Neuronske mreže

Višeslojni perceptroni

Aleksandra Zdravković Kosta Ljujić Mihajlo Srbakoski

Uvod

Neuronske mreže se mogu koristiti kako u regresionim, tako i u klasifikacionim problemima. U daljem radu mi ćemo se baviti problemom klasifikacije. Imajući u vidu da podaci nisu uvek linearno separabilni, klasifikatori sa kojima smo se susretali do sada neće biti od velike pomoći. Moć neuronskih mreža se, između ostalog, ogleda u njihovoj sposobnosti razdvajanja podataka nelinearno. Neuronske mreže su takođe superiornije u odnosu na druge modele u situacijama kada imamo veliku količinu podataka sirove reprezentacije.

Neuronske mreže predstavljaju model računanja inspirisan radom bioloških neurona u mozgu koji paralelno obrađuje podatke i koji se ne programira eksplicitno, već se trenira nad primerima ulaznih i izlaznih podataka koje bi trebalo da generiše. Neuronske mreže sadrže veći broj „jedinica" koje odgovaraju pojedinačnim neuronima, i obično su organizovane u slojeve koji predstavljaju faze kroz koje se ulazni podaci transformišu do izlaznih. Transformacija podataka se može odvijati u jednom prolazu kroz slojeve (eng. *feedforward*) ili mogu postojati ciklusi (eng. *recurrent*). U ovom radu bavimo se prvim.

Potpuno povezana neuronska mreža (u nastavku samo neuronska mreža) se sastoji od osnovnih računskih jedinica koje se nazivaju neuroni i predstavljaju jednostavne parametrizovane funkcije. Svaka jedinica računa linearnu kombinaciju svojih argumenata i nad njom računa neku nelinearnu transformaciju, takozvanu aktivacionu funkciju (eng. *activation function*). Ove jedinice su organizovane u slojeve, tako da jedinice jednog sloja primaju kao svoje argumente izlaze svih jedinica prethodnog sloja i sve jedinice prosleđuju svoje izlaze jedinicama narednog sloja. Zato se zovu potpuno povezanim. Svi slojevi čije jedinice prosleđuju svoje izlaze drugim jedinicama se nazivaju skrivenim slojevima. Ulazi jedinica prvog sloja se nazivaju ulazima mreže. Izlazi jedinica poslednjeg sloja se nazivaju izlazima mreže. Ukoliko neuronska mreža ima više od jednog skrivenog sloja, naziva se dubokom neuronskom mrežom (eng. *deep neural network*). Duboku neuronsku mrežu koja transformaciju podataka odvija u jednom prolazu kroz slojeve zovemo višeslojni perceptron (eng. *deep feedforward network*, ili *multylayer perceptron*).

Vrednosti neurona skrivenih slojeva mreže se mogu smatrati novim atributima tih objekata, nad kojima ostatak neuronske mreže uči aproksimaciju ciljne funkcije. Drugim rečima, neuronska mreža konstruiše nove atribute u svojim skrivenim slojevima. Svaki sloj je u stanju da nadograđuje nad prethodnim i tako gradi složenije i složenije atribute. Ovo svojstvo je posebno uočljivo kod konvolutivnih neuronskih mreža i smatra se da je ova mogućnost konstrukcije novih atributa jedan od glavnih razloga za uspešnost dubokih neuronskih mreža.

Dakle, za ulaz x i poznat izlaz y cilj je da odredimo ocenu - $\hat{y}$.

Arhitektura mreže

Neuron

Neuron je osnovna jedinica građe neuronske mreže. Neuron prihvata n-dimenzioni ulaz i transformiše ga u 1-dimezioni izlaz. Ono što razlikuje izlaze različitih neurona su njihovi parametri koje zovemo *težina*. Funkcija kojoj neuron prosleđuje podatke skalirane težinama naziva se *aktivaciona funkcija*.

Neuron, dakle, prihvata n-dimenzioni vektor podataka x i formira skalarni izlaz a. Neuron je dodatno povezan sa n-dimenzionim vektorom težina w i intercept-om b.

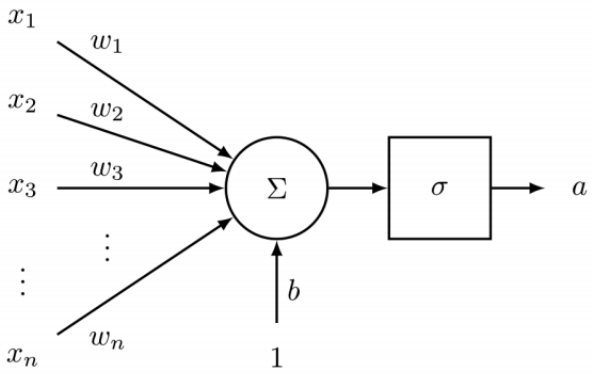

Figure 1: Sigmoidni neuron koji ulazni vektor x skaliran težinama w i sabran sa intercept-om prosleduje aktivacionoj funkciji.

Slojevi (eng. ***layers***)

Višeslojni perceptron se sastoji od najmanje tri sloja čvorova: ulazni sloj, skriveni sloj i izlazni sloj (eng. *an input layer, a hidden layer, an output layer*). Osim ulaznih čvorova, svaki čvor je neuron koji koristi nelinearnu funkciju aktivacije (eng. *activation function*). Za treniranje višeslojnog perceptrona koriste se tehnike vođenog učenja pod nazivom propagacija unazad (enf. *backpropagation*).

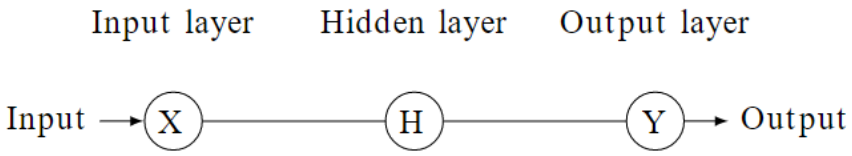

Figure 2: Ilustracija slojeva u neuronskoj mreži

Ulazni sloj neuronske mreže je dizajn matrica, tj. matrica $X = [1 \quad x]$. Izlazni sloj je vektor procenjenih verovatnoća $\hat{y}$. Neuronska mreža sadrži i skriveni sloj, za koji se može reći da je srednja matrica dizajna između ulaza i izlaza. Duboko učenje (eng. *deep learning*) je samo neuronska mreža sa više skrivenih slojeva.

Ako imamo d ulaznih promenljivih, tada će postojati $d+1$ ulazni čvor - po jedan za svaki prediktor i jedan za intercept. U opštem slučaju postoji k izlaznih čvorova, gde je k broj mogućih klasa u koje delimo naše podatke. U skrivenom sloju postoji $h+1$ čvor, od kojih je jedan intercept. Svaki od njih predstavlja linearnu kombinacihu ulaznih podataka nad kojom je primenjena aktivaciona funkcija.

Izlazni sloj predstavlja procenu verovatnoće da objekti pripadaju svakoj od predodređenih klasa. Ulazni sloj sadrži zavisne promenljive i intercept, kao i u već poznatim regresionim/klasifikacionim modelima. Između

ulaznog i izlaznog sloja nalazi se jedan ili više skrivenih slojeva, koji predstavlja transformaciju ulaznog prostora u h-dimenzioni prostor, gde je h broj koji smo odabrali. Nad transformisanim podacima se dalje vrši logistička regresija koja procenjuje klase.

Algoritam:

---

1. Generiše se h različitih linearnih kombinacija prediktora x.
2. Primenjuje se aktivaciona funkcija koja, za svaku obzervaciju, postavlja skriveni čvor na "on" ili "off."
3. Primenjuje se logistička regresija nad h transformisanih prediktora i intercept-om.

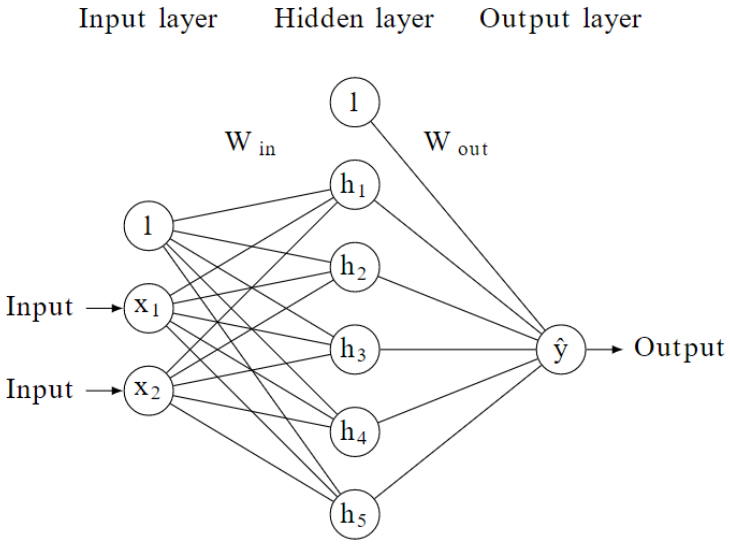

Figure 3: Višeslojni perceptron sa 6 skrivenih cvorova

Aktivaciona funkcija

Kao što je već i pomenuto, aktivaciona funkcija je nelinearna funkcija koje se primenjuje nad linearnom kombinacijom ulaza skaliranog težinama. Aktivaciona funkcija često uzima oblik sigmoidne funkcije:

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

Propagacija unapred (eng. *forward propagation*)

Počevši od ulaza, podatke propagiramo unapred kroz mrežu.

Prvo, računamo linearnu kombinaciju prediktora, koristeći tzv. matricu težina $W_{in} \in \mathbb{R}^{(d+1)\times h}$:

$$z_1 = XW_{in} = [1 \quad x]\, W_{in}$$

Dalje primenjujemo funkciju aktivacije nad linearnim kombinacijama kako bismo dobili čvorove u skrivenom sloju. Skriveni sloj H može se posmatrati kao dizajn matrica koja sadrži izlaz logističke regresije koja klasifikuje da li je svaki od čvorova „aktiviran“ ili ne. Interecept je fiksiran jedinični vektor.

$$h = \sigma(z_1)$$
$$H = [1 \quad h] = [1 \quad \sigma(z_1)] = [1 \quad \sigma(XW_{in})]$$

Za izlazni sloj izračunavamo linearnu kombinaciju skrivenih promenljivih, koristeći drugu matricu težina:

$$z_2 = HW_{out} = [1 \quad h]\, W_{out}$$

Da bismo dobili krajnji izlaz, primenjujemo aktivacionu funkciju još jednom:

$$\hat{y} = \sigma(z_2),$$

gde $\hat{y}$ predstavlja verovatnoće pripadnosti klasama.

Objedinujući rezultate dobijamo:

$$\hat{y} = f\Big(HW_{out}\Big) = f\Big(\, [1 \quad \sigma(XW_{in})]\, W_{out}\Big)$$

Ilustrujmo kako bi u R-u izgledala funkcija koja vrši propagaciju unapred na goreopisan način:

```
sigmoid <- function(k) 1/(1 + exp(-k))

feedforward <- function(k, v1, v2) {
  z1 <- cbind(1, k) %*% v1
  h <- sigmoid(z1)
  z2 <- cbind(1, h) %*% v2
  list(output = sigmoid(z2), h = h)
}
```

Matrica težina u prvoj iteraciji može biti postavljena proizvoljno. U narednim iteracijama je potrebno pametno je ažurirati. Više o tome u delu o treniranju mreže.

Propagacija unazad (eng. *backpropagation*)

Pozabavimo se sada odabirom koeficijenata matrica težina W_{in} i W_{out}. Vrednosti matrica težina nalazimo minimiziranjem funkcije gubitaka (negativna vrednost funkcije maksimalne verodostojnosti):

$$\min_W l(W)$$

gde je funkcija gubitaka za K klasa klasifikacije

$$l(W) = l = -\sum_{i=1}^{n}\sum_{k=1}^{K}\Big(y_i \log \widehat{y_i} + (1-y_i)\log(1-\widehat{y_i})\Big)$$

Da bismo minimizovali funkciju $l = l(W)$ koristićemo algoritam opadajućeg gradijenta. Opadajući gradijent je optimizacioni algoritam koji pronalazi lokalni minimum funkcije tako što izvršava više koraka proporcionalnih negativnoj vrednosti gradijenta odgovarajuće funkcije. U metodi gradijentnog spusta za iterativno ažuriranje vrednosti matrice težina. Zahvaljujući parcijalnim izvodima algoritam je svakom iteracijom bliži globalnom minimumu.

Algoritam počinje slučajno odabranom vrednošću W_0 iz čega se dobija niz elemenata $W_1, W_2, W_3, ...$ tako da važi:

$$W_{t+1} = W_t - \gamma \nabla l(W_t),$$

gde je W_t vrednost matrice težina u iteraciji t, ∇f gradijent funkcije f u odnosu na W, a γ „brzina učenja".

Napomenimo da je odabir brzine učenja γ jako bitan kako bi algoritam konvergirao ka stvarnom minimumu funkcije. Ukoliko je ova vrednost prevelika moguće je prekoračiti minimum (divergirati, tj. udaljavati od minimuma). Kod premale vrednosti, postojaće više iteracija gradijentnog spusta, a samim tim dužina celog procesa treniranja će se povećati.

Shema po kojoj se vrši propagacija unapred u dvoslojnom perceptronu je:

$$X \to XW_1 \to H_1 = \sigma(XW_1) \to H_1W_2 \to H_2 = \sigma(H_1W_2) \to H_2W_3 \overset{\text{softmax}}{\to} \widehat{y}$$

Kako bismo vršili propagaciju unazad neophodno je naći ocene za prikazane matrice W_3, W_2, W_1. Učinimo to sada. Preglednosti radi prikažimo matrice koje će biti korišćene u daljem izvođenju zajedno sa svojim dimenzijama i opisima:

X	$\mathbb{R}^{n\times d}$	matrica ulaznih podataka, gde je n broj observacija, a d broj prediktora
W_1	$\mathbb{R}^{d\times(h_1+1)}$	matrica težina u ulaznom sloju, gde je n broj observacija, a h_1+1 broj cvorova ulaznog sloja
H_1	$\mathbb{R}^{n\times(h_1+1)}$	$H_1 = \sigma(XW_1)$
W_2	$\mathbb{R}^{(h_1+1)\times(h_2+1)}$	matrica težina u skrivenom sloju, gde je h_2+1 broj cvorova skrivenog sloja
H_2	$\mathbb{R}^{n\times(h_2+1)}$	$H_2 = \sigma(H_1W_2)$
W_3	$\mathbb{R}^{(h_2+1)\times 10}$	matrica težina u izlaznom sloju
$\widehat{y}$	$\mathbb{R}^{n\times 10}$	rezultat klasifikacije

Nađimo izvode funkcije gubitaka l u odnosu na matrice težina W_i.

$$l(\widehat{y}) = -\sum_{i=1}^{n}\Big(I\{y_{i1}=0\}\log \widehat{y_{i1}} + \cdots + I\{y_{i10}=9\}log\widehat{y_{i10}}\Big)$$

$$\frac{\partial l}{\partial \widehat{y_{i1}}} = \frac{I\{y_{i1}=0\}}{\widehat{y_{i1}}} - \frac{I\{y_{i2}=1\}}{1-\widehat{y_{i1}}-\cdots-\widehat{y_{i10}}} - ... - \frac{I\{y_{i10}=9\}}{1-\widehat{y_{i1}}-\cdots-\widehat{y_{i9}}} = \frac{I\{y_{i1}=0\}}{\widehat{y_{i1}}} - \frac{I\{y_{i2}=1\}}{\widehat{y_{i2}}} - ... - \frac{I\{y_{i10}=9\}}{\widehat{y_{i10}}}$$

Napomena. @ označava operaciju matričnog množenja, dok $*$ označava pokoordinatno, tj. skalarno množenje.

Izvode funkcije gibitaka računamo u odnosu na matrice težina W_1, W_2 i W_3 i pomoću izračunatih vrednosti ažuriramo vrednosti matrica W_1, W_2 i W_3. Može se pokazati da važi:

$$\frac{\partial l}{\partial W_3} = H_2^T @ (\frac{\partial l}{\partial \widehat{y}} \cdot \nabla softmax(H_2 @ W_3))$$

$$\frac{\partial l}{\partial W_2} = H_1^T @ \left(\left(\left(\frac{\partial l}{\partial \hat{y}} \cdot \nabla softmax(H_2 @ W_3) \right) @ W_3^T \right) \cdot \sigma'(H_1 @ W_2) \right)$$

$$\frac{\partial l}{\partial W_1} = X^T @ \left(\left(\left(\left((\frac{\partial l}{\partial \hat{y}} \cdot \nabla softmax(H_2 @ W_3)) @ W_3^T \right) \cdot \sigma'(H_1 @ W_2) \right) @ W_2^T \right) \cdot \sigma'(X @ W_1) \right)$$

Uočimo da važi:

$$\sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = \sigma(x)(1-\sigma(x))$$

Kako je

$$\frac{\partial softmax}{\partial x_k} = \frac{e^{x_k}(\sum_{i=1}^n e^{x_i}) - e^{x_k} e^{x_k}}{(\sum_{i=1}^n e^{x_i})^2} = \frac{e^{x_k}(\sum_{i=1, i \neq k}^n e^{x_i})}{(\sum_{i=1}^n e^{x_i})^2}$$

imamo da je

$$\nabla softmax(x_1, \dots, x_{10}) = softmax(x_1, \dots, x_{10}) \cdot (1 - softmax(x_1, \dots, x_{10}))$$

Sada izvodi dobijaju sledeći oblik:

$$\frac{\partial l}{\partial W_3} = H_2^T @ (\hat{y} - y)$$

$$\frac{\partial l}{\partial W_2} = H_1^T @ \left(\left((\hat{y} - y) @ W_3^T \right) \cdot \left(\sigma(H_1 @ W_2) \cdot (1 - \sigma(H_1 @ W_2)) \right) \right)$$

$$\frac{\partial l}{\partial W_1} = X^T @ \left(\left(\left(\left((\hat{y} - y) @ W_3^T \right) \cdot \left(\sigma(H_1 @ W_2) \cdot (1 - \sigma(H_1 @ W_2)) \right) \right) @ W_2^T \right) \cdot \left(\sigma(H_1 @ W_2) \cdot (1 - \sigma(H_1 @ W_2)) \right) \right)$$

Ilustrujmo kako bi u R-u izgledala funkcija koja vrši propagaciju unazad na goreopisan način:

```
gradient <- function(x, y, y_hat, w1, w2, w3, h1, h2) {
  dmle <- y_hat - y
  dw3 <- t(h2) %*% dmle
  d <- dmle%*%t(w3)
  dw2 <- t(h1)%*%(d*h2*(1-h2))
  dw1 <- t(cbind(1,x))%*%(((d*h2*(1-h2))%*%t(w2))*h1*(1-h1))

  list(dw1 = dw1, dw2 = dw2, dw3 = dw3)
}
```

Izlazne jedinice i softmax funkcija

Neuronske mreže se mogu koristiti kako za regresiju funkcija sa vrednostima iz $\mathbb{R}^n$, tako i za klasifikaciju.

U slučaju klasifikacije, što je naša tema, na linearne kombinacije jedinica poslednjeg sloja primenjuje se tzv. funkcija mekog maksimuma (eng. *softmax function*). Važi $softmax : \mathbb{R}^m \to \mathbb{R}^m$, tj. preslikava vektor dimenzija m u vektor dimenzija m:

$$softmax(x) = \left(\frac{e^{x_1}}{\sum_{i=1}^{m} e^{x_i}}, \dots, \frac{e^{x_m}}{\sum_{i=1}^{m} e^{x_i}} \right)$$

Primetimo da se vrednosti novog vektora sumiraju na 1, te ih stoga možemo koristiti kao raspodelu verovatnoće. Takođe, primetimo da ova funkcija dodatno naglašava razlike između koordinatama polaznog vektora. Najveća pozitivna vrednost će biti transformisana u novu vrednost koja još više odskače od drugih. Za vrednost aproksimacije se uzima kategorija koja odgovara izlazu sa najvišom vrednošću.

Treniranje mreže

Kada se kaže *treniranje mreže* zapravo se misli na inicijalizaciju težina, propagaciju prediktora unapred kako bi se dobila ocena izlaza, zatim propagaciju greške unazad kako bi se ažurirale težine radi tačnijeg izlaza. Dalje se iterativno propagira unapred i unazad fiksiran broj puta (iteracija).

Podaci

MNIST baza je baza slika ručno napisanih cifara. Sastoji se od 2 skupa podataka - trening podataka (60000 slika) i test podataka (10000 slika).

Svaka slika ima 28 piksela za širinu i 28 piksela za dužinu, dakle ukupno 784 piksela. Svakom pikselu dodeljena je vrednost koja govori koliko je on svteao, tj. taman. Veći brojevi govore da je piksl tamniji. Ova vrednost piksela je ceo broj u intervalu $[0, 255]$.

Skup trening podataka ima 785 kolona. Prva kolona, nazvana „oznaka“, je cifra koju je korisnik nacrtao. Ostatak kolona sadrži vrednosti piksela pridružene slike. Svaka kolona piksela u skupu treninga ima ime poput pikselK, gde je K ceo broj iz intervala $[0, 783]$. Da bismo locirali ovaj piksel na slici, pretpostavimo da smo K razložili kao $K = 28i + j$, gde su i i j celi brojevi u intervalu $[0, 27]$. Tada se pikselK nalazi u redu i u koloni j matrice 28×28. Na primer, piksel31 označava piksel koji se nalazi u četvrtoj koloni s leve strane, a drugi red od vrha.

Ono što klasifikaciju podataka iz baze MNIST čini problemom u kom je izbor neuronskih mreža superiorniji u odnosu na ostale metode su velika količina podataka i učenje na osnovu sirove reprezentacije podataka.

Implementacija u R-u

Učitavamo slike i brojeve koji su predstavljeni slikama. MNIST baza dolazi sa već odvojenim test i trening podacima.

```
# Učitavanje neophodnih biblioteka
library(doParallel)
library(readmnist)
library(imager)
library(foreach)

# Omogućavanje paralelizacije
no_cores <- detectCores()
cl <- makeCluster(no_cores)
registerDoParallel(cl)

# Učitavanje podataka iz baze MNIST
x_training <- Read.mnist("train-images.idx3-ubyte")
y_training <- Read.mnist("train-labels.idx1-ubyte")
x_test <- Read.mnist("t10k-images.idx3-ubyte")
y_test <- Read.mnist("t10k-labels.idx1-ubyte")
```

Učitane slike su sačuvane kao nizovi piksela, te je potrebno podatke prebaciti u oblik pogodan za dalji rad. Svaka slika je niz od 28×28 piksela. Prva slika je sačinjena od piksela u intervalu $[1 : 784]$, druga u intervalu $[1 + 1 * 784, 2 * 784]$, itd.

```
# Obrada podataka se vrši paralelno. Time se vreme potrebno za izvršavanje
# znatno smanjuje
image_training <- foreach(i = 1:60000) %dopar% {
    x_training$pic[(1+(i-1)*784):(i*784)]
}
# Transformacija liste u dataframe
data_training <- data.frame(matrix(unlist(image_training),
                                   nrow=length(image_training),
                                   byrow=TRUE))
# U dataframe ubacujemo i vrednost zavisne promenljive
data_training$y <- y_training$labels

# Isti postupak i za test skup
image_test <- foreach(i = 1:10000) %dopar% {
    x_test$pic[(1+(i-1)*784):(i*784)]
}
data_test <- data.frame(matrix(unlist(image_test),
                               nrow=length(image_test),
                               byrow=TRUE))
data_test$y <- y_test$labels
```

Definišimo aktivacionu (sigmoidnu) i softmaks funkciju.

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

$$softmax(x_1, \dots, x_{10}) = (\frac{e^{x_1}}{\sum_{i=1}^{n} e^{x_i}}, \dots, \frac{e^{x_{10}}}{\sum_{i=1}^{n} e^{x_i}})$$

```
sigmoid <- function(x) 1 / (1 + exp(-x))
softmax <- function(x) exp(x) / rowSums(exp(x))
```

Definišimo funkciju koja vrši propagaciju unapred, na način koji je već opisan.

```
feedforward <- function(x, w1, w2, w3) {
  z1 <- cbind(1, x) %*% w1
  h1 <- sigmoid(z1)
  z2 <- h1 %*% w2
  h2 <- sigmoid(z2)
  z3 <- h2 %*% w3
  list(probs = softmax(z3), h1 = h1, h2 = h2)
}
```

Na već opisan načim nalazimo gradijent matrica W_1, W_2, W_3:

```
gradient <- function(x, y, y_hat, w1, w2, w3, h1, h2) {
  dmle <- y_hat - y
  dw3 <- t(h2) %*% dmle
  d <- dmle%*%t(w3)
  dw2 <- t(h1)%*%(d*h2*(1-h2))
  dw1 <- t(cbind(1,x))%*%(((d*h2*(1-h2))%*%t(w2))*h1*(1-h1))
```

```
  list(dw1 = dw1, dw2 = dw2, dw3 = dw3)
}
```

Algoritam propagacije unazad podrazumeva računanje gradijenta funkcije gubitaka, nakon čega je potrebno ažurirati vrednosti matrica težina po formuli:

$$W_{t+1} = W_t - \gamma \nabla l(W_t),$$

```
backpropagate <- function(x, y, y_hat, w1, w2, w3, h1, h2, learn_rate = 0.0001) {
  grad <- gradient(x, y, y_hat, w1, w2, w3, h1, h2)
  w1 <- w1 - learn_rate * grad$dw1
  w2 <- w2 - learn_rate * grad$dw2
  w3 <- w3 - learn_rate * grad$dw3

  list(w1 = w1, w2 = w2, w3 = w3)
}
```

Implementirajmo sada paralelizovanu verziju funkcije koja vrši propagaciju unazad. Umesto računanja gradijenta odjednom, podatke ćemo podeliti na N delova, gde N predstavlja broj jezgara računara. Dalje ćemo paralelno računati gradijente za N delova, a ukupan gradijent računati kao zbir N dobijenih gradijenata. Ovako definisana funkcija smanjuje dužinu izvršavanja.

```
# Omogućavanje paralelnog izvršavanja
library("doFuture")
registerDoFuture()
plan(multisession)


parallel_backpropagate <- function(x, y, y_hat, w1, w2, w3, h1, h2, learn_rate = 0.0001, N = no_cores) {
  n <- ceiling(nrow(x)/N)
  grads <- foreach(i = 1:N)%dopar%{
    gradient(x[((i-1)*n+1):(i*n),],y[((i-1)*n+1):(i*n),],y_hat[((i-1)*n+1):(i*n),],w1,w2,w3,h1[((i-1)*n
  }

  dw1 <- 0
  dw2 <- 0
  dw3 <- 0
  for(grad in grads){
    dw1 <- dw1 + grad$dw1
    dw2 <- dw2 + grad$dw2
    dw3 <- dw3 + grad$dw3
  }
  w1 <- w1 - learn_rate * dw1
  w2 <- w2 - learn_rate * dw2
  w3 <- w3 - learn_rate * dw3

  list(w1 = w1, w2 = w2, w3 = w3)
}
```

Funkcija gubitaka

```
loss <- function(Y,Y_hat){
  -sum(Y*log(Y_hat))
}
```

Ovde vracamo predikcije

```
predict <- function(model, X){
  ff <- feedforward(X, model$w1, model$w2, model$w3)
  prediction <- apply(ff$probs, 1, function(x) {which.max(x)-1})
  prediction
}
```

Trazimo preciznost, u smislu koliko procentualno smo pogodili rezultata

```
accuracy <- function(y_pred, y) {
  mean(y==y_pred)*100
}
```

Ova funkcija nam sluzi da prtvori vektor vrednosti za y u matricu koja ce za svaku vrednost imati jedinicu na odgovarajucem mestu i ostalo 0.

```
VectorToMatrix <- function(y) {
  Y <- matrix(rep(0, 10*length(y)),nrow=length(y))
  for(i in 1:length(y)){
    Y[i,(y[i]+1)] <- 1
  }
  Y
}
```

```
train <- function(data, data_test , hidden1, hidden2, learn_rate = 0.0001, iterations = 100, type = "cl
  #Ovde ucitavano trening i test podatke
  y <- VectorToMatrix(data$y)
  x <- as.matrix(data[,-785])
  y_test <- VectorToMatrix(data_test$y)
  x_test <- as.matrix(data_test[,-785])

  start_time <- Sys.time()

  #Ovde cemo cuvati preciznost i gubitak
  accuracy_training_plot <- c()
  accuracy_test_plot <- c()
  loss_plot <- c()

  d <- ncol(x)
  if(type == "uniform"){
    w1 <- cbind(0,matrix(runif((d+1) * hidden1), d+1, hidden1))
    w2 <- cbind(0, matrix(runif((hidden1+1)*hidden2), hidden1+1, hidden2))
    w3 <- matrix(runif((hidden2 + 1)*10), hidden2+1, 10)
  }else {
    w1 <- cbind(0,matrix(rnorm((d+1) * hidden1), d+1, hidden1))
    w2 <- cbind(0, matrix(rnorm((hidden1+1)*hidden2), hidden1+1, hidden2))
    w3 <- matrix(rnorm((hidden2 + 1)*10), hidden2+1, 10)
```

```
  }

  for (i in 1:iterations) {
    ff <- feedforward(x, w1, w2, w3)

    if(type == "parallel"){
      bp <- parallel_backpropagate(x = x, y = y, y_hat = ff$probs, w1 = w1, w2 = w2, w3 = w3, h1 = ff$h
    }else if(dynamic == TRUE){
      #ovde moze da se menja kako se dinamicki menja lear_rate
      bp <- backpropagate(x = x, y = y, y_hat = ff$probs, w1 = w1, w2 = w2, w3 = w3, h1 = ff$h1, h2 = f
    }else{
    bp <- backpropagate(x = x, y = y, y_hat = ff$probs, w1 = w1, w2 = w2, w3 = w3, h1 = ff$h1, h2 = ff$
    }

    w1 <- bp$w1
    w2 <- bp$w2
    w3 <- bp$w3

    if(i%%5 == 0) {
      l <- loss(y, ff$probs)
      y_test_pred <- predict(result, x_test)
      y_train_pred <- predict(result, x)
      loss_plot <- c(loss_plot,l)
      accuracy_test_plot <- c(accuracy_test_plot,accuracy(y_test_pred,data_test$y))
      accuracy_training_plot <- c(accuracy_training_plot,accuracy(y_train_pred,data$y))
    }

    #OBRISI OVO
    if(i%%10 == 0){
      print(paste("Iteration:",i))
      print(paste("Loss:",loss(y, ff$probs)))
    }
    result <- list(output = ff$probs, w1 = w1, w2 = w2, w3 = w3, accuracy_training_plot = accuracy_trai
  }

  end_time <- Sys.time()
  if(time == TRUE)
    print(round(end_time-start_time,2))

  result
}
```

Treniramo jednu neuronsku mrezu, referentne vrednosti su krece od oko 2m spusta se na oko 100k za 60k slika, i krece od 100k i spusta se na 20k za 10k slika

Bibliografija