodu čini relativno jednostavno primjenjivom [12]. Partonske distribucijske funkcije modelirane su kao nezavisni gausijanski procesi široko primjenjive funkcije kovarijance $\texttt{exp.quad}(\mathrm{x}, \mathrm{x'}; l_m)$ i trivijalne srednje vrijednosti $\mu(\mathrm{x}) = 0$ kao što je opisano pri ilustraciji ekstrakcije u četvrtom poglavlju. Izrazi za diferencijalne udarne presjeke svih procesa kompaktno se mogu zapisati na sljedeći način:

$$\left(\frac{d\sigma_i}{d\mathrm{x}}\right) = \sum_{m=1}^{9} h_i(\mathrm{x})[m-1] \cdot \texttt{pdf}_m(\mathrm{x}), \tag{6.1}$$

gdje $h_i(\mathrm{x})[m-1]$ označava amplitudu tvrdog raspršenja i-tog procesa i m-tog okusa koja se množi s pripadnom partonskom distribucijskom funkcijom. Kao i prije, diferencijalni udarni presjeci oblika modelirani su tada isto kao gausijanski procesi srednje vrijednosti $\mu(\mathrm{x}) = 0$ čije funkcije kovarijance ovise o funkcijama kovarijance već modeliranih distribucijskih funkcija na sljedeći način:

$$k_i(\mathrm{x}, \mathrm{x}') = \sum_{m=1}^{9} \left\{ h_i(\mathrm{x})[m-1] \cdot h_i(\mathrm{x'})[m-1] \cdot \texttt{exp.quad}(\mathrm{x}, \mathrm{x'}; l_m) \right\}. \tag{6.2}$$

Kao što je spomenuto i u ilustraciji četvrtog poglavlja, zadani procesi raspršenja međusobno su zavisni s obzirom na to da ovise o istim distribucijskim funkcijama te stoga treba uzeti u obzir i funkciju kovarijance koja opisuje njihov odnos:

$$k_{ij}(\mathrm{x}, \mathrm{x'}) = \sum_{m=1}^{9} h_i(\mathrm{x})[m-1] \cdot h_j(\mathrm{x'})[m-1] \cdot \texttt{exp.quad}(\mathrm{x}, \mathrm{x'}; l_m). \tag{6.3}$$

Sada se na isti način opisan u četvrtom poglavlju slaže matrica kovarijance K koja ovisi o hiperparametrima modela $l_1, ..., l_9$, odnosno o hiperparametarskom vektoru $\boldsymbol{\theta}$

koji ima sljedeći oblik:

$$\boldsymbol{\theta} = \begin{bmatrix} l_1 \\ ... \\ l_9 \end{bmatrix}. \tag{6.4}$$

Maskimizacijom logaritma granične vjerojatnosti iz izraza (4.22) uz pomoć gradijenta izračunatog izrazom (4.23) `lsqfitgp` paket je nakon obavljene optimizacije hiperparametarskog vektora $\boldsymbol{\theta}$ u mogućnosti napraviti prilagodbu na podatke diferencijalnih udarnih presjeka svih procesa te istovremeno obaviti ekstrakciju distribucijskih funkcija.

6.2 Metoda neuronskih mreža

Ekstrakcija partonskih distribucijskih funkcija metodom neuronskih mreža obavljena je također u programskom jeziku Python, no uz pomoć `tensorflow` paketa koji nudi relativno jednostavnu implementaciju svih komponenti metode opisanih u petom poglavlju [13]. Funkcija gubitka pri optimizaciji težina i pristranosti mreže zadana je izrazom:

$$\mathcal{L} = \frac{1}{\sum_{i=1}^{9} N_i} \sum_{i=1}^{9} \sum_{n=1}^{N_i} \left[\frac{\left(\frac{d\sigma_i}{d\mathrm{x}}\right)_{prediction} \Big|_{\mathrm{x}=\mathrm{x}_n} - \left(\frac{d\sigma_i}{d\mathrm{x}}\right)_{data_mean} \Big|_{\mathrm{x}=\mathrm{x}_n}}{\left(\frac{d\sigma_i}{d\mathrm{x}}\right)_{data_error} \Big|_{\mathrm{x}=\mathrm{x}_n}} \right]^2, \tag{6.5}$$

gdje prva suma po indeksu i uzima u obzir sve procese, dok druga suma po indeksu n prelazi preko skupa za učenje pojedinog procesa te se za predviđanje diferencijalnih udarnih presjeka u brojniku podrazumijeva korištenje izraza:

$$\left(\frac{d\sigma_i}{d\mathrm{x}}\right)_{prediction} = \sum_{m=1}^{9} h_i(\mathrm{x})[m-1] \cdot \mathtt{pdf}_{m,prediction}(\mathrm{x}). \tag{6.6}$$

Neuronska mreža postavljena je na način da se u izlaznom sloju dobiju predviđanja $\mathtt{pdf}_{m,prediction}(\mathrm{x})$ svih distribucijskih funkcija. Algoritam propagacije unatrag sada počinje derivacijom funkcije gubitka po predviđanju distribucijskih funkcija. Kao optimizacijski algoritam odabran je Adam s preporučenim iznosom globalne stope učenja 0.001. Provedena je Monte Carlo dropout metoda pri kojoj slojevi isključivanja neurona imaju svoju ulogu i pri predviđanju.

6.3 Unakrsna provjera

Prije same ekstrakcije su u svrhu odabira što prikladnijih hiperparametara neuronske mreže s dva skrivena sloja obavljene unakrsne provjere 36 različitih postavki mreže dobivene sljedećim kombinacijama hiperparametara:

broj neurona u 1. skrivenom sloju: [16,32,64],
broj neurona u 2. skrivenom sloju: [16,32,64],
udio neurona koji se isključuje: [0.1,0.2],
aktivacijska funkcija: [ELU, ReLU],

među kojima je odabrana ona kombinacija s najboljim rezultatom odnosno s najmanjom procjenom stvarne pogreške modela. Također, učinjena je i unakrsna provjera metode gausijanskih procesa u svrhu usporedbe. Pri unakrsnim provjerama koristila se funkcija gubitka iz izraza (6.5) te se izvršila podjela simuliranih podataka na pet podskupova podjednakih veličina. Treba napomenuti da obje ispitivane metode kao krajnji rezultat daju funkcijsku raspodjelu predviđanja pojedinih distribucijskih funkcija kao i funkcijsku raspodjelu predviđanja diferencijalnog udarnog presjeka pojedinih procesa. Metoda gausijanskih procesa načinom na koji je postavljena sve to čini povezano. S druge strane, Monte Carlo dropout metoda neuronskih mreža s funkcijom gubitka postavljenom na već opisani način daje samo raspodjelu predviđanja za svaku pojedinu distribucijsku funkciju te se potom raspodjela predviđanja diferencijalnog udarnog presjeka svakog pojedinog procesa dobije uz pomoć izraza (6.6). U brojniku izraza (6.5) pri unakrsnoj provjeri pojedinog modela kao predviđanje je stoga zapravo uzeta srednja vrijednost raspodjele svih predviđanja za pojedini proces. Postavke neuronske mreže s najboljim rezultatom unakrsne provjere su:

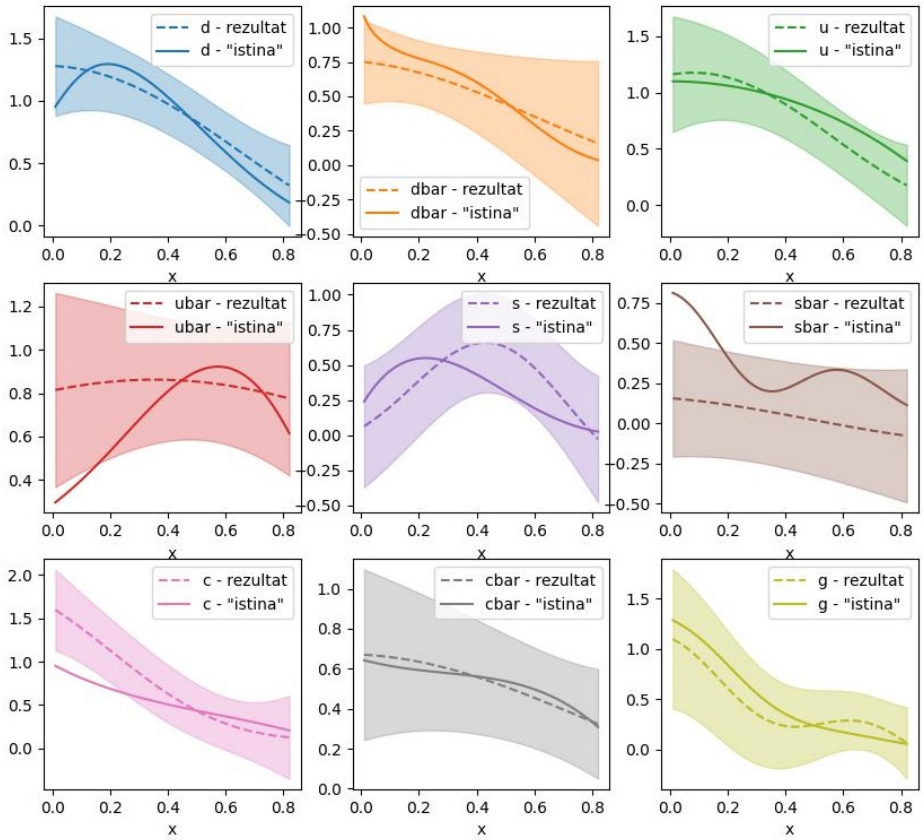
broj neurona u 1. skrivenom sloju: 16,
broj neurona u 2. skrivenom sloju: 16,
udio neurona koji se isključuje: 0.1,
aktivacijska funkcija: ELU,

Rezultat unakrsne provjere neuronske mreže s prethodnim postavkama iznosi otprilike 1.254, dok za usporedbu rezultat metode gausijanskih procesa iznosi otprilike 1.195. Rezultat bliže jedinici trebao bi značiti bolju generalizaciju na neviđene podatke diferencijalnih udarnih presjeka, no takav zaključak ne treba uzeti strogo s ob-

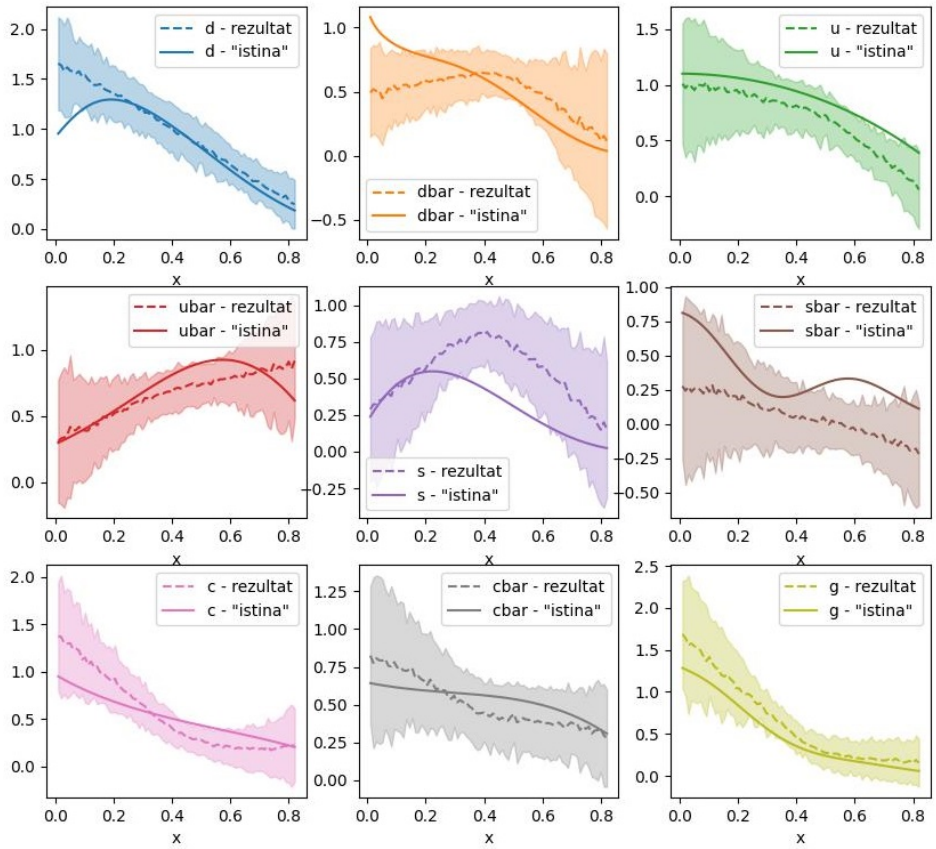
zirom na to da unakrsna provjera s podjelom na pet podskupova podjednakih veličina daje samo procjenu kvalitete modela.

6.4 Rezultati

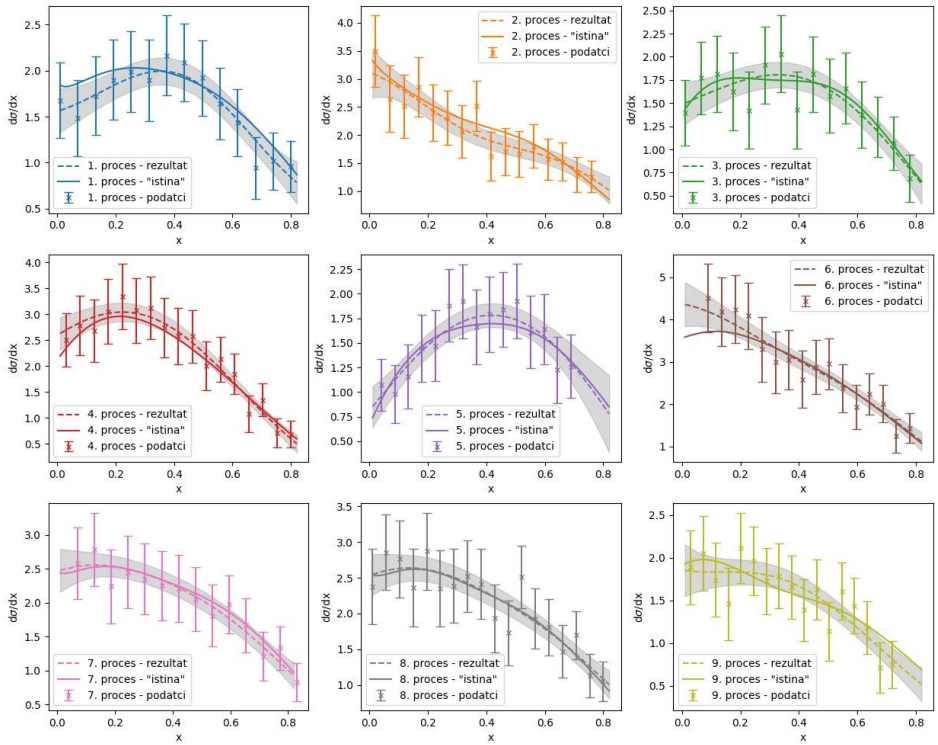
Rezultati ekstrakcije distribucijskih funkcija metodom gausijanskih procesa prikazani su na Slici 6.3, dok se oni dobiveni Monte Carlo dropout metodom neuronskih mreža mogu vidjeti na Slici 6.4 ispod. Naravno, početno zadane distribucijske funkcije daju osjećaj za kvalitetu ekstrakcije budući da je krajnji cilj bio njihova replikacija. Na slikama je vidljivo da su neke distribucijske funkcije (poput sbar) lošije ekstrahirane dok su neke malo bolje (poput d). Uzrok opisanog problema daje usporedba rezultata dvaju metoda koja ukazuje na to da je odgovor očito njihova podzauspljenost u smislu doprinosa zadanim diferencijalnim udarnim presjecima. Ipak, neke distribucijske funkcije (poput ubar i g) bolje su ekstrahirane metodom neuronskih mreža što znači da njihova funkcija kovarijance u metodi gausijanskih procesa može biti prikladnije odabrana za potrebe njihovog boljeg opisa.



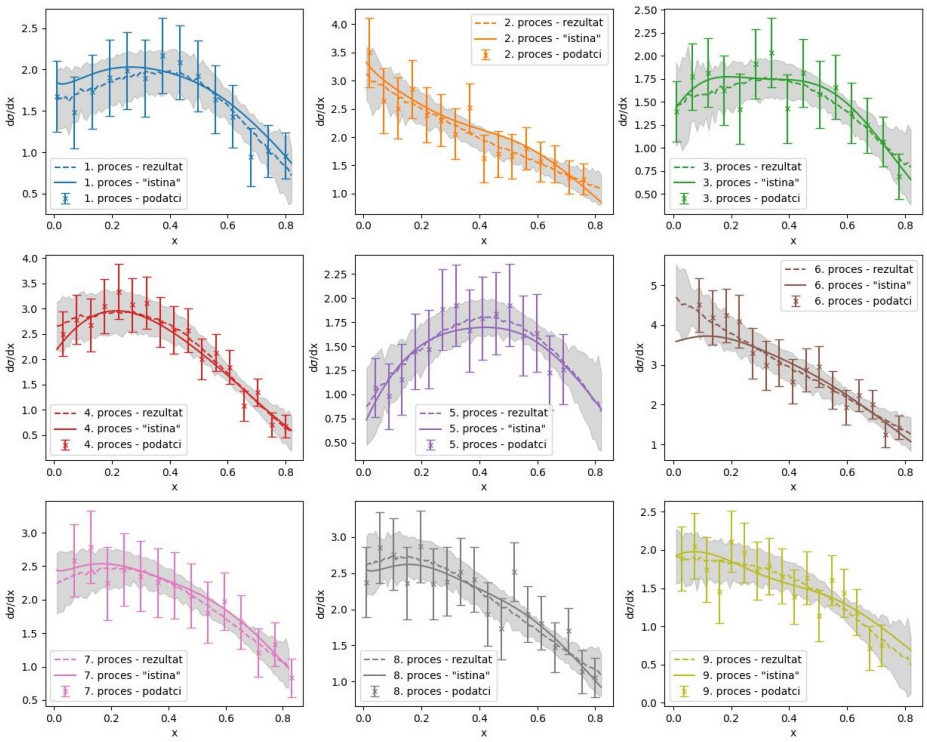
Slika 6.3: Rezultati replikacije partonskih distribucijskih funkcija metodom gausijanskih procesa s pojasom greške $\pm 1.96\sigma$



Slika 6.4: Rezultati replikacije partonskih distribucijskih funkcija metodom neuronske mreže s pojasom greške $\pm 1.96\sigma$



Slika 6.5: Rezultati prilagodbe na podatke diferencijalnih udarnih presjeka metodom gausijanskih procesa s pojasom greške $\pm 1.96\sigma$



Slika 6.6: Rezultati prilagodbe na podatke diferencijalnih udarnih presjeka metodom neuronske mreže s pojasom greške $\pm 1.96\sigma$

S druge strane, prilagodbe na simulirane podatke diferencijalnih udarnih presjeka na Slici 6.5 i na Slici 6.6 daju odgovor na pitanje jeli proces učenja u nekoj od metoda slučajno završio u režimu podnaučenosti ili prenaučenosti. Kao što je vidljivo, prilagodbe na simulirane podatke diferencijalnih udarnih presjeka uglavnom dobro prate svoj izvorno zadani oblik s ponekim manjim znakovima prenaučenosti.

Za kraj je potrebno naglasiti da neodređenosti dobivene gausijanskim procesima prate aproksimaciju bajesijanske statistike gdje se umjesto računanja posteriorne raspodjele hiperparametara maksimizira više puta spomenuta granična vjerojatnost. S druge strane, Monte Carlo dropout metoda neuronskih mreža osmišljena je s istim ciljem, no dobivene neodređenosti ovise o izboru hiperparametara koji se biraju zasebno i time fiksiraju model [10]. Raznolikost funkcija dobivenih tom metodom očekuje se stoga da će biti mala. Dakle, neodređenosti rezultata u metodi gausijanskih procesa pouzdanije su utvrđene.

7 Zaključak

U ovom radu ispitana je metoda gausijanskih procesa kao potencijalno dobrostojeći kandidat za ekstrakciju partonskih distribucijskih funkcija protona iz eksperimentalnih podataka diferencijalnih udarnih presjeka više različitih procesa raspršenja. Također, za potrebe usporedbe ispitana je i Monte Carlo dropout metoda neuronskih mreža. U svrhu samog ispitivanja metode uvedena su određena nefizikalna pojednostavljenja. S istim ciljem zadano je devet nefizikalnih partonskih distribucijskih funkcija koje je onda uz pomoć simuliranih podataka diferencijalnih udarnih presjeka devet različitih nefizikalnih procesa raspršenja bilo potrebno što bolje replicirati.

U metodi gausijanskih procesa partonske distribucijske funkcije modelirane su kao nezavisni gausijanski procesi srednje vrijednosti $\mu(\mathrm{x}) = 0$ i funkcije kovarijance $\mathtt{exp.quad}(\mathrm{x}, \mathrm{x}'; l_m)$. Bolja ekstrakcija metodom neuronskih mreža ponekih partonskih distribucijskih funkcija daje naslutiti da bi odabir funkcija kovarijance koje bolje opisuju pojedine distribucijske funkcije u metodi gausijanskih procesa rezultirao njihovom preciznijom ekstrakcijom odnosno replikacijom. Ipak, manje kvalitetna ekstrakcija ponekih distribucijskih funkcija u obje metode ukazuje na podzastupljenost tih funkcija u smislu doprinosa diferencijalnim udarnim presjecima zadanih procesa.

Metoda gausijanskih procesa daje obećavajuće mogućnosti primjene na stvarne eksperimentalne podatke uz poneke komplikacije izbjegnute u ovom radu koje bi tada trebalo uvesti i uz pomniji odabir funkcija kovarijance koje opisuju pojedine partonske distribucijske funkcije. Također, u usporedbi s Monte Carlo dropout metodom neuronskih mreža neodređenosti dobivene metodom gausijanskih procesa pouzdanije su utvrđene.

Dodaci

Ovdje su priložene dvije Python skripte korištene za simulaciju podataka i dobivanje rezultata. Obje skripte koriste pogodnosti platforme *Google Colab* koja nudi jednostavan pristup njihovom pokretanju i replikaciji njihovih rezultata bez ikakve potrebe za dodatnim softverom [14]. Dodatak A predstavlja skriptu s implementiranom metodom gausijanskih procesa, dok Dodatak B predstavlja onu s implementiranom Monte Carlo dropout metodom neuronskih mreža.

Dodatak A gaussian.ipynb

```
from google.colab import drive
drive.mount('/content/drive')
import pickle
import os

import numpy as np
from functools import partial
from math import ceil
import matplotlib.pyplot as plt
import lsqfitgp as lgp
import gvar

from sklearn.model_selection import KFold
import itertools

import sys
import jax
import jaxlib
import scipy

print(sys. version) #odrađeno na 3.10.12
```

```
print(lgp.__version__) #odrađeno na 0.20.2
print(gvar.__version__) #odrađeno na 13.1.1
print(jax.__version__) #odrađeno na na 0.4.23
print(jaxlib.__version__) #odrađeno na 0.4.23
print(scipy.__version__) #odrađeno na 1.12.0

##PARTONSKE DISTRIBUCIJSKE FUNKCIJE ZA REPLIKACIJU:##

def pdf_truth(x):
  """PDFs."""

  pdfs = (1.1 * (1 - x) * (1 + 0.6 * np.tanh(6 * x))
          * np.exp(-2 * (x - 0.3)**2),

          1.2 * (1 - x**0.5) * (1 + x * np.sin(5 * x)),

          1.1 * np.cosh(0.5 * x) * (1 - x**2),

          1.2 * (1 - x**2) * (1 + np.tanh(2 * x - 1)),

          4.1 * (x + 0.05) * (1 - x)**3 * np.exp(0.2 * x),

          0.6 * (1 - x) * (1 + 0.5 * np.exp(-2 * (x - 0.4)**2)
          * np.cos(3 * np.pi * x)),

          0.6 * (1 - x) * (1 + 0.4 * np.cosh(3 * x - 1)),

          0.7 * (1 - x) * np.cosh(2 * x)
          * np.exp(-0.5 * (x - 0.4)**2),

          1.3 * (1 - x) * (1 + 0.3 * np.sin(2 * np.pi * x)
          * np.cos(1.2 * np.pi * x)) * np.exp(-2 * x)
        )
```

```
  return pdfs

x = np.linspace(0.04, 1)
y=pdf_truth(x)

flavor = np.array(['d', 'dbar', 'u', 'ubar',
                   's', 'sbar', 'c', 'cbar', 'g'])

#Vizualizacija zadanih PDF-a:
for k in range(len(flavor)):

  color='C'+str(k)+'--'
  label=flavor[k]

  plt.plot(x, pdf_truth(x)[k], color, label=label)

plt.xlabel("x")
plt.ylabel("pdf")
plt.legend()
plt.savefig("pdfs.jpg")

##AMPLITUDE TVRDOG RASPRŠENJA:##

num_processes=len(flavor)

def h(x,n,k):

  a=np.zeros([len(flavor)])
  a[k]=1.0 #odabire se okus

  if n==1: #1. PROCES
    return (a[0]*(0.4+0.3*x) + a[1]*1/(1+x)
```

```
            +a[2]*x/(1+x**3)  + a[3]*0.3*x
            +a[4]*0.5*x/x     + a[5]*0.6*x
            +a[6]*x**2        + a[7]*(0.4+0.3*x**2)
            +a[8]*(-0.4*x))

  if n==2: #2. PROCES
    return (a[0]*x/(1+x)      + a[1]*0.8*x/x
            +a[2]*(0.5-0.4*x)  + a[3]*0.7*x/x
            +a[4]*(-0.3+0.3*x) + a[5]*0.3*x/x
            +a[6]*0.4*x        + a[7]*(-0.3*x**2+0.6*x)
            +a[8]*(0.5*x+1.2/(1+2*x)))

  if n==3: #3. PROCES
    return (a[0]*0.6*x/x                + a[1]*(-0.2+0.6*x)
            +a[2]*(-0.1*x/x)            + a[3]*1/(1+x**2)
            +a[4]*(0.3+x/(1+1.2*x))     + a[5]*(x**3+1)/(1+x)
            +a[6]*(0.5*x+x**3/(1+x**2)) + a[7]*(-0.3*x)
            +a[8]*0.4*x**2)

  if n==4: #4. PROCES
    return (a[0]*0.7*x/x     + a[1]*0.3*x
            +a[2]*1/(1+x**3) + a[3]*(0.6-0.5*x**2)
            +a[4]*1/(1+x**2) + a[5]*0.4*x**2
            +a[6]*x/(1+x)    + a[7]*(-0.4*x)
            +a[8]*(0.3*x-0.2*x**2))

  if n==5: #5. PROCES
    return (a[0]*(-0.4*x**2+0.5*x)      + a[1]*x/(1+x**2)
            +a[2]*(0.3*x+x**3/(1+x**2)) + a[3]*0.5*x/x
            +a[4]*(x**3+1)/(1+x)        + a[5]*x/(1+x)
            +a[6]*(-0.2*x)              + a[7]*0.7*x/x
            +a[8]*(-0.1+0.5*x))
```

```
  if n==6: #6. PROCES
    return (a[0]*1/(1+x**3)      + a[1]*(-0.4*x**2+x)
            +a[2]*(x**3+1)/(1+x) + a[3]*(0.3+x/(1+x**2))
            +a[4]*0.6*x          + a[5]*(0.35*x-0.5*x**2)
            +a[6]*0.5*x/x        + a[7]*(0.4-0.6*x)
            +a[8]*0.55*x/x)

  if n==7: #7. PROCES
    return (a[0]*(-0.5*x+0.2)    + a[1]*(x**3+1)/(1+x)
            +a[2]*x/(1+x**2)     + a[3]*1/(1+x)
            +a[4]*(0.5-0.4*x**2) + a[5]*0.5*x
            +a[6]*(-0.1+0.6*x)   + a[7]*(0.5+x/(1+1.2*x))
            +a[8]*(0.3*x+0.4))

  if n==8: #8. PROCES
    return (a[0]*(0.3+x/(1+1.2*x)) + a[1]*1/(1+x)
            +a[2]*(-0.4+0.4*x)      + a[3]*0.7*x
            +a[4]*(0.5-0.2*x**2)    + a[5]*(0.2+x/(1+x))
            +a[6]*0.7*x/x           + a[7]*1/(1+x**2)
            +a[8]*(-0.4*x**2+x))

  if n==9: #9. PROCES
    return (a[0]*(-0.1+0.7*x**2) + a[1]*x/(1+x**2)
            +a[2]*0.5*x/x         + a[3]*(-0.3+0.5*x**2)
            +a[4]*1/(1+x**2)      + a[5]*0.6*x/x
            +a[6]*1/(1+x)         + a[7]*0.7*x
            +a[8]*(0.4*x-0.1))

  else:
    print(f"Greška: drugi argument mora biti integer
        1,2,...,{num_processes}")

##DIFERENCIJALNI UDARNI PRESJECI:##
```

```
def sig(x, n, noise1=0, noise2=0):

  sig=0
  for k in range(len(flavor)):
    sig+=h(x,n,k)*pdf_truth(x)[k]

  sig = (sig * (1 + noise1*np.random.normal(0,1))
                 + noise2*np.random.normal(0,1))

  return sig

##SIMULIRANJE PODATAKA:##

ndata=np.array([14,16,15,17,15,16,14,18,17])

nplot = 80
noise = 0.1

#Točke u kojima će se simulirati podatci:
xdata1 = np.linspace(0.01, 0.8, ndata[0])
xdata2 = np.linspace(0.02, 0.76, ndata[1])
xdata3 = np.linspace(0.01, 0.78, ndata[2])
xdata4 = np.linspace(0.03, 0.8, ndata[3])
xdata5 = np.linspace(0.04, 0.69, ndata[4])
xdata6 = np.linspace(0.09, 0.78, ndata[5])
xdata7 = np.linspace(0.07, 0.83, ndata[6])
xdata8 = np.linspace(0.01, 0.8, ndata[7])
xdata9 = np.linspace(0.03, 0.72, ndata[8])
xdata = np.concatenate([xdata1,xdata2,xdata3,xdata4,xdata5,
                        xdata6,xdata7,xdata8,xdata9],axis=0)

#Točke za interpolaciju/ekstrapolaciju:
```

```
xplot = np.linspace(0.01, 0.82, nplot)

#Računanje standardne devijacije simuliranih podataka:
num_simulations = 100
errors=np.zeros([len(xdata),1])
l=0

for n in range(num_processes):

  for n_sim in range(num_simulations):

    if n_sim==0:
      temp=sig(xdata[l:l+ndata[n]].reshape(-1,1),
               n+1, noise, noise)

    else:
      temp=np.concatenate(
          [temp, sig(xdata[l:l+ndata[n]].reshape(-1,1),
                     n+1, noise, noise)],axis=1)

  errors[l:l+ndata[n],0]=np.std(temp,axis=1)

  l+=ndata[n]

np.random.seed(367)

#Simulacija podataka:
l=0
for n in range(num_processes):

  if n==0:

    data=np.array(
```

```
    [sig(x, n+1, noise, noise) for x in
     xdata[l:l+ndata[n]].reshape(-1,1)])

  else:

    data=np.concatenate([data,np.array(
    [sig(x, n+1, noise, noise) for x in
     xdata[l:l+ndata[n]].reshape(-1,1)])])

  l+=ndata[n]

data=np.concatenate([data,errors],axis=1)

#Amplitude tvrdog raspršenja svih okusa i procesa:
h_f_data=np.zeros([len(xdata),len(flavor)])
for k in range(len(flavor)):
  l=0
  for n in range(num_processes):

    if n==0:

      temp=h(xdata[l:l+ndata[n]],n+1,k)

    else:

      temp=np.concatenate([temp,h(xdata[l:l+ndata[n]],n+1,k)])

    l+=ndata[n]

  h_f_data[:,k]=temp

#"Istine" diferencijalnih udarnih presjeka svih procesa:
yplot=np.zeros([num_processes,nplot])
```

```
for n in range(num_processes):
  yplot[n,:]=sig(xplot,n+1)

#Vizualizacija simuliranih podataka:
fig, ax = plt.subplots(ceil(num_processes/3),3,figsize=[15,12])

l=0
for n in range(num_processes):

  color='C'+str(n)

  label=str(n+1)+'. proces - podatci'

  ax[int(n/3),n-3*int(n/3)].errorbar(
      xdata[l:l+ndata[n]], data[l:l+ndata[n],0],
      1.96*data[l:l+ndata[n],1], marker='x', linestyle='',
      ms=4, capsize=4, color=color, label=label)

  l+=ndata[n]

  label=str(n+1)+'. proces - "istina"'

  ax[int(n/3),n-3*int(n/3)].plot(
      xplot, yplot[n,:], color=color, label=label)

  ax[int(n/3),n-3*int(n/3)].set_xlabel("x")
  ax[int(n/3),n-3*int(n/3)].set_ylabel("d$\sigma$/dx")
  ax[int(n/3),n-3*int(n/3)].legend()

fig.savefig("dsigma_exp.jpg")

##PRIPREMA ZA UNAKRSNU PROVJERU:##

```

```
hyperprior={}

for k in range(len(flavor)):

  hyperprior['log(sigma'+str(k+1)+')'] = gvar.log(gvar.gvar(1,1))
  hyperprior['log(scale'+str(k+1)+')'] = gvar.log(gvar.gvar(0.5,0.5))
  hyperprior['log(beta'+str(k+1)+')'] = gvar.log(gvar.gvar(4,2))

def make_kernel(hp,kernel,k):

  sigma='sigma'+str(k)
  scale='scale'+str(k)
  beta='beta'+str(k)

  if kernel=='expquad':

    return hp[sigma]**2*lgp.ExpQuad(scale=hp[scale])

  if kernel=='cauchy':

    return hp[sigma]**2*lgp.Cauchy(scale=hp[scale], beta=hp[beta])

kf = KFold(n_splits=5, shuffle=True, random_state=42)

param_grid={}

for k in range(len(flavor)):
  param_grid['kernel'+str(k+1)]=['expquad']
              #može se staviti: ['expquad','cauchy']

keys, values = zip(*param_grid.items())
param_comb=[dict(zip(keys, v)) for v in itertools.product(*values)]

```

```
end_idx=len(param_comb)

##UNAKRSNA PROVJERA:##

#Promijeniti po potrebi
results_file='/content/drive/My Drive/checkpoints/gaussian_results.pkl'

#Učitavanje podataka ako već postoje:
if os.path.exists(results_file):

  with open(results_file, 'rb') as f:
    all_results = pickle.load(f)

  start_idx = all_results[-1]['index'] + 1
  best_score = all_results[-1]['best_score_sofar']
  best_params = all_results[-1]['best_params_sofar']

  print("Loaded results from previous run:")
  for result in all_results:
    print(result)

else:

  start_idx = 0
  all_results = []
  best_score = -float('inf')
  best_params = None

#Početak unakrsnih provjera:
for i in range(start_idx,end_idx):

  print("index:",i)
  loss=[]
```

```
351 params = param_comb[i]
352 print(f"Running grid search iteration {i+1}/{end_idx} with
353        parameters: {params}")
354
355 #Unakrsna provjera jednog modela:
356 for train_index, val_index in kf.split(xdata):
357
358   X_train_fold = xdata[train_index]
359   X_val_fold = xdata[val_index]
360
361   y_train_fold = data[train_index,0]
362   y_val_fold = data[val_index,0]
363
364   yerr_train_fold = data[train_index,1]
365   yerr_val_fold = data[val_index,1]
366
367   def makegp(hp):
368     """Definiranje ukupnog gausijanskog procesa:"""
369
370     gp= lgp.GP()
371
372     for k in range(len(flavor)):
373
374       gp=gp.defproc(
375           flavor[k], make_kernel(hp,params['kernel'+str(k+1)],k+1))
376
377       label_train='xf'+str(k+1)+'_train'
378       gp=gp.addx(X_train_fold, label_train, proc=flavor[k])
379
380       label_val='xf'+str(k+1)+'_val'
381       gp=gp.addx(X_val_fold, label_val, proc=flavor[k])
382
383       if k==0:
```

```
        gp=gp.addtransf(
            {label_train:np.diag(h_f_data[train_index,k])},
            'train'+str(k+1),axes=1)

        gp=gp.addtransf(
            {label_val:np.diag(h_f_data[val_index,k])},
            'val'+str(k+1),axes=1)

      else:

        gp=gp.addtransf(
            {'train'+str(k):1.0,
             label_train:np.diag(h_f_data[train_index,k])},
            'train'+str(k+1),axes=1)

        gp=gp.addtransf(
            {'val'+str(k):1.0,
             label_val:np.diag(h_f_data[val_index,k])},
            'val'+str(k+1),axes=1)

    return gp

#Podatci:
information = {
'train'+str(len(flavor)): gvar.gvar(y_train_fold,yerr_train_fold),
}

#Prilagodba (hiper)parametara modela:
fit = lgp.empbayes_fit(hyperprior, makegp, information,
                       method='gradient', raises=True,
                       verbosity=0)

```

```
    #Predviđanje na ispitnom skupu:
    gp = makegp(gvar.mean(fit.p))
    post = gp.predfromdata(information, ['val'+str(len(flavor))])

    ypred_mean = gvar.mean(post['val'+str(len(flavor))])

    #Ispitna pogreška:
    err=(ypred_mean-y_val_fold)/yerr_val_fold
    loss.append(np.mean(np.square(err)))

  #Negativni iznos "prave" pogreške modela:
  score=-np.mean(loss)

  if score > best_score:
    best_score = score
    best_params = params

  print("score = ", score)
  print(f"Best score so far: {best_score} with parameters:
        {best_params}")

  all_results.append(
      {'index':i, 'params': params, 'score': score,
       'best_params_sofar': best_params,
       'best_score_sofar': best_score })

  with open(results_file, 'wb') as f:
      pickle.dump(all_results, f)

print(f"Best score: {best_score} with parameters: {best_params}")

##POSTAVLJANJE NAJBOLJEG MODELA IZ UNAKRSNE PROVJERE:##

```

```
def makegp(hp):

  gp= lgp.GP()

  for k in range(len(flavor)):

    gp=gp.defproc(
        flavor[k], make_kernel(hp,best_params['kernel'+str(k+1)],k+1))

    label_plot='plot_'+flavor[k]
    gp=gp.addx(xplot, label_plot, proc=flavor[k])

  l=0
  for n in range(num_processes):

    gp=gp.deftransf('sum'+str(n+1),
     {flavor[k]: partial(h,n=n+1,k=k) for k in range(len(flavor))})

    gp=gp.addx(
        xdata[l:l+ndata[n]], 'datasum'+str(n+1),proc='sum'+str(n+1))
    l+=ndata[n]

    gp=gp.addx(
        xplot, 'plotsum'+str(n+1),proc='sum'+str(n+1))

  return gp

##PRILAGODBA HIPERPARAMETARA PRETHODNO ODABRANOG MODELA:##

information={}

l=0
for n in range(num_processes):
```

```
  information['datasum'+str(n+1)]=gvar.gvar(data[l:l+ndata[n],0],
                                            data[l:l+ndata[n],1])

  l+=ndata[n]

fit = lgp.empbayes_fit(hyperprior, makegp, information,
                        method='gradient', raises=True, verbosity=0)

##KONAČNA EKSTRAKCIJA PARTONSKIH DISTRIBUCIJSKIH FUNKCIJA:##

gp = makegp(gvar.mean(fit.p))

extract=[]
for k in range(len(flavor)):
  extract.append('plot_'+flavor[k])

for n in range(num_processes):

  extract.append('plotsum'+str(n+1))
  extract.append('datasum'+str(n+1))

#Predviđanja:
post = gp.predfromdata(information, extract)

##VIZUALIZACIJA REZULTATA - DIFERENCIJALNI UDARNI PRESJECI:##

fig,ax = plt.subplots(ceil(num_processes/3),3,figsize=[15,12])

l=0
for n in range(num_processes):

  color='C'+str(n)
```

```
label='plotsum'+str(n+1)

ypred_mean = gvar.mean(post[label])
ypred_sdev = gvar.sdev(post[label])
bottom = ypred_mean - 1.96*ypred_sdev
top    = ypred_mean + 1.96*ypred_sdev
ax[int(n/3),n-3*int(n/3)].fill_between(
    xplot, bottom, top, alpha=0.3, color='grey')

label=str(n+1)+'. proces - rezultat'

ax[int(n/3),n-3*int(n/3)].plot(
    xplot, ypred_mean, color=color, linestyle='--', label=label)

label=str(n+1)+'. proces - podatci'

ax[int(n/3),n-3*int(n/3)].errorbar(
    xdata[l:l+ndata[n]], data[l:l+ndata[n],0],
    1.96*data[l:l+ndata[n],1], marker='x', linestyle='',
    ms=4, capsize=4, color=color, label=label)

l+=ndata[n]

label=str(n+1)+'. proces - "istina"'

ax[int(n/3),n-3*int(n/3)].plot(
    xplot, yplot[n,:], color=color, label=label)

ax[int(n/3),n-3*int(n/3)].set_xlabel("x")
ax[int(n/3),n-3*int(n/3)].set_ylabel("d$\sigma$/dx")
ax[int(n/3),n-3*int(n/3)].legend()
```

```
fig.savefig("dsigma_res_gauss.jpg")

##OCJENA VALJANOSTI PRILAGODBE NA PODATKE:##

for n in range(num_processes):

  if n==0:
    data_pred=gvar.mean(post['datasum'+str(n+1)])

  else:
    data_pred=np.concatenate(
        [data_pred, gvar.mean(post['datasum'+str(n+1)])])

chi2=np.mean(np.square((data_pred-data[:,0])/data[:,1]))

print("chi2/N =",chi2)

##VIZUALIZACIJA REZULTATA - PARTONSKE DISTRIBUCIJSKE FUNKCIJE:##

fig,ax = plt.subplots(ceil(len(flavor)/3),3,figsize=[10,9])

for k in range(len(flavor)):

  color='C'+str(k)

  label='plot_'+flavor[k]

  ypred_mean = gvar.mean(post[label])
  ypred_sdev = gvar.sdev(post[label])
  bottom = ypred_mean - 1.96*ypred_sdev
  top    = ypred_mean + 1.96*ypred_sdev

  ax[int(k/3),k-3*int(k/3)].fill_between(
```

```
        xplot, bottom, top, alpha=0.3, color=color)

    label=flavor[k]+' - rezultat'

    ax[int(k/3),k-3*int(k/3)].plot(
        xplot, ypred_mean, color=color, linestyle='--', label=label)

    label=flavor[k]+' - "istina"'

    ax[int(k/3),k-3*int(k/3)].plot(
        xplot, pdf_truth(xplot)[k], color=color, label=label)

    ax[int(k/3),k-3*int(k/3)].set_xlabel("x")
    ax[int(k/3),k-3*int(k/3)].legend()

fig.savefig("pdfs_gauss.jpg")
```

Dodatak B ffnn.ipynb

```
from google.colab import drive
drive.mount('/content/drive')
import itertools
import os
import pickle
import datetime

import numpy as np
from scipy.integrate import simpson
import matplotlib.pyplot as plt
from math import ceil

import tensorflow as tf
```

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Dropout
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import ELU, ReLU
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import KFold
from tensorflow.keras.initializers import HeNormal
from tensorflow.keras.callbacks import Callback, TensorBoard
from sklearn.preprocessing import StandardScaler
from tensorflow.summary import create_file_writer
from tensorflow.summary import histogram

print(tf.__version__) #odrađeno na 2.15.0

##PARTONSKE DISTRIBUCIJSKE FUNKCIJE ZA REPLIKACIJU:##

def pdf_truth(x):
  """PDFs."""

  pdfs = (1.1 * (1 - x) * (1 + 0.6 * np.tanh(6 * x))
          * np.exp(-2 * (x - 0.3)**2),

          1.2 * (1 - x**0.5) * (1 + x * np.sin(5 * x)),

          1.1 * np.cosh(0.5 * x) * (1 - x**2),

          1.2 * (1 - x**2) * (1 + np.tanh(2 * x - 1)),

          4.1 * (x + 0.05) * (1 - x)**3 * np.exp(0.2 * x),

          0.6 * (1 - x) * (1 + 0.5 * np.exp(-2 * (x - 0.4)**2)
          * np.cos(3 * np.pi * x)),

```

```
        0.6 * (1 - x) * (1 + 0.4 * np.cosh(3 * x - 1)),

        0.7 * (1 - x) * np.cosh(2 * x)
        * np.exp(-0.5 * (x - 0.4)**2),

        1.3 * (1 - x) * (1 + 0.3 * np.sin(2 * np.pi * x)
        * np.cos(1.2 * np.pi * x)) * np.exp(-2 * x)
      )

  return pdfs

x = np.linspace(0.04, 1)
y=pdf_truth(x)

flavor = np.array(['d', 'dbar', 'u', 'ubar',
                   's', 'sbar', 'c', 'cbar', 'g'])

#Vizualizacija zadanih PDF-a:
for k in range(len(flavor)):

  color='C'+str(k)+'--'
  label=flavor[k]

  plt.plot(x, pdf_truth(x)[k], color, label=label)

plt.xlabel("x")
plt.ylabel("pdf")
plt.legend()

##AMPLITUDE TVRDOG RASPRŠENJA:##

num_processes=len(flavor)

```

```
def h(x,n):

  if n==1: #1. PROCES
    return (0.4+0.3*x, 1/(1+x),
            x/(1+x**3), 0.3*x,
            0.5*x/x,    0.6*x,
            x**2,       0.4+0.3*x**2,
            -0.4*x)

  if n==2: #2. PROCES
    return (x/(1+x),   0.8*x/x,
            0.5-0.4*x,  0.7*x/x,
            -0.3+0.3*x, 0.3*x/x,
            0.4*x,      -0.3*x**2+0.6*x,
            0.5*x+1.2/(1+2*x))

  if n==3: #3. PROCES
    return (0.6*x/x,             -0.2+0.6*x,
            -0.1*x/x,            1/(1+x**2),
            0.3+x/(1+1.2*x),     (x**3+1)/(1+x),
            0.5*x+x**3/(1+x**2), -0.3*x,
            0.4*x**2)

  if n==4: #4. PROCES
    return (0.7*x/x,   0.3*x,
            1/(1+x**3), 0.6-0.5*x**2,
            1/(1+x**2), 0.4*x**2,
            x/(1+x),    -0.4*x,
            0.3*x-0.2*x**2)

  if n==5: #5. PROCES
    return (-0.4*x**2+0.5*x,    x/(1+x**2),
            0.3*x+x**3/(1+x**2), 0.5*x/x,
```

```
            (x**3+1)/(1+x),      x/(1+x),
            -0.2*x,              0.7*x/x,
            -0.1+0.5*x)

    if n==6: #6. PROCES
      return (1/(1+x**3),    -0.4*x**2+x,
            (x**3+1)/(1+x), 0.3+x/(1+x**2),
            0.6*x,          0.35*x-0.5*x**2,
            0.5*x/x,        0.4-0.6*x,
            0.55*x/x)

    if n==7: #7. PROCES
      return (-0.5*x+0.2,  (x**3+1)/(1+x),
            x/(1+x**2),   1/(1+x),
            0.5-0.4*x**2, 0.5*x,
            -0.1+0.6*x,   0.5+x/(1+1.2*x),
            0.3*x+0.4)

    if n==8: #8. PROCES
      return (0.3+x/(1+1.2*x), 1/(1+x),
            -0.4+0.4*x,        0.7*x,
            0.5-0.2*x**2,      0.2+x/(1+x),
            0.7*x/x,           1/(1+x**2),
            -0.4*x**2+x)

    if n==9: #9. PROCES
      return (-0.1+0.7*x**2, x/(1+x**2),
            0.5*x/x,        -0.3+0.5*x**2,
            1/(1+x**2),     0.6*x/x,
            1/(1+x),        0.7*x,
            0.4*x-0.1)

    else:
```

```
    print(f"Greška: drugi argument mora biti integer
            1,2,...,{num_processes}")

##DIFERENCIJALNI UDARNI PRESJECI:##

def sig(x, n, noise1=0, noise2=0):

  sig=0
  for k in range(len(flavor)):
    sig+=h(x,n)[k]*pdf_truth(x)[k]

  sig = (sig * (1 + noise1*np.random.normal(0,1))
                + noise2*np.random.normal(0,1))

  return sig

##SIMULIRANJE PODATAKA:##

ndata=np.array([14,16,15,17,15,16,14,18,17])

nplot = 80
noise = 0.1

#Točke u kojima će se simulirati podatci:
xdata1 = np.linspace(0.01, 0.8, ndata[0])
xdata2 = np.linspace(0.02, 0.76, ndata[1])
xdata3 = np.linspace(0.01, 0.78, ndata[2])
xdata4 = np.linspace(0.03, 0.8, ndata[3])
xdata5 = np.linspace(0.04, 0.69, ndata[4])
xdata6 = np.linspace(0.09, 0.78, ndata[5])
xdata7 = np.linspace(0.07, 0.83, ndata[6])
xdata8 = np.linspace(0.01, 0.8, ndata[7])
xdata9 = np.linspace(0.03, 0.72, ndata[8])
```

```
xdata = np.concatenate([xdata1,xdata2,xdata3,xdata4,xdata5,
                        xdata6,xdata7,xdata8,xdata9],axis=0)

#Točke za interpolaciju/ekstrapolaciju:
xplot = np.linspace(0.01, 0.82, nplot)

#Računanje standardne devijacije simuliranih podataka:
num_simulations = 100
errors=np.zeros([len(xdata),1])
l=0

for n in range(num_processes):

  for n_sim in range(num_simulations):

    if n_sim==0:
      temp=sig(xdata[l:l+ndata[n]].reshape(-1,1),
               n+1, noise, noise)

    else:
      temp=np.concatenate(
          [temp,sig(xdata[l:l+ndata[n]].reshape(-1,1),
                    n+1, noise, noise)],axis=1)

  errors[l:l+ndata[n],0]=np.std(temp,axis=1)

  l+=ndata[n]

np.random.seed(367)

#Simulacija podataka:
l=0
for n in range(num_processes):
```

```
  if n==0:

    data=np.array(
    [sig(x, n+1, noise, noise) for x in
     xdata[l:l+ndata[n]].reshape(-1,1)])

  else:

    data=np.concatenate([data,np.array(
    [sig(x, n+1, noise, noise) for x in
     xdata[l:l+ndata[n]].reshape(-1,1)])])

  l+=ndata[n]

data=np.concatenate([data,errors],axis=1)

#Amplitude tvrdog raspršenja svih okusa i procesa:
h_f_data=np.zeros([len(xdata),len(flavor)])
for k in range(len(flavor)):
  l=0
  for n in range(num_processes):

    if n==0:

      temp=h(xdata[l:l+ndata[n]],n+1)[k]

    else:

      temp=np.concatenate([temp,h(xdata[l:l+ndata[n]],n+1)[k]])

    l+=ndata[n]

```

```
  h_f_data[:,k]=temp

#"Istine" diferencijalnih udarnih presjeka svih procesa:
yplot=np.zeros([num_processes,nplot])
for n in range(num_processes):
  yplot[n,:]=sig(xplot,n+1)

#Vizualizacija simuliranih podataka:
fig, ax = plt.subplots(ceil(num_processes/3),3,figsize=[15,12])

l=0
for n in range(num_processes):

  color='C'+str(n)

  label=str(n+1)+'. proces - podatci'

  ax[int(n/3),n-3*int(n/3)].errorbar(
      xdata[l:l+ndata[n]], data[l:l+ndata[n],0],
      1.96*data[l:l+ndata[n],1], marker='x', linestyle='',
      ms=4, capsize=4, color=color, label=label)

  l+=ndata[n]

  label=str(n+1)+'. proces - "istina"'

  ax[int(n/3),n-3*int(n/3)].plot(
      xplot, yplot[n,:], color=color, label=label)

  ax[int(n/3),n-3*int(n/3)].set_xlabel("x")
  ax[int(n/3),n-3*int(n/3)].set_ylabel("d$\sigma$/dx")
  ax[int(n/3),n-3*int(n/3)].legend()

```

```
fig.savefig("dsigma_exp.jpg")

##PRIPREMA PODATAKA:##

#Promjena oblika polja za ulazak u neuralnu mrežu:
xdata=xdata.reshape(-1,1)
xplot=xplot.reshape(-1,1)

#Skaliranje podataka:
scaler=StandardScaler()
xdata_scaled=scaler.fit_transform(xdata)
xplot_scaled=scaler.transform(xplot)

y_combined=np.hstack([data,h_f_data])
print("y_combined.shape=", y_combined.shape)

##POSTAVKE MODELA:##

tf.random.set_seed(42)

class CustomKerasRegressor():
  def __init__(self, layers=2, units1=64, units2=64,
               units3=64, units4=64, batch_normal=False,
               dropout=0.0, kernel_initializer=HeNormal(seed=42),
               activation='elu', alpha=1.0, learning_rate=0.001,
               batch_size=None, epochs=500):

    self.layers=layers
    self.units1=units1
    self.units2=units2
    self.units3=units3
    self.units4=units4
    self.batch_normal=batch_normal
```

```
311     self.dropout=dropout
312
313     self.kernel_initializer=kernel_initializer
314     self.activation=activation
315     self.alpha=alpha
316
317     self.learning_rate=learning_rate
318     self.batch_size=batch_size
319     self.epochs = epochs
320
321   def create_model(self):
322     """Arhitektura neuralne mreže:"""
323
324     #Ulazni sloj:
325     input=Input(shape=(1,), name='input')
326
327     #Prvi skriveni sloj:
328     x=Dense(
329         units=self.units1, kernel_initializer=self.kernel_initializer,
330         name='dense1')(input)
331
332     if self.batch_normal:
333       x=BatchNormalization(name='bn1')(x)
334
335     if self.activation=='elu':
336       x=ELU(alpha=self.alpha, name='elu1')(x)
337
338     if self.activation=='relu':
339       x=ReLU(name='relu1')(x)
340
341     x=Dropout(
342         self.dropout,
343         name='dropout1', seed=42)(x, training=True) #M.C. dropout
```

```
344
345     #Drugi skriveni sloj:
346     x=Dense(
347         units=self.units2, kernel_initializer=self.kernel_initializer,
348         name='dense2')(x)
349
350     if self.batch_normal:
351       x=BatchNormalization(name='bn2')(x)
352
353     if self.activation=='elu':
354       x=ELU(alpha=self.alpha, name='elu2')(x)
355
356     if self.activation=='relu':
357       x=ReLU(name='relu2')(x)
358
359     x=Dropout(
360         self.dropout,
361         name='dropout2', seed=42)(x, training=True) #M.C. dropout
362
363     #Treći skriveni sloj:
364     if self.layers>2:
365
366       x=Dense(
367           units=self.units3, kernel_initializer=self.kernel_initializer,
368           name='dense3')(x)
369
370       if self.batch_normal:
371         x=BatchNormalization(name='bn3')(x)
372
373       if self.activation=='elu':
374         x=ELU(alpha=self.alpha, name='elu3')(x)
375
376       if self.activation=='relu':
```

```
377         x=ReLU(name='relu3')(x)
378
379       x=Dropout(
380           self.dropout,
381           name='dropout3', seed=42)(x, training=True) #M.C. dropout
382
383     #Četvrti skriveni sloj:
384     if self.layers>3:
385
386       x=Dense(
387           units=self.units4, kernel_initializer=self.kernel_initializer,
388           name='dense4')(x)
389
390       if self.batch_normal:
391         x=BatchNormalization(name='bn4')(x)
392
393       if self.activation=='elu':
394         x=ELU(alpha=self.alpha, name='elu4')(x)
395
396       if self.activation=='relu':
397         x=ReLU(name='relu4')(x)
398
399       x=Dropout(
400           self.dropout,
401           name='dropout4', seed=42)(x, training=True) #M.C. dropout
402
403     #Izlazni sloj:
404     output=Dense(
405         units=len(flavor), kernel_initializer=self.kernel_initializer,
406         name='output')(x)
407
408     model=Model(input, output)
409
```

```
410     #Integracija algoritma optimizacije i funkcije gubitka u model:
411     model.compile(
412         optimizer=Adam(learning_rate=self.learning_rate),
413         loss=custom_loss)
414
415     return model
416
417   def summary(self):
418     """Za potrebe ilustracije modela:"""
419
420     self.model=self.create_model()
421
422     self.model.summary(show_trainable=True)
423
424   def fit(self, X, y, callbacks=None, validation_data=None, verbose=1):
425     """Optimizacija težina neuralne mreže:"""
426
427     self.model = self.create_model()
428
429     self.model.fit(
430         X, y, batch_size=self.batch_size, shuffle=True,
431         epochs=self.epochs, validation_data=validation_data,
432         callbacks=callbacks, verbose=verbose)
433
434     return self
435
436   def history(self, string):
437     """Za potrebe vizualizacije funkcije gubitka u procesu učenja:"""
438
439     return self.model.history.history[string]
440
441   def predict_mc(self, X, n_iter=10, verbose=1):
442     """Korištenje Monte Carlo dropout metode pri predviđanju:"""""
```

```
    result = np.zeros([n_iter,X.shape[0],len(flavor)])

    for i in range(n_iter):
      result[i,:,:]=self.model.predict(X, verbose=verbose)

    return result

def custom_loss(y_true, y_pred):
  """Funkcija gubitka za proces učenja:"""

  y_values=y_true[:,0:1]
  y_errors=y_true[:,1:2]

  sigma_pred=0
  for k in range(len(flavor)):
    sigma_pred+=y_true[:,2+k:3+k]*y_pred[:,k:k+1]

  loss = tf.reduce_mean(tf.square((y_values - sigma_pred) / y_errors))

  return loss

##PRIPREMA ZA UNAKRSNU PROVJERU:##

param_grid_2_layers = {
    'learning_rate': [0.001],
    'layers'      : [2],
    'units1'      : [16,32,64],
    'units2'      : [16,32,64],
    'dropout'     : [0.1,0.2],
    'batch_size'  : [16],
    'batch_normal' : [False],
    'epochs'      : [500],
```

```
    'activation'  : ['elu','relu']
}

param_grid_3_layers = {
    'learning_rate': [0.001],
    'layers'      : [3],
    'units1'      : [16,32,64],
    'units2'      : [16,32,64],
    'units3'      : [16,32,64],
    'dropout'     : [0.1,0.2],
    'batch_size'  : [16],
    'batch_normal' : [True,False],
    'epochs'      : [500],
    'activation'  : ['elu','relu']
}

param_grid_4_layers = {
    'learning_rate': [0.001],
    'layers'      : [4],
    'units1'      : [16,32],
    'units2'      : [16,32],
    'units3'      : [16,32],
    'units4'      : [16,32],
    'dropout'     : [0.1,0.2],
    'batch_size'  : [16],
    'batch_normal' : [True,False],
    'epochs'      : [500],
    'activation'  : ['elu','relu']
}

#Ako se ukaže potreba:
#param_grid=[param_grid_2_layers,
#            param_grid_3_layers,
```

```
#           param_grid_4_layers]

param_grid=param_grid_2_layers

##FUNKCIJA UNAKRSNE PROVJERE:##

def grid_search(param_grid_list, X_train, y_train):

  if isinstance(param_grid_list, list):

    all_param_comb = []

    for param_grid in param_grid_list:

      keys, values = zip(*param_grid.items())
      param_comb = [dict(zip(keys, v)) for v in
                    itertools.product(*values)]
      all_param_comb.extend(param_comb)

    param_comb=all_param_comb

  else:

    keys, values = zip(*param_grid_list.items())
    param_comb = [dict(zip(keys, v)) for v in
                  itertools.product(*values)]

  #Promijeniti po potrebi:
  results_file='/content/drive/My Drive/checkpoints/ffnn_results.pkl'

  #Učitavanje podataka ako već postoje:
  if os.path.exists(results_file):

```

```
    with open(results_file, 'rb') as f:
      all_results = pickle.load(f)

    start_idx = all_results[-1]['index'] + 1
    best_score = all_results[-1]['best_score_sofar']
    best_params = all_results[-1]['best_params_sofar']

    print("Loaded results from previous run:")
    for result in all_results:
      print(result)

  else:

    start_idx = 0
    all_results = []
    best_score = -float('inf')
    best_params = None

  end_idx = len(param_comb)

  kf = KFold(n_splits=5, shuffle=True, random_state=42)

  #Početak unakrsnih provjera:
  for i in range(start_idx, end_idx):

    print("index:",i)

    loss=[]
    params = param_comb[i]
    print(f"Running grid search iteration {i+1}/{end_idx} with
           parameters: {params}")

    if params['layers']==2:
```

```
575
576         model = CustomKerasRegressor(
577             layers=params['layers'],
578             units1=params['units1'],
579             units2=params['units2'],
580             batch_normal=params['batch_normal'],
581             dropout=params['dropout'],
582             activation=params['activation'],
583             learning_rate=params['learning_rate'],
584             batch_size=params['batch_size'],
585             epochs=params['epochs']
586             )
587
588     if params['layers']==3:
589
590         model = CustomKerasRegressor(
591             layers=params['layers'],
592             units1=params['units1'],
593             units2=params['units2'],
594             units3=params['units3'],
595             batch_normal=params['batch_normal'],
596             dropout=params['dropout'],
597             activation=params['activation'],
598             learning_rate=params['learning_rate'],
599             batch_size=params['batch_size'],
600             epochs=params['epochs']
601             )
602
603     if params['layers']==4:
604
605         model = CustomKerasRegressor(
606             layers=params['layers'],
607             units1=params['units1'],
```

```
608         units2=params['units2'],
609         units3=params['units3'],
610         units4=params['units4'],
611         batch_normal=params['batch_normal'],
612         dropout=params['dropout'],
613         activation=params['activation'],
614         learning_rate=params['learning_rate'],
615         batch_size=params['batch_size'],
616         epochs=params['epochs']
617         )
618
619     try:
620       #Unakrsna provjera jednog modela:
621       for train_index, val_index in kf.split(X_train):
622
623         X_fold_train = X_train[train_index]
624         X_fold_val  = X_train[val_index]
625
626         y_fold_train = y_train[train_index]
627         y_fold_val  = y_train[val_index]
628
629         #Prilagodba parametara modela:
630         model.fit(X_fold_train, y_fold_train, verbose=0)
631
632         #Predviđanje na ispitnom skupu:
633         n_iter=100
634         y_pred = model.predict_mc(X_fold_val, n_iter=n_iter, verbose=0)
635
636         sig_pred=np.sum(
637             y_pred[:,:,:]*y_fold_val[:,2:2+len(flavor)], axis=2)
638         sig_pred_mean=np.mean(sig_pred, axis=0)
639
640         #Ispitna pogreška:
```

```
641         loss.append(np.mean(np.square((sig_pred_mean-
642                                        y_fold_val[:,0])/y_fold_val[:,1])))
643
644       #Negativni iznos "prave" pogreške modela:
645       score = -np.mean(loss)
646
647       if score > best_score:
648         best_score = score
649         best_params = params
650
651       print("score = ", score)
652       print(f"Best score so far: {best_score} with parameters:
653             {best_params}")
654
655       all_results.append(
656           {'index':i,
657            'params': params,
658            'score': score,
659            'best_params_sofar': best_params,
660            'best_score_sofar': best_score }
661           )
662
663     except KeyboardInterrupt:
664       print("Training interrupted. Saving progress...")
665       break
666
667     #Spremanje podataka
668     with open(results_file, 'wb') as f:
669       pickle.dump(all_results, f)
670
671   print(f"Best score: {best_score} with parameters: {best_params}")
672
673   return best_params, best_score
```

```
674
675 ##UNAKRSNA PROVJERA:##
676
677 params,score = grid_search(param_grid, xdata_scaled, y_combined)
678
679 ##ALGORITAM ZA PRAĆENJE GRADIJENATA PRI UČENJU:##
680
681 class GradientLogger(Callback):
682   def __init__(self, X, y, batch_size, log_dir):
683
684     super().__init__()
685     self.X = X
686     self.y = y
687     self.batch_size = batch_size
688
689     self.log_dir = log_dir
690     self.writer = create_file_writer(self.log_dir)
691
692   def on_epoch_begin(self, epoch, logs=None):
693
694     self.gradients_accum=[tf.zeros_like(var) for var in
695                           self.model.trainable_variables]
696     self.batch_count = 0
697
698   def on_batch_end(self, batch, logs=None):
699
700     self.batch_count += 1
701     start = batch * self.batch_size
702     end = start + self.batch_size
703
704     X_batch = self.X[start:end]
705     y_batch = self.y[start:end]
706
```

```
707     with tf.GradientTape() as tape:
708
709       y_pred = self.model(X_batch, training=True)
710       loss = custom_loss(y_batch, y_pred)
711
712     gradients = tape.gradient(loss, self.model.trainable_variables)
713
714     for i, grad in enumerate(gradients):
715
716       if grad is not None:
717
718         self.gradients_accum[i] += grad
719
720   def on_epoch_end(self, epoch, logs=None):
721
722     with self.writer.as_default():
723
724       for i, grad_accum in enumerate(self.gradients_accum):
725
726         mean_grad = tf.reduce_mean(grad_accum/self.batch_count).numpy()
727         var_name = self.model.trainable_variables[i].name
728
729         histogram(f'gradients/{var_name}',
730                   grad_accum / self.batch_count,
731                   step=epoch)
732
733 ##TENSORBOARD - DEFINIRANJE:##
734
735 #Promijeniti po potrebi:
736 log_dir="logs/fit/"+datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
737
738 gradient_logger = GradientLogger(
739     xdata_scaled, y_combined,
```

```
740     batch_size=params['batch_size'], log_dir=log_dir
741     )
742
743 tensorboard_callback = TensorBoard(
744     log_dir=log_dir,
745     histogram_freq=1,
746     write_graph=True
747     )
748
749 ##POSTAVLJANJE NAJBOLJEG MODELA IZ UNAKRSNE PROVJERE:##
750
751 if params['layers']==2:
752
753   best_model=CustomKerasRegressor(
754       layers=params['layers'],
755       units1=params['units1'],
756       units2=params['units2'],
757       batch_normal=params['batch_normal'],
758       dropout=params['dropout'],
759       activation=params['activation'],
760       learning_rate=params['learning_rate'],
761       batch_size=params['batch_size'],
762       epochs=params['epochs']
763       )
764
765 if params['layers']==3:
766
767   best_model=CustomKerasRegressor(
768       layers=params['layers'],
769       units1=params['units1'],
770       units2=params['units2'],
771       units3=params['units3'],
772       batch_normal=params['batch_normal'],
```

```
773         dropout=params['dropout'],
774         activation=params['activation'],
775         learning_rate=params['learning_rate'],
776         batch_size=params['batch_size'],
777         epochs=params['epochs']
778         )
779
780 if params['layers']==4:
781
782   best_model=CustomKerasRegressor(
783         layers=params['layers'],
784         units1=params['units1'],
785         units2=params['units2'],
786         units3=params['units3'],
787         units4=params['units4'],
788         batch_normal=params['batch_normal'],
789         dropout=params['dropout'],
790         activation=params['activation'],
791         learning_rate=params['learning_rate'],
792         batch_size=params['batch_size'],
793         epochs=params['epochs']
794         )
795
796 ##PRILAGODBA PARAMETARA PRETHODNO ODABRANOG MODELA:##
797
798 best_model.fit(
799     xdata_scaled, y_combined,
800     callbacks=[tensorboard_callback,gradient_logger]
801     )
802
803 ##VIZUALIZACIJA MINIMIZACIJE FUNKCIJE GUBITKA PRI UČENJU:##
804
805 plt.plot(best_model.history('loss'),color='red')
```

```
plt.axhline(y=0, color='blue', linestyle='--')
plt.xlabel('epoha')
plt.ylabel('$\it{L}$')

##EKSTRAKCIJA PARTONSKIH DISTRIBUCIJSKIH FUNKCIJA:##

n_iter=100
pdf_plot = best_model.predict_mc(xplot_scaled,
                                 n_iter=n_iter, verbose=0)
pdf_data = best_model.predict_mc(xdata_scaled,
                                 n_iter=n_iter, verbose=0)
print("pdf_plot.shape =", pdf_plot.shape)
print("pdf_data.shape =", pdf_data.shape)

mean_prediction = np.mean(pdf_plot, axis=0)
print("mean_prediction.shape =", mean_prediction.shape)
std_prediction = np.std(pdf_plot, axis=0)
print("std_prediction.shape =", std_prediction.shape)

##VIZUALIZACIJA REZULTATA - PARTONSKE DISTRIBUCIJSKE FUNKCIJE:##

fig,ax = plt.subplots(ceil(len(flavor)/3),3,figsize=[10,9])

for k in range(len(flavor)):

  color='C'+str(k)

  bottom = mean_prediction[:,k] - 1.96*std_prediction[:,k]
  top = mean_prediction[:,k] + 1.96*std_prediction[:,k]
  ax[int(k/3),k-3*int(k/3)].fill_between(
      xplot.flatten(), bottom, top, alpha=0.3, color=color)

  label=flavor[k]+' - rezultat'
```

```
  ax[int(k/3),k-3*int(k/3)].plot(
      xplot, mean_prediction[:,k],
      color=color, linestyle='--', label=label)

  label=flavor[k]+' - "istina"'

  ax[int(k/3),k-3*int(k/3)].plot(
      xplot, pdf_truth(xplot)[k], color=color, label=label)

  ax[int(k/3),k-3*int(k/3)].set_xlabel("x")
  ax[int(k/3),k-3*int(k/3)].legend()

fig.savefig("pdfs_ffnn.jpg")

##PREDVIĐANJA DIFERENCIJALNIH UDARNIH PRESJEKA:##

sig_plot = np.zeros([n_iter,len(xplot),num_processes])

for n in range(num_processes):
  for k in range(len(flavor)):
    sig_plot[:,:,n]+=h(xplot,n+1)[k].reshape(1,len(xplot))*pdf_plot[:,:,k]

sig_plot_mean = np.mean(sig_plot, axis=0)
sig_plot_sdev = np.std(sig_plot, axis=0)

##VIZUALIZACIJA REZULTATA - DIFERENCIJALNI UDARNI PRESJECI:##

fig,ax = plt.subplots(ceil(num_processes/3),3,figsize=[15,12])

l=0
for n in range(num_processes):

```

```
    color='C'+str(n)

    label=str(n+1)+'. proces - rezultat'

    bottom = sig_plot_mean[:,n] - 1.96*sig_plot_sdev[:,n]
    top = sig_plot_mean[:,n] + 1.96*sig_plot_sdev[:,n]
    ax[int(n/3),n-3*int(n/3)].fill_between(
        xplot.flatten(), bottom, top, alpha=0.3, color='grey')

    ax[int(n/3),n-3*int(n/3)].plot(
        xplot.flatten(), sig_plot_mean[:,n],
        color=color, linestyle='--', label=label)

    label=str(n+1)+'. proces - podatci'

    ax[int(n/3),n-3*int(n/3)].errorbar(
        xdata[l:l+ndata[n]], data[l:l+ndata[n],0],
        1.96*data[l:l+ndata[n],1], marker='x', linestyle='',
        ms=4, capsize=4, color=color, label=label)

    l+=ndata[n]

    label=str(n+1)+'. proces - "istina"'

    ax[int(n/3),n-3*int(n/3)].plot(
        xplot, yplot[n,:], color=color, label=label)

    ax[int(n/3),n-3*int(n/3)].set_xlabel("x")
    ax[int(n/3),n-3*int(n/3)].set_ylabel("d$\sigma$/dx")
    ax[int(n/3),n-3*int(n/3)].legend()

fig.savefig("dsigma_ffnn.jpg")
```

```
##OCJENA VALJANOSTI PRILAGODBE NA PODATKE:##

sig_data=np.sum(pdf_data[:,:,:]*y_combined[:,2:2+len(flavor)], axis=2)
sig_data_mean=np.mean(sig_data, axis=0)
chi2=np.mean(np.square((sig_data_mean-data[:,0])/data[:,1]))
print("chi2/N =", chi2)

##TENSORBOARD AKTIVACIJA:##

%load_ext tensorboard
%tensorboard --logdir logs/fit
```

Literatura

[1] Isaacs A. Oxford Dictionary of Physics. 4th ed. : Oxford University Press, 2000.

[2] Thomson M. Modern Particle Physics : Cambridge University Press, 2013.

[3] Goodfellow I.; Bengio Y.; Courville A. Deep Learning : MIT Press, 2015.

[4] Shalev-Schwartz S.; Ben-David S. Understanding Machine Learning : From Theory to Algorithms : Cambridge University Press, 2014.

[5] Watt J.; Borhani R.; K. Katsaggelos A. Machine Learning Refined : Foundations, Algorithms, and Applications. 2nd ed. : Cambridge University Press, 2020.

[6] Rasmussen C.E.; Williams C.K.I. Gaussian Processes for Machine Learning : MIT Press, 2006.

[7] Bishop C.M. Pattern Recognition and Machine Learning : Springer, 2006.

[8] Biological Motivation for Neural Networks https://cs231n.github.io/neural-networks-1/, [07.08.2024]

[9] Introduction to Different Activation Functions for Deep Learning https://medium.com/@shrutijadon/survey-on-activation-functions-for-deep-learning-9689331ba092, [07.08.2024]

[10] Monte Carlo Dropout: A Practical Guide https://medium.com/@ciaranbench/monte-carlo-dropout-a-practical-guide-4b4dc18014b5, [25.08.2024]

[11] Understanding a Derivation of Bias Correction for the Adam Optimizer https://stats.stackexchange.com/questions/366076/understanding-a-derivation-of-bias-correction-for-the-adam-optimizer, [25.08.2024]

[12] lsqfitgp documentation https://gattocrucco.github.io/lsqfitgp/docs/index.html, [31.08.2024]

[13] tensorflow documentation https://www.tensorflow.org/, [01.09.2024]

[14] Google Colab https://colab.research.google.com/, [02.09.2024]