VJEŽBA 7: RJEŠAVANJE KLASIFIKACIJSKIH I REGRESIJSKIH PROBLEMA POMOĆU UMJETNIH NEURONSKIH MREŽA

I. Cilj vježbe:

Upoznati se s umjetnim neuronskim mrežama te načinom njihovog projektiranja za probleme klasifikacije i regresije.

II. Opis vježbe:

II.1. Nadgledano učenje

U ovoj vježbi razmatra se problem nadgledanog učenja gdje je cilj odrediti nepoznatu funkcionalnu ovisnost između *m* ulaznih veličina $X = [x_1, x_2, \dots, x_m]$ i izlazne veličine *y* na temelju podatkovnih primjera. Kada je izlazna veličina *y* diskretna veličina, radi se o problemu klasifikacije (detekcija rukom pisanih brojeva, odvajanje neželjene pošte (spam), kategorizacija članaka i sl.). U slučaju kontinuirane izlazne veličine *y*, problem se naziva regresijski problem (npr. predikcija temperature, predikcija cijene dionica/artikala, procjena potrošnje i sl.).
Podatkovni primjeri su parovi koji se sastoje od vektora vrijednosti ulaznih veličina i vrijednosti izlazne veličine, stoga se *i*–ti podatkovni primjer ili uzorak može prikazati kao uređeni par $(\mathbf{x}^{(i)}, y^{(i)})$. Vektor ulaznih veličina zapisuje u obliku:

$$\mathbf{x}^{(i)} = \left[x_1^{(i)}, x_2^{(i)}, \dots, x_m^{(i)}\right]^T. \tag{7-1}$$

Skup koji se sastoji od *n* raspoloživih mjernih podataka $\left(\mathbf{x}^{(1)}, y^{(1)}\right), \left(\mathbf{x}^{(2)}, y^{(2)}\right), \dots, (\mathbf{x}^{(n)}, y^{(n)})$ može se zapisati u matričnom obliku:

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_m^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_m^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(n)} & x_2^{(n)} & \dots & x_m^{(n)} \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}. \tag{7-2}$$

Kod regresijskih problema vrijedi $y^{(i)} \in \mathbb{R}$. Kod klasifikacijskih problema ako izlazna veličina ima samo dvije moguće vrijednosti (klase), npr. $y^{(i)} \in \{0,1\}$, tada se problem naziva binarna klasifikacija. U slučaju kada izlazna veličina može poprimit više od dvije vrijednosti , tada se problem naziva višeklasna klasifikacija. U slučaju višeklasne klasifikacije najčešće se koristi 1-od-*C* označavanje gdje je *C* broj klasa. U tom slučaju vrijednost uzorka izlazne veličine $y^{(i)}$ poprima jedno od *C* mogućih vrijednosti, ovisno kojoj klasi uzorak pripada:

$$y^{(i)} \in \left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \dots, \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \right\}. \tag{7-3}$$

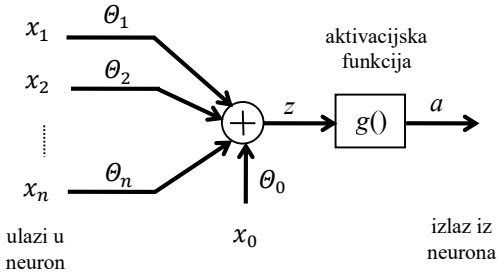
II.2. Umjetne neuronske mreže

Umjetne neuronske mreže (engl. *Artificial Neural Networks* - ANN) predstavljaju jedan od najčešće korištenih modela strojnog učenja za rješavanje (složenih) problema nadgledanog učenja. Ovakvi modeli su sastavljeni od osnovnih gradbenih jedinica koji se nazivaju neuroni. Unaprijedne neuronske mreže (engl. *Feedforward Neural Networks* - FNN) su mreže u kojima se propagacija signala odvija samo u jednom smjeru, od ulaznih veličina prema izlaznoj veličina što znači da mreža ne sadrži povratne veze. U ovoj vježbi koristi se unaprijedna mreža sastavljena od neurona s nelinearnom aktivacijskom funkcijom. Na slici je 7.1 prikazan je umjetni neuron s nelinearnom aktivacijskom funkcijom. S θ_i označeni su parametri neurona koji se često nazivaju i težine. Kao aktivacijska funkcija često se koristi sigmoidna funkcija:

$$g(z) = \frac{1}{1+e^{-z}} \tag{7-4}$$

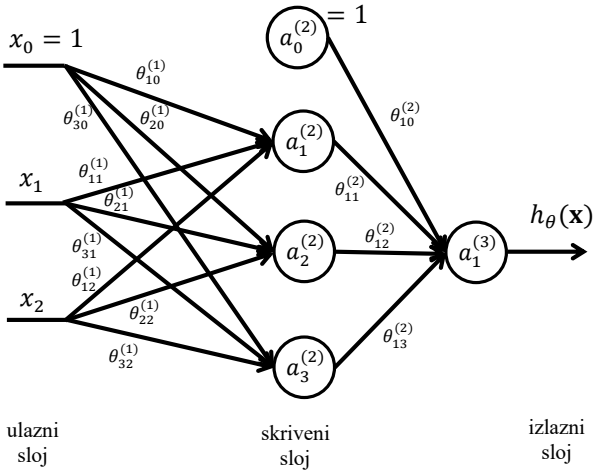
Neuronske mreže mogu imati nekoliko slojeva. Višeslojna neuronska mreža s neuronima s nelinearnom aktivacijskom funkcijom poput funkcije (7-4) često se naziva i MultiLayer Perceptron (MLP). U slučaju MLP mreže za binarnu klasifikaciju, izlazni sloj mreže ima jedan neuron sa sigmoidnom aktivacijskom funkcijom, dok se u slučaju višeklasne

klasifikacije (vidi (7-3)) izlazni sloj sastoji od *C* neurona sa sigmoidnim aktivacijskim funkcijama. Kod rješavanja regresijskih problema pogodno je u izlazni sloj mreže postaviti jedan neuron s linearnom aktivacijskom funkcijom. Na slici 7.2 prikazana je unaprijedna neuronska mreža s jednim ulaznim slojem (dvije ulazne veličine + bias x_0), jednim skrivenim slojem s tri neurona (+ bias $a_0^{(2)}$) te jednim izlaznim slojem s jednim neuronom.



$$a = g(z)$$
$$z = \Theta_0 x_0 + \Theta_1 x_1 + \cdots + \Theta_n x_n$$
$$z = \boldsymbol{\Theta}^T \mathbf{x}$$

Sl. 7.1 Blok prikaz umjetnog neurona.



$$h_\theta(\mathbf{x}) = a_1^{(3)} = g(z_1^{(3)})$$
$$z_1^{(3)} = \boldsymbol{\theta}^{(2)} \boldsymbol{a}^{(2)}$$
$$\boldsymbol{a}^{(2)} = g(\mathbf{z}^{(2)})$$
$$\mathbf{z}^{(2)} = \boldsymbol{\theta}^{(1)} \boldsymbol{a}^{(1)}$$
$$\boldsymbol{a}^{(1)} = \mathbf{x}$$

$$\boldsymbol{\theta}^{(1)} = \begin{bmatrix} \theta_{10}^{(1)} & \theta_{11}^{(1)} & \theta_{12}^{(1)} \\ \theta_{20}^{(1)} & \theta_{21}^{(1)} & \theta_{22}^{(1)} \\ \theta_{30}^{(1)} & \theta_{31}^{(1)} & \theta_{32}^{(1)} \end{bmatrix}$$

$$\boldsymbol{\theta}^{(2)} = \begin{bmatrix} \theta_{10}^{(2)} & \theta_{11}^{(2)} & \theta_{12}^{(2)} & \theta_{13}^{(2)} \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \quad \mathbf{z}^{(2)} = \begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \end{bmatrix} \quad \boldsymbol{a}^{(2)} = \begin{bmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$$

Sl. 7.2 Višeslojna neuronska mreža.

Neuronske mreže već i s manjim brojem neurona mogu opisati različite nelinearne odnose između ulaznih veličina i izlazne veličine. Međutim, neuronske mreže imaju i znatno više parametara (težina) koje je potrebno procijeniti nego što je slučaj kod jednostavnijih modela poput linearnog regresijskog modela ili logističke regresije. Npr., mreža prikazana na 6.2 ima ukupno 13 parametara koji se mogu predstaviti pomoću dvije matrice: matrica 3x3 koja predstavlja vezu između ulaznog i skrivenog sloja, te matrice 4x1 koja predstavlja vezu između skrivenog i izlaznog sloja.

Pod pojmom učenje neuronske mreže podrazumijeva se strukturiranje neuronske mreže (odabir broja slojeva, odabir broja neurona u pojedinom sloju, odabir tipa aktivacijske funkcije neurona), a zatim i procjena nepoznatih parametara neuronske mreže, tj. težina mreže. Uz definiranu strukturu neuronske mreže, procjena parametra mreže provodi se minimizacijom odgovarajuće kriterijske funkcije. Za slučaj klasifikacije, često korištena kriterijska funkcija sadrži i dio za regularizaciju (često se naziva i *weight decay*):

$$J(\boldsymbol{\theta}) = \frac{1}{n}\left[\sum_{i=1}^{n}\sum_{k=1}^{C} y_k^{(i)} \ln h_\theta\left(\mathbf{x}^{(i)}\right)_k + \left(1 - y_k^{(i)}\right) \ln\left(1 - h_\theta\left(\mathbf{x}^{(i)}\right)\right)\right] + \frac{\lambda}{2n}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\theta_{ij}^{(l)}\right)^2, \qquad (7\text{-}5)$$

gdje je *C* broj izlaznih neurona u mreži, λ je regularizacijski koeficijent, L je broj slojeva u mreži, a s_l je broj neurona u sloju *l*. Prilikom rješavanja regresijskih problema koristi se kriterijska funkcija:

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\left[\sum_{i=1}^{n}\left(h_\theta\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2 + \lambda \sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\theta_{ij}^{(l)}\right)^2\right]. \qquad (7\text{-}6)$$

Optimizacija kriterijskih funkcija (6-5) odnosno (6-6) provodi se iterativnim numeričkim postupcima koji se zasnivaju na gradijentu kriterijske funkcije. Algoritam s povratnim rasprostiranjem pogreške (engl. *backpropagation algorithm*) je efikasan algoritam za računanje gradijenta kriterijske funkcije. Prilikom učenja mreže važno je postaviti početne

vrijednosti parametara mreže na „male“ slučajne vrijednosti te nekoliko puta ponoviti učenje mreže jer kriterijska funkcija sadrži velik broj lokalnih minimuma.
Sa stajališta izgradnje predikcijskog modela važno je da mreža generira „točne“ odgovore i za vrijednosti ulaznih veličina koje se nisu koristile prilikom učenja mreže. Ova karakteristika neuronske mreže naziva se poopćavanje (engl. *generalization*). Loše poopćavanje može biti posljedica pretjeranog učenja (engl. *overfitting*). Pretjerano učenje se očituje u tome što mreža s prevelikom fleksibilnošću može opisati različite primjese (šum, stršeće vrijednosti i sl.) iz podataka za učenje, a koje nisu karakteristični za cijelu populaciju podataka. Ovakva mreža pokazuje izvrsne rezultate na podacima na temelju kojih su određeni njeni parametri, međutim, na drugim podacima iz iste populacije ova mreža može znatno griješiti. Jedan od načina kako spriječiti pretjerano učenje je uvođenje postupka među-vrjednovanja (engl. *cross-validation*) kojim se primjerice može odabrati broj neurona u skrivenom sloju ili koeficijent regularizacije λ. Osim toga, prikaz iznosa kriterijske funkcije $J(\boldsymbol{\theta})$ na podacima za učenje i validaciju u svakoj iteraciji optimizacijskog postupka može dati uvid u kvalitetu izgrađenog modela.

Generalni postupak učenja MLP neuronske mreže na određenom problemu u *scikit learn* okruženju može se sažeti u sljedeće korake:

1. Učitati raspoložive podatke. Dio podataka koristi se kao skup podataka za učenje dok se preostali podaci koriste za testiranje izgrađene mreže. Provesti predobradbu podataka (standardizacija, skaliranje, centriranje i sl.).
2. Strukturirati neuronsku mreže (koristiti klasu *MLPClassifier*): odabir broja slojeva, odabir broja neurona u pojedinom sloju, odabir tipa aktivacijske funkcije neurona te eventualno podesiti neke dodatne parametre vezane za postupak učenja neuronske mreže poput željenog numeričkog postupka, dodatni parametre optimizacije, poput koeficijenta regularizacije, broj iteracija numeričkog postupka (tzv. epohe) i sl.
3. Na temelju skupa za učenje odrediti parametre (težine) mreže odgovarajućim numeričkim postupkom.
4. Testiranje mreže kako bi se pokazale predikcijska svojstva mreže (npr. srednja kvadratna pogreška na testnim podacima, matrica zabune i sl.).

III. Priprema za vježbu:

Ponovite gradivo vezano za neuronske mreže.

IV. Rad na vježbi:

1. Klonirajte vaš repozitorij `rusu_lv_2019_20` na računalo pomoću gitbash. Zatim povucite moguće promjene iz izvornog repozitorija pomoću naredbi:

```
git remote add upstream https://gitlab.com/rgrbic/rusu_lv_2019_2020
git fetch upstream
git merge upstream/master
```

2. Riješite dane zadatke, pri čemu Python skripte trebaju imati naziv `zad_x.py` (gdje je `x` broj zadatka) i trebaju biti pohranjene u direktorij `rusu_lv_2019_20/LV7/solutions/`. Svaki zadatak rješavajte u zasebnoj *git* grani koju spojite s glavnom granom kada riješite pojedini zadatak. Pohranite skripte u lokalnu *git* bazu kao i u `rusu_lv_2019_20` repozitorij na vašem korisničkom računu. Svaki puta kada načinite promjene koje se spremaju u *git* sustav napišite i odgovarajuću poruku prilikom izvršavanja `commit` naredbe.
3. Nadopunite postojeću tekstualnu datoteku `rusu_lv_2019_20/LV7/Readme.md` s kratkim opisom vježbe i kratkim opisom rješenja vježbe te pohranite promjene u lokalnu bazu. Na kraju pohranite promjene u udaljeni repozitorij.

Zadatak 1

U prilogu vježbe nalazi se funkcija 7.1. koja služi za generiranje umjetnih podataka kako bi se demonstrirala binarna klasifikacija za slučaj dvije ulazne veličine pomoću neuronske mreže. Funkcija prima cijeli broj koji definira željeni broju uzoraka u skupu, a vraća generirani skup podataka u obliku numpy polja pri čemu su prvi i drugi stupac ulazne veličine, a treći stupac klasa kojoj pojedini uzorak pripada.
Napišite Python kod u kojem ćete demonstrirati upotrebu višeslojne neuronske mreže za rješavanje binarnih klasifikacijskih problema na temelju podataka koji se dobiju pomoću funkcije 7.1. U kodu trebaju biti jasno naznačeni koraci prilikom izgradnje modela te kako je provedeno vrjednovanje modela. Nadalje, potrebno je predstaviti rezultate izgradnje modela i testiranja modela kada se mijenja broj neurona i/ili vrijednost parametra regularizacije. Prikažite granicu odluke koja se dobije pomoću neuronske mreže. Usporedite dobivene rezultate s rezultatima koji se dobiju korištenjem logističke regresije.

Zadatak 2

U prilogu vježbe nalazi se funkcija 7.2. koja služi za generiranje umjetnih podataka kako bi se demonstrirala izgradnja regresijskog modela pomoću umjetne neuronske mreže. Funkcija prima cijeli broj koji definira željeni broj uzoraka u skupu, a vraća generirani skup podataka u obliku numpy polja pri čemu se u prvom stupcu nalaze vrijednosti ulazne veličine *x*, u drugom stupcu su odgovarajuće stvarne vrijednosti izlazne veličine *y*, dok se u trećem stupcu nalaze odgovarajuće mjerene vrijednosti izlazne veličine (stvarna vrijednost + pogreška).
Napišite Python kod u kojem ćete demonstrirati upotrebu višeslojne neuronske mreže za rješavanje regresijskih problema na temelju podataka koji se dobiju pomoću funkcije 7.2. U kodu trebaju biti jasno naznačeni koraci prilikom izgradnje modela te kako je provedeno vrjednovanje modela. Nadalje, potrebno je predstaviti rezultate izgradnje modela i testiranja modela kada se mijenja broj neurona i/ili vrijednost koeficijenta regularizacije. Usporedite dobivene rezultate s rezultatima koji se dobiju korištenjem linearnog regresijskog modela.

V. Izvještaj s vježbe

Kao izvještaj s vježbe prihvaća se web link na repozitorij pod nazivom `rusu_lv_2019_20` koji sadrži rješenja unutar direktorija `rusu_lv_2019_20/LV7/solutions/`.

VI. Dodatak

```
import numpy as np
```

Funkcija 7.1. – generiranje podataka; klasifikacijski problem

```
def generate_data(n):

    #prva klasa
    n1 = n/2
    x1_1 = np.random.normal(0.0, 2, (n1,1));
    #x1_1 = .21*(6.*np.random.standard_normal((n1,1)));
    x2_1 = np.power(x1_1,2) + np.random.standard_normal((n1,1));
    y_1 = np.zeros([n1,1])
    temp1 = np.concatenate((x1_1,x2_1,y_1),axis = 1)

    #druga klasa
    n2 = n - n/2
    x_2 = np.random.multivariate_normal((0,10), [[0.8,0],[0,1.2]], n2);
    y_2 = np.ones([n2,1])
    temp2 = np.concatenate((x_2,y_2),axis = 1)

    data  = np.concatenate((temp1,temp2),axis = 0)

    #permutiraj podatke
    indices = np.random.permutation(n)
    data = data[indices,:]

    return data
```

Funkcija 7.2. – generiranje podataka; regresijski problem

```
import numpy as np

def add_noise(y):

    np.random.seed(14)
    varNoise = np.max(y) - np.min(y)
    y_noisy = y + 0.1*varNoise*np.random.normal(0,1,len(y))
    return y_noisy


def non_func(n):

    x = np.linspace(1,10,n)
    y = 1.6345 - 0.6235*np.cos(0.6067*x) - 1.3501*np.sin(0.6067*x) - 1.1622 * np.cos(2*x*0.6067) - 0.9443*np.sin(2*x*0.6067)
    y_measured = add_noise(y)
    data = np.concatenate((x,y,y_measured),axis = 0)
    data = data.reshape(3,n)
    return data.T
```