Rješavanje običnih diferencijalnih jednadžbi korištenjem neuronske mreže

Badrov, Ivan

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:217:853010

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-23**

Repository / Repozitorij:

Repository of the Faculty of Science - University of Zagreb

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Ivan Badrov

RJEŠAVANJE OBIČNIH DIFERENCIJALNIH JEDNADŽBI KORIŠTENJEM NEURONSKE MREŽE

Diplomski rad

Voditelj rada:
Mladen Jurak

Zagreb, kolovoz 2024.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Svima onima koji su me pratili i na bilo koji način sudjelovali na ovome putu, jedno veliko hvala!

Sadržaj

Sadržaj **iv**

Uvod **2**

1 Preduvjeti za neuronske mreže **3**
1.1 Obične diferencijalne jednadžbe 3
1.2 Numeričke metode 9

2 Neuronske mreže **15**
2.1 Osnove neuronskih mreža 16
2.2 PINN 29

3 Implemetacija PiNNs i rezultati **37**
3.1 Vektorizirani algoritam 37
3.2 Rezultati 42
3.3 Zaključak i završni komentari 49

Bibliografija **53**

Uvod

Živimo u vrijeme *četvrte* ili možda čak već *pete* industrijske, preciznije tehnološke revolucije. Napredak koji čovječanstvo ostvaruje iz godine u godinu posljednjih nekoliko desetljeća je izuzetan. Potrebe koje imamo i problemi koje susrećemo sve više ovise o stupnju razvoja tehnologije.

Jedna od takvih je i ona za obradom velike količine podataka u razumnom vremenu. Razvojem grafičkih kartica i ostalih računalnih komponenti, treniranje mreža koje se oslanjaju na velik broj parametara postalo je dostupno i jeftino. Shodno tome, ostvarivši obećavajuće rezultate, dolazi do procvata u istraživanju i daljnjem korištenju strojnog učenja.

Različiti podaci i problemi zahtijevali su različite inovativne pristupe prilikom bavljenja njima. Time dolazi do pojave velikog broja podvrsta neuronskih mreža. Jedna od tih je i fizikom informirana neuronska mreža (PINN) čije izučavanje je glavni interes ovog rada.

Osnovna ideja je da uz malo podataka, no uz *a priorno* poznavanje njihovih značajki i zakonitosti u ponašanju, usadimo to prethodno znanje u proces učenja čineći ga na taj način efikasnijim i manje zahtjevnim.

Dinamički sustavi pojavljuju se u mnogim znanstevnim disciplinama. Za kontinuirane dinamičke sustave vremenska evolucija se u većini slučajeva može opisati običnim diferencijalnim jednadžbama. Iako postoje mnoge numeričke metode koje se dugi niz godina uspješno bave aproksimacijom rješenja, prostora za napredak uvijek ima.

Rad započinje predstavljanjem područja običnih diferencijalnih jednadžbi i njihove važnosti u svakodnevnim problemima. Zatim, objašnjava poznate rezultate i daje motivaciju za njihovo efikasno rješavanje, odnosno bolje rečeno potrebu aproksimiranja.

Poglavlje se prirodno nastavlja pričom o numeričkim metodama korištenim za brzo i precizno rješavanje spomenutih problema. Riječ je o začetnoj Eulerovoj metodi i danas široko rasprostranjenoj Runge-Kutta četvrtog stupnja. Nju koristimo kao referentnu metodu prilikom nešto kasnije analize.

Nasuprot tradicionalnim pristupima, iduće poglavlje gradi alternativu u obliku neuronskih mreža te prikazuje osnovne pojmove. U njemu se bavimo opravdanjem optimizma i ilustracijom mogućnosti kojima raspolažu općenito. Također, spominjemo se i optimizacijskih tehnika poput metode adaptivne procjene momenta (ADAM) koja igraju ključnu ulogu u pronalasku parametara i u konačnici samoj predikciji modela.

Prethodno poglavlje o neuronskim mrežama služi kao preambula idućem koje u potpunosti počiva na njihovoj arhitekturi i funkcionalnosti.

Konačno, na kraju prezentiramo fizikom informirane neuronske mreže, kao i opravdanje za njihov suočavanje s diferencijalnim jednadžbama. To sve činimo da bi služeći se Pythonom kreirali vlastitu mrežu, zatim ju testirali na odabranim problemima i analizirali dobivene rezultate.

Poglavlje 1

Preduvjeti za neuronske mreže

1.1 Obične diferencijalne jednadžbe

U ovome poglavlju bit će prikazan kraći pregled bogate teorije diferencijalnih jednadžbi. Samo poznavanje teorije nije esencijalno za razumijevanje neuronskih mreža, niti se direktno implementira, no njena vrijednost je ilustrativne i motivacijske prirode.

Diferencijalne jednadžbe su matematički alati koji nam omogućavaju da modeliramo kako se stvari mijenjaju s vremenom ili u prostoru. Iako mnoge od njih već desetljećima znamo egzaktno riješiti, nažalost to nije dovoljno. Kod svakog rješenja osim preciznosti, uvijek su prisutna, a nerijetko i bitnija, pitanja zapisa i vremenskog okvira u kojem stižemo do rješenja.

Kako bismo razumjeli korijen problema i pronašli ispravan kut napada istog, u nastavku se najprije osvrćemo na problem inicijalne zadaće i njene rješivosti. Zatim, analiziramo samo rješenje te prezentiramo njegove tehničke nedostatke i načine kojima moderna znanost doskače u pomoć.

Uvodni primjeri

Diferencijalne jednadžbe se koriste za opisivanje dinamičkih sustava i njihovog puta promjene. Ove jednadžbe su ključne u mnogim oblastima znanosti i tehnologije. U nastavku slijedi nekolicina ilustrativnih primjera iz stvarnog svijeta:

Primjer 1.1.1 (Biologija: Rast populacije)**.** *Zamislite da proučavate rast populacije bakterija u laboratoriji. Bakterije se razmnožavaju vrlo brzo, a brzina njihovog razmnožavanja zavisi od trenutnog broja bakterija.*

Ova situacija može se modelirati diferencijalnom jednadžbom:

$$\frac{dP}{dt} = rP \tag{1.1}$$

Gdje je P populacija bakterija, t je vrijeme, a r je stopa rasta. Rješenjem ove jednadžbe možemo predvidjeti broj bakterija u bilo kojem trenutku u budućnosti.

Primjer 1.1.2 (Ekonomija: Ponuda i potražnja)**.** *Razmotrimo model ponude i potražnje gdje cijena proizvoda P ovisi o vremenu t. Pretpostavimo da je stopa promjene cijene proizvoda proporcionalna razlici između trenutne ponude $S(t)$ i trenutne potražnje $D(t)$.*

Možemo zapisati diferencijalnu jednadžbu ovako:

$$\frac{dP}{dt} = k(D(t) - S(t)) \tag{1.2}$$

Gdje je k pozitivna konstanta koja odražava koliko brzo cijena reagira na razliku između potražnje i ponude.

Primjer 1.1.3 (Fizika: Newtonov zakon hlađenja)**.** *Promotrimo svakodnevni primjer kada ostavite šalicu vruće kave u sobi. Temperatura kave se smanjuje s vremenom dok se ne izjednači s temperaturom sobe. Brzina hlađenja ovisi o razlici u temperaturi između kave i sobe.*

Koristeći Newtonov zakon hlađenja slijedi:

$$\frac{dT}{dt} = -k(T - T_{okolina}) \tag{1.3}$$

Gdje je T temperatura kave, t je vrijeme, $T_{okolina}$ je temperatura sobe, a k je konstanta hlađenja. Ova jednadžba pomaže da predvidimo koliko će vremena biti potrebno da se kava ohladi na određenu temperaturu.

Primjer 1.1.4 (Inženjerstvo: Električni krugovi)**.** *Diferencijalne jednadžbe se koriste za modeliranje strujanja struje u električnim krugovima. Na primjer, u RC krugu (otpornik-kondenzator), napon na kondenzatoru s vremenom može se modelirati diferencijalnom jednadžbom:*

$$\frac{dV}{dt} = -\frac{V}{RC} \tag{1.4}$$

Gdje je V napon, R otpornost, C kapacitet kondenzatora, a t je vrijeme. Prethodna jednadžba opisuje kako se napon mijenja s vremenom kada se kondenzator puni ili prazni.

Teorijski rezultati

Navođeni prethodnim primjerima promotrimo iduću zadaću:

$$y' = f(x, y) \tag{1.5}$$

$$y(x_0) = y_0 \tag{1.6}$$

Pri čemu je $f : \Omega \to \mathbb{R}$ zadana funkcija, $\Omega \subseteq \mathbb{R}^2$ otvoren skup, a $(x_0, y_0) \in \Omega$.

Napomena 1.1.5. *Zadaću jos nazivamo i inicijalnom ili Cauchyevom, a mozemo ju poopčiti i kao problem $n-$ tog stupnja gdje bi umjesto* (1.5) *pisali $y^{(n)} = f(x, y, y', .., y^{(n-1)})$ u eksplicintom obliku kada je to moguće, odnosno $F(x, y, y', .., y^{(n)}) = 0$ u općem ili implicintom obliku.*

Uz pretpostavku neprekidnosti funkcije f, za funkciju y kažemo da je rješenje diferencijalne jednadšbe (1.5) na otvorenom intervalu I ako:

- $y \in C^1(I)$
- graf $(x, y(x)) \in \mathbb{R}^2 : x \in I$ funkcije y je u domeni funkcije f
- $y'(x) = f(x, y(x)), \quad x \in I$

Odnosno, y je rješenje inicijalnog problema (1.5), (1.6) ako:

- y je rješenje jednadžbe (1.5) na intervalu I
- $x_0 \in I$ i $y(x_0) = y_0$

U nastavku jos spominjemo dva fundamentalna teorema u njihovoj najbazičnijoj formi - Peanov i Picardov. Valja imati na umu da su teoremi lokalni, no postoje rezultati koji ih u suštini proširuju globalno, ali tim rezultatima se nećemo baviti obzirom da nisu od užeg interesa u ovoj analizi. Zainteresirani čitatelj upućuje se na [32] kako bi se podrobnije upoznao s dokazima i materijom.

Teorem 1.1.6 (Picard-Lindelö teorem)**.** *Neka je dan inicijalni problem*

$$y'(x) = f(x, y(x)), \quad y(x_0) = x_0.$$

Ako je funkcija f neprekidna na domeni $I_a = [x_0 - a, x_0 + a] \times [y_0 - a, y_0 + a] \subset \mathbf{R}^2$, za neki $a > 0$, i f Lipschitz neprekidna na y, odnosno da postoji $k > 0$ takav da

$$|f(x, y_2) - f(x, y_1)| < k\,|y_2 - y_1|,$$

za svaki $(x, y_2), (x, y_1) \in I_a$*, onda postoji pozitivan* $b < a$ *takav da postoji jedinstveno rješenje y, na domeni* $[x_0 - b, x_0 + b]$*, za prethodno definiran inicijalan problem. [22]*

Dodatna snaga ovog teorema je u dokazu koji eksplicitno i konstruira rješenje oblika:

$$y(x) = y_0 + \int_{x_0}^{x} f(t, y(t))\, dt, \quad x \in I_a \tag{1.7}$$

Uz nešto slabije uvjete spominjemo se i idućeg rezultata:

Teorem 1.1.7 (Peanov teorem)**.** *Neka je* $\Omega \subseteq \mathbb{R}^2$ *otvoren skup, funkcija* $f \in C(\Omega; \mathbb{R})$ *i* $(x_0, y_0) \in \Omega$*. Tada Cauchyeva zadaća*

$$y'(x) = f(x, y(x)), \quad y(x_0) = y_0.$$

ima rješenje čija je domena otvoren skup kojem x_0 *pripada. [32]*

Napomena 1.1.8 (Sustavi diferencijalnih jednadžbi)**.** *Razni problemi zahtijevaju međuovisnost više funkcija. Slična stvar se prelijeva i na diferencijalne jednadžbe. Podižući stvari na višu dimenziju fundamentalni rezultati se uglavnom ne mijenjaju, već prirodno transformiraju na što se osvrće ova napomena.*

$$\tfrac{d}{dx}\mathbf{U}(x) = \mathbf{F}(x, \mathbf{U}) \tag{1.8}$$

Pri čemu je $\mathbf{U} : \mathbb{R} \to \mathbb{R}^n$ *tražena, a* $\mathbf{F} : \Omega \to \mathbb{R}^n$ *zadana funkcija za* $\Omega \subseteq \mathbb{R} \times \mathbb{R}^n$ *otvoren skup. Označimo li s* $(\mathbf{e}_1, \ldots, \mathbf{e}_n)$ *kanonsku bazu u* $\mathbb{R}^n$*, onda vektor* $\mathbf{U}(x) = \sum_{i=1}^{n} U_i(x)\mathbf{e}_i$ *identificiramo s vektorom stupcem* $(U_1(x), \ldots, U_n(x))^\tau$*. Analogno, za funkciju* $\mathbf{F}$*; za zadane* $(x, \mathbf{U}) \in \Omega$ *je* $\mathbf{F}(x, \mathbf{U}) = \sum_{i=1}^{n} F_i(x, \mathbf{U})\mathbf{e}_i$*, gdje su* F_i *realne funkcije definirane na* $\Omega, i = 1, \ldots, n$*. Konačno, raspisani sustav jednadžbi* (1.8) *glasi*

$$\begin{aligned}
\frac{d}{dx}U_1 &= F_1(x, U_1, \ldots, U_n), \\
\frac{d}{dx}U_2 &= F_2(x, U_1, \ldots, U_n), \\
&. \\
\frac{d}{dx}U_n &= F_n(x, U_1, \ldots, U_n).
\end{aligned}$$

Uz diferencijalnu jednadžbu (1.8) *vežemo i odgovarajući inicijalni problem: za zadane* $x_0 \in \mathbb{R}, \mathbf{U}_0 \in \mathbb{R}^n$ *i* $\mathbf{F} : \Omega \to \mathbb{R}^n$ *naci funkciju* $\mathbf{U} : \mathbb{R} \to \mathbb{R}^n$ *takvu da vrijedi*

$$\begin{cases} \frac{d}{dx}\mathbf{U}(x) = \mathbf{F}(x, \mathbf{U}), \\ \mathbf{U}(x_0) = \mathbf{U}_0. \end{cases} \tag{1.9}$$

Pri tome je minimalna pretpostavka da je funkcija F neprekinuta na svojoj domeni. Pojam rješenja inicijalnog problema uvodi se po analogiji sa skalarnom jednadžbom; kažemo da je funkcija $\mathbf{U} : I \to \mathbb{R}^n$ *rješenje problema* (1.9) *na otvorenom intervalu* $I \subset \mathbb{R}$ *ako vrijedi*

- $\mathbf{U} \in C^1(I; \mathbb{R}^n)$
- $\left\{(x, \mathbf{U}(x)) \in \mathbb{R}^{n+1} : x \in I\right\} \subseteq \Omega$
- $x_0 \in I, \quad \mathbf{U}(x_0) = \mathbf{U}_0$
- $\mathbf{U}'(x) = \mathbf{F}(x, \mathbf{U}(x)), \quad x \in I$

Osnovni rezultati postojanja i jedinstvenosti rješenja Cauchyjeve zadaće; Picardov teorem i njegov dokaz, prenosi se uz manje modifikacije na Cauchyjevu zadaću (1.9).

Napomena 1.1.9. *Jednadžba višeg reda $y^{(n)} = f(x, y, y', ..., y^{(n-1)})$ može se svesti na sustav prvog reda:*

$$\frac{d}{dx}\begin{bmatrix} y_1(x) \\ y_2(x) \\ \vdots \\ y_{n-1}(x) \\ y_n(x) \end{bmatrix} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix}\begin{bmatrix} y_1(x) \\ y_2(x) \\ \vdots \\ y_{n-1}(x) \\ y_n(x) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ f(x, y_1, \ldots, y_n) \end{bmatrix} \tag{1.10}$$

Za koji vrijede sljedeće oznake:

$$y_1 = y, \quad y_2 = y', \quad ..., \quad y_n = y^{(n-1)}$$

$$y_1' = y' = y_2, \quad ..., \quad y_n' = y^{(n)} = f(x, y, y', ..., y^{(n-1)}) = f(x, y_1, ..., y_n)$$

Za kraj spomenimo još poseban tip običnih diferencijalnih jednadžbi prvog tipa poznatiji kao linearne diferencijalne jednadžbe. Iako je teorija ove discipline mnogo bogatija od prikazanih osnova, razlike i svojstva koja će upravo biti navedena više su nego dovoljni za motivaciju idućeg poglavlja.

Odgovarajuća zadaća oblika

$$y'(x) = a(x)y + b(x), \quad y(x_0) = y_0 \tag{1.11}$$

gdje su a i b neprekidne funkcije na intervalu $I = (t_1, t_2)$ koji sadrži točku x_0, odnosno $y_0 \in \mathbb{R}$; ima jedinstveno rješenje y na tom istom intervalu dano formulom:

$$y(x) = e^{A(x)}\left(y_0 + \int_{x_0}^{x} e^{-A(t)} b(t) dt\right) \tag{1.12}$$

gdje je $A(x) = \int_{x_0}^{x} a(t)dt$. [22]

Neka od svojstava koje zadovoljava rješenje linearne diferencijalne jednadžbe su:

- Postoji eksplicitan izraz za rješenje
- Za svaki početni uvjet $y_0 \in \mathbb{R}$ postoji jedinstveno rješenje
- Za svaki početni uvjet $y_0 \in \mathbb{R}$ rješenje $y(x)$ je definirano za sve (t_1, t_2)

S druge strane ista svojstva ne uočavamo nužno kod nelinearnih sustava. Promatrajući Picard-Lindeloefov teorem zapažamo iduće:

- Ne postoji opća eksplicitna formula za nelinearan sustav
- U slučaju izostanka uvjeta Lipschitz neprekidnosti funkcije f, rješenje ne mora biti jedinstveno
- Domena rješenja y može varirati ovisno inicijalnoj promjeni na y_0

Prvo na što treba obratiti pozornost je da postoji izvjestan broj jednandžbi koje nemaju analitičko rješenje. Osim toga, čak i kada postoji kao u primjeru (1.12), odnosno elementarnije (1.7); ostaje problematika kompleksnosti izračuna; kako rješiti potrebne integrale ili nešto treće.

Nadalje, valja podsjetiti i na uvijek prisutnu važnost zapisa problema u računalu te izazova s kojima se tu susrećemo, a kojima se bavi cijela jedna znanstvena disciplina, numerička matematika. To uključuje sve od preciznost zapisa i greški prilikom evaluacije pa do vremenske komponente izračuna koja je fundamentalna za bilo kakvu praktičnu primjenu.

Prethodni problemi upućuju nas da rješenje i potporu pri izračunu potražimo na drugom mjestu - put nas dalje prirodno vodi do već najavljenog idućeg poglavlja numeričkih metoda.

1.2 Numeričke metode

Numerička matematika je grana matematike koja se bavi razvojem i primjenom algoritama za rješavanje matematičkih problema korištenjem brojčanih aproksimacija. Za razliku od analitičkih rješenja, gdje se traže egzaktni izrazi, numeričke metode omogućuju rješavanje kompleksnih problema koji često nemaju jednostavno analitičko rješenje.

Numeričke metode obuhvaćaju tehnike poput rješavanja sustava linearnih i nelinearnih jednadžbi, integracije i derivacije, optimizacije, te rješavanja parcijalnih i običnih diferencijalnih jednadžbi. Ove metode omogućuju rješavanje problema koji su previše složeni za ručno računanje, te koriste računala za brze i točne aproksimacije rješenja.

Preciznost i brzina izračuna ključni su faktori pri ocjenjivanju učinkovitosti numeričkih metoda. Numeričke metode daju aproksimacije, a njihova točnost ovisi o različitim čimbenicima, uključujući način diskretizacije, broj iteracija, veličinu koraka ili mreže, te stabilnost metode. U mnogim slučajevima moguće je kontrolirati preciznost rješenja povećanjem broja točaka u mreži ili smanjenjem koraka, no to obično dolazi uz cijenu povećanja vremena izračuna.

Brzina izračuna ovisi o složenosti algoritma i resursima dostupnim za računalnu obradu. Neke numeričke metode, poput Newton-Raphson metode za nelinearne jednadžbe, konvergiraju brzo do rješenja, ali mogu biti osjetljive na početne uvjete. S druge strane, metode poput metode konačnih elemenata ili Monte Carlo simulacija mogu zahtijevati mnogo više računskih resursa, posebno za visoke dimenzije ili složene geometrije.

U usporedbi s analitičkim rješenjima, koja mogu dati egzaktne odgovore, numeričke metode nude fleksibilnost i mogućnost rješavanja problema gdje analitičko rješenje ne postoji ili je previše složeno za dobivanje. Analitička rješenja su precizna i brza ako su dostupna, no njihova primjena je često ograničena na jednostavnije probleme. S druge strane, numeričke metode pružaju rješenja za širok spektar praktičnih problema, iako su ograničene kompromisom između točnosti i vremena izračuna.

Eulerova metoda

Jednu od prvih i fundamentalnih ideja pripisujemo možda najvećem matematičaru svih vremena Leonardu Euleru kome u čast danas nosi ime. Metoda i danas nalazi svoju primjenu obzirom na jednostavnost i direktan pristup pri rješavanju običnih diferencijalnih jednadžbi.

Za početak promotrimo inicijalni sustav:

$$\begin{aligned} \frac{dy}{dt} &= f(t, x, y), \quad y(t_0) = y_0 \\ \frac{dx}{dt} &= g(t, x, y), \quad x(t_0) = x_0 \end{aligned} \tag{1.13}$$

Služeći se definicijom derivacije slijedi

$$\begin{aligned} y' &= \lim_{h \to 0} \frac{f(t+h) - f(t)}{h} \\ x' &= \lim_{h \to 0} \frac{g(t+h) - g(t)}{h} \end{aligned} \tag{1.14}$$

za mali $h > 0$, što dalje implicira razumnu pretpostavku za aproksimaciju derivacija u obliku

$$\begin{aligned} y' &\approx \frac{f(t+h) - f(t)}{h} \\ x' &\approx \frac{g(t+h) - g(t)}{h} \end{aligned} \tag{1.15}$$

Odnosno uvrštavanjem u (1.13) dobivamo

$$\begin{aligned} f(t, x, y) &= \frac{f(t+h) - f(t)}{h} \\ g(t, x, y) &= \frac{g(t+h) - g(t)}{h} \end{aligned} \tag{1.16}$$

Daljnim odabirom veličine koraka h i na taj način profinjenjem intervala od interesa, dolazimo konačno do:

$$\begin{aligned} y_{k+1} &= y_k + hf(x_k, y_k) \\ x_{k+1} &= x_k + hg(x_k, y_k) \end{aligned} \tag{1.17}$$

Ako tako dobiveno rješenje linearno interpoliramo, dobiva se po dijelovima afina funkcija na podsegmentima $[t_{k-1}, t_k]$ za $k \in \{0, 1, ..., n-1\}$

$$\begin{aligned} y(t) &= y_k + f(x_k, y_k)(t - t_k) \\ x(t) &= x_k + g(x_k, y_k)(t - t_k) \end{aligned} \tag{1.18}$$

Iz izvoda se vidi jednostavnost algoritma koji izvrjednjuje funkcije po prikazanom, a ujedino velika je prednost i brzina koja slijedi iz jednog računanja funkcije f po koraku. Također, garantirana je konvergencija metode prema egzaktnom rješenju, no sama preciznost uvijek ovisi o cijeni koju smo voljni platiti. Postoje mnoge modifikacije ove metode koje su danas u širokoj primjeni.

Runge-Kutta metoda

Jedna od najpopularnijih i raširenijih metoda današnjice za rješavanje diferencijalnih jednandžbi je Runge-Kutta četvrtog reda. Obzirom na efikasnu primjenu na rješavanje diferencijalnih jednandžbi, ne čudi da se ona često odabire kao referentna numerička metoda u danom kontekstu.

Uz pretpostavke dovoljne glatkoće Taylorov red funkcije jedne varijable u točki glasi

$$y(x) = \sum_{n=0}^{\infty} \frac{y^{(n)}}{n!}(x - x_0)^n \tag{1.19}$$

gdje uz $x = x_0 + h$ dobivamo

$$y(x) = y(x_0) + hy'(x_0) + h^2\frac{y''(x_0)}{2!} + ... \tag{1.20}$$

Slična situacija vrijedi i za funkciju f u više varijabli, no zbog jednostavnosti prikaza promotrimo samo slučaj $n = 2$:

$$f(x, y) = \sum_{n=0}^{\infty} \frac{1}{n!}\left\{(x - x_0)\frac{\partial}{\partial x} + (y - y_0)\frac{\partial}{\partial y}\right\}^i f(x, y) \tag{1.21}$$

Ponovno uz $x = x_0 + mh$ i $y = y_0 + nh$ dobivamo izraz:

$$f(x, y) = f(x_0, y_0) + h\Big(mf_x + nf_y\Big) + \frac{h^2}{2!}\Big(m^2 f_{xx} + 2mnf_{xy} + n^2 f_{yy}\Big) + ... \tag{1.22}$$

Vratimo se sada izvornom problemu

$$\frac{dy}{dx} = f(x, y)$$

Prilično je jasno da uz danu notaciju vrijedi i

$$df = \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy$$

$$y'' = f' = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}\frac{dy}{dx} \tag{1.23}$$

$$y'' = f_x + ff_y$$

Koristeći Taylorov razvoj u točki (x_0, y_0) i daljnji izračun kako je prikazano prethodno uvrštavanjem dobivamo:

$$y(x) - y(x_0) = hf(x_0) + \frac{h^2}{2!}\left(f_x + ff_y\right) + \frac{h^3}{3!}\left(f_{xx} + 2ff_{xy} + f^2 f_{yy} + f_x f_y + ff_y^2\right) + ... \quad (1.24)$$

Ideja metode je u odabiru nekoliko točaka tako da razvoj iz (1.22) odgovaraju terminima desne strane jednadžbe (1.24).

Neka su

$$\begin{aligned} m_1 &= hf\left(x_0, y_0\right) \\ m_2 &= hf\left(x_0 + nh, y_0 + nm_1\right) \\ m_3 &= hf\left(x_0 + ph, y_0 + pm_2\right) \\ m_4 &= hf\left(x_0 + qh, y_0 + qm_3\right) \end{aligned} \quad (1.25)$$

Tada korištenjem (1.24) vrijedi:

$$\begin{aligned} m_1 &= hf \\ m_2 &= h\left[f + nh\left(f_x + ff_y\right) + \frac{(nh)^2}{2}\left(f_{xx} + 2ff_{xy} + f^2 f_{yy}\right) + \cdots\right] \\ m_3 &= h\left[f + ph\left(f_x + ff_y\right) + \frac{(ph)^2}{2}\left(f_{xx} + 2ff_{xy} + f^2 f_{yy}\right) + 2np\left(f_x f_y + ff_y{}^2\right) + \cdots\right] \\ m_4 &= h\left[f + qhc + \frac{(qh)^2}{2}\left(f_{xx} + 2ff_{xy} + f^2 f_{yy}\right) + 2pq\left(f_x f_y + ff_y{}^2\right) + \cdots\right] \end{aligned} \quad (1.26)$$

Promotrimo li izraz $am_1 + bm_2 + cm_3 + dm_4$ i koeficijente uz redom izraze:

$$\begin{aligned} \boldsymbol{hf} &: a + b + c + d = 1 \\ \boldsymbol{hf\left(f_x + ff_y\right)} &: bn + cp + dq = \frac{1}{2} \\ \boldsymbol{hf\left(f_{xx} + 2ff_{xy} + f^2 f_{yy}\right)} &: bn^2 + cp^2 + dq^2 = \frac{1}{3} \\ \boldsymbol{hf\left(f_x f_y + ff_y{}^2\right)} &: cnp + dpq = \frac{1}{6} \end{aligned} \quad (1.27)$$

Uočavamo da je riječ o sustavu 4 jednadžbe i 7 nepoznanica.
Kako bismo ga rješili, proizvoljno odabiremo 3 - $n = p = \frac{1}{2}$ i $q = 1$. Time dolazimo do rješenja (1.27):

$$a = d = \frac{1}{6}, \quad b = c = \frac{1}{3}$$

Uvrštavajući prethodno izračunate koeficijente u izraz (1.24) dolazimo do:

$$y(x) - y(x_0) = am_1 + bm_2 + cm_3 + dm_4 \tag{1.28}$$

Odnosno, općnito vrijedi:

$$y_{n+1} = y_n + \frac{1}{6}(m_1 + 2m_2 + 2m_3 + m_4) \tag{1.29}$$

uz oznake:

$$m_1 = hf(x_n, y_n) \tag{1.30}$$

$$m_2 = hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}m_1\right) \tag{1.31}$$

$$m_3 = hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}m_2\right) \tag{1.32}$$

$$m_4 = hf(x_n + h, y_n + m_3) \tag{1.33}$$

Treba istaknuti da je opisana Runge Kutta metoda (RK4) četvrtog reda, odnosno da je greška kraćenja lokalno reda veličine $O\left(h^5\right)$ i akumulirana greška reda $O\left(h^4\right)$.

Analogno, može se proširiti formula na sustave. Promotrimo slučaj $n = 2$:

$$\begin{aligned} \frac{dy}{dt} &= f(t, x, y), \quad y(t_0) = y_0 \\ \frac{dx}{dt} &= g(t, x, y), \quad x(t_0) = x_0 \end{aligned} \tag{1.34}$$

Pripadno rješenje je oblika:

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}(m_1 + 2m_2 + 2m_3 + m_4) \\ x_{n+1} &= x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned} \tag{1.35}$$

Gdje su:

$$
\begin{aligned}
k_1 &= hf\left(t_n, x_n, y_n\right) & m_1 &= hg\left(t_n, x_n, y_n\right) \\
k_2 &= hf\left(t_n + \frac{1}{2}h, x_n + \frac{1}{2}k_1, y_n + \frac{1}{2}m_1\right) & m_2 &= hg\left(t_n + \frac{1}{2}h, x_n + \frac{1}{2}k_1, y_n + \frac{1}{2}m_1\right) \\
k_3 &= hf\left(t_n + \frac{1}{2}h, x_n + \frac{1}{2}k_2, y_n + \frac{1}{2}m_2\right) & m_3 &= hg\left(t_n + \frac{1}{2}h, x_n + \frac{1}{2}k_2, y_n + \frac{1}{2}m_2\right) \\
k_4 &= hf\left(t_n + h, x_n + k_3, y_n + m_3\right) & m_4 &= hg\left(t_n + h, x_n + k_3, y_n + m_3\right)
\end{aligned}
\tag{1.36}
$$

Eulerova metoda je jednostavan numerički pristup za rješavanje običnih diferencijalnih jednadžbi, koji koristi linearne aproksimacije za korak integracije. Iako je jednostavna za implementaciju, metoda pati od niske točnosti jer je samo prvog reda, što znači da lokalna greška raste proporcionalno s korakom h, a ukupna greška raste kao $O(h)$.

S druge strane, Runge-Kutta 4. reda (RK4) je metoda koja koristi više procjena nagiba unutar svakog koraka integracije kako bi postigla puno veću točnost. RK4 ima ukupnu grešku reda $O(h^4)$, što omogućava mnogo točnije rezultate s većim korakom h, za razliku od Eulerove metode.

Općenito, prednosti RK4 su mnoge i uključuju njezinu znatno veću točnost i stabilnost. Bez obzira na jednostavnost implementacije nedostaci uključuju veći računski trošak jer je potrebno izračunati četiri procjene nagiba po svakom koraku, što može biti zahtjevno kod složenih sustava ili vrlo dugih simulacija. Isto tako, prilikom računanja koristi fiksnu veličinu koraka, što zna biti problematično kod dinamičkih sustava kod kojih se adaptivni koraci pokazuju poželjnijom varijantom. Njena nestabilnost također zna doći do izražaja kod krutih sustava kao i sekvencijalnost provedbe koraka koja onemugućuje brže, paralelne implementacije.

Naravno, postoje mnoge inačice ove metode koje adresiraju spomenute probleme (Runge-Kutta-Fehlberg, implicitni RK, itd.) zbog čega ova metoda pronalazi mnoge primjene dan danas. Imajući na umu pokazano, u nastavku obrađujemo neuronske mreže, moderni pristup sve većim računalnim i podatkovnim zahtjevima današnjice. One pronalaze svoje mjesto u sve širem i širem spektru, a njihova svojstva se pokazuju itekako pogodnima za suočavanjem s problemom diferencijalnih jednadžbi.

Poglavlje 2

Neuronske mreže

Neuronske mreže, kao temelj moderne umjetne inteligencije, predstavljaju spoj biologije i računalne znanosti. Inspirirane su strukturom i funkcijom ljudskog mozga, a sastoje se od međusobno povezanih čvorova ili *neurona*. U digitalnom svijetu, neuronske mreže su modeli strojnog učenja koji su sposobni učiti iz podataka i prepoznavati složene obrasce.

Pronašle su primjenu u različitim područjima, uključujući prepoznavanje govora, računalni vid, prirodnu obradu jezika, robotiku i mnoge druge. Njihova sposobnost da generaliziraju iz podataka čini ih neizostavnim alatom u rješavanju problema koji su prethodno bili izvan dosega tradicionalnih algoritama.

Osnovna jedinica u neuronskoj mreži je umjetni neuron, koji prima jedan ili više ulaza, obrađuje ih pomoću funkcije aktivacije, te proizvodi izlaz. Ovi neuroni su organizirani u slojeve, stvarajući složene mreže koje su sposobne modelirati izrazito kompleksne odnose.

U ovom poglavlju ćemo najprije istražiti njihovu arhitekturu, zatim različite vrste aktivacijskih funkcija, funkcije gubitka, metode treniranja i optimizacije parametara, te se na koncu osvrnuti na posebnu vrstu neuronskih mreža poznatu kao Physics Informed Neural Networks (PINNs).

PINNs su posebno zanimljive jer inkorporiraju fizičke zakone unutar strukture mreže, omogućujući preciznije i robusnije modele u područjima gdje su podaci rijetki ili skupi za prikupljanje.

2.1 Osnove neuronskih mreža

Arhitektura neuronskih mreža

Osnovna arhitektura neuronske mreže može se opisati kao skup neurona organiziranih u slojeve. Postoje tri glavne vrste slojeva:

- **Ulazni sloj** (*Input layer*): prima ulazne podatke.
- **Skriveni slojevi** (*Hidden layers*): obrađuju podatke pomoću aktivacijske funkcije i zadanih transformacija.
- **Izlazni sloj** (*Output layer*): proizvodi konačni izlaz mreže.

Matematički, izlaz jednog neurona u sloju može se opisati kao:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.1}$$

gdje su x_i ulazne vrijednosti, w_i težine, b pristranost (bias) te f aktivacijska funkcija.

Drugim riječima, za ulazne podatke x, parametri w_i odreduju koliku važnost imaju različiti segmenti ulaza, dok pristranost b i aktivacijska funkcija f modeliraju problem prema konačnoj formi. Učenje neuronske mreže svodi se na pronalaženje optimalnih vrijednosti parametara w i b takvih da za ulazne podatke x izlaz y predstavlja *dovoljno dobru* aproksimaciju funkcije od interesa.

Pitanja na koja prvo treba odgovoriti su pitanja egzistencije rješenja; odnosno u afirmativnom slučaju, pod kojim uvjetima i na koji način odabiremo i pronalazimo sastavne elemente neuronske mreže.

Aktivacijske funkcije

Aktivacijske funkcije igraju ključnu ulogu u neuronskim mrežama jer omogućuju modeliranje složenih nelinearnih odnosa među podacima. Bez njih, neuronske mreže bi se svela na jednostavne linearne modele, čime bi se znatno smanjila njihova moć izražavanja.
Postoji nekoliko vrsta aktivacijskih funkcija koje se često koriste:

Sigmoidna funkcija

Sigmoidna funkcija je jedna od najstarijih i najčešće korištenih aktivacijskih funkcija. Njena formula glasi:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

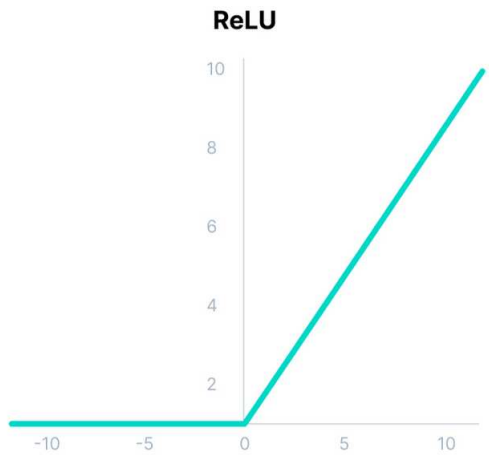
Sigmoidna funkcija transformira ulazne vrijednosti u interval (0, 1), što ju čini korisnom za modele gdje je potrebno predviđati vjerojatnosti. Međutim, može uzrokovati probleme sa *zasićenjem* gradijenata, što otežava treniranje dubokih mreža.

Zasićenje gradijenata (vasnishing gradients) problem se događa prilikom propagiranja unatrag (backpropagation) gdje gradijenti, množeći se malim brojevi, nestaju dok prolaze sve dublje i dublje kroz slojeve. To dovodi da promjene težine postaju zanemarive što rezultira izrazito sporom, ako ikakvom učenju.

ReLU (Rectified Linear Unit)

ReLU je postala vrlo popularna zbog svoje jednostavnosti i učinkovitosti u treniranju dubokih mreža. Definira se kao:

$$\mathrm{ReLU}(x) = \max(0, x) \tag{2.3}$$



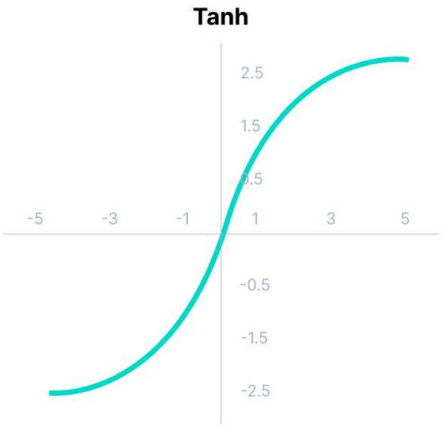
ReLU funkcija ima prednost nad sigmoidnom funkcijom jer nema problem zasićenja za pozitivne ulazne vrijednosti. Međutim, može biti sklona problemu *mrtvih neurona*, gdje neuroni ostanu zaglavljeni na izlazu 0. To dalje ima posljedice prilikom računanja back-propagationa zbog deaktivacije i pasiviziranja neurona.

Jedna od prednosti ove metode je konvergencija gradijentna prema globalnom minimumu funkcije gubitka

Tanh funkcija

Tanh funkcija je hiperbolička tangenta, definirana kao:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.4}$$



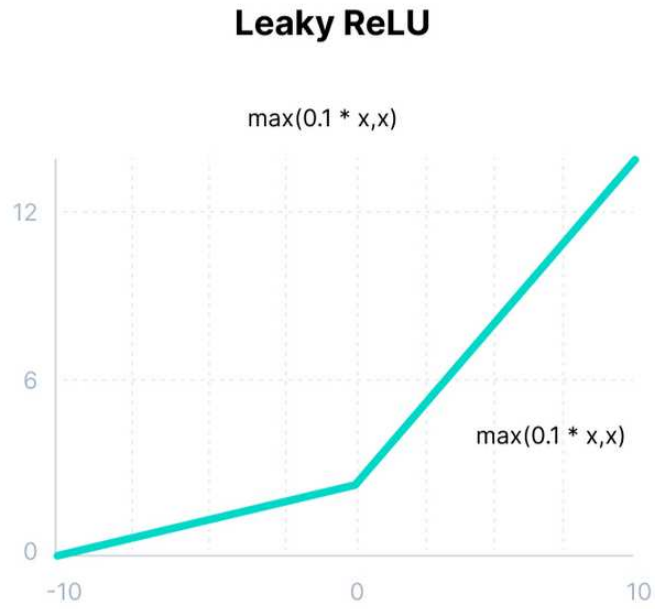
Ona transformira ulazne vrijednosti u interval (-1, 1). Tanh funkcija često se koristi kao alternativa sigmoidnoj funkciji jer se srednje vrijednosti bliže 0, što može pomoći u bržem treniranju.

Leaky ReLU

Leaky ReLU je varijacija ReLU funkcije koja pokušava riješiti problem mrtvih neurona dopuštajući malom nagibu da prolazi kroz, čak i kada je ulaz negativan:

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{ako je } x > 0 \\ \alpha x & \text{inače} \end{cases} \tag{2.5}$$

gdje je α mali broj, obično 0.01.



Osim što ne rješava konzistentno problem mrtvih neurona, jedan od limitirajućih faktora je i taj što pristup rješavanju različitih problema uvelike ovisi o izboru parametra. Samim time, potrebno je biti posebno osjetljiv i prema prirodi problema koji se promatra.

Softmax funkcija

Softmax funkcija se koristi u izlaznim slojevima neuronskih mreža za klasifikacijske zadatke s više klasa. Transformira skup ulaznih vrijednosti u vektor vjerojatnosti:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.6}$$

Softmax osigurava da su sve izlazne vrijednosti u intervalu (0, 1) i da zbroj svih vrijednosti iznosi 1, što ih čini prikladnima za interpretaciju kao vjerojatnosti.

Prednosti i nedostaci različitih aktivacijskih funkcija

Svaka aktivacijska funkcija ima svoje prednosti i nedostatke:

- **Sigmoidna funkcija** može uzrokovati problem ”exploding/vanishing gradients”, što otežava treniranje dubokih mreža.
- **ReLU** je jednostavna i učinkovita, ali može dovesti do problema mrtvih neurona.

- **Tanh funkcija** nudi bolje performanse od sigmoidne funkcije, ali i dalje može patiti od problema zasićenja.
- **Leaky ReLU** pokušava riješiti problem mrtvih neurona dopuštajući malom nagibu da prolazi kroz negativne ulazne vrijednosti.
- **Softmax funkcija** je idealna za klasifikacijske zadatke jer pretvara izlazne vrijednosti u vjerojatnosti.

Izbor aktivacijske funkcije ovisi o specifičnom problemu i arhitekturi mreže te često zahtijeva eksperimentiranje kako bi se pronašla najprikladnija funkcija za određeni zadatak.

Univerzalni aproksimacijski teorem

Vratimo se ukratno na problem egzistencije rješenja i opravdanosti uvođenja aktivacijske funkcije. Sljedeći rezultat govori upravo o njezinoj nuznosti.

Univerzalni aproksimacijski teorem navodi da neuronska mreža s jednim skrivenim slojem može aproksimirati bilo koju kontinuiranu funkciju na kompaktnom skupu, pod uvjetom da ima dovoljno neurona. Matematički gledano, to znači da za svaku neprekidnu funkciju f i za svaki $\epsilon > 0$, postoji neuronska mreža s konačnim brojem neurona koja može aproksimirati f sa greškom manjom od ϵ.

Navodimo osnovni rezultat koji se dalje moze poopciti na šire klase funkcija koje odgovaraju potrebama dubokog učenja.

Teorem 2.1.1 (Cybenkov univerzalni aproksimacijski teorem)**.** *Neka je $I^d = [0,1]^d \subset \mathbb{R}^d$ jedinična kocka i $C\left(I^d\right)$ skup neprekidnih funkcija na I^d s normom $\|f\|_\infty = \sup_{x \in I^d} |f(x)|$.*

Također, neka je $f \in C\left(I^d\right)$. Za svaki $\epsilon > 0$ postoji funkcija g oblika

$$g(\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i \sigma\left(\boldsymbol{w}_i^T \boldsymbol{x} + b_i\right),$$

s težinama $w_i \in \mathbb{R}^d$ i $\alpha_i, b_i \in \mathbb{R}$ i proizvoljnom nepolinomijalnom funkcijom σ, tako da

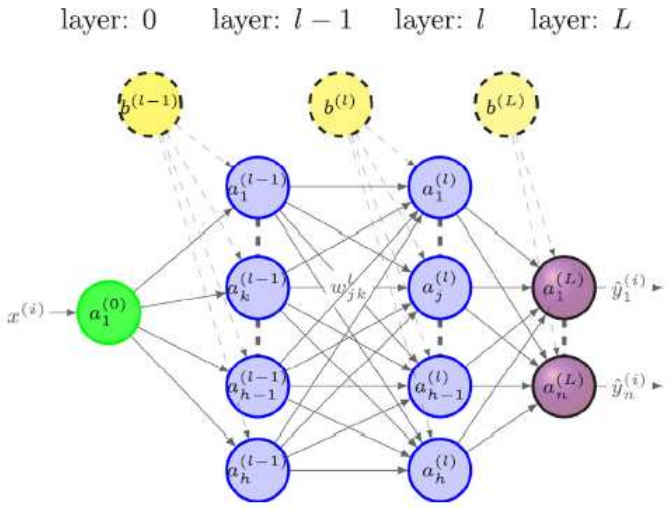
$$\|f - g\|_\infty < \epsilon$$

Teorem kaže da je linearni podprostor $C\left(I^d\right)$ razapet funkcijama oblika $\sigma\left(\boldsymbol{w}^T \boldsymbol{x} + b\right)$ gust u $C\left(I^d\right)$. Kako funkcija σ može biti proizvoljna nepolinomijalna funkcija, obuhvaćene su aktivacijske funkcije koje se koriste u neuronskim mrežama.

Napomena 2.1.2. *Iako teorem garantira egzistenciju rješenja, sama konstrukcija i pronalaženje ostaje izvan njegove sfere. Taj aspekt problema translatira se na odabir tipa neuronske mreže, njezinu arhitekturu, kao i izbor funkcije gubitka te treniranje s čime se bavimo u nastavku.*
Za detaljnije rezultate, kao i dokaze tvrdnji zainteresirani čitatelj se posebno upućuje na [18], odakle su prethodno prezentirani isključivo najosnovniji rezultati.

Unaprijedne ili feedforward mreže

Feedforward neuronske mreže (FFNN) su najjednostavniji tip neuronskih mreža gdje informacije teku samo u jednom smjeru - od ulaznog sloja prema izlaznom, kroz jedan ili više skrivenih slojeva. Svaki neuron u jednom sloju povezan je sa svakim neuronom u sljedećem sloju.



Slika 2.1: Potpuno povezana unaprijedna neuronska mreža FFNN

Osim za opću predodžbu neuronskih mreža i njenih osnovnih koncepata, iskoristimo prethodni primjer za uvod u različite tipove neuronskih mreža, odnosno do potrebe za njihovim razvojem.

Prilično je jasno promatrajući prikazanu strukturu da je moguće primjeniti svakojake modifikacije na mreži; od mjestimičnih promjena aktivacijske funkcije; dodavanje, odnosno redefiniranja veza među neuronima pa do uvođenja novih parametara i slično.

Sve te promjene imaju za svrhu unaprijeđenja preciznosti i efikasnosti pri dobivanju rješenja.

Tako nastaju različiti tipovi neuronskih mreža. Neki od poznatijih primjera su reukuretna, konvolucijska ili u ovom radu od posebnog interesa - fizikom informirana neuronska mreža.

Treba imati na umu da ne postoje uvijek striktne granice među modelima, odnosno postoje modeli koji kombiniraju više arhitektura. Primjerice, ponekad je model spoj nekoliko različitih stukrura, kao u slučaju fizikom informiranih mreža koje su nama od posebnog interesa. Njihov kostur može biti bilo koja od prethodno spomenutih vrsta, dok modifikacije na dijelu od interesa su one koje ju čine izuzetnom.

Riječ o ulogama inheretnih struktura u prethodno spomenutom slučaju biti će u idućem poglavlju ili se može nešto detaljnije pogledati poglavlje 2.1 [10].

No Free Lunch teorem

Teorem "no free lunch" za strojno učenje tvrdi da, kada uzmemo u obzir sve moguće načine generiranja podataka, svaki algoritam za klasifikaciju ima istu prosječnu pogrešku prilikom klasifikacije novih podataka. Drugim riječima, nijedan algoritam strojnog učenja nije univerzalno bolji od drugih. Čak i najsofisticiraniji algoritam postiže isti prosječni rezultat kao i najsimplističkija metoda koja pretpostavlja da sve točke pripadaju istoj klasi.

Međutim, ovaj teorem vrijedi samo kada uzmemo u obzir sve moguće načine generiranja podataka. Ako pretpostavimo određene vrste distribucija koje su karakteristične za stvarne aplikacije, tada možemo stvoriti algoritme učenja koji će biti uspješniji na tim specifičnim distribucijama.

Stoga, cilj istraživanja u području strojnog učenja nije pronaći jedan univerzalni algoritam koji će uvijek biti najbolji, već razumjeti koje vrste distribucija podataka su relevantne za stvarni svijet te razviti algoritme koji će na tim podacima postići najbolje rezultate. [12]

Funkcije gubitka i treniranje

Cilj svake neuronske mreže je dobiti izlaz za dane ulazne podatke. Ne treba posebno isticati da je naglasak na što preciznijem rezultatu uz minimalan trošak resursa. Prvo pitanje koje se tu nameće je metodologije mjerenja kvalitete tog izlaza.

Osim međusobnog rangiranja različitih rezultata, ostaje i otvoreno pitanje kada stati, odnosno koji rezultat je dovoljno dobar rezultat. Za tu svrhu tradicionalno se koristi posebna klasa funkcija - funkcije gubitka.

Funkcija gubitka (*loss function*) mjeri koliko dobro neuronska mreža predviđa ciljne vrijed-

nosti. Za dane ulazne podatke, njen izlaz je najčešće realan broj koji predstavlja udaljenost rezultata od egzaktno dostupnih podataka.

Primjeri funkcija gubitka

Funkcije gubitka mogu poprimiti razne oblike u ovisnosti na promatrani prostor podatka i korištenu regularizaciju. Osnovni primjeri funkcija gubitka uključuju:

- **Mean Squared Error (MSE)**

  - $MSE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$

  - koristi se uglavnom za regresijske zadatke

- **Cross-Entropy Loss**

  - $CE(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{N} y_i \log \hat{y}_i$

  - koristi se uglavnom za klasifikacijske zadatke.

U skladu s prethodno spomenutim, postoji cijela klasa funkcija gubitka. Konkretan odabir ostaje na izbor i predstavlja izazov sam po sebi. Ne postoji univerzalan odabir koji odgovara svim situacijama, već se odabire u odnosnu na podatke, arhitekturu i problem koji se rješava.

Učenje mreža

Usko vezan uz funkciju gubitka je i pojam treninga ili učenja neuronske mreže. Cilj treniranja je minimizirati funkciju gubitka pomoću algoritama optimizacije.

Učenje se dijeli u dvije osnovne kategorije:

Nadzirano učenje

Nadzirano učenje (supervised learning) je pristup strojnog učenja gdje se model trenira na skupu podataka koji sadrži ulazno-izlazne parove. Svaki ulaz $\mathbf{x}_i$ ima odgovarajući izlaz y_i. Cilj je naučiti funkciju f koja mapira ulaze na izlaze $y = f(\mathbf{x})$.

Matematički, imamo skup podataka $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, gdje je $\mathbf{x}_i$ ulazni vektor, y_i odgovarajući izlaz, a N broj uzoraka. Model se trenira kako bi minimizirao funkciju gubitka L, koja mjeri razliku između predviđenog izlaza $\hat{y}_i$ i stvarnog izlaza y_i:

$$\min_{\theta} \sum_{i=1}^{N} L(f(\mathbf{x}_i; \theta), y_i)$$

gdje je θ skup parametara modela.

Primjeri nadziranog učenja uključuju:

- **Klasifikacija:** Dodjeljivanje ulaznih podataka jednoj od unaprijed definiranih klasa, npr. prepoznavanje rukom pisanih znamenki ili klasifikacija e-mailova kao spam ili ne-spam.
- **Regresija:** Predviđanje kontinuirane vrijednosti, npr. predviđanje cijena nekretnina ili temperature.

Nenadzirano učenje

Nenadzirano učenje (unsupervised learning) je pristup strojnog učenja gdje model uči iz podataka koji nisu označeni. Cilj je pronaći skrivene strukture ili obrasce unutar podataka.

Matematički, imamo skup podataka $\{\mathbf{x}_i\}_{i=1}^{N}$ bez odgovarajućih izlaza. Model traži strukture unutar podataka koje minimiziraju ili maksimiziraju određeni kriterij.

Primjeri nenadziranog učenja uključuju:

- **Klasteriranje:** Grupiranje podataka u klastere na temelju sličnosti. Jedan od najpoznatijih algoritama je k-sredina (k-means), koji dijeli podatke u k klastera minimizirajući udaljenost unutar klastera:

  $$\min \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} \|\mathbf{x}_j - \mu_i\|^2$$

  gdje je C_i skup podataka u klasteru i, a μ_i centroid klastera.

- **Smanjenje dimenzionalnosti:** Smanjivanje broja varijabli u podacima zadržavajući što više informacija. Metode uključuju Principal Component Analysis (PCA), koja projicira podatke u nižu dimenzionalnu prostoriju tako da se maksimalno očuva varijanca podataka:

  $$\max \|W^T X\|^2 \quad \text{s.t.} \quad W^T W = I$$

  gdje je X matrica podataka, a W matrica projektiranja.

Nenadzirano učenje omogućuje modelima da otkriju skrivene obrasce bez potrebe za označenim podacima, što je posebno korisno kada označavanje podataka nije praktično ili je preskupo.

Prilikom bilo kakvog učenja, ono nad čime imamo kontrolu jesu parametri u vidu težina w i pristranosti b. Obzirom da su oni ti koji najznačajnije utječu na odnose pojedinih, odnosno sačinjavaju srž samih neurona, od glavnog interesa je promatranje promjene funkcije gubitka po tim istim paramterima.

Imajući jos u vidu njihov izniman broj, esecijalan je efikasan izračun takvih gradijenata - čija suprotna vrijednost predstavlja smjer najbržeg pada promatrane funkcije za danu vrijednost. Na taj način problem optimizacije i pronalaska minimuma postaje centralan. Tu na scenu stupa backpropagation algoritam.

Backpropagation

Backpropagation je algoritam za treniranje neuronskih mreža koji koristi gradijentnu optimizaciju za minimizaciju funkcije gubitka.

Za definiranu mrežu i funkciju greške, povratno širenje omogućava računanje gradijenta te funkcije u odnosu na težine mreže. Sam naziv dolazi iz procesa računanja gradijenta koji se odvija najprije izračunavanjem gradijenta konačnog sloja težina, nakon čega slijedi postupno računanje gradijenata slojeva unazad, sve do prvog sloja gdje se proces u suštini svodi na *lančano pravilo* derivacije.

Njegova efikasnost slijedi iz činjenice da djelomični izračuni gradijenta jednog sloja ponovno se koriste u izračunima gradijenta za prethodni sloj.

Ovaj povratni tok informacija o grešci omogućava učinkovit izračun gradijenta na svakom sloju, za razliku od naivnog pristupa koji bi zahtijevao zasebno računanje gradijenta za svaki sloj. Povratno širenje tako značajno smanjuje računalnu složenost procesa treniranja neuronskih mreža, omogućujući njihovu primjenu u složenim zadacima strojnog učenja.

Ključni koraci uključuju:

1. **Forward pass**: izračunavanje predikcija mreže.
2. **Izračunavanje gubitka**: korištenjem funkcije gubitka.
3. **Backward pass**: računanje gradijenata gubitka u odnosu na težine.
4. **Ažuriranje težina**: korištenjem optimizacijskog algoritma.

Optimizacija: ADAM metoda

Svrha optimizacijskog algoritma je ubrzavanje konvergencije i poboljšanje procesa treniranja. Prisjetimo se treniranje je proces pronalaska parametara za koje promatrana funkcija gubitka poprima minimum.

Pažljivi čitatelj tu primjećuje dva osnovna pojma: vrijednost funkcije gubitka i parametre, preciznije smanjenje gubitka promjenom parametara. Preduvjet za postojanje bilo kakve neuronske mreže je efikasno računanje potrebnog gradijenta. Obzirom na kompleksnost izračuna koja je reda veličine izvrjednjavanja funkcije, taj uvijet je zadovoljen.

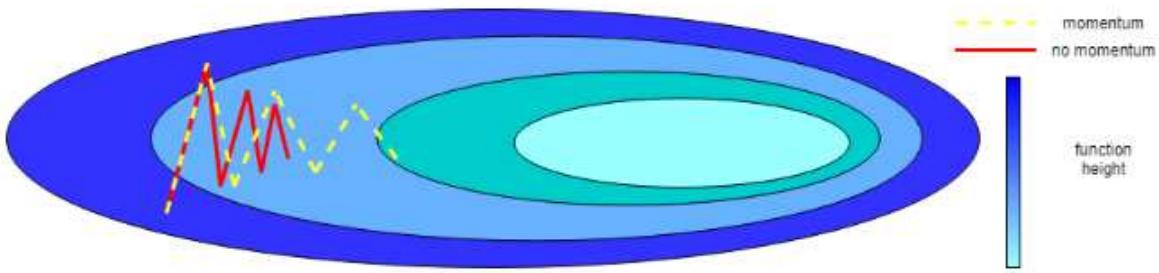
Promjenom parametara u smjeru negativnog gradijenta minimizira se funkcija gubitka što je ujedino osnovni cilj procesa. Prevelik korak u tom smjeru može dovesti do preskakanja rješenja, dok premali usporava proces konvergencije. Dakle, nije cilj isključivo pronalazak optimalne vrijednosti, nego i što bržeg puta do nje.

ADAM (*Adaptive Moment Estimation*) je popularna metoda optimizacije koja kombinira prednosti dviju drugih metoda: AdaGrad i RMSProp. Od AdaGrad-a je preuzeta sposobnost suočavanja s rijetkim gradijentima, dok od RMSProp-a prilagodljivost promjenjivim stopama učenja temeljenim na prosječnim veličinama prethodnih gradijenata. Adaptivna procjena momenta koristi prve i druge trenutke gradijenata za prilagodbu koraka učenja po idućem algoritmu iz [14]:

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Ključna značajka ADAM-a je prilagodljiva stopa učenja. U usporedbi s tradicionalnim gradijentnim metodama, ova prilagođava stopu učenja svakog parametra ponaosob na temelju učestalosti njihovog ažuriranja. Ažuriranje parametara se odvija pojedinačno i u ovisnosti o prethodno definiranim varijablama. Pod time se misli da trenutna stopa učenja ovisi o prethodnim vrijednostima koje su promatrani parametri poprimili za vrijeme prethodnih iteracija. Na taj način ostvaruje se djelotvorniji učinak na učenje i osnažuje se ponašanje mreže kao jedinstvene cjeline.



Veće stope prilikom učenja mogu dovesti do brže konvergencije što nosi rizik preskakanja optimalne vrijednosti. S druge strane, manje osiguravaju stabilnost pri konvergenciji uz moguću cijenu predugog vremena ili zapinjanja u lokalnim ekstremima. Idealno - prilagodljiva stopa obzirom na prostor potrage.

Ne ulazeći rigorozno u matematičke ograde, ovakva uvjetovanost prethodnim iskustvima vodi do ograničavanja prostora potrage i stvaranja okoline od povjerenja. Daljnji korak upućuje prema najizglednijoj okolini cilja. Za konkretnije brojeve i stop konvergencije, upućuje se na već spomenuti rad [14].

Korištenjem prvog i drugog momenta gradijenta, ADAM ima svojstvo prilagodljivosti koje dovodi učinkovitije i djelotvornije optimizacije. Time zadržava jednostavnost implementacije istovremeno odgovarajući adekvatno na problem veličine stope učenja i prikazane loše kondicioniranosti (čestih promjena smjera).

Razlozi zbog kojih ADAM je danas jedna od najčešće odabranih metoda optimizacije:

1. **Treniranje velikih modela**

   - Pogodan za modele s velikim brojem parametara; niski memorijska zahtjevi

2. **Brzina konvergencije**

   - Zahvaljujući prilagodljivom učenju, osigurava bržu konvergenciju u usporedbi sa stohastickim gradijentnim spustom

3. **Rad s rijetkim podacima**

   - Učinkovit u radu s podacima koji dovode do rijetkih gradijenata ili dubokim slojevima

4. **Online i batch treniranje**

5. **Robustan i jednostavan algoritam za implementaciju**

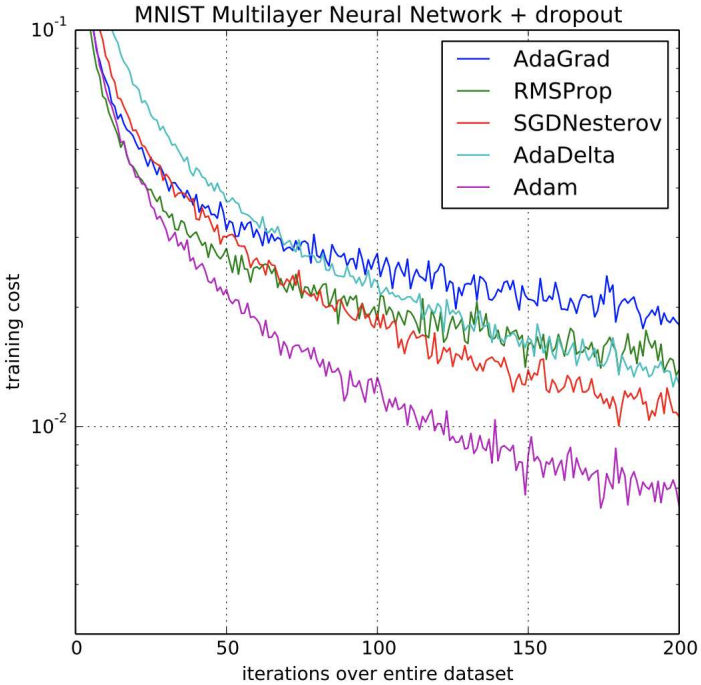
Dodatno, neka ograničenja koje treba imati na umu:

1. **Izbor i upotreba hiperparametara**

2. **Osjetljivost na outliere u podacima**

3. **Izbor funkcije gubitka**

4. **Računalna zahtjevnost i prepreke**



Slika 2.2: Usporedba različitih optimizacijskih metoda preuzeta iz [14]

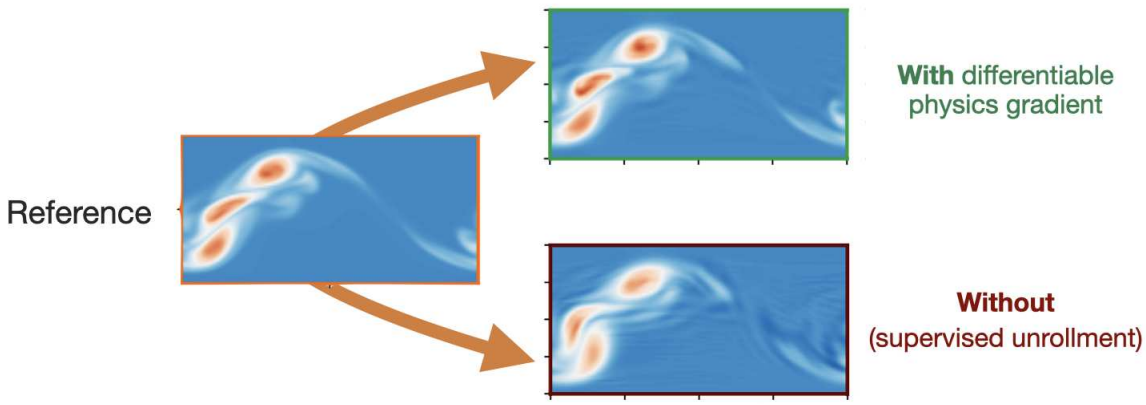
2.2 Physics Informed Neural Networks

Fizikom informirane neuronske mreže (PINNs) su posebna vrsta neuronskih mreža koje inkorporiraju fizičke zakone u obliku parcijalnih diferencijalnih jednadžbi (PDEs) unutar strukture mreže. Ovo omogućuje mreži da koristi ne samo podatke već i temeljne fizičke principe za učenje i predikciju.

Klasična paradigma strojnog učenja je da kroz proces treniranja mreža analizirajući podatke pronalazi obrasce i uzorke na temelju kojih daje predikcije. Točnost predikcije time ovisi o količini i kvaliteti podataka. Za razliku od tradicionalnih neuronskih mreža koje se oslanjaju isključivo na podatke, PINNs integriraju fizičke zakone direktno u proces učenja, omogucujući im da iskoriste stručno znanje i poboljšaju predikcije u situacijama gdje su podaci rijetki ili skupi za prikupljanje.

Drugim riječima, ukoliko su neke zakonitosti o promatranom fenomenu već poznate, moguće ih je usaditi u model čime se proces učenja usmjerava i olakšava smanjujući potrebu za početnim podacima. Jasno je da u situaciji gdje postoji dovoljno velik skup podataka, mreža bi učenjem prepoznala te zakonitosti te ih inkorporirala u rješenje, no zbog prirode problema to nije uvijek moguće.

PINNs zaobilaze manjkavosti uzrokovane količinom i kvalitetom podataka optimizirajući posebno osmišljenu funkciju gubitka. Uz standardnu minimizaciju reziduala, funkcija gubitka vrednuje i željeno, odnosno zahtjevano ponašanje koje opisuje promatrani diferencijalni sustav. To se ponašanje analizira u nastavku.



Metodologija

Podsjetimo se ukratko zapisa problema diferencijalnih jednadžbi. U najopćenitijoj formi promotrimo sustav parcijalnih diferencijalnih jednadzbi:

$$\begin{aligned} \mathcal{F}(\boldsymbol{u}(z);\gamma) &= \boldsymbol{f}(z) \quad z \text{ in } \Omega \\ \mathcal{B}(\boldsymbol{u}(z)) &= \boldsymbol{g}(z) \quad z \text{ in } \partial\Omega \end{aligned} \tag{2.7}$$

gdje su $\Omega \subset \mathbb{R}^d$ i $\partial\Omega$ njegov rub, a $z = (\boldsymbol{x}, t)$ zajednički prikaz *prostorne* komponente dimenzije $d-1$ i jednodimenzionalne vremenske varijable.

Funkcija $\boldsymbol{u}$ označava traženo nepoznato rješenje, a funkcije $\boldsymbol{f}$ i $\mathcal{F}$ su redom funkcija zadanog problema i nelinearni diferencijalni operator. Operator $\mathcal{B}$ predstavlja i rubne i inicijalne uvjete, dok je $\boldsymbol{g}$ zadana pripadajuća funkcija.

Kao i u svakoj zadaći, cilj je pronaći funkciju $\boldsymbol{u}$ za neke odredene parametre γ, odnosno u inverznom problemu za dane podatke potrebno je odrediti i parametare γ.

Ono što neuronska mreža čini jest aproksimira rješenje

$$\hat{\boldsymbol{u}}_\theta(z) \approx \boldsymbol{u}(z)$$

gdje $\hat{\boldsymbol{u}}_\theta(z)$ predstavlja funkciju aproksimacije s parametrizacijom θ koju odreduje mreža. Upravo zbog pažljive konstrukcije neuronskih mreža i garancija univerzalnog aproksimacijskog teorema, proces se svodi na pronalazak odgovarajućih parametara θ.

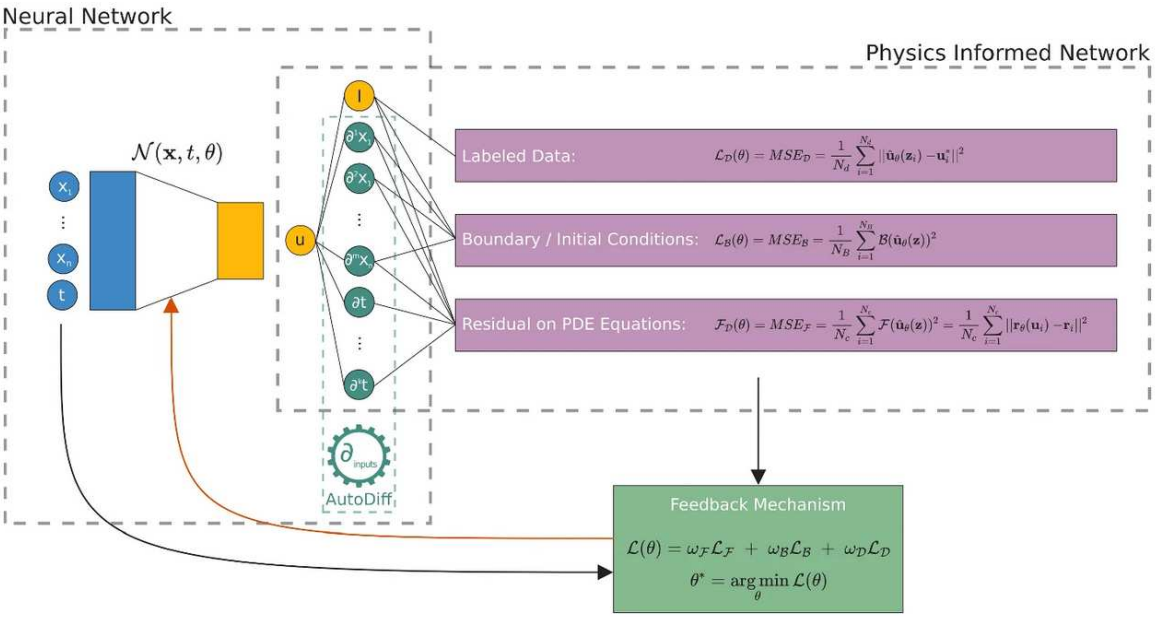
Kod gotovo svake neuronske mreže tražene parametre θ pronalazimo minimizirajući funkciju gubitka, odnosno rješavajući:

$$\theta^* = \arg\min_\theta \mathcal{L}(\theta)$$

Posebnost PINNs se očituje upravo u specifičnoj funkciji gubitka $\mathcal{L}$ za koju vrijedi:

$$\mathcal{L}(\theta) = \omega_{\mathcal{F}}\mathcal{L}_{\mathcal{F}}(\theta) + \omega_{\mathcal{B}}\mathcal{L}_{\mathcal{B}}(\theta) + \omega_{\mathcal{D}}\mathcal{L}_{\mathcal{D}}(\theta), \tag{2.8}$$

a sastoji se redom od komponenti diferencijalne jednadžbe $\mathcal{L}_{\mathcal{F}}$, graničnih uvjeta $\mathcal{L}_{\mathcal{B}}$ i opaženih podataka $\mathcal{L}_{\mathcal{D}}$ uz odgovarajuće težine.



Slika 2.3: Glavni elementi fizikom inspirirane neuronske mreže

Bazna neuronska mreža

Prva prikazana komponenta je neuronska mreža čiji je zadatak za dani ulaz napraviti predikcijsku funkciju. U prethodnom su poglavlju objašnjeni svi osnovni mehanizmi koji se ovdje odvijaju.

Valja ponoviti da bi odabir neuronske mreže uvijek trebao ovisiti o tipu problema i podacima kojima se raspolaže, što nalaze sam teorem *besplatnog ručka*. Samim time, imajući na umu da je cijelo područje PINNs novo i neistraženo u potpunosti, veliki broj trenutnih istraživanja upravo otpada na optimalan odabir aktivacijskih funkcija kao i čitave arhitekture mreže.

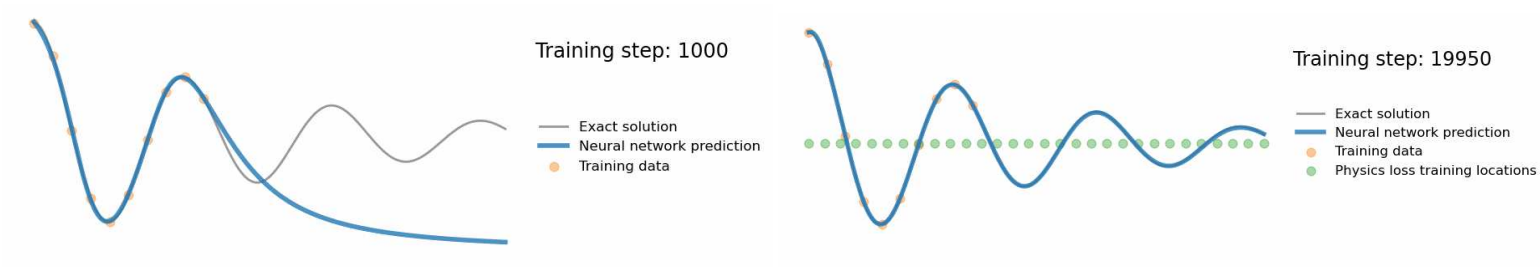
Najpopularnije su trenutno potpuno povezane unaprijedne neuronske mreže FFNN, konvolucijske CNNs, rekurentne RNNs, kao i nešto novije Auto-Encoder AE, Deep Belief DBN, Generative Adversarial Network GANs i Bayesian Deep Learning BDL i ostale. Za pregled liste istraživanja koji proučavaju svaku pojedinu upućuje se na Tablicu 1 [10].

Najutjecajnije istraživanje PINNs koje obrađuje stabilnost i konvergenciju u slučajevima od interesa ovog rada je [26]. Obzirom na prezentirane empirijske dokaze FFNN se nametnula kao trenutno najplodonosnija i samim time odabrana kao bazna za kasniji eksperiment.

Fizikom informirana mreža i povratni mehanizam

Preostale komponente na slici odnose se na dio svojstven fizikom inspiriranim neuronskim mrežama. Optimizacijski problem pronalaska parametra θ svodi se aproksimaciju diferencijalnih jednadžbi minimizirajući (2.8) po uputama povratnog mehanizma.

Toliko jednostavna, a zapravo genijalna ideja omogućuje svođenje rjesavanja zadace (2.7) na (nekonveksni) optimizacijski problem. Takva transformacija uglavnom eliminira jedninstvenost rješenja koja je svojstvena prilikom korištenja klasičnih numeričkih pristupa. S druge strane, dobiveno rješenje je zatvorene forme te također diferencijabilno koristeći analitičke gradijente.



(a) Obična neuronska mreža (b) Fizikom informirana neuronska mreža

Slika 2.4: Trenutak postizanja konvergencije

Fundamentalna značajka koja omogućuje efikasnu implementaciju fizičkih zakonitosti je automatska diferencijacija (AD). Njezina uloga je izračun sastavnih dijelova i povezivanje neuronske mreže s fizikom informiranom mrežom koje povratni mehanizam koristi pri konačnoj procjeni. U nastavku, analizom glavnih odrednica funkcije gubitka konkretiziramo njenu važnost. Za općenitije mehanizme djelovanje i teorijsku podlogu zainteresirani čitatelj se upucuje na [20], [7] ili srodnu literaturu obzirom na sveprisutnost algoritma.

Sastavnice funkcije gubitke $\mathcal{L}$

Tri su osnovne komponente prikazane u (2.8). Redom to su greške koje opisuju reziduale diferencijalnog sustava; reziduali graničnih i inicijalnih uvjeta i konačno odstupanje od empirjiski poznatih vrijednosti.

Komponenta $\mathcal{L}_{\mathcal{F}}$ predstavlja najzanimljivi dio, odnosno onaj koji implementira fizičke zakone. Njezin doprinos leži u gubitku koji proizlazi iz odstupanja dobivene predikcije $\hat{\boldsymbol{u}}_\theta(z)$, preciznije njene derivacije i zadanog sustava $\boldsymbol{f}$.

Tipična implementacija je oblika:

$$\mathcal{L}_{\mathcal{F}}(\theta) = MSE_{\mathcal{F}} = \frac{1}{N_C} \sum_{i=1}^{N_C} \|\mathcal{F}(\hat{\boldsymbol{u}}_\theta(z_i)) - \boldsymbol{f}(z_i)\|^2$$

Vrednovanje se odvija u N_C proizvoljno odabranih točaka domene Ω. Odabir istih je najčešće nasumičan, no njihova gustoća i povezanost s poznatim podacima utječe na konvergeciju i preciznost rješenja.

Važnost automatske diferencijacije se posebno očituje u ovom dijelu računajući potrebne derivacije u točkama interesa funkcije $\hat{\boldsymbol{u}}_\theta(z)$. Brzina i preciznost AD ne dolazi do izražaja samo u konačnoj formi rezultata, već omogućuje bilo kakav trening koji je ovisan o višestrukom uspješnom izračunu traženih derivacija. Upravo ovo svojstvo omogućuje pronalazak rješenja bez dirketnog numeričkog ili bilo kakvog drugog *rješavanja* diferencijalnog sustava.

Impresivan dio je zapravo sposobnost generiranja čitavog stupca Jacobijeve matrice u jednom prolasku kroz mrežu. Uz to, obrnuta automatske diferencijacije analogno ima mogućnost izračuna jednog retka matrice, što se pokazuje esecijalnim u određenim situacijama, a posebno kod velikih sustava i nerazmjernog broja ulaznih i izlaznih varijabli. Na taj način moguće je dobiti i rješenje inverznog problema koristeći istu funkciju gubitka uz nikakve ili minimalne modifickacije.

Unatoč optimističnim rezultatima koji se postižu korištenjem AD-a, treba biti oprezan. Bez obzira na inovativnost algoritma, njegove simplističke odrednice mogu biti izvor problema kod konvergencije rezultata. Uvođenjem i kombiniranjem elemenata numeričkog deriviranja, moguće je ne samo ubrzati proces treninga, vec pospješiti same rezultate na što ukazuje Chiu et al (2022) [9].

Općenito, velika preciznost rješenja može se postići povećavanjem N_C obzirom da automatska diferencijacija u suštini ni na koji način ne prati promatrani diferencijalni sustav, već naprotiv isključivo koristi točke interpolacije. Numeričke metode čine baš to, uzimaju u obzir diferencijalni sustav i njegova svojstva ne vodeći se isključivo optimizacijskim i drugim tehnikama. Stoga, njihovom ugradnjom u automatsku diferencijaciju ostvaruju se bolji i robusniji rezultati. Naravno, prostor za unaprijeđenja i daljnja istraživanja je izuzetan.

Preostale dvije komponente dobivaju se iz poznatih podataka. Kada pričamo o inicijalnim i graničim uvjetima pripadajuća funkcija gubitka najčešće poprima oblik:

$$\mathcal{L}_{\mathcal{B}}(\theta) = MSE_{\mathcal{B}} = \frac{1}{N_B} \sum_{i=1}^{N_B} \|\mathcal{B}\left(\hat{\boldsymbol{u}}_\theta(z_i)\right) - \boldsymbol{g}\left(z_i\right)\|^2$$

Jasno je vidiljivo da ovaj dio regulira utjecaj na odstupanje od poznatih normi. Ponekad je moguće, a primjerice bit će korišteno u kasnijem modelu, *a priori* ugradnjom početnih uvjeta u model eliminirati ovaj dio greške - iznosi 0.

Takva provedba naziva se *hard constraint*, dok suprotno *soft constraint* predstavlja nešto labaviji pristup. Njime se omogućuje slobodnija potraga optimalnih parametara, no nedostatak je što nije osigurano poštivanje zadanih uvjeta uz dodatno neistražen utjecaj koji odstupanje od postavljenih uvjeta ima na prostor i smjer potrage raspoloživih rješenja.

Posljednji dio je onaj koji se nalazi u većini klasičnih funkcija gubitaka:

$$\mathcal{L}_{\mathcal{D}}(\theta) = MSE_{\mathcal{D}} = \frac{1}{N_d} \sum_{i=1}^{N_d} \left\|\hat{\boldsymbol{u}}_\theta\left(z_i\right) - u_i^*\right\|^2$$

Ovaj dio predstavlja odstupanje dobivenog rješenja za empirijske podatke.
Iako je prethodno naglašena efikasnost PINNs za mali broj takvih podataka, njihovu važnost nikako ne treba zanemariti. Veći broj podataka pospješuje procjenu i omogućuje efikasnije izvlačenje više značajki iz modela time osiguravajući stabilniju i precizniju predikciju.

Dodatno, istraživanja poput Wang et al (2022a) [33] pokazuju da PINNs nakupljaju greške upravo udaljavajući se od poznatog (početnih uvjeta) prema nepoznatom.
Predlaže se da bi algoritmi trebali biti usklađeniji u odnosu prema načinu širenja informacija i skladu temeljnim pravilima razvoja sustava. Time značajno raste stabilnost te konvergencija modela i u kaotičnim i glatkim područjima.

Primjene PINNs

PINNs su primjenjivi u širokom spektru znanstvenih i inženjerskih problema, uključujući:

- **Mehanika fluida**: Rješavanje Navier-Stokesovih jednadžbi za modeliranje strujanja fluida i opcenito problemima tokova
- **Toplinska dinamika**: Simulacija prijenosa topline pomoću Fourierovih jednadžbi.
- **Elektromagnetizam**: Rješavanje Maxwellovih jednadžbi za simulaciju elektromagnetskih polja.
- **Kvantna mehanika**: Predikcija ponašanja kvantnih sustava koristeći Schrödingerovu jednadžbu.

Prethodno su navedeni samo neke od primjena, no obzirom na dosadašnje rezultate te činjenicu da je područje rastuće i testira se na širokoj klasi problema, i više je nego dovoljno razloga za optimizam. Za potpuniju listu pogledati [10].

Budućnost PINNs

PINNs nude nekoliko ključnih prednosti:

- **Inkorporacija stručnog znanja**: Integriranjem fizičkih zakona, odnosno prethodno poznatih pravilnosti, PINNs mogu bolje generalizirati iz manjih količina podataka.
- **Preciznije predikcije**: Poštujući poznate fizičke zakone, PINNs proizvode fizički konzistentne i realistične predikcije.
- **Smanjenje potrebe za podacima**: PINNs smanjuju potrebu za velikim količinama podataka jer koriste fizičke principe za vođenje procesa učenja.

Dodatno, neke od prednosti PINNs u odnosu na standardne metode su neovisnost o izboru točaka prilikom učenja te iznimno bitno - diferencijablna rješenja izražena u zatvorenoj formi. Tu treba nadodati i mogućnost jednako efikasnog rješavanja inverznog problema te sposobnost kontrole nad kompleksnim geometrijama. Povrh navedenog, pokazano je da se PINNs dobro nose s *prokletstvom dimnezije* i problemima pri radu s velikim dimenzijama upotrebom odgovarajuće arhitekture. O tome svjedoče radovi Siddhartha Mishre i Roberta Molinaro.

Vodeći se ovim benefitima, trenutno većina istraživanja pokušava personalizirati pojedine sastavne dijelove PINNs (arhitekturu, aktivacijske funkcije, funkciju gubitka, optimizacijske algoritme i ostalo) za vlastite *specifične* probleme. Većina poznatog oslanja se

isključivo na empirijske pokazatelje bez snažne teorijske podloge što ulijeva dodatnu dozu rezerviranosti i skeptičnosti dijela autora prema tehnologiji.

Ono što je najvažnije za razvoj PINNs, nedovoljno su zastupljena teorijska istraživanja vezana za generalnu stabilnost i konvergenciju metode. Tu treba spomenuti primjerice [29] koji ulazi detaljnije u spomenutu problematiku opisujući dovoljne uvjete konvergencije PINN za eliptičke paraboličke i linearne parcijalne diferencijalne jednadžbe.

Da nije sve idealno potvrđuje i istraživanje [16] koje ukazuje na probleme s konvergencijom za nekompleksne sustave. Ista istraživanja pokazuju da nedostaci ne proizlaze zbog manjkavosti neuronskih mreža općenito, već su svojstvena *soft contraints*-ima fizički informiranih neuronskih mreža.

Iako sam rad predlaže rješenje u vidu postepenog treniranja i tuniranja težinskih parametara funkcije gubitka te razbijanja domene i rješenja na manje, sekvencijalne intervale (niz više modela i rješenja koji se u konačnici povežu, no u suštini svodi se na razdvajanje jednog problema na više). Do sličnih zaključaka dolaze i drugi autori te svakako treba biti na oprezu.

Jedan od pionira u području i autor izvornih radova o PINNs upozorava da predložena metoda nikako ne bi trebala biti promatrana kao zamjena klasičnih numeričkih metoda. Obzirom na njihovu dugovječnost i dokazanost, one već zadovoljavaju sve dosadašnje potrebne standarde. Samu metodu PINNs treba promatrati u kontekstu puta prema eri znanstvenog računanja vođenog podacima. Istraživanja poput Chiu et al (2022) potvrđuju te slutnje Raissija o međusobnoj koegsitenciji i komplementarnosti ovih metoda.

Poglavlje 3

Primjena neuronskih mreža na obične diferencijalne jednadžbe

U prethodnim poglavljima prezentirane su teorijske pretpostavke potrebne za razumijevanje i rješavanje sustava diferencijalnih jednadžbi te uloge neuronskih mreža u tome. Smisao ovog rada je upoznavanje s područjem i rekreacija rezultata prikazanih u [11]. Dugogodišnja znanstvena istraživanja i doprinosi u području od strane Raissija et al. ([26], [24], [25]) daju više nego dovoljno razloga za upuštanje u dodatno ispitivanje rezultata.

Potrebno je razviti vektorizirani algoritam koji

3.1 Vektorizirani algoritam

Promotrimo najprije sustav običnih diferencijalnih jednadzbi:

$$\begin{aligned}
\tfrac{dy_1}{dt} &= f_1\left(t, y_1, y_2, \ldots, y_n\right) \\
\tfrac{dy_2}{dt} &= f_2\left(t, y_1, y_2, \ldots, y_n\right) \\
\vdots \qquad & \qquad \vdots \\
\tfrac{dy_n}{dt} &= f_n\left(t, y_1, y_2, \ldots, y_n\right)
\end{aligned}$$

na domeni $a < t < b$, uz inicijalne uvijete $y_1(a) = a_1, \ldots, y_n(a) = a_n$.

Odnosno kompaktnije,

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(t, \mathbf{y}), \quad \mathbf{y}(0) = \mathbf{y}_0$$

gdje je $\mathbf{y} = [y_1, y_2, \ldots, y_n]^T$ nepoznati vekor dimenzije $n \times 1$ i

$$\mathbf{f}(t, \mathbf{y}) = \begin{bmatrix} f_1(t, y_1, y_2, \ldots, y_n) \\ f_2(t, y_1, y_2, \ldots, y_n) \\ \vdots \\ f_n(t, y_1, y_2, \ldots, y_n) \end{bmatrix}$$

vektor zadanih vrijednosti funkcije iste dimenzije.

Rješavanju problema pristupamo stvaranjem potpuno povezane unaprijedne neuronske mreže FFNN koju dalje modificiramo našim potrebama.

Algoritam se sastoji od idućih dijelova:

1. *Ulazni podaci*

   Potrebno je odabrati m točaka na domeni (a, b) i kreirati matricu $X = [t_1, t_2, \ldots, t_m]$ dimenzije $1 \times m$.

   Odabrane točke smatraju se trening skupom za dani proces nenadzirano učenje. Za odabrane vrijednosti ne postoji nikakav targetirani i dodjeljeni skup rješenja.

   Zadani skup točaka označava interpolacijske vrijednosti koje se koriste u aproksimacijskoj funkciji. U nastavku detaljnije o implementaciji.

2. *Struktura neuronske mreže*

   Idući korak je određivanje broja skrivenih slojeva L, kao i broja pojedinih neurona h_l za svaki od njih. Izuzetak su ulazni sloj s jedim neuronom (ulazni podaci) i izlazni sloj veličine n koji predstavlja veličinu sustava.

3. *Inicijalizacija parametara*

   Označimo skup parametara s θ_j za $j = 1, 2, ..., n$ i uz $2 \leq l \leq L - 1$ definiramo matrice:

   - W_l dimenzije $h_l \times h_{l-1}$
   - b_l dimenzije $h_l \times 1$

   uz W_1 dimenzije $h_1 \times 1$.

   Za svaku od prethodnih matrica W_l, odnosno b_l također se incijaliziraju pripadajuće nul-matrice M_j^l i V_j^l koje predstavljaju odgovarajuće parametre metode procjene adaptivnog momenta.

4. *Forward propagation*

   Skrivene slojeve računamo po pravilu:

$$Z_l = W_l A_{l-1} + b_l$$

$$A_l = \sigma_l(Z_l)$$

   gdje σ_l predstavlja aktivacijsku funkciju sloja l uz $A_0 = X$. Za aktivacijsku funkciju u ovom slučaju empirijska testiranja upućuju na korištenje tangensa hiperbolnog.

   Konačno, izlazni sloj definira se bez upotrebe aktivacijske funkcije (empirijske studije i vlastita provjera) na nacin: $N(X, P_j) = Z_L$

   Označimo s $\boldsymbol{\mathcal{N}} \in \mathbb{R}^{n\times m}$ izlaznu matricu. $\mathcal{N}_j(t_i, \theta_j)$ predstavlja j-ti izlaz (nepoznanice) za i-ti uzorak dobiven koristeći skup parametara θ_j. Uz prethodne oznake za svaki $t \in [a, b]$ rješenje je dano s

$$\hat{y}_j(t, \theta_j) = a_j + (t - a)\mathcal{N}_j(t, \theta_j), \qquad j = 1, \ldots, n \tag{3.1}$$

   Treba uočiti da su ovakvom definicijom inicijalni uvjeti zadovoljeni.

5. *Optimizacija funkcije gubitka*

   Posljednji korak zahtijeva minimizaciju funkcije gubitka, odnosno njenu konvergenciju prema nuli. Vodeća komponenta gubitka se dobije računajući

$$\mathcal{L}_{\mathcal{F}} = \sum_{i=1}^{m} \sum_{j=1}^{n} \left[\frac{d\hat{y}_j}{dt} - f_j\left(t_i, \hat{y}_j\left(t_i, \theta_j\right)\right) \right]^2$$

   Za racučanje potrebnih derivacija implementira se automatska diferencijacija iz biblioteke AutoGard. Obzirom na ugradene inicijalne uvjete, dio gubitka povezan s njima iznosi 0.

   Valja primjetiti, da uz manje modifikacije slična funkcija gubitka može se koristiti i u slučaju parcijalnih diferencijalnih jednadžbi što je predviđeno u prethodnome poglavlju. No, za potrebe ovog rada nije neophodno ulaziti u predloženo proširenje.

   Minimiziranje funkcije gubitka odvija se nadogradnjom parametara služeći se koracima algoritma adaptivne metode momenta ADAM objašnjenim u [20].

Implementacija

Algoritam je realiziran u Pythonu uz korištenje paketa *autogard.numpy*. Standardne biblioteke poput *DeepXDE, NeuroDiffEq, Modulus, SciANN, PyDENs, NeuralPDE.jl, TensorDiffEq, IDRLnet*; odnosno *TensorFlow, PyTorch* i ostale nisu korištene. Razlog tome leži u činjenici da naglasak na implementaciji nije bio isključivo na efikasnosti i jednostavnosti, koju bi uređeni paketi poput ovih zasigurno osigurali, već na samoj strukturi (mehanizmima i edukaciji) te razumijevanju svih pozadinih odrednica.

Primjerice, na ovaj način inicijalizacija parametara i njihova pohrana te operacije nad njima bile su pod punom kontrolom. To je omogućilo da direktnim korištenjem *elemetwise_gard*-ove automatske diferencijacije, konkretno u proces izračuna vrijednosti gubitka i prilikom ažuriranja parametara, taj dio bude direktno implementiran i pod nadzorom. Cijeli proces je detaljnije opisan prethodno u 4. i 5. koraku algoritma. Na taj način osigurana je dodatna razina razumijevanja pozadinskih procesa unutar neuronske mreže i same arhitekture. Sastavni dio upravo opisane implementacije, prikazan je idućom ilustracijom realiziranog koda.

```
106 ##############################################################################
107
108 def Y_hat_i(X, param, i, bc, t0 = 0, model = None):
109     L = len(param)//2
110     y = X
111
112     for j in range(1, L + 1):#+1
113         W = param['W' + str(j)]
114         b = param['b' + str(j)]
115         if j == L:
116             y = np.dot(W, y) + b
117             break;
118         y = sigmoid(np.dot(W, y) + b, model)
119     #y = np.arctanh(y)
120     return bc[i-1] + (X-t0)*y[i-1]
121
122 def Y_hat(X, params, bc, t0 = 0, model = None):
123     L = len(params) // 3
124     y = []
125
126     for i in range(1, L+1):
127         y_hat_i_result = Y_hat_i(X, params['P' + str(i)], i, bc, t0, model)
128         y.append(y_hat_i_result)
129
130     y = np.array(y)
131
132     return y
133
134 ##############################################################################
```

Slika 3.1: Pomoćna funkcija za dobivanje željene vrijednosti u jednom prolazu kroz mrežu

```
134 ################################################################################
135
136 def forward_propagation(X, params, f, bc, t0 = 0, model = None):
137     d = len(params) // 3
138     grads = {}
139
140     def loss(*args):
141         L = len(args)
142         Y_k = []
143
144         for i in range(1, L+1):
145             y_hat_i_result = Y_hat_i(X, args[i-1], i, bc, t0, model)
146             Y_k.append(y_hat_i_result)
147
148         Y_k = np.array(Y_k)
149
150         rs = f(X, Y_k)
151         cost = np.zeros_like(Y_k[0])
152
153         for i in range(1, L+1):
154             dy = egrad(Y_hat_i, argnum = 0)(X, args[i-1], i, bc, t0, model)
155             cost += (dy - rs[i-1])**2
156
157         return np.sum(cost) / L
158
159     inner_dicts = [params[f'P{i}'] for i in range(1, d + 1)]
160     cost = loss(*inner_dicts)
161
162     for i in range(1, d + 1):
163         grads['P' + str(i)] = egrad(loss, argnum = i-1)(*inner_dicts)
164
165     return cost, grads
166
167 ################################################################################
```

Slika 3.2: Okosnica koda za izračun vrijednosti trenutnog reziduala i potrebnih gradijenata za korak optimizacije

Oznaka *egrad* predstavlja spomenutu funkciju iz biblioteke *autogard.numpy* korištenu za automatsku diferencijaciju modela. Uvođenjem novih restrikcija ili sastavnica funkcije gubitka, moguće je dodatno regulirati proces učenja. Jedan od prijedloga bio bi provođenje studije oko uvođenja težina za kombinaciju specifične točke u procesu učenja i odgovarajućeg parametara θ_j koji se uzima u obzir pri konačnom vrednovanju i formiranju matrice $\mathcal{N}$. Dio reziduala povezan s tom kombinacijom ima najveći utjecaj na grešku procjene.

Metoda optimizacije ADAM, također je samostalno konstruirana koristeći opisani algoritam što ne predstavlja osobit problem obzirom na već naglašenu jednostavnost implementacije koraka. Nešto na što valja obratiti pozornost je mogućnost fiksiranja koraka iteracije koje neke empirjske studije pokazuju da pospješuje konvergenciju. No, takve odluke ne treba donositi proizvoljno, nego u skladu s rezultatima.

3.2 Rezultati

U ovome dijelu prezentiraju se podaci dobiveni implementacijom prethodnog algoritma. Cilj ove simulacije je ispitati svojstva prezentirana u prethodnome poglavlju i usporediti efikasnost metode s dokazanom metodom Runge-Kutta 4.

Ponajprije, razmatra se ponašanje konvergencije u odnosu na broj skrivenih slojeva, tj. neurona po pojedinom sloju.

Nadalje, slijedi usporedba analitičkog rješenje s rezultatima dobivenih neuronskom mrežom i zatim numeričkom metodom Runge-Kutta 4. Neće se posebno naglašavati, no podrazumijeva se da za vrijeme provedbe specifičnih testova, svi parametri osim promatranih su nepromjenjeni.

Naposljetku, odabiremo dodatan sustav te promatramo i komentiramo rezultate u sklopu završnih opažanja.

Za svrhe simulacije koristimo problem prezentiran u [17]:

$$\frac{dy_1}{dt} = \cos(t) + y_1^2 + y_2 - \left(1 + t^2 + \sin^2(t)\right)$$
$$\frac{dy_2}{dt} = 2t - \left(1 + t^2\right)\sin(t) + y_1 y_2$$

gdje $t \in [0, 1]$ uz inicijalne uvjete $y_1(0) = 0$ i $y_2(0) = 1$.

Pripadajuća analitička rješenja su:

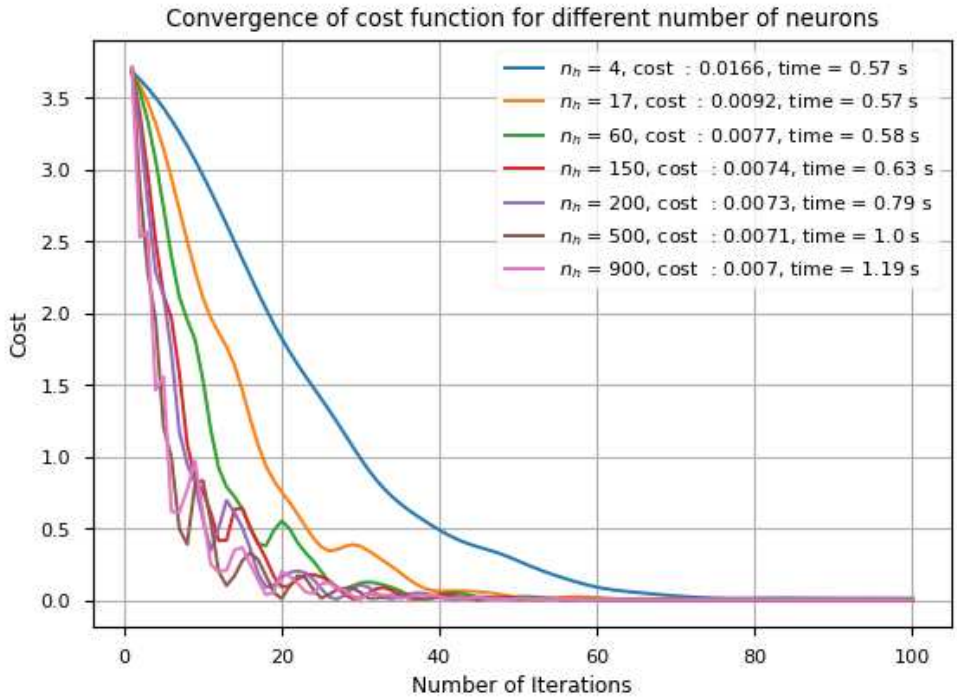
$$y_1(t) = \sin(t)$$

$$y_2(t) = 1 + t^2$$

.

Simulacija neuronske mreže

Počinjemo s promatranjem procesa konvergencije u slučaju jednog skrivenog sloja s različitim brojem neurona; $n_h = 4, 17, 60, 150, 200, 500$ i 900. Valja naglasiti da manje promjene, $\pm$ nekoliko neurona ne čini razliku - empirijski provjeravano i potvrđeno - te za svaku od vrijednosti prikazujemo vrijeme zaustavljanja kao i posljednju vrijednost funkcije gubitka.

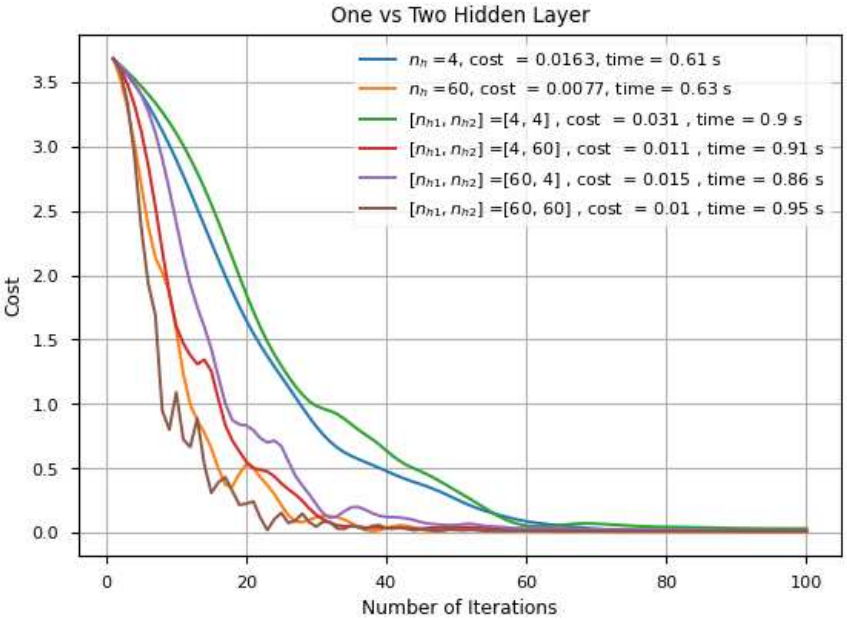


Slika 3.3: Konvergencija funkcije gubitka za različit broj neurona

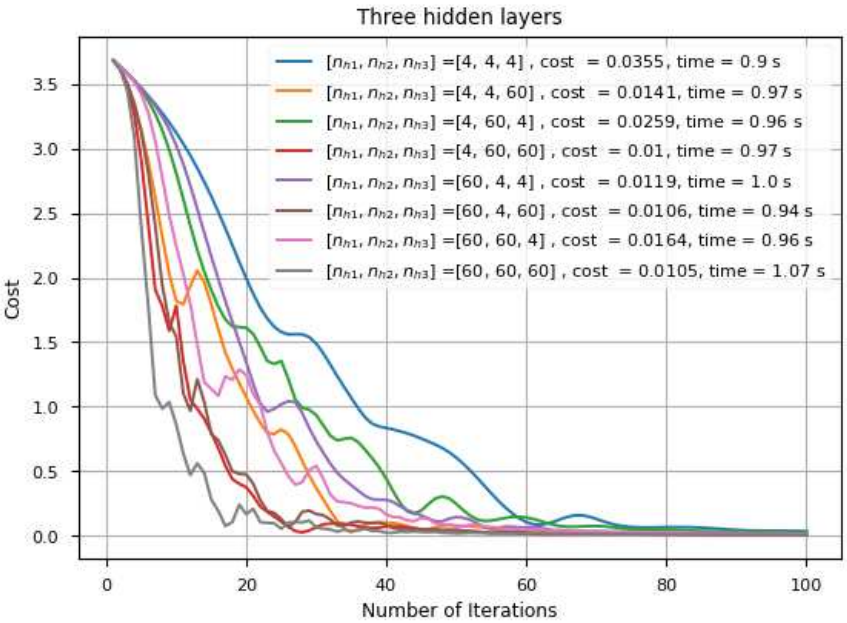
Prvo što se uočava je da se poprilično velika razina preciznosti $\epsilon \approx 0.017$ postiže s jednim skrivenim slojevem. Mogućnost takvog raspleta garantira i Cybenkov univerzalni teorem, no ovdje vidimo da ne ostaje samo u sferi teorije.

Dalje, primjetimo da preciznost raste povećanjem broja neurona, no osim što to dolazi uz cijenu povećanja kompleksnosti mreže (povećanje memorijskih i vremenskih zahtjeva), može se primjetiti da se iznos funkcije gubitka ustabilizirao već za 60 neurona. Time ostaje upitan smisao povećanja njihovog broja te u nastavku koristimo tu vrijednost kao oglednu.

Prirodno slijedi pitanje, ima li smisla povećavati broj skrivenih slojeva, ako već nema smisla povećavati broj neurona. Pod ruku ide i pitanje broja neurona za dodane slojeve.



Slika 3.4: Konvergencija funkcije gubitka za različit broj skrivenih slojeva



Slika 3.5: Konvergencija funkcije gubitka za slučaj više (tri) skrivenih slojeva

Sada postaje još evidentnije da nasumično i proizvoljno dodavanje parametara - u vidu broja neurona, odnosno skrivenih slojeva - nema smisla. Takav proces treba bit suvisao i potkrijepljen empirijskim ili drugim dokazima.

Vidljivo je da je već mreža s 60 neurona dovoljna za aproksimaciju ovog sustava. Štoviše, dodavanje dodatnih slojeva gotovo da i ne pospješuje konvergenciju, a u nekim situacijama čineći mrežu prekompleksnom, pogoršava rezultate. Dakle, dolazi do usporavanja i gubitka preciznosti, pogotovo u uspredbi s 1 slojem sastavljenim od 60 neurona.

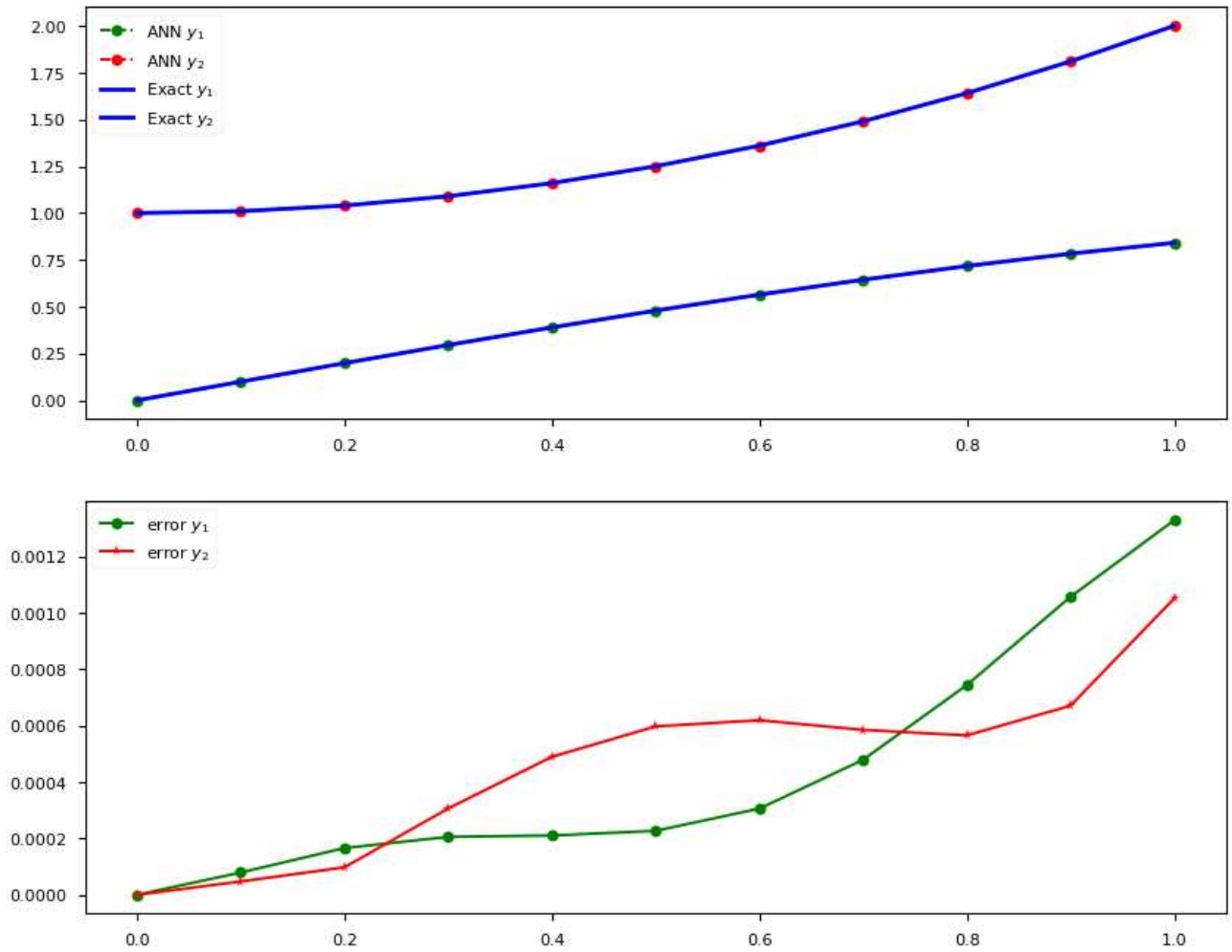
Numerička rješenja

Trenirajući neuronsku mrežu na ekvidistantnoj mreži od 11 točaka domene dobivamo:

t	y_1 ANN	y_1 Analytic	y_2 ANN	y_2 Analytic	error y_1	error y_2
0.0	0.000000	0.000000	1.000000	1.00	0.000000	0.000000
0.1	0.099754	0.099833	1.010047	1.01	0.000079	0.000047
0.2	0.198504	0.198669	1.039903	1.04	0.000166	0.000097
0.3	0.295315	0.295520	1.089694	1.09	0.000205	0.000306
0.4	0.389208	0.389418	1.159511	1.16	0.000210	0.000489
0.5	0.479199	0.479426	1.249403	1,27	0.000227	0.000597
0.6	0.564337	0.564642	1.359381	1.36	0.000306	0.000619
0.7	0.643739	0.644218	1.489415	1.49	0.000479	0.000585
0.8	0.716612	0.717356	1.639435	1.64	0.000744	0.000565
0.9	0.782270	0.783327	1.809330	1.81	0.001057	0.000670
1.0	0.840143	0.841471	1.998949	2.00	0.001328	0.001051

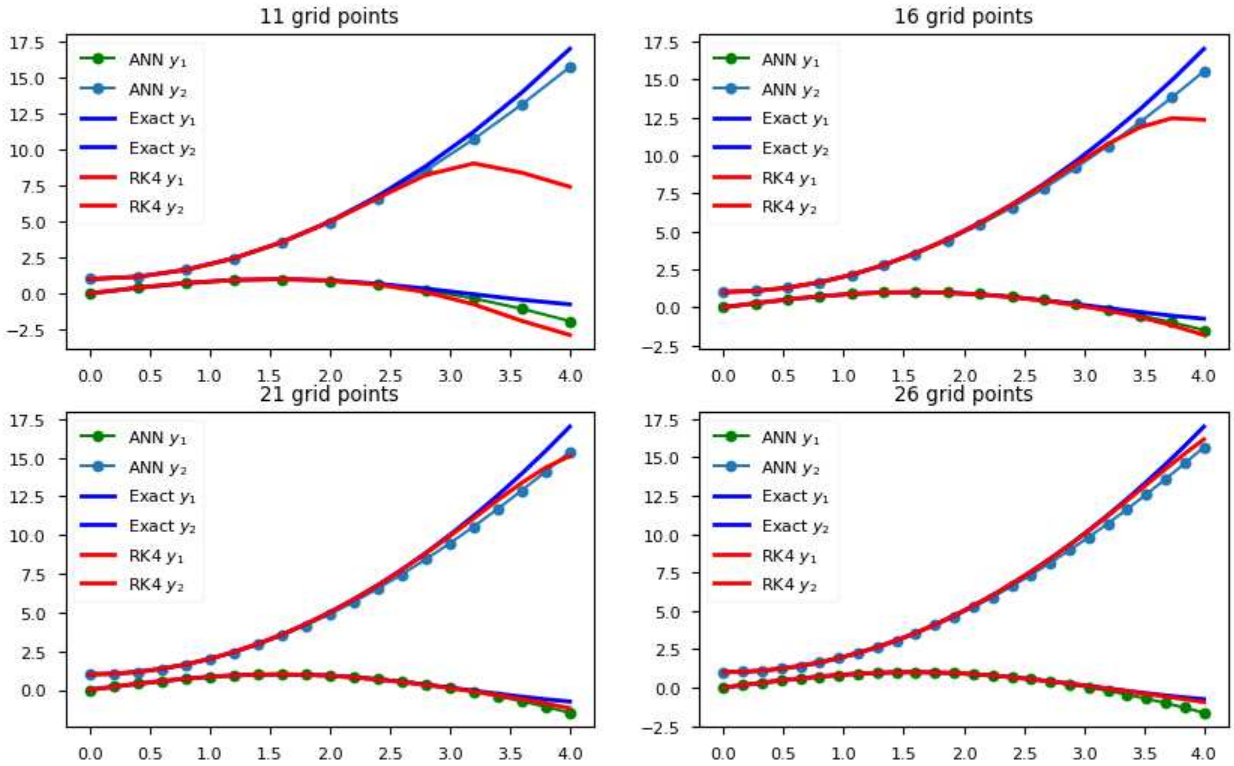
Tablica 3.1: Vrijednosti neuronske mreže i analitičkog rješenja

Lokalna greška je veličine reda treće decimale. Za većinu svakodnevnih primjena to je i više nego dovoljno. Dodatno, uzevši u obzir da niti su uvođene dodatne restrikcije na model, niti je odabir arhitekture (kao niti funkcije gubitke i drugih hiperparametara) u ovom slučaju posebno prilagođen promatranom problemu, prostora za napredak je mnogo.



Slika 3.6: Grafički prikaz aproksimacije neuronskom mrežom i egzaktnog rješenja (gore) i graf reziduala (ispod)

Konačno, uspoređujemo Runge-Kutta 4 metodu za različite mreže - 11, 16, 21, 26 točaka na domeni $[0, 4]$ - s aproksimacijom dobivenom s neuronskom mrežom.



Slika 3.7: Usporedba preciznosti neuronske mreže i četvrte Runga-Kutta za različite gustoće mreže

Za manje točaka neuronska mreža daje bolje predikcije, dok se povećanjem gustoće mreže to mijenja u korist numeričke metode. Takav rezultat ne iznenađuje previše jer numerička metoda poput ove postiže proizvoljnu preciznost obzirom na gustoću mreže. To ne znači da treba odustati od dodatnog istraživanja neuronskih mreža i njihove sposobnosti konvergencije. Razumijevanje limita oboje metode, omogućava ispravan odabir i implementaciju u područjima od interesa, ali i pomaže u unaprjeđenju istih.

Grid points	ANN error	RK4 error
11	1.263	9.588
16	1.482	4.668
21	1.620	1.902
26	1.343	0.825

Tablica 3.2: Iznos greške za krajnju točku $t = 4$

Posljednja tablica greške krajnje vrijednosti točke svoju primjenu pronalazi kod pokazivanja trendova konvergencije, odnosno divergencije i zaostajanja u odnosnu na rješenje. Na taj način, postaje jasno da povećanje gustoće nema nužno izražen utjecaj na neuronske mreže kakav ima kod RK4 te da je za poboljšanje metode možda potrebno gledati u drugom smjeru.

3.3 Zaključak i završni komentari

Da sumiramo, rješavanje običnih diferencijalnih jednadžbi s fizikom informiranom neuronskom mrežom pokazuje obećavajuće rezultate, vrijedne usporedbe s etabliranim numeričkim metodama poput Runga-Kutte četvrtog stupnja.

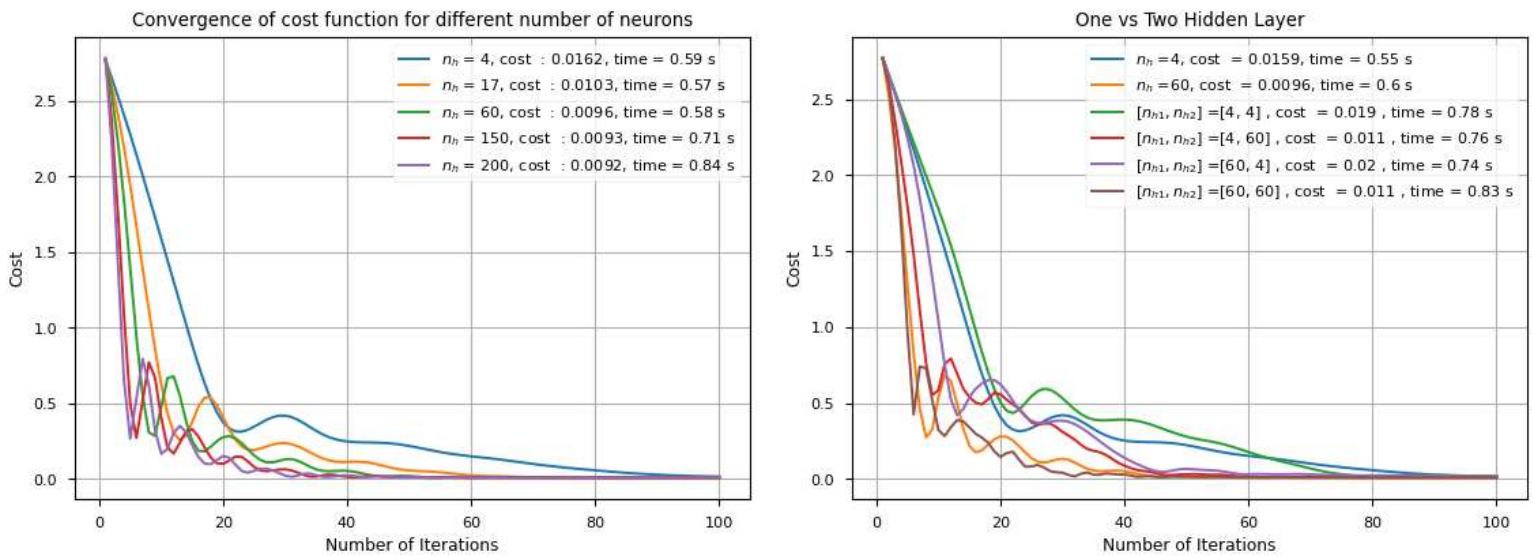
Koristeći vektorizirani algoritam implementiran u Pythonu, čak uz najjednostavnije modele i bez posebnog targetiranja problema, odnosno dodavanja svojstvenih eksluzivnih značajki, moguće je postići visoke preciznosti aproksimacije uz minimalne utroške memorije i vremena.

Ponavljajući studiju za vlastito odabrani sustav:

$$\frac{dy_1}{dt} = \cos^2(t) + y_1^2 - y_2 - 2t\sin(t)$$
$$\frac{dy_2}{dt} = 2y_1 + \sin(t)$$

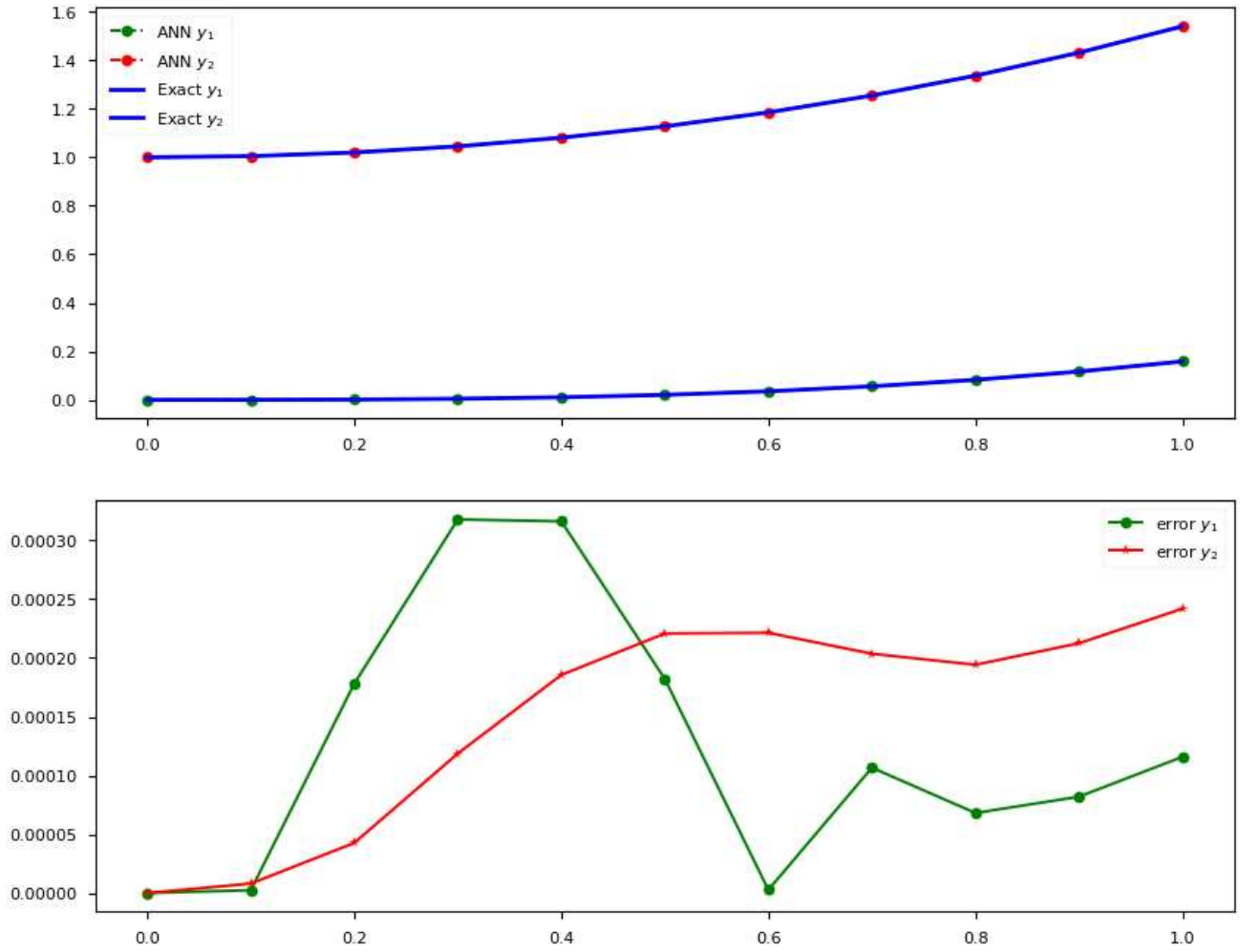
uz istu domenu $[0, 1]$ i inicijalne uvjete: $y_1(0) = 0$ i $y_2(0) = 1$ te egzaktna rješenja:

$$y_1(t) = t - sin(t) \qquad y_2(t) = t^2 + cos(t)$$



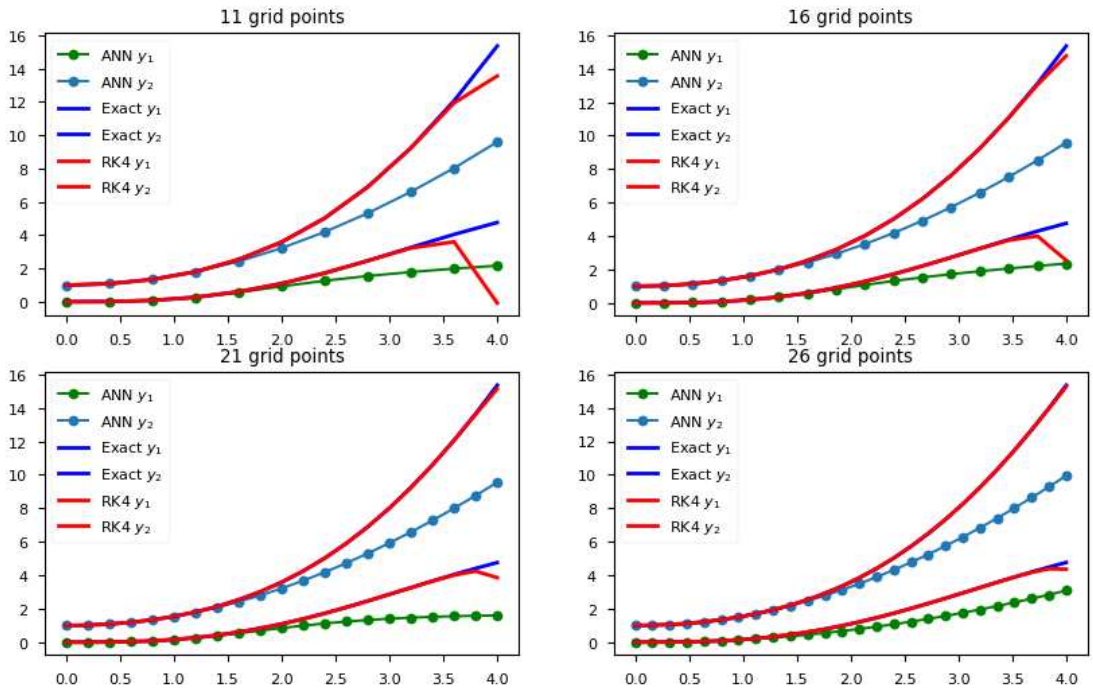
Slika 3.8: Ponovljena studija broja neurona i skrivenih slojeva

Postižu se isti, odnosno dodatno potvrđuju empirijski rezultati: povećanje i proizvoljno podešavanje parametara nije optimalno. Čak za mali broj parametara neuronska mreža s jednim skrivenim slojem ima mogućnost precizne aproksimacije rješenja što je opet vidljivo:



Slika 3.9: Grafički prikaz funkcija i reziduala

Graf reziduala ponovno pokazuje na lokalnu grešku tek na četvrtoj decimali. Da se stvar nešto komplicira proširenjem domene vidljivo je na sljedećem prikazu.



Slika 3.10: Usporedba PINNs i RK4

Ovdje vidimo da povećanjem intervala i udaljavanjem od inicijalnih uvjeta, dolazi do akumuliranja greške i neuspjeha pri zarobljavanju toka funkcije unutar prihvaljive greške na čitavoj domeni. Povećanjem gustoće mreže, greška RK4 očekivano se topi, dok utjecaj na PINNs je gotovo nepostojeći.

Odgovore na probleme robusnosti metode i stabilnosti konvergencije treba potražiti u redefiniranju i korištenju drugačijih baznih struktura kod implementacije neuronske mreže (promjena arhitekture, odabir aktivacijske funkcije, itd.). Prilagodba problema promatranom sustavu i raspoloživim podacima vodeći je faktor koji treba uzeti prilikom rješavanja sustava i odabira značajki.

Ponovimo radovi poput [16] ukazuju da uzrok problema ne leži u sposobnostima aproksimacije struktura neuronskih mreža, već da su posljedica nedovoljnih ili prelabavih ograničenja inkorporiranih unutar promatrane funkcije gubitka. Kao posljedica dolazi do isčezavanja povezanosti među optimizacijom gubitka i rješavanja sustava kojim se bavimo. Upozoravaju da čak i za jednostavne funkcije poput gore promatranog, može doći do velikih

odstupanja. Od istih autora predlaže se usitnjavanje problema - rastavljanje na više uzastopnih mreža gdje svaka radi isključivo na svome intervalu - kao jedan od odgovora čemu u prilog ide graf reziduala.

Iako se s istim problemima loše aproksimacije susreću i numeričke metode, ograničenje iznosa greške je uglavnom poznato i podložno kontroli. Upravo je dobra istraženost ta koja razlikuje numeričke metode od strojnog učenja. Posebice istraživanja koja testiraju uvjete konvergencije i optimalan odabir arhitekture su ona kojima bi se trebalo dodatno baviti.

Isto tako, na neuronske mreže ne treba gledati kao zamjenu za numeričke metode, već nadopunu. Također, povezivanje ovih dvaju pristupa pokazuje itekako dobre rezultate te razloga za optimizam je jednako mnogo kao i prostora za napredak.

Bibliografija

[1] *Complete Guide to Adam Optimization*, https://medium.com/@LayanSA/complete-guide-to-adam-optimization-1e5f29532c3d.

[2] *IntechOpen Chapter*, https://www.intechopen.com/chapters/84738.

[3] *Neural Networks Activation Functions*, https://www.v7labs.com/blog/neural-networks-activation-functions#3-types-of-neural-networks-activation-functions.

[4] *Physics-Based Deep Learning: DiffPhys Examples*, https://physicsbaseddeeplearning.org/diffphys-examples.html.

[5] *Physics-Informed Machine Learning (PIML) Tutorial*, https://2prime.github.io/files/SML/piml_aaai.pdf.

[6] *What is a Physics-Informed Neural Network?*, https://benmoseley.blog/my-research/so-what-is-a-physics-informed-neural-network/.

[7] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul i Jeffrey Mark Siskind, *Automatic Differentiation in Machine Learning: A Survey*, (2018).

[8] Snehashish Chakraverty i Susmita Mall, *Artificial Neural Networks for Engineers and Scientists: Solving Ordinary Differential Equations*.

[9] Pao Hsiung Chiu, Jian Cheng Wong, Chinchun Ooi, My Ha Dao i Yew Soon Ong, *CAN-PINN: A Fast Physics-Informed Neural Network Based on Coupled-Automatic–Numerical Differentiation Method*, Computer Methods in Applied Mechanics and Engineering **398** (2022), 11490.

[10] Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi i Francesco Piccialli, *Scientific Machine Learning through Physics-Informed Neural Networks: Where we are and What's next*, https://arxiv.org/pdf/2201.05624.

[11] Tamirat Temesgen Dufera, *Deep Neural Network for System of Ordinary Differential Equations: Vectorized Algorithm and Simulation*.

[12] Ian Goodfellow, Yoshua Bengio i Aaron Courville, *Deep Learning*, MIT Press, 2016.

[13] A. J. Meade Jr. i A. A. Fernandez, *The Numerical Solution of Linear Ordinary Differential Equations by Feedforward Neural Networks*, 1994.

[14] Diederik P. Kingma i Jimmy Lei Ba, *ADAM: A Method for Stochastic Optimization*, International Conference on Learning Representations (ICLR) (2015).

[15] Stefan Kollmannsberger, Davide D'Angella, Moritz Jokeit i Leon Herrmann, *Deep Learning in Computational Mechanics*.

[16] Aditi S. Krishnapriyan, Amir Gholami, Shandian Zhe, Robert M. Kirby i Michael W. Mahoney, *Characterizing possible failure modes in physics-informed neural networks*, `https://arxiv.org/pdf/2109.01050`.

[17] Isaac Elias Lagaris, Aristidis Likas i Dimitrios I. Fotiadis, *Artificial Neural Networks for Solving Ordinary and Partial Differential Equations*, IEEE Transactions on Neural Networks **9** (1998), br. 5, 987–1000.

[18] Martin Lotz, *Mathematics of Machine Learning*.

[19] LU LU, XUHUI MENG, ZHIPING MAO i GEORGE EM KARNIADAKIS, *DEEPXDE: A DEEP LEARNING LIBRARY FOR SOLVING DIFFERENTIAL EQUATIONS*, `https://arxiv.org/pdf/1907.04502`.

[20] Charles C. Margossian, *A Review of Automatic Differentiation and its Efficient Implementation*, `https://arxiv.org/pdf/1811.05031`.

[21] Siddhartha Mishra i Roberto Molinaro, *Estimates on the Generalization Error of Physics Informed Neural Networks (PINNs) for Approximating PDEs*, 2023, `https://arxiv.org/pdf/2006.16144`.

[22] Gabriel Nagy, *Ordinary Differential Equations*, 2021, `https://users.math.msu.edu/users/gnagy/teaching/ode.pdf`.

[23] Renato G. Nascimento, Kajetan Fricke i Felipe A.C. Viana, *A tutorial on solving ordinary differential equations using Python and hybrid physics-informed neural network*.

[24] Maziar Raissi, Paris Perdikaris i George Em Karniadakis, *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations*, `https://arxiv.org/pdf/1711.10561`.

[25] ———, *Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations*, https://arxiv.org/pdf/1711.10566.

[26] M. Raissi, P. Perdikaris i G.E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, (2019), https://faculty.sites.iastate.edu/hliu/files/inline-files/PINN_RPK_2019_1.pdf.

[27] Keith Rudd i Silvia Ferrari, *A Constrained Integration (CINT) Approach to Solving Partial Differential Equations using Artificial Neural Networks*, 2013.

[28] Koki Saitoh, *Deep Learning from the Basics - Python and Deep Learning: Theory and Implementation*.

[29] Yeonjong Shin, Jerome Darbon i George Em Karniadakis, *On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs*, https://arxiv.org/pdf/2004.01806.

[30] Edward Small, *An Analysis of Physics-Informed Neural Networks*, https://arxiv.org/pdf/2303.02890.

[31] Kaustubh Sridhar, Souradeep Dutta, James Weimer i Insup Lee, *Guaranteed Conformance of Neurosymbolic Models to Natural Constraints*, https://arxiv.org/pdf/2212.01346.

[32] Zvonimir Tutek i Marko Vrdoljak, *Obične diferencijalne jednadžbe*, Matematički odsjek, PMF, 2019.

[33] Sifan Wang, Shyam Sankaran i Paris Perdikaris, *Respecting Causality is All You Need for Training Physics-Informed Neural Networks*, http://arxiv.org/abs/2203.0740.

[34] Sifan Wang, Yujun Teng i Paris Perdikaris, *Understanding and Mitigating Gradient Pathologies in Physics-Informed Neural Networks*, 2020, https://arxiv.org/pdf/2001.04536.

[35] Neha Yadav, Anupam Yadav i Manoj Kumar, *An Introduction to Neural Network Methods for Differential Equations*.

Sažetak

U ovom radu predstavljena je i implementirana nova vrsta neuronskih mreža koja je u posljednje vrijeme pobudila interes mnogih znanstvenika. Riječ je o fizikom informiranim neuronskim mrežama koje poprilično uspješno koristeći moderne tehnike, transformiraju dobro poznati problem rješavanja sustava diferencijalnih jednadžbi.

U uvodnom poglavlju predstavlja se potrebna teorijska podloga za razumijevanje problema koji se adresira, a to su diferencijalne jednadžbe i njihovi sustavi. Zatim, nakon upoznavanja klasičnih numeričkih metoda i njihovih ograničenja, dolazimo do osnova neuronskih mreža.

Konačno, na red stižu i najavljene fizikom informirane mreže: njihova funkcionalnost i upoznavanje s ulogom, odnosno položajem u trenutnom znanstvenom i industrijskom okruženju. Za kraj bavimo se implementacijom metode i analizom rezultata za odabrane sustave.

Glavni cilj rada bila je realizacija vektoriziranog algoritma kroz čiju konstrukciju je potrebno spoznati ograničena i prednosti, te osvijetliti smjer i potencijal razvoja u budućnosti. Glavni rezultati ukazuju na optimizam oko komplementarnosti ovog tipa neuronske mreže i numeričkih metoda koje su sazrijele i preživjele zub vremena.

Summary

In this paper, a new type of neural network that has recently attracted the interest of many scientists is presented and implemented. These are Physics-Informed Neural Networks (PINNs), which quite successfully transform and deal with the well-known problem of solving systems of differential equations using modern techniques.

The introductory chapter presents the necessary theoretical background for understanding the addressed problem, which includes differential equations and their systems.

After introducing classical numerical methods and their limitations, the basics of neural networks are covered.

Finally, the announced Physics-Informed Neural Networks are discussed: their functionality and their role or position in the current scientific and industrial landscape. The paper concludes with the implementation of the method and an analysis of the results for selected systems.

The main goal of the work was the realization of a vectorized algorithm, through whose construction it was necessary to understand its limitations and advantages, and to shed light on the direction and potential for future development.

The main results indicate optimism about the complementarity of this type of neural network with numerical methods that have matured and withstood the test of time.

Životopis

Rođen sam u Zagrebu 4. rujna 1999. godine gdje sam pohađao Osnovnu školu Antuna Branka Šimića. Poslije nje, put me dalje vodi u XV. Gimnaziju nakon koje 2018. upisujem i ovom obranom završavam Prirodoslovno-matematički faukultet u Zagrebu, smjer Financijske i poslovne matematike. Osim navedenih institucija, sudjelovao sam i semestar na Erasmus razmjeni zbog kojeg sam bio gost u Španjolskoj, odnosno Zaragozi na njihovoj inačici matematičkog odsjeka.