Chatbot

Gajić, Borna

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni preddiplomski studij matematike i računarstva

Borna Gajić

Chatbot

Završni rad

Osijek, 2021.

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni preddiplomski studij matematike i računarstva

Borna Gajić

Chatbot

Završni rad

Mentor: doc. dr. sc. Domagoj Ševerdija

Osijek, 2021.

Sažetak

U ovome radu upoznati ćemo se s pojmom chatbota, povijesti chatbota, programskog okvira u kojem je moguće napraviti vlastitog digitalnog asistenta i glavne arhitekture Rasa programskog okvira koja omogućuje klasifikaciju namjera i entiteta u rečenici. Osim toga, u radu su objašnjene tehnike strojnog učenja te ćemo se upoznati s neuronskim mrežama. Na kraju je predložen jednostavan način implementacije vlastitog digitalnog asistenta.

Ključne riječi

Chatbot, povijest chatbota, strojno učenje, neuronske mreže, mehanizam samopažnje, DIET, Rasa, vektori riječi

Abstract

In this paper, we will be introduced to the concept of a chatbot, the history of a chatbot, a programming framework in which it is possible to create your own digital assistant, and the main architecture of the Rasa programming framework that allows classification of intentions and entities in a sentence. In addition, the paper explains machine learning techniques and we will get acquainted with neural networks. Finally, we will show a simple way to implement your own digital assistant.

Key words

Chatbot, chatbot history, machine learning, neural networks, self-attention mechanism, DIET, Rasa, word vectors

Sadržaj

1 Uvod

Chatbot je oblik umjetne inteligencije čija je zadaća komunikacija s ljudima putem mreže kroz sustav za slanje poruka (eng. *chat*). Pošto su dizajnirani da uvjerljivo simuliraju čovjekov način komunikacije, koriste se u korisničkim uslugama, tehničkoj podršci, bolnicama, bankama, restoranima to jest može se koristiti u svakoj grani koja jednim dijelom može zamijeniti pravu osobu ili olakšati odnosno ubrzati određeni zadatak. Dijelimo ih na deklarativne i konverzacijske. Jedna od mogućih zadaća deklarativnog chatbota je odgovaranje na često postavljena pitanja, dok je konverzacijski zaslužan za složeniju vrstu komunikacije u smislu posjedovanja konteksta unutar razgovora, personaliziranih poruka, reagiranje na trenutno stanje korisnika i slično. Primjerice, trgovine koje se bave prodajom tekstilne odjeće obično bi odabrale deklarativnog chatbota koji bi odgovarao na pitanja o povratu robe, zamijeni umjesto povrata, posjeduje li trgovina katalog i slično, dok bi banke uzele konverzacijskog chatbota koji bi mogao prikazati stanje računa, obavljati slanje novca na račun te plaćanje računa.

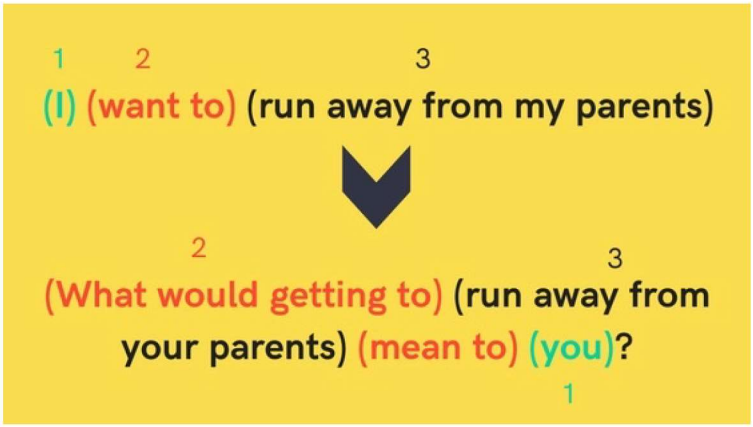
Bio deklarativni ili konverzacijski chatbot, oba koriste tehnike obrade prirodnog jezika koje razbijaju tekst na dijelove koje onda računalo može koristiti za shvaćanje napisanoga. Tehnike se dijele na analiziranje sintakse, parsiranje teksta i semantike. Neke tehnike pod sintaksom su učenje formalne gramatike tj. generiranje formalne gramatike koja opisuje sintaksu danog jezika, lematizaciju tj. svođenje riječi na kanonski oblik koji je najčešće infinitiv za glagole i jednina za imenice te označavanje vrsta riječi u rečenici; u parsiranju teksta koristimo razdvajanje rečenica, segmentaciju i korjenovanje riječi, a pod semantiku spadaju leksička semantika, razumijevanje prirodnog jezika itd. Osim tehnika obrade prirodnog teksta koriste se i tehnike strojnog učenja koji su nužni korak za rad digitalnog asistenta.

1.1 Povijest

Sve pametniji chatboti su dolazili s razvojem strojnog učenja i tehnika obrade prirodnog jezika. Trenutno vodstvo imaju Googleov Google Assistant, Appleova Cortana i Amazonova Alexa. No, pogledajmo neke povijesne digitalne asistente koji su otvorili novo poglavlje u matematici i računarstvu.

1.1.1 ELIZA

Prvog chatbota napravio je MIT profesor Joseph Weizenbaum 1964. godine u svrhu pokazivanja da je komunikacija između računala i čovjeka umjetna. Chatbota je nazvao ELIZA i dao ulogu psihologa. Posao kojeg je radio temeljio se na jednostavnim Rogersovim pravilima za psihoterapiju, koja je profesor Weizenbaum simulirao u ELIZA-inoj skripti. Odlučio je o poretku važnosti vrsta riječi unutar rečenice; počevši od najmanje: zamjenice, glagoli te ostatak rečenice koji u sebi nosi namjeru iste. Odgovor se postiže promjenom poretka riječi u rečenici, te zamjenom označenih riječi s drugima u svrhu postizanja odgovora. ELIZA je bio prvi chatbot koji je prošao Turingov test. (vidi [1])



Slika 1: Ulazna rečenica korisnika i izlazna rečenica ELIZA-e

Konkretno, promatramo ulaznu rečenicu korisnika koja glasi

Želim pobjeći od svojih roditelja

ELIZA uzima riječ "(ja) želim" i ostatak rečenice, zamjenjuje im redoslijed riječi, mijenja označene riječi i kao rezultat dobiva izlazno pitanje:

Što bi za tebe značilo bježanje od roditelja?

Možemo primijetiti da se "(ja) želim" pretvorilo u "što bi za tebe značilo" a ostatak je sličan kao i ostatak ulazne rečenice.

1.1.2 PARRY

Nakon Weizenbaumovog chatbota je 1972. psiholog Kenneth Colby napravio PARRY, chatbota koji glumi pacijente sa šizofrenijom, čija je svrha bila simuliranje bolesti. Program je temeljen na tehnikama obrade prirodnog jezika koja je omogućavala umjetni način razmišljanja takve osobnosti. Konkretno, emocionalne reakcije su se događale u ovisnosti o težinama postavljenim na određenim riječima koje bi korisnik koristio. Kao i ELIZA, PARRY je također prošao Turingov test.

1.1.3 Jabberwacky

1988. Godine je nastao Jabberwacky, chatbot programiran od strane Rollo Carpentera, čija je svrha bila simuliranje prirodnog ljudskog razgovora. Prvi je chatbot koji je učio kako razgovarati s ljudima, razgovaranjem s ljudima; tj. sustav sprema svako pitanje i odgovor koje korisnik da, a zauzvrat Jabberwacky pronađe najbolji odgovor pomoću tehnike podudaranja kontekstualnog uzorka. Zbog drugačijeg ponašanja čovjeka u slučaju pričanja s onime s čime nema fizičkog kontakta, Jabberwacky se u nekim trenucima ne adekvatno izražava. Osim toga, problemi postoje i kada ljudi ignoriraju što Jabberwacky govori, tj. nastavljaju pričati ono što su krenuli ne obaziruć se na odgovore chatbota, takav način komunikacije vodi do lošijih odgovora drugim ljudima. Nakon Jabberwackyja nastali su još mnogi slični chatboti koji su se međusobno natjecali za Loebnerovu nagradu tj. koji će od njih imati najuvjerljiviji prirodni razgovor s ljudima pa i s tim dobiti najveću ocjenu na Turingovom testu. (vidi [2])

2 Strojno učenje u izradi digitalnog asistenta

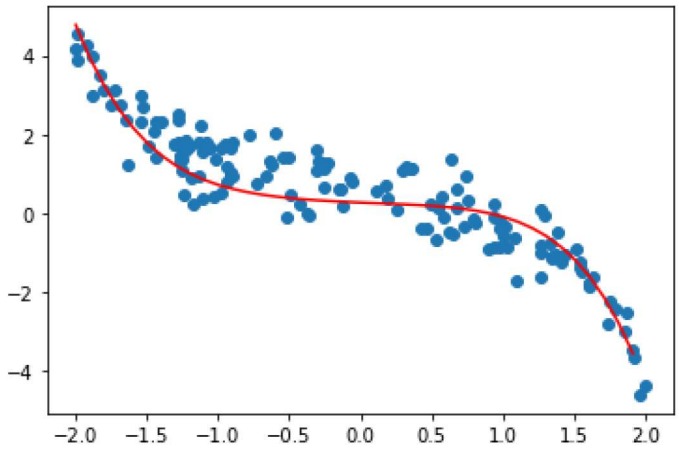
2.1 Tehnologija

Možemo naslutiti da je proces stvaranja takve umjetne inteligencije izrazito kompliciran, stoga postoje programski okviri koji daju mogućnost kreiranja vlastitog deklarativnog ili konverzacijskog chatbota u nekom programskom jeziku. Za izgradnju vlastitog digitalnog asistenta koristili smo Rasa programski okvir u Python programskom jeziku kojeg ću u nastavku rada obraditi. Digitalni asistent kojeg smo napravili može se preuzeti na GitHub repozitoriju [11]. Prije nego što započnemo s objašnjenjem rada arhitekture Rasa programskog okvira, moramo uvesti i pojasniti pojmove strojnog učenja, arhitekture, modela i ostalih pojmova vezanih uz obradu rečenica.

2.2 Strojno učenje

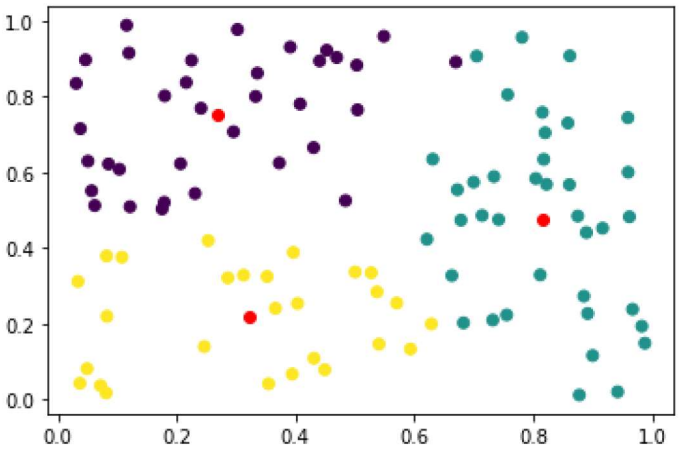
Strojno učenje je programiranje računala na način da optimiziraju neku kriterijsku funkciju temeljem podatkovnih primjera ili prethodnog iskustva (vidi [8]). Takve algoritme dijelimo na algoritme nadziranog (eng. *supervised*) i ne nadziranog (eng. *non-supervised*) učenja. Nadzirano učenje temelji se na izgradnji modela zasnivanog na bazi struktuiranih podataka (poznavanje "točnog" odgovora), dok kod ne nadziranog učenja nemamo takve podatke, već pokušavamo stvoriti takvu strukturu klasteriranjem podataka.

- Primjer nadziranog učenja



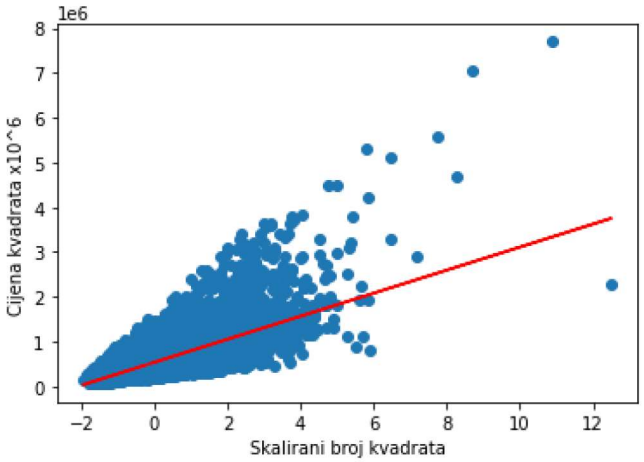
Slika 2: Rezultat učenja nad struktuiranim podacima

- Primjer ne nadziranog učenja



Slika 3: Klasteriranje podataka

2.2.1 Linearna regresija



Slika 4: Cijene kvadrata nekih nekretnina

Neka je dan skup podataka kao na slici, takve podatke zovemo trening podaci i označavamo ih s $(x^{(i)}, y^{(i)})$, $\forall\, i = 1, \ldots, m$ pri čemu m, x, y redom predstavljaju: broj podataka, ulaznu varijablu (eng. *feature*) i izlaznu varijablu. U našem slučaju x je broj kvadrata, a y je cijena kvadrata pomnožena s 10^6. Želimo iz ovih podataka znati što bolje aproksimirati cijenu kvadrata iduće nekretnine, a to postižemo pronalaskom parametra θ funkcije

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Takvu funkciju zovemo model funkcija. Cilj nam je pronaći $h_\theta(x)$ koji prolazi što bliže y za (x, y) iz skupa za treniranje. Funkcije koje nam daju optimalne parametre θ za $h_\theta(x)$ zovemo kriterijske (eng. *loss*) funkcije, konkretno za ovaj primjer ćemo koristiti

$$J(x) = \frac{1}{2m} \cdot \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \to \min_\theta$$

Optimalna metoda za minimizaciju $J(\theta)$ je gradijentna metoda koja garantira minimum ako je funkcija koju minimiziramo konveksna. Ideja te metode je postizanje točke minimuma nakon ne određenog broja deriviranja funkcije $J(\theta)$ i ažuriranja parametra θ:

$$\theta^{\text{new}} = \theta - \alpha \cdot \frac{\partial\, J(\theta)}{\partial\, \theta}$$

Kada je razlika funkcije troška u dva koraka izrazito mala metoda prestaje. Postupak dobivanja optimalnih parametara θ zovemo treniranje. Rezultat treniranja nad skupom podataka sa slike 4 je funkcija obojana u crveno.

2.2.2 Problem klasifikacije u dvije kategorije

U praksi se često javlja problem ovakvog oblika gdje je potrebno klasificirati podatak u jednu od dvije moguće kategorije. Primjerice, je li e-mail *spam* ili nije *spam*, je li tumor maligni ili benigni i slično. Takav problem možemo rješiti uz pomoć logističke regresije. Model funkcija je definirana s $h_\theta : \mathbb{R}^n \to <0,\, 1>$, tj. $h_\theta(x)$ definira vjerojatnosnu distribuciju na trening podacima.

$$h_\Theta(x) = P\left(y = 1 \,|\, x;\ \theta\right) = \sigma\left(\Theta^T x\right)$$

Pri čemu je $\sigma(z) = \dfrac{1}{1+e^{-z}}$ logistička funkcija ili sigmoid, $\Theta = [\theta_0\, \theta_1 \cdots \theta_n]^T$ i $x = [1\, x_1 \cdots x_n]$.

Podatak klasificiramo u prvu kategoriju ako je $h_\theta(x) \geq 0.5$. Ideja za odabir funkcije troška je maksimiziranje vjerojatnosti da $h_\theta(x)$ preslikava x u pravu kategoriju. Taj uvjet nam ispunjava funkcija maskinalne vjerodostojnosti (eng. *maximum likelihood*).

$$\mathring{J}(\Theta) = \sum_{i=1}^{m} P\left(y^{(i)} \,\middle|\, x^{(i)};\ \Theta\right) \to \max_{\Theta}$$

Možemo svesti gornji izraz na traženje minimuma.

$$J(\Theta) = -\log\left(\sum_{i=1}^{m} P\left(y^{(i)} \,\middle|\, x^{(i)};\ \Theta\right)\right)$$

$$J(\Theta) = -\sum_{i=1}^{m} \log P\left(y^{(i)} \,\middle|\, x^{(i)};\ \Theta\right)$$

$$J(\Theta) = -\sum_{i=1}^{m} \log \sigma\left(y^{(i)} \Theta^T x^{(i)}\right)$$

$$J(\Theta) = \sum_{i=1}^{m} \log\left(1 + e^{-y^{(i)} \Theta^T x^{(i)}}\right) \to \min_{\Theta}$$

Pošto je ona konveksna možemo pronaći optimalne parametre gradijentnom metodom.

Algoritam 1: Gradijentna metoda

Postavi Θ na neku početnu vrijednost;
while *ne konvergira* **do**
$\quad \Theta_j = \Theta_j - \alpha \cdot \dfrac{\partial}{\partial \Theta_j} J(\Theta), \quad \forall j = 0, \ldots, n$

Pri čemu je gradijent funkcije troška dan s

$$\frac{\partial}{\partial \Theta_j} J(\Theta) = \left(h_\Theta(\Theta^T x) - y\right) x$$

2.2.3 Problem klasifikacije u više kategorija

Ne rijetko se pokazuje potreba za klasifikacijom u više kategorija, npr. prognoza vremena, vrsta riječi u rečenici, u kojem padežu je riječ napisana i slično. U ovom slučaju nam trening podaci imaju oblik $(x^{(i)}, y^{(i)})$, $i = 1, \ldots, m$, $y^{(i)} \in \{1, \ldots, k\}$. Ideja za rješavanje ovakvog problema je generalizacija koncepta logističke regresije tako da nam je funkcija model $h_\Theta : \mathbb{R}^n \to \mathbb{R}^k$ oblika

$$h_\Theta(x) = \begin{bmatrix} P(y = 1 \,|\, x;\ \Theta) \\ P(y = 2 \,|\, x;\ \Theta) \\ \vdots \\ P(y = k \,|\, x;\ \Theta) \end{bmatrix}, \ \sum_{i=1}^{k} P\left(y = i \,|\, x;\ \Theta\right) = 1$$

Konkretno, naša model funkcija $h_\Theta : \mathbb{R}^n \to \mathbb{R}^k$ je definirana s

$$h_\Theta(x) = \sigma\left(\Theta^T x\right), \ \Theta \in \mathbb{R}^{k \times (n+1)}, \ x \in \mathbb{R}^{n+1}$$

Pri čemu funkciju $\sigma : \mathbb{R}^k \to \mathbb{R}^k$ zovemo *soft-max*

$$\sigma(z)_j = \sigma_j = \frac{e^{z_j}}{\sum_{l=1}^{k} e^{z_l}},$$
$$\sum_{j=1}^{k} \sigma(z)_j = 1$$

Izgled model funkcije je onda

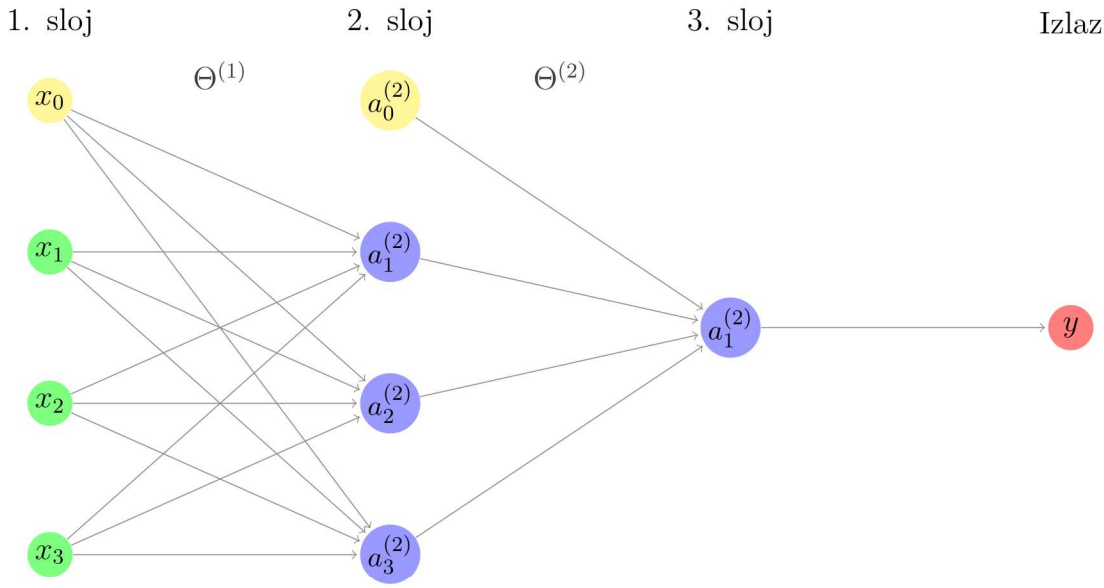
$$h_\Theta(x) = \sigma \begin{bmatrix} {\Theta^{(1)}}^T x \\ {\Theta^{(2)}}^T x \\ \vdots \\ {\Theta^{(k)}}^T x \end{bmatrix} = \begin{bmatrix} \sigma({\Theta^{(1)}}^T x) \\ \sigma({\Theta^{(2)}}^T x) \\ \vdots \\ \sigma({\Theta^{(k)}}^T x) \end{bmatrix} = \frac{1}{\sum_{l=1}^{k} e^{{\Theta^{(l)}}^T x}} \begin{bmatrix} e^{{\Theta^{(1)}}^T x} \\ e^{{\Theta^{(2)}}^T x} \\ \vdots \\ e^{{\Theta^{(k)}}^T x} \end{bmatrix}$$

Kao i u klasifikaciji na dvije kategorije, koristimo *maximum likelihood* funkciju troška za koju znamo da možemo izračunati optimalne parametre.

$$J(\Theta) = -\log \mathring{J}(\Theta)$$
$$J(\Theta) = -\sum_{i=1}^{m} \log P\left(y^{(i)} \,\middle|\, x^{(i)};\ \Theta\right)$$
$$J(\Theta) = -\sum_{i=1}^{m} \sum_{j=1}^{k} I^j(y^{(i)}) \log \left(\frac{e^{{\Theta^{(j)}}^T x^{(i)}}}{\sum_{l=1}^{k} e^{{\Theta^{(l)}}^T x^{(i)}}} \right) \to \min_\Theta, \ I^j(y^{(i)}) = \begin{cases} 1, & y^{(i)} = j \\ 0, & \text{inače} \end{cases}$$

2.3 Neuronske mreže

Kako bismo efikasnije trenirali znatno veći skup podataka koristimo neuronske mreže. Formalno, neuronska mreža je usmjeren aciklički graf $G = (V,\ E)$ s funkcijom težine $\theta : E \to \mathbb{R}$ gdje vrhovi predstavljaju neurone, a u svakom neuronu pridružena je aktivacijska funkcija $a : \mathbb{R} \to \mathbb{R}$. Odabirom (V, E, a) dobivamo arhitekturu neuronske mreže.



Slika 5: Aciklička neuronska mreža (eng. *feed-forward neural network*)

Između j i $j+1$ sloja se nalaze težine koje su oblika $\theta_j^{(l)}$ gdje l predstavlja neuron iz kojeg je veza povučena. Matricu takvih težina $\Theta^{(j)}$ nazivamo matrica parametara. $a_i^{(j)}$ označava aktivacijsku funkciju i-tog neurona u j-tom sloju. Svi slojevi između ulaznog i izlaznog se nazivaju skriveni. U svakom sloju osim zadnjem se uvijek (osim ako drugačije nije navedeno) nalazi još jedan neuron kojeg zovemo *bias* i njegova vrijednost je 1.

Izgled matrice parametara po slojevima neuronske mreže na slici iznad je idući

$$\Theta^{(1)} = \begin{bmatrix} \theta_{10}^{(1)} & \theta_{11}^{(1)} & \theta_{12}^{(1)} & \theta_{13}^{(1)} \\ \theta_{20}^{(1)} & \theta_{21}^{(1)} & \theta_{22}^{(1)} & \theta_{23}^{(1)} \\ \theta_{30}^{(1)} & \theta_{31}^{(1)} & \theta_{32}^{(1)} & \theta_{33}^{(1)} \end{bmatrix},\ \Theta^{(2)} = \begin{bmatrix} \theta_{10}^{(2)} & \theta_{11}^{(2)} & \theta_{12}^{(2)} & \theta_{13}^{(2)} \end{bmatrix}$$

Dok aktivacijske funkcije zapisujemo s

$$\begin{aligned} a_1^{(2)} &= a(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3), \\ a_2^{(2)} &= a(\theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3), \\ a_3^{(2)} &= a(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3), \\ a_1^{(3)} &= a(\theta_{10}^{(2)} \cdot 1 + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)}), \end{aligned}$$

To jest matričnim zapisom

$$a^{(j)} = a\left(\Theta^{(j-1)} \cdot a^{(j-1)}\right),\ \forall j = 2, 3, \ldots, l$$

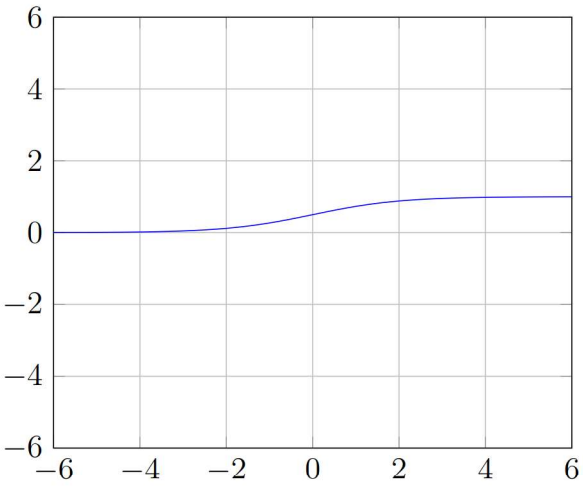
Pri čemu je $a_0^{(1)} = 1$, $a_1^{(1)} = x_1$, $a_1^{(2)} = x_2$, $a_1^{(3)} = x_3$, a l ukupan broj slojeva.

2.3.1 Izbor aktivacijske funkcije

Postoji mnogo aktivacijskih funkcija no navesti ćemo samo nekoliko bitnih.

- Logistička funkcija ili sigmoid

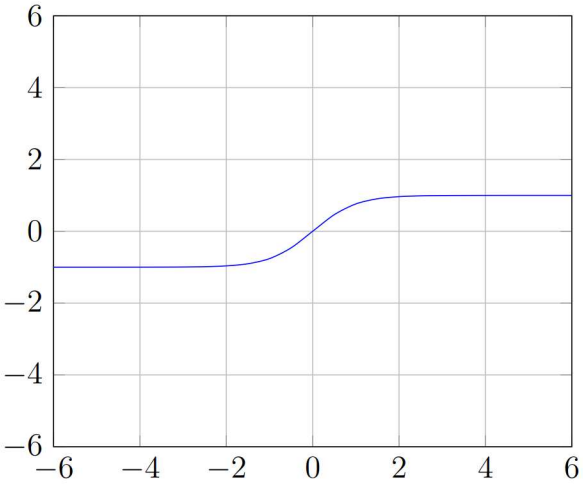
$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \in <0, 1>$$



Slika 6: Sigmoid

- Hiperbolna tangens funkcija

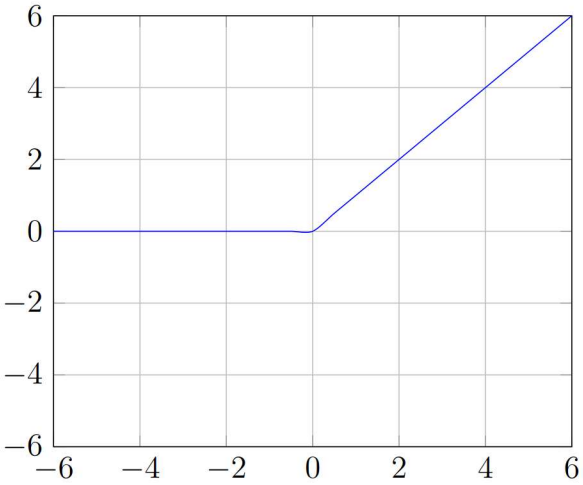
$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{-x} + e^{x}} \quad \in <-1, 1>$$



Slika 7: Hiperbolni tangens

- ReLU (eng. *Rectified Linear Unit*)

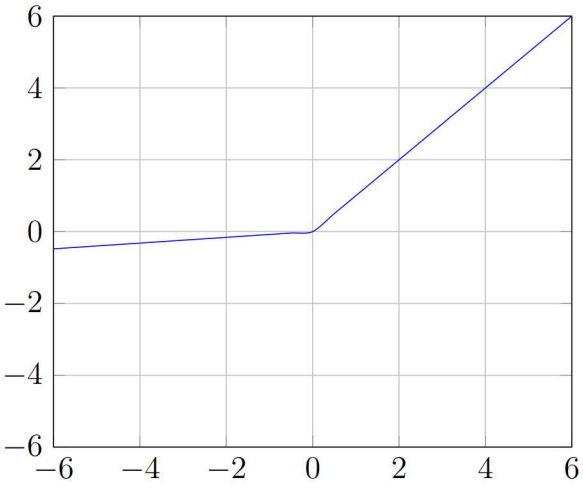
$$\mathrm{R}(x) = \max\{0,\, x\}$$



Slika 8: ReLU

- LeakyReLU (eng. *Rectified Linear Unit*)

$$\mathrm{LeakyReLU}(x) = \max\{0,\, x\} + \epsilon \cdot \min\{0,\, x\}$$



Slika 9: Leaky ReLU

Odabirom aktivacijske funkcije uvodimo ne linearnost u neuronsku mrežu.

2.4 Osnovna građa programskog okvira Rasa

Definirajmo pojmove: namjere, entiteta, radnje i odgovora te važnosti konteksta unutar rečenica. Uzmimo za primjer rečenice:

P1: Želim otići u pekaru Alfa.

P2: Želim otići u pekaru Alfa, jer mi je tamo bolji kruh nego u pekari Beta.

P3: Katapultirati ću te na Mars

P4: Katapultirati ću te na Mars da kupiš najbolji ručnik za autostopere.

P5: Koje je radno vrijeme Gradske knjižnice?

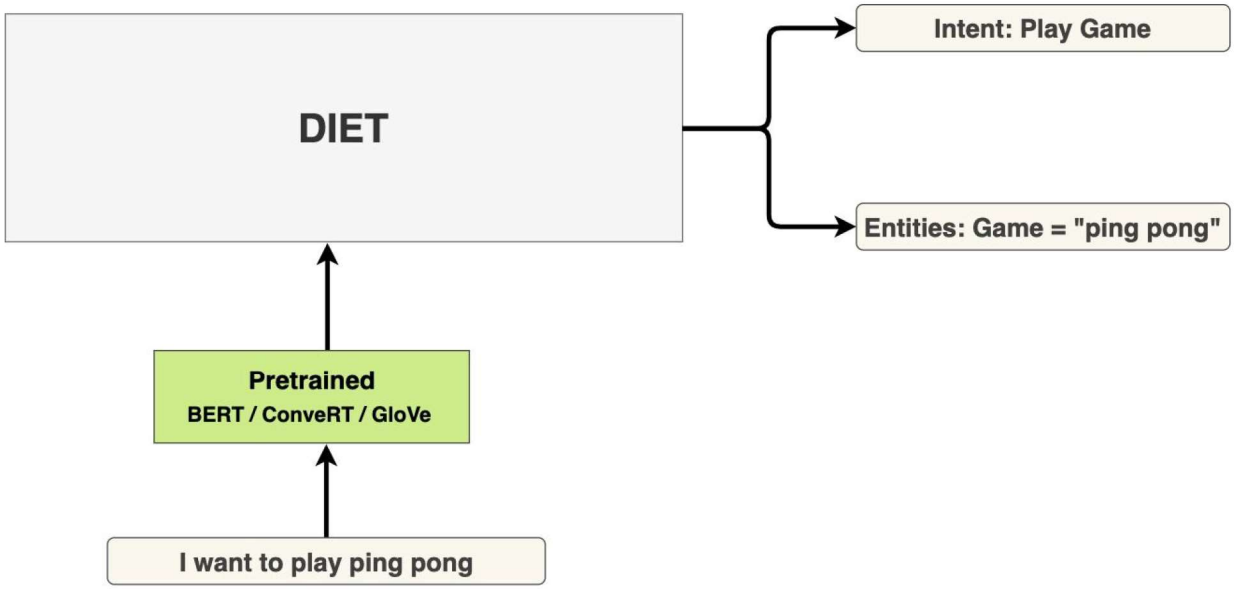
U svakoj rečenici možemo zaključiti namjeru osobe koja ju je napisala ili izrekla a to izrazito ovisi o kontekstu. Kontekst (lat. *contextus*: spajanje; spoj), u jezikoslovlju i fonetici, je naziv za istovjetne jedinice izričaja koje dolaze u blizini jedinice što ju promatramo, ili uz nju; svojom prisutnošću one uvjetuju prisutnost, oblik ili funkciju promatrane jedinice (vidi [7]). Mi s lakoćom iz rečenica izvlačimo namjeru i entitete pa u odnosu o njima izvršavamo radnje koje rezultiraju odgovorom, gestom ili mimikom.
U tom smislu, namjera iz rečenice P1 je odlazak u pekaru, a entitet je Alfa, tj. entitet je u ovom slučaju pekara u koju bi osoba htjela otići. Pošto je to izjavna rečenica na nju možemo odgovoriti s prihvaćanjem, možda kada bismo imali više konteksta bi mogli bolje odgovoriti ili čak pokrenuti razgovor s tom osobom. Računalo nema sposobnost shvaćanja konteksta kao ljudska osoba stoga mora stvoriti umjetno shvaćanje konteksta.

Primjer P2 pokazuje kako dodavanjem konteksta razgovor možemo odvesti u drugom smjeru, sada umjesto mnoštva odgovora koji su mogli proizaći iz primjera P1 imamo veću mogućnost razgovora o usporedbi dviju pekara i vlastitim iskustvima s njima. Kao i u prošlom primjeru namjera je i dalje odlazak u pekaru, no sada imamo dva entiteta: Alfa i Beta. Ovakva rečenica daje bogatiju sliku računalu s kakvom osobom priča ako razgovor nastavi u takvom obliku jer računalo može bolje upoznati osobu s kojom priča nego u razgovoru oblika P1.

Primjer P3 je interesantan jer se iz njega može izvući više namjera, gdje se ponovnim nedostatkom konteksta teže odlučiti za jednu. Primjer možemo protumačiti kao prijetnju, šalu ili doslovno značenje; računalo se ne može sa sigurnošću odlučiti za niti jednu od tri (uz uvjet da su sve tri namjere poznate računalu) te na kraju odabire onu za koju najviše ima primjera korištenja. Entitet u ovoj rečenici je Mars.

Primjer P4 daje jasniju sliku o namjeri iza te rečenice; u tom smislu mogli bi eliminirati prijetnju kao manje vjerojatnu namjeru, dok su šala i doslovno značenje vjerojatniji. Sada, u ovisnosti o prošlim iskustvima i razgovorima, bi odredili namjeru iza takve rečenice; na takav način i računalo bira jednu od te dvije namjere, uz uvjet da je eliminiralo prijetnju kao moguću namjeru. S odabirom entiteta možemo dobiti profinjeniji odgovor: recimo da smo odabrali namjeru doslovnog značenja, nama entitet Mars više nije dovoljan za adekvatan odgovor stoga uzimamo riječ ručnik kao drugi entitet. Računalo bi prepoznalo da mora kupiti ručnik na Marsu te bi takva radnja na kraju i izvršila kupovinu, ali taj ručnik ne bi morao biti ni za autostopere niti najbolji, što trivijalno riješimo uvođenjem dva dodatna entiteta: najbolji i autostopere.

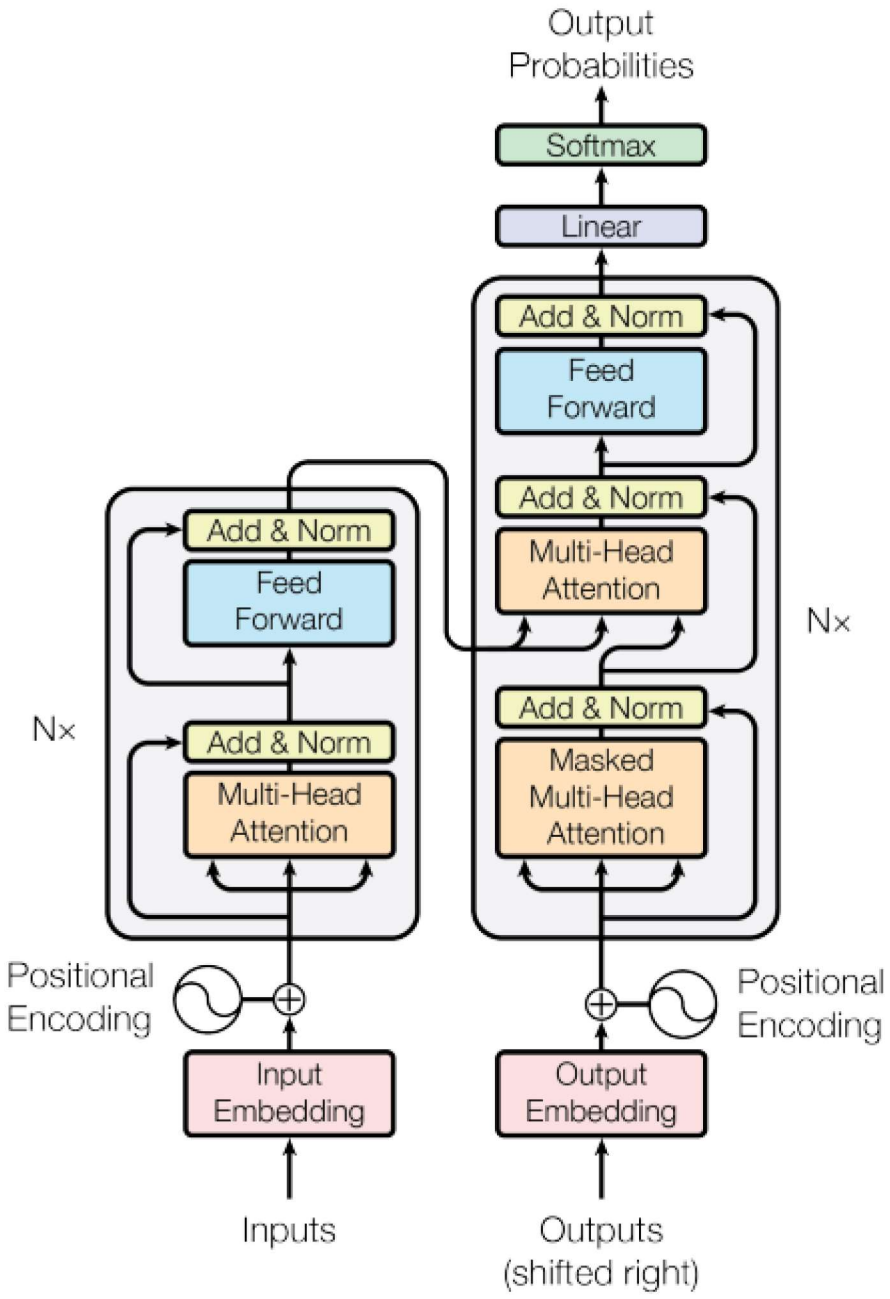
Entiteti nisu uvijek važni za razgovor, to možemo primijetiti iz primjera P5 gdje je namjera upit radnog vremena Gradske knjižnice. Pošto takvo pitanje ima samo jedan odgovor nije bitno odrediti entitet, osim ako je cilj izrada asistenta koji odgovara na samo jedno pitanje u kojem se mijenjaju ustanove.
Rasa programski okvir koristi njihovu DIET arhitekturu (Dual Intent Entity Transformer) koja dohvaća namjeru i entitete iz ulaznog teksta. Ako nas chatbot u razgovoru pita „Što želiš raditi?“, a mi odgovorimo sa „Igrati ping pong.“ naša namjera je igranje igre, a entitet je ping pong. Radnje koje će chatbot odraditi ovise o namjeri koju shvati iz teksta. Prema tim namjerama, u ovisnosti o entitetima, chatbot nam odgovara na našu izjavu ili pitanje.



Slika 10: Pojednostavljeni izgled DIET arhitekture

DIET (Dual Intent and Entity Transformer) je arhitektura s više zadataka: za klasifikaciju namjera i prepoznavanje entiteta. Arhitektura se temelji na transformatoru koji služi za oba zadatka. Entiteti su predviđeni kroz sloj označavanja uvjetnog slučajnog polja (eng. *Conditional Random Field*) koji se nalazi na vrhu izlaznog slijeda Transformatora koji odgovara ulaznom nizu tokena. Na sličan se način definiraju i namjere. Gubitak skalarnog produkta koristimo kako bismo povećali sličnost s ciljnim tokenom i smanjili sličnosti s negativnim uzorcima. U idućem poglavlju koristimo informacije iz [9] i [10] osim na mjestima gdje je navedeno drugačije.

2.5 Transformer arhitektura



Slika 11: Transformer

Transformer je arhitektura koja u sebi sadrži dio za enkodiranje i dekodiranje podatka (eng. *encoder-decoder*). Na slici se s lijeve strane nalazi dio za enkodiranje, a desno za dekodiranje. Zadatak *encodera* je pretvaranje ulaznog niza vektora riječi $(x_1, \ldots, x_n)$ u niz kontinuiranih reprezentacija $z = (z_1, \ldots, z_n)$. Dekoder zatim generira izlazni niz $(y_1, \ldots, y_m)$ simbola za svaki element iz z. [6]

2.5.1 Enkoder

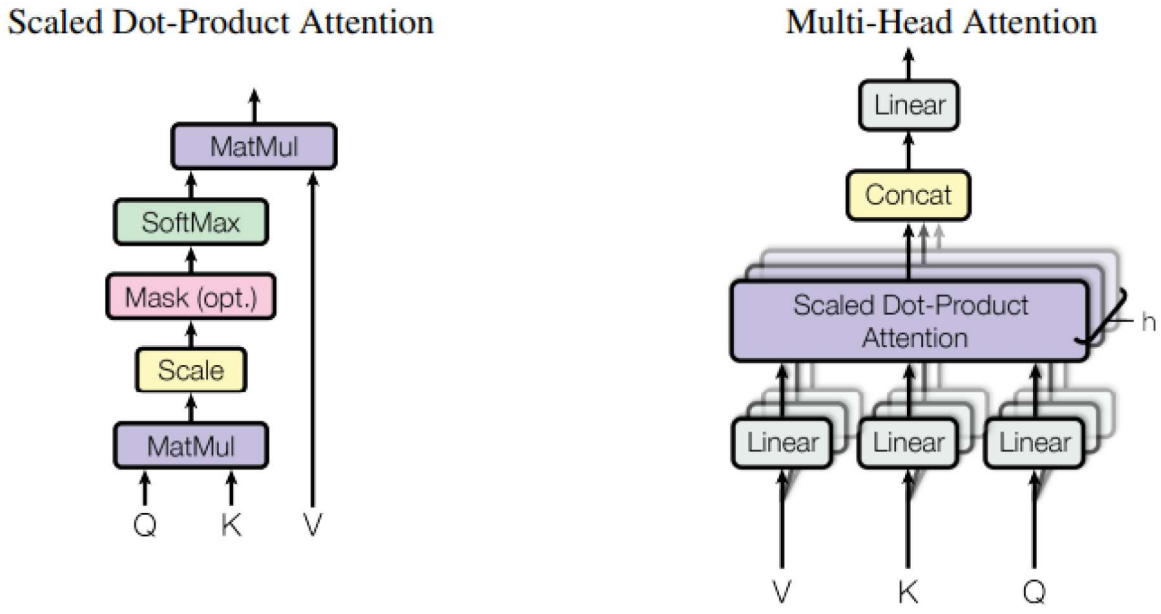
Enkoder se sastoji od N identičnih slojeva pri čemu svaki sloj sadrži dva pod-sloja. Prvi pod-sloj je paralelizirani mehanizam samopažnje (eng. *multi-head self-attention*), a drugi je aciklička neuronska mreža s ReLU aktivacijskom funkcijom. Ulaz svakog pod-sloja se dijeli na put koji trenira i ne trenira parametre pri čemu oba završavaju sa slojem normalizacije, tj. izlaz svakog pod-sloja je dan s formulom $\text{Norma}(x + \text{Pod-sloj}(x))$ pri čemu je $\text{Pod-sloj}(x)$ implementacija trenutnog pod-sloja. Na svakom izlazu pod-slojeva se generira vektor dimenzije $d_{\text{model}} = 512$.

2.5.2 Dekoder

Dekoder se također sastoji od N identičnih slojeva, no za razliku od enkodera, u sebi sadrži tri pod-sloja. Treći pod-sloj je paralelizirani mehanizam samopažnje koji na ulazu prima izlaz enkodera. Slično kao i kod enkodera, u dekoderu se također put dijeli na put koji trenira i ne trenira parametre nakon čega ponovno kod svakog slijedi normalizacijski sloj. U dekoderu je promijenjen mehanizam samopažnje tako da riječi koje su na trenutnoj poziciji ne ovise o riječima koje dolaze nakon nje, tj. predikcija za riječ na poziciji i može samo ovisiti o riječima koje se nalaze na mjestima manje od i.

2.5.3 Samopažnja

Funkciju samopažnje moguće je opisati kao preslikavanje upita (eng. *query*) sa skupom parova ključeva (eng. *keys*) i vrijednosti (eng. *values*) na neki izlaz, pri čemu su upiti, ključevi, vrijednosti i izlaz vektori. Svakoj riječi pridružujemo navedena tri vektora množenjem vektora riječi s matricom W_Q za dobivanje upitnog vektora, W_K za ključ i W_V za vrijednost pri čemu su matrice W_Q, W_K, W_V dobivene treniranjem modela.



Slika 12: Samopažnja skalarnim produktom (lijevo) i paralelizirana samopažnja (desno)

2.5.4 Samopažnja skalarnim produktom

Ulaz se sastoji od upita i ključeva dimenzije d_k i vrijednosti dimenzije d_v. Formula za izračun samopažnje je dana s

$$\text{Samopažnja}(Q,\, K,\, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Na ovaj način možemo samo nad jednoj riječi vršiti funkciju samopažnje.

2.5.5 Paralelizirana samopažnja

U ovom slučaju moguće je provoditi samopažnju nad više riječi koje se nalaze na različitim pozicijama. U nastavku je dana formula za računanje paralelizirane samopažnje.

$$\begin{gathered}\text{ParalelnaSamopažnja}(Q,\, K,\, V) = \text{Konkatenacija}(\text{nit}_1,\, \ldots,\, \text{nit}_h)W^O,\\ W^O \in \mathbb{R}^{h\cdot d_v \times d_{\text{model}}}\end{gathered}$$

pri čemu su $\text{nit}_1, \ldots, \text{nit}_h$

$$\begin{gathered}\text{nit}_i = \text{Samopažnja}(QW_i^Q,\, KW_i^K,\, VW_i^V),\\ W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k},\\ W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k},\\ W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}\end{gathered}$$

2.5.6 Aciklička neuronska mreža

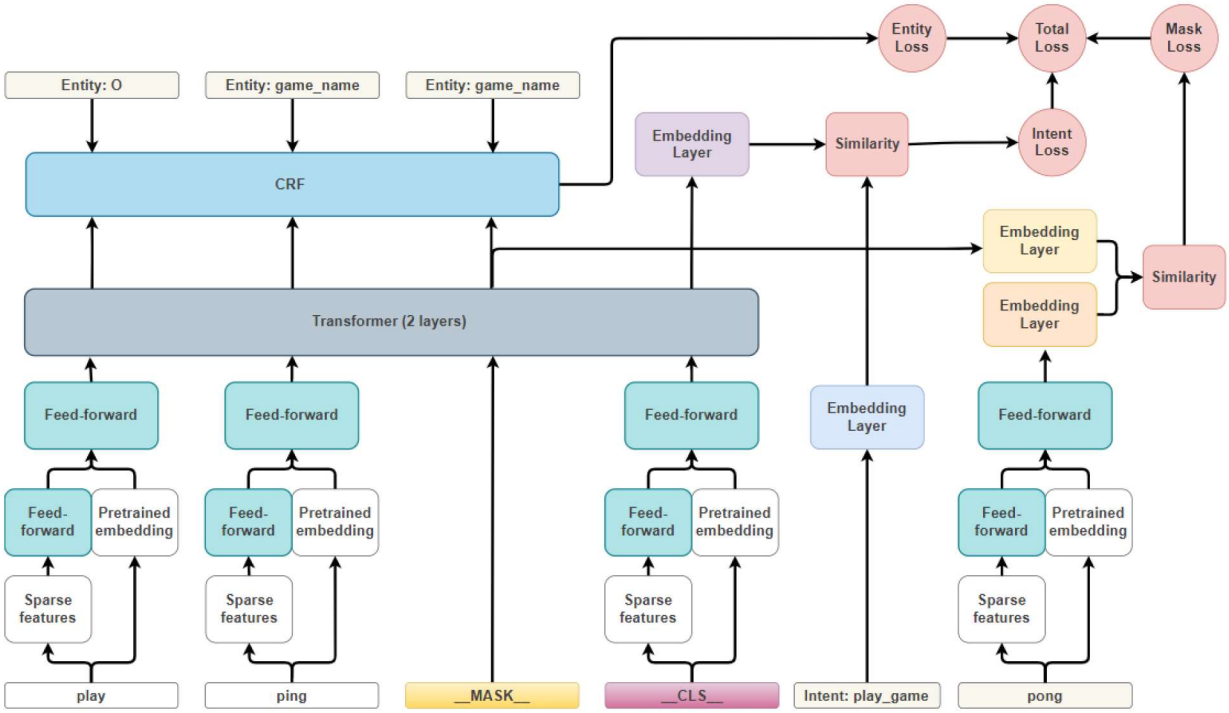
U svakom pod-sloju enkodera i dekodera se nalazi i aciklička neuronska mreža koja ne dijeli težine s ostalim mrežama. Ona se sastoji od dva linearna sloja s ReLU aktivacijskom funkcijom koja ih povezuje.

$$\text{FFN}(x) = \max(0,\, xW_1 + b_1)W_2 + b_2$$

2.5.7 Pozicijsko enkodiranje

Model ne uzima podatke u nizu kao što to radi rekurentna neuronska mreža, stoga informacija o bitnosti redoslijeda riječi unutar rečenice uvodi se kroz pozicijsko enkodiranje (eng. *positional encoding*) postavljeno na početku enkodera i dekodera.

2.6 DIET arhitektura



Slika 13: DIET arhitektura [4]

Iz prošlog primjera digitalni asistent je prepoznao našu namjeru „igranje igre“, i entitet „ping pong“. Vidljivo sa slike, namjere, entiteti i *token*i (sve ostale riječi) imaju svoje mjesto ulaska u arhitekturu. Pogledajmo *token* „play“. Postoje dva smjera koja će rukovati s *token*om: prethodno trenirani smjer u kojem se nalazi jedna od prethodno treniranih neuronskih mreža čija je zadaća prihvatiti *token* i vratiti vektor iz $\mathbb{R}^n$. Možemo primijetiti da se na više mjesta pojavljuje isti smjer s nazivom *pretrained embedding* koji riječ pretvara u vektor riječi, te drugi smjer u kojem se iz *token*a vade vektori koji predstavljaju neku informaciju o tom *token*u. Riječ „play“ možemo zapisati kao One Hot Encoding vektor iz rječnika:

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

slova unutar riječi:

$$\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 1 \\ 1 \end{bmatrix}$$

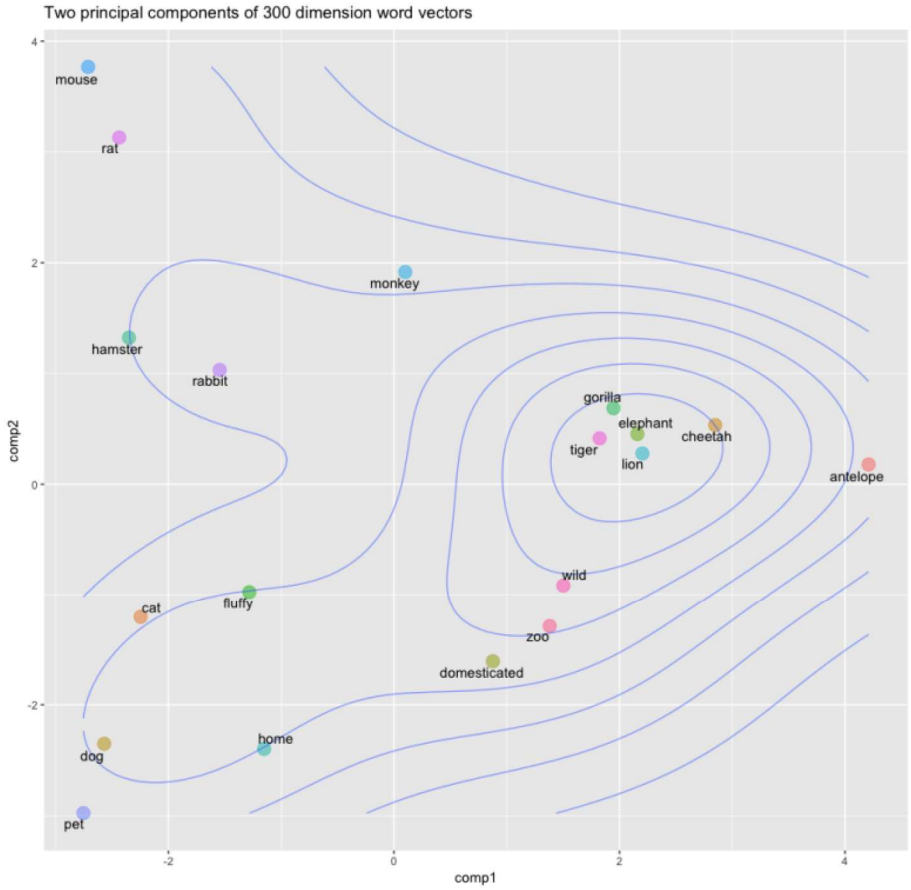
i slično.

Tako dobiveni vektori se konkateniraju u jedan vektor koji se prosljeđuje na ulaz acikličkoj neuronskoj mreži. Izlaz prvog smjera i drugog smjera se konkateniraju i ponovno prosljeđuju kao ulaz u acikličku neuronsku mrežu, izlazni vektor je tada iz $\mathbb{R}^{256\times1}$. U svrhu sprječavanja pre-treniranog modela, 80% veza između slojeva se odbaci (eng. *dropout*); pre-trenirani modeli su oni koji s velikom preciznošću klasificiraju već viđene podatke iz skupa za treniranje, no jako loše sve ostale. Na isti način rade svi *token*i koji na kraju ulaze u Transformer sloj. Pogledajmo ___CLS___ *token*. Isto kao i kod prethodnih postoje dva smjera, u prethodno treniranom smjeru izlazni vektor je srednja vrijednost izlaza iz istog smjera ostalih *token*a, dok je prvi smjer suma svih vektora koji se dobiju izlazom iz prvog smjera. Analogno ostalim smjerovima oba izlaza se konkateniraju i prosljeđuju ulazu acikličkoj neuronskoj mreži čiji je izlaz vektor iz $\mathbb{R}^{256\times1}$. Možemo zaključiti da je ___CLS___ svojstvo koje opisuje cijelu rečenicu. Težine acikličkih neuronskih mreža su međusobno dijeljene u svojim raznima. U svrhu pretvaranja modela u generalizirali model, tj. onaj koji se može prilagoditi novim podacima, DIET arhitektura u sebi sadrži i *token* ___MASK___. Algoritam nasumično makne riječi s ulaza i stavi ih u ___MASK___, nakon čega pokuša pogoditi koje su se riječi nalazile na praznim mjestima gdje su one originalno stajale. Takav pothvat također služi za regularizaciju modela, tj. smanjenje pre-naučenosti. U našem primjeru s prošle slike, *token* „Pong“ je nasumično odabran te je stavljen u ___MASK___. Kao i ostali *token*i, prolazi kroz oba smjera te se na kraju uspoređuje sličnost pretpostavljene riječi s originalnom. Razlika se sprema u varijablu *Mask Loss*; u ovom smislu razlika je definirana kao kosinus udaljenosti između dva vektora riječi.

Sve informacije i slike o vektorima riječi možemo pronaći na [5], ukratko, riječ možemo prikazati kao vektor stupac iz $\mathbb{R}^k$ gdje je u svakom retku skrivena informacija o bliskosti s nekim pojmom.

Slika 14: Vektori riječi u četiri dimenzije

Prikazivanjem takvih vektora dobivamo jasnu korelaciju između riječi u vidu udaljenosti među njima. Ako je udaljenost velika onda je sličnost te dvije riječi mala, analogno ako je udaljenost mala sličnost je velika. Primjerice, riječ „Mačka“ i „Pas“ imaju veliku udaljenosti, dok „Mačka“ i „Lav“ imaju manju.



Slika 15: Prikaz vektora riječi

Riječi "kralj" i "kraljica" su slične riječi jer ih možemo opisati sa sličnim pojmovima: obitelj, plemstvo, moć, povijest, spol i slično.

Kao što ___MASK___ u našem primjeru pogađa riječ „Pong“, ___CLS___ pokušava pogoditi koja je namjera naše rečenice, te se promotri razlika između pogođene namjere i stvarne namjere gdje se vrijednost razlike sprema u varijabli *Intent Loss*. Promotrimo sloj Transformera. U DIET arhitekturi se nalaze dva sloja Transformera koji čija je zadaća proširiti kontekst unutar rečenice. Idući blok je CRF (eng. *Conditional Random Field*) koji pokušava pogoditi koji *token*i su odabrani za entitete. Kao i kod ___MASK___ i ___CLS___ postoji pogreška predikcije koja se sprema u varijablu Entity Loss, koja je zadnji korak za finalnu pogrešku Total Loss koja je suma ostalih, a koja služi za treniranje modela.

2.7 Implementacija

Implementacija vlastitog digitalnog asistenta u Rasa programskom okviru započinje s pisanjem trening podataka.

2.7.1 Trening podaci

- **NLU podaci** (eng. *NLU data*)
  Kako bi napravili digitalnog asistenta u Rasi potrebno je napraviti pet koraka. Digitalnom asistentu je potrebno razumijevanje namjere, stoga ih u prvom koraku definiramo zajedno s primjerima rečenica koji predstavljaju tu namjeru. To radimo da bi pomoćnik prepoznao ono što korisnik govori bez obzira na to kako korisnik formulira poruku.

```
nlu:
- intent: greet
  examples: |
    - Hej
    - Bok!
    - Hello
    - Dobar dan!
    - Dobro jutro!

- intent: subscribe
  examples: |
    - Želim primati obavijesti o novim proizvodima.
    - Možete mi slati obavijesti?
    - Obavijestite me kada dođe novi proizvod.

- intent: inform
  examples: |
    - Moj email je primjer@primjer.com
    - primjer@primjer.com
    - Molim vas pošaljite na primjer@primjer.com
    - Email je primjer@primjer.com
```

Slika 16: NLU podaci

- **Priče** (eng. *Stories*)
  Priče su primjeri razgovora koji uče asistenta da pravilno odgovori, ovisno o tome što je korisnik prethodno rekao u razgovoru. Format priče prikazuje namjeru korisničke poruke nakon koje slijedi pomoćnikova radnja ili odgovor.

```
stories:
  - story: greet and subscribe
    steps:
    - intent: greet
    - action: utter_greet
    - intent: subscribe
    - action: newsletter_form
    - active_loop: newsletter_form
```

Slika 17: Priče

- **Odgovori** (eng. *Responses*)
  Sada kad digitalni asistent razumije neke rečenice potrebni su mu odgovori koje može poslati nazad korisniku.

```yaml
responses:
  utter_greet:
    - text: |
        Hej! Kako ti mogu pomoći?
    - text: |
        Hej!
  utter_ask_email:
    - text: |
        Koja je tvoja email adresa?
  utter_subscribed:
    - text: |
        Pogledaj svoj e-pretinac na {email} kako bi završio pretplatu na obavijesti!
    - text: |
        Gotovo! Pogledaj svoj {email} za završetak pretplate na obavijesti.
```

Slika 18: Odgovori

- **Obrasci** (eng. *Forms*)
  Postoje mnoge situacije u kojima pomoćnik mora prikupljati podatke od korisnika. Na primjer, kada se korisnik želi pretplatiti za obavijesti, asistent mora zatražiti e-adresu.

```yaml
forms:
  newsletter_form:
    required_slots:
      email:
      - type: from_text
```

Slika 19: Obrasci

- **Pravila** (eng. *Rules*)
  Pravila opisuju dijelove razgovora koji uvijek trebaju ići istim putem, bez obzira na to što je prethodno rečeno u razgovoru. Želimo da naš pomoćnik uvijek reagira na određenu namjeru s određenom radnjom.

```yaml
rules:
 - rule: activate subscribe form
   steps:
   - intent: subscribe
   - action: newsletter_form
   - active_loop: newsletter_form

 - rule: submit form
   condition:
   - active_loop: newsletter_form
   steps:
   - action: newsletter_form
   - active_loop: null
   - action: utter_subscribed
```

Slika 20: Pravila

- **Prilagođena radnja (eng. *Custom Action*)**
  Prilagođena radnja je Python skripta u kojoj je moguće raditi složeniju vrstu odgovora. Možemo se spojiti na bazu podataka ili programirati neku funkcionalnost.

```
from typing import Text, Dict, Any, List
from rasa_sdk import Action
from rasa_sdk.events import SlotSet

class ActionCheckRestaurants(Action):
   def name(self) -> Text:
      return "action_check_restaurants"

   def run(self,
           dispatcher: CollectingDispatcher,
           tracker: Tracker,
           domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

      cuisine = tracker.get_slot('cuisine')
      q = "select * from restaurants where cuisine='{0}' limit 1".format(cuisine)
      result = db.query(q)

      return [SlotSet("matches", result if result is not None else [])]
```

Slika 21: Pravila

2.7.2 Pokretanje

Kada smo gotovi s trening podacima, možemo ga početi trenirati. Naredbom *Rasa train* pokrećemo treniranje te nakon što završi, s digitalnim asistent možemo pričati putem konzole naredbom *Rasa shell*. Ako želimo isprobati kako radi Rasa bez preuzimanja programskog okvira na osobno računalo, na njihovoj web stranici je moguće napraviti svog probnog digitalnog asistenta te se okušati u komunikaciji [12]. Za izradu digitalnog asistenta s više mogućnosti pogledajte [9]. Osim dvije naredbe koje služe za treniranje i pokretanje digitalnog asistenta, bitno je naglasiti i još neke koje mogu biti bitne za implemntaciju.

- ***rasa data split nlu*** - podjela 80/20 za skup za treniranje i testiranje
- ***rasa test*** - testiranje modela
- ***rasa run*** - pokretanje poslužitelja s trenutnim modelom
- ***rasa run actions*** - pokretanje poslužitelja prilagođenih radnji
- ***rasa visualize*** - generiranje vizualne reprezentacije Priča
- ***rasa data validate*** - testiranje validnosti strukture projekta
- ***rasa -h*** - prikazivanje svih naredbi

3 Zaključak

Na početku rada smo se dotakli povijesti chatbota, kako su nastali i zašto postoje. Već u uvodnom dijelu rada možemo naslutiti kompliciranost stvaranja takve tehnologije te je zbog toga jasno vidljiva potražnja za jednostavnijim pristupom dobivanja digitalnog asistenta. Kao moguće rješenje predstavljen je programski okvir Rasa u kojem je moguće na jednostavan način napraviti vlastitog digitalnog asistenta. Dotakli smo se osnovnih pojmova strojnog učenja i neuronskih mreža. Prošli smo kroz funkcionalnost DIET arhitekture koja je ujedno i glavna arhitektura za klasifikaciju namjera i entiteta u rečenici, te smo se upoznali s pojmom samopažnje i Transformer arhitekture koja je ključni DIET arhitekture. Cilj rada bio je upoznavanje čitatelja s tehnologijom koja je sveprisutna u današnjici.

Literatura

[1] *Alex Debecker, August 11th, 2017* https://blog.ubisend.com/discover-chatbots/chatbot-eliza

[2] *Chatbot Jabberwacky* https://en-academic.com/dic.nsf/enwiki/371044

[3] *Mady Mantha, March 9th, 2020* https://blog.rasa.com/introducing-dual-intent-and-entity-transformer-diet-state-of-the-art-performance-

[4] *Vincent D. Warmerdam, DIET architecture interactive demo* http://bl.ocks.org/koaning/raw/f40ca790612a03067caca2bde81e7aaf/

[5] *Jayesh Bapu Ahire, March 14th, 2018* https://dzone.com/articles/introduction-to-word-vectors

[6] *Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin, Attention is all you need, 2017* https://papers.nips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[7] *kontekst. Hrvatska enciklopedija, mrežno izdanje. Leksikografski zavod Miroslav Krleža, 2021.* https://www.enciklopedija.hr/Natuknica.aspx?ID=32920

[8] *E. Alppaydin, Introduction to Machine Learning, The MIT Press, London, 2010.*

[9] *Rasa Dokumentacija* https://rasa.com/docs/rasa/

[10] *Rasa Alghoritm Whiteboard* https://www.youtube.com/watch?v=23XUv0T9L5c&list=RDCMUCJ0V6493mLvqdiVwOKWBODQ&ab_channel=Rasa

[11] *GitHub repozitorij* https://github.com/BornaGajic/FAQ-Mathos-Chatbot

[12] *Rasa Playground* https://rasa.com/docs/rasa/playground