Multimodalni jezični modeli kao asistent u nastavi informatike

Stipaničev, Toni

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:217:718724

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

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

Repository / Repozitorij:

Repository of the Faculty of Science - University of Zagreb

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

Toni Stipaničev

MULTIMODALNI JEZIČNI MODELI KAO ASISTENT U NASTAVI INFORMATIKE

Diplomski rad

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

INTEGRIRANI PREDDIPLOMSKI I DIPLOMSKI SVEUČILIŠNI STUDIJ
FIZIKA I INFORMATIKA; SMJER NASTAVNIČKI

Toni Stipaničev

Diplomski rad

Multimodalni jezični modeli kao asistent u nastavi informatike

Voditelj diplomskog rada: prof. dr. sc. Davor Horvatić

Ocjena diplomskog rada: _____

Povjerenstvo: 1. _____

2. _____

3. _____

Datum polaganja: _____
Zagreb, 2024.

Zahvaljujem se svome mentoru, prof. dr. sc. Davoru Horvatiću na ukazanom povjerenju i velikoj pomoći pruženoj pri izradi diplomskog rada. Također htio bih se zahvaliti svojoj obitelji i prijateljima na velikoj pomoći i podršci tijekom studiranja.

Sažetak

Neuronske mreže omogućuju računalima da uče iz podataka, prepoznaju obrasce, predviđaju ishode i čak kreiraju sadržaje. One su ključni alat u razvoju dubokog učenja, grane umjetne inteligencije koja nastoji simulirati složenost ljudskog razmišljanja, otvarajući time mogućnosti u mnogim disciplinama. U ovom radu istražujemo kako multimodalni jezični modeli, poput ChatGPT-a i Claudea, mogu simulirati rad nastavnika i pružiti podršku učenicima kroz interaktivno vođeno učenje. Pokazali smo da ovi modeli mogu pomoći u prevladavanju čestih poteškoća koje djeca susreću u procesu učenja. Koristeći ChatGPT i Claude kao primjere, utvrdili smo da oba modela mogu biti adekvatan alat u ponavljanju gradiva i vježbanju. Međutim, važno je naglasiti da ovi modeli, iako korisni, ne mogu i ne smiju zamijeniti ulogu nastavnika. Nastavnici osim prenošenja stručnog znanja, pružaju pedagošku i društvenu podršku koju nijedan AI model ne može replicirati. Stoga, ključno je da se ne zaboravi niti zanemari nezamjenjiva uloga nastavnika u obrazovnom procesu.

Ključne riječi: duboko učenje, neuronska mreža, povratna propagacija, vođeno učenje

Multimodal language models as assistent in computer science education

Abstract

Neural networks enable computers to learn from data, recognize patterns, predict outcomes, and even generate content. They are a crucial tool in the development of deep learning, a branch of artificial intelligence that aims to simulate the complexity of human thinking, thus opening up possibilities in numerous disciplines. In this paper, we explore how multimodal language models, such as ChatGPT and Claude, can simulate the work of teachers and provide support to students through interactive guided learning. We have demonstrated that these models can assist in overcoming common difficulties children encounter in the learning process. Using ChatGPT and Claude as examples, we have determined that both models can serve as adequate tools for revision and practice. However, it is crucial to emphasize that these models, although useful, cannot and should not replace the role of teachers. In addition to imparting expert knowledge, teachers provide pedagogical and social support that no AI model can replicate. Therefore, it is essential not to forget or neglect the irreplaceable role of teachers in the educational process.

Keywords: deep learning, neural network, backpropagation, supervised learning

Sadržaj

1 Uvod **1**

2 Duboko učenje **2**

2.1 Nadzirano učenje 2

2.2 Neuronske mreže 3

2.2.1 Umjetni neuron 3

2.2.2 Duboke neuronske mreže 5

2.3 Aktivacijske funkcije 6

2.4 Funkcija gubitka 10

2.5 Propagacija pogreške unatrag 11

2.6 Multimodalni jezični modeli 13

2.7 Transformeri 14

3 Multimodalni jezični modeli kao asistent u nastavi informatike **22**

3.1 Uvjeti grananja 22

3.2 Petlje 32

3.3 Liste 38

3.4 Funkcije 44

3.5 Klase i objekti 50

3.6 Analiza sposobnosti jezičnih modela u vođenju učenika kroz zadani nastavni zadatak 58

4 Zaključak **68**

Literatura **69**

1 Uvod

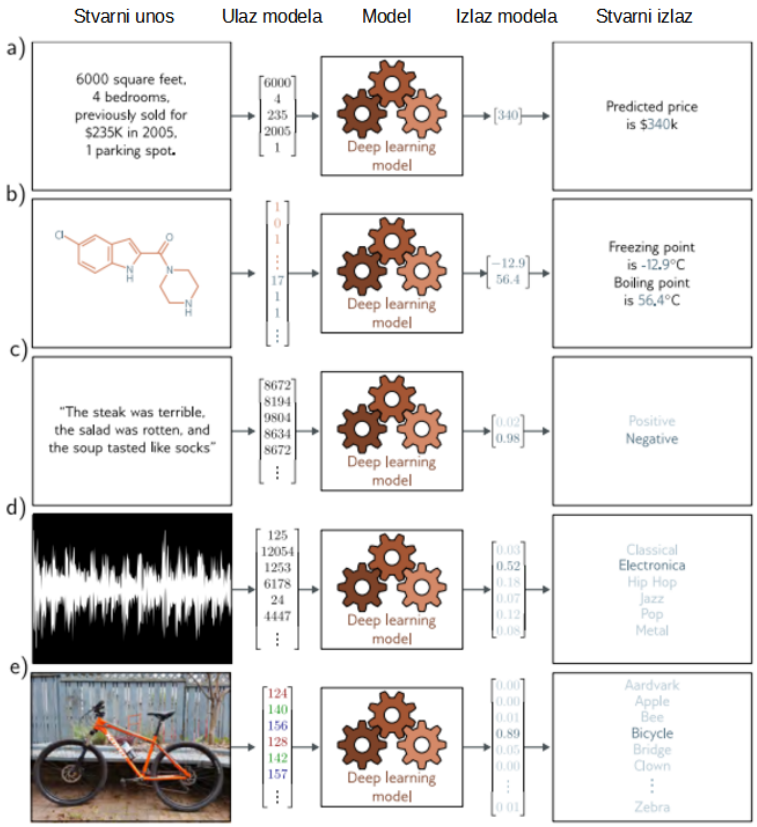
Umjetna inteligencija predstavlja jednu od najbrže rastućih grana tehnologije, s utjecajem na gotovo svaki aspekt života, mijenjajući način na koji izvršavamo zadatke, učimo ili radimo. Razvoj sve naprednijih multimodalnih jezičnih modela ima sve širu primjenu, uključujući i područje obrazovanja. S obzirom na sve veće potrebe za unapređivanje školstva proteklih nekoliko desetljeća klasično frontalno podučavanje sve se više zamjenjuje novim nastavnim metodama, gdje se fokus s profesora prebacuje na učenike. Pokazalo se da su interaktivne metode znatno poboljšale razumijevanje i savladavanje gradiva kod učenika. Jedan od najuspješnijih obrazovnih pristupa u tom kontekstu je učenje vođenjem, u kojem nastavnik ili mentor aktivno usmjerava učenika kroz proces učenja. Cilj ovakvog pristupa je razvijati razumijevanje i provjeravati točnost koncepata. S obzirom na sve veću izloženost digitalnim sadržajima i novim multimodalnim jezičnim modelima poput ChatGPT-ja i Claudea, postavlja se pitanje kako ove modele učiniti korisnima za razumijevanje gradiva. Kako bi postali alat za učenje, a ne samo kao sredstvo za dobivanje odgovora. U drugom poglavlju dajemo prikaz funkcioniranja dubokog učenja, opisujemo evoluciju od neuronskih mreža do multimodalnih jezičnih modela te objašnjavamo princip rada transformatora. U trećem poglavlju istražujemo primjenu multimodalnih jezičnih modela kao asistenata u nastavi informatike. Cilj je prikazati metodologiju pravilnog formuliranja upita ovim modelima, s naglaskom na unaprjeđenje kvalitete obrazovnog procesa i olakšavanje usvajanja gradiva kod učenika. Dodatno, analiziramo potencijalne prednosti i izazove integracije ovih tehnologija u obrazovno okruženje.

2 Duboko učenje

Strojno učenje je podskup umjetne inteligencije koji omogućuje donošenje odluke prilagođavanjem matematičkih modela promatranim podacima. Duboko učenje je specifičan model strojnog učenja, koji koristi višeslojne neuronske mreže za analizu velikih količina podataka i prepoznavanje složenih uzoraka. Ova tehnologija omogućuje AI sustavima obavljanje zahtjevnih zadataka s visokom preciznošću. [1]

2.1 Nadzirano učenje

Nadzirano učenje je kategorija strojnog učenja u kojoj model uči iz skupa parova ulazno-izlaznih istinitih informacija (označeni podatci) kako bi mogao pravilno označiti nove podatke. Ovaj pristup omogućuje rješavanje problema klasifikacije i regresije. U regresiji, model predviđa kontinuiranu izlaznu vrijednost za novi ulazni podatak na temelju naučenog obrasca iz skupa označenih podataka. U klasifikaciji, model određuje diskretnu izlaznu vrijednost novom ulaznog podatka, pridružujući ga jednoj od klasa iz skupa označenih podataka.



Slika 2.1: Primjeri regresija i klasifikacija. (Slika djelomično preuzeta iz [1])

Slika 2.1 prikazuje nekoliko primjera regresije i klasifikacije. U svim slučajevima, postoji smislen unos iz stvarnog svijeta (rečenica, zvučna datoteka, slika itd.), koji kodiran kao vektor brojeva, koristi kao ulaz modela. Model zatim preslikava ulazni vektor u izlazni vektor, koji se "prevodi" natrag u smisleno predviđanje u stvarnom svijetu. Model prikazan na slici 2.1a predviđa cijenu kuće na temelju ulaznih karakteristika poput kvadratura i broj spavaćih soba. Ovo je problem regresije jer model vraća kontinuirani broj, a ne dodjeljuje kategorije. Model prikazan na slici 2.1b koristi kemijsku strukturu molekule kao unos i predviđa točke smrzavanja i vrelišta. Ovo je problem multivarijatna regresija jer predviđa više od jednog kontinuiranog broja. Model prikazan na slici 2.1c prima tekstualni niz koji sadrži recenziju restorana kao ulaz i predviđa je li recenzija pozitivna ili negativna. Ovo je problem binarne klasifikacije jer model pokušava dodijeliti ulazni vektor jednoj od dvije kategorije. Slike 2.1d i 2.1e prikazuju primjere klasifikacije više klasa. U ovim slučajevima, izlazni vektor sadrži vjerojatnosti da ulazni vektor pripada svakoj od N kategorija. Model u prvom primjeru uzima audio datoteku kao ulaz i predviđa koji glazbeni žanr sadrži. U drugom primjeru, ulaz je slika, a model predviđa koji objekt prikazuje. U oba slučaja, model vraća vektor veličine N koji sadrži vjerojatnosti za svaku od N kategorija.

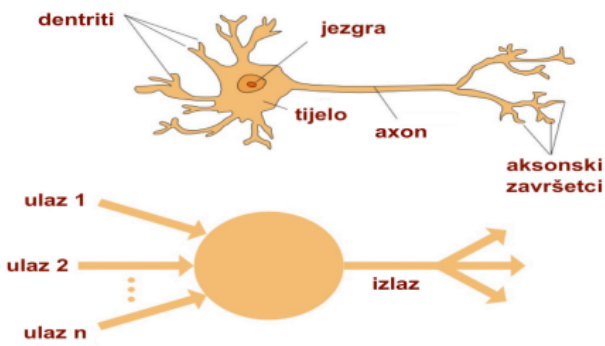
2.2 Neuronske mreže

Mnogi zadatci koji zahtijevaju inteligenciju ili prepoznavanje uzoraka, koje ljudi uspijevaju izvesti vrlo lako, iznimno je teško automatizirati. Primjerice, prepoznavanje različitih predmeta i izvlačenje smisla iz velike količine vizualnih informacija okoline. Modeli umjetne inteligencije, inspirirani konstrukcijom ljudskog mozga, sposobni rješavati takve probleme nazivaju se neuronske mreže. [1]

2.2.1 Umjetni neuron

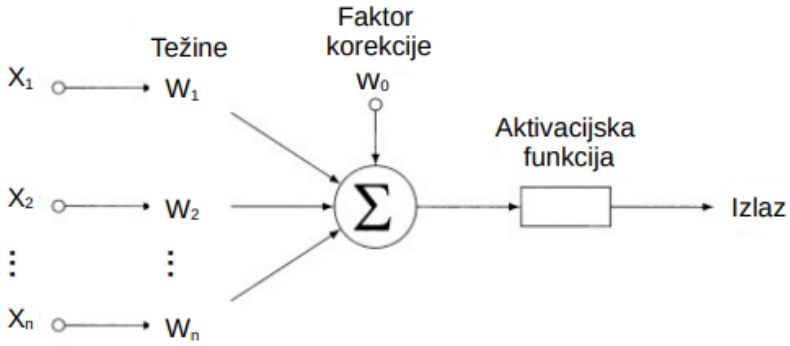
Istraživanjem bioloških organizama otkriveno je da se neuronski sustav sastoji od međusobno povezanih neurona (živčanih stanica). Biološki neuroni (Slika 2.2) sastoje se od tijela stanice, ulaznih niti (dendrita) i izlazne niti (aksona). Dendriti primaju električne impulse iz drugih stanica, koje tijelo neurona obrađuje, a potom se aksonima signal prenosi sljedećoj razini neurona. Tako živčani sustav bioloških organizama prenosi informacije koje prikupljaju osjetila. Umjetni neuroni su matematičke

funkcije koje oponašaju rad bioloških neurona. U ovom kontekstu, dendriti predstavljaju ulazne podatke, aksoni izlazne podatke, dok tijelo neurona provodi funkciju sumiranja i obrade ulaznih vrijednosti. [2]



Slika 2.2: Biološki i umjetni neuron. [2]

Perceptron (Slika 2.3) je računalna implementacija umjetnog neurona. Odnosno algoritam linearne regresije kojem je dodana aktivacijska funkcija koja preslikava ulazni vektor u skalarni izlaz. On je samostalna računalna jedinica koja ulazne vrijednosti množi težinama, zbraja ih i zatim djeluje aktivacijskom funkcijom na sumu. Aktivacijska funkcija uspoređuje vrijednost sume s graničnom vrijednosti koju nazivamo *prag*, te na temelju te usporedbe određuje izlazni signal.



Slika 2.3: Računalna interpretacija perceptrona.

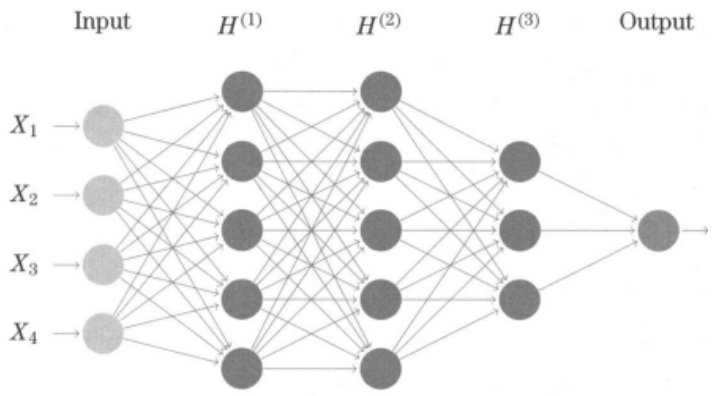
Ulazna vrijednost perceprona računa se kao:

$$h_w(x) = w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n \tag{2.1}$$

Težina w_0 određuje prag iznad kojega suma $(w_1x_1 + w_2x_2 + ... + w_nx_n)$ na izlazu daje pozitivnu vrijednost. Zbog toga se w_o naziva *faktor korekcije* (engl. *Bias*) te se često označava s b. Za projektiranje perceptrona potrebno je odrediti težine i faktor korekcije, koji se najbolje uklapaju u podatke.

2.2.2 Duboke neuronske mreže

Umjetna neuronska mreža spada u nadzirane i polunadzirane metode strojnog učenja. Mreža se sastoji od umjetnih neurona, najčešće organiziranih u slojeve, pri čemu se razlikuju: ulazni sloj, skriveni slojevi i izlazni sloj. U slučaju da umjetna neuronska mreža ima samo jedan skriveni sloj naziva se *plitka neuronska mreža*. Kako se broj neurona u skrivenom sloju povećava, plitka neuronska mreža postaje sve moćniji alat. Međutim, pokazalo se da za neke funkcije potreban broj neurona postaje nepraktično velik. Dok korištenje više skrivenih slojeva s istim brojem parametara, mreža postiže veći kapacitet, što omogućuje opisivanje šire obitelji funkcija. Postoje razne arhitekture spajanja neurona u neuronsku mrežu, pri čemu svi neuroni skrivenih slojeva moraju biti povezani s barem jednim neuronom uzastopnog sloja. Ako su svi neuroni svakog sloja povezani sa svim neuronima sljedećeg sloja kažemo da je neuronska mreža *potpuno povezana* (Slika 2.4). [1]



Slika 2.4: Duboka neuronska mreža. [4]

Neuronskoj mreži ulazni podatci dolaze u obliku vektora $\mathbf{x}^{(i)} = (x_1^{(i)}, ..., x_n^{(i)})^T$, gdje su $x_1^{(i)}, ..., x_n^{(i)}$ vrijednosti pojedinih značajki i-tog primjera. U slučaju nadziranog učenja, označene izlazne vrijednosti, označavaju se kao $\mathbf{y}^{(i)} = (y_1^{(i)}, ..., y_k^{(i)})^T$, gdje je k broj izlaznih neurona. Ako je neuronska mreža potpuno povezana, dodavanjem faktora korekcije težinama, matrica težine između slojeva $l-1$ i l je:

$$\mathbf{w}^{(l)} = \begin{bmatrix} w_{10}^{(i)} & x_{11} & \dots & x_{1N_{l-1}} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N_l 0} & x_{N_l 1} & \dots & x_{N_l N_{l-1}} \end{bmatrix}. \tag{2.2}$$

2.3 Aktivacijske funkcije

Nakon što se unutar neurona izračuna težinska suma vrijednosti neurona iz prethodnog sloja, na tu vrijednost primjenjuje se aktivacijska funkcija. Zbog bolje optimizacije, poželjno je da ta funkcija bude neprekidna i diferencijabilna. Neke od aktivacijskih funkcija koje se često koriste su:

Step funkcija

$$s(x) = \begin{cases} 1, & \text{ako je } \ \mathrm{x} \geq \theta \\ 0, & \text{ako je } \ \mathrm{x} < \theta \end{cases}, \tag{2.3}$$

često se koristi u izlaznom sloju binarne klasifikacije, gdje je $\theta \in \mathbb{R}$. Mana joj je što može odvojiti samo linearno odvojive klase.

ReLU funkcija

$$ReLU(x) = \max(0, x), \tag{2.4}$$

(engl. *Rectified Linear Unit*) je po dijelovima linearna funkcija koja se koristi zbog jednostavnosti optimizacije. U slučaju velikog broja negativnih vrijednosti neurona, zbog gubljena tih informacija, funkcija postaje problematična.

Propusna ReLU funkcija

$$LReLU(x) = \max(0, x) + 0,001x, \tag{2.5}$$

iako rješava problem negativnih vrijednosti neurona, i dalje ne rješava problem računanja gradijenta u $x = 0$.

Softplus funkcija

$$\zeta(x) = \log(1 + e^x), \tag{2.6}$$

predstavlja glatku verziju ReLU funkcije.

Sigmoidna funkcija

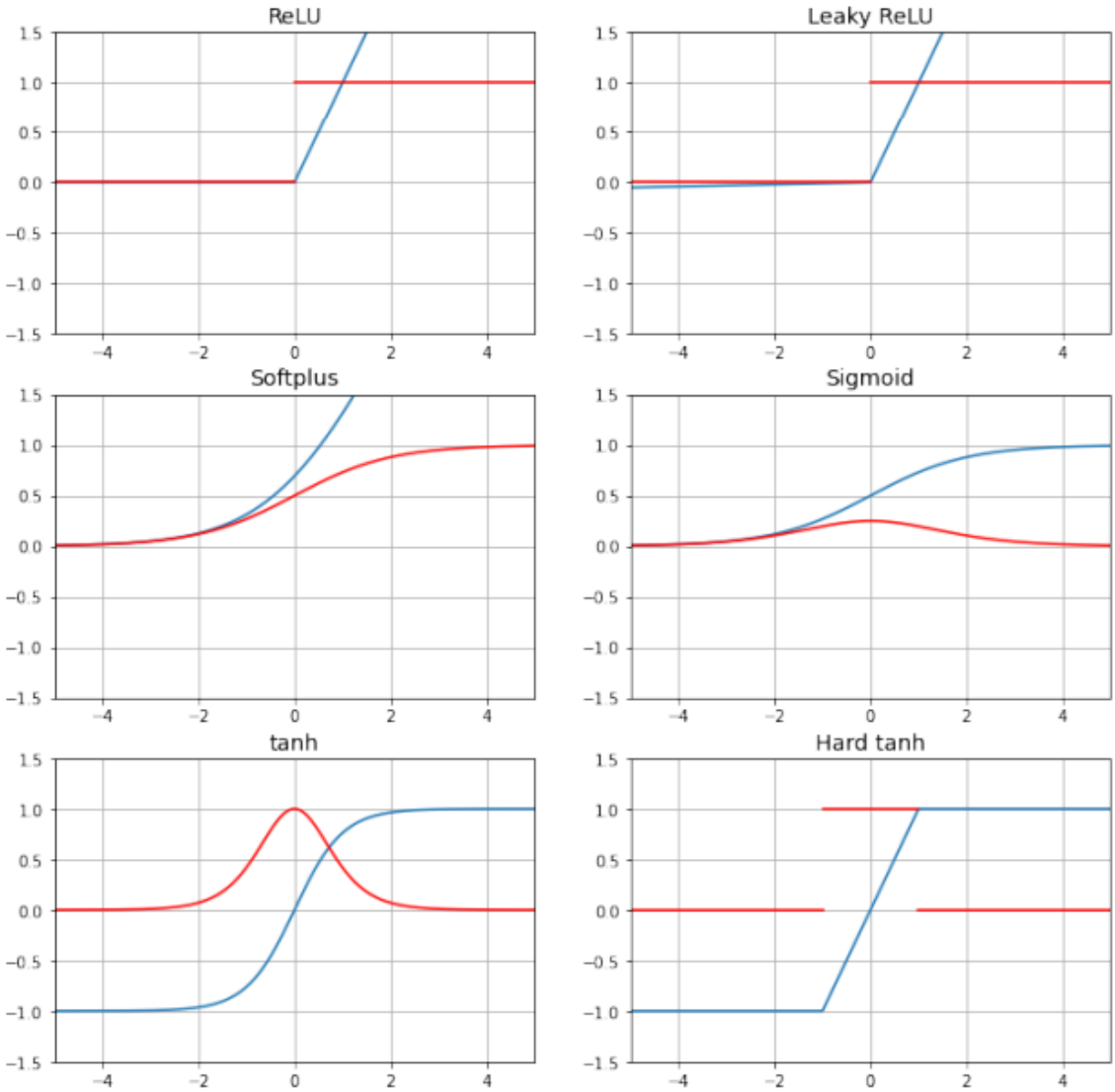
$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.7}$$

koristi se prilikom predviđanja vjerojatnosti klasifikacije više klasa jer vraća vrijednost u intervalu (0,1). Pogodna je za analizu jer je derivabilna u svim točkama.

Tangens hiperbolni funkcija

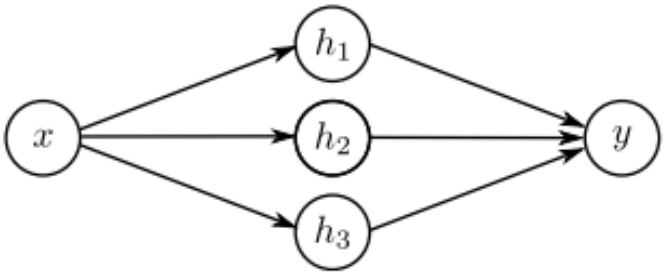
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{2.8}$$

je skalirana i translatirana verzija sigmoidne funkcije. Često prikazuje bolje rezultate od sigmoidne funkcije te se krajem prošlog stoljeća počela češće upotrebljavati.



Slika 2.5: Grafovi aktivacijskih funkcija (plavo) i pripadne derivacije (crveno). [6]

Za ilustraciju primjene ReLU aktivacijske funkcije, razmotrit ćemo plitku neuronsku mrežu koja se sastoji od jednog neurona u ulaznom sloju, tri neurona u jednom skrivenom sloju i jednog neurona u izlaznom sloju. Vizualizacija plitke neuronske mreže prikazana je na slici 2.6. Ovaj primjer zorno prikazuje kako se od linearnih ulaza mogu dobiti nelinearni izlazi, što je ključna karakteristika neuronskih mreža. U ovom modelu, ulazni podatak x prolazi kroz tri neurona skrivenog sloja. Za svaki



Slika 2.6: Primjer plitke neuronske mreže. [6]

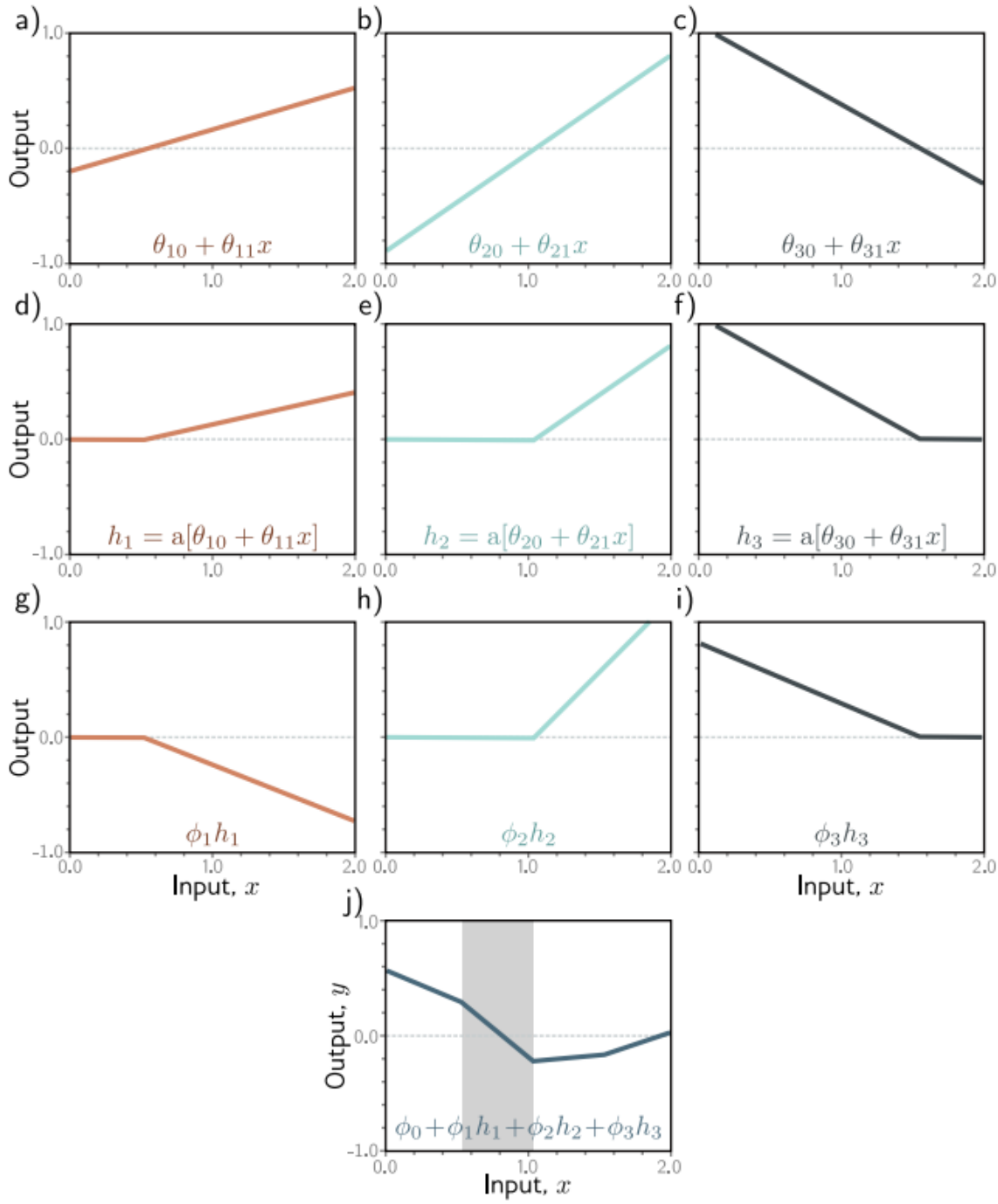
neuron, ulaz se transformira linearnom funkcijom oblika:

$$\begin{aligned} h_1 &= w_1 x + w_0^{(1)} \\ h_2 &= w_2 x + w_0^{(2)} \\ h_3 &= w_3 x + w_0^{(3)} \end{aligned} \tag{2.9}$$

gdje w_i predstavlja težinu i-tog neurona, a $w_0^{(i)}$ prag i-tog neurona, za $i \in 1, 2, 3$. Zbog različitih vrijednosti težina i pragova, ove linearne funkcije rezultiraju različitim pravcima, što je prikazano na Slici 2.7 (a, b, c). Nakon linearne transformacije, rezultati prolaze kroz ReLU aktivacijsku funkciju. ReLU funkcija definirana je kao:

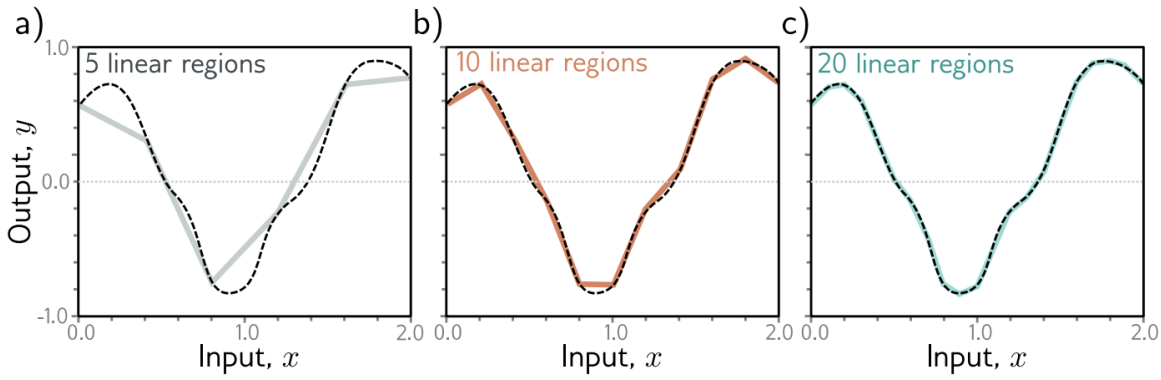
$$ReLU(h) = \max(0, h) \tag{2.10}$$

što znači da propušta samo pozitivne vrijednosti linearne funkcije, dok negativne vrijednosti postavlja na nulu. Ovaj proces je ilustriran na Slici 2.7 (d, e, f). Upravo ovaj korak uvodi nelinearnost u model, omogućujući mreži da aproksimira složenije funkcije. Izlazi iz aktivacijske funkcije zatim se prosljeđuju u izlazni sloj, gdje se množe s odgovarajućim težinama izlaznog neurona, što je prikazano na Slici 2.7 (g, h, i). Konačno, izlazi skrivenog sloja se zbrajaju kako bi se formirao konačni izlaz mreže, što je vidljivo na Slici 2.7 (j). Važno je uočiti da rezultirajuća izlazna funkcija može imati najviše n+1 linearnih podintervala, gdje je n broj neurona u skrivenom sloju. Ovo proizlazi iz činjenice da se za svaku točku u domeni ulaza zbrajaju doprinosi svih neurona skrivenog sloja, pri čemu svaki neuron može biti aktivan (pozitivan izlaz) ili neaktivan (nulti izlaz). Ovaj primjer jasno demonstrira kako plitka neuronska mreža s ReLU aktivacijskom funkcijom transformira linearni ulaz u nelinearni izlaz. Svaki neuron u skrivenom sloju doprinosi jednom ”zglobnom” točkom, gdje se mi-



Slika 2.7: Prikaz računanja izlaza za mrežu prikazanu na Slici 2.6. [6]

jenja nagib izlazne funkcije. Rezultat je po dijelovima linearna funkcija koja može aproksimirati širok spektar nelinearnih odnosa. Nadalje, ovakva struktura mreže ilustrira temeljni princip univerzalnog teorema aproksimacije za neuronske mreže. Povećanjem broja neurona u skrivenom sloju, mreža postaje sposobna aproksimirati sve složenije funkcije s većom preciznošću. Na Slici 2.8 prikazane su izlazne funkcije za mreže s jednim skrivenim slojem i $n \in 4, 9, 19$ neurona. Prikazane mreže pokazuju kako dodavanjem neurona i podjelom na sve manje podintervale možemo dobiti



Slika 2.8: Prikaz izlaza plitke neuronske mreže s n neurona. [6]

sve preciznije izlaze. Ovo svojstvo čini neuronske mreže iznimno fleksibilnim alatom u modeliranju kompleksnih sustava i analizi podataka. Prikazani primjer ne samo da objašnjava mehanizam funkcioniranja plitke neuronske mreže s ReLU aktivacijom već i pruža uvid u njezinu sposobnost modeliranja nelinearnih odnosa.

2.4 *Funkcija gubitka*

Funkcija gubitka (eng. *loss function*) je funkcija parametara modela $(w_0, w_1, ..., w_n)$, koja mjeri koliko model loše predviđa izlazne vrijednosti u odnosu na stvarne vrijednosti. Funkcija gubitka pokazuje koliko je hipoteza za određeni skup parametara udaljena od stvarnih vrijednosti u skupu podataka. Cilj korištenja funkcije gubitka je odrediti kada model radi dobro, a kada ne. Za provjeru modela koriste se označeni parovi ulazno-izlaznih podataka $\{(x_i, y_i)\}_{i=1}^{N}$. Za svaki takav par podataka funkcija

$$L_i(f(x_i, W), y_i), \tag{2.11}$$

provjerava točnost predviđanja modela. Ukupna vrijednost funkcija gubitka modela računa se kao prosječna vrijednost zbroja pojedinačnih funkcija svakog para označenih podataka:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i). \tag{2.12}$$

U strojnom učenju koriste se brojne funkcije gubitka, ovisno o vrsti zadatka koju rješava. Također, može se koristiti više funkcija gubitka za jedan model kako bi se odredilo koji ga najbolje opisuje. Primjeri nekih funkcija gubitka:

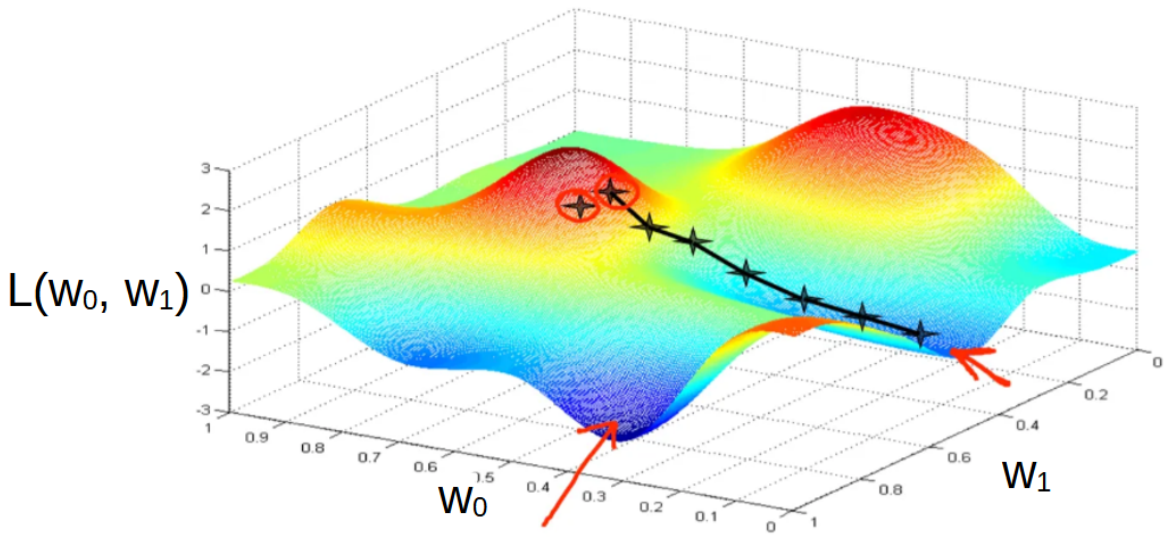
Srednja kvadratna pogreška

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - (f(x_i, W))^2, \tag{2.13}$$

često se koristi za regresijske probleme. Računa prosjek kvadrata razlike između stvarnih vrijednosti i predviđenih vrijednosti modela.

2.5 Propagacija pogreške unatrag

Propagacija pogreške unatrag, poznata i kao (engl. *backpropagation*), jedan je od ključnih algoritama koji omogućuju treniranje neuronskih mreža. Razvijen krajem 1980-ih, u kombinaciji s tehnikom gradijentnog spusta, postao je nezamjenjiv alat za optimizaciju težina u neuronskim mrežama. Cilj algoritma je minimizirati funkciju gubitka, kako bi model što točnije predviđao ispravne izlaze. Pojam propagacija unatrag odnosi se na prenošenje pogreške od izlaznog sloja natrag kroz mrežu, prema ulaznim slojevima, kako bi se prilagodile težine svih povezanih neurona. Osnovna svrha propagacije pogreške je ažuriranje težina neurona kako bi se minimizirala funkcija gubitka $L(w_0, w_1, ..., w_n)$. Funkcija gubitka mjeri koliko se predikcija mreže razlikuje od stvarnog izlaza. Pronalaženje minimuma funkcije gubitka ključno je za treniranje neuronske mreže, a ovaj proces vodi do bolje točnosti modela. Minimum funkcije, odnosno točka gdje je gradijent nula, označava mjesto u kojem je mreža najbolje optimizirana za zadani zadatak. Gradijent funkcije gubitka $\nabla L(w_0, w_1, ..., w_n)$

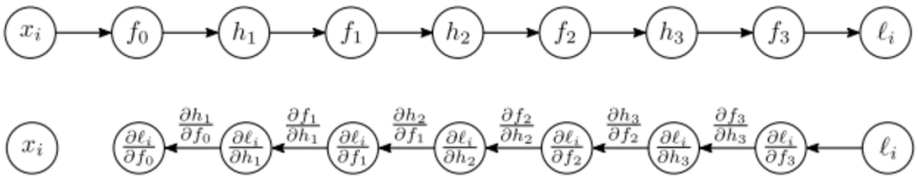


Slika 2.9: Primjer gradijentnog spusta u lokalni minimum. [9]

pokazuje smjer najveće promjene u funkciji. Gradijentni spust koristi ovaj gradijent kako bi pronašao točku u kojoj funkcija dostiže minimum, smanjujući funkciju gubitka za male korake koji su proporcionalni gradijentu. To se postiže uzimajući negativni gradijent, čime se omogućuje kretanje u smjeru najvećeg pada funkcije. Algoritam se iterativno ponavlja, dok se ne pronađe lokalni ili globalni minimum. Primjer gradijentnog spusta za dvije težine, w_0 i w_1, prikazan je na slici (Slika 2.9). Svaka inicijalizacija težina vodi mrežu prema određenom minimumu, što implicira da različite početne težine mogu dovesti do različitih lokalnih minimuma. Ovaj fenomen je poznat kao problem lokalnog minimuma u optimizaciji neuronskih mreža. Ažuriranje težina svakog neurona, za zadanu funkciju gubitka, računa se prema sljedećem izrazu:

$$w_j = w_j^{'} - \alpha \nabla L(w). \tag{2.14}$$

Gdje je $w_j^{'}$ početna vrijednost težine, dok je w_j nova vrijednost nakon ažuriranja. Gradijent $\frac{\partial}{\partial w_j} L(w)$ označava smjer najvećeg opadanja funkcije gubitka, dok α predstavlja faktor učenja (engl. *learning rate*), koji određuje veličinu koraka. Faktor učenja može biti konstantan, ali često se prilagođava kako bi optimizacija bila što učinkovitija. Ovaj postupak omogućuje mreži da uči iz grešaka i prilagođava svoja predviđanja tijekom vremena. Proces propagacije pogreške unatrag započinje *forward passom*, gdje se ulazni podaci propuštaju kroz mrežu da bi se dobio izlaz. Zatim se izračunava greška usporedbom predviđenog izlaza s pravim vrijednostima. U backward passu, ova greška se propagira unatrag kroz mrežu. Za svaki neuron, propagacija pogreške unatrag izračunava koliko je taj neuron pridonio ukupnoj grešci, a zatim kako bi se trebale ažurirati težine koje vode do tog neurona da bi se smanjila greška.



Slika 2.10: Primjer propagacije pogreške unatrag. [9]

Ključni koncept u backpropagaciji je lančano pravilo derivacija. Ovo pravilo omogućuje da se gradijent kompleksne kompozitne funkcije (kakva je neuronska mreža) izračuna kao produkt jednostavnijih gradijenata. Za neuron u skrivenom sloju, gra-

dijent se računa kombiniranjem gradijenta iz sljedećeg sloja s lokalnim gradijentom aktivacijske funkcije tog neurona. Matematički, za težinu w_{ij} koja povezuje neuron i u jednom sloju s neuronom j u sljedećem sloju, gradijent se može izraziti kao:

$$\frac{\partial \mathrm{w}}{\partial \mathrm{w}_{ij}} = \frac{\partial \mathrm{w}}{\partial \mathrm{a}_j} \cdot \frac{\partial \mathrm{a}_j}{\partial \mathrm{z}_j} \cdot \frac{\partial \mathrm{z}_j}{\partial \mathrm{w}_{ij}}, \tag{2.15}$$

gdje je E funkcija gubitka, a_j je aktivacija neurona j, a z_j je težinska suma ulaza u neuron j. Ovaj izraz pokazuje kako se greška propagira unatrag $\frac{\partial \mathrm{w}}{\partial \mathrm{a}_j}$, kako se mijenja aktivacijska funkcija $\frac{\partial \mathrm{a}_j}{\partial \mathrm{z}_j}$, i kako težina utječe na ulaz neurona $\frac{\partial \mathrm{z}_j}{\partial \mathrm{w}_{ij}}$. Efikasnost backpropagacije leži u činjenici da se mnogi od ovih parcijalnih derivata mogu ponovno koristiti pri izračunu gradijenata za različite težine, čime se značajno smanjuje računska složenost u usporedbi s direktnim izračunom svakog gradijenta zasebno. Međutim, backpropagacija i gradijent spusta suočavaju se s nekoliko izazova u praksi. Jedan od glavnih problema je nestajući gradijent, gdje gradijenti postaju vrlo mali kako se propagiraju unatrag kroz duboke mreže, što otežava učenje ranijih slojeva. Ovaj problem se često javlja s određenim aktivacijskim funkcijama poput sigmoidne ili tanh funkcije, koje imaju gradijente ograničene na mali raspon. Za rješavanje ovog problema, razvijene su različite tehnike. Korištenje ReLU aktivacijske funkcije pomaže jer njezin gradijent ostaje konstantan za pozitivne ulaze. Normalizacija slojeva (batch normalization) također pomaže stabiliziranjem distribucije aktivacija kroz mrežu. Unatoč ovim izazovima, backpropagacija i gradijent spusta ostaju fundamentalni za treniranje neuronskih mreža. Njihova jednostavnost i efikasnost, u kombinaciji s modernim unapređenjima, omogućili su razvoj izuzetno dubokih i složenih modela koji postižu vrhunske rezultate u različitim domenama, od računalnog vida do obrade prirodnog jezika.

2.6 Multimodalni jezični modeli

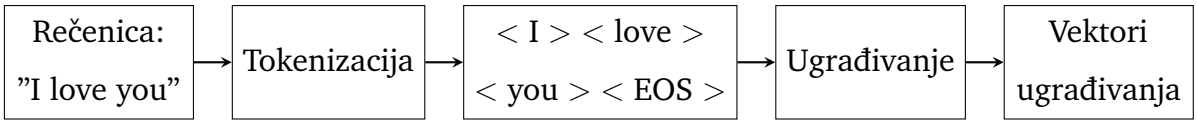
Razvoj neuronskih mreža od jednostavnih struktura do složenih multimodalnih jezičnih modela bio je postepen proces obilježen brojnim inovacijama. Početne neuronske mreže, poput perceptrona, imale su ograničene mogućnosti i mogle su rješavati samo linearne probleme. Uvođenje višeslojnih mreža i algoritma propagacije pogreške unatrag omogućilo je rješavanje složenijih, nelinearnih problema. Značajan napredak dogodio se s razvojem dubokih neuronskih mreža, koje su imale mnogo

slojeva i mogle su učiti složenije reprezentacije podataka. Ovo je dovelo do razvoja konvolucijskih neuronskih mreža (CNN) za obradu slika i rekurentnih neuronskih mreža (RNN) za obradu sekvencijalnih podataka poput teksta. Sljedeći veliki korak bio je razvoj modela pažnje i transformera, koji su revolucionirali obradu prirodnog jezika. Ovi modeli su omogućili učinkovitu obradu dugih sekvenci i doveli do stvaranja moćnih jezičnih modela. Paralelno s ovim razvojem, istraživači su radili na integraciji različitih modaliteta podataka, što je dovelo do stvaranja multimodalnih jezičnih modela. Za razliku od klasičnih jezičnih modela koji samo obrađuju tekst, multimodalni jezični modeli obrađuju više modaliteta (tekst, slika, zvuk, video) generirajući složenije i preciznije odgovore. Primjerice, model koji može analizirati i tekst i slike može bolje razumjeti sadržaj koji uključuje oboje, poput ilustriranog članka. Ova sposobnost otvara vrata mnogim naprednim aplikacijama, uključujući poboljšano pretraživanje slika, automatsko generiranje opisa za vizualni sadržaj, i stvaranje sofisticiranijih virtualnih asistenata koji mogu interpretirati i odgovarati na upite koji uključuju više vrsta medija. Ovi modeli mogu generirati tekst koji je kontekstualno relevantan za danu sliku, ili čak stvoriti slike na temelju tekstualnog opisa. U području robotike i interakcije čovjek-računalo, multimodalni modeli omogućuju stvaranje inteligentnijih sustava koji mogu interpretirati ljudske geste, izraze lica i govor, pružajući prirodnije i intuitivnije sučelje za interakciju. Ovo je posebno važno za razvoj naprednih asistenata u zdravstvu, obrazovanju i korisničkoj podršci. S kontinuiranim napretkom u arhitekturama modela, tehnikama treninga i dostupnosti velikih multimodalnih skupova podataka. Očekuje se da će ovi modeli igrati ključnu ulogu u razvoju sljedeće generacije AI sustava, omogućujući još prirodniju i intuitivniju interakciju između ljudi i strojeva.

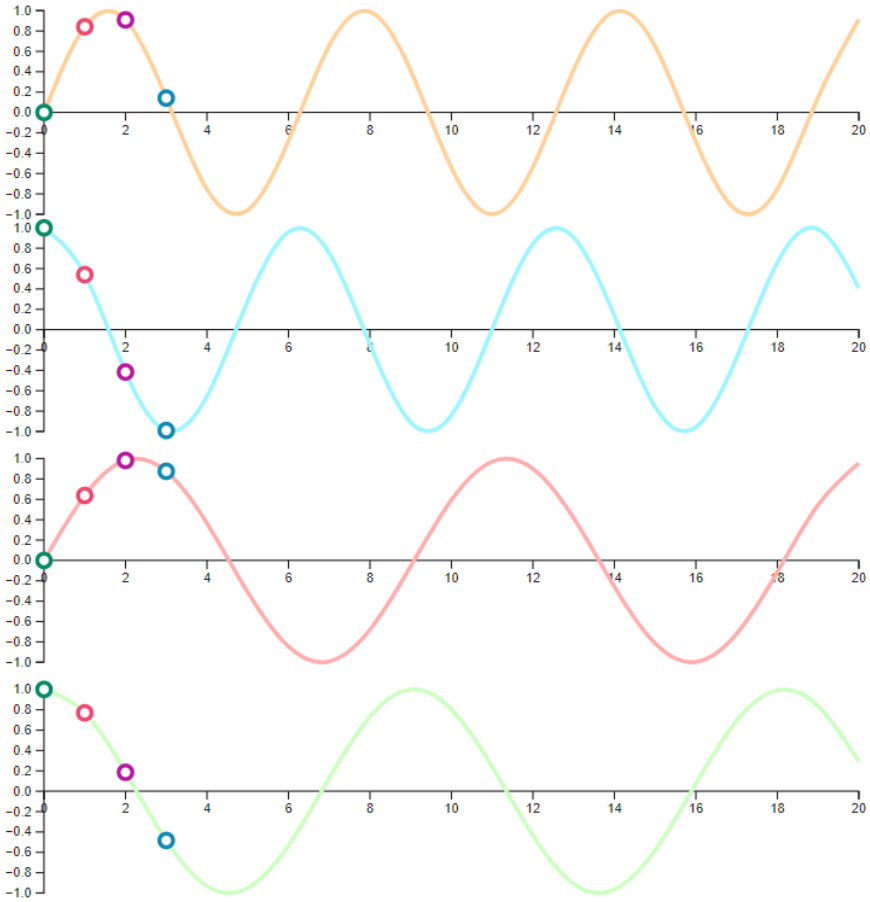
2.7 Transformeri

Transformeri predstavljaju revolucionarnu arhitekturu neuronske mreže, prvi put predstavljenu u značajnom radu ”Attention Is All You Need” [12]. Ova inovativna arhitektura postala je temelj mnogih suvremenih multimodalnih jezičnih modela, omogućujući značajan napredak u području obrade prirodnog jezika i drugih sekvencijalnih zadataka. Za ilustraciju osnovnog principa transformera, analizirat ćemo proces prevođenja jednostavne engleske rečenice ”I love you” na francuski ”Je t’aime”.

Ulazni podaci se inicijalno segmentiraju u manje jedinice zvane tokeni, procesom poznatim kao *tokenizacija*. U jezičnim modelima, tokeni mogu predstavljati cijele riječi, dijelove riječi ili pojedinačne znakove, dok kod obrade slika mogu biti segmenti slike ili pojedinačni pikseli. U našem primjeru, tokenizacija dijeli izraz na četiri tokena: svakoj riječi dodijeljen je jedan token, a posljednji token $<$ EOS $>$ označava kraj teksta (engl. *end of sequence*).



Nakon tokenizacije ulazne rečenice "I love you", svaka riječ se pretvara u vektorsku reprezentaciju pomoću procesa ugrađivanja (engl. *embedding layer*). Ugrađivanje, ključna komponenta mnogih modela dubokog učenja, pretvara diskretne ulazne podatke u kontinuirane vektorske reprezentacije. Glavna prednost ugrađivanja je njegova sposobnost zahvaćanja semantičkih odnosa između riječi: riječi sličnog značenja često se nalaze blizu jedna drugoj u vektorskom prostoru. Ovo svojstvo čini ugrađivanje izuzetno korisnim u raznim NLP zadacima poput strojnog prevođenja, klasifikacije teksta ili analize sentimenta. Dodatno, ugrađivanje pomaže u smanjenju dimenzionalnosti ulaznih podataka, značajno poboljšavajući računalnu učinkovitost. Prije ulaska u transformer, provodi se još jedan ključan proces: *pozicijsko kodiranje*. Ovaj postupak generira jedinstveni numerički zapis za svaku poziciju u ulaznoj sekvenci koristeći sinusne i kosinusne funkcije različitih frekvencija. Svaka dimenzija vektora koristi drugu frekvenciju, stvarajući kompleksan uzorak jedinstven za svaku poziciju. Prije ulaska u transformer, provodi se *pozicijsko kodiranje*. Ovaj postupak generira jedinstveni numerički zapis za svaku poziciju u ulaznoj sekvenci koristeći sinusne i kosinusne funkcije različitih frekvencija. Prednosti ovog pristupa su višestruke: omogućuje modelu razlikovanje riječi na različitim pozicijama, čak i kad su identične, te olakšava izračunavanje relativnih pozicija između riječi, što je ključno za razumijevanje strukture rečenice. Na Slici 2.11 prikazane su funkcije sinusa i kosinusa koje se koriste za pozicijsko kodiranje. Os apscisa predstavlja poziciju riječi u rečenici, dok os ordinata prikazuje vrijednosti kodiranja. Svaka riječ u rečenici označena je kružićem različite boje: zelenim za prvu riječ 'I', crvenim za 'love', ljubičastim za 'you', te plavim za oznaku kraja rečenice 'EOS'. Pozicijsko kodiranje se provodi za



Slika 2.11: Primjer pozicijskog kodiranja. [17]

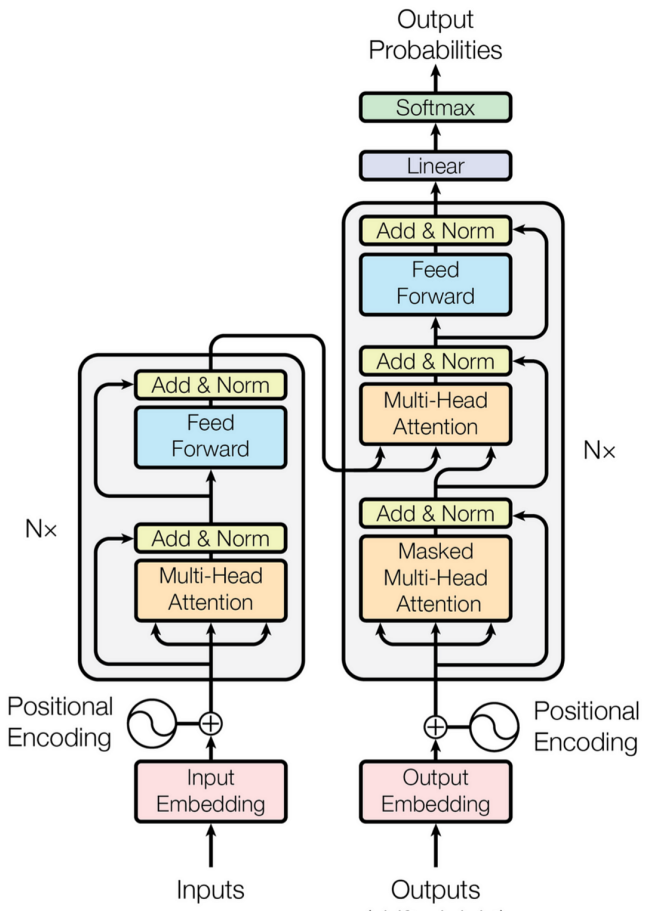
svaku dimenziju vektora riječi. U ovom primjeru prikazane su četiri dimenzije, svaka predstavljena zasebnom sinusnom ili kosinusnom funkcijom različite frekvencije. Za svaku riječ, vrijednosti na ordinati se zbrajaju kako bi se dobio jedinstveni pozicijski kod. Rezultate ovog procesa prikazali smo u Tablici 2.1.

	< I >	< love >	< you >	< EOS >
1	0.000	0.841	0.909	0.141
2	1.000	0.540	-0.416	-0.990
3	0.000	0.638	0.983	0.875
4	1.000	0.770	0.186	-0.484

Tablica 2.1: Vrijednosti pozicijskog kodiranja tokena

Konačno, pozicijski kodovi se zbrajaju s vektorima riječi, stvarajući obogaćene vektore koji sadrže informacije o značenju riječi i njihovu položaju u rečenici. Ovi vektori služe kao ulaz u transformer model. Transformeri se sastoje od dva glavna dijela: enkodera i dekodera (Slika 2.12). Enkoder preuzima ulaznu sekvencu tokena i pretvara ih u unutarnje reprezentacije, dok dekoder koristi te reprezentacije za generiranje iz-

lazne sekvence. Tijekom treniranja transformera primjenjuje se tehnika poznata kao prisilno podučavanje (engl. *Teacher Forcing*). Ova metoda osigurava da model koristi stvarne vrijednosti iz skupa za treniranje kao ulaz za sljedeći korak, umjesto vlastitih predviđenih izlaza. Važan aspekt arhitekture transformera je mehanizam maskiranja u dekoderu, koji osigurava da predikcija svake riječi ovisi samo o prethodnim riječima, čime se očuva kauzalnost i sprječava korištenje informacija iz budućih tokena.



Slika 2.12: Arhitektura transformera. [11]

Jedna od ključnih inovacija transformera je mehanizam pažnje (engl. *attention*), koji omogućava modelu da efikasno obradi dugoročne zavisnosti između riječi ili tokena u sekvenci. Ovaj mehanizam predstavlja značajan napredak u odnosu na prethodne arhitekture koje su često imale poteškoća s obradom dugih sekvenci. Mehanizam pažnje funkcionira tako da omogućuje svakom tokenu da "obrati pažnju" na sve ostale tokene u sekvenci, ovisno o njihovoj relevantnosti. Ovaj proces pomaže modelu da identificira koje riječi unutar rečenice imaju najveći utjecaj na trenutni token, neovisno o njihovoj udaljenosti u sekvenci. Za dani primjer 'I love you', mehanizam

pažnje uspoređuje token $< \mathrm{I} >$ sa svim drugim tokenima, uključujući i sebe. Taj postupak se ponavlja za svaki token u sekvenci. Konkretno: Token $< \mathrm{I} >$ se uspoređuje s $< \mathrm{I} >$, $<$ love $>$, $<$ you $>$ i $<$ EOS $>$. Zatim se $<$ love $>$ uspoređuje sa svim tokenima. Isto se ponavlja za $<$ you $>$ i $<$ EOS $>$. Ove usporedbe rezultiraju "težinama pažnje" koje kvantificiraju koliko je svaki token relevantan za obradu trenutnog tokena. Na primjer, za token $<$ love $>$, model može dodijeliti veću težinu tokenu $< \mathrm{I} >$ (subjekt) i $<$ you $>$ (objekt), nego tokenu $<$ EOS $>$. Važno je napomenuti da transformer koristi višeglavu pažnju (engl. *multi-head attention*), što znači da se ovaj proces pažnje ponavlja nekoliko puta paralelno, svaki put s različitim parametrima. Ovo omogućuje modelu da uhvati različite vrste odnosa između tokena, poput gramatičkih veza, semantičkih sličnosti ili kontekstualnih asocijacija. Mehanizam pažnje u transformerima započinje s ulaznom sekvencom vektora $\mathbf{E} = (\mathbf{E}_1, \mathbf{E}_2, ..., \mathbf{E}_n)$, gdje svaki $\mathbf{E}_i$ predstavlja embeddirani vektor i-te riječi ili tokena u sekvenci. Za svaki element ove sekvence, mehanizam pažnje izračunava vektor upita (**Q**), vektor ključa (**K**) i vektor vrijednosti (**V**). Ovi vektori se dobivaju množenjem ulaznog vektora $\mathbf{E}_i$ s odgovarajućim matricama težina:

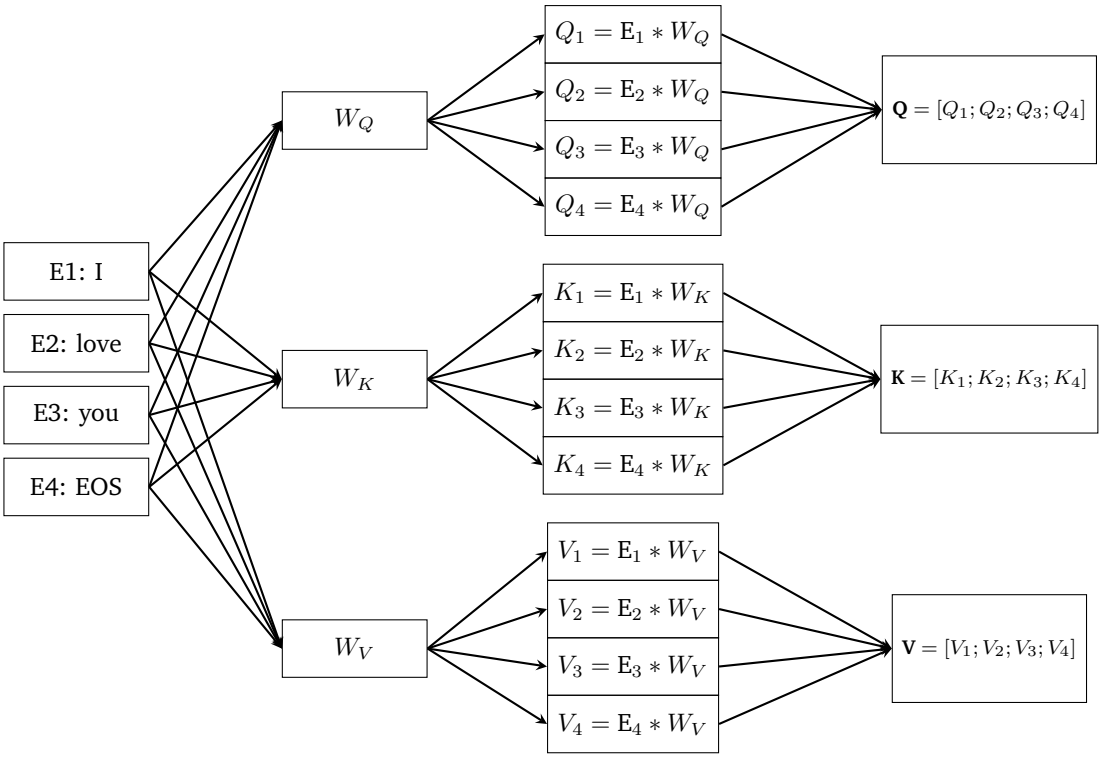
$$\begin{aligned} \mathbf{Q}_i &= \mathbf{E}_i W_Q \\ \mathbf{K}_i &= \mathbf{E}_i W_K \\ \mathbf{V}_i &= \mathbf{E}_i W_V \end{aligned} \tag{2.16}$$

gdje su W_Q, W_K i W_V naučene matrice težina specifične za upite, ključeve i vrijednosti. Ovaj proces se ponavlja za svaki token u ulaznoj sekvenci, rezultirajući matricama Q, K i V:

$$\begin{aligned} \mathbf{Q} &= \mathbf{E} W_Q \\ \mathbf{K} &= \mathbf{E} W_K \\ \mathbf{V} &= \mathbf{E} W_V \end{aligned} \tag{2.17}$$

gdje je **E** matrica ulaznih vektora, a svaki redak u **Q**, **K** i **V** odgovara jednom tokenu iz ulazne sekvence. Dijagram na Slici 2.13 prikazuje proces izračunavanja vektora upita (**Q**), ključa (**K**) i vrijednosti (**V**) u mehanizmu pažnje transformera, korištenjem primjera rečenice "I love you".

Korištenjem dobivenih vektora, za svaku riječ u sekvenci računamo umnožak njene vrijednosti upita sa vrijednosti ključa svake riječi u sekvenci. Za riječ 'I' i njenu vri-



Slika 2.13: Dijagram mehanizma pažnje za rečenicu "I love you"

jednost vektora $\mathbf{Q}_1$ računamo umnožak sa svakom vrijednosti ključa $\mathbf{K}_1, \mathbf{K}_2, \mathbf{K}_3 i \mathbf{K}_1$,. Na kraju dobiveni rezultat sumiramo kako bi dobili $\sum_i^4 \mathbf{Q}_1 * \mathbf{K}_i$. Na sumu djelujemo softmax funkcijom kako bi rezultate sveli na interval između 0 i 1. dobiven rezultat množimo s vrijednosti vektora $\mathbf{V}_1$. Taj proces ponavljamo za svaku riječ u sekvenci te se dobiva matrični zapis:

$$\Delta\mathbf{E} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathrm{T}}}{\sqrt{d_n}}\right)\mathbf{V}. \tag{2.18}$$

Izraz $\sqrt{d_n}$ koristi se za stabilizaciju rezultata, gdje je d_n dimenzionalnost vektora (u našem slučaju 4). Dobiveni vektor $\Delta\mathbf{E}$ zatim se dodaje početnom vektoru kako bi se prilagodio kontekstu:

$$\mathbf{E'} = \mathbf{E} + \Delta\mathbf{E} \tag{2.19}$$

Ova operacija naziva se glava pažnje. Transformeri obično koriste višestruke glave pažnje, što omogućuje modelu da paralelno obradi više različitih kontekstualnih informacija. Omogućavajući modelu da precizno poveže kontekst cijele rečenice. Konačni izgled vektora ulaza prilagođava se:

$$\mathbf{E}' = \mathbf{E} + \sum_i^N \Delta\mathbf{E}_i. \tag{2.20}$$

Nakon što vektori prođu kroz blok pažnje, slijedi sloj feed-forward neuronske mreže. U ovom koraku, tokeni međusobno ne komuniciraju, već se svaki token obrađuje neovisno, što omogućava modelu da obrađuje podatke paralelno. Proces unutar feed-forward mreže sastoji se od niza linearnih transformacija, koje prilagođavaju vektorske reprezentacije tokena, pomažući modelu da bolje razumije kompleksne relacije u podacima. Ulazni vektori **E**, koji su već prilagođeni pomoću mehanizma pažnje, dalje se obrađuju tako da se množe s matricom težina **R**. Ovaj proces može se izraziti kao linearna transformacija:

$$\mathbf{H} = \mathbf{E} \cdot \mathbf{R} + \mathbf{b} \tag{2.21}$$

gdje **b** predstavlja dodatni vektor pomaka (engl. *bias*). Ova operacija linearno transformira ulazne vektore, ali kako bi model mogao bolje naučiti nelinearne odnose između podataka, rezultat se propušta kroz nelinearnu aktivacijsku funkciju, najčešće ReLU (Rectified Linear Unit):

$$ReLU(x) = \max(0, x). \tag{2.22}$$

Funkcija ReLU postavlja sve negativne vrijednosti u dobivenom vektoru na nulu, što pomaže modelu u upravljanju složenim nelinearnim obrascima unutar podataka. Nakon primjene ReLU-a, dobiveni vektor ponovno prolazi kroz još jednu linearnu transformaciju, u kojoj se ponovno množi s matricom težina i dodaje vektor pomaka:

$$\mathbf{E}' = \mathbf{H} \cdot \mathbf{R}' + \mathbf{b}' \tag{2.23}$$

Ova druga transformacija vraća vektor u izvornu dimenzionalnost, čime se omogućuje da vektor zadrži odgovarajuću reprezentaciju za daljnju obradu u modelu. Feed-forward slojevi su identični za svaki element sekvence, što omogućuje paralelnu obradu podataka. Na kraju, nakon višestrukih prolaza kroz slojeve pažnje i feed-forward neuronske mreže, dobiven je konačni izgled ulaznog vektora. U dekoder dijelu dekodiranje započinje s posebnim tokenom ”start” i generira prijevod riječ po riječ. Za rečenicu ’I love you’ generira prijevod ”Je amour toi”. Potom kao i u enkoder dijelu izračunavaju se matrice **Q**, **K** i **V**. Za dobivene matrice računamo kao u izrazu 2.18 povezanosti riječi, ali pošto prevodimo s engleskog na francuski ne želimo samo

provjeravati za riječi u dekoder sekvenci nego i za riječi iz enkoder sekvence. Za generiranje prve riječi ”Je”, dekoder koristi svoje Q vektore za ”Je” i uspoređuje ih s K vektorima enkodera i dekodera. Najviša težina pažnje (0.94) je na riječi ”I”, što rezultira generiranjem ”Je”. Slično, za generiranje ”t’”, Q vektor za ”t’” se uspoređuje s K vektorima enkodera, pri čemu je najveća težina (0.88) na riječi ”you”. Konačno, za ”aime”, Q vektor se ponovno uspoređuje s K vektorima enkodera, s najvišom težinom (0.95) na riječi ”love”.

	I	love	you
je	0.94	0.02	0.04
t’	0.11	0.01	0.88
aime	0.03	0.95	0.02

Tablica 2.2: Matrica poravnanja za prijevod ”I love you” na francuski ”Je t’aime”

Ovaj proces se može vizualizirati kroz matricu poravnanja koja pokazuje kako se pažnja distribuira tijekom procesa dekodiranja. Za našu rečenicu, matrica bi izgledala ovako: za ”Je”, težine su 0.94 za ”I”, 0.02 za ”love” i 0.04 za ”you”; za ”t’”, težine su 0.11, 0.01 i 0.88; a za ”aime”, težine su 0.03, 0.95 i 0.02. Visoke vrijednosti izvan dijagonale ukazuju na sofisticiranost mehanizma pažnje u hvatanju složenih jezičnih odnosa. Analiza procesa prevođenja ”I love you” u ”Je t’aime” u transformeru otkriva složenu interakciju između matrica ključa, vrijednosti i upita. Mehanizam pažnje omogućuje modelu da dinamički usmjeri fokus na relevantne dijelove ulazne sekvence, što rezultira preciznim i kontekstualno prikladnim prijevodom. Ovaj primjer ilustrira kako transformeri uspješno prevladavaju ograničenja ranijih pristupa strojnom prevođenju, omogućujući fleksibilno i efikasno rukovanje jezičnim strukturama.

3 Multimodalni jezični modeli kao asistent u nastavi informatike

U ovom poglavlju istražujemo primjenu multimodalnih jezičnih modela kao asistenata u nastavi informatike. Uspoređujemo učinkovitost modela pri rješavanju zadataka iz raznovrsnih nastavnih cjelina te analiziramo njihovu sposobnost da kroz vođeno učenje usmjeravaju učenike prema točnim rješenjima. Cilj je usporediti dva istaknuta modela koristeći pažljivo osmišljena pitanja:

- Chat GPT-4o, razvijen od strane OpenAI-a, s bazom znanja ažuriranom do listopada 2023. godine.
- Claude 3.5 Sonnet, razvijen od strane Anthropica, s bazom znanja ažuriranom do travnja 2024. godine.

Ova usporedba će nam omogućiti uvid u napredak tehnologije umjetne inteligencije u obrazovnom kontekstu te potencijalne prednosti i ograničenja svakog modela. U nastavku su prikazani razgovori s modelima. Kako bi se odgovori modela lakše vizualno razlikovali, postavljeni su u tekstualne okvire, s jasno označenim upitom i odgovorom. Prilikom rješavanja zadataka koji su zadani modelima, kreirani su primjeri koji sadrže najčešće pogreške koje učenici rade u programiranju. Na kraju ovog poglavlja, analizirat ćemo koja vrsta modela pokazuje veću uspješnost u ispravljanju grešaka i vođenju učenika prema ispravnim rješenjima. Ova usporedba pomoći će u razumijevanju prednosti i slabosti svakog modela te pružiti smjernice o tome koji model može biti korisniji u obrazovnom okruženju i zašto.

3.1 Uvjeti grananja

Prilikom obrade ove nastavne cjeline, učenici miješaju operatore za usporedbe s operatorom za dodjelu vrijednosti. Također, zaboravljaju predvidjeti sve moguće ishode grananja ili pogrešno postavljaju redoslijed provjera uvjeta.

Korisnikov upit GPT-4o modelu

Postavi se u ulogu nastavnika informatike u Hrvatskoj. Predaješ drugom razredu srednje škole. Obradio si nastavnu cjelinu: 'Uvjeti grananja'. Ja sam učenik kojem trebaš zadati zadatak iz te nastavne cjeline u Pythonu. Tvoj posao je vođenim učenjem dovesti me do rješenja zadatka.

Odgovor GPT-4o modela

Razumijem! Evo zadatka koji će ti pomoći da vježbaš uvjete grananja:

Zadatak:

Napiši Python program koji traži od korisnika da unese tri broja. Program zatim treba ispisati:

- Koji je od ta tri broja najveći.
- Koji je od ta tri broja najmanji.
- Jesu li sva tri broja jednaka.

Kada napišeš rješenje, pošalji mi svoj kod pa ćemo zajedno proći kroz njega. Ako negdje zapneš, dat ću ti smjernice da dođeš do ispravnog rješenja.

Korisnikov upit GPT-4o modelu

Evo programa:

```
a = int(input())
b = int(input())
c = int(input())
if (a > b and a > c):
    print('Najveci broj je:' + str(a))
elif (b > a and b > c):
    print('Najveci broj je:' + str(b))
if (c > a and c > b):
    print('Najveci broj je:' + str(c))
if (a < b and a < c):
    print('Najmanji broj je:' + str(a))
elif (b < a and b < c):
    print('Najmanji broj je:' + str(b))
if (c < a and c < b):
    print('Najmanji broj je:' + str(c))
if (a == b and b = c):
    print('Svi su jednaki')
```

Odgovor GPT-4o modela

Tvoj kod ima dobar temelj, ali ima nekoliko grešaka i nedosljednosti koje trebamo ispraviti. Evo nekoliko smjernica koje će ti pomoći da ih uočiš i popraviš:

- **Ispisivanje brojeva:** Kada koristiš `print`, trebaš spojiti tekst i varijablu na ispravan način. Trenutno pokušavaš dodati broj (`a`, `b`, `c`) na tekstualni niz bez pretvaranja broja u string, što će uzrokovati grešku. U Pythonu to radiš pomoću funkcije `str()` ili tako da koristiš f-stringove.
- **Logička greška u zadnjem uvjetu:** U zadnjem uvjetu, gdje provjeravaš jesu li svi brojevi jednaki, koristiš `b = c` što je greška. Trebalo bi `b == c`.
- **Logika za određivanje najmanjeg i najvećeg broja:** Trenutno tvoj kod miješa uvjete, što može dovesti do krivih rezultata. Moramo razdvojiti te uvjete i osigurati da svaki blok koda pravilno identificira i ispisuje najveći i najmanji broj.
- **Redoslijed ispisa:** Trenutno ispisuješ ”Najmanji broj je:” i ”Najveći broj je:” u različitim uvjetima, što može zbuniti korisnika. Potrebno je ispraviti tekstualni ispis.

Korisnikov upit GPT-4o modelu

```
a = int(input())
b = int(input())
c = int(input())
if(a == b and b == c):
    print('Svi su jednaki')
else:
    if (a > b and a > c):
        print(f'Najveci broj je: {a}')
    elif (b > a and b > c):
        print(f'Najveci broj je: {b}')
    elif(c > a and c > b):
        print(f'Najveci broj je: {c}')

    if (a < b and a < c):
        print(f'Najmanji broj je: {a}')
    elif (b < a and b < c):
        print(f'Najmanji broj je: {b}')
    elif(c < a and c < b):
        print(f'Najmanji broj je: {c}')
```

Odgovor GPT-4o modela

Tvoj kod je gotovo ispravan, ali još uvijek ima jedan manji problem koji treba riješiti kako bi sve funkcioniralo kako treba. Naime, kod u kojem pronalaziš najmanji i najveći broj može imati problema u slučajevima kada se brojevi izjednače. Također, postoji nekoliko poboljšanja u kodu koje će učiniti kod jasnijim i učinkovitijim.

Poboljšanja:

- **Prepoznavanje najmanjeg i najvećeg broja:** Treba osigurati da se brojevi pravilno provjeravaju bez preklapanja uvjeta. Ako u jednom uvjetu pronađeš najveći broj, provjeri sve preostale uvjete za najmanji broj.
- **Redoslijed ispitivanja:** S obzirom da si već provjerio slučaj kada su svi brojevi jednaki, možeš zanemariti taj slučaj u provjeri za najmanji i najveći broj.

Evo ispravljenog koda koji poboljšava ove aspekte:

```
a = int(input())
b = int(input())
c = int(input())
if a == b and b == c:
    print('Svi su brojevi jednaki')
else:
    if a >= b and a >= c:
        print(f'Najveci broj je: {a}')
    elif b >= a and b >= c:
        print(f'Najveci broj je: {b}')
    else:
        print(f'Najveci broj je: {c}')
    if a <= b and a <= c:
        print(f'Najmanji broj je: {a}')
    elif b <= a and b <= c:
        print(f'Najmanji broj je: {b}')
    else:
        print(f'Najmanji broj je: {c}')
```

Isto pitanje postavlja se i Claudu 3.5.

Korisnikov upit Claude 3.5 Sonnet modelu

Postavi se u ulogu nastavnika informatike u Hrvatskoj. Predaješ drugom razredu srednje škole. Obradio si nastavnu cjelinu: 'Uvjeti grananja'. Ja sam učenik kojem trebaš zadati zadatak iz te nastavne cjeline u Pythonu. Tvoj posao je vođenim učenjem dovesti me do rješenja zadatka.

Odgovor Claude 3.5 Sonnet modela

Drago mi je što ste spremni za vježbu iz nastavne cjeline o grananju u Pythonu. Evo zadatka za vas:

Zadatak: Napišite program koji će korisniku omogućiti unos temperature u Celzijevim stupnjevima. Program treba ispisati odgovarajuću poruku ovisno o unesenoj temperaturi:

- Ako je temperatura ispod 0°C, ispišite "Ledeno je!"
- Ako je temperatura između 0°C i 10°C (uključujući 0°C), ispišite "Hladno je."
- Ako je temperatura između 10°C i 20°C (uključujući 10°C), ispišite "Ugodno je."
- Ako je temperatura između 20°C i 30°C (uključujući 20°C), ispišite "Toplo je."
- Ako je temperatura 30°C ili viša, ispišite "Vruće je!"

Započnimo s osnovnom strukturom programa. Koji bi bio prvi korak u rješavanju ovog zadatka?

Korisnikov upit Claude 3.5 Sonnet modelu

```
temp = int(input())
if (temp < 0):
    print('Ledeno je')
elif (temp >= 0 ):
    if(temp < 10):
        print('Hladno je')
    elif(temp < 20):
        print('Ugodno je')
    else:
        print('Toplo je')

if(temp > 30):
    print('Vruce je')
```

Odgovor Claude 3.5 Sonnet modela

Hvala vam na vašem pokušaju! Vaš kod je dobar početak i pokazuje razumijevanje koncepta grananja. Međutim, postoji nekoliko stvari koje možemo poboljšati. Hajdemo zajedno proći kroz kod i napraviti neke izmjene.

Unos temperature je dobar:

```
temp = int(input())
```

Prvi uvjet je točan:

```
if (temp < 0):
    print('Ledeno je!')
```

Za ostale uvjete, možemo koristiti `elif` umjesto ugniježđenih `if` naredbi. To će učiniti kod čitljivijim:

```
if (temp >= 0 and temp < 10):
    print('Hladno je.')
elif (temp >= 10 and temp < 20):
    print('Ugodno je.')
elif (temp >= 20 and temp < 30):
    print('Toplo je.')
elif (temp >= 30):
    print('Vruce je!')
```

Primijetite da ne trebamo poseban uvjet za temperature iznad 30°C jer će to biti pokriveno `else` dijelom. Što mislite o ovim izmjenama? Možete li objasniti zašto je ova struktura bolja od one s ugniježđenim `if` naredbama?

Korisnikov upit Claude 3.5 Sonnet modelu

Ovakav kod neće ispisivati dva različita odgovora za temperature iznad 30°C, za razliku od moga.

Odgovor Claude 3.5 Sonnet modela

Imate pravo, hvala na tom zapažanju! To je izvrsno uočeno. U vašem originalnom kodu postojala je mogućnost dvostrukog ispisa za temperature iznad 30°C, što nije bilo poželjno. Hajdemo to detaljnije razmotriti:

U vašem originalnom kodu, za temperaturu iznad 30°C, program bi prvo ušao u else dio unutar prvog if-elif bloka i ispisao "Toplo je". Zatim bi, neovisno o prethodnom uvjetu, provjerio je li temperatura veća od 30°C i ispisao "Vruće je".

To bi rezultiralo dvostrukim ispisom za temperature iznad 30°C, što nije ono što želimo. U ispravljenoj verziji koju sam predložio, ovaj problem je riješen korištenjem međusobno isključivih elif uvjeta i završnog else uvjeta. Ova struktura osigurava da se za svaku temperaturu ispiše samo jedna poruka.

```
temp = int(input("Unesite temperaturu: "))
if temp < 0:
    print('Ledeno je!')
elif temp < 10:
    print('Hladno je.')
elif temp < 20:
    print('Ugodno je.')
elif temp < 30:
    print('Toplo je.')
else:
    print('Vruce je!')
```

Primijetite da smo dodatno pojednostavili uvjete: Ne trebamo pisati temp $>=$ 0 and temp $<$ 10 jer ako temperatura nije manja od 0 (prvi uvjet), a manja je od 10, onda je sigurno između 0 i 10. Isto vrijedi za ostale uvjete.

Ova struktura je efikasnija jer:

- Osigurava da se ispiše samo jedna poruka za svaku temperaturu.
- Kod je čitljiviji i lakši za održavanje.
- Izbjegava nepotrebne provjere (npr. ne moramo posebno provjeravati je li temp $>=$ 30).

Zadatak koji je postavio ChatGPT nešto je zahtjevniji jer zahtijeva od učenika da pre-

pozna mogućnost unosa dva ista broja i prilagodi svoj kod u skladu s tim. S druge strane, Claudeov zadatak jasnije definira što se točno traži. Primjećujemo kako modeli koriste različite metodologije u rješavanju problema. ChatGPT naglašava potencijalne greške i očekuje od učenika da ih sam uoči i ispravi te preda novi kod, dok Claude nudi specifične primjere dijelova koda koji trebaju ispravak te postavlja pitanja kako bi provjerio učenikovo razumijevanje. Kada su greške svedene na minimum, ChatGPT automatski generira točno rješenje, ne pokušavajući dublje ući u problematiku ili dodatno analizirati situaciju. Ovakav pristup može biti koristan kada je učenik blizu rješenja, ali ne nužno kad je potrebno detaljnije objašnjenje ili analiza. S druge strane, Claude 3.5, suočen s istim pitanjem, često postavlja dodatna pitanja ili nudi preciznija objašnjenja, pokušavajući osigurati da učenik u potpunosti razumije problem i korake prema ispravnom rješenju. Ova razlika u pristupu ukazuje na to da ChatGPT preferira efikasnost u zadavanju ispravnog rješenja, dok Claude teži dubljem vođenju kroz proces učenja, osiguravajući da učenik ne samo ispravi grešku već i razumije zašto je do nje došlo. U nastavnoj cjelini "Uvjeti grananja", oba modela postavili su zadatke na visokoj razini kvalitete, uzimajući u obzir da je učenicima ovo jedan od prvih susreta s programiranjem.

3.2 Petlje

U programiranju, petlje su jedan od ključnih koncepata koji omogućavaju ponavljanje određenih radnji. Međutim, mnogi učenici često nailaze na nesporazume i poteškoće prilikom njihovog razumijevanja. Miskoncepcije se javljaju kada učenici ne shvate pravilno kako petlje funkcioniraju, posebno kada je riječ o uvjetima prekida, inicijalizaciji varijabli ili brojanju interakcija. Ove pogreške mogu dovesti do beskonačnih petlji, nepredviđenih rezultata ili neefikasnih rješenja. Razumijevanje i ispravljanje ovih zabluda ključno je za razvoj uspješnih programerskih vještina.

Korisnikov upit GPT-4o modelu

Postavi se u ulogu nastavnika informatike u Hrvatskoj. Predaješ drugom razredu srednje škole. Obradio si nastavnu cjelinu: 'Petlje'. Ja sam učenik kojem trebaš zadati zadatak iz te nastavne cjeline u Pythonu. Tvoj posao je vođenim učenjem dovesti me do rješenja zadatka.

Odgovor GPT-4o modela

Napiši Python program koji će pronaći sve proste brojeve između dva broja koje unese korisnik. Detalji zadatka:

- Program treba zatražiti od korisnika da unese dva cijela broja: `a` i `b`.
- Zatim treba provjeriti koji su brojevi između `a` i `b` (uključujući oba) prosti.
- Prosti brojevi su oni brojevi koji su veći od 1 i dijeljivi su samo s 1 i samim sobom.
- Program treba ispisati sve proste brojeve u tom rasponu. Ako nema prostih brojeva, ispiši poruku koja to govori.

Korisnikov upit GPT-4o modelu

```
import math as m
a = int(input())
b = int(input())
for i in range(a,b+1):
    j = 2
    k = l = 0
    while(j < m.sqrt(i)):
        if(i%j==0):
            k = 1
            break
        else:
            j = j + 1
    if(k == 0):
        l = l + 1
        print(f'{i} je prost broj')
if(l == 0):
    print('Nema prostih brojeva')
```

Odgovor GPT-4o modela

Tvoj kod već radi jako dobro i gotovo sve je točno! No, postoje dvije manje stvari koje bismo mogli popraviti:

1. Uvjet u while petlji (while(j $<$ m.sqrt(i))): Trenutno uvjet provjerava je li j manji od $\sqrt{i}$. Međutim, to bi trebalo biti $\leq$, jer želimo provjeriti sve brojeve od 2 do $\sqrt{i}$ uključujući i korijen. Na primjer, za broj 4, trebaš provjeriti i broj 2 jer je 4 djeljivo s 2.

2. Provjera za proste brojeve: Kod ti ne prepoznaje da broj 2 i 3 trebaju biti prosti jer $\sqrt{i}$ za njih iznosi manje od 2, pa while petlja ne pokreće nikakvu provjeru. Zbog toga ti treba posebna provjera za brojeve manje od 4, jer su 2 i 3 prosti brojevi.

   Također, varijabla l koju koristiš za brojanje prostih brojeva bi trebala biti definirana izvan petlje, jer u suprotnom svaki put se postavlja na 0 unutar petlje.

Korisnikov upit GPT-4o modelu

```
import math as m
a = int(input())
b = int(input())
l = 0
for i in range(a, b + 1):
    if i > 1:
        k = 0
        for j in range(2, int(m.sqrt(i)) + 1):
            if i % j == 0:
                k = 1
                break
        if k == 0:
            l = l + 1
            print(f'{i} je prost broj')
if l == 0:
    print('Nema prostih brojeva')
```

Odgovor GPT-4o modela

Tvoj kod sada izgleda ispravno i dobro radi! Jasno provjeravaš jesu li svi brojevi jednaki, a zatim pravilno pronalaziš najveći i najmanji broj. Logika je dobro postavljena, a korištenje operatora $>=$ i $<=$ omogućava pravilno uspoređivanje brojeva, uključujući i slučajeve kada su neki brojevi jednaki. Ako ispravno radi i dobivaš očekivane rezultate, to znači da si zadatak uspješno riješio. Svaka čast! Ako imaš još nekih pitanja ili želiš dodatni zadatak, slobodno se javi!

Isto pitanje postavlja se i Claudu 3.5.

Korisnikov upit Claude 3.5 Sonnet modelu

Postavi se u ulogu nastavnika informatike u Hrvatskoj. Predaješ drugom razredu srednje škole. Obradio si nastavnu cjelinu: 'Petlje'. Ja sam učenik kojem trebaš zadati zadatak iz te nastavne cjeline u Pythonu. Tvoj posao je vođenim učenjem dovesti me do rješenja zadatka.

Odgovor Claude 3.5 Sonnet modela

Razumijem, postavit ću se u ulogu nastavnika informatike koji predaje drugom razredu srednje škole u Hrvatskoj. Upravo smo obradili nastavnu cjelinu o for petljama u Pythonu. Kao učeniku, zadati ću vam zadatak i voditi vas kroz proces rješavanja. Evo zadatka: Zadatak: Napišite program koji će ispisati tablicu množenja za brojeve od 1 do 5. Počnimo s rješavanjem korak po korak. Prvo, razmislite o strukturi ovog problema. Što mislite, koliko for petlji će nam trebati za rješavanje ovog zadatka?

Korisnikov upit Claude 3.5 Sonnet modelu

Ja mislim da će nam trebati dvije, pošto koristimo dva broja od 1 do 5.

Odgovor Claude 3.5 Sonnet modela

Odlično razmišljanje! U pravu ste, trebat će nam dvije for petlje. Jedna za prvi faktor množenja, a druga za drugi faktor. Krenimo s prvim korakom. Možete li napisati vanjsku for petlju koja će prolaziti kroz brojeve od 1 do 5?

Korisnikov upit Claude 3.5 Sonnet modelu

```
for i in range(5):
```

Odgovor Claude 3.5 Sonnet modela

Dobro ste započeli, ali imamo malu korekciju. Budući da želimo brojeve od 1 do 5 (uključujući 5), trebamo malo prilagoditi raspon. Trenutno vaša petlja ide od 0 do 4. Kako bismo to ispravili? Pokušajte modificirati vašu petlju tako da ide od 1 do 5 (uključujući 5).

Korisnikov upit Claude 3.5 Sonnet modelu

```
for i in range(1,6):
```

Odgovor Claude 3.5 Sonnet modela

Izvrsno! To je točno ono što nam treba. Vaša petlja će sada prolaziti kroz brojeve 1, 2, 3, 4 i 5. Sada nam treba unutarnja petlja. Možete li unutar ove petlje dodati još jednu for petlju koja će također prolaziti kroz brojeve od 1 do 5? Ova će predstavljati drugi faktor u našoj tablici množenja.

Korisnikov upit Claude 3.5 Sonnet modelu

```
for i in range(1,6):
    for j in range(1,6):
```

Odgovor Claude 3.5 Sonnet modela

Odlično! Vaša struktura petlji je savršena. Sada imamo dvije ugniježđene petlje koje će nam omogućiti da prođemo kroz sve kombinacije brojeva od 1 do 5. Sljedeći korak je da unutar unutarnje petlje ispišemo rezultat množenja. Možete li dodati print naredbu koja će ispisati umnožak i i j? Razmislite kako biste formatirali ispis da bude pregledan. Na primjer, možda želite da svaki rezultat zauzima određenu širinu kako bi stupci bili poravnati.

Korisnikov upit Claude 3.5 Sonnet modelu

```
for i in range (1,6):
    for j in range(1,6):
        print(i*j, end=' ')
    print()
```

Odgovor Claude 3.5 Sonnet modela

Izvrsno! Vaše rješenje je točno i elegantno. Analizirajmo što ste napravili: Koristite dvije ugniježđene petlje kako smo i planirali. Unutar unutarnje petlje, ispisujete umnožak i * j. Koristite end=' ' kako biste osigurali da se brojevi u istom redu ispisuju jedan pored drugog, odvojeni razmakom. Nakon završetka unutarnje petlje, koristite print() bez argumenata kako biste prešli u novi red za sljedeći redak tablice. Ovo rješenje će proizvesti tablicu množenja, ali možda možemo još malo poboljšati čitljivost. Što mislite o ideji da dodamo zaglavlje stupaca i redaka? To bi moglo izgledati ovako:

```
print("  ", end='')
for i in range(1, 6):
    print(f"{i:2}", end=' ')
print()

for i in range(1, 6):
    print(f"{i}:", end=' ')
    for j in range(1, 6):
        print(f"{i*j:2}", end=' ')
    print()
```

Kao i u prethodnom odjeljku, zadatak koji je postavio ChatGPT je zahtjevniji u usporedbi s onim koji je postavio Claude. Claude nas je vodio korak po korak do rješenja, pružajući detaljna uputstva i objašnjenja, a na kraju je ponudio i svoju verziju koda koja poboljšava vizualni prikaz rješenja. S druge strane, zadatak koji je zadao ChatGPT nudi veću fleksibilnost jer se može riješiti na više načina. Međutim, jedan od problema koji se javlja u njegovom pristupu je nedostatak provjere optimizacije koda. Na primjer, ChatGPT ne sugerira da bi bilo bolje koristiti while petlju umjesto for petlje kako bi se smanjio broj potrebnih prolaza i povećala efikasnost

koda. Claudeov pristup, koji uključuje vođenje kroz svaki korak i nudi poboljšanja, čini se prikladnijim za učenike koji su tek u procesu učenja, dok ChatGPT više izaziva učenike da sami prepoznaju i isprave potencijalne nedostatke, što može biti korisno u kasnijim fazama obrazovanja kada se naglasak stavlja na samostalnost.

3.3 Liste

Listovi su osnovna struktura podataka u programiranju, koja omogućava pohranu i manipulaciju nizom elemenata. Iako se čine jednostavnima, učenici često razvijaju određene zablude prilikom njihovog korištenja. Miskoncepcije se često javljaju kod pristupa elementima liste, razlike između indeksa i vrijednosti, kao i manipulacije listama kroz metode kao što su dodavanje, brisanje ili sortiranje. Ove pogreške mogu rezultirati nepravilnim ponašanjem programa.

Korisnikov upit GPT-4o modelu

Postavi se u ulogu nastavnika informatike u Hrvatskoj. Predaješ drugom razredu srednje škole. Obradio si nastavnu cjelinu: 'Liste'. Ja sam učenik kojem trebaš zadati zadatak iz te nastavne cjeline u Pythonu. Tvoj posao je vođenim učenjem dovesti me do rješenja zadatka.

Odgovor GPT-4o modela

Super! Krenimo s jednostavnim zadatkom vezanim uz liste. Cilj nam je da kroz vođeno učenje stigneš do rješenja. Zadatak: Napiši program koji će od korisnika tražiti da unese 5 brojeva. Ti brojevi trebaju biti pohranjeni u listu. Nakon toga, program treba ispisati:
Listu svih unesenih brojeva.
Najveći broj iz te liste.
Najmanji broj iz te liste.
Prosjek svih unesenih brojeva.
Prvi korak je da postaviš osnovnu strukturu programa: Traži unos brojeva od korisnika i pohrani ih u listu. Probaj napisati taj dio koda. Ako zapneš, tu sam da pomognem!

Korisnikov upit GPT-4o modelu

```
L = []
for i in range(5):
    a = int(input('Unesi broj:'))
    L.append(a)
print(L)
print(max(L))
print(min(L))
print(sum(L)/len(L))
```

Odgovor GPT-4o modela

Odlično! Tvoj kod savršeno rješava zadatak. Evo kratke analize:

Stvaraš praznu listu `L` i kroz petlju for tražiš od korisnika da unese 5 brojeva. Svaki uneseni broj dodaješ u listu pomoću metode append(). Nakon unosa brojeva, ispisuješ listu brojeva. Koristiš funkcije $max()$, $min()$ i $sum()$ kako bi ispisao najveći, najmanji i prosječni broj.

Isto pitanje postavljam Claudu 3.5.

Korisnikov upit Claude 3.5 Sonnet modelu

Postavi se u ulogu nastavnika informatike u Hrvatskoj. Predaješ drugom razredu srednje škole. Obradio si nastavnu cjelinu: 'Liste'. Ja sam učenik kojem trebaš zadati zadatak iz te nastavne cjeline u Pythonu. Tvoj posao je vođenim učenjem dovesti me do rješenja zadatka.

Odgovor Claude 3.5 Sonnet modela

U redu, postavit ću se u ulogu nastavnika informatike koji predaje drugom razredu srednje škole u Hrvatskoj. Upravo smo obradili nastavnu cjelinu o listama u Pythonu. Kao učeniku, zadat ću vam zadatak vezan uz liste i voditi vas kroz proces rješavanja. Zadatak: Napišite program koji će stvoriti listu od 5 omiljenih filmova, a zatim izvršiti sljedeće operacije:

Ispisati cijelu listu

Ispisati treći film u listi

Dodati novi film na kraj liste

Ukloniti drugi film iz liste

Ispisati konačnu listu

Počnimo s prvim korakom. Možete li napisati kod za stvaranje liste s 5 omiljenih filmova?

Korisnikov upit Claude 3.5 Sonnet modelu

```
L = ['Forrest Gump', 'Gladijator', 'Genijalni um',  'Kum',
    'Kum 2']
```

Odgovor Claude 3.5 Sonnet modela

Odlično! Stvorili ste listu s 5 filmova. To je izvrsno za početak našeg zadatka. Sada, možete li napisati naredbu za ispis cijele liste? To će biti prvi korak našeg zadatka.

Korisnikov upit Claude 3.5 Sonnet modelu

```
print(L)
```

Odgovor Claude 3.5 Sonnet modela

Izvrsno! Točno ste ispisali cijelu listu. Idemo na drugi korak našeg zadatka. Možete li napisati naredbu koja će ispisati samo treći film u listi? Prisjetite se, indeksiranje u Pythonu počinje od 0.

Korisnikov upit Claude 3.5 Sonnet modelu

```
print(L[3])
```

Odgovor Claude 3.5 Sonnet modela

Dobro razmišljanje, ali imamo malu grešku u indeksiranju. Prisjetimo se:

Prvi element ima indeks 0

Drugi element ima indeks 1

Treći element ima indeks 2

Dakle, koji indeks trebamo koristiti da bismo dobili treći film iz liste?

Korisnikov upit Claude 3.5 Sonnet modelu

```
print(L[2])
```

Odgovor Claude 3.5 Sonnet modela

Izvrsno! To je točno. Koristili ste indeks 2 da biste dobili treći element liste, što je potpuno ispravno. Sada prelazimo na treći korak našeg zadatka. Trebamo dodati novi film na kraj liste. Za ovo ćemo koristiti metodu koja dodaje element na kraj liste. Možete li se sjetiti koja je to metoda i kako bismo je koristili da dodamo, recimo, film "Inception" na kraj liste?

Korisnikov upit Claude 3.5 Sonnet modelu

```
L += 'Inception'
```

Odgovor Claude 3.5 Sonnet modela

Vaš pristup je blizu, ali imamo malu korekciju. Operator += se koristi za spajanje lista, ali u ovom slučaju dodajemo string 'Inception' kao da je lista, što može dovesti do neočekivanog rezultata - dodalo bi svako slovo kao zaseban element. Za dodavanje jednog elementa na kraj liste, koristimo metodu append(). Možete li pokušati napisati naredbu koristeći `append()` metodu da dodate 'Inception' na kraj liste?

Korisnikov upit Claude 3.5 Sonnet modelu

```
L.append('Inception')
```

Odgovor Claude 3.5 Sonnet modela

Odlično! To je točno ono što nam treba. Koristili ste `append()` metodu ispravno da biste dodali 'Inception' na kraj liste. Sada prelazimo na četvrti korak našeg zadatka. Trebamo ukloniti drugi film iz liste. Prisjetimo se, drugi film je na indeksu 1. Za uklanjanje elementa iz liste možemo koristiti metodu `pop()` ili ključnu riječ del. Možete li pokušati napisati naredbu koja će ukloniti drugi film iz liste koristeći jednu od ovih metoda?

Korisnikov upit Claude 3.5 Sonnet modelu

```
L.pop(1)
```

Odgovor Claude 3.5 Sonnet modela

Izvrsno! Vaša upotreba metode `pop(1)` je potpuno točna. Ova naredba će ukloniti drugi element iz liste (onaj na indeksu 1) i vratiti ga ako ga želimo spremiti ili ispisati. Sad smo došli do posljednjeg koraka našeg zadatka. Trebamo ispisati konačnu listu nakon svih izmjena koje smo napravili. Ovo bi trebalo biti jednostavno s obzirom na ono što smo već radili. Možete li napisati naredbu za ispis konačne liste?

Korisnikov upit Claude 3.5 Sonnet modelu

```
print(L)
```

Odgovor Claude 3.5 Sonnet modela

Savršeno! Vaša naredba print(L) će ispisati cijelu listu sa svim promjenama koje smo napravili. Sada kada smo završili sve korake, pogledajmo cjelokupno rješenje našeg zadatka:

```
L = ['Forrest Gump','Gladijator','Genijalni um', 'Kum','Kum 2']
print(L)
print(L[2])
L.append('Inception')
L.pop(1)
print(L)
```

Možete li mi objasniti što očekujete da će biti konačni ispis ovog programa? Kako će izgledati završna lista?

Korisnikov upit Claude 3.5 Sonnet modelu

['Forrest Gump', 'Genijalni um', 'Kum','Kum 2', 'Inception']

Odgovor Claude 3.5 Sonnet modela

Vaš odgovor je točan! Odlično ste pratili sve promjene koje smo napravili na listi. Ovo pokazuje da ste dobro razumjeli kako funkcioniraju operacije s listama koje smo koristili: uklanjanje elementa s određenog indeksa i dodavanje novog elementa na kraj liste.

U oba slučaja, modeli su dali početne zadatke za rad s listama. Claude se fokusirao na provjeru razumijevanja indeksiranja unutar liste koristeći listu stringova, dok se ChatGPT odlučio za matematički pristup s listom brojeva. Claudeov zadatak pomaže učenicima da bolje razumiju osnovne principe manipulacije listama, dok ChatGPT-ov zadatak testira učenikovo znanje o ugrađenim funkcijama u Pythonu. Međutim, iako ChatGPT tako provjerava tehničko poznavanje funkcija, ne fokusira se toliko na provjeru dubljeg razumijevanja samog procesa prolaska kroz listu. U školama je često praksa da učenicima ne dopuštaju korištenje ugrađenih metoda već ih potiču da sami osmisle način kako proći kroz listu i pronaći tražene vrijednosti. Ovdje možemo primijetiti da ChatGPT nije postavio zadatak koji bi testirao to ručno prolazak kroz listu, što bi učenicima dalo bolju priliku da vježbaju temeljne koncepte programiranja bez oslanjanja na gotove metode.

3.4 Funkcije

Funkcije su jedan od temeljnih elemenata programiranja, jer omogućuju modularnost i ponovno korištenje koda. Ipak, učenici često nailaze na pogrešna shvaćanja kada počinju raditi s funkcijama. Najčešće miskoncepcije uključuju zbunjenost oko prijenosa argumenata, povratnih vrijednosti, opsega varijabli te svrhe funkcija. Ove pogreške mogu dovesti do nejasnih ili neefikasnih rješenja, što otežava pisanje i razumijevanje koda.

Korisnikov upit GPT-4o modelu

Postavi se u ulogu nastavnika informatike u Hrvatskoj. Predaješ trećem razredu srednje škole. Obradio si nastavnu cjelinu: 'Funkcije'. Ja sam učenik kojem trebaš zadati zadatak iz te nastavne cjeline u Pythonu. Tvoj posao je vođenim učenjem dovesti me do rješenja zadatka.

Odgovor GPT-4o modela

Krenimo s funkcijama! Zadatak će ti pomoći da shvatiš kako organizirati kod i ponovo koristiti dijelove kroz funkcije.
Zadatak: Napiši program u kojem ćeš definirati funkcije za sljedeće operacije:

- Unos brojeva (korisnik unosi 5 brojeva i ti ih vraćaš kao listu).
- Pronalaženje najvećeg broja.
- Pronalaženje najmanjeg broja.
- Računanje prosjeka brojeva.
- Ispis rezultata.

Prvi korak je da definiraš funkciju koja traži unos brojeva od korisnika i vraća listu. Ova funkcija neće imati argumente, već će samo prikupljati brojeve i vraćati listu.

Korisnikov upit GPT-4o modelu

```
def funkcija():
    L = []
    for i in range(5):
        L.append(int(input('Unesi broj:')))
    return L
```

Odgovor GPT-4o modela

Odlično! Definirao si funkciju koja uspješno traži unos brojeva od korisnika i vraća listu. To je dobar početak.

Sada ćemo definirati funkcije za ostale operacije:

- Najveći broj: Definiraj funkciju koja prima listu kao argument i vraća najveći broj.
- Najmanji broj: Slično kao za najveći, definiraj funkciju koja vraća najmanji broj.
- Prosjek: Definiraj funkciju koja prima listu i vraća prosjek brojeva.

Korisnikov upit GPT-4o modelu

```
def najveci(a):
    return max(a)

def najmanji(a):
    return min(a)

def prosjek(a):
    return (sum(a)/len(a))
```

Odgovor GPT-4o modela

Odlično, sve funkcije su točno definirane! Sada imamo funkcije za:

- Pronalaženje najvećeg broja,
- Pronalaženje najmanjeg broja,
- Računanje prosjeka.

Sljedeći korak je da sve ove funkcije upotrijebimo zajedno i kreiramo glavnu funkciju koja će koordinirati pozive. Možemo ih pozvati ovako:

- Pozvati funkciju za unos brojeva i pohraniti vraćenu listu.
- Koristiti tu listu kao argument za funkcije najveci, najmanji i prosjek.
- Ispisati rezultate.

Korisnikov upit GPT-4o modelu

```
def funkcija():
    L = []
    for i in range(5):
        L.append(int(input('Unesi broj:')))
    return L

def najveci(a):
    return max(a)

def najmanji(a):
    return min(a)

def prosjek(a):
    return (sum(a)/len(a))

a = funkcija()

print(najveci(a), najmanji(a), prosjek(a))
```

Odgovor GPT-4o modela

Sjajno, sada tvoj kod radi sve potrebne operacije! Ispisuje najveći broj, najmanji broj i prosjek na temelju unesenih brojeva. Također, lijepo si sve organizirao kroz funkcije.

Isto pitanje postavljeno je i Claudeu 3.5.

Korisnikov upit Claude 3.5 Sonnet modelu

Postavi se u ulogu nastavnika informatike u Hrvatskoj. Predaješ trećem razredu srednje škole. Obradio si nastavnu cjelinu: 'Funkcije'. Ja sam učenik kojem trebaš zadati zadatak iz te nastavne cjeline u Pythonu. Tvoj posao je vođenim učenjem dovesti me do rješenja zadatka.

Odgovor Claude 3.5 Sonnet modela

U redu, postavit ću se u ulogu nastavnika informatike koji predaje drugom razredu srednje škole u Hrvatskoj. Upravo smo obradili nastavnu cjelinu o funkcijama u Pythonu. Kao učeniku, zadat ću vam zadatak vezan uz funkcije i voditi vas kroz proces rješavanja. Zadatak: Napišite funkciju koja prima dva broja kao argumente i vraća njihov najveći zajednički djelitelj (NZD). Zatim napišite glavni program koji će od korisnika tražiti unos dva broja, pozvati funkciju s tim brojevima i ispisati rezultat. Počnimo s definiranjem funkcije. Možete li napisati početak definicije funkcije koja prima dva argumenta? Nazovimo je nzd.

Korisnikov upit Claude 3.5 Sonnet modelu

```
def nzd(x,y):
```

Odgovor Claude 3.5 Sonnet modela

Odlično! Točno ste započeli definiciju funkcije s dva parametra. Sada trebamo implementirati algoritam za pronalaženje najvećeg zajedničkog djelitelja. Jedan od najčešćih načina za to je korištenje Euklidovog algoritma. Evo kako on funkcionira:

- Usporedimo brojeve x i y
- Ako je y = 0, NZD je x
- Inače, izračunamo ostatak dijeljenja x sa y
- Postavimo x na vrijednost y, a y na vrijednost ostatka
- Ponavljamo korake dok y ne postane 0

Možete li pokušati implementirati ovu logiku unutar naše funkcije? Koristite while petlju

Korisnikov upit Claude 3.5 Sonnet modelu

```
def nzd(x,y):
    while y != 0:
        x,y = y, x%y
    print(x)
```

Odgovor Claude 3.5 Sonnet modela

Vaša implementacija je vrlo blizu točnom rješenju! Napravili ste odličan posao u implementaciji Euklidovog algoritma. Međutim, funkcija trenutno ispisuje rezultat umjesto da ga vrati. U Pythonu, funkcije obično vraćaju vrijednost pomoću ključne riječi return.

Evo kako možemo malo modificirati vašu funkciju:

```
def nzd(x, y):
    while y != 0:
        x, y = y, x % y
    return x
```

Ova verzija funkcije će vratiti rezultat umjesto da ga ispiše.

Sada, možete li napisati glavni dio programa koji će:

- Tražiti od korisnika da unese dva broja
- Pozvati funkciju nzd s tim brojevima
- Ispisati rezultat

Pokušajte to implementirati!

Korisnikov upit Claude 3.5 Sonnet modelu

```
def nzd(x,y):
    while y != 0:
        x,y = y, x%y
    return (x)

x = int(input())
y = int(input())
print(nzd(x, y))
```

Odgovor Claude 3.5 Sonnet modela

Izvrsno! Vaše rješenje je točno i cjelovito. Analizirajmo što ste napravili:

Funkcija nzd je ispravno implementirana. Ona koristi Euklidov algoritam za pronalaženje najvećeg zajedničkog djelitelja i vraća rezultat koristeći return.

Koristite `input()` za unos brojeva od korisnika.

Pretvarate unesene vrijednosti u cijele brojeve pomoću `int()`.

Pozivate funkciju nzd s unesenim brojevima.

Ispisujete rezultat funkcije.

Vaš kod će raditi kako je zamišljeno.

Oba modela pružila su jasne i precizne upute za vođeno učenje, prilagođene razini znanja učenika. Zadatci su bili na odgovarajućoj razini složenosti, omogućujući učenicima da uvježbaju temeljne koncepte funkcija u programiranju. Iako su oba modela ponudila kvalitetne zadatke, Claudeov zadatak ponovno je bio nešto zahtjevniji. ChatGPT je ponudio zadatak s nešto jednostavnijim pristupom, omogućivši učenicima lakše utvrđivanje gradiva. Zaključno, oba modela pružila su učinkovitu potporu u učenju funkcija, s različitim pristupima koji se mogu prilagoditi različitim razinama učenika.

3.5 Klase i objekti

Uvođenje klasa i objekata predstavlja značajan korak u razumijevanju objektno orijentiranog programiranja, omogućujući organizaciju koda i modeliranje stvarnih entiteta. Međutim, učenici često susreću izazove u ovom području zbog nesporazuma oko osnovnih pojmova kao što su nasljeđivanje, enkapsulacija i interakcija između objekata. Česte miskoncepcije uključuju zbunjenost oko razlike između klasa i objekata, nepravilno korištenje konstruktornih metoda i nejasnoće u vezi s pristupom varijablama i metodama.

Korisnikov upit GPT-4o modelu

Postavi se u ulogu nastavnika informatike u Hrvatskoj. Predaješ četvrtom razredu srednje škole. Obradio si nastavnu cjelinu: 'Objekti i klase objekata'. Ja sam učenik kojem trebaš zadati zadatak iz te nastavne cjeline u Pythonu. Tvoj posao je vođenim učenjem dovesti me do rješenja zadatka.

Odgovor GPT-4o modela

Napiši Python program koji definira klasu "Pravokutnik". Klasa treba imati:
Atribute: dužinu i širinu.
Metode: izračunavanje opsega i izračunavanje površine.
Prvi korak: Definiraj klasu Pravokutnik s konstruktorom init, koji prima dužinu i širinu kao argumente i postavlja te vrijednosti na atribute objekta. Probaj ovo napisati, pa idemo dalje!

Korisnikov upit GPT-4o modelu

```
class Pravokutnik:
    def __init__ (self, a=1, b=1):
        self.a = a
        self.b = b
```

Odgovor GPT-4o modela

Upravo si stvorio osnovnu klasu Pravokutnik s konstruktorom koji inicijalizira dužinu (a) i širinu (b). To je odličan prvi korak.

Sada ćemo dodati dvije metode u klasu:

- Metoda za izračunavanje opsega: Koristimo formulu 2 * (a + b).
- Metoda za izračunavanje površine: Koristimo formulu a * b.

Dodaj ove metode unutar klase:

```
class Pravokutnik:
    def __init__(self, a=1, b=1):
        self.a = a
        self.b = b

    def opseg(self):
        return 2 * (self.a + self.b)

    def povrsina(self):
        return self.a * self.b
```

Probaj sada koristiti ove metode za računanje opsega i površine pravokutnika. Stvori objekt klase Pravokutnik i pozovi metode na tom objektu. Ako zapneš, tu sam da pomognem!

Korisnikov upit GPT-4o modelu

```
class Pravokutnik:
    def __init__ (self, a=1, b=1):
        self.a = a
        self.b = b
    def opseg(self):
        return 2*(self.a+self.b)
    def povrsina(self):
        return self.a*self.b

a = Pravokutnik(5,10)

print(a.opseg(), a.povrsina())
```

Odgovor GPT-4o modela

Odlično! Tvoj kod savršeno radi. Kreirao si objekt klase Pravokutnik s dimenzijama 5 i 10, i ispravno koristiš metode za izračunavanje opsega i površine. Rezultat ispisa će biti:

Opseg: 2 * (5 + 10) = 30

Površina: 5 * 10 = 50

Isto pitanje postavljeno je i Claude 3.5.

Korisnikov upit Claude 3.5 Sonnet modelu

Postavi se u ulogu nastavnika informatike u Hrvatskoj. Predaješ četvrtom razredu srednje škole. Obradio si nastavnu cjelinu: 'Objekti i klase objekata'. Ja sam učenik kojem trebaš zadati zadatak iz te nastavne cjeline u Pythonu. Tvoj posao je vođenim učenjem dovesti me do rješenja zadatka.

Odgovor Claude 3.5 Sonnet modela

Razumijem, postavit ću se u ulogu nastavnika informatike koji predaje četvrtom razredu srednje škole u Hrvatskoj. Obradili smo nastavnu cjelinu o objektima i klasama u Pythonu. Kao učeniku, zadat ću vam zadatak koji će testirati vaše razumijevanje ovih koncepata. Zadatak: Napišite program koji implementira sustav za upravljanje knjižnicom koristeći objekte i klase. Trebate stvoriti klasu Knjiga i klasu Knjiznica. Počnimo s klasom Knjiga. Ova klasa treba imati sljedeće atribute:

naslov

autor

godina izdanja

dostupnost (`True` ako je knjiga dostupna, `False` ako je posuđena)

Također, klasa Knjiga treba imati metode za:

posudbu knjige

vraćanje knjige

Možete li pokušati definirati klasu Knjiga s ovim atributima i metodama?

Korisnikov upit Claude 3.5 Sonnet modelu

```
class knjiga:
    def init(self, naslov='', autor='', godina_izdanja=1,
        dostupnost = True):
        self.naslov = naslov
        self.autor = autor
        self.godina = godina_izdanja
        self.dostupnost = dostupnost
    def posudba():
        if(self.dostupnost):
            dostupnost = False
            print('Knjiga je posudena.')
        else:
            print('Knjiga nije dostupna.')
    def vracanje():
        dostupnost = True
        print('Knjiga je posudena.')
```

Odgovor Claude 3.5 Sonnet modela

Vaš pristup je vrlo dobar! Napravili ste odličan posao u definiranju klase Knjiga. Međutim, postoji nekoliko malih detalja koje možemo popraviti kako bi kod bio potpuno ispravan:

Ime klase bi trebalo početi velikim slovom prema Python konvencijama, dakle Knjiga umjesto knjiga.

U inicijalizatoru init, koristite dvostruke donje crte umjesto jednostrukih.

U metodama posudba i vracanje, trebate dodati self kao prvi parametar.

U metodi posudba, trebate koristiti self.dostupnost umjesto samo dostupnost.

U metodi vracanje, poruka bi trebala biti "Knjiga je vraćena" umjesto "Knjiga je posuđena".

Evo ispravljene verzije vaše klase:

```
class Knjiga:
    def __init__(self, naslov='', autor='', godina_izdanja=1,
        dostupnost=True):
        self.naslov = naslov
        self.autor = autor
        self.godina = godina_izdanja
        self.dostupnost = dostupnost
    def posudba(self):
        if self.dostupnost:
            self.dostupnost = False
            print('Knjiga je posudena.')
        else:
            print('Knjiga nije dostupna.')
    def vracanje(self):
        self.dostupnost = True
        print('Knjiga je vracena.')
```

Sada kada smo definirali klasu Knjiga, možemo prijeći na sljedeći korak. Možete li pokušati definirati klasu Knjiznica koja će imati:

Atribut knjige (lista koja sadrži objekte klase Knjiga)

Metodu dodaj knjigu koja dodaje novu knjigu u knjižnicu

Metodu prikazi knjige koja prikazuje sve knjige u knjižnici

Pokušajte implementirati ovu klasu.

Korisnikov upit Claude 3.5 Sonnet modelu

```
class Knjiznica:
    def init (self):
        self.L = []

    def dodaj_knjigu(self, Knjiga):
        self.L.append(Knjiga)

    def prikazi_knjige(self):
        for i in self.L:
            print(i.naslov, end= ' ')
```

Odgovor Claude 3.5 Sonnet modela

Izvrsno! Vaša implementacija klase Knjiznica je gotovo savršena. Napravili ste nekoliko vrlo dobrih odluka:

Pravilno ste inicijalizirali praznu listu u init metodi.

Metoda dodaj knjigu ispravno dodaje novu knjigu u listu.

Metoda prikazi knjige prolazi kroz sve knjige i ispisuje njihove naslove.

Sada kada imamo obje klase, možemo napisati glavni dio programa koji će koristiti ove klase za stvaranje knjižnice, dodavanje knjiga i interakciju s korisnikom. Možete li napisati glavni dio programa koji će:

Stvoriti objekt klase Knjiznica

Dodati nekoliko knjiga u knjižnicu

Prikazati sve knjige u knjižnici

Omogućiti korisniku da posudi ili vrati knjigu (po naslovu)

Pokušajte implementirati ovaj dio programa.

Korisnikov upit Claude 3.5 Sonnet modelu

```
def init(self, naslov='', autor='', godina_izdanja=1,
    dostupnost = True):
        self.naslov = naslov
        self.autor = autor
        self.godina = godina_izdanja
        self.dostupnost = dostupnost
    def posudba(self):
        if(self.dostupnost):
            self.dostupnost = False
            print('Knjiga je posudena.')
        else:
            print('Knjiga nije dostupna.')
    def vracanje(self):
        self.dostupnost = True
        print('Knjiga je vracena.')

class Knjiznica:
    def init (self):
        self.L = []
    def dodaj_knjigu(self, x,y,z,k):
        self.L.append(Knjiga(x,y,z,k))
    def prikazi_knjige(self):
        for i in self.L:
            print(i.naslov, end= ' ')
    def posudba(self, ime):
        for i in self.L:
            if i.naslov == ime:
                i.posudba()

knjiznica = Knjiznica()
knjiznica.dodaj_knjigu('Zlocin i kazna','Fjodor Dostojevski'
    ,1866,True)
knjiznica.dodaj_knjigu('Proces','Franz Kafka',1925,True)
knjiznica.dodaj_knjigu('Rat i mir','Lav Tolstoj',1965,True)
knjiznica.prikazi_knjige()
knjiznica.posudba('Zlocin i kazna')
```

Odgovor Claude 3.5 Sonnet modela

Odlično! Vaš kod pokazuje dobro razumijevanje koncepta klasa i objekata. Napravili ste nekoliko pametnih odluka i implementirali osnovnu funkcionalnost knjižnice.

Oba modela ponudila su slične tipove zadataka, no Claudeov zadatak zahtijevao je više vremena za rješavanje. Claude je učenika vodio kroz proces stvaranja rješenja, pružajući priliku da sami napišu potrebne funkcije. S druge strane, ChatGPT je odmah dao gotove funkcije bez vođenja učenika kroz pisanje. Ovaj pristup može biti koristan za brzu demonstraciju rješenja, no učenicima može oduzeti priliku za samostalno rješavanje problema i bolje razumijevanje gradiva. Claudeov pristup, iako zahtjevniji, potiče dublje razumijevanje koncepta objekata i klasa i implementaciju objekata u listu. Dok je ChatGPT-ov brži pristup prikladniji za brze primjere ili kada je učeniku potrebno brže rješenje za uvid u cjelokupnu strukturu koda.

3.6 Analiza sposobnosti jezičnih modela u vođenju učenika kroz zadani nastavni zadatak

Do sada smo analizirali sposobnost modela da za određenu temu formuliraju zadatak i vode korisnika kroz njegovo rješavanje. U ovom odjeljku istražujemo kako se modeli ponašaju kada im se prezentira zadatak koji je nastavnik već formulirao za učenike. Cilj je utvrditi mogu li modeli jednako učinkovito voditi učenike kroz rješavanje prethodno definiranih zadataka, kao što to čine s vlastitim formulacijama. Značaj ovog istraživanja leži u procjeni sposobnosti modela da razumiju i pomognu u rješavanju zadataka postavljenih od strane nastavnika, što je ključno za njihovu potencijalnu primjenu u obrazovanju. Analizom načina na koji modeli vode učenike kroz rješavanje zadataka možemo dobiti uvid u njihove prednosti i ograničenja u usporedbi s ljudskim tutorima.

Korisnikov upit GPT-4o modelu

Nastavnik je zadao zadatak: Napiši program koji simulira igricu pogađanja broja. Pogađaš nasumično izabrani broj između 1 i 100. Za svaki pokušaj program treba ispisati je li tvoj broj manji ili veći. Igra se nastavlja dok korisnik ne pogodi broj ili dok ne odluči prekinuti igru, upisivanjem broja 0. Postavi se u ulogu asistenta u nastavi te mi vođenim učenje pomozi doći do konačnog rješenja.

Odgovor GPT-4o modela

Razumijem, hajdemo vođenim učenjem doći do konačnog rješenja, korak po korak. Zadatak je jasan, pa ćemo ga postepeno rješavati.

Korak 1: Generiranje nasumičnog broja.

Trebat ćemo koristiti random modul da nasumično generiramo broj između 1 i 100. Ovo ćemo postaviti na početku programa.

Zadatak za tebe: Napiši kod koji nasumično generira broj i spremi ga u varijablu. U ovom trenutku korisnik još ne pogađa broj.

Kad napišeš ovaj kod, javi, pa ćemo ići na sljedeći korak.

Korisnikov upit GPT-4o modelu

```
import random
x = random.randint(1,100)
```

Odgovor GPT-4o modela

Korak 2: Unos korisnika i kontrola toka Sada ćemo omogućiti korisniku da pogađa broj. Trebamo:
Napraviti beskonačnu petlju (dok korisnik ne pogodi broj ili ne unese 0). Unos korisnika treba biti provjeren tako da program javi je li uneseni broj veći ili manji od nasumično generiranog broja. Tvoj sljedeći zadatak:

- Napiši petlju koja će se ponavljati sve dok korisnik ne unese 0 ili dok ne pogodi broj.
- Provjeri je li broj koji je korisnik unio veći ili manji od nasumično generiranog broja, i ispisi odgovarajuću poruku.

Pokušaj ovo, pa ćemo dalje prilagoditi kod.

Korisnikov upit GPT-4o modelu

```
import random
x = random.randint(1, 100)
y = -1
while x != y:
    y = int(input())
    if y > x:
        print('Probaj manji broj.')
    elif y < x:
        print('Probaj veci broj')
    elif y==0:
        print('Odustali ste od igre.')
        break
    else:
        print('Pogodili ste broj')
```

Odgovor GPT-4o modela

Kod je skoro ispravan, ali postoji nekoliko stvari koje treba ispraviti:

- Redoslijed provjere: Trebalo bi prvo provjeriti je li korisnik unio 0 (da se može odmah prekinuti), a zatim provjeriti je li broj manji ili veći.
- Logika za "pogodio si broj": Ova provjera treba biti postavljena nakon što provjerimo je li unos 0 i nakon provjere da li je unos veći ili manji od tajnog broja.
- Inicijalizacija varijable: Varijabla y se koristi za unos korisnika, pa se njeno postavljanje unutar petlje mora provesti odmah po svakom unosu.

Evo ispravljenog koda:

```
import random
x = random.randint(1, 100)
while True:
    y = int(input("Unesi broj izmedu 1 i 100 (ili 0 za prekid):
        "))

    if y == 0:  # Ako korisnik unese 0, izlazimo iz igre
        print("Odustali ste od igre.")
        break
    elif y > x:  # Ako je unos veci od nasumicno odabranog
       broja
        print('Probaj manji broj.')
    elif y < x:  # Ako je unos manji od nasumicno odabranog
       broja
        print('Probaj veci broj.')
    else:  # Ako korisnik pogodi broj
        print('Pogodili ste broj!')
        break  # Prekida petlju nakon pogodka
```

Isto pitanje postavljeno je i Claude 3.5.

Korisnikov upit Claude 3.5 Sonnet modelu

Nastavnik je zadao zadatak: Napiši program koji simulira igricu pogađanja broja. Pogađaš nasumično izabrani broj između 1 i 100. Za svaki pokušaj program treba ispisati je li tvoj broj manji ili veći. Igra se nastavlja dok korisnik ne pogodi broj ili dok ne odluči prekinuti igru, upisivanjem broja 0. Postavi se u ulogu asistenta u nastavi te mi vođenim učenje pomozi doći do konačnog rješenja.

Odgovor Claude 3.5 Sonnet modela

Razumijem, postavit ću se u ulogu asistenta u nastavi i voditi vas kroz proces razvoja ovog programa. Počnimo s osnovama i postupno ćemo graditi program.

- Prvo, razmislimo o strukturi programa. Koje korake mislite da bi naš program trebao imati?
- Za generiranje nasumičnog broja, koji Python modul bismo trebali koristiti? Pokušajte se sjetiti ili pogledati u svoje bilješke.
- Nakon što uvezemo odgovarajući modul, kako bismo mogli generirati nasumični broj između 1 i 100?

Pokušajte napisati prve linije koda koje će uvesti potreban modul i generirati nasumični broj. Kada završite, pokažite mi što ste napisali, i zajedno ćemo pregledati i nastaviti dalje.

Korisnikov upit Claude 3.5 Sonnet modelu

```
import random
a = random.randint(1,100)
```

Odgovor Claude 3.5 Sonnet modela

Odlično! To je izvrsno za početak. Pravilno ste uvezli modul random i koristili funkciju randint() za generiranje nasumičnog broja između 1 i 100. Samo bih predložio da umjesto a koristimo opisnije ime varijable, recimo tajni broj. To će učiniti kod čitljivijim.

Dakle, imamo:

```
import random
tajni_broj = random.randint(1, 100)
```

Sada, razmislimo o sljedećim koracima:

- Kako bismo mogli ispisati poruku dobrodošlice i objasniti pravila igre korisniku?
- Trebamo omogućiti korisniku da više puta pogađa. Koju vrstu petlje bismo mogli koristiti za to?
- Unutar te petlje, kako bismo mogli tražiti unos od korisnika?

Pokušajte napisati kod za ove korake. Ne brinite ako nije savršeno, zajedno ćemo ga pregledati i poboljšati.

Korisnikov upit Claude 3.5 Sonnet modelu

```
print("Dobrodosli u igricu pogadanja broja!")
print("Pogodite broj izmedu 1 i 100. Za prekid igre unesi 0.")

y = -1
while y != tajni_broj:
    y = int(input())
```

Odgovor Claude 3.5 Sonnet modela

Izvrsno! Napravili ste dobar napredak. Vaše ispise dobrodošlice i uputa su odlični. Petlja je dobra ideja, ali imamo nekoliko stvari koje trebamo popraviti:

- Trebamo dodati logiku za provjeru je li korisnik pogodio broj ili želi prekinuti igru.
- Bilo bi dobro dodati povratne informacije korisniku nakon svakog pokušaja.
- mogli bi preimenovati y u nešto opisnije, poput pokusaj.
- Kako bismo mogli provjeriti je li korisnik unio 0 za prekid igre?
- Kako bismo mogli usporediti pokusaj s tajni broj i dati odgovarajuću povratnu informaciju?

Korisnikov upit Claude 3.5 Sonnet modelu

```
import random
tajni_broj = random.randint(1, 100)
print("Dobrodosli u igricu pogadanja broja!")
print("Pogodite broj izmedu 1 i 100. Za prekid igre unesi 0.")
pokusaj = -1
while pokusaj != tajni_broj:
    pokusaj = int(input())
    if pokusaj > tajni_broj:
        print('Probaj manji broj.')
    elif pokusaj < tajni_broj:
        print('Probaj veci broj')
    elif y==0:
        print('Odustali ste od igre.')
        break
    else:
        print('Pogodili ste broj')
```

Odgovor Claude 3.5 Sonnet modela

Izvrsno! Napravili ste značajan napredak. Vaš kod je vrlo blizu konačnom rješenju. Hajdemo ga malo doraditi:

- Odlično što ste dodali break za prekid igre kad korisnik unese 0.
- Dobro je što ste dodali else blok, koji će se izvršiti kad korisnik pogodi broj.
- Premjestimo provjeru za 0 na početak petlje.

Pokušajte implementirati ove promjene.

Korisnikov upit Claude 3.5 Sonnet modelu

```
import random
tajni_broj = random.randint(1, 100)
print("Dobrodosli u igricu pogadanja broja!")
print("Pogodite broj izmedu 1 i 100. Za prekid igre unesi 0.")
pokusaj = -1
while pokusaj != tajni_broj:
    pokusaj = int(input())
    if pokusaj == 0:
        print('Odustali ste od igre.')
        break
    if pokusaj > tajni_broj:
        print('Probaj manji broj.')
    elif pokusaj < tajni_broj:
        print('Probaj veci broj')
    else:
        print('Pogodili ste broj')
```

Odgovor Claude 3.5 Sonnet modela

Odlično! Vaš kod je sada potpun i funkcionalan. Napravili ste sjajan posao u implementaciji igre pogađanja broja. Hajdemo zajedno pregledati vaše rješenje:

- Pravilno ste uvezli random modul i generirali tajni broj.
- Poruke dobrodošlice i upute su jasne.
- Struktura petlje je dobra, s uvjetom pokusaj != tajni broj.
- Ispravno ste implementirali provjeru za prekid igre (unos 0).
- Dajete odgovarajuće povratne informacije za svaki pokušaj (veći, manji).
- Igra se pravilno završava kada korisnik pogodi broj.

Jedina manja sugestija koju bih dao je da razmislite o dodavanju broja pokušaja.

Oba modela uspješno su započela vođenje učenika kroz zadatak, precizno zadajući što učenik treba napraviti. Dok je Claude učenika vodio kroz greške, pružajući priliku da sami isprave potrebne funkcije, ChatGPT je odmah dao ispravljen kod bez vođenja učenika kroz pisanje. Ovaj pristup može biti koristan za brzu demonstraciju rješenja, no učenicima može oduzeti priliku za samostalno rješavanje problema i bolje razumijevanje gradiva. Iako su oba modela rekla kako uvjet za izlaz iz programa treba biti na početku petlje, niti jedan nije pokušao objasniti zašto je to tako. Ovo je propuštena prilika za dodatno obrazloženje logike programa. Objašnjenje bi moglo uključivati činjenicu da provjera uvjeta za izlaz na početku petlje osigurava da se nepotrebne operacije ne izvršavaju ako korisnik želi prekinuti igru. Claude nije komentirao kako nije dobro postaviti inicijalnu vrijednost varijable van petlje u slučaju da se promijeni interval brojeva koje pogađamo, što može dovesti do problema ako se broj pojavi u tom intervalu. S druge strane, ChatGPT je predvidio ovaj problem i prilagodio kod postavljanjem True vrijednosti u uvjet while petlje, iako nije objasnio učeniku zašto je to napravio. Dodatno, moglo bi se spomenuti kako bi bilo korisno da je ChatGPT objasnio važnost korištenja opisnih imena varijabli (npr. tajni broj umjesto x) za bolju čitljivost koda, kao što je to napravio Claude. Ovaj primjer je

dobar pokazatelj da, iako su multimodalni jezični modeli snažan alat za rad, i dalje trebamo nastavnikovo iskustvo u vođenju učenika. Nastavnik može pružiti kontekstualno relevantna objašnjenja, potaknuti kritičko razmišljanje i osigurati da učenici dubinski razumiju koncepte, a ne samo mehanički reproduciraju kod.

4 Zaključak

Uvođenje multimodalnih jezičnih modela kao asistenta u nastavi informatike nudi značajne prednosti, ali donosi i određene izazove. Ovaj rad analizirao je različite aspekte primjene AI u obrazovnom procesu, ističući kako ova tehnologija može unaprijediti obrazovanje, istovremeno ukazujući na potencijalne nedostatke. Jedna od ključnih prednosti korištenja multimodalnih jezičnih modela u nastavi informatike je mogućnost personaliziranog učenja. Takvi modeli mogu prilagoditi tempo i sadržaj lekcija svakom učeniku pojedinačno, čime se omogućuje učinkovitije učenje koje odgovara specifičnim potrebama svakog učenika. Ova tehnologija omogućava učenicima da dobiju pomoć u bilo kojem trenutku, dok brza povratna informacija na zadatke i kodove poboljšava kvalitetu učenja kroz trenutnu evaluaciju. Precizno praćenje napretka učenika omogućava identifikaciju područja koja zahtijevaju dodatnu pažnju, a rasterećenje nastavnika od rutinskih zadataka omogućava više vremena za kreativni rad i individualni pristup. Ažuriranje znanja i prilagodba različitim stilovima učenja dodatno doprinose kvaliteti obrazovnog procesa. Međutim, korištenje multimodalnih jezičnih modela u obrazovanju također donosi određene izazove. Oni uključuju smanjenje ljudskog kontakta te ograničavanje nastavnikovog nadzora i praćenja napretka učenika, što može umanjiti važne socijalne interakcije u učionici. Unatoč tome, važno je naglasiti da ovi modeli mogu istovremeno služiti kao alat za osobni rast i profesionalni razvoj nastavnika, omogućujući im pristup novim resursima i metodama podučavanja. Dok modeli, iako korisni, nisu potpuno nepogrešivi i mogu pružati netočne informacije. Ovisnost o tehnologiji može ometati razvoj samostalnog kritičkog mišljenja kod učenika, dok ograničeno razumijevanje konteksta može predstavljati dodatne prepreke u komunikaciji. Važno je napomenuti da optimalno korištenje multimodalnih jezičnih modela u nastavi informatike zahtijeva pažljivo balansiranje između tehnoloških mogućnosti i pedagoških potreba, uz kontinuiranu evaluaciju i prilagodbu pristupa.

Literatura

[1] Prince, S. J. D. Understanding Deep Learning. Massachusetts : The MIT Press, 2023.

[2] Stipaničev, D.; Šerić, Lj.; Braović, M. Uvod u umjetnu inteligenciju. Split : Manualia universitatis studiorum Spalatiensis, 2021.

[3] Čupić, M. Umjetne neuronske mreže, self-published, 2016, http://java.zemris.fer.hr/nastava/ui/ann/ann-20180604.pdf

[4] Jiang, B.; Wu, T.; Zheng, C.; Wong, W. H. Approximate Bayesian computation via deep neural network https://www.jstor.org/stable/26384090

[5] Alfieri, L.; Brooks, P. J.; Aldrich, N. J.; Tenenbaum, H. R. Does discovery-based instruction enhance learning?, City University of New York, 2011.

[6] Antonio Jovanović (Diplomski rad, 2021). Primjena rekurentnih neuronskih mreža u analizi programskog koda. Matematički odsjek, Sveučilište J. J. Strossmayera u Osjeku

[7] Johnson, Justin. Lecture 3: Linear Classifiers, University of Michigan, pristupljeno 19. kolovoza 2024. https://youtu.be/qcSEP17uKKY?si=YxKX2aMqySCv2YNW

[8] Johnson, Justin. Lecture 6: Beckpropagation, University of Michigan, pristupljeno 28. kolovoza 2024. https://youtu.be/dB-u77Y5a6A?si=GRvrJkS-FtROisdh

[9] Ng, Andrew. Lecture 2: Machine Learning - Linear Regression and Gradient Descent,Stanford, pristupljeno 28. kolovoza 2024. https://youtu.be/4b4MUYve_U8?si=9TahIXMJnTO2gfyl

[10] Žanko, Žana: Miskoncepcije u uvodnoj nastavi programiranja, pristupljeno 8.rujna 2024. https://www.pmfst.unist.hr/odjel-za-matematiku/wpcontent/uploads/2018/09/zana_zanko_seminar1.pdf

[11] Heidloff, Niklas: Foundation Models, Transformers, BERT and GPT, pristupljeno 12.rujna 2024. https://heidloff.net/article/foundation-models-transformers-bert-and-gpt/

[12] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. In Advances in Neural Information Processing Systems, NIPS 2017

[13] Sanderson, Grant. But what is a GPT? Visual intro to transformers — Chapter 5, Deep Learning, pristupljeno 13.rujna 2024. https://youtu.be/wjZofJX0v4M?si=8NtKA83behJjHKzu

[14] Sanderson, Grant. Attention in transformers, visually explained — Chapter 6, Deep Learning, pristupljeno 13.rujna 2024. https://youtu.be/eMlx5fFNoYc?si=Hmo1XGv-KcVnhkhC

[15] Sanderson, Grant. How might LLMs store facts — Chapter 7, Deep Learning, pristupljeno 13.rujna 2024. https://youtu.be/9-Jl0dxWQs8?si=FEyLlhjcH_HAm0yp

[16] Attention (machine learning), pristupljeno 16.rujna 2024. https://en.wikipedia.org/wiki/Attention_(machine_learning)

[17] Understanding Positional Encoding in Transformers, pristupljeno 16. rujna 2024. https://erdem.pl/2021/05/understanding-positional-encoding-in-transformers