POGLAVLJE 10

Uvod u veštačke neuronske mreže sa Kerasom

Ptice su nas nadahnule da poletimo, čičak je doveo do pronalaska čičak trake, a sama priroda je bila uzor za bezbroj drugih pronalazaka. Čini se sasvim logično da u arhitekturi mozga tražimo inspiraciju o tome kako da izgradimo inteligentnu mašinu. To je logika koja je pokrenula *veštačke neuronske mreže* (eng. *Artificial Neuron Networks*, *ANN*): veštačka neuronska mreža je model mašinskog učenja inspirisan mrežama bioloških neurona koje postoje u našim mozgovima. Međutim, mada su avionima uzor bile ptice, oni ne moraju da mašu svojim krilima. Slično tome, veštačke neuronske mreže su vremenom postale prilično drugačije od svojih bioloških rođaka. Neki istraživači tvrde čak i da bi trebalo da u potpunosti odbacimo biološku analogiju (npr. da govorimo „jedinice" umesto „neuroni"), da ne bismo svoju kreativnost ograničili samo sisteme koji su biološki mogući.[1]

Veštačke neuronske mreže čine samo jezgro *dubokog učenja* (eng. *Deep Learning*). One su raznovrsne, moćne i skalabilne, što ih čini savršenim za obavljanje obimnih i vrlo složenih zadataka mašinskog učenja kao što su klasifikovanje milijardi slika (npr. Google Images), pružanje usluga prepoznavanja govora (npr. Appleov Siri), preporučivanje najboljih video snimaka stotinama miliona korisnika svakog dana (npr. na YouTubeu) ili učenje kako pobediti svetskog šampina u igri Go (DeepMindov AlphaGo).

Prvi deo ovog poglavlja uvodi veštačke neuronske mreže, tako što prvo ukratko opisuje rane arhitekture ANN a zatim dolazi do *višeslojnih perceptrona* (eng. *Multilayer Perceptrons*, *MLP*), koji se danas široko koriste (druge arhitekture su objašnjenje u poglavljima koja slede). U drugom delu, razmotrićemo implementiranje neuronskih mreža pomoću popularnog API-ja Keras. To je vrlo lepo dizajniran i jednostavan API visokog nivoa za izradu, obuku, procenjivanje i upotrebu neuronskih mreža. Ali neka vas ne zavarava njegova jednostavnost: taj API je dovoljno izražajan i fleksibilan za izradu široke lepeze različitih arhitektura neuronskih mreža. U stvari, on će vam verovatno biti dovoljan za većinu vaših slučajeva upotrebe. A ako vam ikad zatreba dodatna fleksibilnost, uvek možete napisati prilagođene Keras komponente pomoću njegovog API-ja nižeg nivoa, kao što ćemo videti u poglavlju 12.

Ali pre toga, vratićemo se prošlost da bismo videli kako su nastale veštačke neuronske mreže!

[1] Možete izvući najbolje iz oba sveta ako ste otvoreni za biološke inspiracije i ne plašite se da pravite biološki nerealne modele, sve dok oni rade ispravno.

Od bioloških do veštačkih neurona

Iznenađuje činjenica da veštačke neuronske mreže postoje već prilično dugo: prvi put su ih predstavili 1943. godine neurofiziolog Warren McCulloch i matematičar Walter Pitts. U svom referentnom radu[2] „A Logical Calculus of Ideas Immanent in Nervous Activity" (https://homl.info/43), McCulloch i Pitts u predstavili pojednostavljen matematički model načina na koji bi biološki neuroni mogli da sarađuju u mozgovima životinja da bi obavljali složene računske operacije koristeći *logiku zasnovanu na tvrdnjama* (eng. *propositional logic*). To je bila prva arhitektura veštačke neuronske mreže. Posle nje su izmišljene i mnoge druge arhitekture, kao što ćemo videti.

Rani uspesi veštačkih neuronskih mreža doveli su do vrlo raširenog verovanja da ćemo uskoro pričati sa zaista inteligentnim mašinama. Kada je u šezdestim godinama postalo jasno da će to obećanje ostati neispunjeno (barem još duže vreme), novčana podrška je preusmerena na druge strane, a veštačke neuronske mreže su upale u dugu zimu. Ranih osamdesetih godina pronađene su nove arhitekture i razvijene bolje tehnike obuke, što je ponovo probudilo zanimanje za *konekcionizam* (eng. *connectionism*) (proučavanje neuronskih mreža). Ali napredak je bio spor do devedestih godina, kada su otkrivene druge moćne tehnike mašinskog učenja, kao što su mašine s vektorima podrške (poglavlje 5). Činilo se da te tehnike pružaju bolje rezultate i jače teorijske osnove nego veštačke neuronske mreže, pa je zato još jednom proučavanje neuronskih mreža zapostavljeno.

Danas smo svedoci još jednog talasa zanimanja za veštačke neuronske mreže. Da li će taj talas isčeznuti kao što su prethodni isčezli? Pa, postoji nekoliko dobrih razloga da verujemo kako je ovog puta drugačije i da će obnovljeno zanimanje za veštačke neuronske mreže imati znatno veći uticaj na naše živote:

- Sada imamo na raspolaganju ogromne količine podataka za obuku neuronskih mreža i veštačke neuronske mreže često nadmašuju druge tehnike MU u slučajevima veoma obimnih i složenih problema.
- Fantatstičan porast računarske snage od devedestih godina do danas omogućava obuku velikih neuronskih mreža u razumnom vremenu. Delimičan razlog tome je Mooreov zakon (broj komponenata u integrisanim kolima se udvostričivao svake dve godine, poslednjih 50 godina), ali zahvaljujući i industriji računarskih igara, koja je podstakla proizvodnju miliona komada moćnih GPU kartica. Osim toga, platforme za rad u oblaku su omogućile da ta moć bude dostupna svakome.
- Poboljšani su algoritmi za obuku. Pošteno govoreći, danas su samo neznatno drugačiji od onih koji su se koristili devedesetih godina, ali i ta relativno mala poboljšanja imala su ogroman pozitivan efekat.
- Određena teorijska ograničenja veštačkih neuronskih mreža pokazalal su se u praksi kao lako rešiva. Na primer, mnogi ljudi su mislili da su algoritmi za obuku ANN bili

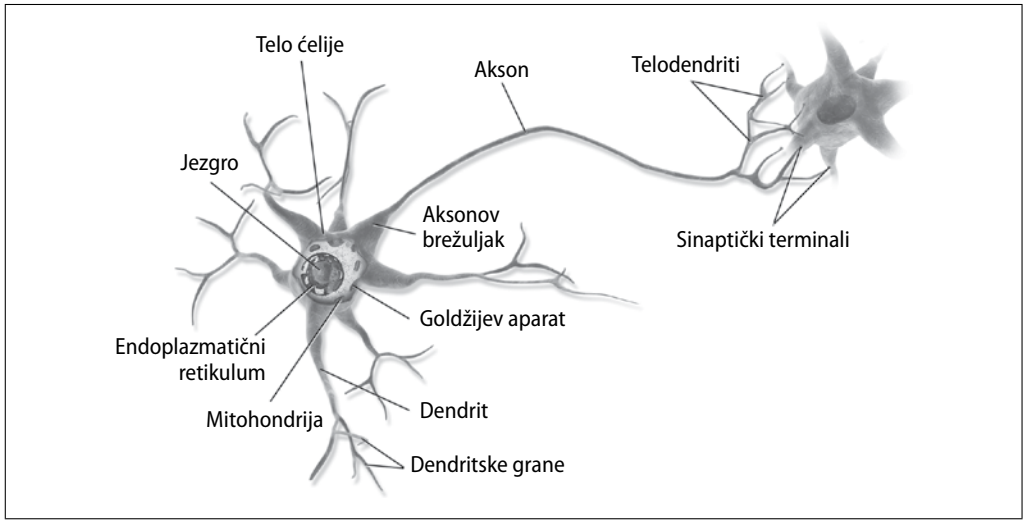
[2] Warren S. McCulloch i Walter Pitts, „A Logical Calculus of the Ideas Immanent in Nervous Activity„ *The Bulletin of Mathematical Biology* 5, br. 4 (1943): 115–113.

neupotrebljivi zato što se dešavalo da se zaglave na nekom lokalnom optimumu, ali se ispostavilo da je to u praksi redak slučaj (a kada se to i dogodi, uglavnom je prilično blizu globalnom optimumu).

- Izgleda kao da su veštačke neuronske mreže upale u savršen krug finansiranja i napretka. Nerverovatni proizvodi zasnovani na veštačkim neuronskim mrežama redovno su teme glavnih naslova u medjima, što na njih privlači još više pažnje i novčanih ulaganja, što rezultuje novim napretkom i novim još neverovatnijim proizvodima.

Biološki neuroni

Pre nego što pređemo na veštačke neurone, razmotrićemo ukratko biološki neuron (prikazan je na slici 10-1). To je ćelija neobičnog izgleda, kakve se obično nalaze u životinjskom mozgu. Sastoji se od *tela ćelije* koje sadrži jezgro i najveći deo ostalih složenih komponenata ćelije, zatim više granastih izraslina – *dendrita* i jednu veoma dugačku izraslinu – *akson*. *Dužina* aksona može biti nekoliko dužina tela ćelije ili više desetina hiljada puta. Akson se na svom kraju deli na više grana – *telodendrita*, a na krajevima tih grana nalaze se sićušne strukture – *sinaptički terminali* (ili samo *sinapse*), koje su povezane s dendritima ili telima ćelija drugih neurona.[3] Biološki neuroni proizvode kratke električne impulse – *akcione potencijale* (AP ili samo *signali*) koji putuju duž aksona i čine da sinapse šalju hemijske signale – *neurotransmitere*. Kada neuron primi dovoljnu količinu neurotransmitera u roku od nekoliko milisekundi, on šalje vlastite električne impulse (to zavisi od vrste neurotransmitera, pošto neki od njih sprečavaju da se neuron aktivira).

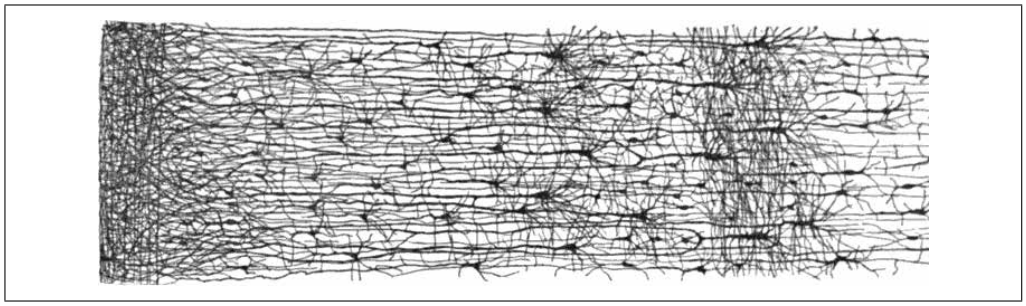


Slika 10-1. Biološki neuron[4] (slika u boji na www.mikroknjiga.rs)

[3] Nisu zaista povezani, nego su samo dovoljno blizu da mogu da razmenjuju hemijske signale.

[4] Autor slike je Bruce Blaus (Creative Commons 3.0 (*https://creativecommons.org/licenses/by/3.0/*)). Reprodukovano sa *https://en.wikipedia.org/wiki/Neuron*.

Dakle, izgleda kao da je ponašanje pojedinačnih bioloških neurona prilično jednostavno, ali su oni organizovani u obimne mreže od više milijardi njih, gde je svaki neuron obično povezan s hiljadama drugih neurona. Mreža sasvim jednostavnih neurona može da obavlja veoma složene proračune, vrlo slično kao kada složen mravinjak nastane kombinovanim naporima jednostavnih mrava. Arhitektura *bioloških neuronskih mreža* (eng. *Biological Neuron Netvorks*, *BNN*)[5] i dalje je predmet aktivnih istraživanja, ali razjašnjeni su neki delovi mozga i izgleda kao da su neuroni često organizovani u zaporedne slojeve, naročito u cerebralnom korteksu (tj. u spoljašnjoj kori mozga), kao što je prikazano na slici 10-2.



Slika 10-2. Više slojeva biološke neuronske mreže (korteks čovečijeg mozga)[6]

Logička izračunavanja pomoću neurona

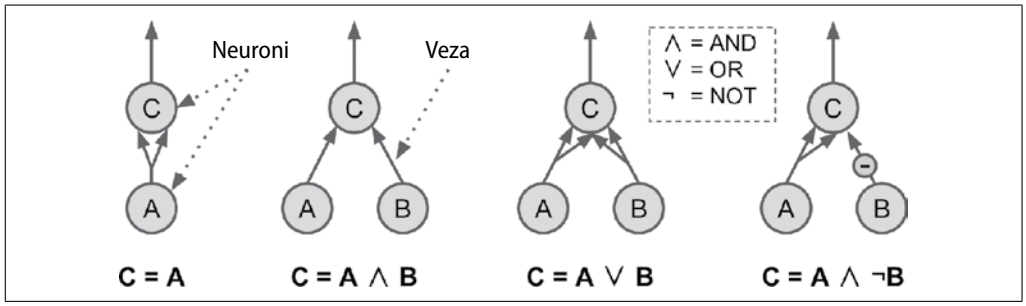
McCulloch i Pitts su predložili vrlo jednostavan model biološkog neurona, koji je kasnije postao poznat kao *veštački neuron* (eng. *artificial neuron*): on ima jedan ili više binarnih (uključeno/isključeno) ulaza i jedan binarni izlaz. Veštački neuron aktivira svoj izlaz kada je aktivno više od određenog broja njegovih ulaza. U svom radu dokazali su da je čak i s takvim pojednostavljenim modelom moguće izgraditi mrežu veštačkih neurona koja izračunava proizvoljnu logičku tvrdnju. Da biste videli kako radi jedna takva mreža, napravićemo nekoliko veštačkih neuronskih mreža koje obavljaju razne logičke proračune (slika 10-3), s pretpostavkom da je neuron aktiviran ako su barem dva njegova ulaza aktivna.

Razmotrimo šta rade ove mreže:

- Prva mreža s leva je identitetska funkcija: ako je aktiviran neuron A, aktivira se i neuron C (pošto dobija dva ulazna sigala od neurona A); ali ako neuron A nije aktivan, neaktivan je i neuron C.
- Druga mreža obavlja logičku operaciju AND: neuron C se aktivira samo kada su aktivirana oba neurona A i B (samo jedan ulazni signal nije dovoljan da aktivira neuron C).
- Treća mreža obavlja logičku operaciju OR: neuron C se aktivira ako je aktiviran neuron A ili neuron B (ili oba).

[5] U kontekstu mašinskog učenja, izraz „neuronska mreža" uglavnim se odnosi ANN, ne na BNN.

[6] Crtež kortikoidnog laminata čiji autor je S. Ramon y Cajal (u javnom vlasništvu). Reprodukovano sa *https://en.wikipedia.org/wiki/Cerebral_cortex*.



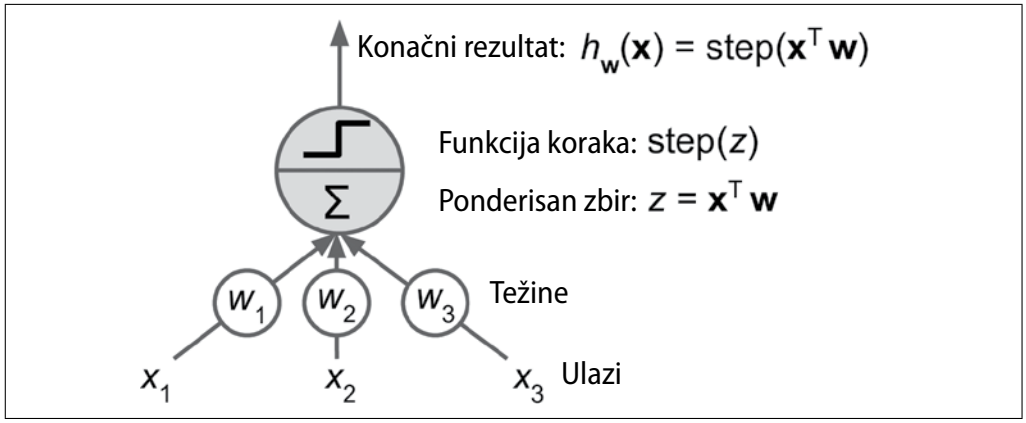
Slika 10-3. Veštačke neuronske mreže koje obavljaju jednostavne logičke proračune

- I najzad, ako pretpostavimo da jedna od ulaznih veza može da spreči aktiviranje neurona (što je slučaj kod bioloških neurona), onda četvrta mreža obavlja nešto složeniju logičku operaciju: neuron C se aktivira samo ako je aktivan neuron A, a neuron B nije. Ako je neuron A aktivan celo vreme, dobijate logičko NOT: neuron C je aktivan kada neuron B nije i obrnuto.

Možete zamisliti kako se ove mreže mogu kombinovati radi izračunavanja složenih logičkih izraza (primer ćete naći u vežbama na kraju ovog poglavlja).

Perceptron

Perceptron je je jedna od najjednostavnijih arhitektura ANN, a izumeo ga je Frank Rosenblatt 1957. godine. Zasniva se nešto drugačijem veštačkom neuronu (slika 10-4) koji se zove *jedinica praga logike* (eng. *threshold logic unit*, *TLU*) ili ponekad *jedinica linearnog praga* (eng. *linear threshold unit*, *LTU*). Ulazi i izlazi su brojevi (umesto binarnih vrednosti uključeno/isključeno) a svakom ulazu je pridružena određena težina. TLU izračunava ponderisan zbir svojih ulaza ($z = w_1 x_1 + w_2 x_2 + \ldots + wn\, xn = \mathbf{x}^\top \mathbf{w}$), a zatim na taj zbir primenjuje *funkciju koraka* (eng. *step function*) da bi proizveo konačan rezultat: $h\mathbf{w}(\mathbf{x}) = \text{step}(z)$, gde $z = \mathbf{x}^\top \mathbf{w}$.



Slika 10-4. Jedinica praga logike: veštački neuron koji izračunava ponderisani zbir svojih ulaza, na koji zatim primenjuje funkciju koraka

Najuobičajenija funkcija koraka koja se koristi u perceptronima je Hevisajdova (*Heaviside*) *funkcija koraka* (jednačina 10-1). Ponekad se umesto nje koristi funkcija *predznak* (eng. *sign*).

Jednačina 10-1 Uobičajene funkcije koraka koje se koriste u perceptronima (ovde je prag = 0)

$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

Jedan TLU može se upotrebiti za jednostavnu linearnu binarnu klasifikaciju. Izračunava linearnu kombinaciju ulaza i ako rezultat premašuje određeni prag, vraća pozitivnu klasu. U suprotnom slučaju vraća negativnu klasu (potpuno isto kao klasifikator logistička regresija ili mašina s vektorima podrške). Na primer, možete upotrebiti jedan TLU za klasifikovanje cvetova perunike na osnovu dužine i širine latice (dodajte mu još i dopunski otklon $x_0 = 1$, isto kao što smo radili u prethodnim poglavljima). Obučavanje TLU-a u ovom slučaju znači pronalaženje odgovarajućih vrednosti za w_0, w_1 i w_2 (algoritam za obuku razmatramo u nastavku teksta).

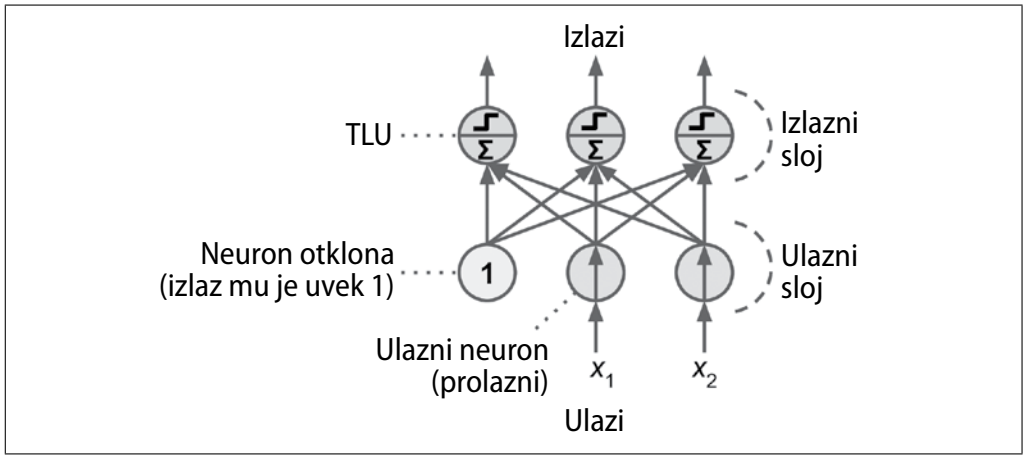
Perceptron se sastoji jednog sloja koji se sastoji od više TLU-ova,[7] gde je svaki TLU povezan sa svim ulazima. Kada su svi neuroni u jednom sloju povezani sa svakim neuronom u prethodnom sloju (tj. sa neuronima na svojim ulazima), to se zove *potpuno povezan sloj* (eng. *fully connected layer*) ili a *gusti sloj* (eng. *dense layer*). Perceptron prima ulazne podatke od specijalnih prolaznih neurona koji se zovu *ulazni neuroni* (eng. *input neurons*): oni prosleđuju dalje sve što prime na svojim ulazima. Svi ulazni neuroni čine *ulazni sloj* (engl *input layer*). Osim toga, najčešće se dodaje još jedan dopunski član ($x_0 = 1$): obično predstavlja se pomoću specijalne vrste neurona koji se zove *neuron otklona* (eng. *bias neuron*), čiji izlaz je uvek 1. Na slici 10-5 prilazan je perceptron s dva ulaza i tri izlaza. Taj perceptron može da razvrstava primere u tri različite binarne klase istovremeno, što ga čini višerezultatnim klasifikatorom.

Zahvaljujući čaroliji linearne algebre, jednačina 10-2 omogućava efikasno izračunavanje vrednosti na izlazima jednog sloja veštačkih neurona za više primera istovremeno.

Jednačina 10-2 Izračunavanje vrednosti na izlazima jednog potpuno povezanog sloja

$$h_{\mathbf{W},\mathbf{b}}(\mathbf{X}) = \phi(\mathbf{X}\mathbf{W} + \mathbf{b})$$

[7] Naziv *perceptron* se ponekad koristi sa značenjem sićušne mreže koj se sastoji od samo jednog TLU-a.



Slika 10-5. Arhitektura perceptrona s dva ulazna neurona, jednim neuronom otklona i tri izlazna neurona

U ovoj jednačini:

- Kao i obično, **X** predstavlja matricu ulaznih osobina. Sadrži jedan red po primeru i jednu kolonu po osobini.
- Matrica težina **W** sadrži težine svih veza osim neurona otklona. Sadrži jedan red po ulaznom neuronu i jednu kolonu po veštačkom neuronu u sloju.
- Vektor otklona **b** sadrži sve težine veza između neurona otklona i veštačkih neurona. Sadrži po jedan razdvojni član za svaki veštački neuron.
- Funkcija ϕ se zove *aktivirajuća funkcija* (eng. *activation function*): kada su veštački neuroni TLU komponente, to je funkcija koraka (ali u nastavku teksta razmotrićemo i druge aktivirajuće funkcije).

Dakle, kako se obučava perceptron? Algoritam za obuku perceptrona, koji je predložio Rosenblatt, najvećim delom je inspirisan *Hebovim pravilom* (eng. *Hebb's rule*). U svojoj knjizi iz 1949. godine, *The Organization of Behavior* (izdavač je Wiley), Donald Hebb je naveo da kada jedan biološki neuron često aktivira drugi neuron, veza između ta dva neurona postaje sve jača. Siegrid Löwel je kasnije sveo Hebbovu ideju na lako pamtljiv izraz, „Ćelije koje okidaju zajedno, vezuju se zajedno"; odnosno, težina veze između dva neurona raste kada se oni aktiviraju istovremeno. Ovo pravilo je kasnije postalo poznato kao Hebovo pravilo ili *Hebovo učenje* (eng. *Hebbian learning*). Perceptroni se obučavaju pomoću varijante ovog pravila koja uzima u obzir grešku koju mreža pravi kada izračunava predviđanje; pravilo za učenje perceptrona ojačava veze koje doprinose smanjivanju greške. Konkretno, perceptronu se prosleđuju primeri jedan po jedan i za svaki od njih, on izračunava svoje predviđanje. U svakom izlaznom neuronu koji je dao pogrešno predviđanje, pojačava težinu veza na ulazima koji bi doprineli tačnom predviđanju. To pravilo je prikazano u jednačini 10-3.

Jednačina 10-3 Pravilo za obuku perceptrona (ažuriranje težina)

$$w_{i,j}^{(\text{sledeći korak})} = w_{i,j} + \eta\left(y_j - \hat{y}_j\right)x_i$$

U ovoj jednačini:

- $w_{i,j}$ je težina veze između i-tog ulaznog neurona i j-tog izlaznog neurona.
- x_i je i-ta ulazna vrednost tekućeg primera za obuku.
- $\hat{y}$ je rezultat na izlazu j-tog izlaznog neurona za tekući primer za obuku.
- y_j je ciljni rezultat j-tog izlaznog neurona za tekući primer za obuku.
- η je brzina učenja.

Pošto je granica odlučivanja svakog izlaznog neurona linearna, perceptroni nisu u stanju da nauče složene šablone (isto kao i klasifikatori logistička regresija). Međutim, ako su primeri za obuku linearno razdvojivi, Rosenblatt je dokazao da bi ovaj algoritam konvergirao ka rešenju.[8] To se zove *teorema konvergencije perceptrona* (eng. *Perceptron convergence theorem*).

Scikit-Learn sadrži klasu `Perceptron` koja implementira mrežu od jednog TLU-a koja se može koristiti otprilike kao što biste i očekivali–na primer, na skupu podataka iris (uveden je u poglavlju 4):

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron

iris = load_iris()
X = iris.data[:, (2, 3)]  # dužina latice, širina latice
y = (iris.target == 0).astype(np.int)  # Iris setosa?

per_clf = Perceptron()
per_clf.fit(X, y)

y_pred = per_clf.predict([[2, 0.5]])
```

Možda ste zapazili da algoritam za obuku perceptrona mnogo liči na stohastički gradijentni spust. U stvari, klasa `Perceptron` u biblioteci Scikit-Learn ekvialentna je upotrebi klasifikatora `SGDClassifier` sa sledećim hiperparametrima: `loss="perceptron"`, `learning_rate="constant"`, `eta0=1` (brzina učenja) i `penalty=None` (bez regularizacije).

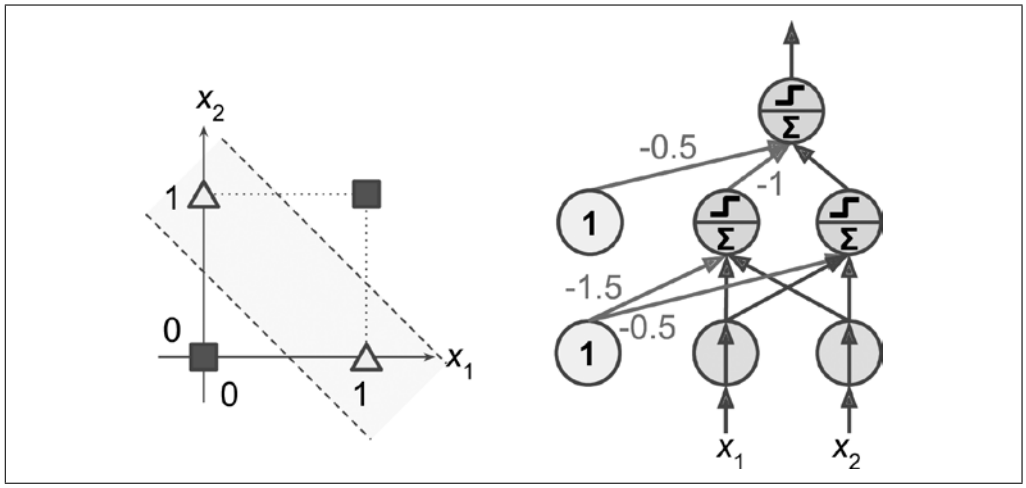
Obratite pažnju na to da za razliku od klasifikatora logistička regresija, perceptroni ne vraćaju verovatnoću pripadanja određenoj klasi nego izračunavaju predviđanja na osnovu zadatog praga. To je jedan od razloga prednosti logističke regresije nad preceptronom.

U monografiji *Perceptrons* izdatoj 1969. godine, Marvin Minsky i Seymour Papert su istakli više ozbiljnih slabosti perceptrona–konkretno, činjenicu da nisu u stanju da reše neke trivi-

[8] Imajte u vidu da rešenje nije jedinstveno: kada su podaci linearno razdvojivi, postoji beskonačan broj hiperravni koje ih mogu razdvojiti.

jalne probleme (npr. problem *Exclusive OR*, tj. XOR klasifikacije); leva strana na slici 10-6). To važi i za svaki drugi linearni klasifikacijski model (kao što su klasifikatori logistička regresija), ali su istraživači toliko mnogo očekvali od perceptrona i toliko su bili razočarani da su neki od njih u potpunosti odbacili neuronske mreže u korist problema višeg nivoa, kao što su logika, rešavanje problema i pretraživanje.

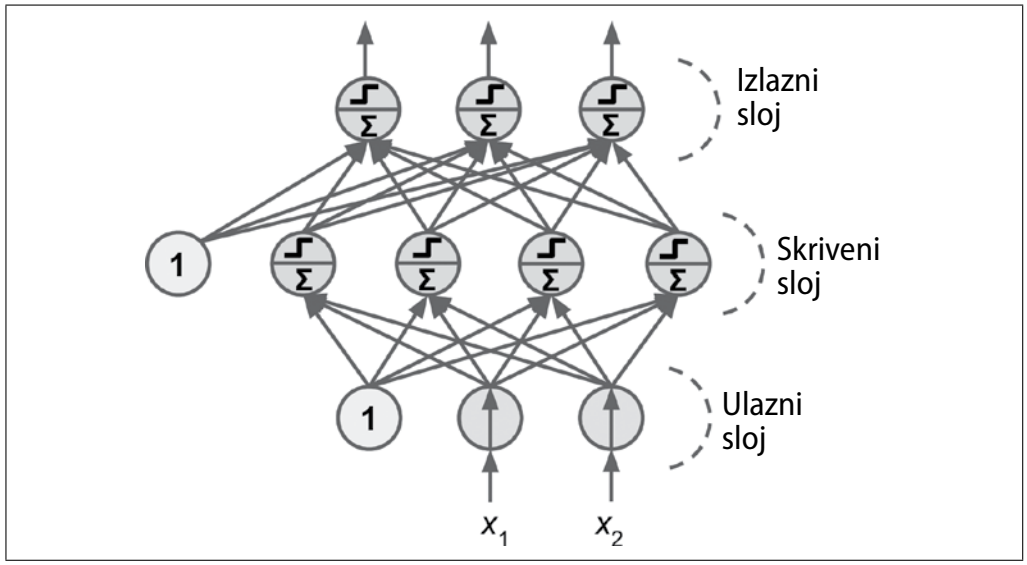
Ispostavilo se da se neka ograničenja perceptrona mogu zaobići kombinovanjem više perceptrona. Rezultujuća mreža veštačkih neurona nazvana je *višeslojni perceptron* (eng. *Multilayer Perceptron*, MLP). MLP može da reši XOR problem, u šta se možete uveriti ako izračunate izlaz MLP-a koji je prikazan na desnoj strani slike 10-6: kada na ulazima dobije (0, 0) ili (1, 1), mreža vraća 0, dok za (0, 1) ili (1, 0) na ulazima vraća 1. Svim vezama je pridružen težina 1, osim za četiri veze za koje je prikazana drugačija vrednost. Pokušajte da proverite da ova mreža zaista rešava XOR problem!



Slika 10-6. Problem XOR klasifikacije i MLP koji ga rešava (slika u boji na www.mikroknjiga.rs)

Višeslojni perceptron i povratna propagacija

Jedan višeslojni perceptron (MLP) se sastoji od jednog (prolaznog) *ulaznog sloja* (eng. *input layer*) jednog ili više slojeva TLU komponenata – *skrivenih slojeva* (eng. *hidden layers*) i jednog završnog sloja TLU komponenata – *izlaznog sloja* (eng. *output layer*) (slika 10-7). Slojevi koji se nalaze bliže ulaznom sloju zovu se *niži slojevi* (eng. *lower layers*) a oni bliži izlazima – *viši slojevi* (eng. *upper layers*). Svaki sloj, osim izlaznog sloja, sadrži neuron otklona i potpuno je povezan s narednim slojem.



Slika 10-7. Arhitektura višeslojnog perceptrona sa dva ulaza, jednim skrivenim slojem sa četiri neurona i tri izlazna neurona (prikazani su neuroni otklona, ali su oni obično implicitni)

Signal protiče samo u jednom smeru (od ulaza ka izlazima), tako da je ova arhitektura primer neuronske mreže s prosleđivanjem prosleđivanja unapred (eng. *feedforward neural network*, FNN).

Kada jedna mreža veštačkih neurona sadrži veliki broj skrivenih slojeva,[9] zove se *duboka neuronska mreža* (eng. *deep neural network*, DNN). Oblast dubokog učenja bavi se proučavanjem dubokih neuronskih mreža i drugih modela koji sadrže velike brojeve slojeva proračuna. Uprkos tome, mnogi ljudi govore o dubokom učenju kad god su u pitanju neuronske mreže (čak i one plitke).

Istraživači su se godinama bezuspešno borili da nađu način za obučavanje višeslojnih perceptrona. Ali 1986. gdine, David Rumelhart, Geoffrey Hinton i Ronald Williams su objavili kapitalan rad (*https://homl.info/44*)[10] koji je uveo algoritam za obuku sa *povratnom propagacijom* (eng. *backpropagation*), koji se i danas koristi. Ukratko, to je gradijentni spust (uveden je u poglavlju 4) koji koristi efikasnu tehniku za automatsko izračunavanje gradijenata:[11] u samo dva prolaza kroz mrežu (jedan unapred, drugi u suprotnom smeru), algoritam za

[9] 90-ih godina, MVN s više od dva skrivena sloja smatrala se dubokom mrežom. U današnje vreme, sasvim su uobičajene mreže veštačkih neurona s više desetina, pa čak i stotina slojeva, tako da je definicija šta je „duboko" prilično neodređena.

[10] David Rumelhart i dr. „Learning Internal Representations by Error Propagation," (Defense Technical Information Center technical report, septembar 1985.)

[11] Ovu tehniku je zapravo pronašlo, nezavisno jedni od drugih, više puta više istraživača u različitim oblastima, a prvi je bio Paul Werbos 1974. godine.

povratnu propagaciju je u stanju da izračuna gradijent greške mreže u odnosu na svaki pojedinačni parametar modela. Drugim rečima, algoritam može da pronađe kako treba podesiti težinu svake veze i svaki član pomeranja da bi se greška smanjila. Kada izračuna te gradijente, izvrši još samo korak običnog gradijentnog spusta i ceo postupak se ponavlja dok mreža ne konvergira ka rešenju.

Automatsko izračunavanje gradijenata zove se *automatsko diferenciranje* (eng. *automatic differentiation*) ili *autodif*. Postoje razne autodif tehnike, svaka sa svojim prednostima i manama. Ona koja se koristi u povratnoj propagaciji zove se *autodif obrnutog režima* (eng. *reverse-mode autodiff*). Brza je i precizna, pogodna kada funkcija koja se diferencira ima veliki broj promenljivih (npr. težine) a mali broj izlaznih rezultata (npr. jedan gubitak). Ako želite da saznate više o autodifu, pogledajte dodatak A.

Sada ćemo detaljnije razmotriti ovaj algoritam:

- Prihvata jedan po jedan mini paket (svaki sadrži, recimo, 32 primera) i više puta prolazi kroz ceo skup za obuku. Svaki prolaz se zove *ciklus* (eng. *epoch*).
- Svaki mini paket se prosleđuje ulaznom sloju mreže, koji ga šalje prvom skrivenom sloju. Algoritam zatim izračunava rezultat svih neurona u tom sloju (za svaki primer u mini paketu). Taj rezultat prosleđuje narednom sloju, izračunava njegov rezultat i prosleđuje ga narednom sloju i tako sve dok ne dobijemo rezultat poslednjeg sloja, tj. izlaznog sloja. Ovo je *prolaz napred* (eng. *forward pass*): isto je kao pravljenje predviđanja, osim što se međurezultati čuvaju jer će biti potrebni u povratnom prolazu.
- Dalje, algoritam meri grešku izlaznog rezultata cele mreže (tj. koristi funkciju gubitka koja poredi očekivani rezultat sa stvarnim rezultatom mreže i vraća određenu meru greške).
- Zatim izračunava koliko je svaka izlazna veza doprinela grešci. To radi analitički, primenom *lančanog pravila* (eng. *chain rule*) (možda najvažnijeg pravila u računanju), koje ovaj korak čini brzim i preciznim.
- Algoritam onda meri koliko od tih doprinosa potiče iz svake pojedinačne veze u sloju ispod, ponovo primenjujući lančano pravilo. Tako algoritam radi unazad sve dok ne dođe do ulaznog sloja. Kao što je ranije obašnjeno, ovaj prolaz unazad efikasno meri gradijent greške za težine sveke veze unutar mreže tako što gradijent greške prosleđuje unazad kroz mrežu (otuda i ime algoritma).
- I najzad, algoritam izvršava korak gradijentnog spusta da bi podesio sve težine u mreži, pomoću gradijenata grešaka koje je upravo izračunao.

Ovaj algoritam je toliko važan da vredi da ga još jednom ukratko opišemo: za svaki primer za obuku, algoritam povratne propagacije prvo izračunava predviđanje (prolaz napred) i meri grešku, zatim prolazi kroz svaki sloj u smeru nazad da bi izmerio doprinos grešci svake veze (obrnut prolaz) i na kraju podešava težine svih veza da bi smanjio grešku (korak gradijentng spusta).

Važno je da težine veza unutar skrivenih slojeva dobiju nasumično izabrane početne vrednosti jer inače obuka neće uspeti. Na primer, ako sve težine i članove odstupanja inicializujete na nulu, onda će svi neuroni u datom sloju biti potpuno identiični, pa će zato povratna propagacija delovati na sve njih na potpuno isti način, usled čega će oni ostati identični. Drugim rečima, uprkos stotinama neurona po sloju, vaš model će se ponašati kao da ima samo jedan neuron po sloju: neće biti previše pametan. Ako umesto toga težine inicializujete nasumičnim vrednostima, razbićete simetriju (eng. *break the symmetry*) i omogućiti povratnoj propagaciji da obučava različite grupe neurona.

Da bi ovaj algoritam radio ispravno, njegovi autori su načinili ključnu izmenu arhitekture višeslojnih perceptrona: funkciju koraka su zamenili logističkom (sigmoidnom) funkcijom, $\sigma(z) = 1 / (1 + \exp(-z))$. To je bilo ključno zato što funkcija koraka sadrži samo ravne segmente, pa zato nema gradijenta sa kojim bi se radilo (gradijentni spust je neizvodljiv na ravnoj površini), dok logistička funkcija ima u svakoj svojoj tački dobro definisan izvod različit od nule, što algoritmu gradijentnog spusta omogućava da u određenoj meri napreduje u svakom koraku. U stvari, algoritam povratne propagacije radi dobro i s mnogim drugim aktivirajućim funkcijama, ne samo s logističkom funkcijom. Druga dva popularna izbora su:

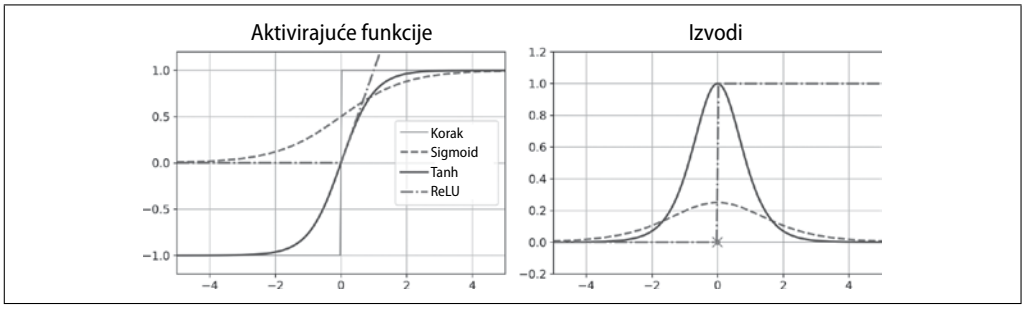
Funkcija hiperbolični tangens: $\tanh(z) = 2\sigma(2z) - 1$
: Isto kao i logistička funkcija, ova aktivirajuća funkcija ima oblik slova S, neprekidna je i diferencijabilna, ali njena izlazna vrednost se nalazi u opsegu od –1 do 1 (umesto od 0 do 1 kao logistička funkcije). Taj opseg čini da na početku obuke izlaz svakog sloja bude više ili manje centriran oko nule, što često ubrzava konvergiranje algoritma.

Ispravljena linearna jedinična funkcija: $\mathrm{ReLU}(z) = \max(0, z)$
: Funkcija ReLU je neprekidna ali nažalost nije diferencijabilna u tački $z = 0$ (nagib se skokovito menja, što može dovesti do toga da algoritam gradijentnog spusta skakuće oko te tačke), a njen izvod je 0 za $z < 0$. Međutim, u praksi radi vrlo dobro i pruža tu prednost da se brzo izračunava, pa je zato ova funkcija postala standardni početni izbor.[12] Što je još važnije, činjenica da nema maksimalnu izlaznu vrednost doprinosi ublažavanju nekih problema tokom gradijentnog spusta (vratićemo se na ovu temu u poglavlju 11).

Ove popularne aktivirajuće funkcije i njihovi izvodi predstavljeni su na slici 10-8. Ali čekajte! Zašto nam uopšte trebaju aktivirajuće funkcije? Pa, ako nadovežete više linearnih transformacija u lanac, sve što ćete dobiti je samo linearna transformacija. Na primer, ako $f(x) = 2x + 3$ i $g(x) = 5x - 1$, nadovezivanjem ove dve linearne funkcije dobijate novu linearnu funkciju: $f(g(x)) = 2(5x - 1) + 3 = 10x + 1$. Zbog toga, ako između slojeva ne uvedete određenu nelinearnost, onda je čak i veliki broj slojeva ekvialentan jednom sloju, s kojim ne možete rešavati

[12] Pošto izgleda kao da biološki neuroni implementiraju grubo sigmoidnu aktivirajuću funkciju (u obliku slova *S*), istraživači su se veoma dugo vremena držali sigmoidnih funkcija. Ali ispostavilo se da je za mreže veštačkih neurona uglavnom bolja funkcija ReLU. Ovo je jedan od slučajeva kada je biološka analogija vodlia u pogrešnom pravcu.

mnogo složene probleme. Nasuprot tome, dovoljno veliki DNN s nelineranim aktivacijama može da aproksimira teorijski svaku neprekidnu funkciju.



Slika 10-8. Aktivirajuće funkcije i njihovi izvodi

U redu! Sada znate odakle potiču neuronske mreže, kakva im je arhitektura i kako se izračunavaju nihovi rezultati. Savladali ste i algoritam povratne propagacije. Ali šta biste tačno mogli da radite s njima?

MLP regresija

Prvo, višeslojni perceptroni (MLP) mogu se koristiti za zadatke izračunavanja regresije. Ako želite da predvidite jednu vrednost (npr. cenu kuće, ako je poznato više njenih osobina), potreban vam je samo jedan izlazni neuron: njegov rezultat biće predviđena vrednost. Za multivarijantnu regresiju (predviđenje više vrednosti istovremeno), potreban vam je po jedan izlazni neuron po izlaznoj dimenziji. Na primer, da biste pronašli centar određenog predmeta na slici, potrebno je da predvidite njegove 2D koordinate, pa vam zato trebaju dva izlazna neurona. Ako želite da oko tog predmeta postavite i pokretni okvir, potrebna su vam još dva broja: širina i visina premeta. Dakle, na kraju imate četiri izlazna neurona.

Uglavnom, kada gradite MLP za regresiju, najbolje je da u izlaznim neuronima ne koristite nikakvu aktivirajuću funkciju, da bi oni mogli da slobodno proizvode proizvoljne opsege vrednosti. Ako želite da garantujete da će rezultat uvek biti pozitivan, u izlaznom sloju možete upotrebiti aktivirajuću funkciju ReLU. Druga mogućnost je aktivirajuća funkcija *softplus*, koja je glatka varjanta funkcije ReLU: $\text{softplus}(z) = \log(1 + \exp(z))$. Njena vrednost je blizu 0 kada je z is negativno, a blizu z kada je z pozitivno. I najzad, ako želte da garantujete da će predviđanja biti uvek unutar datog opsega vrednosti, možete upotrebiti logističku fukciju ili hiperbolični tangens i da oznake primera zatim skalirate na odgovarajući opseg vrednosti: od 0 do 1 za logističku funkciju, odnosno od –1 do 1 za hiperbolični tangens.

Funkcija gubitka koju ćete koristiti tokom obuke obično je srednja vrednost kvadrata greške, ali ako u skupu za obuku imate veliki broj netipičnih primera, možda ćete radije koristiti srednju vrednost apsolutne greške. Druga mogućnost je da korisite Huberov gubitak, koja je kombinacija obe.

Huberov gubitak je funkcija koja je kvadratna kada je greška ispod određenog praga δ (tipična vrednost je 1) ali linearna kada je greška iznad δ. Linearni deo je čini manje osetljivom na netipične primere od srednjeg kvadrata greške, a kvadratni deo omogućava da konvergira brže i da bude preciznija od srednje apsolutne greške.

Tabela 10-1 sumira tipičnu arhitektura MLP za izračunavanje regresije.

Tabela 10-1 Tipična arhitektura MLP za izračunavanje regresije

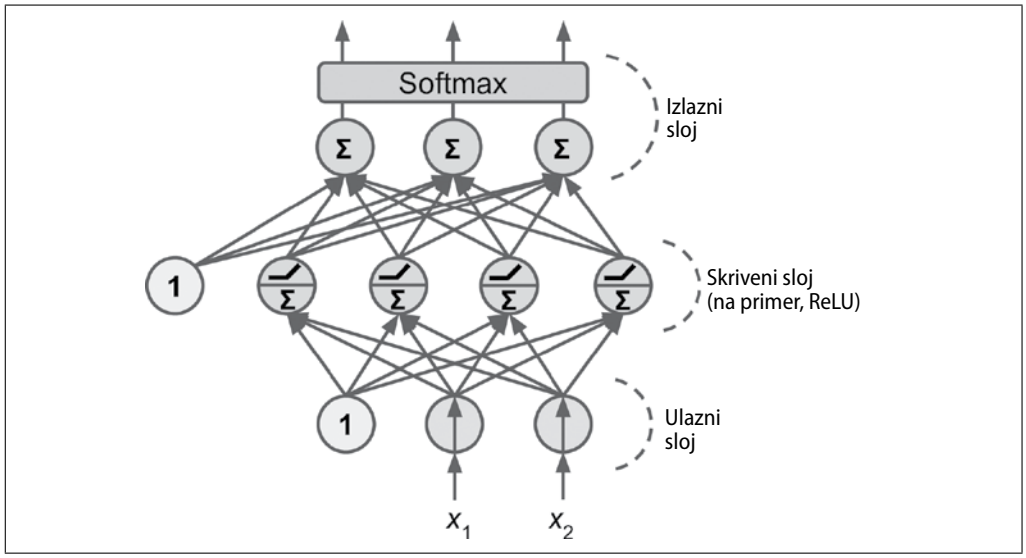
Hiperparametar	Tipična vrednost
# ulazni neuroni	Jedan po ulaznoj osobini (npr. 28 x 28 = 784 u skupu MNIST)
# skriveni slojevi	Zavisi od problema, ali uglavnom od 1 to 5
# neurona po skrivenom sloju	Zavisi od problema, ali uglavnom od 10 do 100
# izlazni neuroni	Jedan po dimenziji koja se predviđa
Aktivirajuća funkcija u skrivenim neuronima	ReLU (ili SELU, poglavlje 11)
Aktivirajuća funkcija u izlaznim neuronima	Nema, ili ReLU/softplus (ako su izlazne vrednosti pozitivne) Ii logistička/tanh (ako su izlazne vrednosti u datim opsezima)
Funkcija gubitka	MSE ili MAE/Huber (ako ima netipičnih primera)

MLP klasifikacija

MLP-ovi se mogu koristiti i za zadatke klasifikacije. Za problem binarne klasifikacije, potreban vam je samo jedan izlazni neuron, koji koristi logističku aktivirajuću funkciju: rezultat će biti broj u opsegu od 0 do 1, koji možete tumačiti kao procenjenu verovatnoću pozitivne klase. Procenjena verovatnoća za negativnu klasu jednaka 1 minus taj broj.

MLP-ovi mogu da s lakoćom obavljaju zadatke klasifikacije na više oznaka (poglavlje 3). Na primer, možete imati sistem za klasifikaciju poruka e-pošte koji predviđa da li je svaka primljena poruka prihvatljiva ili neželjena, a u isto vreme predviđa i da li je hitna ili ne. U tom slučaju, trebala bi vam dva izlazna neurona, koji oba koriste logisitčku aktivirajuću funkciju: prvi bi vraćao verovatnoću da je poruka neželjena, a drugi bi vraćao verovatnoću da je hitna. U opštem slučaju, dodelili biste po jedan izlazni neuron za svaku pozitivnu klasu. Imajte u vidu da zbir njihovih rezultata neće obavezno biti 1. To omogućava modelu da vraća proizvoljnu kombinaciju oznaka: možete imati poruku koja je prihvatljiva ali nije hitna, prihvatljiva i hitna, neželjena i nije hitna, a možda čak i neku koja je neželjena ali hitna (mada bi to verovatno bila greška).

Ako svaki primer može pripadati samo jednoj od tri ili više mogućih klasa (npr. klase od 0 do 9 za klasifikaciju slika cifara), onda će vam trebati po jedan izlazni neuron po klasi i trebalo bi da u celom izlaznom sloju koristite aktivirajuću funkciju softmax (slika 10-9). Funkcija softmax (uvedena je u poglavlju 4) obezbediće da sve procenjene verovatnoće budu u ospegu od 0 do 1 i da njihov zbir bude 1 (što je obavezno ako su klase međusobno isključive). To se zove višeklasna ili multiprogramska klasifikacija.



Slika 10-9. Savremeni MLP (sa funkcijama ReLU i softmax) za klasifikaciju

U vezi sa fukcijom gubitka, prošto predviđamo distribucije verovatnoća, gubitak zbog unakrsne entropije (koji se zove i logistički gubitak, poglavlje 4) uglavnom je dobar izbor.

Tabela 10-2 sumira tipičnu arhitekturu klasifikacijskog MLP-a.

Tabela 10-2 Tipična arhitktura klasifikacijskog MLP-a

Hiperparameter	Binarna klasifikacija	Višeznačna binarna klasifikacija	Višeklasna klasifikacija
Ulazni i skriveni slojevi	Isto kao regresija	Isto kao regresija	Isto kao regresija
# izlazni neuroni	1	1 po oznaci	1 po klasi
Aktivirajuća funkcija u izlaznom sloju	Logistička	Logistička	Softmax
Funkcija gubitka	Unakrsna entropija	Unakrsna entropija	Unakrsna entropija

Pre nego što nastavimo, preporučujem vam da uradite vežbu 1 na kraju ovog poglavlja. Igraćete se s raznim vrstama arhitektura neuronskih mreža i vizuelizovati njihove rezultate pomoću alatke *TensorFlow Playground*. To će biti veoma korisno za bolje razumevanje MLP-ova, uključujući i efekte svih hiperparametara (broj slojeva i neurona, aktivirajuće funkcije i drugo).

Pošto ste dosad savldali sve koncepte, vreme je da počnete implementiranje MLP-ova pomoću API-ja Keras!