Evolucija koncepta umjetne inteligencije

Pantar, Antonija

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Antonija Pantar

EVOLUCIJA KONCEPTA UMJETNE INTELIGENCIJE

Diplomski rad

Voditelj rada:
doc. dr. sc. Vedran Čačić

Zagreb, srpanj, 2021.

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

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

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Umjetna inteligencija (*UI*) kao disciplina smatra se svojevrsnim „filozofskim inženjerstvom". Na osnovnoj razini umjetna inteligencija pokušava napraviti strojeve koji mogu misliti, znati, razumjeti, ponašati se racionalno, nositi se s nesigurnostima i surađivati, odnosno, pokušava filozofske koncepte i ideje pretvoriti u strojeve i algoritme. Rijetko ćemo se susresti s konkretnim opisima algoritama UI kao što su program za identifikaciju zlonamjernog softvera, aplikacija koja daje formalni dokaz teorema ili algoritam koji prepoznaje slova na skeniranom papiru — svi ovi algoritmi bili su vrlo poznati i bitni za razvoj koncepta UI, međutim, kada su implementirani nismo ih više smatrali „inteligentnima" nego su postali samo razrađena izračunavanja. Kada je sustav potpuno specificiran i primjenjuju se samo poznati alati, strojevi koji koriste ove algoritme prestaju biti mistični i nalik ljudima te postaju obična računala. UI ima tendenciju vraćanja na stare probleme i po tome je slična filozofiji, ali zahtijeva mjerljiv napredak (nove tehnike moraju donijeti nešto novo — bolje rezultate, formulaciju novog problema, generalizaciju, . . .) te je po tome bliska inženjerstvu. Ovdje je ključna razlika između filozofije i inženjerstva — filozofija se stalno vraća starim problemima dok se inženjeri, kada završe jedan problem, na njega više ne vraćaju (smatraju ga riješenim), već to rješenje koriste pri rješavanju budućih većih problema.

Počeci umjetne inteligencije dvije su filozofske ideje Gottfrieda Leibniza, filozofa i matematičara koji je živio u 17. stoljeću. Jedna od njih bila je *characteristica universalis*, idealizirani jezik na koji bi se mogle svesti sve znanosti i svi prirodni jezici. Kao takav, bio bi to svojevrstan univerzalni jezik čistog znanja (činjenica), bez lingvističkih tehnikalija. Taj jezik bi se koristio u svrhu objašnjenja racionalnog razmišljanja i to toliko precizno da bi se mogao napraviti stroj, pod nazivom *calculus ratiocinator*, s kojim se tim jezikom može komunicirati.

Ljude je oduvijek fascinirao pojam inteligencije: kako ona nastaje te možemo li izgraditi umjetni sustav koji je inteligentan. Znanstvenici su željeli izgraditi sustav za obradu podataka koji bi oponašao biološke neurone i njihovu međusobnu povezanost, a učio bi samostalno (takozvani *konektivistički pristup* umjetnoj inteligenciji). Definiraju se i grade jednostavne procesne jedinice (umjetni neuroni) koji se potom povezuju u paralelne strukture različitih arhitektura — *umjetne neuronske mreže*. Prvi značajan rad u ovom području

je *A Logical Calculus of Ideas Immanent in Nervous Activity*, članak Warrena McCullocha i Waltera H. Pittsa koji su napisali 1943. godine. U njemu predstavljaju prvi model umjetne neuronske mreže. Konektivistički pristup nastavlja se razvijati sve do pojave perceptrona (1958. godine; 3.3) za koji se 1969. godine dokazuje da ne može izračunati osnovne logičke operacije (konkretno operator XOR).

U prvoj polovici 20. stoljeća logika se smatrala usko povezanom s procesima razmišljanja i inteligencijom pa se tako kognitivna znanost, nakon što neuronske mreže nisu „urodile plodom", okreće novom *simboličkom pristupu*. 1956. godine održava se *Darthmouth Summer Research Project on Artificial Inteligence* — prijedlog sastanka bio je da se svaki aspekt učenja i inteligencije može toliko precizno opisati da se može napraviti stroj koji će ga simulirati. Time klasična UI postaje *logička (simbolička) UI*.

U međuvremenu, Thomas Kuhn 1962. predlaže pomak paradigme (*paradigm shift*) koji pomaže razvitku kognitivne znanosti i vraćanju konektivističkom pristupu. Po prvi puta se činilo dobrim napustiti najnovije ideje i vratiti se starima. 1986. godine D. E. Rumelhart, G. E. Hinton i R. J. Williams objavljuju članak *Learning internal representations by error propagation* u kojem predstavljaju *backpropagation* algoritam, čime počinje kognitivno razdoblje dubokog učenja.

Poglavlje 1

Početci UI

Većina informacija u ovom poglavlju preuzeta je iz [12].

Norbert Wiener i Arturo Rosenblueth, zaslužni su za početak sustavnog proučavanja umjetne inteligencije. Rosenblueth je tada vodio mjesečne sastanke u dvorani Vanderbilt, na kojima se raspravljalo o znanstvenoj metodi. Nakon večere netko iz grupe, ili gostujući znanstvenik, pročitao bi članak na neku znanstvenu temu te bi se o njemu raspravljalo. Norbert Weiner pridružio se grupi nakon sudjelovanja na seminaru Josiaha Roycea na Harwardu, također s temom znanstvene metode, na kojem je sudjelovao između 1911. i 1913. godine. Bio je aktivni član Rosenbluethove grupe sve dok Rosenblueth nije pozvan u Meksiko 1944. godine. Ulaskom SAD-a u rat, dolazi do prekida financiranja takvih „neesencijalnih" projekata pa tako i ovih sastanaka. Weiner i Rosenbleuth bili su uvjereni da kako bi se znanost općenito razvijala, moraju se istražiti područja „između" razvijenih polja znanosti. Obično svaki znanstvenik ima svoje područje rada. U tom području on će znati terminologiju, literaturu i postupke, ali smatrat će da se nema što uplitati u posao kolege iz drugog područja, uglavnom zato što ne poznaje literaturu i terminologiju tog područja. Rosenbleuth je bio uvjeren da ovakve interdisciplinarne probleme mora istraživati grupa znanstvenika, od kojih je svaki specijaliziran za svoje područje znanosti, ali se također snalazi i u drugim područjima potrebnima za razumijevanje određenog problema.

U ljetu 1940. godine Wiener se okrenuo razvoju računala (strojeva za računanje) u svrhu rješavanja parcijalnih deferencijalnih jednadžbi. U to vrijeme postojao je tzv. *differential analyzer* Vannevara Busha koji je lako rješavao obične diferencijalne jednadžbe, ali s parcijalnima je imao poteškoća. Wiener je zaključio da brzina elementarnih procesa mora biti maksimalna moguća te da svaki od tih elementarnih procesa mora dati dovoljno precizan rezultat kako se velikim brojem ponavljanja ne bi stvarala kumulativna pogreška koja bi utjecala na efikasnost stroja. Tada je predložio slijedeće zahtjeve:

1. Središnja jedinica za zbrajanje i množenje bi trebala biti digitalna, za razliku od Bushevog *differential analyzera* koji je radio na temelju mjerenja (odnosno analogno).

2. Mehanizmi, koji su zapravo sklopni uređaji, trebali bi biti izgrađeni od električnih cijevi, umjesto od zupčanika i ostalih mehaničkih sklopova.

3. Središnja jedinica za zbrajanje i množenje trebala bi koristiti binarni sustav umjesto dekadskog.

4. Cijeli niz operacija trebao bi se odviti na računalu bez ljudske intervencije, odnosno računalo bi trebalo samostalno raditi od trenutka kada se unesu podatci do ispisa finalnog rezultata.

5. Računalo bi trebalo imati i jedinicu za pohranu podataka i to takvu da podatke brzo sprema, čuva ih do brisanja, brzo ih čita, brzo ih briše te je nakon brisanja odmah spremno za pohranu novih podataka.

Ovi zahtjevi odmah su poslani Vannevaru Bushu zbog moguće primjene u ratu, no ipak se nije odmah počelo raditi na ostvarenju ovakvog stroja. Bez obzira na to, ovo su ideje koje su prethodile modernom računalu. Wiener je ove ideje povezivao s proučavanjem živčanog sustava.

Još jedan projekt na kojemu je Wiener radio, s Julianom H. Bigelowom, bio je poboljšanje protuavionske artiljerije. Istraživali su teoriju predviđanja, odnosno, željeli su predvidjeti gdje će se avion nalaziti u trenutku kada ga projektil treba pogoditi i pokušali izgraditi uređaj koji bi koristio takvu teoriju. Wiener također proučava pojam pod nazivom *feedback* (sustav povratnih informacija). Kada želimo da ljudski pokret prati određeni obrazac, uzmemo razliku između stvarnog kretanja i traženog obrasca i iskoristimo taj podatak kako bismo regulirali kretanje pomičnog dijela, odnosno, kako bi se taj dio kretao točnije po traženom obrascu. U nekim slučajevima ovaj mehanizam pomaže, ali postoje i slučajevi kada je feedback preoštar te premaši vrijednosti koje nalaže obrazac te tada dolazi do nekontroliranih oscilacija i kvara mehanizma. Dakle, Wiener i u ovom kontekstu radi na nečemu što zamjenjuje ljudskog agenta te oponaša ljudsko ponašanje ili mišljenje.

1.1 Kibernetika

Zamislimo da želimo podići olovku. Kako bismo ovo učinili, moramo pokrenuti određene mišiće — ne znamo svjesno koje, a ipak prolaskom vremena pomičemo ruku sve bliže olovci. Kako bismo ovo učinili, mora postojati izvještaj našem mozgu o tome koliko još trebamo pomaknuti ruku da bismo došli do olovke. Ovaj podatak, naravno, dolazi od naših osjetila — u konkretnom slučaju, vida. Rosenblueth upoznaje Wienera i Bigelowa s činjenicom da postoji stanje zvano *purpose tremor*, izazvano oštećenjima moždanog debla, u kojem ljudska ruka ne može podići olovku već oscilira oko nje. Iz ovog primjera Wiener zaključuje kako centralni živčani sustav nije samodostatan već radi na temelju kružnih

procesa koji se odvijaju od živčanog sustava, preko mišića, zatim organa osjeta da bi se opet vratili u živčani sustav. Ovo je toliko zaintrigiralo Rosenbluetha, Wienera i Bigelowa da su 1943. godine o tome napisali članak *Behaviour, Purpose, and Teleology*.

Wiener i Bigelow zaključuju da su problemi kontrole i komunikacije nerazdvojni te su centrirani oko pojma *poruke*, bez obzira na način prenošenja te poruke (elektronički, mehanički ili živčani). Definiraju poruku kao konačan i kontinuiran niz mjerljivih događaja u vremenu — ono što su statističari nazvali *vremenskim nizom*. Nakon ovog zaključka, mnogi problemi inženjerskog dizajna mogli su se riješiti, te se ovo područje počelo smatrati više znanstvenim područjem nego područjem umjetnosti (kakvim se smatralo do tada). Wiener, Bigelow i Rosenblueth shvatili su da se mnogo problema inženjerskog dizajna može riješiti na isti način te su ih počeli rješavati: na primjer, dizajn filtera zvučnih valova (kako bi se riješio problem pozadinskog šuma). Ovime su komunikacijski inženjerski dizajn uklopili u paradigmu statističke mehanike.

Kako bi riješili problem prenošenja poruka, morali su razviti statističku teoriju količine podataka. Definirali su jediničnu količinu podataka kao jednu odluku između dviju jednako vjerojatnih alternativa. Ovu ideju, u isto vrijeme, imalo je nekoliko znanstvenika, među njima je bio i statističar R. A. Fisher te Claude Shannon iz *Bell Telephone Laboratories*.

Dakle, nekoliko znanstvenika već je bilo svjesno problema koji se tiču komunikacije, kontrole i statističke mehanike, i njihovih međuodnosa, bilo u računalu ili u živom biću. S druge strane nije bilo zajedničke, ujedinjene literature ni terminologije koja bi se ticala ovih problema. Tada su Wiener i Rosenblueth odlučili dati ime polju kontrole i komunikacijske teorije i na taj način ujediniti terminologiju i zanstvenike koji su radili u tom polju. Ime koje su tom polju odlučili dati je *kibernetika* (od grčke riječi za kormilara). Iako se ovo imenovanje dogodilo tek 1947. godine, razvoj polja započeo je mnogo ranije. Od 1942. godine, kibernetika se razvija na više fronti. Rosenblueth komentira ideje iz članka, koji su napisali Bigelow, Wiener i sam Rosenblueth, na sastanku u New Yorku pod pokroviteljstvom zaklade *Joshiah Macy*. Na istom sastanku bio je i Warren McCulloch, koji je bio zainteresiran za proučavanje organizacije korteksa mozga, te je već otprije bio u kontaktu s Rosenbluethom i Wienerom.

Pod McCullochovim krilom našao se i Walter Pitts, koji je u područje kibernetike prešao iz područja matematičke logike. McCulloch i Pitts zajedno su radili na temi nervnih vlakana i sinapsi te njihovom međusobnom povezivanju u sustave. 1943. godine napisali su članak *A Logical Calculus of Ideas Immanent in Nervous Activity*. Cilj im je bio stvoriti stroj koji bi mogao implementirati logičko zaključivanje, inspiriran biološkim neuronima. U svom radu također daju ideju *umjetne neuronske mreže* te način da se logički predikat realizira na takvoj mreži. Neurone su podijelili u dvije grupe: *ulazne* i *izlazne* (skriveni sloj tada nije postojao, pojavio se tek 1970-ih uz pojam backpropagationa). Svaki neuron bi u svakom trenutku bio u jednom od dva moguća stanja: stanju u kojem „okida" (šalje impuls) ili stanju u kojem miruje. Za svaki neuron i definiraju logički predikat $N_i(t)$, koji

je istinit ukoliko u trenutku t neuron i „okida". Tada ekvivalenciju oblika $N_i(t) \equiv B$, gdje je B konjunkcija okidanja ulaznih neurona u prethodnom trenutku, a i nije ulazni neuron, nazivamo *rješenjem* neuronske mreže. Ovakva ekvivalencija je zadovoljiva u neuronskoj mreži ako i samo ako ju mreža može izračunati. Sve formule za koje bi postojala neuronska mreža koja ih može izračunati nazvali su *temporal propositional expression* (TPE). Osim definiranja umjetne neuronske mreže, jedan od najvažnijih rezultata njihovog članka bio je da se bilo koji TPE može izračunati umjetnom neuronskom mrežom. Neke od ideja, nenamjerno su posudili od Shannona (primjena tehnika Boolove algebre na promatranje sklopnih sistema u električnom inženjerstvu) i Turinga (korištenje vremena kao parametra, razmatranje postojanja ciklusa u mrežama, sinaptička i druga odgađanja). Pitts se u jesen 1943. pridružio Wieneru i Rosenbluethu na *Massachusetts Institute of Technology*, kako bi produbio svoje matematičko znanje i proučavao kibernetiku. Bio je zainteresiran za (tada moderne) *vakuumske cijevi*, koje mu je Wiener predložio kao najbolje sredstvo za ostvarenje neuronskih sklopova i sustava. Marvin Minsky je u to vrijeme (1954. godine) završavao studij na Princetonu te je naslov njegovog završnog rada bio *Neural Nets and the Brain Model Problem*. U ovom radu Minsky je prvi sakupio sve ideje, rezultate i teoreme koji se tiču neuronskih mreža te je istaknuo nekoliko tehničkih problema. 1951. godine Minsky je izgradio stroj (financirao ga je *Air Force Office of Scientific Research*), zvan SNARC (*Stochastic Neural Analog Reinforcement Calculator*), koji implementira neuronske mreže. Ovo je bila prva veća implementacija neuronskih mreža na računalu.

Tada je postalo jasno da računalo kakvo su zamišljali mora predstavljati gotovo idealnu sliku biološkog neuronskog sustava. Postoji način aktivacije neurona koji se zove „sve ili ništa", koji je gotovo analogan izboru jednog bita. *Sinapsa* je u tom kontekstu mehanizam koji određuje hoće li se slijedeći neuron aktivirati s obzirom na kombinaciju izlaza iz prethodnih neurona. Također, interpretacija varijabilnosti i prirode sjećanja u živim bićima ima analogni problem konstrukcije umjetnih sjećanja za stroj.

Pokazalo se da je konstrukcija računala ipak bila od velike važnosti za rat te se na raznim mjestima (Harvard, Probni teren u Aberdeenu, Pensilvanijsko sveučilište) započela konstrukcija takvih strojeva. Proces je bio spor, ali sa svakom iteracijom ti strojevi bili su sve bliže pravilima koje je Wiener poslao Bushu 1940. U to doba Wiener i Rosenblueth imali su priliku proširiti svoje ideje te su razgovarali s Aikenom (Harvard), Johnom von Neumannom (*Institute for Advanced Study*) i Hermanom Goldstineom (Pensilvanijsko sveučilište).

John von Neumann i Wiener odlučili su održati zajednički sastanak svih zainteresiranih za kibernetiku. Sastanak se održao na Princetonu, kasne zime 1943–1944. Na njemu su sudjelovali razni računalni dizajneri (Goldstine), fiziolozi (McCulloch, Lorente de No) i matematičari (von Neumann, Pitts, Wiener). Svatko je predstavio svoju stranu priče, problema i metoda te su, na kraju sastanka, došli do zaključka da im je većina ideja zajednička i da žele pokušati raditi na tom novom polju zajedno, odnosno stvoriti zajednički

vokabular.

Rosenblueth nije prisustvovao ovom sastanku jer je baš u to vrijeme prihvatio pozivnicu za rad kao voditelj laboratorija fiziologije na (institutu)*Instituto Nacional de Cardiologia* u Meksiku. U proljeće 1945. pridružuje mu se i Wiener, koji u to vrijeme dobiva pozivnicu Meksičkog matematičkog društva za sudjelovanje na sastanku u Guadalajari. Rosenblueth i Wiener proučavaju rad srca te 1946. objavljuju rad *The Mathematical Formulation of the Problem of Conduction of Impulses in a Network of Connected Excitable Elements, Specifically in Cardiac Muscle*,a glavne ideje predstavljaju na sastanku u Guadalajari. Walter Pitts koristi statističke tehnike iz njihovog rada i proširuje ih na djelovanje na neuronskim mrežama.

U proljeće 1946. godine formira se grupa znanstvenika koji se sastaju u New Yorku i raspravljaju na temu feedbacka. Sudionici su uglavnom isti kao na sastanku na Princetonu 1944., ali McCulloch i Fremonth-Smith uvidjeli su da je tamo potrebno i nekoliko psihologa, sociologa i antropologa. Neki od psihologa bili su Kluver (Sveučilište u Chicagu), Kurt Lewin (*Massachusetts Institute of Technology*) i M. Ericson. Oni su bili tamo kako bi pomogli u shvaćanju povezanosti živčanog sustava i samog uma, odnosno, kako čovjek prepoznaje da je nešto kvadrat, bez obzira na poziciju, veličinu i orijentaciju — takozvani *gestalt*. Sociolozi i antropolozi imali su ulogu u komunikaciji i povezivanju različitih jedinki, odnosno, mehanizmima prenošenja informacija i organizaciji sustava jedinki. Neki od njih bili su Bateson, Margaret Mead, Schneirla i Morgenstern.

Ljeti 1946. Wiener opet odlazi u Mexico raditi s Rosenbluethom, a na jesen predstavljaju svoje rezultate. Nakon toga su isplanirali petogodišnju znanstvenu suradnju, uz potporu instituta u Meksiku. Unutar tih 5 godina predviđeno je da se na institutu dalje eksperimentalno i teorijski istražuje područje kibernetike, alternirajući svake godine. U međuvremenu, bilo je potrebno smisliti i plan kako educirati ljude koji su zainteresirani za područje kibernetike.

1.2 Prvi koncepti umjetne inteligencije

McCulloch je dobio zadatak da omogući slijepcima čitanje tiskanog teksta po zvuku. Ideja je bila da zvuk ostane približno isti za isto slovo, bez obzira na veličinu fonta. Ovo je analogon problema percepcije forme, gestalta. Na jednom od prijašnjih Macyjevih sastanaka, Wiener je predstavio dijagram stroja koji je trebao imati sličnu ulogu, koji je odmah privukao pozornost dr. von Bonina, koji je pomislio kako je to dijagram četvrtog sloja vizualnog korteksa u mozgu. McCulloch je tada dobio ideju koju je, uz pomoć Pittsa, razvio i predstavio u proljeće 1947. godine na sastanku pod pokroviteljstvom zaklade Macy i još jednom sastanku na *New York Academy of Sciences*. McCullochova teorija uključuje proces skeniranja koji se odvija u ciklusima, odnosno postoji vremenski period potreban za jedan ciklus. Taj vremenski period analogan je televizijskom *time of sweep* ili takozvanom

α-ritmu mozga (za koji se tada mislilo da nastaje vizualnim podražajima). Za skeniranje je koristio tri horizontalne linije postavljene jednu iznad druge u visini fonta koji treba pročitati. Svaka od te tri linije proizvodila bi drugačiji ton (recimo najniža najniži ton, a najviša najviši), a ton bi se proizvodio samo u trenutku kada stroj pri čitanju prelazi preko linije u slovu. Na primjer slovo „E" bilo pročitano tako da bi sve tri linije ispustile svoj ton u dugom intervalu, dok bi recimo slovo „F" proizvelo dugi signal na gornje dvije linije i kratki signal na donjoj.

Dakle, McCulloch je iskoristio ideju iz komunikacijskog inženjerstva kako bi se zamijenilo izgubljeno osjetilo, odnosno, kako bi se omogućilo slijepcima da „vide" što piše na stranici teksta. Wiener je planirao napraviti sličnu stvar s umjetnim udovima. Gubitak uda podrazumijeva ne samo gubitak mehaničkog produžetka i mišićnog tkiva već i gubitak osjeta dodira na tom području. U to vrijeme već se radilo na prva dva gubitka (mehanika i mišići). Postojala je obična drvena noga, za koju nije bilo potrebno osjetiti gdje se nalazi u prostoru i slično, međutim postojali su i artikulirani udovi, s mobilnim koljenom i gležnjem, koje bi korisnik prilikom hodanja izbacivao naprijed svojim preostalim mišićima. Takvi umjetni udovi korisniku su stvarali nelagodu zbog nesigurnosti kretanja po neravnom terenu. Wiener je želio ovo olakšati instaliranjem raznih mjerača pritiska na potplat i zglobove umjetnog uda, koji bi onda vibracijom javljali korisniku izmjerene podatke, međutim, njegovi pokušaji nisu urodili plodom kod ulagača.

Wiener zaključuje da je došlo vrijeme druge industrijske revolucije, kada se mogu konstruirati umjetni strojevi gotovo bilo kakve složenosti: automatizirane tvornice već su moguće, samo treba uložiti truda i vremena. Kao što je u prvoj industrijskoj revoluciji parni stroj zamijenio čovjeka pri fizičkom radu i mnogim ljudima oduzeo posao, tako ova nova revolucija zamjenjivim čini ljudski mozak. Naravno, kao što su najbolji drvodjelje, mehaničari i krojači preživjeli prvu industrijsku revoluciju, tako će i ovu novu preživjeti najbolji znanstvenici i administratori — međutim, ljudi prosječnih ili manjih mogućnosti neće moći naći posao. Također, Wiener zna da se ovo novo znanje može upotrijebiti za dobro i za zlo te pokušava usmjeriti znanost više prema psihologiji i sociologiji, kako bi se što manje koristila u ratne svrhe. Dakle, već tada postavljalo se pitanje moralnosti vezano uz umjetnu inteligenciju (kibernetiku).

Kako je 17. i rano 18. stoljeće bilo doba satova, a kasno 18. i 19. stoljeće predstavljaju doba parnog stroja, tako je 20. stoljeće doba komunikacije i kontrole — tako bar kaže Norbert Wiener. U električnom inženjerstvu u 19. stoljeću postoji sraz između takozvane *tehnike slabe struje* i *tehnike jake struje*, koje Wiener razlikuje kao električno i komunikacijsko inženjerstvo. Po njemu, ovaj sraz razdvaja prošlo doba od njegovog te kao glavnu razliku, koja razdvaja komunikacijsko inženjerstvo od električnog, navodi da je glavni interes komunikacijskog inženjerstva točan prijenos *signala* umjesto ekonomije električne energije. Ovaj signal može biti lupkanje tipke koje se prenosi telegrafom, zvuk koji se prenosi telefonom ili kut pod kojim je okrenuto kormilo broda, odnosno bilo koja

informacija koja se prenosi na drugo mjesto. Tada se razvijaju i prve ideje za razvoj „ultra brzog računala". Sve od Dedala ili Herona iz Aleksandrije ljude je intrigirala sposobnost izrade stroja koji oponaša živući organizam. U doba magije, postoji bizaran koncept *Golema*, bića od gline. U doba Newtona imamo glazbenu kutiju sa satnim mehanizmom na kojoj se okreće mala figurica. U 19. stoljeću toplinski stroj, koji umjesto glikogena koristi zapaljivo gorivo. U 20. stoljeću to su automatska vrata ili stroj koji rješava diferencijalne jednadžbe. Grana komunikacijskog inženjerstva u dvadesetom se stoljeću bavila automatima; kako mehaničkim tako i prirodnim. Naime, neki su znanstvenici životinje smatrali automatima, kako se ne bi narušila kršćanska dogma da životinje nemaju dušu. Glavni interesi ove grane su poruka, količina ometanja ili takozvana buka, količina informacija, tehnika kodiranja i slično. U takvoj teoriji postoje automati koji su uključeni u vanjski svijet, ne samo svojim metabolizmom ili protokom enerije već i razmjenom informacija s vanjskim svijetom. „Organi" uz pomoć kojih automat prima informacije napravljeni su po uzoru na ljudske i životinjske osjetilne organe. Sastoje se od fotoelektričnih ćelija i drugih *receptora* za svjetlo, radarskih sustava, termometara, mikrofona i tako dalje. Što se tiče *efektora*, odnosno fizičkih naprava za djelovanje u stvarnom svijetu, oni mogu biti električni motori, elektromagnetni, grijuće zavojnice ili drugi instrumenti različitih vrsta. Između receptora i efekora nalazi se srednji sloj elemenata čija je funkcija organizacija ulaznih podataka tako da budu u pravom obliku za upotrebu u efektorima. Te ulazne informacije često sadrže poruku o tome što efektori trebaju raditi. Ovaj središnji sloj elemenata odgovara živčanom sustavu ili kinestetičkim i drugim receptorima u ljudskom tijelu, koji su zaslužni za određivanje pozicije određenih dijelova tijela ili slične nesvjesne akcije. Štoviše, automat dobivene informacije ne mora iskoristiti odmah, već ih može spremiti kako bi ih koristio kasnije. Ovo je analogon *pamćenja*. Konačno, dok god automat radi, njegov način rada podložan je promjeni s obzirom na podatke koje je tijekom tog vremena promio receptorima. Ovaj proces mogli bismo nazvati *učenjem*. U 20. stoljeću neki od ovakvih automata bili su termostati, žiroskopi u brodovima, samonavođeni projektili, protuzračni sustavi te tadašnja ultra-brza računala.

1.3 Zapis i količina informacija

Postoji velika klasa fenomena koje zapisujemo numeričkim vrijednostima ili nizovima numeričkih vrijednosti raspoređenim u vremenu. Na primjer, temperatura u danu određena termometrom (i zapisana, na primjer, svakih 5 minuta) čini jedan ovakav niz — *vremenski niz*. Za izračune vezane uz nizove koji se sporo mijenjaju možemo koristiti ručno računanje ili samo jednostavna pomagala. Međutim, postoje vremenski nizovi koji se vrlo brzo i drastično mijenjaju — na primjer, napon u telefonskoj liniji. Kako bi ispravno radio, u telefonu se operacije moraju odvijati vrlo brzo, odnosno, za svaku promjenu ulaznih podataka (napona) stroj mora nešto izračunati i vratiti prije no što se ulazni podatci ponovo

promijene. Uređaji poput telefona, filtera zvučnih valova, automatskih uređaja za kodiranje zvuka, ... zapravo vrlo brze aritmetičke naprave. Sve što im je potrebno za rad već je unaprijed ubačeno u njih, kako bi se izbjegla ljudska greška i višestruko ubrzao cijeli postupak. Svi spomenuti uređaji služe za snimanje, čuvanje, prenošenje i/ili korištenje informacija. Jedan od najosnovnijih oblika informacije sastoji se u zapisanom izboru između dvije jednako vjerojatne jednostavne opcije od kojih se bar jedna mora dogoditi. Jedan ovakav izbor zvat ćemo *odluka*. Ako se sada pitamo koliko je to informacija, odnosno tražimo *količinu informacija* u smislu broja između A i B, koji može uniformnom vjerojatnošću ležati bilo gdje u ovom intervalu, vidjet ćemo da ako stavimo $A = 0$ i $B = 1$, a traženu vrijednost zapišemo u bazi 2 kao broj $0.a_1a_2a_3 \cdots a_n \cdots$, tada je broj odluka i, posljedično, količina informacija beskonačna. Ovdje je

$$0.a_1a_2a_3 \cdots a_n \cdots = \frac{1}{2}a_1 + \frac{1}{2^2}a_2 + \frac{1}{2^3}a_3 + \cdots + \frac{1}{2^n}a_n + \cdots$$

Međutim, nijedno mjerenje koje izvršimo nikada nije savršeno. Ako mjerenje ima uniformno distribuiranu pogrešku koja je (ograničena s) $0.b_1b_2b_3 \ldots b_n \ldots$, gdje je b_k prva jedinica, vidjet ćemo da su sve odluke od a_1 do a_{k-1} i (možda) a_k značajne, a sve ostale nisu značajne. Tada je broj učinjenih odluka, odnosno količina informacija, približno k.

1.4 *Feedback* i oscilacija

Kako bismo efektivno djelovali na vanjski svijet, potrebni su nam dobri efektori, ali ne samo oni. Potrebno je da pravilno nadgledamo performanse spomenutih efektora i sakupljene informacije pošaljemo centralnom živčanom sustavu te da se te informacije pravilno kombiniraju s informacijama dobivenim od osjetila kako bi proizvele prave izlazne podatke za efektore. Postoje određena stanja (bolesti) u kojima čovjek ne može podići olovku sa stola jer mu ruka počne nekontrolirano oscilirati oko olovke umjesto da je podigne. Slično vrijedi i za mehaničke sustave. Na primjer, za sustav semafora postoji kontrolna ploča na kojoj stoje sva trenutna stanja efektora (semafora), kako bi čovjek koji njima upravlja bolje znao koje je stanje, odnosno, kako se ne bi zabunio gledajući u stvarne efektore (semafore). Ovo je ekvivalent ponavljanja zapovijedi u vojsci ili mornarici. Primijetimo da ovdje postoji ljudski faktor, odnosno korak u prijenosu informacija koje izvodi osoba, a ne stroj. Takvo vraćanje informacija zovemo *feedback* — lanac povratnih informacija. Naravno, postoje i lanci povratnih informacija u kojima ne sudjeluje čovjek (već samo stroj). Na primjer, uzmimo termostat. Termostat ima ugrađeni termometar koji mjeri sobnu temperaturu određenom frekvencijom. Informaciju o očitanoj temperaturi šalje u svoj centralni sustav te centralni sustav provjerava je li trenutna temperatura viša ili niža od zadane; ako je niža , termostat nastavi grijati, a ako je viša, on ugasi grijanje. Primijetimo da su povratne

informacije uglavnom negativnog učinka jer služe za stabilizaciju sustava — usporavanje, odnosno zaustavljanje nekog procesa.

Postoje sustavi u kojima jedan feedback nije dovoljan, odnosno, potreban nam je niz feedbacka da bismo postigli željeni rezultat. Tako, u ljudskom tijelu, postoji *voljni* (svjesni) feedback (na primjer onaj koji nam govori koliko nam još nedostaje da sa stola uzmemo olovku) kojem pomažu drugi sustavi feedbacka. Ove ostale sustave feedbacka zvat ćemo, jednim imenom, *posturalni feedback*. Sustav feedbacka koji je podložan nekontroliranim oscilacijama, recimo u slučaju cerebralne ozljede, je sustav voljnog feedbacka, jer tremor (oscilacija) se ne događa dok god čovjek ne odluči napraviti svjesnu akciju. Nasuprot tome, Prakinsonova bolest je bolest posturalnog sustava feedbacka, jer se događa u stanju mirovanja (bez prethodne voljne akcije).

Postojanje periodičkog nesinusoidalnog osciliranja sugerira da sustav nije linearan, barem ne u varijabli koju proučavamo. Druga velika razlika između linearnih i nelinearnih oscilirajućih sustava je amplituda. U linearnim sustavima amplituda osciliranja ne ovisi o frekvenciji, dok kod nelinearnih sustava postoji jedna amplituda ili diskretan skup amplituda za koje će sustav oscilirati određenom frekvencijom te diskretan skup frekvencija za koje će sustav oscilirati. Postoje i sustavi koji su svojevrsni most između linearnih i nelinearnih sustava — varijable u kojima se sustav razlikuje od linearnog mijenjaju se toliko sporo da bi se, tijekom određenog razdoblja oscilacije, mogle smatrati konstantama. Ovakve sustave riješavamo svođenjem na sustav nehomogenih linearnih jednadžbi.

Još jedna zanimljiva varijanta feedbacka slična je ljudskom ponašanju prilikom vožnje na zaleđenoj cesti. Naše ponašanje uvelike ovisi o znanju da je cesta zaleđena ili skliska. Sve akcije obavljat ćemo nizom kratkih i blagih akcija koje sustav neće izbaciti iz ravnoteže (nećemo proklizati), a dobit ćemo povratnu informaciju o jačini akcije koja je potrebna da automobil prokliže te, sukladno tome, prilagoditi svoje ponašanje. Ovu metodu kontrole nazivamo *kontrola informativnim feedbackom*.

Još jedan sustav feedbacka kod čovjeka (ili životinje) je takozvani *homeostatski sustav feedbacka* koji služi za održavanje razina temperature, krvnog tlaka, otkucaja srca, imuniteta te raznih tvari potrebnih tijelu za normalno (zdravo) funkcioniranje. Ovaj sustav je općenito sporiji od voljnog i posturalnog feedbacka.

1.5 Računala

Računski strojevi su, u suštini, strojevi za zapisivanje brojeva, računanje njima i davanje rezultata u numeričkom obliku. Veliki dio njihove cijene, u novcu i konstrukciji, sastoji se u jasnom i točnom prikazu brojeva. Kao najjednostavniji način čini se prikaz na uniformnoj skali, s pokazivačem koji se po toj skali pomiče. Ako želimo zapisati broj uz preciznost $\frac{1}{n}$, podijelit ćemo ovu skalu na n područja. Tada će za količinu informacija $\log_2 n$ cijena pohrane biti oblika $a \cdot n$, gdje je a konstanta (ili skoro svuda konstanta). Pošto vrijedi da

ako točno odredimo $n-1$ područje na skali, bit će automatski određeno i zadnje područje, cijena spremanja količine informacija I bit će:

$$(2^I - 1)a.$$

Ako ovu količinu informacija podijelimo na dvije skale od kojih svaka ima umanjenu točnost, cijena će biti:

$$2(2^{\frac{I}{2}} - 1)a.$$

Kada bismo informacije podijelili na N skala, cijena bi bila otprilike:

$$N(2^{\frac{I}{N}} - 1)a.$$

Ovo će biti minimalno kada vrijedi

$$2^{\frac{I}{N}} - 1 = \frac{I}{N} 2^{\frac{I}{N}} \log_2 2$$

ili ako stavimo $\frac{I}{N} log 2 = x$, kada je $x = \frac{e^x - 1}{e^x} = 1 - e^{-x}$. To se događa samo u slučaju da je $x = 0$ iz čega slijedi $N = \infty$. Dakle, N treba biti što veći kako bi cijena pohrane podataka bila što manja. Također, znamo da $2^{\frac{I}{N}}$ mora biti cijeli broj te da 1 nije smislena vrijednost. Dakle, najbolja vrijednost za $2^{\frac{I}{N}}$ je 2 te ćemo broj zapisati na nekoliko nezavisnih skala od kojih se svaka sastoji od dva jednaka dijela. Drugim riječima, broj ćemo zapisati u binarnom sustavu, odnosno, u obliku:

$$v = v_0 + \frac{1}{2} v_1 + \frac{1}{2^2} v_2 + \cdots + \frac{1}{2^n} v_n + \ldots$$

gdje svaki v_n ima vrijednost 1 ili 0.

U ovom vremenskom periodu postoje dva tipa računskih strojeva. Prvog tipa je Bushev *differential analyzer* (*Journal of the Franklin Institute*, razni članci, od 1930. na dalje) i slični strojevi — *analogni* strojevi — u kojima su podaci prikazani mjerama na neprekidnoj skali, te je točnost stroja predodređena njegovom izradom. Drugog tipa su stolni strojevi za računanje — *digitalni* strojevi — na kojima su podaci prikazani skupom odabira između nekoliko slučajeva, a točnost im je određena brojem slučajeva i njihovom međusobnom razlikom te danim brojem odabira. U smislu točnosti, kao što smo upravo pokazali, bolji su digitalni strojevi, pogotovo oni kod kojih za svaki odabir imamo dva slučaja (biramo između dvije opcije), odnosno oni koji rade u binarnom sustavu.

Računalne strojeve općenito koristimo jer su strojne metode brže od ručnih metoda. Kad kombiniramo nekoliko metoda računanja, ona najsporija dat će nam red veličine utrošenog vremena. Zato je poželjno, gdje je to moguće, izbaciti ljudski element u računanju. Ljudsko djelovanje bit će potrebno samo na početku i na samom kraju izračunavanja. Pošto je

čovjeku prirodno koristiti dekadski sustav, a u računalu želimo binarni, na samom početku izračunavanja, nakon što čovjek unese željene podatke u računalo (u dekadskom sustavu), ti podaci bit će prevedeni u binarni sustav te će se svi međukoraci odvijati u binarnom sustavu, a krajnji rezultat opet će biti preveden u dekadski sustav.

To znači da, na početku, u računalo moraju biti uneseni svi numerički podaci i pravila za njihovo međusobno kombiniranje u obliku instrukcija koje pokrivaju svaku moguću situaciju tijekom računanja. Dakle, računalo mora biti logički i aritmetički stroj koji kombinira slučajeve u skladu s nekim sustavnim algoritmom — npr. Booleovom algebrom. Svi podaci koje unosimo u računalo su u obliku niza odabira između dvije alternative, a sve operacije na ovim podacima svode se na nove odabire ovisne o starom skupu odabira. Zaključujemo da se struktura računala treba sastojati od prekidača od kojih svaki može biti u dva stanja, na primjer 1 i 0 ili „da" i „ne". Sve operacije na takvim podacima mogu se svesti na tri osnovne operacije Booleove algebre, a to su negacija, logičko zbrajanje (veznik „ili") i logičko množenje (veznik „i").

Životinjski (i ljudski) živčani sustav sadrži elemente koji djeluju poput ovakvih prekidača: to su neuroni (živčane stanice). Pod normalnim uvjetima, ove stanice funkcioniraju po principu „sve ili ništa", odnosno, ili se odmaraju (polarizirani su) ili su aktivni (depolarizirani su). Još jedna vrlo bitna funkcija živčanog sustava je *memorija* (pamćenje), odnosno, mogućnost čuvanja rezultata prošlih operacija za korištenje u budućnosti. U računalu želimo više tipova memorije. Prvo, u računalu trebamo memoriju za provođenje trenutne operacije. Takva memorija treba brzo pisati, čitati i brisati podatke. S druge strane, trebamo memoriju za trajno čuvanje podataka. Ova dugotrajna memorija utječe na daljnje ponašanje stroja (kao što ljudsko dugoročno pamćenje utječe na odluke čovjeka u budućnosti). Za razliku od ljudskog mozga, računalo je predviđeno za mnogo uzastopnih pokretanja između kojih i ta dugoročna memorija može biti izbrisana.

Poglavlje 2

Logička umjetna inteligencija

Većina informacija u ovom poglavlju preuzeta je iz [3].

1954. godine vojska SAD-a željela je program koji bi automatski prevodio ruske dokumente i znanstvene radove. Znanstvenici su potcijenili složenost jezika, odnosno lingvistike i značenja riječi. Rezultati nisu bili najbolji, — poznat je prijevod s engleskog na ruski pa natrag na engleski rečenice „*the spirit was willing but the flesh was weak*" koja se time pretvorila u „*the vodka was good but the meat was rotten*". Zbog loših rezultata došlo je do povlačenja financiranja (*National Research Council* formira *Automatic Language Processing Advisory Committee*, ALPAC, te od 1966. povlače sve financije za projekte strojnog prevođenja).

31. kolovoza 1955. godine John McCarthy, Marvin Minsky, Nathaniel Rochester i Claude Shannon daju prijedlog za održavanje *Darthmouth Summer Research Project on Artificial Inteligence* [8] — jednog od najvažnijih događaja u ranoj povijesti umjetne inteligencije. U njemu su sudjelovali John McCarthy, Marvin Minsky, Julian Bigelow, Donald MacKay, Ray Solomonoff, John Holland, Claude Shannon, Nathaniel Rochester, Oliver Selfridge, Allen Newell i Herbert Simon, a održao se 1956. godine. Prijedlog je bio:

> *Proceed on the basis of the conjecture that every aspect of learning or any other feature of inteligence can in principle be so precisely described that a machine can be made to simulate it.* (Svaki aspekt učenja i inteligencije može se toliko precizno opisati da se može napraviti stroj koji će ga simulirati.)

Time klasična UI postaje *logička (simbolička) UI* te će se značajne promjene dogoditi tek u 21. stoljeću pojavom dubokog učenja.

Do ovoga vode i rezultati koje su davali sustavi temeljeni na logičkom zaključivanju (dva najpoznatija bili su *Logic Theorist* i *General Problem Solver* koje su razvili Herbert Simon, Cliff Shaw i Allen Newell) gdje su neuronske mreže zakazale, odnosno, neuronske mreže su se koristile u druge svrhe koje se tada nisu činile inteligentnima. Naime, shvaćanje

inteligencije bilo je drugačije (smatralo se, primjerice, da je inteligentnije računalo koje igra šah nego ono koje razaznaje što se nalazi na fotografiji). Tek 1980-ih Hans Moravec predstavlja hijerarhiju inteligencije gdje kaže da je simboličko razmišljanje rijetka i poželjna pojava kod čovjeka, a računalima je prirodno, dok nasuprot, računalima je teško shvatiti neke čovjeku prirodne i jednostavne zadatke.

2.1 Simboli i inteligencija

Koncept umjetne inteligencije, prije svog znanstvenog ostvarenja, stoljećima se razvijao kroz umjetnost, literaturu i filozofiju. Razlikujemo dvije vrlo poznate i dobro razvijene teme u djelima znanstvene fantastike. S jedne strane su razna bića, čudovišta i androidi, koji nalikuju na čovjeka, koje je čovjek stvorio (Hefest, dr. Frankenstein). S druge strane imamo razne mehaničke „robote" (*radnike*), koji umjesto živčanog sustava imaju žice, umjesto nogu kotačiće i slično. Dok se prva, čudovišna, tema uglavnom poziva na tajnovitost i crnu magiju, roboti su obično ekstrapolacije najnovijeg tehnološkog čuda u industriji. Rani dizajni robota temeljili su se na zamršenim mehanizmima i zupčanicima koji su svojom pojavom očarali Europu u doba kada su satni mehanizmi bili novost. Nakon toga pojavili su se parni strojevi, hidrauličko upravljanje, telefonske razvodne ploče i drugi izumi koji su dalje razbuktali ljudsku maštu. Suvremena umjetna inteligencija temelji se, naravno, na sofisticiranim programabilnim sustavima, ali sama ideja stvaranja umjetnog inteligentnog bića proizašla je iz ovih koncepata. Zašto onda nitko od uglednih znanstvenika nije pokušao stvoriti čak ni inteligentni satni mehanizam, a kamoli androida? Pravo pitanje nema nikakve veze s naprednim tehnologijama, već s dubokim teorijskim pretpostavkama. Prema središnjoj tradiciji zapadne filozofije, razmišljanje (intelekt) je u osnovi racionalna manipulacija mentalnim simbolima (idejama). Međutim, satovi i razvodne ploče uopće ne rade ništa nalik na racionalnu manipulaciju simbolima. S druge strane, računala mogu manipulirati proizvoljnim „objektima" na bilo koji određeni način — pa da bismo dobili stroj koji razmišlja, očito trebamo postići da ti objekti budu simboli, a manipulacije koje računalo obavlja trebaju biti racionalne. Drugim riječima, UI je nova i drugačija jer, za razliku od dotadašnjih strojeva, računala zasnovana na umjetnoj inteligenciji zapravo čine nešto vrlo slično onome što bi umovi trebali raditi.

Nazvati nešto simbolom ili manipulacijom znači sasvim apstraktno to okarakterizirati. To ne znači da je karakterizacija nejasna, bezoblična pa niti teško razumljiva, već da su iz nje izostavljeni nebitni detalji. Prema teoriji manipulacije simbolima, inteligencija ovisi samo o organizaciji sustava i njegovom manipuliranju simbolima, koji trebaju zadovoljavati samo apstraktnu specifikaciju. Drugim riječima, razni implementacijski detalji, poput toga je li struktura elektronička ili fiziološka (ili hidraulička ili optička ili bilo kakva druga), potpuno su nevažni. Sukladno tome, suvremena računalna tehnologija relevantna je samo iz ekonomskih razloga: elektronički sklopovi su upravo najjeftiniji način za izgradnju

fleksibilnih sustava za manipulaciju simbolima. Ali pouka ide dublje: ako umjetna inteligencija doista nema puno veze s računalnom tehnologijom, već s apstraktnim načelima mentalne organizacije, tada se razlike između UI, psihologije, pa čak i filozofije uma tope. Za ovo novo „objedinjeno" polje smišljen je naziv *kognitivna znanost*. Nadalje ćemo o umjetnoj inteligenciji razgovarati u ovom smislu (kao o grani kognitivne znanosti). U svojoj knjizi [3] John Haugeland kaže da je zapravo antropomorfna predrasuda, „ljudski šovinizam", ugrađena u sam naš koncept inteligencije. Ovaj bi se koncept, naravno, i dalje mogao primijeniti na razna bića. Poanta je u tome da je to jedini koncept koji imamo — da smo izbjegli svoju pristranost, ne bismo znali o čemu govorimo. Jedini teoretski razlog da se suvremena umjetna inteligencija shvati ozbiljno, moćna je sugestija da naš vlastiti um radi na istom principu. Drugim riječma, zanima nas UI kao dio teorije da su ljudi računala — ono za što smo zapravo zainteresirani su ljudi odnosno ljudsko mišljenje.

2.2 Što je inteligencija?

Turingov test

Kako definirati inteligenciju? Alan Turing 1950. godine objavljuje rad u kojem predstavlja *Turingov test* kojim se određuje može li se računalo smatrati inteligentnim. Cilj mu nije izmjeriti inteligenciju, kao u standardnim testovima inteligencije, već utvrditi postoji li ona uopće. Turinga su nervirali jalovi sporovi oko značenja riječi; mislio je da nikad neće moći saznati ništa zanimljivo o tome što strojevi mogu raditi filozofirajući o tome što mislimo pod pojmovima „misliti" i „inteligentno". Stoga je predložio da zanemarimo verbalno pitanje i usvojimo jednostavan test koji je smislio; tada bismo se mogli koncentrirati na izgradnju i promatranje samih strojeva. Predvidio je da će do 2000. godine računalni sustavi prolaziti skromnu verziju njegova testa i da će druge „definicije" na kraju izgledati glupo.

Turingov test temelji se na igri koja se naziva „igra oponašanja" (*imitation game*), a igraju je trojica stranaca. Dvoje od njih su „svjedoci" i oni su suprotnog spola. Treći igrač, „ispitivač", pokušava pogoditi kojeg je spola koji svjedok, čisto na temelju načina na koji odgovaraju na pitanja. Trik je u tome što jedan svjedok (recimo muškarac) pokušava zavarati ispitivača (sustavnim pretvaranjem da je žena), dok drugi (žena) čini sve što može kako bi pomogao ispitivaču. Ako ispitivač pogodi, žena pobjeđuje, inače pobjeđuje muškarac. Da bi se izbjeglo odavanje bilo kakvih irelevantnih informacija, poput tona glasa, sva pitanja i odgovori prenose se u pisanom obliku. Za sada nisu uključena računala. Turingova je ideja, međutim, bila zamijeniti muškog svjedoka računalom i vidjeti može li protiv prosječnih protivnica žena zavarati prosječnog (ljudskog) ispitivača onoliko često koliko to može prosječni muškarac. Računalo prolazi test ukoliko igra igru ne bitno slabije od prosječnog muškarca. Ali zašto bi tako neobična igra bila test za opću (ljudsku) inteligenciju? Suština testa je *razgovor*: razgovara li računalo poput osobe?

Zašto bi to trebao biti znak opće inteligencije? Što je tako posebno u razgovoru? Turing kaže: „Čini se da je metoda pitanja i odgovora prikladna za predstavljanje gotovo bilo kojeg od polja ljudskog razmišljanja koje želimo uključiti." Odnosno, možemo razgovarati o gotovo bilo čemu. Još važnije, da biste razgovarali izvan najpovršnije razine, morate znati o čemu govorite. Odnosno, samo razumijevanje riječi nije dovoljno, morate razumjeti i temu.

Primjer 2.2.1. Turing ističe (1950, str. 446) koliko je njegova igra oponašanja slična usmenom kvizu te daje primjerak jednog takvog razgovora.

Interrogator: *In the first line of your sonnet which reads „Shall I compare thee to a summer's day," would not „a spring day" do as well or better?*

Witness: *It wouldn't scan.*

Interrogator: *How about „a winter's day"? That would scan all right.*

Witness: *Yes, but nobody wants to be compared to a winter's day.*

Interrogator: *Would you say Mr. Pickwick reminded you of Christmas?*

Witness: *In a way.*

Interrogator: *Yet Christmas is a winter's day, and I do not think Mr. Pickwick would mind the comparison.*

Witness: *I don't think you're serious. By a winter's day one means a typical winter's day, rather than a special one like Christmas.*

Ovaj je sugovornik pokazao ne samo znanje engleskog jezika, već i prolazno razumijevanje poezije, godišnjih doba, osjećaja i tako dalje — sve samo razgovarajući. Isto bi se moglo učiniti za politiku, dobra vina, elektrotehniku, filozofiju i bilo koju drugu temu koje se možemo sjetiti.

Zato je Turingov test tako moćan i uvjerljiv. Prihvaćajući Turingov test, znanstvenici se mogu gotovo prestati baviti neurednim slučajnostima i „računalnom psihologijom" i u potpunosti se koncentrirati na „kognitivne" aspekte problema: koja bi unutarnja struktura i operacije omogućile sustavu da kaže pravu stvar u pravo vrijeme? Ovaj test do danas ostaje jedan od najraširenijih testova (uz mnoge kritike i modifikacije originala) za umjetnu inteligenciju. Turingov rad smatra se jednim od prvih koraka u stvaranju umjetne inteligencije.

Kreativnost i sloboda

Mnogi ljudi posebno sumnjaju u „automatiziranje" kreativnosti, slobode i slično. Pretpostavljaju da nijedno računalo nikada ne bi moglo biti uistinu inventivno jer „može raditi samo ono za što je programirano". Sve ovisi samo o tome što ovo navodno ograničenje znači. U jednom tehničkom i dosadnom smislu, naravno, potpuno je točno da računala uvijek slijede svoje programe, jer program nije ništa drugo do pažljiva specifikacija svih relevantnih procesa unutar stroja. Međutim, to ne dokazuje ništa jer, pod pretpostavkom da postoji „precizna specifikacija" svih relevantnih procesa u našem mozgu, sličan stav bi se mogao izreći i o nama. Ali, očito, nijedan takav argument ne bi mogao pokazati da nikada nismo kreativni ili slobodni. Osnovni problem argumenta je taj što zanemaruje hijerarhijske razlike u organizacijskom stupnju. Niti jedna pojedinačna moždana funkcija (pa tako ni jedna operacija u računalu) ne može se nazvati slobodnom ili kreativnom; takvi opisi pripadaju potpuno drugoj razini, onoj na kojoj se o sustavu ili osobi govori kao o cjelini. Nažalost, zabune ostaju jer je pojam „programirano" dvosmislen. Umjesto upravo navedenog smisla, u kojem se *program* odnosi na detaljnu specifikaciju internih procesa, pojam se može upotrijebiti šire za opis cjelokupnog dizajna sustava ili predviđenih mogućnosti. Na primjer, mogli bismo reći „ovo je računalo programirano da vodi račune plaća" ili „ovo računalo je programirano da pronađe najbolji put leta u lošim vremenskim uvjetima". Ovi se opisi primjenjuju na sustav u cjelini, no čak i na ovoj razini čini se da sustavi „mogu raditi samo ono za što su programirani" — dokle god funkcioniraju ispravno (bez kvarova). Zašto onda sustav jednostavno ne programiramo tako da bude kreativan? Tada bi odmah imao te karakteristike. Sam izraz „programiran za kreativnost" može zvučati kao kontradikcija, ali zapravo to nije, kao što možemo vidjeti ponovnim razmatranjem sebe. Kada smo zdravi (bez kvara), „radimo samo ono za što smo dizajnirani". Ali onda, pod pretpostavkom da kreativnost i sloboda nisu (uvijek) nezdravi, moramo biti „dizajnirani za kreativnost". Ipak, reći da nas je evolucija „dizajnirala", samo je metafora. Evolucija nije stvarni dizajner, već bezumni prirodni proces. S druge strane, računala su doslovno programirana od strane stvarnih (ljudskih) programera. Kada je čovjek kreativan, taj kreativni rad smatra se njegovim, ali kada računalni ispis sadrži nešto umjetničko, to je onda umjetnost programera, a ne stroja. Zašto bi potencijal entiteta za inventivnost trebao biti određen prema njegovom podrijetlu, a ne prema očitoj vlastitoj sposobnosti? Naravno, da su relevantni programeri te „izume" unaprijed zamislili i samo ih pohranili u stroj za kasniju „reprodukciju", tada bi zasluga bila njihova. Ali to uopće nije način na koji UI djeluje. Ono što se izravno programira je samo hrpa općih informacija i principa. Što se događa nakon toga, što sustav radi sa svim tim unosima, nije predvidljivo od strane dizajnera (programera). Najupečatljiviji su primjeri šahovskih strojeva koji nadigravaju svoje programere, nudeći briljantne poteze koje potonji možda nikada ne bi pronašli.

Znanje i učenje

Moglo bi se činiti da već imamo sasvim razuman standard inteligencije: rezultat na testu inteligencije. Međutim, postoje dva problema s tom pretpostavkom. Prvo, testovi inteligencije namijenjeni su mjerenju *stupnja* inteligencije, pod pretpostavkom da ispitanik ima inteligenciju za mjerenje. Ali za računala je ta pretpostavka problematična. Moramo znati ima li smisla uopće im pripisivati inteligenciju prije nego što možemo pitati u kolikoj mjeri je imaju. Drugo, testovi inteligencije dizajnirani su posebno za ljude te bi se moglo ustanoviti da je neka neobična vještina, poput rješavanja logičkih problema, u korelaciji s ljudskom inteligencijom. Ali to ne znači da takav zadatak zapravo zahtijeva inteligenciju. Činjenica da mnogi ljudi ne mogu napamet izračunati drugi korijen ne znači da su džepni kalkulatori inteligentniji od njih. Ono što nam treba je općeniti test je li entitet uopće inteligentan (u ljudskom smislu).

Osnovni cilj UI trebao bi biti izgradnja sustava koji *uče*. Međutim, u ovom smislu, UI više-manje ignorira učenje. Učenje je stjecanje znanja, vještina i sličnog. Kako bismo sustav koji može znati učinili sposobnim za stjecanje znanja? Ovo prešutno pretpostavlja da je primjena znanja izravna i da je njegovo stjecanje ili prilagodba najteži dio, no to se pokazalo lažnim. UI je otkrila da je samo znanje izvanredno složeno i teško ga je primijeniti — toliko da čak ni opća struktura sustava sa zdravim razumom još nije jasna. Dakle, umjetna inteligencija mora započeti pokušajem razumijevanja znanja (i vještina i svega drugog što se stječe), a zatim se na toj osnovi baviti učenjem. Svakako je sposobnost učenja ključna za potpunu inteligenciju i UI bez toga ne može uspjeti. No, ne čini se da je učenje najosnovniji problem, a kamoli prečac ili prirodno polazište.

2.3 Računala: interpretirani automatski formalni sustavi

Formalne igre

Formalni sustav je poput igre u kojoj se manipulira simbolima prema pravilima, kako bismo vidjeli kakve konfiguracije možemo dobiti. Zapravo, mnoge su poznate igre — među njima šah, dama, go, i križić-kružić — formalni sustavi. Ali druge igre — poput pikula, biljara, i bejzbola — uopće nisu formalne (u ovom smislu). Koja je razlika? Sve formalne igre imaju tri bitne značajke: igraju se manipulacijom simbolima, digitalne su i skup pravila im je konačan. Simboli (tzv. *tokeni*) formalne igre samo su „figure" („markeri", „žetoni" ili bilo što drugo) kojima se igra. Konkretno, u šahu, dami ili gou, to bi bili mali fizički predmeti koje je lako manipulati rukama (prstima). Međutim, tokeni ne moraju biti fizički. Na primjer, u igri križić-kružić to su znakovi koje crtamo na papiru, ploči ili slično (obično znakovi $\times$ i $\bigcirc$). Za mnoge elektroničke igre, tokeni su zapravo postavke prekidača i uzorci boja, odnosno, rasporedi grafičkih elemenata na zaslonu. Manipuliranje tokenima može

značiti:

1. njihovo premještanje (npr. pomicanje po nekoj ploči ili polju za igru)
2. mijenjanje svojstava tokena (ili zamjena drugim tokenima)
3. dodavanje novih tokena na određenu poziciju
4. micanje nekih tokena iz igre.

Često se koristi izraz *potez* za bilo kakvu ograničenu manipulaciju simbolima (tokenima). Kako bismo u potpunosti definirali bilo koju igru u kojoj se manipulira tokenima — bilo koji formalni sustav — moramo navesti:

1. što su tokeni,
2. koja je početna konfiguracija (ili koje su moguće početne konfiuracije) tokena, te
3. koji bi potezi (manipulacije) bili dopušteni u bilo kojoj konfiguraciji tokena (ili uz određenu povijest konfiguracija) — odnosno kakva su pravila.

Igra započinje nekom početnom konfiguracijom tokena i nastavlja se mijenjanjem konfiguracije, korak po korak. Puno formalnih igara ima standardnu početnu konfiguraciju koja je uvijek ista, ali, na primjer, Go ima nekoliko početnih konfiguracija, ovisno o relativnoj snazi igrača te se može definirati bilo koji broj mogućih početnih konfiguracija. Konfiguracije se mijenjaju *dopuštenim potezima*, odnosno, manipulacijama tokena prema propisanim pravilima. Dopuštene manipulacije određene su trenutnom konfiguracijom tokena ili, u nekim slučajevima, nekoliko ili svim prošlim konfiguracijama. Dakle, potez koji je legalan u jednoj konfiguraciji, možda nije legalan u drugoj, ali u istoj konfiguraciji (ili povijesti konfiguracija) uvijek će biti dopušteni isti potezi. Oni formalni sustavi koje smatramo igrama obično imaju posebnu klasu konfiguracija koje su označene kao *cilj* ili „*pobjeda*". U takvim igrama igrač ili igrači nastoje postići jedan od tih položaja pomnim planiranjem i odabirom poteza. Kada u igri sudjeluje više igrača, oni su često protivnici, koji se natječu u tome tko će prvi doći do pobjedničke konfiguracije. No, neophodno je shvatiti da nisu svi formalni sustavi natjecateljske igre, a neki čak i nemaju ciljne konfiguracije, odnosno, u njima ne možemo „pobijediti". Najjednostavniji formalni sustavi imaju samo jednu vrstu tokena, dok u zanimljivijim formalnim sustavima postoje brojne različite vrste tokena. Primjerice, u šahu ih ima šest (za svaku stranu): kralj, dama, top, lovac, skakač i pješak. Formalni tokeni su zamjenjivi ako i samo ako su iste vrste. Izmjena žetona iste vrste nije potez i ne utječe na konfiguraciju. Mijenjanje tokena kao jedna od vrsta manipulacije tokenima podrazumijeva zamjenu drugom vrstom tokena.

Digitalni sustavi

Riječ „digitalno" općenito podsjeća na elektronička računala ili možda diskretna numerička „očitanja", ali digitalni sustavi uključuju daleko više od toga. Svaki formalni sustav je digitalan, ali nije svaki digitalni sustav formalan. Abeceda je digitalna, kao i svaka valuta (novac) i igraće karte dok, na primjer, slika na platnu nije digitalna. Digitalni sustav skup je pozitivnih i pouzdanih tehnika (metoda, uređaja) za stvaranje i kasnije prepoznavanje tokena, iz nekog unaprijed određenog skupa tipova tokena. Drugim riječima, digitalni sustav je skup *pozitivnih* tehnika pisanja i čitanja. Pozitivna tehnika je ona koja može *apsolutno* uspjeti. Suprotne od pozitivnih bile bi metode koje mogu uspjeti samo relativno, djelomično ili „gotovo točno", „u gotovo svakom pogledu" i slično. Je li zadana tehnika pozitivna uvelike ovisi o tome što se računa kao uspjeh (na primjer mjerenje duljine ravnalom bila bi pozitivna metoda ako se uspjehom smatra odrediti duljinu s točnošću od jednog milimetra, dok ista metoda možda ne bi bila pozitivna ako se traži točnije mjerenje). Pozitivna tehnika ne mora uvijek uspjeti; uspjeh čak ne mora biti vrlo vjerojatan. Pitanje nije hoće li uspjeti (ili koliko često), već koliko dobro *može* uspjeti: pozitivna tehnika ima mogućnost savršenog uspjeha. Kad se pitamo o vjerojatnosti uspjeha, tada govorimo o *pouzdanosti*. Mnoge su tehnike pozitivne i pouzdane. Gađanje košarkaške lopte u koš je pozitivna metoda, budući da može uspjeti apsolutno i bez kvalifikacija — štoviše, za nadarene igrače, prilično je pouzdana. Brojenje je pozitivna tehnika za određivanje broja olovaka u kutiji i većina ljudi može to pouzdano učiniti. Digitalne tehnike su tehnike pisanja i čitanja. „Pisati" token znači proizvesti jednu od zadanih vrsta tokena. „Čitati" token znači odrediti kojeg je tipa. Sustav čitanja i pisanja je uspješan ako je tip tokena određen čitanjem isti kao tip tokena zadan pisanjem. Digitalni sustav je skup tehnika pisanja i čitanja koje su pozitivne i pouzdane u odnosu na ovaj standard za pisanje i čitanje. U stvarnom svijetu uvijek postoje varijacije i pogreške. Lopta nikada dva puta ne prolazi na isti način kroz koš, kut prekidača je mikroskopski različit svaki put kad se postavi, nijedno slovo `A` nije potpuno isto kao drugo, ako ga tiskamo na papiru (bilo strojem ili rukom). Često kažemo „ništa nije savršeno", ali digitalni sustavi (ponekad) postižu savršenstvo, unatoč ograničenjima stvarnog svijeta. Kako? U osnovi, dopuštaju određenu marginu za pogrešku, unutar koje su sve izvedbe jednake i uspješne. Stoga točna svojstva tokena nisu važna, sve dok ostaju unutar tolerancije.

Jedan od razloga zašto je ovo važno jest to da nam dopušta kompleksnost koju bismo inače teško ili nikako mogli postići. Usporedimo dvije konvencije za praćenje količine (ili iznosa) novca u igri pokera. Svaka koristi različite boje za različite vrijednosti: plava: sto, crvena: deset i bijela: jedan. U jednom sustavu jedinica svake denominacije je obojeni plastični disk (*poker chip*), dok je u drugom sustavu to žlica sitnog, obojenog pijeska. Prednost pijeska je što možemo uložiti djelomični iznos, upotrebom manje od pune žlice bijelog pijeska. Sa žetonima za poker to ne možemo učiniti. Sustav žetona je digitalan: postoji pozitivna tehnika za određivanje iznosa bilo koje oklade (brojeći žetone svake boje)

i stoga je svaki ulog egzaktan. Prednost digitalnog aranžmana postaje očita u slučaju velikih, preciznih uloga. Budući da volumen pijeska nikad ne možemo izmjeriti točno, uvijek bi bilo malih pogreška — recimo ±2%. Sad zamislimo da pokušavamo uložiti 914 jedinica. Sa žetonima je lako: izbrojimo devet plavih, jedan crveni i četiri bijela. S pijeskom stvari nisu tako lijepe. Dva posto od 900 jedinica je 18 jedinica pa je očekivana pogreška na plavi pijesak veće vrijednosti nego sav crveni i bijeli pijesak zajedno. Zapravo, nepreciznost u velikim apoenima preplavljuje čitav značaj malih apoena i čini ih suvišnima. Ali skromne „nesavršenosti" ne utječu na vrijednost žetona. Čak i kada su ogrebani i istrošeni, plavi žetoni još uvijek vrijede točno 100 bijelih. Prema definiciji, svaki formalni sustav je digitalni sustav. Formalni sustavi tako mogu postati vrlo kompleksni bez gubitka na točnosti.

Neovisnost o mediju i formalna ekvivalencija

Šahovske figure postoje u mnogim stilovima i veličinama. Njihova je boja obično važna, kako bi se razlikovale suprotne strane. Ili točnije, ono što je zaista važno je da svaki token bude moguće pozitivno identificirati. U skladu s tim ograničenjem, međutim, nebitno je u kojem su mediju tokeni ostvareni ili kako su prikazani. Ovo svojstvo nazivamo *neovisnost o mediju*. Digitalnost čini neovisnost o mediju izvedivom. Jasno je da tokeni formalnog sustava, u određenom mediju, moraju biti dovoljno upravljivi da se dopušteni potezi mogu izvoditi (pisati) i također biti dovoljno trajni da se konfiguracije i dalje mogu pozitivno prepoznati (čitati). Dva formalna sustava su *ekvivalentna* ako i samo ako postoji relacija između njihovih konfiguracija takva da vrijedi:

1. za svaku konfiguraciju u jednom sustavu postoji točno jedna odgovarajuća konfiguracija u drugom sustavu;
2. ako je potez dopušten u jednom sustavu, tada je odgovarajući potez (iz odgovarajuće konfiguracije u odgovarajuću konfiguraciju) dopušten u drugom sustavu;
3. (sve) početne konfiguracije su međusobno odgovarajuće.

Pojmovi neovisnosti o mediju i formalne ekvivalencije presudno su važni za umjetnu inteligenciju. Moždane stanice i elektronički sklopovi očito su različiti „mediji", ali možda na nekoj prikladnoj razini apstrakcije, mogu poslužiti za ekvivalentne formalne sustave.

Konačna mogućnost reprodukcije i algoritmi

Koje granice postoje (ako postoje) za veličinu i složenost formalnih sustava? Moći igrati formalnu igru znači biti u mogućnosti slijediti pravila. Ali što to točno uključuje? Zapravo, igrači moraju uvijek (u bilo kojoj konfiguraciji) biti u mogućnosti

1. reći, za bilo koji predloženi potez, bi li taj potez bio dopušten (u toj konfiguraciji),

2. povući barem jedan dopušteni potez (ili pokazati da takav ne postoji).

Algoritam je konačni niz precizno definiranih uputa. Najjednostavnija vrsta algoritma je izravni raspored uputa: prvo napravite A, a zatim $B \ldots i$, konačno, učinite Z. Ne treba puno da bismo „slijedili" takav algoritam. U osnovi, moramo pročitati trenutnu uputu, izvršiti je, a zatim prijeći na sljedeću uputu na popisu. No, te sposobnosti praćenja redoslijeda uputa bitno se razlikuju od sposobnosti za izvođenje navedenog (govoriti što treba raditi nije isto što i raditi to). Igraču su potrebne sposobnosti obje vrste za provođenje algoritma. Međutim, problem s ovakvim algoritmima je u tome što su potpuno nefleksibilni: propisan je potpuno isti slijed koraka, bez obzira na ulazne podatke ili bilo kakve međurezultate. Zato općenito algoritmi u sebi uključuju mogućnost izbora odnosno ponavljanja koraka ovisno o ulaznim podacima ili međurezultatima.

Složeni tokeni

Proučimo kratko arapske brojeve. Oni se sastoje od znamenaka pa čine jedan primjer *složenih tokena*. Jednostavni tokeni su znamenke, kojih je konačno mnogo, a svaki arapski broj dobiven je nizanjem jedne ili više znamenaka. Promatrajući množenje arapskih brojeva, vidimo da se ono svodi na množenje pojedinih znamenaka. U ovom su slučaju tokeni složeni i njihova „uporaba" ovisi o njihovom sastavu, a pritom nisu uključena nikakva značenja — sustav je sasvim formalan. Ovaj primjer također ilustrira snagu kombiniranja složenih tokena sa složenim algoritmima, jer ovdje imamo sustav s beskonačno mnogo različitih dopuštenih poteza, a „igranje" ostaje konačno.

Automatski sustavi

Automatski formalni sustav je formalni sustav koji „djeluje" (ili „igra") sam od sebe. Točnije, riječ je o fizičkom uređaju (stroju), sa sljedećim karakteristikama:

1. neki od njegovih dijelova ili stanja identificirani su kao simboli (tokeni), u konfiguraciji nekog formalnog sustava;

2. u svom normalnom radu sustav automatski manipulira tim simbolima prema pravilima tog sustava.

Stvarno igranje igre uključuje više od samih konfiguracija i poteza: također mora postojati jedan ili više *igrača* i *sudac*. Igrači povlače poteze, kad dođu na red. Sudac ne čini nikakve poteze, već određuje tko je na redu — to jest, koji bi igrač trebao krenuti sljedeći, i, možda, na kojim bi tokenima trebao raditi. Sudac također zadaje početnu

konfiguraciju, odlučuje kada je igra gotova, izriče rezultate itd. Za prijateljske igre skloni smo previdjeti sudačku funkciju, jer je često tako jednostavna da igrači to izvode za sebe. Ali strogo govoreći, suđenje je uvijek odvojeno od igranja. Budući da automatski sustav igru zapravo igra sam, on mora uključivati sve te elemente. Ponekad je formalna igra samo djelomično automatizirana. Na primjer, elektronički šah obično automatizira samo jednog od igrača (i suca). Što algoritmi imaju s automatskim formalnim sustavima? Na neki način, sve. Izvršavanje algoritma zahtijeva fiksni, konačni repertoar primitivnih sposobnosti — primitivne operacije koje sustav obavlja i primitivni recept koji slijedi. Algoritam može biti izvršen na automatskom sustavu sastavljenom od igrača i suca. Ali što je s tim primitivnim sposobnostima? Svaku primitivnu operaciju u formalnim igrama možemo rasčlaniti na tako jednostavne operacije da bi ih čak i stroj mogao izvesti. To je **princip automatizacije**: kad god su dopušteni potezi formalnog sustava u potpunosti određeni algoritmom, taj sustav može biti automatiziran. Zbog toga je automatizacija poteza šahovskog sustava (teoretski) trivijalna. Međutim, komponenta birača poteza u ovo vrijeme bila je druga priča. Ako je u popisu poteza naveden samo jedan dopušteni potez, nema izbora pa nema ni problema. Ali šah je nedeterminističan, odnosno u većini konfiguracija, bilo koji od nekoliko različitih poteza bio bi dopušten. Nasuprot tome, algoritmi su uvijek deterministični: po definiciji su postupci koji nikada ne dopuštaju nikakve mogućnosti ili nesigurnosti oko sljedećeg koraka. Kako onda sustav koji dopušta izbore može biti automatiziran? Birač poteza može, na primjer, uvijek odabrati potez koji se pojavio na vrhu popisa. Ali takvi bi sustavi igrali vrlo čudan šah. Pravi je problem bio, dakle, dizajnirati (razumnni, inteligentni) birač koji bira dobre poteze. Očito birač ne mora uvijek pronaći najbolji potez, jer čak ni svjetski prvaci ne igraju savršeni šah. Cilj je bio sustav koji relativno dobro odabire većinu vremena. Drugim riječima, nepogrešiv algoritam za bolji potez zapravo nije potreban. Bilo bi dovoljno imati pouzdan test kojim bi stroj mogao eliminirati najgore izbore i odlučiti se za neki prilično dobar potez. Tako, pogrešni, ali „prilično pouzdani" postupci nazivaju se *heuristikama*. Stoga heuristika (tako definirana) nije algoritam.

Smisao

Budući da su formalni sustavi (po definiciji) samostalni, oni ne podliježu semantičkim interpretacijama u uobičajenom smislu. Rezultat toga je da problemi shvaćanja značenja ne utječu na formalne sustave. Doista, formalno slijeđenje pravila može se u potpunosti mehanizirati, u automatskim formalnim sustavima. Međutim, ako želimo bolje razumjeti umjetnu inteligenciju ili čak ljudski um, moramo odgovoriti i na pitanje što je to značenje (smisao) i što ono podrazumijeva. Kako možemo objasniti značenje misli? Kognitivna znanost i umjetna inteligencija temelje se na pretpostavci: razmišljanje je poput razgovora. Stoga se možemo voditi, barem dijelomično onime što znamo o značenju jezika.

Primjer 2.3.1. Razmotrimo što je zajedničko sljedećim rečenicama:

Tvrdim: *Fido jede insekte.*

Raspitujem se: *Jede li Fido insekte?*

Zahtijevam: *Pazite da Fido jede insekte.*

Ispričavam se: *Žao mi je što Fido jede insekte.*

Sve se tiču Fida i toga kako on jede insekte ili (kako filozofi kažu) dijele sudovni kontekst da Fido jede insekte, ali svaka rečenica drugačije predstavlja taj sadržaj, kao tvrdnju, pitanje, zahtjev i ispriku. Ovako možemo isti sadržaj predstaviti na drugi način. Dakle, u rečenici razlikujemo osnovni sadržaj od načina izražaja.

Unatoč sličnostima govora i misli, ne možemo tvrditi da se sve misli mogu izraziti govorom, pa čak ni da bilo kakvu misao možemo u potpunosti izraziti riječima. Kognitivna znanost, međutim, posvećena je ovim sličnostima te tvrdi da su sve misli slične lingvističkom izričaju, ali u nekom apstraktnijem smislu. Važno apstraktno svojstvo je *simboličnost* sustava. To znači dvije stvari:

1. značenja jednostavnih simbola (npr. riječi) su proizvoljna;

2. značenja složenih simbola (npr. rečenica) sustavno su određena njihovim sastavom.

„Sastav" složenog simbola ne ovisi samo o tome od kojih se jednostavnih simbola sastoji, već i od toga kako su povezani — „forma" ili „gramatika". Jezici su barem približno simbolički sustavi. Značenja riječi su proizvoljna, odnosno nema intrinzičnog razloga da za određeni pojam kažemo određenu riječ. Njihova stvarna značenja utvrđena su samo uobičajenom uporabom unutar jezične zajednice. Budući da se značenja rečenica sustavno određuju pomoću njihova sastava, običan govornik engleskog jezika, s rječnikom od četrdeset ili pedeset tisuća riječi, može razumjeti mnogo različitih rečenica, o bezbroj različitih tema. Na prvi pogled, čini se da je misao još svestranija i fleksibilnija nego jezik. S obzirom na to da je naš mozak ograničen — samo deset milijardi neurona — postavlja se pitanje: Kako takav „maleni" sustav može imati takav veliki repertoar?

Misli su općenito složeni simboli te se sustavno grade iz razmjerno skromnog skupa jednostavnih simbola. Svestranost misli pripisuje se kombinatornoj strukturi simboličkih sustava. Umjetna inteligencija prihvaća ovaj pristup, da su misli simbolički izražene — ali to ne znači da ljudi misle na hrvatskom, engleskom ili bilo kojem jeziku, već samo da su misli poput simboličkog sustava u kojem razlikujemo sadržaj i način izražavanja.

Interpretacija

Interpretirati znači nečemu naći smisao. Da bi se simbolički sustav oznaka ili tokena protumačio, svakoj oznaci moramo odrediti značenje. Specifikacija obično ima dva dijela:

- što znače jednostavni simboli, i
- kako su značenja složenih simbola određena njihovim sastavom (komponente plus struktura).

Dakle, to je poput prevođenja novootkrivenog jezika. Tumačenje dajemo izradom „priručnika za prijevod", koji se, otprilike, sastoji od *rječnika* i *gramatike*, a objašnjenje je dano unutar nekog drugog, već poznatog, sustava. Kada je interpretacija uspješna? Zamislimo da imamo kriptogram. Zašto bismo ga interpretirali na jedan način umjesto na neki drugi? Zašto je to uopće kriptogram, zašto taj niz slova i riječi nije sam po sebi interpretacija (pod pretpostavkom da su zadani simboli slova hrvatske ili engleske abecede)? Do neke mjere, očito je da takva tumačenja ne bi bile prave interpretacije jer nemaju smisla, nemaju značenje. Može se dogoditi da su sama slova, ili pak riječi u rečenici, u potpunosti izmiješana. Ali što onda ima smisla? Govorenje istine je komponenta u stvaranju smisla (slučajna laž je vrsta nesuvislosti). Ali smislenost je očito širi pojam od istinitosti. Razumni ljudi mogu pogriješiti, možda ne vide situaciju iz nekog ugla. Također, možemo smisleno pričati o nepostojećim entitetima, na primjer o djedu Mrazu i njegovim vilenjacima. Pronalaženje koherentnosti, smisla, temelj je svih semantičkih interpretacija. Filozofski, ovo je teško i mutno područje, ali intuitivno možemo odrediti što znači da nešto *ima smisla*.

Interpretirani formalni sustavi

Formalni sustavi mogu se interpretirati: njihovim tokenima mogu se dodijeliti značenja. Štoviše, ako su postulati simboličke umjetne inteligencije točni, sami um je (poseban) interpretirani formalni sustav. Opću teoriju interpretacije i simboličkog značenja zovemo *semantika*. Dodijeljena značenja, kao i sve odnose ili karakteristike interpretiranih tokena, nazivamo *semantičkim svojstvima*. Na primjer, ako je „Fido" formalni token koji predstavlja određenu žabu, tada je povezanost tog imena sa žabom semantičko svojstvo. Istina i laž su semantička svojstva; kada bi istinitost jedne rečenice značila istinitost druge, to bi bio semantički odnos među njima.

Nasuprot semantici stoji *sintaksa*. Kad se raspravlja o formalnim sustavima, „sintaksni" zapravo znači „formalni", ali izraz se tipično koristi samo kada je kontrast sa semantikom bitan. Na primjer, bilo bi čudno (iako ne i netočno) govoriti o „sintaksi" šahovske pozicije, jer šah nije inerpretirani sustav.

Interpretacija i semantika nadilaze strogo formalno jer formalni sustavi, kao takvi, moraju biti samostalni. Stoga, interpretirani tokeni vode dva „života":

- **sintaksni život** u kojem su beznačajne oznake, koje se pomiču po određenim pravilima neke formalne igre.
- **semantički život** u kojem imaju značenje i vezu s vanjskim svijetom.

Athol je otkrio neobičnu igru [3]: njegovi tokeni su slova A–O (po jedno za svaku znamenku dekadskog brojevnog sustava te znakove +, −, ×, : i =). Početna pozicija je red slova koji završava određenim slovom (onim koje predstavlja znak =) koje se pojavljuje samo na kraju reda, a potez se sastoji od dodavanja još slova (osim ovog koje predstavlja znak =) na kraj tog reda. Za svaku početnu konfiguraciju postoji točno jedan dopušten potez, nakon kojeg je igra gotova. Atholova odvažna pretpostavka bila je da su ova slova interpretacije ili izravni prijevodi arapskih brojeva i znakova za računanje, odnosno, to su brojevi i znakovi, ali u alternativnom zapisu. Jedan od načina mapiranja ovih simbola i znamenaka slovima daje nizove koji izgledaju kao jednadžbe i stvarno jesu istinite jednadžbe. Odnosno, nijedno mapiranje nije interpretacija, osim jednog, koje ima smisla. „Formalni" izračun, naravno, nije revolucionirao matematiku, ovdje su važno otkriće bili *formalni aksiomatski sustavi*. Aksiomatski sustav sastoji se od posebnog skupa iskaza (nazvanih *aksiomima* i/ili *definicijama*), te niza zaključaka i pravila za izvođenje daljnjih tvrdnji, nazvanih teoremima. Osnovni uvjeti koje taj sustav treba zadovoljiti su:

1. neproturječnost (konzistentnost) — od dva bilo koja proturječna iskaza barem za jedan od njih mora biti isključena mogućnost dokazivanja u okviru sustava;

2. potpunost — od dva proturječna iskaza, na koja se aksiomatski sustav odnosi, barem za jedan mora postojati mogućnost dokazivanja u okviru sustava;

3. neovisnost — kako bi broj aksioma bio što manji, treba ukloniti aksiome koji ovise o drugim aksiomima u smislu da se iz njih mogu izvesti.

Kao formalna igra, aksiomatski sustav igra se pisanjem složenih tokena nazvanih *formulama*. Prvo se, kao početna konfiguracija, zapisuju aksiomi u obliku formula. Pravila tada dopuštaju dodavanje drugih formula, ovisno o konfiguraciji. Jednom napisani, tokeni nikada nisu izmijenjeni ili uklonjeni, tako da konfiguracije postaju sve veće kako se igra nastavlja. Svaka novododana formula postaje formalni teorem sustava, a igra koja vodi dodavanju te formule je formalni dokaz tog teorema. Ideja je, očito, dizajnirati ove formalne sustave tako da se mogu u intuitivnom smislu protumačiti kao aksiomatski sustavi. Ovo zahtijeva dvije stvari od sustava:

1. aksiomi trebaju biti istiniti;

2. pravila trebaju čuvati istinu.

Kad sustav ima sve aksiome istinite i pravila koja se brinu da istina bude očuvana, zajamčeno je da je svaka formula u svakoj dopuštenoj konfiguraciji istinita. Drugim riječima, svi formalni teoremi bit će istiniti. Ako formalna (sintaksna) pravila određuju tekstove i ako (semantika) interpretacija mora imati smisla za sve te tekstove, tada je jednostavno igranje po pravilima siguran put do teksta koji ima smisla. Ako se pridržavamo

formalnih aritmetičkih pravila, na primjer, sve što napišemo bit će istinito. To je razlog zašto su interpretirani formalni sustavi uopće zanimljivi i važni.

Moto formalista glasi:

> Ako se brinete o sintaksi, semantika će se pobrinuti sama za sebe.

Briga o sintaksi jednostavno je igranje po pravilima, a tada će automatski sve imati smisla (semantički). Na taj način se spajaju dva „života" interpretiranog formalnog sustava. Međutim, ovako se može postići samo prvo svojstvo (konzistentnost), ne i potpunost (to nam kažu Gödelovi teoremi) i neovisnost (ovo je neodlučivo, Church).

Računala

Računalo je interpretirani automatski formalni sustav — stroj za manipulaciju simbolima.

Ova definicija ima dvije osnovne klauzule:

1. sudac i (barem neki) igrači tog formalnog sustava su automatizirani — to su „crne kutije" koje automatski manipuliraju tokenima u skladu s pravilima;

2. ti tokeni su zapravo simboli — oni su interpretirani na način da legalni potezi „imaju smisla" u kontekstu u kojem su napravljeni.

Najjednostavniji primjer je džepni kalkulator — automatizacija Atholove igre slovima (ali pomoću poznatijih znakova). Svaki novi početni položaj unosi se tipkama, a kad se unese završni token (tipkom =) stroj automatski ispisuje jedinstveni legalan potez za zadanu poziciju. Crne kutije koje predstavljaju igrača ili suca ovdje su elektronički dio kalkulatora, međutim, sadržaj tih crnih kutija nije bitan za status sustava kao računala. Jedino što je važno je da crne kutije u dopuštenim konfiguracijama generiraju dopuštene poteze. Nažalost, primjer kalkulatora toliko je jednostavan da ne dolazi do izražaja važan problem. Čak i kad se provodi ručno, aritmetički proračun u potpunosti je deterministčkan: svaki potez je u potpunosti određen samo pravilima. Za sofisticiranije sustave pravila gotovo nikad nisu deterministična, odnosno, igrač bira koji će potez, od nekoliko dopuštenih, odigrati. Dakle, potez koji će biti odigran ovisi o pravilima i igračevom odabiru.

Načelo interpretacije uvijek je koherentnost. Ako redoslijed u tekstu proizlazi iz dvaju različitih izvora, tada interpretacija može pronaći smisao na dvije razine — osnovnoj i profinjenoj. Ovdje osnovna razina predstavlja pusto poštivanje pravila, a profinjena razina dolazi od igračevog izbora poteza uz određeni razlog/cilj (npr. nalaženje elegantnih dokaza ili zanimljivih teorema). Razmotrimo, na primjer, jednostavnu aksiomatizaciju srednjoškolske algebre. Jedino ograničenje u njenoj interpretaciji jest da svi teoremi — sve formule koje legalno možemo dobiti iz aksioma — moraju biti istiniti. To je osnovna

razina smisla. Ali to ne govori ništa o drugoj razini smisla koja nam govori o tome koji se teoremi dokazuju i kako. Na primjer, bilo bi besmisleno dokazivati niz teorema:

$$a = a$$
$$a + a = a + a$$
$$a + a + a = a + a + a$$
$$a + a + a + a = a + a + a + a$$
$$\ldots$$

U takvim slučajevima razlikujemo interpretaciju samog sustava od procjene igrača. Poanta je da ne postoji (poznat) način da se svaki sustav („igra") podijeli tako da jedan dio (dovoljan za interpretaciju) može zajamčiti formalna pravila, dok se ostalo pripisuje inteligenciji igrača. Zapravo, broj sustava u kojima je to moguće vrlo je ograničen. Za manualne formalne sustave, razlika između formalnih ograničenja i izbora igrača je eksplicitna i oštra. Stoga tumačenje manualnih sustava zahtijeva podjelu rada: svi legalni potezi moraju imati smisla (iz interpretacije), bez obzira na to tko ih izvodi i zašto. Međutim, automatski sustavi su, ili barem mogu biti, drugačiji. Automatski sustav je stvarni fizički uređaj sa „ugrađenim" tokenima za igru, igračima i sucem, a njegovo stvarno ponašanje opisano je tekstom, koji može biti interpretiran u postojećem obliku. Odnosno, čak i kada ne bi bilo načina za razlikovanje pravila od izbora igrača, sustav u cjelini još uvijek može biti interpretiran. Usporedimo dva slučaja.

Tip A

Ako uzmemo prethodno interpretirani formalni sustav i automatiziramo ga, tada će njegovi automatski izlazi i dalje poštivati ista formalna pravila te će imati smisla prema istoj osnovnoj interpretaciji. Dakle, ostaje izvorna podjela rada. Naravno, isti ti izlazi mogu se pojaviti u profinjenijem redoslijedu. Naime, ako pretpostavimo da je sustav nedeterministična, uz pretpostavku da je automatiziran na sofisticiran način („dobri" igrači), rezultati možda neće biti samo smisleni, već i pametni, dobro motivirani, zanimljivi itd. Uzmimo za primjer automatskog igrača šaha. Šahovske figure se ne interpretiraju pa ne postoji „semantički" kriterij po kojem bi se rad mogao podijeliti, ali problem povlačenja dobrih (za razliku od povlačenja bilo kojeg dopuštenog) poteza nastaje i u šahu. Tako, možemo podijeliti ovog zamišljenog automatiziranog igrača na dva dijela: jedan navodi sve dopuštene poteze u trenutnom položaju (svi ti potezi imaju osnovnu razinu smisla — dopušteni su), a drugi među tim potezima odabire jedan (neki „zanimljiv" ili „pametan" potez). U ovom jednostavnom slučaju, podjela rada bila bi konkretno određena samom strukturom stroja. Stvarni automatski sustav, međutim, mogao bi biti tipa A čak i ako nije prvotno zamišljen kao interpretirani manualni sustav i zatim automatiziran. Važno je da možemo pronaći skup formalnih pravila koja učinkovito dijele rad u semantičkom smislu: moraju dozvoliti svaki

legalan potez (igrač ima izbor), a istovremeno podržavati osnovnu interpretaciju (logička valjanost, odnosno matematička ispravnost).

Tip B

Zaista uzbudljiva perspektiva je izgradnja sustava koji ne održavaju podjelu rada i stoga nisu ograničeni na domene gdje se prikladna podjela (formalizacija) može naći. Primjerice, računalo koje bi razgovaralo na engleskom jeziku, moglo bi se promatrati samo kao cjelina, odnosno, tek kad se govornikov izbor redoslijeda riječi nametne skupu legalnih poteza (gramatici), izlaz pokazuje dovoljno strukture da održi interpretaciju.

2.4 Good Old-Fashioned Artificial Intelligence — GOFAI

1985. godine John Haugeland u svojoj knjizi *Artificial Intelligence — The very idea* simboličkoj umjetnoj inteligenciji daje ime *Good Old-Fashioned Artificial Intelligence*, skraćeno GOFAI. Haugeland odvaja simboličku UI kao važnu temu rasprave i daljnjeg razvoja u području umjetne inteligencije. Kao grana kognitivne znanosti, GOFAI počiva na teoriji inteligencije i misli — što je u osnovi, Hobbesova ideja (*ratiocination is computation*), koja ne mora biti prihvaćena u svakom znanstvenom pristupu, odnosno, neki je mogu smatrati lažnom. Dakle, otvara se mogućnost postojanja drugih pristupa ovom polju. Međutim, kada bismo pretpostavili postojanje drugih pristupa umjetnoj inteligenciji, dogodilo bi se sljedeće:

- Kada bi neka alternativna teorija inteligencije (ne-GOFAI) uspjela, trebalo bi biti moguće izgraditi stroj koji bi po toj teoriji radio, koji posljedično ne bi bio GOFAI.
- Uzmemo li istu teoriju, hipotetski bi je se moglo simulirati na računalu (poput simulacije prirodne katastrofe, prometa ili sinteze proteina).

Ovdje je važno primijetiti da simulirani sustav i računalo na kojem je sustav simuliran ne rade po istim principima. Naime, računalo radi na temelju simbola, često brojeva, i zakona koji su mu zadani u svrhu simulacije, kako bi što točnije opisalo simulirani sustav. Tako bi i simulacija intelekta radila na isti način, iako (po pretpostavci) sam simulirani sustav (intelekt) potencijalno radi na neki drugi način. S druge strane, teza GOFAI-a nije da se procesi koje povezujemo s inteligencijom mogu opisati simbolički, već da oni jesu simboli. Točnije, esencijalne tvrdnje povezane s teorijom GOFAI su:

- Naša sposobnost inteligentnog ponašanja dolazi od vještine razumnog razmišljanja,
- koje se sastoji od unutarnje „automatske” manipulacije simbolima.

Iz ovoga odmah slijede dvije tvrdnje:

- Ove unutarnje manipulacije simbolima inteligentne su misli, koje se tiču vanjskog svijeta.
- Inteligentni sustav mora sadržavati neke računalne podsustave za izvršavanje tih „razumnih" unutarnjih manipulacija.

Haugeland ove podsustave uspoređuje s homunkulusima [3]. On kaže:

> Ako se ljudska inteligencija može objasniti samo uz pretpostavku inteligentnog homunkulusa koji manipulira mislima, time nismo ništa postigli, jer i dalje nismo objasnili pojam inteligencije. Međutim, ako umjesto toga pretpostavimo da psihom jednog čovjeka upravlja više homunkulusa, koji imaju neku razinu inteligencije, ali samo u cjelini mogu upravljati ljudskim tokom misli na razini ljudske inteligencije — te za svakog od tih homunkulusa pretpostavimo da njime također upravlja skupina homunkulusa niže razine inteligencije, možemo se tako spuštati razinu po razinu sve dok ne dođemo do homunkulusa bez inteligencije.

Ovdje se opet postavlja pitanje, kako postići bilo koju razinu inteligencije, ma koliko malu.

Vratimo se računalima. Znamo da deterministične formalne sustave možemo u potpunosti automatizirati i, kao takve, interpretirati. Na primjeru šaha vidjeli smo da automatizirati možemo i nedeterministične sustave, kao kombinaciju dvaju unutarnjih deterministčnih sustava. Konkretno, vidjeli smo kako se isti sustav može interpretirati na dva različita načina (algoritamski izračun odnosno heurističke odluke) te kako bismo šah direktno pretvorili u računalo tipa A. Dakle imamo, ili nam se tako čini, računalo s nekom razinom inteligencije. Ako bismo na ovo primijenili Haugelandovo razmišljanje o homunkulusima, teoretski bismo mogli dobiti ljudsku razinu inteligencije. Ideja je jednostavna, grupiramo sustave niže inteligencije i organziramo ih tako da zajedno čine jedan sustav veće inteligencije od pojedinog dijela i tako sve do računala tipa B. Međutim, još uvijek nije riješeno što to znači „neka" (mala, velika, manja, veća) razina inteligencije te se postavlja pitanje kako postaviti standard. Odgovor koji Haugeland daje otprilike glasi: Ako GOFAI jednoga dana uspije raščlaniti ljudsku inteligenciju u primitivne manipulacije simbolima, tada ta analiza postavlja standard „neke" inteligencije.

Sustavi GOFAI sastoje se od (internog) igraćeg polja, raznih tokena, jednog ili više igrača koji manipuliraju tim tokenima, i suca. Ove interne manipulacije tokenima interpretiramo kao misaone procese na temelju kojih sustav nalazi smisao i ponaša se inteligentno. Odnosno, sveukupna inteligencija sustava opisana je manjim (manje inteligentnim) komponentama, čija su međudjelovanja, zapravo, dio unutarnjeg djelovanja većeg sustava. Ako sada pretpostavimo da igrači obraćaju pozornost na značenja simbola, to znači da oni ne

mogu biti čisto mehanički, ali ako oni ipak ignoriraju značenje, tada manipulacije ne mogu biti razumne jer razumnost proizlazi iz značenja tih simbola. Čak i ako pretpostavimo da inteligencija dolazi u stupnjevima, ne možemo isto pretpostaviti i za značenje. Naime, simboli ili imaju značenje ili ne (mogu ili ne mogu biti interpretirani). Dakle, ovaj problem zahtjeva u potpunosti drugačiji pristup — *reinterpretaciju*, odnosno viđenje iste stvari (simbola) na drugi način. Tako je, s jedne strane, unutarnji igrač samo automatski formalni sustav, koji manipulira tokenima po određenim pravilima, a s druge strane, taj isti igrač manipulira istim tim tokenima (sada reinterpretiranim kao simbolima) na način koji odgovara njihovim značenjima.

Haugeland ovu razliku pokazuje na primjeru razlike između pisanog teksta i misli. Ako na primjer na papir napišemo:

- Sve žabe jedu insekte.

- Fido je žaba.

iz ove dvije rečenice neće proizaći novo značenje. Čak i da razrežemo taj papir u komadiće tako da na svakom piše druga riječ i pustimo da ih neka prirodna pojava rasporedi drugačije, u samom fizičkom svijetu neće se pojaviti nova misao, novo značenje. Međutim, sasvim je jasno da iz ove dvije rečenice slijedi da Fido jede insekte. Dakle, možemo zaključiti da ljudi razumiju svoje misli i njihova značenja, dok papir (ili neki drugi neživi medij) nema takve mogućnosti. Haugland zato misaone simbole naziva *semantički aktivnima*, dok su izražene (izrečene, zapisane) misli *semantički inertni* simboli. Inertni simboli tako dobivaju značenje samo u interakciji s korisnicima (ljudima koji ih pišu ili čitaju). Tako ovi inertni simboli dobivaju izvedeno značenje, kada ih korisnik interpretira na svoj način. Haugeland ovdje uvodi pojam *originalnog značenja* (značenje koje je nepromjenjivo, nije izvedeno, postoji bez interakcije s korisnikom) koje onda mogu imati samo aktivni simboli. Naravno, ovi „aktivni" simboli nisu aktivni sami po sebi već se takvima čine jer se procesi njihove interpretacije odvijaju u pozadini. Oni su interpretirana stanja ili dijelovi cjelovitog sustava unutar kojeg su automatski manipulirani. Dakle, simboli mogu biti aktivni samo u kontekstu brojnih moždanih procesa koji na njih reagiraju, utječu i povezuju ih te sami po sebi mogu izgledati aktivno samo ako sve te procese sakrijemo. Ovime smo zapravo opisali i računalo. U kontekstu cjelokupnog sustava, interpretirani tokeni međudjeluju dovodeći sustav u drugo stanje, a sve je to u potpunosti automatski i semantički primjereno. Komponente igrača i suca mogu i ne moraju biti sakrivene, ovisno o gledištu i potrebi. Zaključno, sustav kao cjelina manipulira tokenima u skladu s njihovim značenjima bez vanjske intervencije, te je time semantički aktivan.

2.5 Arhitektura računala

Prvo značajno računalo dizajnirao je Charles Babbage (1792.–1871.), engleski matematičar i filozof. Njegov prvi pokušaj dizajniranja računala naziva se *diferencijski stroj* (*difference engine*), razvijen je između 1823. i 1833. godine, a služio je za računanje vrijednosti polinomnih funkcija. Izgrađeno je nekoliko prototipova ovog stroja, ali projekt nikada nije zaživio jer je Babbage stalno zahtijevao modifikacije i poboljšanja te je sama konstrukcija bila izrazito skupa. Njegov drugi dizajn, *analitički stroj*, kojemu se posvetio oko 1833. godine, bio je uspješniji. Ovaj novi stroj temeljio se na dvije sasvim nove ideje koje će postati osnovom cijele računalne znanosti:

- sve operacije u potpunosti su *programabilne*;
- programi mogu sadržavati *uvjetovne grane*.

Nažalost, prije no što je stroj završen, Babbage je umro, a nije postojalo dovoljno objavljenih informacija da se konstrukcija nastavi nakon njegove smrti. Međutim, 1842. godine, talijanski inženjer L. F. Menabrea objavio je sažetak bilješki s predavanja koja mu je držao Babbage (bilješke su bile na francuskom), a slijedeće godine te je bilješke Ada Lovelace — Babbageova učenica, kći lorda Byrona — prevela na engleski jezik. Ada je kasnije postala poznata kao prvi programer u povijesti, na temelju opisa nekih programa za analitički stroj (iako neki tvrde da je to ipak bio sam Babbage) [13].

Analitički stroj imao je tri dijela:

- aritmetičku jedinicu (*mill*)
- podatkovnu memoriju (*store*)
- kontrolnu jedinicu

Tokeni kojima ovaj sustav manipulira su cijeli brojevi, a igraća ploča je podatkovna memorija. Aritmetička jedinica podržava četiri standardne aritmetičke operacije, koristeći određene memorijske lokacije za operande i rezultat. Ovdje aritmetička jedinica predstavlja četiri unutarnja igrača (po jedan za svaku operaciju), a kontrolna jedinica je sudac. Igra koju će sustav igrati određena je unosom korisnika te, unutar svojih (fizičkih) ograničenja, analitički stroj može simulirati bilo koji formalni sustav. Postupak „pretvaranja" ovog stroja opće namjene u specifični automatski formalni sustav prikladnim opisivanjem naziva se *programiranje*. Programi za analitički stroj nisu bili zapisani u podatkovnoj memoriji već su bili kodirani na *bušenim karticama* koje su bile povezane trakom. Te trake nisu služile samo za povezivanje kartica već i za mogućnost ponovnog prolaska kroz iste kartice, odnosno, mogućnost ostvarenja (implementacije) programske petlje.

Prije no što je osmislio Turingov test, Alan Turing je stvorio prvu matematički preciznu teoriju izračunljivosti, uključujući i neke nevjerojatne napretke u teorijskom zasnivanju računala. Ključan je bio izum *Turingovog stroja*, nove arhitekture računala. Postoji beskonačno mnogo različitih Turingovih strojeva od kojih ni jedan nije toliko bitan sam po sebi — bitna je generalna kategorija takvih uređaja i teoremi koje je Turing dokazao. Turingovi strojevi zapravo su vrlo jednostavni i tipično vrlo mali: sastoje se od glave za pisanje i čitanje te trake za pisanje (koja služi kao memorija). Međutim, za razliku od Babbagea koji je htio izgraditi stroj praktičan u primjeni, Turinga su zanimala samo teorijska i apstraktna pitanja. Iz tog razloga Turingovi strojevi su elegantni za opisivanje i dokazivanje teorema, ali vrlo nepraktični za primjenu.

Turingov stroj lako možemo opisati kao automatsku igru. Traka je igraća ploča, a simboli na njoj su tokeni u igri. Unutarnja stanja odgovaraju unutarnjim igračima. Svaki igrač može pročitati trenutni token te, na temelju onoga što pročita, zamijeniti token nekim drugim tokenom, a zatim reći sucu u kojem smjeru da pomakne traku te kojeg igrača da prozove slijedećeg. Kod deterministčnog turingovog stroja, jedini posao suca je da na početku prozove prvog igrača, a zatim radi ono što mu igrači kažu dok netko ne kaže HALT (stani).

Oko 1936. godine Turing je postavio **Turingovu tezu**: Za svaki deterministički automatski formalni sustav, postoji ekvivalentni Turingov stroj.

Također, Turing je pokazao da postoji takozvani *univerzalni Turingov stroj* koji može imitirati rad bilo kojeg Turingovog stroja. Dakle kada bismo imali univerzalni Turingov stroj, on bi mogao obavljati posao bilo kojeg deterministčnog automatskog formalnog sustava.

Dokazano je da postoji mnogo drugih arhitektura za univerzalne formalne sustave, uključujući Babbageov analitički stroj.

Nakon drugog svjetskog rata dolazi do procvata računarstva u SAD-u te se razvija prvo praktično računalo generalne namjene. U ovom projektu sudjelovao je i John von Neumann. Tako je razvijena arhitekutra koju danas nazivamo *von Neumannova arhitektura* iako von Neumann nije samostalno zaslužan za ovaj projekt. Prvotno je bio uključen u projekt ENIAC, koji je u to vrijeme bio najveće računalo na svijetu. Međutim, IAS-ovo (*Institute for Advanced Study*) računalo, koje je bilo manje od ENIAC-a, bilo je jače i jednostavnije za programiranje te se pokazalo kao dobra osnova za budući dizajn računala. Za von Neumannovo računalo, ključna je bila velika memorija kojoj se moglo pristupiti na dva različita načina: relativno i apsolutno. Relativni pristup određuje sljedeću lokaciju u odnosu na trenutnu lokaciju dok apsolutni pristup zadaje točnu specifičnu lokaciju (adresu) njenim imenom (ili brojem). Druga namjena ove velike memorije bila je čuvanje samih programa koje stroj izvodi. Jedna velika prednost von Neumannove arhitekture je apsolutni pristup kodovima već implementiranih subrutina (potprograma) koje veći programi mogu pozvati da se izvrše kao jedan korak (umjesto pisanja tih potprograma svaki puta ispočetka).

Pošto subrutinu pozivamo iz različitih programa, s različitih mjesta u memoriji, najlakše ju je pozvati imenom (adresom), ali problem nastaje kada se treba vratiti na mjesto s kojeg smo je pozvali jer to mjesto ne mora biti isto svaki put. U tu svrhu povratna adresa pri pozivu se zapisuje na za to predodređeno mjesto u memoriji — *stog*. Za razliku od Babbageovog i Turingovog stroja, u von Neumannovom računalu sudac ima složeniju strukturu, čak ima i svoj mali dio memorije (stog) koji služi za pozivanje subrutina i vraćanje iz istih. Ova ideja subrutina zapravo je dovela do razvoja prvih *programskih jezika*. Svaki od njih sastoji se od „knjižnice" pomno odabranih općenito korisnih subrutina koje programer može pozvati kada mu zatrebaju.

LISP

1956. godine John McCarthy predstavlja pojam „umjetne inteligencije" (u GOFAI-smislu) i organizira prvu znanstvenu konferenciju ovog novog polja — *Darthmouth Summer Research Project on Artificial Inteligence*, a 1958. McCarthy i Marvin Minsky su osnovali prvi laboratorij umjetne inteligencije na MIT-u i 1963. McCarthy osniva još jedan laboratorij na Stanfordu. Oba laboratorija smatrali su se svjetskim centrima umjetne inteligencije. Osim toga, McCarthy razvija prvi operativni sustav s mogućnošću *time-sharinga* (jedno centralno računalo podržava više istovremenih neovisnih korisnika) te 1959. godine osmišlja potpuno novu vrstu računala zvanih *LISP* (*LISt Processing*) [9].
McCarthy je inspiriran starijim sustavima nazvanim *IPLs* (*Information Processing Languages*) koje su dizajnirali Allen Newell, Cliff Shaw i Herbert Simon. LISP-ovi, odnosno McCarthyjevi strojevi, kako ih neki zovu, imaju jednostavnu i elegantnu strukturu koja je izrazito fleksibilna, a u isto vrijeme potpuno drugačija od strukture Turingovog ili von Neumannovog stroja. U usporedbi sa svojim prethodnicima LISP se ističe u dva smisla: organizaciji memorije i strukturi kontrole.

Više o povijesti LISP-a možete pronaći u [7].

Memorija

Memoriju obično zamišljamo kao niz praznih posuda i atoma koje stavljamo u njih, po jedan u svaku posudu. Ovako možemo zamisliti razne tipove memorije kao što je memorija Turingovog stroja (po kojoj se možemo micati samo za po jednu posudu ulijevo ili udesno — relativni pristup) ili von Neumannovog računala (možemo odabrati bilo koju posudu u nizu — apsolutni pristup). Primijetimo da spomenute memorije imaju istu organizaciju te se razlikuju samo u načinu pristupa. Njihova struktura je:

- **linearna** (u nizu posuda ne postoje grananja),
- **unitarna** (postoji samo jedan niz posuda) i

- **unaprijed određenog oblika** (niz je uvijek isti).

Međutim, ove karakteristike nisu nužne za računalnu memoriju.

Zamislimo umjesto posuda hrpu takozvanih *Y-spojeva* — uređaja u obliku engleskog slova Y s *utikačem* (bazom) na jednom kraju (donjem) i dvije *utičnice* na preostala dva (gornja). Jedan primjer česte uporabe ovakvih uređaja bilo bi razdvajanje dotoka struje iz zidne utičnice.

Važno je primijetiti da bazu jednog Y-spoja možemo spojiti u utičnicu drugog i da se ovaj proces može nastaviti proizvoljno dugo. Ovakvim spajanjem Y-spojeva dobivamo *stabla*. Svako stablo imat će jednu *izloženu* bazu na dnu, *korijen*, i proizvoljan broj praznih utičnica na vrhu. Zamislimo sada podatke kao ovakva stabla ili *atome* koje možemo spojiti u prazne utičnice drugih stabala poput *listova*. Ovo je osnovna ideja LISP memorije.

Opisane Y-spojeve zvat ćemo *čvorovima*. Atome koji su spojeni u krajnje čvorove stabla također smatramo čvorovima te ih zovemo *terminalnim* čvorovima. U LISP-u ni jedna utičnica ne smije ostati prazna, stoga postoji poseban *atomarni čvor* (možemo ga zamisliti kao list bez podataka) — *NIL* — koji služi kao „praznina" ili „ništa".

LISP specificira da u svakom Y-spoju postoje dvije grane — „lijeva" i „desna" (originalno *CAR* i *CDR*). Zbog toga možemo pronaći (i specificirati) bilo koji čvor u bilo kojem stablu na temelju niza vrijednosti „lijevo" i/ili „desno" od specificiranog početka (korijena). Ovo je nešto između apsolutnog i relativnog pristupa memoriji. Razlikovanje lijeve i desne grane također omogućava implementaciju *liste* kao posebnog stabla u kojem svaki Y-spoj predstavlja jedan *element* liste. Precizno, elementi su spojeni u lijeve grane, tako da je svaki Y-spoj počevši od drugoga spojen u desnu granu prethodnog te je u desnu granu zadnjeg Y-spoja spojen NIL.

Razlike u odnosu na Turingovu i von Neumannovu memoriju su očite:

- LISP-ova memorija **nije linearna**, ali može simulirati linearnu memoriju listama;
- LISP-ova memorija **nije unitarna** (jer može postojati više stabala u istom trenutku)
- LISP-ova memorija **nije unaprijed određenog oblika** (sustav je zamišljen na način da se struktura trenutno korištenog stabla mijenja po potrebi)

Ovo ima dvije bitne posljedice:

1. Stabla nisu uvijek istog oblika, stoga su ti oblici stabala sami po sebi jedan dio podataka u memoriji dok u dotadašnjim tipovima memorije položaj samih spremnika podataka nije sadržavao dodatne informacije.
2. Stabla su građena u poretku u kojem je izvršavanje programa zamišljeno (počevši od korijena) i zbog toga nema potrebe za informacijom o veličini stabla ili o tome kako stablo izgleda „kasnije",odnosno što će program morati napraviti nakon nekoliko koraka.

Kontrola

Kontrola je određivanje tko će kada što raditi.

Primjer 2.5.1. Pretpostavimo da želimo poslužiti doručak: hladni sok od naranče i dva pečena jaja. Prvo ćemo se obratiti „glavnom kuharu", koji u kuhinji predstavlja suca. Kada bi kuhinja bila organizirana po von Neumanovom sustavu, taj sudac bi zvao „specijaliste" (igrače) po rasporedu:

- *NARANČA*
- *OCIJEDI ovu naranču*
- *ČAŠA*
- *OHLADI to što je ocijeđeno*
- *JAJE*
- *JAJE*
- *RAZBIJ ova jaja*
- *TAVA*
- *ISPECI ova jaja u ovoj tavi*
- *POSLUŽI ovo što je ohlađeno i ovo što je ispečeno*

Ovdje velika slova označavaju kojeg se „specijalista" poziva. Redoslijed izvašanja je linearan i ovisi isključivo o redoslijedu pisanja poziva (odnosno naredbi).

Što se događa kada je kuhinja organizirana kao McCarthyjeva memorija? Iste primitivne akcije bit će obavljene u istom redoslijedu, ali su organizirane „sa suprotne strane". McCarthyjev sudac uvijek će započeti traženim rezultatom i tražiti što mu je potrebno kako bi ga ostvario. Dakle, rekao bi POSLUŽI *hladni sok od naranče i dva pečena jaja* a zatim bi čekao da vidi da li je još nešto potrebno da bi zadatak bio izvršen. U ovom slučaju potrebna su mu *dva pečena jaja od „specijalista"* ISPECI i *ohlađeni sok od naranče od „specijalista"* OHLADI pa poziva ove specijaliste i čeka treba li im nešto kako bi izvršili svoje zadatke. Pozivanje se tako provodi dok god se ne zaustavi na nekom „specijalistu" koji nema dodatne zahtjeve. Kada se to dogodi, sudac može početi ispunjavati sve zahtjeve koje je do sada imao. Na kraju će *„specijalist"* POSLUŽI imati sve što mu je potrebno da stvarno posluži doručak. Ovo je centralna ideja kontrole u LISP-u.

Matematički rečeno, „specijaliste" koji međudjeluju na ovaj način zovemo *funkcijama*, njihove zahtjeve (ako ih ima) zovemo *argumentma*, a rezultate koje vraćaju nazivamo *povratnim vrijednostima*. Funkcije koje ne zahtijevaju argumente nazivamo *konstantama* (uvijek vraćaju istu vrijednost). Argumenti i povratna vrijednost funkcije mogu biti bilo što, dok je god ispunjen sljedeći uvjet:

- **funkcionalnost**: za svaki dopustivi argument (ili kombinaciju argumenata), postoji jedinstvena (pridružena) vrijednost

Svi programi u LISP-u su funkcije, ali razlikuju se od klasičnih matematičkih funkcija u dva aspekta:

- oni su aktivni „igrači" koji automatski, pri pozivu, evaluiraju svoju vrijednost te
- njihovi (neinterpretirani) argumenti moraju biti LISP-ova stabla

Kada funkcija u LISP-u primi argument, ona pretpostavlja da je taj argument druga funkcija koju također treba pozvati, a ukoliko nema argumente onda je konstanta pa vraća svoju (konstantnu) vrijednost.

Kako bismo programirali LISP stroj trebamo *definirati* željene funkcije uz pomoć već postojećih (definiranih) funkcija (kao što je na primjer kvadriranje definirano pomoću množenja). Definirane funkcije vraćaju vrijednosti za dane argumente, ali jedino što zapravo naprave je da kažu sucu koga idućeg treba pozvati. Očito je da ne možemo sve funkcije definirati uz pomoć nekih drugih; mora postojati skup *primitivnih funkcija* od kojih možemo početi. U LISP-u postoji samo 6 takvih funkcija, od kojih se sve ostale mogu izgraditi. To su:

LEFT$[x]$	RIGHT$[x]$	JOIN$[x, y]$
EQUAL$[x, y]$	ATOM$[x]$	IF$[x, y, z]$

Svaka od ovih primitivnih funkcija za argumente prima samo LISP-ova stabla i vraća samo LISP-ova stabla kao vrijednosti. Uz pomoć ovih funkcija moguće je transformirati (izgraditi, modificirati, uništiti) oblik LISP-ovog stabla proizvoljne složenosti, na bilo koji, algoritamski prikaziv, način te je moguće definirati složenu funkciju koja će vraćati takve transformacije kao vrijednosti, za bilo koji ulaz, zbog čega je LISP univerzalan. Ove funkcije izvorno su bile malo drugačije, više u [9].

Prve tri od ovih 6 primitivnih funkcija koriste se za stvaranje (ili uništavanje) proizvoljnih stabala. Vrijednost funkcije LEFT$[x]$ je ono što je spojeno u lijevu utičnicu Y-spoja na samoj bazi od x, dakle, vraća lijevu stranu svojeg arumenta, a ukoliko x nema lijevu stranu funkcija LEFT$[x]$ prijavit će grešku. Funkcija RIGHT$[x]$ analogna je funkciji LEFT$[x]$, ali umjesto lijeve strane vraća desnu. JOIN$[x, y]$ uzima dva argumenta i povezuje ih Y-spojem;

x spaja u lijevu stranu, Y u desnu, a vraća korijen novostvorenog stabla. Preostale tri funkcije su LISP-ove alternative za testiranje i grananje. Funkcija EQUAL$[x, y]$ vraća vrijednost TRUE ako su vrijednosti x-a i Y-a jednake, a u suprotnom vraća FALSE. ATOM$[x]$ vraća TRUE ukoliko je (vrijednost) x samo jedan atomarni čvor, a inače vraća FALSE. Funkcija IF$[x, y, z]$ vraća vrijednost svog drugog ili trećeg argumenta, ovisno o tome je li vrijednost prvog argumenta TRUE (vraća drugi argument) ili FALSE (vraća treći argument). Bitno je primijetiti da se u LISP-u programi i strukture podataka prikazuju na potpuno isti način — u obliku stabla.

Poglavlje 3

Duboko učenje

Većina informacija u ovom poglavlju uzeta je iz [11]. *Strojno učenje* je polje u umjetnoj inteligenciji i kognitivnoj znanosti. U kontekstu UI podijeljeno je u tri glavne grane:

- nadzirano učenje (*supervised learning*)
- nenadzirano učenje (*unsupervised learning*)
- podržano učenje (*reinforcement learning*)

Duboko učenje je poseban pristup u strojnom učenju koji, uz to što pokriva tri spomenute grane, proširuje strojno učenje na razne probleme umjetne inteligencije kao što su reprezentacija znanja, planiranje, rasuđivanje i slično te u svom radu koristi *duboke* neuronske mreže. Općenito se bavi problemima koji nastaju pri dodavanju više slojeva *plitkoj* neuronskoj mreži.

3.1 Nadzirano učenje

Nadziranim učenjem mogu se rješavati problemi *klasifikacije* i *regresije*. Kod klasifikacije primjeru pridružujemo *klasu*, a kod regresije neku numeričku vrijednost. Razlika je u tome je li ciljana varijabla diskretna ili kontinuirana.

Klasifikacija

Cilj klasifikacije je odrediti klasu C kojoj *primjerak* $\mathbf{x}$ pripada. Primjerak je definiran kao vektor *značajki* $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$, koji možemo interpretirati kao točku u n-dimenzionalnom vektorskom prostoru koji nazivamo *ulazni prostor* (*input space)* ili *prostor primjeraka* (*instance space*). Značajke mogu biti *numeričke*, *ordinalne* ili *kategoričke*. Numeričke značajke su realni brojevi (uređene su i možemo njima računati), ordinalne značajke

su slične rednim brojevima (samo su uređene), a kategoričke značajke mogu biti neke apstraktne oznake, riječi i slično (nemaju ni jedno od dva svojstva numeričkih značajki). Ako značajke nisu numeričke, prije računanja ih moramo prilagoditi algoritmima strojnog učenja. Obično se to radi metodom *one-hot-encoding*. Umjesto jedne značajke imali bismo onoliko značajki koliko ima različitih vrijednosti originalne značajke, a vrijednosti novih značajki bile bi binarne znamenke (0 ili 1).
Neka je $\mathcal{X}$ skup svih mogućih primjeraka. Pretpostavka svih algoritama strojnog učenja je da su primjerci iz $\mathcal{X}$ uzorkovani nezavisno i iz zajedničke distribucije (*iid — independent and identically distributed*). Kod nadziranog učenja unaprijed nam je poznata oznaka klase Y kojoj pripada primjerak, a zovemo ju *ciljna oznaka*. Ako se ograničimo na *binarnu klasifikaciju*, odnosno podjelu na samo dvije klase, tada je $y \in \{0, 1\}$, gdje $y = 1$ označava da primjerak pripada klasi, a $y = 0$ suprotno. *Skup primjeraka za učenje* $\mathcal{D}$ sastoji se od uređenih parova primjeraka i pripadnih oznaka, $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, gdje je N ukupan broj primjeraka za učenje. Skup $\mathcal{D}$ možemo prikazati matrično:

$$\left[\begin{array}{ccc|c} x_1^{(1)} & \cdots & x_n^{(1)} & y^{(1)} \\ \vdots & \ddots & \vdots & \vdots \\ x_1^{(N)} & \cdots & x_n^{(N)} & y^{(N)} \end{array}\right].$$

Zadaća klasifikatora je izabrati *hipotezu* $h : \mathcal{X} \to \{0, 1\}$ koja određuje pripada li primjer $\mathbf{x}$ klasi C ili ne:

$$h(\mathbf{x}) = \begin{cases} 1, & \mathbf{x} \text{ pripada klasi } C \\ 0, & \mathbf{x} \text{ ne pripada klasi } C \end{cases}$$

Skup svih mogućih hipoteza (na primjer samo hiperravnine) $\mathcal{H}$ nazivamo *model* ili *prostor hipoteza*. Kažemo da primjerak $\mathbf{x} \in C$ *zadovoljava* hipotezu $h \in \mathcal{H}$ ako i samo ako je $h(\mathbf{x}) = 1$. Kažemo da je hipoteza konzistentna s primjerom za učenje $(\mathbf{x}, t)$ ako i samo ako je $h(\mathbf{x}) = y$.

Učenje se svodi na nalaženje najbolje hipoteze $h \in \mathcal{H}$. U većini slučajeva hipotezu ćemo zapisati parametarski, u obliku vektora θ. Najbolja hipoteza je ona koja najtočnije klasificira primjere. Koliko dobro hipoteza h klasificira primjere za učenje iskazuje *pogreška učenja* (*training error*) $E(h|\mathcal{D})$ koju dobivamo kao očekivanje *funkcije gubitka* L na primjercima iz skupa za učenje (više u odjeljku 3.3). Hipoteza je *konzistentna* s primjercima za učenje ako i samo ako $E(h|\mathcal{D}) = 0$. Moguće je da takva hipoteza ne postoji, odnosno za svaki $h \in \mathcal{H}$ je $E(h|\mathcal{D}) > 0$. Tada kažemo da $\mathcal{H}$ nije dovoljnog *kapaciteta* ili *složenosti*.

Jedan način iskazivanja kapaciteta modela je *Vapnik–Chervonenkisova dimenzija* — najveći broj primjerkaa koje model $\mathcal{H}$ može razdvojiti, a označava se s $VC(\mathcal{H})$. Može se pokazati da hiperravnina u prostoru $\mathbb{R}^n$ može (u svakom slučaju) klasificirati najviše $n + 1$ točaka. VC-dimenzija je prilično pesimistična ocjena kapaciteta jer ne uzima u obzir distribuciju primjeraka.

Ako se radi o *višeklasnoj klasifikaciji* (*multiclass classification*) pogodno je ciljne oznake zapisati u obliku $\mathbf{y}^{(i)} = (y_1^{(i)}, \ldots, y_K^{(i)})^T$, gdje je K broj klasa, a $y_j^{(i)} = 1$ ako i samo ako $\mathbf{x}^{(i)}$ pripada klasi C_j. Ovdje smatramo da postoji jedinstveni takav j. U općenitijem slučaju $\mathbf{x}$ može pripadati u više klasa istovremeno. To nazivamo *klasifikacija s višestrukim oznakama* (*multilabel classification*) ili *klasifikacija jedan-na-više*.

Hipotezu nećemo tražiti na slijepo. Skup pretpostavki koji nam omogućuje induktivno učenje nazivamo *induktivna pristranost*, a temeljem kojih klasifikacija (novog) primjera slijedi deduktivno. Razlikujemo dvije vrste induktivne pristranosti:

- *pristranost ograničavanjem* ili *pristranost jezika* (*restriction bias*, *language bias*) — odabirom modela $\mathcal{H}$ ograničavamo skup hipoteza
- *pristranost preferencijom* ili *pristranost pretraživanja* (*preference bias*, *search bias*) — definiranjem načina pretraživanja hipoteza unutar $\mathcal{H}$ dajemo prednost jednim hipotezama u odnosu na druge

Većina algoritama učenja koristi kombinaciju ovih dvaju tipova pristranosti.

Šum je neželjena anomalija u podacima. Mogući uzroci su: neprecizno mjerenje, pogreške u označavanju (*teacher noise*), postojanje skrivenih značajki, pojedinačni primjerci koji znatno odskaču od većine (*outliers*). Kada postoji šum, granica između klasa nije jasna te nije moguće odvojiti šum od pravih podataka. Zbog toga jednostavni modeli ne mogu ostvariti $E(h|\mathcal{D}) = 0$.

Odabir modela svodi se na optimizaciju *parametara* — vektora θ. Što je kapacitet modela veći to je veća šansa da smanjimo pogrešku, no moramo paziti jer svrha hipoteze nije da klasificira primjerke za učenje već nove, buduće primjerke, odnosno da ima svojstvo *generalizacije*. Zapravo preferiramo jednostavnije modele jer se bolje poopćavaju, lakše ih je koristiti i tumačiti. Ovo načelo u filozofiji znanosti poznato je kao *Occamova britva* (*Occam's razor*). Ovdje susrećemo dvije krajnosti:

- *prenaučenost* (*overfitting*) — model je previše složen u odnosu na modeliranu funkciju, dobivamo hipoteze koje pretpostavljaju više no što postoji u podacima, uče šum
- *podnaučenost* (*underfitting*) — model je prejednostavan u odnosu na modeliranu funkciju, loše opisuje i same podatke iz skupa za učenje

Jednostavan model ima malu varijancu, a veću statističku *pristranost*, u smislu odstupanja prosjeka (ili očekivanja) od ciljne vrijednosti. Što je induktivna pristranost bolja to je statistička pristranost manja. Optimalan model je onaj koji minimizira i pristranost i varijancu (*bias-variance dilemma*) te na taj način daje najbolju generalizaciju. **Pretpostavka induktivnog učenja** kaže: Ako je pogreška hipoteze na dovoljno velikom skupu primjera

za učenje mala i ako model nije suviše složen, hipoteza će dobro klasificirati nove, slične primjere. Ovdje je važno primijetiti da novi primjeri trebaju biti slični (iz iste distribucije) primjerima za učenje.

Regresija

Kod regresije je ciljna vrijednost $y \in \mathbb{R}$. Na temelju skupa primjeraka za učenje $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_i$ potrebno je odrediti nepoznatu funkciju $f : \mathcal{X} \to \mathbb{R}$ koja procjenjuje izlazne vrijednosti. U idealnom slučaju želimo $y^{(i)} = f(\mathbf{x}^{(i)})$. Međutim, zbog prisutstva šuma, zapravo učimo funkciju $y^{(i)} = f(\mathbf{x}^{(i)}) + \varepsilon$, gdje je ε slučajni šum. Regresijom dobivamo funkciju h, hipotezu, kao procjenu funkcije f. Najjednostavniji oblik funkcije pogreške je:

$$E(h|\mathcal{D}) = \frac{1}{2} \sum_{i=0}^{N} \left(y^{(i)} - h(\mathbf{x}^{(i)}) \right)^2 .$$

Faktor $\frac{1}{2}$ ovdje je uvršten zbog jednostavnosti računanja. Za model odabiremo linearan model

$$h(\mathbf{x}) = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + w_0 = \sum_{i=0}^{n} w_i x_i + w_0 = \mathbf{w}^T \mathbf{x} + w_0,$$

gdje su w_i parametri koje treba naučiti na temelju skupa primjeraka $\mathcal{D}$. Ovu vrstu regresije nazivamo *linearna regresija* jer vrijednost $h(\mathbf{x})$ linearno ovisi o ulaznim vrijednostima $\mathbf{x}$ (ovo je zapravo afina funkcija, ali postaje linearna ako $\mathbf{x}$ proširimo jedinicom). Cilj nam je pronaći hipotezu koja minimizira pogrešku. Hipoteza je određena parametrima $w_0, \ldots, w_n$. U slučaju jednodimenzionalne linearne regresije ($\mathcal{X} = \mathbb{R}$) i u posebnom slučaju poopćenja linearnog modela — polinomnoj regresiji, rješenje postoji u zatvorenoj formi. U slučajevima kada analitičko rješenje ne postoji okrećemo se iterativnim optimizacijskim metodama.

Unakrsna provjera i ocjena izbora modela

Unakrsna provjera (*cross-validation*) je jednostavni način da se ocijeni izbor modela. Skup primjeraka $\mathcal{D}$ razdvajamo na dva dijela: *skup za učenje* (*training set*) i *skup za provjeru* (*validation set*). Model učimo na skupu za učenje, a provjeravamo na skupu za provjeru. Na ovaj način možemo procijeniti kako će se model ponašati na novim primjerima. Pogreška koja se dobiva na podacima iz skupa za provjeru naziva se *pogreška generalizacije* (*off-training-set error*), a pogreška na skupu za učenje naziva se *pogreška učenja*. Ako se unakrsna provjera koristi za utvrđivanje konačne pogreške već odabranog modela tada se skup na kojem se ta pogreška mjeri naziva *ispitni skup* (*test set*), a pogreška *ispitna pogreška*.

Skup za učenje, skup za provjeru i ispitni skup moraju biti međusobno disjunktni — inače ne možemo ispravno odrediti pogrešku generalizacije.

Neke mjere ocjene izbora modela su *točnost* (*accuracy*), *preciznost* (*precision*) i *odziv* (*recall*). Kažemo da je primjer *pozitivan* za klasu C ako joj pripada, a *negativan* ako joj ne pripada. Ovisno o klasifikatoru i klasi C, primjere iz skupa za učenje možemo razvrstati u četiri skupine:

- *true positive* (*TP*) — klasifikator **je** svrstao ovaj podatak u klasu C i podatak stvarno **jest** iz klase C;
- *false positive* (*FP*) — klasifikator **je** svrstao ovaj podatak u klasu C a podatak **nije** iz klase C;
- *true negative* (*TN*) — klasifikator **nije** ovaj podatak svrstao u klasu C i podatak stvarno **nije** iz klase C;
- *false negative* (*FN*) — klasifikator **nije** ovaj podatak svrstao u klasu C a podatak **jest** iz klase C.

Ako označimo s TP, FP, TN, FN broj primjera u pojedinoj skupini dobivamo:

- **točnost** — koliko je klasifikator dobar u sortiranju:

$$\mathrm{Acc} = \frac{\mathrm{TP} + \mathrm{TN}}{N}, \quad N = \mathrm{TP} + \mathrm{FP} + \mathrm{TN} + \mathrm{FN}$$

- **preciznost** — koliko je klasifikator dobar u izbjegavanju lažnih pozitivnih vrijednosti:

$$\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}},$$

- **odziv** — koliko smo pozitivnih primjera uspjeli „uhvatiti”:

$$\mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}.$$

Standardan način prikaza podataka o broju TP, FP, TN, FN je tablica koju zovemo *matrica zabune* (*confusion matrix*). Za binarnu klasifikaciju matrica zabune bila bi tablica veličine 2×2 oblika 3.1. Kada imamo matricu zabune, iz nje lako možemo pročitati točnost, preciznost, odziv ili bilo koju drugu ocjenu. Ovakve ocjene poprimaju vrijednosti iz intervala $[0, 1]$ i zbog toga ih možemo protumačiti kao vjerojatnosti. Primijetimo da postoje trivijalni načini da ili preciznost ili odziv poprime vrijednost 1, ali ne oba u isto vrijeme (za preciznost klasifikator napravimo tako da ne smatra ni jednu vrijednost pozitivnom, a obrnuto za odziv. Zato su nam potrebne sve tri ocjene kako bismo dobili značajan uvid u kvalitetu klasifikatora.

	Klasifikator kaže **da**	Klasifikator kaže **ne**
Stvarno **da**	TP	FN
Stvarno **ne**	FN	TN

Tablica 3.1: Matrica zabune za binarnu klasifikaciju

Pristupi nadziranom učenju

Postupke nadziranog učenja možemo podijeliti s obzirom na način modeliranja na:

- *generativne* modele
  - pretpostavljaju da je vjerojatnost $P(C|\mathbf{x})$ proporcionalna vjerojatnosti presjeka $P(\mathbf{x}, C)$;
  - modeliraju zajedničku vjerojatnost i temeljem nje, primjenom Bayesovog teorema, računaju vjerojatnost da primjer $\mathbf{x}$ pripada klasi C;
  - modeliraju generiranje samih podataka te se mogu koristiti i za generiranje sintetičkih primjera u ulaznom prostoru, uzorkovanjem iz zajedničke distribucije;
  - ponekad se u literaturi nazivaju *zajednički modeli* (*joint models*).
- *diskriminativne* modele ili *metode temeljene na granici*
  - izravno modeliraju pripadnost primjera $\mathbf{x}$ klasi C
  - Oni mogu biti:
    - *probabilistički* ili *uvjetni* modeli (*conditional models*) — izravno modeliraju vjerojatnost $P(C|\mathbf{x})$ i
    - *neprobabilistički* modeli ili *diskriminacijske funkcije* — modeliraju funkciju $h(\mathbf{x})$ koja primjeru $\mathbf{x}$ izravno dodjeljuje oznaku klase C.

Kod probabilističkih pristupa klasifikacija se odvija u dvije faze: faza *zaključivanja* i faza *odlučivanja*. U fazi zaključivanja računamo $P(C|\mathbf{x})$ uz pomoć primjera za učenje, a u fazi odlučivanja koristimo istu vjerojatnost kako bismo klasificirali nove podatke. Kod neprobabilističkih modela ova dva koraka rade se odjednom. Iako generativni modeli imaju mnogo prednosti, za neke su probleme nepotrebno složeni.

Još jedna podjela nadziranih postupaka odnosi se na broj primjera za učenje i broj parametara modela. Ovdje se modeli dijele na

- *parametarske* — složenost modela ne ovisi o broju primjera za učenje, odnosno, probabilistički parametarski postupci pretpostavljaju da se podaci pokoravaju nekoj teorijskoj razdiobi; učenje se svodi na traženje parametara pretpostavljene distribucije

	Generativni	**Diskriminativni**
Parametarski	• Bayesov klasifikator • Bayesove mreže • latentna Dirichletova alokacija • skriven Markovljev model	• logistička regresija • perceptron • višeslojni perceptron • stroj s potpornim vektorima (primarna formulacija) • linearna diskriminantna analiza
Neparametarski		• model k-najbližih susjeda • stabla odluke • klasifikacijska pravila • stroj s potpornim vektorima (dualna formulacija)

Tablica 3.2: Podjela nadziranih modela

- *neparametarske* — broj parametara raste s brojem primjera za učenje; ovdje ne pretpostavljamo da se podaci za učenje pokoravaju nekoj teorijskoj distribuciji; ovi modeli također imaju parametre, ali to nisu parametri neke pretpostavljene distribucije.

U tablici 3.2 možemo vidjeti neke primjere modela razvrstane prema ovoj podjeli.

Konačno, modele nadziranog učenja možemo podijeliti s obzirom na granicu kojom razdvajaju pozitivne primjere od negativnih (klasifikacija) ili krivulju kojom aproksimiraju modeliranu funkciju (regresija). Postoje:

- *linearni modeli*
  - povlače linearnu granicu između klasa (funkciju aproksimiraju linearnim modelom)
  - granica je pravac, ravnina ili hiperravnina, ovisno o dimenziji prostora $\mathcal{X}$ (ako je $\dim \mathcal{X} = n$, tada je dimenzija hiperravnine $n - 1$)
  - neki primjeri linearnih modela su: naivni Bayes, logistička regresija, perceptron, stroj s potpornim vektorima (SVM)
- *nelinearni modeli*
  - granica koju povlače je nelinearna (funkciju aproksimiraju nelinearnom hiperravninom)
  - neki primjeri nelinearnih modela su: model k-najbližih susjeda, model stabla odluke, model višeslojnog perceptrona, SVM s jezgrenim funkcijama i polinomijalna regresija

Nelinearni modeli očito imaju veći kapacitet (VC-dimenziju) te zato mogu riješiti neke probleme koje linearni modeli ne mogu (npr. XOR). U praksi teško nalazimo linearno razdvojive probleme (koji su zanimljivi). Međutim, ako primjeraka nije mnogo više nego što je dimenzija ulaznog prostora, velika je vjerojatnost da je problem linearno razdvojiv. Ova činjenica daje nam podlogu za takozvane *jezgrene metode* (*kernel methods*) gdje se povećanjem dimenzije ulaznog prostora postiže linearna razdvojivost.

3.2 Nenadzirano i podržano učenje

Nenadzirano učenje čine pristupi kod kojih je skup podataka za učenje oblika $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$, odnosno nemamo zadane ciljne oznake. U nenadzirano učenje spadaju postupci poput *grupiranja* (*clustering*), otkrivanja stršećih ili novih vrijednosti (*outlier detection*, *novelty detection*), smanjenja dimenzionalnosti (*dimensionality reduction*) ili postupci otkrivanja veza među uzorcima (*discovering graph structure*). Ovisno o tome što znamo o podacima, zadaća grupiranja može biti razdijeliti uzorke u već poznate razrede ili pak odrediti i same razrede u koje ćemo razvrstati podatke. Podatke dijelimo u razrede na temelju njihove međusobne udaljenosti (u odnosu na neku odabranu metriku). Zadatak postupaka smanjenja dimenzionalnosti je smanjiti broj značajki koje proučavamo. Na primjer, ako u trodimenzionalnom prostoru imamo točke koje se približno nalaze u istoj ravnini, želimo ih prikazati u dvije dimenzije zbog jednostavnosti računa (uz mali gubitak bitnih podataka).

Podržano učenje predstavlja dio strojnog učenja koji se bavi optimizacijom ponašanja — proučavamo interakciju agenta i okoline. Agent na temelju informacija iz okoline obavlja akcije i kao odgovor za svaku akciju dobije nagradu ili kaznu. Zadaća podržanog učenja je otkriti optimalnu strategiju ponašanja agenta, tako da agent maksimizira dugotrajnu dobit. Više o podržanom učenju u [15].

U ovom se radu nećemo baviti ovim oblicima učenja.

3.3 Neuronske mreže

Većina informacija u ovom poglavlju uzeta je iz [14] i [16]. Svaka neuronska mreža sačinjena je od jednostavnih elemenata: *neurona* i *veza* između njih. Svaka neuronska mreža sadrži *ulazni* i *izlazni sloj* neurona. Između ova dva sloja može se nalaziti *skriveni sloj* ili više njih, koji se sastoji od „radnih" neurona. S obzirom na broj slojeva i njihovu međusobnu povezanost možemo razdvojiti četiri glavne vrste neuronskih mreža. To su:

- plitke neuronske mreže bez povratnih veza (*simple feedforward neural networks*) — 2 ili 3 sloja

- duboke neuronske mreže bez povratnih veza (*multi-layer feedforward networks*) — više od tri sloja
- mreže s povratnim vezama (*recurrent networks*)
- ljestvičaste mreže (*lattice structures*)

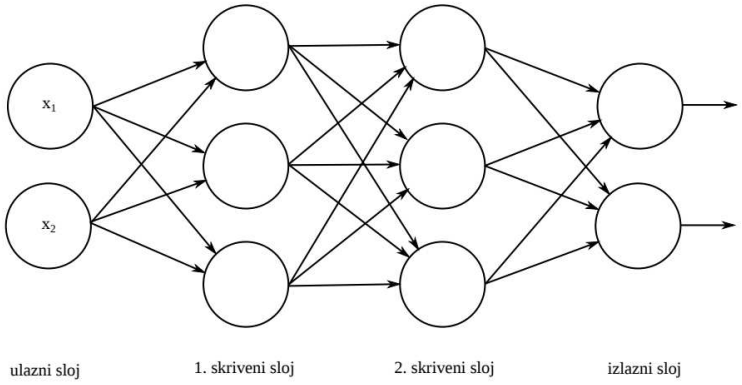
Neuroni u uzastopnim slojevima mogu biti povezani, a svaki neuron mora biti povezan bar s još jednim neuronom. Neuronska mreža može biti *potpuno* ili *djelomično* povezana. Potpuno je povezana ako je svaki neuron u svakom sloju povezan sa svim neuronima u sljedećem sloju, a inače je djelomično povezana. Svaka veza između dva neurona ima *težinu* — vrijednost kojom će se pomnožiti izlaz prvog neurona prije no što se pošalje drugom neuronu. Težinu koja pripada vezi između neurona j u sloju $n-1$ i neurona m u sloju n označit ćemo s $w_{jm}^{(n)}$. Prije početka rada, težine treba postaviti na početne vrijednosti. Jedna preporuka je postaviti težine na broj u okolini nule, a postoji i pravilo koje daje dobre rezultate a kaže da za neuron koji prima ulaze od k drugih neurona sve težine treba postaviti na nasumične vrijednosti iz intervala $[-2.4/k, 2.4/k]$. Na početku rada ulaznim neuronima šaljemo ulazne podatke. Ulaznih podataka može biti najviše onoliko koliko ima ulaznih neurona, a ako ih ima manje, preostalim neuronima šalje se vrijednost 0. Nakon ulaznog sloja tok informacija ide iz sloja u sloj (s lijeva na desno) na način da svaki neuron šalje svoju izlaznu vrijednost svim vezama koje idu iz njega; ta vrijednost se pomnoži odgovarajućom težinom te se šalje drugom neuronu. Ulazna vrijednost pojedinog neurona naziva se *logit*, a obično ju označavamo sa z. Ova vrijednost računa se po formuli $z = b + w_1x_1 + w_2x_2 + w_3x_3 \ldots w_nx_n$, gdje su x_i primljeni ulazni podaci, w_i pripadajuće težine, a b je takozvana *pristranost* (*bias*, možemo je smatrati samo posebnom težinom koja ne pripada vezi između neurona već je definirana za svaki neuron). Neki jednostavniji modeli neuronskih mreža kao izlaz daju direktno logit, ali većina, prije samog izlaza, na logit primjenjuje i nelinearnu funkciju koju zovemo *aktivacijska funkcija*, a označavamo ju sa s. Svaki sloj neuronske mreže može imati svoju aktivacijsku funkciju, ali unutar jednog sloja aktivacijska funkcija mora biti ista. Na kraju dobivamo izlazne podatke u izlaznim neuronima. Jedan primjer shematskog prikaza neuronske mreže možemo vidjeti na slici 3.1.

Neke aktivacijske funkcije

Step-funkcija

$$s(x) = \begin{cases} 1, & x \geq \theta \\ 0, & \text{inače} \end{cases}, \tag{3.1}$$

gdje je $\theta \in \mathbb{R}$. Ako koristimo ovu funkciju u neuronskoj mreži koja je sačinjena od samo jednog neurona dobivamo takozvani *TLU-perceptron* (*Threshold Logic Unit*). Ova



Slika 3.1: Shematski prikaz umjetne neuronske mreže [14]

funkcija često se koristi u izlaznom sloju binarne klasifikacije. U ovu svrhu prvi su ju koristili McCulloch i Pitts u svojoj definiciji prve neuronske mreže. Ako u neuronskoj mreži koristimo samo ovu funkciju, možemo njome razdvojiti samo linearno odvojive klase. Još jedna mana je nemogućnost postupaka učenja koji koriste derivacije (na cijelom $\mathbb{R}$ derivacija je 0 osim u θ gdje nije definirana).

Sigmoidalna ili logistička funkcija

$$\sigma(x) = \frac{1}{1 + e^{-ax}}, \tag{3.2}$$

gdje je $a \in \mathbb{R}$ (određuje nagib funkcije). Najčešće se koristi $a = 1$. Ova funkcija često se koristila u neuronskim mrežama u 90-im godinama prošlog stoljeća. Pogodna je za analizu jer je derivabilna u svim točkama.

Tangens hiperbolni

$$\text{th}(x) = \text{th}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{3.3}$$

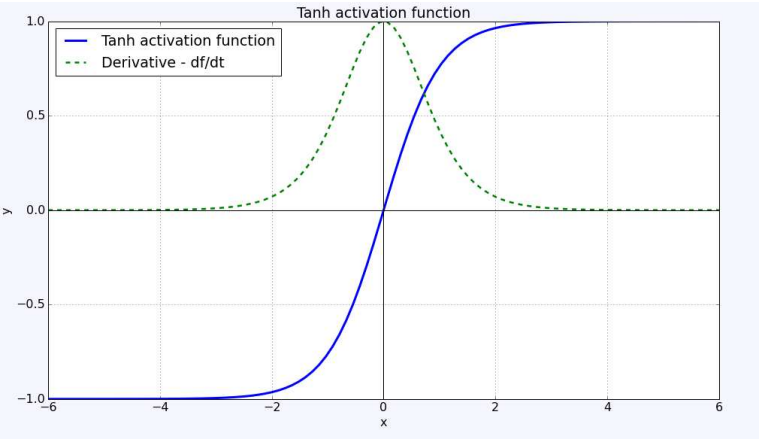
Tangens hiperbolni ne razlikuje se bitno od sigmoidalne funkcije. Naime, vrijedi:

$$\text{th}(x) = 2\sigma(2x) - 1.$$

Tangens hiperbolni često je pokazivao bolje rezultate od sigmoidalne funkcije te se krajem prošlog stoljeća počeo češće upotrebljavati.

Obje ove funkcije sve se češće koriste samo na izlaznom sloju zbog problema *nestajanja gradijenta* (*vanishing gradient problem*) koji se javlja prilikom korištenja algoritma za *backpropagation*.

Grafički prikaz može se vidjeti na slici 3.2.



Slika 3.2: Graf funkcije tangens hiperbolni i njene derivacije [6]

Zglobnica ili ReLU — *Rectified Linear Unit*

$$\mathrm{ReLU}(x) = \max 0, x. \tag{3.4}$$

Ova funkcija predstavljena je 2000. godine s biološkom motivacijom, a 2011. prvi je puta demonstrirano da njeno korištenje u dubokim neuronskim mrežama može poboljšati učenje [6]. Neke od prednosti su lako računanje gradijenta, te linearnost po dijelovima. Manu predstavlja vrijednost gradijenta 0 za negativne ulaze, što može uzrokovati da neke težine stagniraju (u tim čvorovima mreža više ne uči). Iz tog razloga napravljena je i modifikacija, takozvana *parametrizirana zglobnica*:

$$\mathrm{ReLU}(x) = \begin{cases} x, & x > 0 \\ \alpha x, & \text{inače} \end{cases}, \tag{3.5}$$

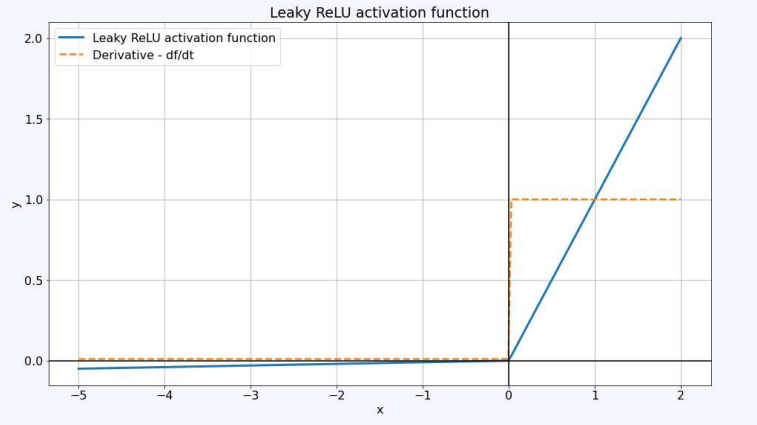
gdje je α mali pozitivan broj. Posebno, ako je $\alpha = 0.01$ zove se *leaky ReLU* (*ℓReLU*). Funkcija ℓReLU se preporučuje kao aktivacijska funkcija u skrivenim slojevima dubokih neuronskih mreža.

Grafički prikaz funkcije ℓReLU može se vidjeti na slici 3.3.

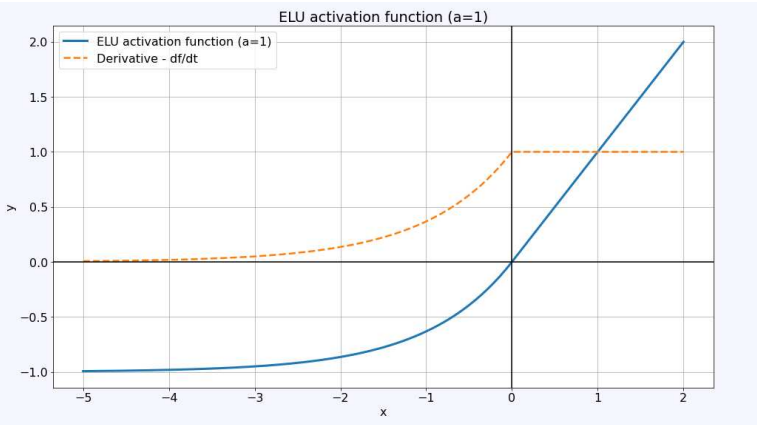
Exponential Linear Unit — ELU

$$\mathrm{ELU}(x) = \begin{cases} x, & x > 0 \\ \alpha \left(e^x - 1\right), & x < 0 \end{cases}, \tag{3.6}$$

Moglo bi se reći da je ovo najnoviji oblik funkcije ReLU. [1] Grafički prikaz funkcije ELU i njene derivacije može se vidjeti na slici 3.4.



Slika 3.3: Grafički prikaz parametrizirane funkcije ReLU i njene derivacije [6]



Slika 3.4: Grafički prikaz funkcije ELU i njene derivacije [6]

Prikaz podataka

Neuronskoj mreži primjerke za učenje šajemo u obliku vektora $\mathbf{x}^{(i)} = (x_1^{(i)}, \ldots, x_d^{(i)})^T$, $\mathbf{x}^{(i)} \in \mathcal{X}$ gdje su $x_1^{(i)}, \ldots, x_d^{(i)}$ vrijednosti pojedinih značajki i-tog primjera, $i \in \{1, \ldots, n\}$, a n je ukupan broj primjera za učenje. U slučaju nadziranog učenja bit će nam poznata i ciljna oznaka $\mathbf{y} = (y_1^{(i)}, \ldots, y_K^{(i)})$ (*expected value* ili *true label*) čiji su elementi obično znamenke 0 ili 1 (mogu biti i oznake klasa), a K je broj izlaznih neurona. Svaka značajka $x_j^{(i)}$ bit će poslana jednom neuronu u ulaznom sloju. Radi jednostavnijeg računanja, ulazne podatke prikazat ćemo jednostupčanom *ulaznom matricom* $\mathbf{x}^{(i)}$. Skup svih primjeraka za učenje s pripadajućim ciljnim oznakama možemo označiti s $\mathcal{D}$ kao u odjeljku 3.1.

Željeli bismo prikazati i težine i pristranosti u obliku matrice, ali zbog dimenzionalnosti pristranost nam smeta. Kako bismo riješili ovaj problem, radimo *apsorpciju pristranosti* — pristranost smatramo dodatnom težinom (nekada se označava w_0) a tada nam treba i vrijednost koju ćemo tom težinom pomnožiti u matrici ulaznih vrijednosti pa definiramo $x_0 = 1$ te dodajemo taj element u matricu ulaznih vrijednosti. Tada, ako je neuronska mreža potpuno povezana, težine i pristranosti za svaki sloj neuronske mreže možemo čuvati u matrici. Matricu težina između slojeva $l-1$ i l označavamo s $\mathbf{w}^{(l)}$, za svaki $l \in \{2, \ldots, L\}$, gdje je L broj slojeva u mreži. Ako s N_l označimo broj neurona u sloju l, matrica težina bila bi oblika:

$$\mathbf{w}^{(l)} = \begin{bmatrix} b_1^{(l)} & w_{11}^{(l)} & \cdots & w_{N_{l-1}1}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ b_{N_l}^{(l)} & w_{1N_l}^{(l)} & \cdots & w_{N_{l-1}N_l}^{(l)} \end{bmatrix},$$

gdje je $b_i^{(l)}$ pristranost i-tog neurona u l-tom sloju. Tada bi ulazna matrica izgledala ovako:

$$\mathbf{x}^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \\ \vdots \\ x_{N_1}^{(i)} \end{bmatrix}.$$

Logit $\mathbf{z}_i^{(l)}$ je ulaz, a *aktivacija* $\mathbf{a}_i^{(l)}$ izlaz l-tog sloja neurona u neuronskoj mreži za i-ti primjerak $\mathbf{x}^{(i)}$ (zbog jednostavnosti zapisa u buduće taj i nećemo pisati). Aktivacija pojedinog sloja dobiva se primjenom aktivacijske funkcije na logit.

Napomena 3.3.1. Ulazni sloj neurona nema aktivacijsku funkciju. Njegov logit i aktivacija su sami ulazni podaci.

$$\boldsymbol{z}^{(1)} = \boldsymbol{a}^{(1)} = \boldsymbol{x}.$$

Za sve $l \in \{2, \ldots, L\}$, logit l-tog sloja dobijemo množenjem matrice $\mathbf{w}^{(l)}$ matricom $\mathbf{a}^{(l-1)}$. Dobivena matrica je dimenzija $S_l \times 1$ u kojoj element $\mathbf{z}_i^{(l)}$ predstavlja logit i-tog neurona u l-tom sloju neuronske mreže.

$$\mathbf{z}^{(l)} = \mathbf{w}\mathbf{x} = \begin{bmatrix} \mathbf{z}_1^{(l)} \\ \vdots \\ \mathbf{z}_{N_l}^{(l)} \end{bmatrix}$$

Aktivaciju l-tog sloja dobijemo primjenom aktivacijske funkcije l-tog sloja s_l na logit $\mathbf{z}^{(l)}$. Dobivamo:

$$\mathbf{a}^{(l)} = s_l\left(\mathbf{z}^{(l)}\right) = \begin{bmatrix} s_l\left(\mathbf{z}_1^{(l)}\right) \\ \vdots \\ s_l\left(\mathbf{z}_{N_l}^{(l)}\right) \end{bmatrix}.$$

Izlaz neuronske mreže također će biti matrica koju ćemo označiti s $h(\mathbf{x})$, a računa se kao:

$$h(\mathbf{x}) = \mathbf{a}^{(L)} = s_L\left(\mathbf{z}^{(L)}\right) = \begin{bmatrix} s_L\left(\mathbf{z}_1^{(L)}\right) \\ \vdots \\ s_L\left(\mathbf{z}_{N_L}^{(L)}\right) \end{bmatrix},$$

Cilj strojnog učenja je postići $h(\mathbf{x}^{(i)}) \approx \mathbf{y}^{(i)}$ za svaki uređeni par $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ iz skupa primjeraka za učenje. Ovdje vrijednosti težina i pristranosti ne ovise o ulaznim podacima te ih nazivamo *parametrima* mreže. Logit i aktivacija ovise o parametrima mreže i o ulaznim podacima. Druge veličine u neuronskoj mreži, koje također ne ovise o ulaznim podacima, a razlikuju se od parametara po tome što ih mreža ne uči (već se ručno unose) nazivamo *hiperparametrima*. Opisani postupak prolaska ulaznih informacija kroz neuronsku mrežu nazivamo *proslijeđivanje unaprijed*.

Da bismo u potpunosti definirali neuronsku mrežu potrebno je znati:

- broj slojeva,
- broj neurona u svakom sloju,
- način povezanosti mreže,
- početne vrijednosti težina,
- početne vrijednosti pristranosti,
- aktivacijske funkcije.

Funkcija gubitka i pogreške

Većina informacija u ovom odjeljku preuzeta je iz [2]. Kako bismo izmjerili koliko je neka hipoteza dobra moramo imati *funkciju gubitka* (*loss function*, *cost function*) $L\left(y^{(i)}, h(\mathbf{x}^{(i)}|\theta)\right)$ kojom mjerimo koliku količinu korisnosti (ulaznih podataka) smo izgubili izborom hipoteze h kada je ulaz $\mathbf{x}^{(i)}$, a pripadajuća ciljna oznaka $y^{(i)}$. Funkcija gubitka kod regresije je tipično definirana kao kvadratno odstupanje:

$$L\left(y^{(i)}, h(\mathbf{x}^{(i)}|\theta)\right) = \left(h(\mathbf{x}^{(i)}|\theta) - y^{(i)}\right)^2, \tag{3.7}$$

a kod klasifikacije kao:

$$L\left(y^{(i)}, h\left(\mathbf{x}^{(i)}|\theta\right)\right) = \mathbf{1}\{h\left(\mathbf{x}^{(i)}|\theta\right) \neq y^{(i)}\}. \tag{3.8}$$

Empirijska pogreška definira se kao očekivanje funkcije gubitka nad svim primjerima iz skupa za učenje, uz pretpostavku uniformne distribucije primjera.

$$E(\theta|\mathcal{D}) = \mathbb{E}_{\mathcal{D},\theta}[L] = \frac{1}{N}\sum_{i=1}^{N} L\left(y^{(i)}, h\left(\mathbf{x}^{(i)}|\theta\right)\right), \tag{3.9}$$

Faktor $\frac{1}{N}$ nebitan je za nalaženje minimuma funkcije E te se tipično koristi faktor $\frac{1}{2}$.

U nekim slučajevima gubitci su asimetrični. U takvim se slučajevima funkcija gubitka definira pomoću *matrice gubitka* (*loss matrix*) $L = [L_{kj}]_{k,j}$ gdje je L_{kj} gubitak koji nastaje pri klasifikaciji primjera koji zapravo pripada klasi C_k, u klasu C_j ($L_{kk} = 0$). Postoji i *matrica gubitka nula-jedan* (*zero-one loss matrix*) za koju je $L_{kj} = \mathbf{1}\{k \neq j\}$ koja odgovara gubitku definiranom s (3.8).

Pravilo perceptrona

Jedan od prvih modela umjetne neuronske mreže razvio je Frank Rosenblatt. U svom radu koristio je teoriju neuroznanstvenika Donalda Hebba i stariji model umjetnog neurona (McCulloch i Pitts, *A Logical Calculus of Ideas Immanent in Nervous Activity*, 1943.). Rosenblatt 1958. godine prvi predstavlja tzv. pravilo perceptrona (*perceptron learning rule*) — pravilo po kojem se ažuriraju težine (brojevi koji predstavljaju važnost pojedinog neurona) u neuronskim mrežama. *IBM 704* je računalo na kojem je razvijen program s takvim perceptronima, na *Cornell Aeronautical Laboratory* 1957. *Mark I Perceptron* je prvo računalo koje je izgrađeno samo u svrhu implementiranja neuronske mreže s pravilom perceptrona. Rosenblatt 1962. još piše knjigu *Principles of neurodinamics*, koja prikazuje nekoliko arhitektura i diskutira ideje višeslojnih mreža sličnih modernim konvolucijskim mrežama. On ih zove *C-systems*. Ovo možemo smatrati teorijskim rođenjem *dubokog učenja*. 1962. godine dokazana je konvergencija algoritma za učenje težina u perceptronu. Perceptron je prvotno zamišljen kao stroj za prepoznavanje slika, čija je implementacija ugrađena u sam hardver. Sastojao se od jednog neurona koji je imao *step*-funkciju (*binary threshold neuron*) kao aktivacijsku funkciju i *pravilo perceptrona* za ažuriranje težina u neuronskoj mreži (koja osim ulaznog sloja ima samo jednan neuron):

$$y = \begin{cases} 1, & z \geq \theta \\ 0, & \text{inače} \end{cases}, \tag{3.10}$$

gdje je z logit, definiran kao $z = \sum_i w_i x_i$, a θ je granica (prag).
U smislu problema klasifikacije, ovo bi bila binarna klasifikacija hiperravninom $\sum_i w_i x_i = \theta$.

Zbog konzistentnosti, u nastavku uzimamo da je $\theta = 0$, a logit $z = b + \sum_i w_i x_i$. Tada pri računanju, kao zadnji element, ulaznom vektoru dodajemo 1, a vektoru težina pristranost b.

Pravilo perceptrona:

1. Izaberi slučaj za treniranje.
2. Ako izlaz odgovara ciljnoj oznaci nemoj ništa raditi.
3. Ako je perceptron predvidio 0, a trebao je predvidjeti 1, zbroji ulazni vektor s vektorom težina.
4. Ako je perceptron predvidio 1, a trebao je predvidjeti 0, oduzmi ulazni vektor od vektora težina.

Klasičan upit u logici i teorijskom računarstvu je *parnost*. Ovaj upit izvodi se nad binarnim nizovima podataka, a samo oni nizovi s parnim brojem jedinica dobivaju oznaku 1. Možemo uzeti u obzir samo konačne vektore binarnih podataka duljine n i takvu funkciju nazvati $\texttt{parnost}_n(x_0, x_1, \ldots, x_n)$. Funkcija $\texttt{parnost}_2$ je zapravo XOR. Ako je naš problem XOR ili bilo koja druga funkcija parnosti, perceptron ne može naučiti (točno) klasificirati ulaz. To znači da perceptron s dva ulazna neurona (po jedan za svaki ulaz XOR-a) ne može prilagoditi svoje težine tako da razdvajaju nule od jedinica. Ako s w_1 i w_2 označimo težine, a s b pristranost u takvoj mreži, dobivamo četiri nejednakosti:

$$\begin{aligned} w_1 + w_2 &\geq b \\ 0 &\geq b \\ w_1 &< b \\ w_2 &< b \end{aligned}$$

Svaka od ovih nejednakosti vrijedi za jedan slučaj u XOR-u, odnosno, mora biti $w_1 x_1 + w_2 x_2 \geq b$, ako želimo dobiti izlaz 1, a $w_1 x_1 + w_2 x_2 < b$ inače. Sada ako za (x_1, x_2) uvrstimo $(1, 1)$ dobivamo prvu nejednakost, ako uvrstimo $(0, 0)$ dobivamo drugu nejednakost, $(1, 0)$ nam daje treću, a $(0, 1)$ četvrtu nejednakost. Sada ako zbrojimo prve dvije nejednakosti dobivamo $w_1 + w_2 \geq b$, a ako zbrojimo druge dvije dobivamo $w_1 + w_2 < b$ i odmah je jasno da ovaj sustav nema rješenja. Dakle, perceptron ne može izračunati funkciju XOR.

U smislu osnovnog problema klasifikacije, hiperravninu koja bi razdvajala ove dvije klase (0 i 1) možemo zapisati kao $w_1 x_1 + w_2 x_2 = -b$ odnosno $w_1 x_1 + w_2 x_2 + b = 0$ (implicitni oblik jednadžbe pravca). Dakle perceptron s dva ulaza možemo zamisliti kao

binarnu klasifikaciju pravcem. Zamislimo jednostavan koordinatni sustav u dvije dimenzije gdje na svakoj osi imamo samo 0 i 1,

$$\begin{aligned} \text{XOR}(0,1) &= 1, & \text{XOR}(1,1) &= 0, \\ \text{XOR}(0,0) &= 0, & \text{XOR}(1,0) &= 1; \end{aligned}$$

ako zapišemo rezultate na tim koordinatama, ne možemo razdvojiti nule od jedinica pravcem (a mogli bismo krivuljom). Predloženo rješenje bilo je napraviti perceptron s više slojeva.

Marvin Minsky i Seymour Papert dokazali su ovo u knjizi *Perceptrons: An Introduction to Computational Geometry* 1969. godine. Naizgled, nijedna neuronska mreža nije mogla provoditi osnovne logičke operacije, što su strojevi ostvareni logičkim sklopovima mogli bez problema.

Još jedan udarac umjetnoj inteligenciji zadao je James Lighthill kada je svojim člankom *Artificial inteligence: A General Survey* (koji je predao u *British Science Research Council* 1973. godine) uzrokovao zatvaranje svih osim tri zavoda za UI u Ujedinjenom Kraljevstvu. Mnogo znanstvenika tada je odustalo od rada na tom polju. Jedan od tri preostala zavoda bio je u Edinburghu te je jedan od njihovih profesora morao dati izjavu. U ovoj izjavi prvi se puta spominje kognitivna znanost i definira ugrubo njen opseg. Ovaj profesor zvao se Christopher Longuet-Higgins i započeo je rad na UI 1967. kada se zaposlio na edinburškom sveučilištu. Dao je i odgovor na pitanje zašto zadržati UI: ne trebamo UI da bismo gradili strojeve, već da bismo razumjeli ljude. Lighthill ovdje kaže da se proučavanje neuronskih mreža može shvatiti kao računalno modeliranje centralnih neuronskih sustava čovjeka, ali mora se poklapati sa stvarnim zaključcima neurologije, bez pojednostavljivanja i varijacija, na što mu Higgins odgovara: kako bismo razumjeli što čini računalo moramo promatrati softver, tako i da bismo razumjeli što radi čovjek moramo promatrati njegove mentalne procese i kako oni međudjeluju; najvažnije znanje dobiveno iz istraživanja UI je razumijevanje, modeliranje i formalizacija interakcije procesa.

Zato dolazi do dugogodišnjeg zanemarivanja neuronskih mreža. Ideja neuronskih mreža ostala je zanimljiva samo nekolicini ljudi, ali uz razvitak određenih grana filozofije i psihologije dolazi do ponovnog proučavanja neuronskih mreža 1980-ih godina. U međuvremenu, Thomas Kuhn 1962. predlaže *pomak paradigme* (*paradigm shift*) koji pomaže razvitku kognitivne znanosti. Ovaj pomak dogodio se u šest disciplina koje postaju osnovne discipline kognitivne znanosti: antropologija, računarstvo, lingvistika, neurologija, filozofija i psihologija. Po prvi puta se činilo dobrim napustiti najnovije ideje i vratiti se starim idejama. Prelazi se s proučavanja nepromjenjivih struktura na proučavanje promjena. [11]

Propagacija unatrag — *backpropagation*

Dakle, do tada se znalo kako trenirati jednoslojnu neuronsku mrežu, naslućivalo se i da bi postojanje skrivenog sloja znatno povećalo snagu neuronskih mreža, ali nitko nije znao

kako proširiti pravilo perceptrona na više slojeva. Jednim rješenjem činilo se uvođenje novog pravila koje bi moglo učiti težine preko slojeva. Paul Werbos 1975. godine otkriva *propagaciju unatrag* (*backpropagation*) — način da prenese informacije kroz srednji, skriveni sloj — ali javnost ovo ne zamjećuje. Tek 1981. godine David Parker ponovo otkriva i 1985. objavljuje ovaj rezultat. Yann LeCun također otkriva *backpropagation* i objavljuje to u radu 1985. Zadnji puta, *backpropagation* otkrivaju D. E. Rumelhart, G. E. Hinton i R. J. Williams u San Diegu, te 1986. godine objavljuju članak *Learning internal representations by error propagation*, čime počinje kognitivno razdoblje dubokog učenja. Neki od znanstvenika koji su sudjelovali u programu u San Diegu su David Rumelhart, Terry Sejnowski, John Hopfield, Jeffrey Elman i Michael I. Jordan.

Gradijentni spust

Većina informacija u ovom odjeljku preuzeta je iz [2].

Kako bismo dobili što bolju procjenu, želimo da greška bude minimalna. Dakle, ako pronađemo $\mathbf{w}^*$ takav da je $E(\mathbf{w}^*) = \min_{\mathbf{w}} E(\mathbf{w})$ pronašli smo najbolju hipotezu. Ovdje ćemo koristiti iterativnu metodu traženja minimuma pod imenom *gradijentni spust*. Za derivabilne funkcije $f : \mathbb{R} \to \mathbb{R}$, broj $f'(c)$ predstavlja koeficijent smjera tangente na te funkcije u točki $(c, f(c))$. Tada iz teorema o srednjoj vrijednosti (Lagrange) vrijedi:

$$f(c + \varepsilon) \approx f(c) + \varepsilon f'(c).$$

Dakle, možemo smanjiti vrijednost funkcije f mijenjajući argument za malu vrijednost ε u željenom smjeru (biramo $\varepsilon > 0$ ako je $f'(c) < 0$, a $\varepsilon < 0$ ako je $f'(c) > 0$).

Slično vrijedi i za funkcije više varijabli. Neka je $g : \mathbb{R}^n \to \mathbb{R}$ diferencijabilna, tada je *gradijent* funkcije g jednak

$$\nabla g(c) = \left(\frac{\partial g}{\partial x_1}(c), \ldots, \frac{\partial g}{\partial x_n}(c) \right).$$

Funkcija g iz točke c raste najbrže u smjeru gradijenta $\nabla g(c)$, a pada najbrže u smjeru *antigradijenta* $-\nabla g(c)$. Na ovoj se ideji temelji gradijentni spust.

Funkciju g minimiziramo tako da krenemo od proizvoljne točke x_0 i zatim računamo nove točke kao:

$$x_{k+1} = x_k - \alpha_k \nabla g(x_k), \quad k = 0, 1, \ldots$$

gdje je vrijednost α_k duljina *koraka* kojim se pomičemo u smjeru antigradijenta. Često uzimamo fiksnu veličinu za sve α_k te ju tada označavamo samo s α. Ovo ponavljamo unaprijed zadan broj puta ili dok se vrijednost x_k ne ustabili (dobivamo male razlike u susjednim x_k). Važno je napomenuti da gradijentni spust može zapeti u nekom od lokalnih minimuma ako funkcija nije konveksna. U slučaju neuronskih mreža minimizirat ćemo funkciju pogreške, a argumenti pomoću kojih minimiziramo bit će težine i pristranosti.

U strojnom učenju razlikujemo nekoliko verzija gradijentnog spusta:

- deterministički ili grupni gradijentni spust (*batch gradient descent*) — u svakom koraku koristimo cijeli skup podataka za učenje (gradijent računamo kao aritmetičku sredinu gradijenta za svaki podatak posebno); konvergencija je zagarantirana, ali je metoda vrlo spora
- stohastički gradijentni spust (*stohastic gradient descent*) — u svakom koraku koristimo samo jedan primjerak podatka za učenje (računamo gradijent samo po tom jednom podatku); često brži od determinističkog pristupa, ali ne garantira konvergenciju, odnosno, može oscilirati oko minimuma bez da se na njemu zaustavi
- gradijentni spust s mini-grupama (*mini-batch gradient descent*) — skup podataka za učenje podijelimo u nekoliko grupa jednake veličine (u svakom koraku računamo gradijent za sve primjere iz trenutne grupe); predstavlja kompromis prethodna dva pristupa

Algoritam za propagaciju unatrag

Većina informacija u ovom odjeljku preuzeta je iz [2] Gradijentnim spustom možemo promijeniti težine u izlaznom sloju, ali ne i u skrivenim slojevima — jer ne znamo pogreške u skrivenim slojevima već samo onu u izlaznom sloju. Zato ovu poznatu pogrešku šaljemo (*propagiramo*) unatrag prema skrivenim slojevima.

Općeniti oblik funkcije troška koji se često koristi je:

$$L(\mathbf{w}) = \frac{1}{2n} \sum_{i=1}^{n} \sum_{k=1}^{K} \left((h(\mathbf{x}^{(i)})_k - y_k^{(i)} \right)^2,$$

gdje je n broj primjera za učenje, K je broj neurona u izlaznom sloju, $h(\mathbf{x}^{(i)})_k$ je vrijednost k-tog neurona u izlaznom sloju za i-ti primjerak za učenje $\mathbf{x}^{(i)}$, a $y_k^{(i)}$ je ciljna oznaka k-tog neurona u izlaznom sloju za $\mathbf{x}^{(i)}$. Ovdje ćemo funkciju troška promatrati samo na jednom primjeru. Tada ona ima oblik:

$$L(\mathbf{w}) = \frac{1}{2} \sum_{k=1}^{S_L} \left(a_k^{(L)} - y_k \right)^2.$$

Zanima nas kako će se ponašati funkcija gubitka ako malo promijenimo težine. Računamo parcijalnu derivaciju $\frac{\partial L}{\partial w_{jk}^{(l)}}$, gdje je $w_{jk}^{(l)}$ težina između j-tog neurona u sloju $l-1$ i k-tog neurona u sloju l. Krenimo od zadnjeg sloja:

$$\frac{\partial L}{\partial w_{jk}^{(L)}} = \frac{\partial L}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}}, \tag{3.11}$$

gdje su parcijalne derivacije:

$$\frac{\partial L}{\partial a_j^{(L)}} = \frac{\partial}{\partial a_j^{(L)}} \left(\frac{1}{2} \sum_{k=1}^{S_L} \left(a_k^{(L)} - y_k \right)^2 \right) = \left(a_j^{(L)} - y_j \right), \tag{3.12}$$

$$\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} = \frac{\partial}{\partial z_j^{(L)}} s\left(z_j^{(L)} \right) = s'\left(z_j^{(L)} \right), \tag{3.13}$$

$$\frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}} = \frac{\partial}{\partial w_{jk}^{(L)}} \left(\sum_{i=1}^{S_{L-1}} w_{ji}^{(L)} a_i^{(L-1)} + b_j \right) = a_k^{(L-1)}. \tag{3.14}$$

Nakon uvrštavanja u (3.11) dobivamo:

$$\frac{\partial L}{\partial w_{jk}^{(L)}} = \left(a_j^{(L)} - y_j \right) s'\left(z_j^{(L)} \right) a_k^{(L-1)}.$$

Slično, za pristranosti vrijedi:

$$\frac{\partial L}{\partial b_j} = \frac{\partial E}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}},$$

gdje je

$$\frac{\partial z_j^{(L)}}{\partial b_j^{(L)}} = \frac{\partial}{\partial b_j^{(L)}} \left(\sum_{i=1}^{S_{L-1}} w_{ji}^{(L)} a_i^{(L-1)} + b_j \right) = 1$$

pa uvrštavanjem dobivamo

$$\frac{\partial L}{\partial b_j} = \left(a_j^{(L)} - y_j \right) s'\left(z_j^{(L)} \right).$$

Sada računamo $\frac{\partial L}{\partial w_{jk}^{(l)}}$, za $l = 1, \ldots, L-1$.

$$\frac{\partial E}{\partial w_{jk}^{(l)}} = \frac{\partial L}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}}.$$

gdje zadnja dva člana računamo istim postupkom kao prije, odnosno:

$$\frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} = \frac{\partial}{\partial z_j^{(l)}} s\left(z_j^{(l)} \right) = s'\left(z_j^{(l)} \right), \tag{3.15}$$

$$\frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} = \frac{\partial}{\partial w_{jk}^{(l)}} \left(\sum_{i=1}^{S_{l-1}} w_{ji}^{(l)} a_i^{(l-1)} + b_j \right) = a_k^{(l-1)}, \tag{3.16}$$

a $\frac{\partial L}{\partial a_j^{(l)}}$ je drugačiji. Čim malo promijenimo $a_j^{(l)}$, u idućem sloju se mijenjaju $a_i^{(l+1)}$ pa vrijedi:

$$\frac{\partial L}{\partial a_j^{(l)}} = \sum_{i=1}^{S_{l+1}} \frac{\partial L}{\partial a_i^{(l+1)}} \frac{\partial a_i^{(l+1)}}{\partial z_i^{(l+1)}} \frac{\partial z_i^{(l+1)}}{\partial a_i^{(l)}}.$$

Ovdje je

$$\frac{\partial a_i^{(l+1)}}{\partial z_i^{(l+1)}} = \frac{\partial}{\partial z_i^{(l+1)}} s\left(z_j^{(l+1)}\right) = s'\left(z_j^{(l+1)}\right), \tag{3.17}$$

$$\frac{\partial z_i^{(l+1)}}{\partial a_i^{(l)}} = \frac{\partial}{\partial a_i^{(l)}} \left(\sum_{j=0}^{S_l} w_{ij}^{(l+1)} a_i^l + i \right) = w_{ij}^{(l+1)}. \tag{3.18}$$

Dobili smo

$$\frac{\partial L}{\partial a_j^{(l)}} = \sum_{i=0}^{S_{l+1}} w_{ij}^{(l+1)} s'\left(z_i^{(l+1)}\right) \frac{\partial E}{\partial a_i^{(l+1)}}.$$

Analogno za pristranosti dobivamo

$$\frac{\partial L}{\partial b_j^{(l)}} = s'\left(z_j^{(l)}\right) \frac{\partial L}{\partial a_i^{(l+1)}}.$$

Ako označimo

$$\Delta_j^{(l)} := \frac{\partial L}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}}$$

dobivamo:

$$\Delta_j^{(l)} = \left(\sum_{i=0}^{S_{l+1}} w_{ij}^{(l+1)} s'\left(z_i^{(l+1)}\right) \frac{\partial L}{\partial a_i^{(l+1)}} \right) s'\left(z_j^{(l)}\right) \tag{3.19}$$

$$= s'\left(z_j^{(l)}\right) \sum_{i=0}^{S_{l+1}} w_{ij} \Delta_i^{(l+1)}. \tag{3.20}$$

Dakle za slojeve $l = 1, \ldots, L-1$ vrijedi:

$$\Delta_j^{(l)} = s'\left(z_j^{(l)}\right) \sum_{i=0}^{S_{l+1}} w_{ij} \Delta_i^{(l+1)}, \tag{3.21}$$

a za izlazni sloj L je

$$\Delta_j^{(L)} = \left(a_j^{(L)} - y_j\right) s'\left(z_j^{(L)}\right). \tag{3.22}$$

Sljedeće ažuriramo parametre mreže (težine i pristranosti), počevši od izlaznog sloja, po formulama

$$w_{jk}^{(l)} \leftarrow w_{jk}^{(l)} - \alpha \Delta_j^{(l)} a_k^{(l-1)},$$

$$b_j^{(l)} \leftarrow b_j^{(l)} - \alpha \Delta_j^{(l)}.$$

Ovime je završena jedna iteracija algoritma za *backpropagation*.

Ovdje također razikujemo stohastički i grupni pristup (i *batch*). Pri stohastičkom pristupu nakon svakog ovako obrađenog ulaznog podatka ažuriramo težine, dok u grupnom pristupu to radimo tek nakon cijele epohe, s time da se korekcija Δ za to vrijeme akumulira.

Napomena 3.3.2. *Ovaj algoritam primjenjiv je samo na neuronske mreže koje koriste derivabilne aktivacijske funkcije.*

Regularizacija

Većina informacija u ovom odjeljku preuzeta je iz [11]. Već smo vidjeli da kod nekih modela može doći do prenaučenosti. Tome su osobito skloni nelinearni modeli. Linearni modeli su jednostavniji te je kod njih taj rizik manji — ipak, prenaučenost je moguća i kod linearnih modela, pogotovo ako prostor primjera ima veliku dimenziju, a skup za učenje je malen (tada model daje preveliku težinu nepotrebnim značajkama). Također, kod logističke regresije javlja se problem prenaučenosti kod linearno odvojivih problema (u tom slučaju gradijent pogreške nema minimum te gradijentni spust neće konvergirati). Jedan način rješavanja ovog problema je ponovni odabir modela, ali postoji i drugi način — *regularizacija*. Ukratko, regularizacija se sastoji u tome da se u funkciju pogreške ugradi mjera složenosti modela. Na taj se način spriječava da model postane previše složen, jer će sa složenošću rasti i ukupna pogreška. Regularizacija povezuje minimizaciju *empirijskog rizika* (empirijske pogreške) s minimizacijom *strukturnog rizika* (složenosti modela). Ovim postupkom omogućeno je učenje složenih modela na manjim skupovima podataka bez velikog rizika od prenaučenosti.

Općeniti oblik *regularizacijskog izraza* je

$$\frac{\lambda}{2} \sum_{j=1}^{n} |w_j|^q,$$

gdje je $\mathbf{w} = (w_1, \ldots, w_n)$ vektor težina koje želimo regularizirati, a λ je takozvani *regularizacijski faktor*. Što je λ veći to se povećava vrijednost pogreške za složene modele, odnosno

povećava se potreba za manjim težinama. Za $\lambda = 0$ funkcija pogreške je zapravo neregularizirana i model koji dobijemo optimizacijom imat će najmanju empirijsku pogrešku, ali će ujedno biti i najsloženiji. Regularizirana funkcija pogreške bit će oblika:

$$E(\mathbf{w}|\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N} L\left(y^{(i)}, h\left(\mathbf{x}^{(i)}|\mathbf{w}\right)\right) + \frac{\lambda}{2}\sum_{j=1}^{n} |w_j|^q.$$

Odabirom parametra q kontroliramo koju vrstu regularizacije ćemo koristiti. Ako odaberemo $q = 1$ dobivamo L_1-regularizaciju, koja dovodi do takozvanih *rijetkih modela* (*sparse models*) — većina dimenzija je zanemarena jer su težine jednake 0. L_1-regularizacija nekada se naziva i *lasso* ili *basis pursuit denoising*, a prvi je puta predložena 1996. godine (Robert Tibshirani). Za $q = 2$ dobivamo L_2 regularizaciju, koja se tipično koristi kod logističke regresije i analitički je pogodna. Primijetimo da se u regularizacijskom izrazu ne uzima u obzir pristranost: ona zapravo određuje pomak hiperravnine u prostoru pa bi regularizacija ovog parametra davala hiperravninu koja prolazi kroz ishodište — što ne želimo, osim ako smo prethodno centrirali podatke. Tada je gradijentni vektor za funkciju pogreške (posebno za b, a posebno za $\mathbf{w}$):

$$\begin{aligned} \nabla E(b) &= \sum_{i=1}^{N}\left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right) \\ \nabla E(\mathbf{w}) &= \sum_{i=1}^{N}\left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)\mathbf{x}^{(i)} + \lambda\mathbf{w}. \end{aligned} \tag{3.23}$$

U svakom koraku gradijentnog spusta vektor težina ažurira se na sljedeći način:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta\left(\sum_{i=1}^{N}\left(h(\mathbf{x}^{(i)}) - y^{(i)}\right)\mathbf{x}^{(i)} + \lambda\mathbf{w}\right),$$

što možemo zapisati u obliku

$$\mathbf{w} \leftarrow \mathbf{w}(1 - \eta\lambda) - \eta\sum_{i=1}^{N}\left(h(\mathbf{x}^{(i)}) - y^{(i)}\right)\mathbf{x}^{(i)}.$$

Kada bi drugi pribrojnik bio konstantan, težine bi se u svakom koraku smanjivale proporcionalno s $(1-\eta\lambda)$. Ovaj efekt naziva se *propadanje težina* (*weight decay*). Primijetimo da promjena težina ne ovisi samo o parametru η već i o broju primjera N. Zbog toga stopu učenja η treba korigirati u ovisnosti o broju primjeraka: primjerice umjesto η koristiti $\frac{\eta}{N}$.

Regularizaciju općenito možemo definirati i kao bilo koju tehniku koja smanjuje razliku između pogreške učenja i ispitne pogreške pa možemo reći da su i sljedeće dvije tehnike zapravo tehnike regularizacije.

Ponekad trebamo način za izbjegavanje lokalnih minimuma funkcije pogreške (kako bismo došli do globalnog). Za to će nam poslužiti *moment* (predstavljen u članku Rumelharta, Hintona i Williamsa [10]). Intuitivno, zadržavamo neku količinu vrijednosti težina ne samo iz prošle iteracije već i iz one prije nje. Računa se po formuli

$$\mu \left(|w_j^{[i-1]} - w_j^{[i-2]}| \right), \tag{3.24}$$

gdje je $w_j^{[i]}$ j-ta težina u i-toj iteraciji ažuriranja težina, a $\mu \in [0, 1]$ je takozvana *stopa momenta*. Da bismo primijenili moment, pri računanju nove težine izraz (3.24) zbrajamo s regulariziranom vrijednošću nove težine.

Još jedna zanimljiva tehnika je *dropout*. Ova tehnika smanjuje razliku između pogreške učenja i ispitne pogreške pa time smanjuje i prenaučenost. Kako bismo primijenili tehniku *dropout*, uvodimo još jedan hiperparametar $\pi \in [0, 1]$ te se u svakoj iteraciji svaka težina postavlja na 0 s vjerojatnošću π (a inače ostaje ista). Ako se u iteraciji i težina w_j postavi na 0 tada se u epohi $i + 1$ umjesto $w_j^{[i]}$ koristi $w_j^{[i-1]}$. Ova tehnika prisiljava mrežu da uči redundantne činjenice kako bi bolje izolirala bitne značajke. Obično je $\pi = 0.2$.

Problem nestajućih i eksplodirajućih gradijenata

Pri treniranju neuronske mreže gradijentom i *backpropagation*-om ponekad dolazi do *problema nestajućeg gradijenta* (*vanishing gradient problem*). Ako u neuronskoj mreži postoji velik broj skrivenih slojeva, a vrijednosti derivacija aktivacijskih funkcija su brojevi između 0 i 1, prilikom korištenja ovih algoritama dolazi do uzastopnog množenja tih brojeva što u konačnici daje vrlo male „nestajuće" brojeve — dolazi do prigušivanja. Zbog ovoga dolazi do prestanka procesa učenja u „prednjim" slojevima mreže. Ako, nasuprot tome, koristimo aktivacijske funkcije čije su vrijednosti derivacija velike, dolazi do *eksplodiranja gradijenata*. Zbog ovoga prikazani algoritam za *backpropagation* na takvim mrežama postaje gotovo neupotrebljiv. Tek početkom 21. stoljeća, uvođenjem novih aktivacijskih funkcija poput ℓReLU te novim tehnikama učenja, poput tehnike *dropout* nastavlja se razvoj neuronskih mreža.

3.4 Povratne neuronske mreže

Većina informacija u ovom poglavlju preuzeta je iz [5] Povratne neuronske mreže (*Reccurent Neural Networks — RNN*) za razliku od *unaprijednih* (*feedforward*) na ulazu obrađuju podatke u obliku niza, a težine su joj međusobno vremenski ovisne. Kombinirajući te informacije, povratne mreže donose odluke na osnovu više prethodno viđenih podataka te su zato prikladne za učenje značajki iz sekvencijalnih podataka kao što je recimo tekst. Za povratne neuronske mreže može se reći da posjeduju memoriju koja pomaže pri lakšoj

obradi i prepoznavanju sekvencijalnih podataka. Obrađena informacija se čuva u skrivenim jedinicama povratne neuronske mreže te utječe na obradu novih ulaza. Pomoću toga pronalazi korelacije između vremenski razdvojenih ulaza koje se nazivaju *dugoročne korelacije*.

Matrice težina i *tranzicijska matrica* služe kao parametri koji određuju koliku važnost treba pridati sadašnjem ili prošlom stanju, a greška koju uzrokuju će se vratiti prilagođenim algoritmom za *backpropagation* zvanog *propagiranje pogreške unatrag u vremenu* (Backpropagation Through Time) i iskoristiti za prilagođavanje njihovih vrijednosti dok greška ne padne na minimalnu vrijednost. Korištenjem takvog algoritma otkriven je problem nestajućeg i eksplodirajućeg gradijenta koji je učenje povratne neuronske mreže činio jako teškim ili nemogućim. Kako bi se riješio taj problem, sredinom 90-tih godina prošlog stoljeća izmišljene su povratne neuronske mreže s jedinicama za *dugotrajno kratkoročno pamćenje* (*Long Short-Term Memory units - LSTM*). LSTM jedinice pomažu pri održavanju konstantnije vrijednosti pogreške tijekom učenja te tako daju mogućnost povratnoj mreži da uči u mnogo više koraka bez da gradijent nestane ili eksplodira. LSTM jedinica se može shvatiti kao ćelija s tri različita stanja, a to su ulaz, izlaz i brisanje. Svako od stanja ima vlastiti vektor naučenih težina na osnovu kojeg odlučuje treba li ulazni signal dopustiti ulaz u iterativni proces, izlaz iz procesa ili ga treba izbrisati iz procesa. Korištenje LSTM jedinica rješava problem nestajućeg gradijenta, ali očito unosi mnogo više parametara za učenje.

3.5 Konvolucijske neuronske mreže

Većina informacija u ovom poglavlju preuzeta je iz [2]

Konvolucijske neuronske mreže (*CNN*) poseban su tip neuronskih mreža za obradu podataka koji imaju topologiju nalik mreži. Najčešće se kao ulazni podaci koriste slike ili vremenski nizovi.

Skup podataka MNIST

Zamislimo da želimo prepoznati rukom pisanu znamenku sa slike. Da bismo to učinili potreban nam je velik broj slika koje prikazuju razne rukom pisane znamenke. Skup podataka *MNIST* (*a Modification of the National Institute of Standards and Technology of the United States dataset*) sastoji se od 60 tisuća takvih slika. Originalni skup podataka opisan je u knjizi *NIST special database 19: handprinted forms and characters database* P. J. Grothera iz 1995. godine, a MNIST je modifikacija dviju posebnih baza podataka iz originalnog skupa koje su sastavili Yann LeCun, Corinna Cortes i Christopher J. C. Burges. MNIST je danas dostupan iz mnogo izvora, ali najviše se koristi *Kaggle* gdje se podaci čuvaju u datoteci formata CSV.

Slike u MNIST-u su formata 28×28 piksela u formatu *grayscale*. Svaki piksel poprima vrijednost između 0 (bijelo) i 255 (crno), što je obrnuto od uobičajenog (inače 0 predstavlja crnu, a 255 bijelu boju). Na taj način sliku možemo predati neuronskoj mreži kao polje od $28^2 = 784$ brojeva. Ako sada u takvu mrežu dodamo skrivene slojeve, možemo ju *backpropagation* algoritmom naučiti da uz veliku točnost prepoznaje rukom pisane znamenke u istom formatu. Problem koji se javlja za ovaj primjer je što ako mreži damo sliku na kojoj znamenka nije centrirana, ona je neće prepoznati. To možemo riješiti na način da promatramo manje dijelove slike sve dok ne nađemo dio slike na kojem je znamenka centrirana. Još jedan način bio bi da skupu za učenje dodamo primjere na kojima znamenka nije centrirana, ali tada bi neuronska mreža morala imati dodatne skrivene slojeve kako bi mogla naučiti kompliciranije uzorke. Ovakva ideja pojavila se još krajem 1960-ih kada dolazi do pojave *dubokih neuronskih mreža*. U to vrijeme ovakve su mreže imale poteškoće s učenjem zbog svoje veličine, ali danas je taj problem uglavnom riješen (razvitkom tehnologije). Ipak, prvi pristup ima više smisla jer će neuronska mreža prepoznavati znamenku na isti način ma gdje se ona na slici nalazila. Rješenje za ovaj problem daju upravo konvolucijske neuronske mreže, koje na skupu podataka MNIST daju grešku od 0.21%.

Promotrimo slike u formatu RGB. Svaka takva slika sastoji se od tri komponente koje nazivamo *kanalima* — crveni, zeleni i plavi. Kada ih preklopimo dobivamo potpunu sliku u boji. Pikseli svake od komponenata poprimaju vrijednosti od 0 do 255 pa ih možemo, svaki posebno, promatrati kao slike u formatu *grayscale*. Postoji nekoliko pristupa obradi slika u formatu RGB. Za svaku sliku koju želimo obraditi možemo učiniti jedno od sljedećeg:

- Stvorimo novu sliku u kojoj svaki piksel poprima prosječnu vrijednost piksela na istoj poziciji iz sva tri kanala — ovo je uobičajeni način pretvaranja slike iz formata RGB u *grayscale*.

- Razdvojimo kanale i treniramo tri različita klasifikatora, svaki na jednom kanalu — takozvani *odbor* (*commitee*) klasifikatora.

- Svaku sliku u skupu podataka za učenje razdvojimo na 3 slike (po jedna za svaki kanal), sve tako dobivene slike nasumično poredamo i treniramo jedan klasifikator na svima — ovaj pristup se naziva *proširenje skupa podataka* (*dataset augmentation*).

- Svaki kanal pretvorimo u posebnu sliku i na dobivenim slikama treniramo tri instance istog klasifikatora, po jednu za svaki kanal, a zatim koristimo četvrti klasifikator za konačnu obradu — ovaj pristup vodi *konvolucijskim neuronskim mrežama*.

Svaki od ovih pristupa ima svoje prednosti i, ovisno o problemu, bilo koji od njih može biti dobar izbor.

Konvolucija

Definicija 3.5.1. Neka su $f, g : \mathbb{Z} \to \mathbb{R}$. Tada funkciju $f * g$ definiranu s

$$(f * g)(t) \coloneqq \sum_{\tau=-\infty}^{+\infty} f(\tau) f(t - \tau),$$

nazivamo *diskretnom konvolucijom* funkcija f i g.

Konvolucija je komutativna, asocijativna i distributivna (prema zbrajanju) operacija. U kontekstu konvolucijskih neuronskih mreža, prvi argument konvolucije (funkcija f) je *ulaz*, a drugi argument (funkcija g) je *jezgra* (*kernel*) koja se još često naziva *filtrom* (predstavlja parametre koje želimo podesiti prilikom učenja neuronske mreže), dok rezultat (funkciju $f * g$) nazivamo *matricom značajki*. Obično pretpostavljamo da su funkcije f i g nula svuda osim na konačnom skupu. Zbog toga u praksi možemo implmentirati beskonačnu sumu kao sumu konačno mnogo elemenata višedimenzionalnog polja. Ako je ulazni podatak dvodimenzionalna slika I, vjerojatno ćemo koristiti i dvodimenzionalni filtar K pa konvolucija glasi

$$S(i, j) = (I * K)_{ij} = \sum_m \sum_n I(m, n) K(i - m, j - n).$$

Kako je konvolucija komutativna, ovo možemo zapisati i kao

$$S(i, j) = (K * I)_{ij} = \sum_m \sum_n I(i - m, j - n) K(m, n).$$

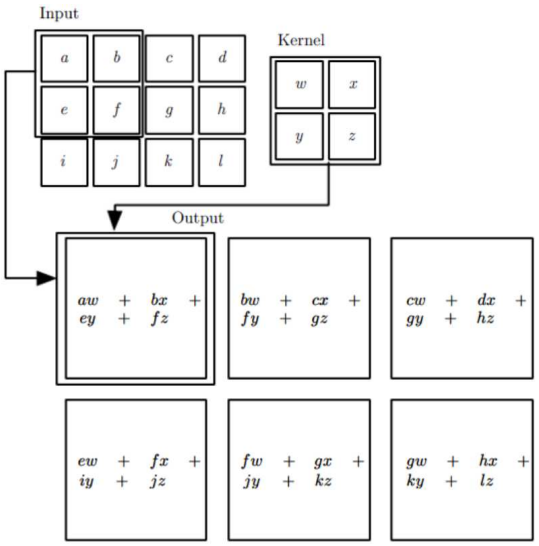
Neke biblioteke vezane za neuronske mreže umjesto konvolucije implementiraju *unakrsnu korelaciju*

$$S(i, j) = (K * I)_{ij} = \sum_m \sum_n I(i + m, j + n) K(m, n).$$

i nju nazivaju konvolucijom. Prilikom implementacije neuronske mreže ovo nije jako bitno jer je jedina razlika u tome što unakrsna korelacija nije komutativna (jer nema zrcaljenja jezgre, a to uvijek možemo učiniti).

Arhitektura konvolucijskih neuronskih mreža

Konvolucijska mreža osim ulaznog i izaznog sloja može imati još tri tipa slojeva, a to su *konvolucijski sloj*, *sloj sažimanja* te *potpuno povezani sloj*. Ulazni sloj je obično neka dvodimenzionalna ili trodimenzionalna „matrica" značajki. Zatim se izmjenjuju konvolucijski slojevi i slojevi sažimanja te na kraju dolazi potpuno povezani sloj.



Slika 3.5: Postupak konvolucije [2]

Konvolucijski sloj

Ulaz konvolucijskog sloja predstavlja neki objekt, obično sliku, dimenzija $h \times w \times d$, gdje je h visina, w širina, a d dubina. Ako se radi o slici h i w predstavljaju visinu i širinu, a dubina je broj kanala (za *grayscale* je $d = 1$, a za RGB $d = 3$). Ako je dubina jednaka 1 obično dimenzije pišemo samo kao $h \times w$.

Svaki konvolucijski sloj ima određen broj jezgri unaprijed fiksirane visine i širine. One čuvaju parametre koje konvolucijska mreža podešava pri učenju. Dubina jezgre uvijek odgovara dubini ulaza, dok su joj visina i širina obično puno manje od visine i širine ulaznih primjera (najčešće neparni brojevi).

Konvolucija se odvija nad ulazom i jezgrama na način prikazan na slici 3.5. Zamislimo da smo postavili jezgru (filtar) preko ulaznih podataka na način prikazan na slici. Izlazni podatak računamo na način da množimo vrijednosti postavljene jednu preko druge te sve rezultate zbrojimo. Svaku sljedeću izlaznu vrijednost dobivamo pomicanjem jezgre u desno, a kada dođemo do desnog kraja, vraćamo se na krajnji lijevi rub, ali se pomaknemo prema dolje. Primjećujemo da je dimenzionalnost izlaznih podataka manja nego dimenzionalnost ulaznih podataka (čim jezgra ima neku dimenziju veću od 1). Ovaj postupak zapravo je unakrsna korelacija. Ako želimo pravu konvoluciju, prvo moramo zrcaliti jezgru.

Kako bismo u potpunosti odredili konvolucijski sloj potrebno nam je znanje o vrijednostima nekoliko hiperparametara:

- širina ruba P,

- veličina jezgre F,
- broj jezgara N,
- veličina pomaka S.

Nadopunjavanje

Opisanim postupkom u svakom sljedećem konvolucijskom sloju dobivali bismo podatke sve manjih dimenzija. Također, ako promotrimo malo bolje sliku 3.5 vidjet ćemo da element a sudjeluje samo u jednom izlaznom elementu, dok na primjer element g sudjeluje u njih četiri. Na ovaj način dolazi do gubljenja informacija koje se nalaze na rubu (koriste se rjeđe od središnjih elemenata). Ako ovo želimo izbjeći, oko ulazne slike (matrice) dodajemo rub širine P koji popunjavamo nulama i taj rub nazivamo *nadopuna* (*padding*). Širina ruba P može biti proizvoljna, ali najčešće se koriste sljedeće dvije varijante:

- Bez nadopune (*no padding*) — koristimo izvornu matricu, odnosno $P = 0$
- Jednaka nadopuna (*same padding*) — ulaz nadopunjavamo tako da izlaz ima iste dimenzije kao ulaz

Korak

Korak (*stride*) S nam govori za koliko ćemo piksela pomaknuti jezgru udesno (prema dolje) pri računanju konvolucije ulaza i jezgre. Korak se obično gleda kao jedna vrijednost S pa se za toliko pomičemo i u desno i prema dolje, ali možemo imati i različite korake S_w i S_h kojima bismo mogli posebno odrediti svaku od vrijednosti (S_d nema smisla jer jezgra ima istu dubinu kao ulaz).

Jezgra

Neki od najpoznatijih jezgara su:

- **jezgra srednje vrijednosti** — svaki piksel zamijeni prosjekom susjednih piksela (zamućuje sliku)

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- **Gaussova jezgra** — slična jezgri srednje vrijednosti, ali ovaj puta uzimamo težine tako da pikseli oko središnjeg imaju veću važnost od rubnih piksela

$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

- **jezgre za izoštravanje slike** — postoji više verzija, jedna je

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix},$$

- **jezgre za detekciju rubova** — postoji više verzija, najpoznatija je *Sobel*; ovdje koristimo dvije jezgre, po jednu za svaki smjer:

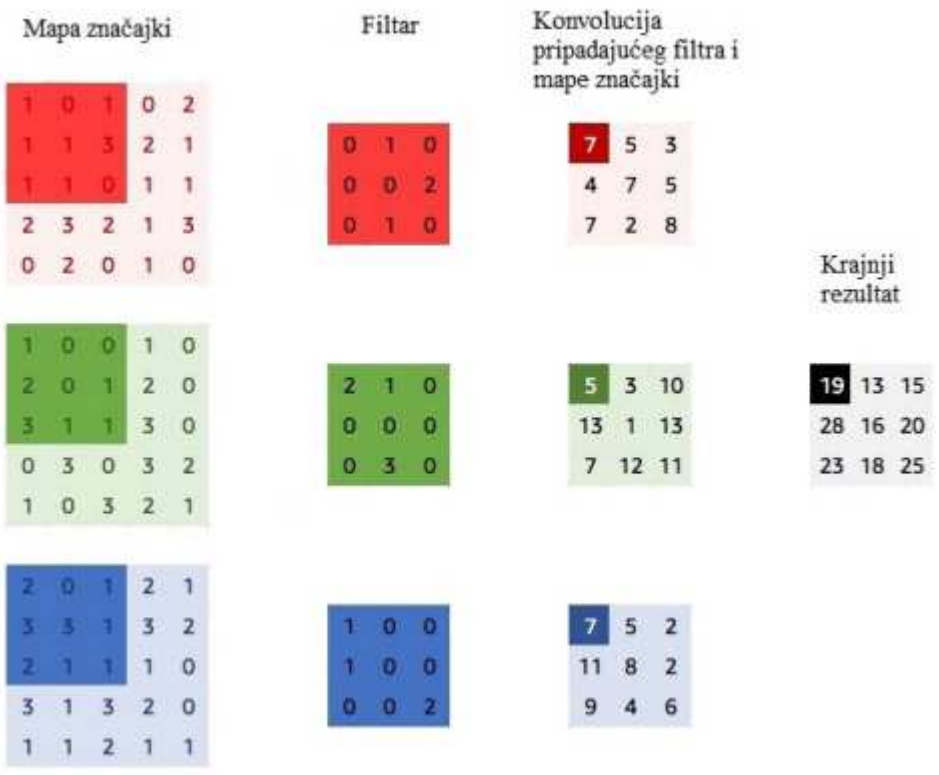
$$K_x := \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad K_y := \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix};$$

  K_x detektira vertikalne rubove, a K_y horizontalne. Konvoluciju računamo posebno za K_x i K_y te dobijemo vrijednosti x i Y. Novu vrijednost piksela tada dobijemo kao $\sqrt{x^2 + y^2}$.

Ako se bavimo nadziranim učenjem, ne moramo ručno birati jezgru, već ćemo dati mreži da nauči upravo najbolju jezgru. Elementi matrice jezgre ovdje predstavljaju parametre za učenje, kao što su to bile težine za klasičnu neuronsku mrežu. Kako bismo odredili jezgru, moramo joj odrediti visinu, širinu i dubinu. Za dubinu vrijedi da mora biti jednaka kao kod ulazne mape značajki. Za visinu i širinu obično se uzima jednaka vrijednost te se tada označava s F. Ako želimo različite vrijednosti visine i širine tada ih označavamo s F_h i F_w. Vrijednost F najčešće je neparna jer tada jezgra ima centralni element što nekada može olakšati pozicioniranje jezgre.

Osim veličine, određujemo i broj jezgri. Broj jezgri određuje dubinu izlazne mape značajki konvolucijskog sloja. Primjenom različitih jezgri želimo detektirati različite značajke; zato u početnim slojevima konvolucijske neuronske mreže imamo manji broj jezgri nego kasnije — prvo tražimo osnovne značajke (koje je lakše prepoznati), a zatim kompleksnije (za koje je potrebno više jezgri).

Promotrimo sada što se događa ako imamo ulaz dubine veće od 1. Za takav ulaz potrebna nam je i jezgra iste dubine. Tada razdvojimo ulaz i jezgru po dubini i izračunamo



Slika 3.6: Primjer konvolucije na ulazu dubine 3 [2]

konvoluciju za svaki „sloj" posebno. Na kraju dobivene matrice zbrojimo da bismo dobili izlaznu vrijednost dubine 1. Na slici 3.6 vidi se kako bismo to učinili za jedan primjer slike u formatu RGB.

Sloj sažimanja

Konvolucijski sloj stvara mapu značajki za ulazni podatak. Problem je u tome što su zapisane točno na onom mjestu na kojem se nalaze na slici pa će se ista značajka na dva različita mjesta u matrici smatrati dvjema različitim značajkama (slično problemu prepoznavanja znamenke koja nije centrirana). Cilj je povećati prostornu invarijantnost, odnosno napraviti verziju matrice značajki niže kvalitete u svrhu izdvajanja bitnih strukturalnih značajki. Ovo rade *slojevi sažimanja* (*pooling layer*). Osim povećanja prostorne invarijantnosti, ti slojevi se koriste i za smanjivanje dimenzija matrice značajki (dubina ostaje ista). Tako smanjujemo broj parametara mreže.

Definiramo veličinu jezgre F te pomak S. Nadopunjenje se u slojevima sažimanja obično ne koristi. Postupak je vrlo sličan kao kod konvolucije, promatramo $F \times F$ podataka i neku *funkciju sažimanja* koja ove podatke preslikava u jednu vrijednost. Zatim se pomaknemo za S. U ovom sloju nema parametara koje želimo naučiti. Postoji nekoliko funkcija sažimanja:

- **sažimanje maksimalnom vrijednošću** (*max-pooling*) — maksimum po svim elementima unutar okvira $F \times F$
- **sažimanje srednjom vrijednošću** — unutar svakog okvira $F \times F$ računamo srednju vrijednost svih elemenata
- **sažimanje L_2-normom** — rezultat je suma kvadrata svih elemenata unutar okvira
- **sažimanje težinskim usrednjavanjem** — težine opadaju s udaljenošću od centralnog elementa (slično kao kod konvolucije s Gaussovom jezgrom)

Potpuno povezani sloj

Potpuno povezani sloj (*fully connected layer*) je zapravo klasična potpuno povezana neuronska mreža s proizvoljnim brojem skrivenih slojeva. Ulazni podaci su jednodimenzionalno polje dobiveno od višedimenzionalne mape značajki iz prethodnog sloja (*flattening*), najčešće iz sloja sažimanja. Izlaz iz potpuno povezanog sloja ujedno je i izlaz cijele mreže. U potpuno povezanom sloju između svaka dva (unutarnja) sloja imamo matricu težina veličine $M \times N$ ako je u prvom sloju M neurona, a u drugom N. To je puno parametara za naučiti, zato ovaj sloj obično stavljamo na kraj, kada se dimenzija smanji primjenom slojeva sažimanja.

Propagacija unaprijed

S $F^{(l)}$, $P^{(l)}$, $S^{(l)}$ i $N^{(l)}$ označit ćemo parametre l-tog konvolucijskog sloja ili sloja sažimanja u konvolucijskoj neuronskoj mreži. Neka je ulazna mapa značajki u sloj l dimenzija $h^{(l-1)} \times w^{(l-1)} \times d^{(l-1)}$. Tada pripadna izlazna mapa značajki ima dimenzije $h^{(l)} \times w^{(l)} \times d^{(l)}$, gdje vrijedi:

$$\begin{aligned} h^{(l)} &= \left\lfloor \frac{h^{(l-1)} + 2P^{(l)} - F^{(l)}}{S^{(l)}} + 1 \right\rfloor \\ w^{(l)} &= \left\lfloor \frac{w^{(l-1)} + 2P^{(l)} - F^{(l)}}{S^{(l)}} + 1 \right\rfloor \\ d^{(l)} &= N^{(l)} \end{aligned} \tag{3.25}$$

Kao i kod klasičnih neuronskih mreža i ovdje imamo aktivaciju $A^{(l)}$, koja predstavlja izlaz sloja l. Dobit ćemo ju na sljedeći način:

- napravimo konvoluciju ulazne mape značajki (odnosno izlazne mape značajki iz sloja $l-1$):
$$A^{(l-1)} * K^{(l)}$$
- dodamo pristranost $b^{(l)}$ — to je vektor dimenzije $N^{(l)}$, za svaku jezgru po jedna vrijednost — pri konvoluciji ulazne mape značajki s i-tom jezgrom dobivamo novu značajku dimenzija $h^{(l)} \times w^{(l)}$ kojoj onda (svakom elementu posebno) pribrajamo vrijednost $b_i^{(l)}$;
- na svaki element mape značajki primijenimo aktivacijsku funkciju σ da bismo dobili konačni izlaz sloja l.

U sloju sažimanja nema pristranosti niti aktivacijske funkcije (dakle, takav sloj nema parametre). Aktivacija u potpuno povezanom sloju ista je kao kod klasičnih neuronskih mreža.

3.6 Primjena

U ranim 1990-im godinama nije se puno toga događalo u proučavanju umjetne inteligencije. U svijetu UI dolazi do prebacivanja generalnog zanimanja na *Support Vector Machines* (SVM) koji su bili matematički dobro potkrijepljeni — za razliku od neuronskih mreža koje su više bile zanimljive s filozofskog stajališta i uglavnom su ih razrađivali psiholozi i kognitivni znanstvenici. Također, činilo se da SVM-ovi daju bolje rezultate. Na prijelazu stoljeća događa se slijedeće:

- 1997. Hochreiter i Schmidhuber izumili su tzv. *long short-term memory*,
- 1998. LeCun, Bottou, Bengio i Haffner stvaraju prvu konvolucijsku neuronsku mrežu zvanu *LaNet-5* koja ostvaruje značajne rezultate (na bazi podataka MNIST).
- Konačno, 2006. Hinton, Osindero i Teh objavljuju rad u kojem predstavljaju *deep belief networks* (DMB) koje daju znatno bolje rezultate na bazi podataka MNIST. [4]

2013. godine objavljen je *word2vec* — tehnika obrade prirodnog jezika za učenje povezanosti između riječi. Izradili su ga znanstvenici u Googleu pod vodstvom Tomasa Mikolova. 2016. godine Google razvija takozvani *Tensor Processing Unit* (*TPU*) — čip na kojemu se odvijaju operacije množenja matrica i primjenjuju aktivacijske funkcije koristeći *TensorFlow*. Do 2021. postoji 5 verzija ovakvih čipova (TPUv1, TPUv2, TPUv3, Edge v1 i

TPUv4). Nakon toga, 2017. godine (članak objavljen tek 2018.) tim DeepMind predstavlja AlphaZero, računalo koje igra šah, shogi i go. Prije njega iste je godine predstavljen i AlphaGo Zero koji je igrao samo go. 2019. DeepMind objavljuje novi članak u kojem daju detalje o novom algoritnu *MuZero* koji igra igre kojima ne zna pravila. Još jedan zanimljvi program je *DALL-E* — on na temelju tekstualnih opisa stvara sliku. Razvili su ga *OpenAI*, a predstavili su ga javnosti 5.1.2021. godine. Koristi verziju *GPT-3* modela uz pomoć koje procesira prirodni jezik i generira sliku. Može generirati realistične slike stvarnih objekata ili generirati sliku objekta koji ne postoji, čak postoji i nekoliko različitih stilova slika.

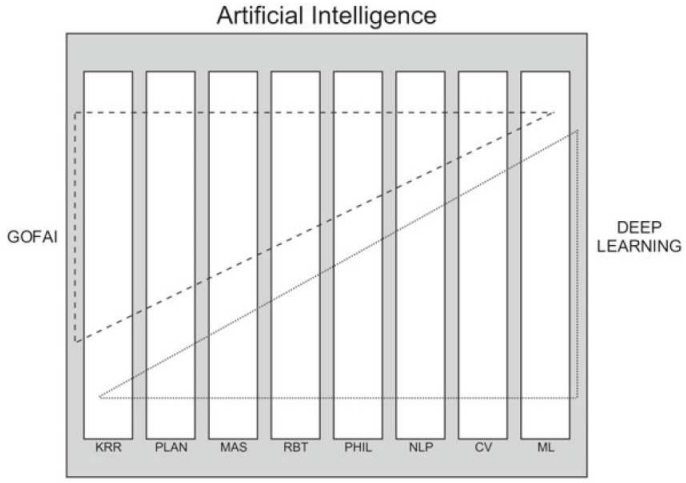
2006. Postoje dva veća društva koja daju formalnu klasifikaciju UI: *American Mathematical Society* (AMS) i *Association for Computing Machinery* (ACM). [11]

AMS se drži takozvane *Mathematics Subject Classification 2010* koja dijeli UI na sljedeća polja: općenito (*General*), prilagodljivi sustavi učenja (*Learning and adaptive systems*), prepoznavanje govora i uzoraka (*Pattern recognition and speech recognition*), dokazivanje teorema (*Theorem proving*), rješavanje problema (*Problem solving*), logika u UI (*Logic in artificial intelligence*), reprezentacija znanja (*Knowledge representation*), jezici i softverski sustavi (*Languages and software systems*), rasuđivanje uz nesigurnost (*Reasoning and uncertainty*), robotika (*Robotics*), tehnologija agenata (*Agent technology*), strojni vid i razumijevanje scene (*Machine vision and scene understanding*) te obrada prirodnog jezika (*Natural language processing*).

ACM daje klasifikaciju prema kojoj su osnovna polja UI: obrada prirodnog jezika (*Natural language processing*), reprezentacija znanja i rasuđivanje (*Knowledge representation and reasoning*), planiranje i raspoređivanje (*Planning and scheduling*), metodologije pretraživanja (*Search methodologies*), metode kontrole (*Control methods*), filozofska / teorijska uporišta UI (*Philosophical / theoretical foundations of AI*), distribuirana UI (*Distributed AI*) i strojni vid (*Computer vision*).

Iz ove dvije podjele, možemo zaključiti da je UI podijeljena na nekoliko širih klasa:

- reprezentacija znanja i rasuđivanje (*Knowledge representation and reasoning*, KRR)
- obrada prirodnog jezika (*Natural language processing*, NLP)
- strojno učenje (*Machine learning*, ML)
- planiranje (*Planning*, PLAN)
- sustavi s više agenata (*Multi-agent systems*, MAS)
- strojni vid (*Computer vision*, CV)
- robotika (*Robotics*, RBT)
- filozofski aspekti (*Philosophical aspects*, PHIL)



Slika 3.7: Prikaz odnosa GOFAI i dubokog učenja

Danas već postoje sustavi koji ne spadaju u nijednu ovdje navedenu kategoriju. Možemo zaključiti da se UI u današnje vrijeme brzo razvija i čak i da ovdje damo novu podjelu, za nekoliko bismo godina ponovo morali provjeravati i nadopunjavati ovaj popis. Pogledamo zato malo jednostavniju podjelu UI na duboko učenje i GOFAI. Kada klase na koje smo podijelili UI zamislimo kao vertikalne komponente, a GOFAI (Good Old-Fashioned AI — logička UI) i duboko učenje kao horizontalne komponente, možemo vidjeti da se GOFAI puno više bavi reprezentacijom znanja i rasuđivanjem nego kompjuterskim vidom, dok je kod dubokog učenja obrnuto. To je prikazano shematski na slici 3.7.

Bibliografija

[1] Djork Arné Clevert, Thomas Unterthiner i Sepp Hochreiter, *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*, 2016.

[2] Ema Dogančić, *Algoritmi strojnog učenja*, 2019., https://repozitorij.pmf.unizg.hr/islandora/object/pmf%3A8372/datastream/PDF/view.

[3] J. Haugeland, *Artificial Intelligence: The Very Idea*, The MIT Press, 1985.

[4] G. E. Hinton, *Deep belief networks*, Scholarpedia **4** (2009), br. 5, 5947, revision #91189.

[5] Iva Kresnik, *Umjetne neuronske mreže u prepoznavanju govora*, 2019., https://repozitorij.fsb.unizg.hr/islandora/object/fsb%3A5223/datastream/PDF/view.

[6] Philippe Lucidarme, *Most popular activation functions for deep learning*, https://lucidar.me/en/neural-networks/most-popular-activation-functions-for-deep-learning/.

[7] John McCarthy, *History of LISP*, History of programming languages, 1978, str. 173–185.

[8] John McCarthy, Marvin L Minsky, Nathaniel Rochester i Claude E Shannon, *A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955*, AI magazine **27** (2006), br. 4, 12–12.

[9] J. McCarthy, *Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part 1*, (1960), http://www-formal.stanford.edu/jmc/recursive.pdf.

[10] David E Rumelhart, Geoffrey E Hinton i Ronald J Williams, *Learning internal representations by error propagation*, Teh. izv., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[11] S. Skansi, *Introduction to Deep Learning: From logical Calculus to Artificial Intelligence*, Springer International Publishing AG, 2018.

[12] N. Wiener, *Cybernetics or the Control and Communication in the Animal and the Machine*, The MIT Press, 1961.

[13] Wikipedia contributors, *Ada Lovelace — Wikipedia, The Free Encyclopedia*, may 2021., https://en.wikipedia.org/wiki/Ada_Lovelace.

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

[15] _____, *Podržano učenje*, self-published, 2020, http://java.zemris.fer.hr/nastava/ui/rl/rl-20200401.pdf.

[16] _____, *Uvod u strojno učenje*, 2020, http://java.zemris.fer.hr/nastava/ui/ml/ml-20200410.pdf.

Sažetak

U prvom poglavlju opisuje se razdoblje u kojem se razvijaju prvi koncepti umjetne inteligencije i formiraju se osnove modernog računarstva. Znanstvenici organiziraju niz sastanaka na kojima raspravljaju o pitanjima inteligencije, razmišljanja i informacija te konačno formiraju znanost — kibernetiku. Neki od najutjecajnijih ljudi uključenih u ovo poglavlje su Norbert Wiener, Arturo Rosenblueth, John McCarthy, Marvin Minsky, Warren McCulloch i Walter Pitts.

Drugo poglavlje opisuje prijelaz sa konektivističkog na simbolički pristup proučavanju i razvoju računala. Prvi puta se umjetna inteligencija formira kao polje znanosti i uvodi se pojam GOFAI. Prikazuje se računalo kao automatizirani interpretirani formalni sustav. Kao jedni od najbitnijih ljudi navode se Charles Babbage, Alan Turing i John von Neumann zajedno s njihovim arhitekturama računala. Kao poseban tip arhitekture ističe se LISP koji je razvio John McCarthy.

U trećem poglavlju obrađuje se tema dubokog učenja, odnosno proces vraćanja konektivističkom pristupu i daljnjeg razvoja. Detaljno se obrađuje tema nadziranog učenja i neuronskih mreža, odnosno kako smo od jednog neurona došli do dubokih neuronskih mreža. Zaključuje se nedavnim uspjesima u svijetu umjetne inteligencije i podjelom umjetne inteligencije.

Summary

In the first chapter the time in which the first concepts of Artificial Intelligence are developed and the bases of modern computer science are formed is described. Scientists organize a series of meetings in which they debate on intelligence, thinking and information, and finnaly they form a science — Cybernetics. Some of the most notable people included in this chapter are Norbert Wiener, Arturo Rosenblueth, John McCarthy, Marvin Minsky, Warren McCulloch and Walter Pitts.

The second chapter revolves around the change from connectivist AI to symbolic AI. Artificial Intelligence is formed as a field of science for the first time, and GOFAI is introduced. A computer is represented as an automated interpreted formal system. Charles Babbage, Alan Turing and John von Neumann, as some of the most important people of that time, and the computer architectures they made are mentioned. John McCarthy's LISP stands out as a special type of architecture.

The third chapter deals with the topic of deep learning, and more of the process of returning to the connectivist approach and further developement. The topic of supervised learning and neural networks is discussed in detail, ie how we came from one neuron to deep neural networks. It concludes with recent successes in the world of artificial intelligence and the division of artificial intelligence.

Životopis

- **OSOBNI PODACI**

  Ime i prezime: Antonija Pantar

  Adresa: Lešće 25, 10430 Samobor

  Država: Hrvatska

  Telefon: 091 332 6502

  Email: antonija.hrcak.pantar@gmail.com

  Mjesto i datum rođenja: Zagreb, 14. travnja 1995.

- **OBRAZOVANJE**

  Stečena kvalifikacija: sveučilišna prvostupnica matematike

  Vrijeme (od–do): 2013.-2017.

  Institucija: Prirodoslovno-matematički fakultet, Horvatovac 102A, 10000 Zagreb

- **RADNO ISKUSTVO**

  Vrijeme (od–do): 11/2017 – 08/2019

  Poslodavac: OŠ Ljudevit Gaj, Zaprešić

  Radno mjesto: Nastavnik matematike

- **DODATNA ZNANJA**

  Strani jezici: Engleski

  Ostale sposobnosti: Rad na računalu, programski jezici C, C++, Python, MS Office, komunikacijski programi, društvene mreže.

  Vozačka dozvola: B kategorija