Razpoznavanje vzorcev

Simulacija triplastnega nevronskega omrežja (perceptrona)

Jernej Sabadin

Univerza *v Ljubljani*
Fakulteta *za elektrotehniko*

Mentorja: izr. prof. dr. Simon Dobrišek, as. dr. Klemen Grm

`Predmet:  Razpoznavanje vzorcev`
`Datum:  9 December 2022`

Contents

Ključne besede: Nevronsko omrežje, XOR problem, Razvrščanje angleških besed

---

1 Naloge

- Opis triplastnega nevronskega omrežja (perceptrona)
- Programiranje perceptrona v okolju python
- preizkus na XOR problemu
- preizkus razvrščanja izgovorjenih angleških črk

2 Grob opis nevronskega omrežja

"Umetno nevronsko omrežje je umetna oblika človekovih možganov. Ti so se sposobni naučiti novih stvari in se prilagajati na spreminjajoče se okolje. Na primer dojenček prepozna mater po vonju in glasu. Brati zmoremo pisave drugih ljudi. V slabi vidljivosti prepoznamo predmete po njihovi obliki. Zaradi svoje sposobnosti nadziranja telesa, razmišljanja, vizualizacije, sanjanja, domišljanja in učenja so možgani bolj zmogljivi kot najnaprednejši računalnik.

Možgani so sestavljeni iz nevronov, njihove povezave pa tvorijo nevronsko omrežje. Biološki nevron je sestavljen iz celičnega telesa, aksona in dendrita. Nevron sprejema signale od drugih nevronov prek dendritov. Ko moč signala preseže določen prag, ta nevron sproži lasten signal, ki se prenese na naslednje nevrone."[4]

3 Model nevrona

"Gradnik nevronskega omrežja je nevron. Nevron prejme več vhodov in ima en sam izhod. Priključeni vhodi so pomnoženi z utežmi. Uteženi vhodi se seštejejo s pragom nevrona. Prag nevrona predstavlja vrednost, pri kateri se aktivira izhod. Utežena vsota vstopa v aktivacijsko funkcijo, ki definira, kako se ta pretvori v izhod.

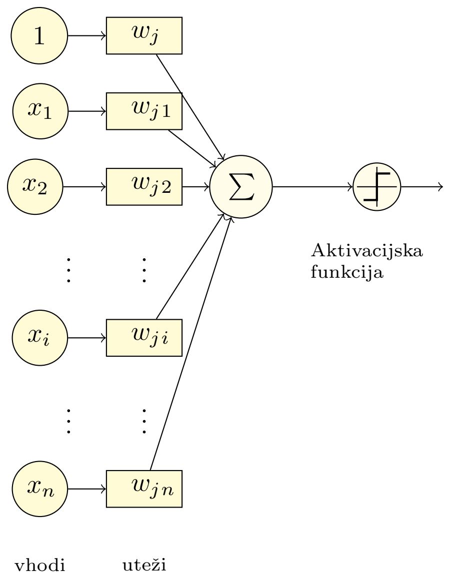

Fig. 1: Slika prikazuje model nevrona

Utež med i-tim in j-tim nevronom predstavimo z w_{ji}. Vsoto vhodov v j-ti nevron in njenega praga w_j zapišemo z izrazom:

$$z_j = \left(\sum_{i=1}^{n} w_{ji} x_i \right) - w_j \tag{1}$$

Utežena vsota vstopa v nelinearen element oz. aktivacijsko funkcijo. Izhod iz te definiramo kot:

$$y_j = f(z_j) \tag{2}$$

Aktivacijska funkcija je podobna aktivnostim v naših možganih, kjer se različni nevroni sprožijo z različnimi dražljaji. Najpogostejša v praksi je sigmoidalna funkcija. "[4]

4 Opis triplastnega nevronskega omrežja (perceptrona)

"V naši nalogi uporabimo triplastno nevronsko omrežje, ki ga opišemo s pomočjo knjige Razpoznavanje vzorcev Nikole Pavšiča.

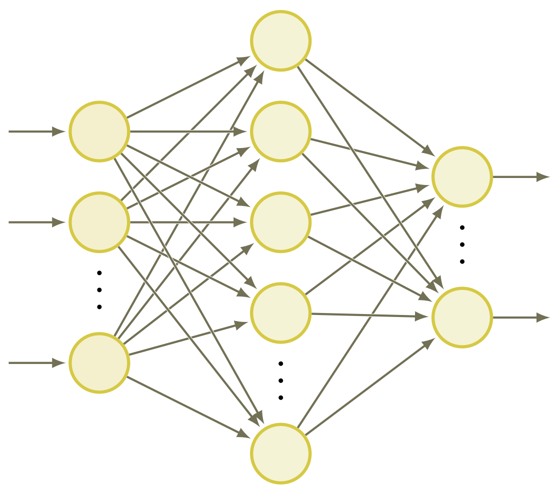

Fig. 2: Shema nevronskega omrežja

Izrazimo j-ti nevron v l-ti plasti z

$$x_j^{(l)} = f\left(\sum_{i=1}^{n_{l-1}+1} w_{ji}^{(l)} x_i^{(l-1)} \right) \tag{3}$$

kjer upoštevamo, da so vhodi $x_{n_{l-1}+1}^{(l-1)} = 1$. Števila nevronov v posamezni plasti predstavimo z n_l. Število $x_i^{(l-1)}$ pa predstavlja i-ti nevron v plasti $l-1$.

Za aktivacijsko funkcijo $f(z)$ uporabimo sigmoidalno funckijo definirano z:

$$f(x) = \frac{1}{1+e^{-x}} \tag{4}$$

Značilke $x_i^{(0)}$ vzorca, ki ga razvrščamo, razvrstimo v razred vzorcev C_i, če za izhode nevronov zadnje plasti velja

$$x_i^{(2)} > x_j^{(2)} \quad \text{za} \quad j = 1, 2, ..., M; \quad j \neq i, \tag{5}$$

pri čemer je M število vseh razredov.

4.1 Učenje triplastnega perceptrona

V postopku vzvratnega učenja (angl. back-propagation training algorithm) minimiziramo razliko med želenim poznanim $x^{(4)}$ in dejanskim $\mathbf{t}$ izhodom. Uteži se nastavijo na take vrednosti, da minimizirajo napako.

Postopek učenja temelji na gradientni metodi. Uteži se skozi postopek učenja popravljajo, dokler se ne ustalijo na določeni vrednosti. Nove vrednosti uteži izračunamo s pomočjo delnih odvodov funkcije napake. To storimo za vsako plast in vsak učni vzorec. V $k+1$ koraku popravimo uteži po naslednjem izrazu:

$$w_{ji}(k+1)^{(l)} = w_{ji}^{(l)}(k) - \beta \frac{\partial e\left(w(k)\right)}{\partial w_{ji}^{(l)}(k)} = w_{ji}^{(l)}(k) + \Delta w_{ji}^{(l)}(k), \tag{6}$$

kjer je $e(w(k))$ napaka nevronskega omrežja v k-ti ponovitvi postopka:

$$e(w(k)) = \frac{1}{2} \sum_{p=1}^{n_2} \left(t_p(k) - x_p^{(2)}(k) \right)^2 \tag{7}$$

Pri tem je t želen odziv omrežja. Označimo ga z vektorjem ničel in eno enko, ki ponazarja dani razred.

Faktor β, ki je manjši od ena, določa korak premika k minimumu funkcije napake. Če ga nastavimo na vrednost 1, se bomo proti minimumu premikali prehitro in ga bomo preskočili. Če pa izberemo premajhnega, se računska zahtevnost učenja poveča.

Učenje nevronskega omrežja poteka tako s pomočjo delnih odvodov napake."[4]

Za izhodno plast pišemo:

$$
\begin{aligned}
\frac{\partial e\left(w(k)\right)}{\partial w_{ji}^{(2)}(k)} &= \frac{\partial e\left(w(k)\right)}{\partial x_j^{(2)}(k)} \cdot \frac{\partial x_j^{(2)}(k)}{\partial z_j^{(2)}(k)} \cdot \frac{\partial z_j^{(2)}(k)}{\partial w_{ji}^{(2)}(k)} \\
&= \left(t_j(k) - x_j^{(2)}(k)\right)\left(1 - x_j^{(2)}\right) x_j^{(2)}(k) x_i^{(1)}(k) \\
&= -d_j^{(2)}(k) x_i^{(1)}(k)
\end{aligned} \tag{8}
$$

kjer je

$$
d_j^{(2)}(k) = \left(t_j(k) - x_j^{(2)}(k)\right)\left(1 - x_j^{(2)}\right) x_j^{(2)}(k) \tag{9}
$$

Popravki uteži $\Delta w_{ji}^{(2)}$ so tako definirani kot:

$$
\Delta w_{ji}^{(2)} = \beta d_j^{(2)}(k) x_i^{(1)}(k) \tag{10}
$$

poprkavki uteži v plasi $l = 1$ pa so definirani kot:

$$
\Delta w_{ji}^{(1)} = \beta d_j^{(1)}(k) x_i^{(0)}(k) \tag{11}
$$

za $i = 1, 2, ..., (n_{l-1} + 1)$, $j = 1, 2, ..., n_l$ in $l = 1$, kjer je:

$$
d_j^{(1)}(k) = \left(1 - x_j^{(1)}\right) x_j^{(1)}(k) \sum_{p=1}^{n_2} d_p^{(2)}(k) w_{pj}^{(2)}(k) \tag{12}
$$

Postopek učenja poteka tako, da za vhodne označene podatke izračunamo izhode. Glede na pričakovane izhode in dejanske izračunamo napako ter na podlagi nje popravimo vse uteži. To ponovimo tolikokrat, dokler se uteži ne umirijo.

5 Triplastno nevronsko omrežje v okolju Python

Nevronsko omrežje predstavimo z razredom TCP().

TCP dovoljuje preizkušanje omrežja s poljubnim podanim številom nevronov druge vmesne prikrite plasti. Število nevronov prve plasti je prepisano z razsežnostjo vzorcev. Število nevronov izhodne plasti pa mora biti enako številu razredov.

Uteži definiramo z funkcijo:

```python
def init_weights(self):
    # Initialization of weights and bias
    self.w = []
    self.b = []
    for i in range(self.layer_sizes.shape[0]-1):
        # weights are set as random numbers betwen -0.5 in 0.5
        self.w.append(np.random.uniform(-0.5,0.5,size=[self.layer_sizes[i+1],self.layer_sizes[i]]))
        self.b.append(np.zeros((self.layer_sizes[i+1],1)))
    self.w = np.array(self.w, dtype=object)
    self.b = np.array(self.b, dtype=object)
```

Fig. 3: Koda za inicalizacijo uteži

Nevrone pa z funkcijo:

```python
def init_layers(self):
    # Initialization of layers
    self.x = [np.empty((layer,1)) for layer in self.layer_sizes]
```

Fig. 4: Koda za inicalizacijo nevronov

kjer so w uteži, b pragovi, x nevroni, layer_sizes pa dimenzije matrike nevronov pri poljubno podanem številu nevronov v prikiriti plasti N_l2. Pri tem smo pozorni na dimenzije matrik, ki jih definiramo, zato enostaven primer nevronskega omrežja rešimo na list papirja. V primeru da imamo 2 razreda, 2 značilke na vzorec in 3 nevrone v prikriti plasti, mora biti layer_sizes dimenzij (2,3,2), matrika uteži w vsebovati podmatriki dimezij (3,2) (2,3), matrika nevronov mora vsebovati vektorje dimenzij (2,1),(3,1),(2,1) in matrika pragovov vsebovati vektorje dimenzij (3,1), (2,1).

Glede na tako definirane uteži, pragove in nevrone definiramo funkcijo, ki glede na vhodne podatke izračuna izhode po formuli (3):

```python
def forward_propagation(self,sample):
    x_l = sample
    self.x[0] = x_l
    for i in range(len(self.w)):
        x_l = self.sigmoid(np.matmul(self.w[i],self.x[i]) + self.b[i])
        self.x[i+1] = x_l
```

Fig. 5: Koda za izračun izhodov

kjer je sigmoid() sigmoidalna funkcija iz enačbe (4).

Funkcijo, ki na podlagi dejanskh in izračunanih izhodov popravi uteži in pragove po formuli (12), (11), (10) in (9) definiramo kot:

```python
def backward_propagation(self,y_target):
    d = (y_target - self.x[-1])*self.sigmoid_derivate(self.x[-1]) # * is elementwise operator! thats what we want!
    for i in range(1,len(self.w)+1):
        delta_w = self.Beta*np.matmul(d,np.transpose(self.x[-i-1]))
        delta_b = self.Beta*d
        self.w[-i] += delta_w
        self.b[-i] += delta_b
        d = self.sigmoid_derivate(self.x[-i-1])*np.matmul(np.transpose(self.w[-i]),d)
```

Fig. 6: Koda izračun izhodov

kjer so delta_w popravki uteži, delta_b popravki pragov in d vektorji členov iz enačb (9) in (12).

Za prej omenjen enostaven primer morajo biti matrike delta_w dimenzij (3,2) (2,3), matrika delta_b dimenzij (3,1),(2,1) in vektorji d dimenzij (2,1) in (3,1).

Funkcijo učenja, ki spreminja uteži omrežja definiramo kot:

```python
def train(self):
    print("training has started:")
    for epoch in tqdm(range(1,self.epochs+1)):
        self.y_predicted = []
        self.init_layers()
        train_acc = 0
        test_acc = 0
        for feature, name in zip(self.X,self.Y):
            # For every sample we do forward and backward propagation
            self.forward_propagation(feature.reshape(self.layer_sizes[0],1)) # (2,) ->(2,1) pretvorba v vektor
            self.backward_propagation(name.reshape(self.layer_sizes[2],1))   # ista pretvorba
            train_acc += self.evaluate(feature,name)
        train_acc = train_acc/len(self.X)
        self.train_acc.append(train_acc)
        # if we added test set:
        if(isinstance(self.X_test, (np.ndarray)) & isinstance(self.y_test, (np.ndarray))):
            for feature, name in zip(self.X_test,self.y_test):
                self.forward_propagation(feature.reshape(self.layer_sizes[0],1))
                test_acc += self.evaluate(feature,name)
                self.y_predicted.append(self.predict(feature))
            test_acc = test_acc/len(self.X_test)
            self.test_acc.append(test_acc)
            print( f" training acc = {int(self.train_acc[-1]*100)}%,testing acc = {int(self.test_acc[-1]*100)}%, epoch = {epoch}")
            # if we didnt add test list
        else:
            print( f" training acc = {int(self.train_acc[-1]*100)}%, epoch = {epoch}")
    print(" training is finished, all weights are corrected")
```

Fig. 7: Koda funckije train()

kjer je epochs število vseh ponovitev popravljanja uteži, funckija sigmoid_derivate() pa je definirana z predpisom iz enačbe (9) in (12) kot $\left(1 - x_j^{(l)}\right) x_j^{(l)}(k)$.

Med postopkom učenja si shranjujemo odstotek pravilno razpoznanih vzorcev, ki ga v nadaljevanju uporabimo za prikaz rezultatov.

Funckija evaluate() vrne 1, če je vzorec pravilno razpoznan. Pri tem pa uporablja funckijo predict(), ki vrne predikcijo razreda, ki mu vzorec pripada. To je vektor ničel z enko na mestu, kjer je stopnja razpoznavanja največja oziroma nevron najbolj "aktiven" po enačbi (5). Funkcija predict() pri tem uporabi funkcijo to_categorical(), ki transformira vektor v vektor ničel z eno enko na mestu največjega elementa. Na primer vektor [0.98, 0.1] v vektor [1, 0].

Če vnesemo testne podatke z funkcijo fit(), izvedemo tudi sprotno testiranje našega omrežja med fazo učenja.

Omenjene funkcije prikažemo na spodnji sliki:

```
def fit(self, X_test,y_test):
    # To fit new test data!
    self.X_test = X_test
    self.y_test = y_test

def evaluate(self,X,Y):
    # returns 1 if X  is clasified as Y and 0 if not.
    prediction = self.predict(X)
    return int(np.all(Y==prediction))

def predict(self,X):
    # We retrun prediciton of class for. X has to be np.array!
    X_ = X.reshape(self.layer_sizes[0],1)
    self.init_layers()
    self.forward_propagation(X_)
    prediction = self.to_categorical(np.transpose(self.x[-1]))
    return prediction

def to_categorical(self,y):
    # We retrun vector with one 1 and all other elements 0. For example [1 0 0 0]
    categorical = np.zeros_like(y.reshape(self.layer_sizes[2]))
    categorical[np.argmax(y)] = 1
    return categorical
```

Fig. 8: Koda omenjenih funkcij evaluate(), predict(), to_categorical() in fit()

Rezultate izrišemo z funckijo plt_results():

```
def plt_results(self):
    '''
            @param "train": then the train results will be plotet
            @param "test": then the test results will be plotet
            plt.show() has to be used afterward
            example of usage -> plt.figure()
                                plt_result()
                                plt.show()

    '''
    epochs = np.arange(1,self.epochs+1)
    if(isinstance(self.X_test, (np.ndarray)) & isinstance(self.y_test, (np.ndarray))):
        plt.title("Training and testing accuarcy")
        plt.xlabel("epochs")
        plt.ylabel("accuarcy in %")
        plt.plot(epochs,np.array(self.train_acc)*100, label = f"Train, Beta={self.Beta}, N_12 ={self.N_12} ")
        plt.plot(epochs,np.array(self.test_acc)*100, label = f"Test, Beta={self.Beta}, N_12 ={self.N_12}")
        plt.legend()
    else:
        plt.title("Training accuarcy")
        plt.xlabel("epochs")
        plt.ylabel("accuarcy in %")
        plt.plot(epochs,np.array(self.train_acc)*100, label = f"Train, Beta={self.Beta}, N_12 ={self.N_12} ")
        plt.plot(epochs,np.array(self.train_acc)*100)
        plt.legend()
```

Fig. 9: Koda funkcije izrisa

Spodaj prikažemo kodo za izračun in izris konfuzijske matrike pri podanih testnih podatkih.

```
def plt_confusion_matrix_test_data(self,axes_names):
    cm = np.zeros((self.layer_sizes[2], self.layer_sizes[2]))
    y_test = [np.argmax(self.y_test[i]) for i in range(len(self.y_test))]
    y_predicted =[np.argmax(self.y_predicted[i]) for i in range(len(self.y_predicted))]
    for test,true in zip(y_test,y_predicted):
        cm[test][true] +=1
    for i in range(len(axes_names)):
        cm[i,:]  =  cm[i,:]/np.sum(cm[i,:])
    plt.imshow(cm)
    plt.xticks(np.arange(len(axes_names)), axes_names)
    plt.yticks(np.arange(len(axes_names)), axes_names)
    plt.colorbar()
    plt.tight_layout()
    plt.show()
```

Fig. 10: Koda za izračun in izris konfuzijske matrike

6 XOR problem

XOR problem predstavimo s podatki na sliki:

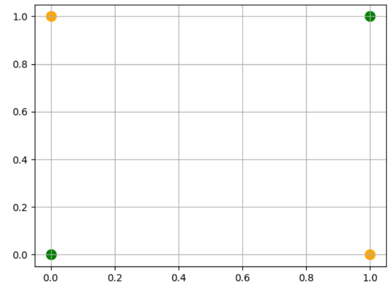

Fig. 11: XOR podatki

kjer oranžni točki označimo z $\omega_1 = [0, 1]$ in zeleni z $\omega_2 = [1, 0]$. ω_1 je oznaka razreda C_1 in ω_2 je oznaka razreda C_2

Točke želimo razvrstiti z odločitveno funkcijo tako kot prikazuje spodnja slika:

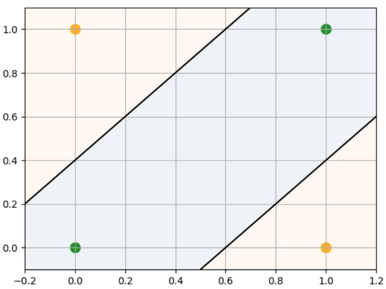

Fig. 12: ločilna meja, ki ločuje razreda

Izvedemo poskus razvrščanja s spodnjo kodo:

```
if __name__ == "__main__":
    ########################################################
    # XOR Problem
    X_train = np.array([[0,0], [0,1],[1,0],[1,1]])
    y_train = np.array(([[0,1],[1,0],[1,0],[0,1]]))

    # Defining and training our network
    ann = TLP(X_train,y_train,4,7000,0.2)
    ann.train()

    #Showing results of our network
    for x,y in zip(X_train,y_train):
        print(f"sample {x}, marked as {y}, is classified as {ann.predict(x)}  ")


    ann.plt_results()
```

Fig. 13: koda za rešitev problema XOR

in prikažemo rezultate:

```
sample [0 0], marked as [0 1], is classified as [0. 1.]
sample [0 1], marked as [1 0], is classified as [1. 0.]
sample [1 0], marked as [1 0], is classified as [1. 0.]
sample [1 1], marked as [0 1], is classified as [0. 1.]
```

Fig. 14: XOR rešitev

prikažemo tudi kako se je odstotek razpoznavanja spreminjal skozi učenje.

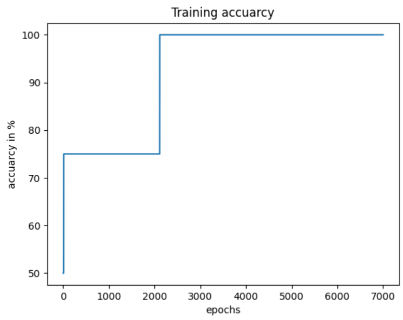

Fig. 15: Odstotek pravilnega razvrščanja skozi korake učenja

7 Razvrščanje izgovorjenih angleških črk

7.1 Kratek opis zbirke

Baza učnih podatkov vsebuje 6238 vzorcev dolžin 617 značilk, baza testnih podatkov pa 1559 vzorcev dolžin 617 značilk. Vzorci so razporejeni v 26 razredov, kjer vsak razred predstavlja eno črko angleške abecede. Tako je A označen kot 1, B označen kot 2 itd.

Velikosti posameznih razredov so skoraj da enake (240 vzorcev na učni razred, le pri črki F manjkata 2 vzorca, 60 vzorcev na testni razred, kjer pa en vzorec manjka pri črki M)

7.2 Poskus razpoznavanja

Predno se lotimo razvrščanja primerno uredimo zbirko in označimo vzorce z vektorji dolžin 26, ki imajo le en elemnt neničeln in sicer 1. Tako je črka A označena z $\omega_1 = [1, 0, , ..., 0]$, črka B z $\omega_2 = [0, 1, 0, ..., 0]$,..., in črka Z z $\omega_{26} = [0, ..., 0, 1]$. Pri čemer je ω_1 oznaka razreda C_1, ω_2 oznaka razreda C_2,..., ω_{26} oznaka razreda C_{26}.

Spodaj prikažemo kako smo bazo podatkov shranili v okolju python:

```python
# WORDS PROBLEM
# Reading train and teset data and saving it into np.array
read_csv_train = pd.read_csv('isolet1+2+3+4.csv',header=None)
all_train_data = np.array([read_csv_train.iloc[i] for i in range(len(read_csv_train))])# we sort data into 6238 sets, each long 618.
read_csv_test = pd.read_csv('isolet5.csv',header=None)
all_test_data = np.array([read_csv_test.iloc[i] for i in range(len(read_csv_test))])# we sort data into 1559 sets, each long 618.
# A-Z -> 26 classes
num_class = int(np.max(np.array(read_csv_train.iloc[: , -1]))) #Number of classes

# Saving data for classifcation problem
X_train = []
y_train = []
X_test = []
y_test = []
for data_train in all_train_data:
    X_train.append((data_train[:-1]))
    y_train.append(transform_to_vector(int(data_train[-1]),num_class))
X_train,y_train = np.array(X_train),np.array(y_train)

for data_test in all_test_data:
    X_test.append((data_test[:-1]))
    y_test.append(transform_to_vector(int(data_test[-1]),num_class))
X_test,y_test = np.array(X_test),np.array(y_test)
```

Fig. 16: Koda za shranjevanje vzorcev in njihovih oznak

Poskus razvrščanja izvedemo tako, da spreminjamo število nevronov skrite plasti, učni faktor oziroma faktor vztrajnosti. Za učenje triplastnega perceptrona uporabimo učni del zbirke. Testni del uporabimo za preizkus razpoznavalnika.

Koda v pythonu izgleda tako:

```python
### EXPERIMENT ###
#Changing learning rate, and number of neurons in hidden layer
rezultati_train = []
rezultati_test = []
for Beta in [0.01, 0.05, 0.1, 0.3, 0.6, 0.9]:
    rezultati_za_ta_Beta_train = []
    rezultati_za_ta_Beta_test = []
    for N_l in [10,20,50,80,110,140,170,200]:
        ann = TLP(X_train,y_train,N_l,15,Beta)
        ann.fit(X_test,y_test)
        ann.train()
        uspesnosti_train = ann.train_acc[-1]
        uspesnosti_test = ann.test_acc[-1]
        rezultati_za_ta_Beta_train.append(np.mean(uspesnosti_train))
        rezultati_za_ta_Beta_test.append(np.mean(uspesnosti_test))
    rezultati_train.append(rezultati_za_ta_Beta_train)
    rezultati_test.append(rezultati_za_ta_Beta_test)
np.savetxt("rezultati_train.csv", rezultati_train, delimiter=",")
np.savetxt("rezultati_test.csv", rezultati_test, delimiter=",")
```

Fig. 17: Koda za preizkus razpoznavalnika pri različnih faktorjih β in različnem številu nevronov v izhodni plasti

Spodaj prikažemo rezultate experimenta:

Beta	N_l2 = 10	N_l2 = 20	N_l2 = 50	N_l2 = 80	N_l2 = 110	N_l2 = 140	N_l2 = 170	N_l2 = 200
0.01	7,65%	55,98%	89,52%	89,58%	94,53%	93,27%	95,25%	95,56%
0.05	53,83%	97,40%	98,78%	99,01%	99,07%	99,17%	99,17%	99,26%
0.1	81,55%	98,41%	99,37%	99,50%	99,33%	99,36%	99,23%	99,13%
0.3	74,66%	97,45%	99,23%	99,41%	99,44%	99,26%	99,20%	98,85%
0.6	71,48%	94,24%	96,68%	96,15%	94,10%	85,14%	59,09%	50,08%
0.9	79,53%	88,20%	84,98%	70,09%	50,30%	39,08%	35,97%	12,68%

Fig. 18: Rezultati učenja pri različnih faktorjih učenja in različnem številu nevronov v drugi plasti

Beta	N_l2 = 10	N_l2 = 20	N_l2 = 50	N_l2 = 80	N_l2 = 110	N_l2 = 140	N_l2 = 170	N_l2 = 200
0.01	7,70%	55,81%	87,24%	88,39%	91,15%	90,31%	92,82%	92,37%
0.05	51,64%	91,53%	94,42%	94,10%	94,42%	94,87%	95,19%	93,91%
0.1	69,85%	90,96%	94,29%	95,25%	94,80%	95,54%	94,55%	94,87%
0.3	47,85%	90,96%	94,10%	94,68%	95,38%	95,57%	95,45%	95,00%
0.6	36,11%	89,22%	89,80%	88,33%	89,54%	82,55%	56,70%	52,92%
0.9	68,18%	74,47%	77,16%	68,25%	47,72%	43,55%	33,61%	12,89%

Fig. 19: Rezultati testiranja pri različnih faktorjih učenja in različnem številu nevronov v drugi plasti

Opazimo, da zna naš razpoznavalnik zelo dobro razpoznavati učne podatke, testne podatke pa razvršča rahlo slabše, kar je pričakovano. Dobro se na testni zbirki odreže parameter Beta oz. faktor učenja nastavljen na vrednost 0.3 pri 140 nevronih v prikriti plasti. Opazimo, da pri zelo majhem faktorju učenja ne uspemo doseči minimuma funkcije napake, saj so popravki uteži v vsaki iteraciji premajhni. Pri prevelikem faktorju Beta 0.9 pa minimum funkcije napake preskočimo.

Spodaj prikažemo rezultat poskusa s parametri, ki dosežejo največji odstotek pravilno razpoznanih vzorcev na testni zbirki v zgoraj opisanem eksperimentu.

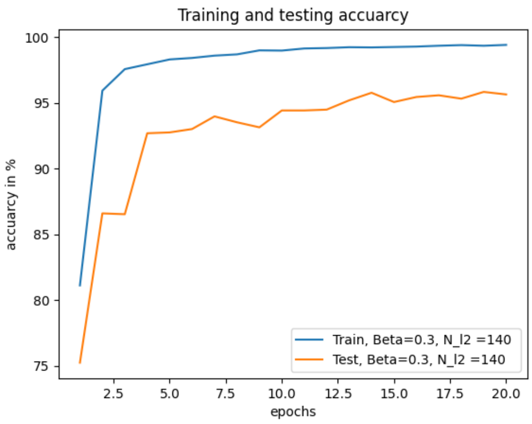

Fig. 20: Rezultati poskusa z parametri Beta = 0.3 in N_l2 = 140

Opazimo, da model razvršča testne podatke 95% natančno, učne podatke pa 99% natančno. Na splošno ni nenavadno, da je natančnost testne zbirke nižja od natančnosti učne zbirke, zlasti če je model kompleksen in ima veliko parametrov. Vendar pa lahko velik razkorak med natančnostjo učenja in preizkusa kaže na overfitting ali težavo z modelom ali procesom učenja. Če želimo izboljšati posplošitev modela, moramo uporabiti nekatere tehnike, kot so zbiranje večje količine podatkov za učenje, uporaba zgodnje ustavitve, različne arhitekture modela ali nastavitve hiperparametrov.

Prikažemo tudi normirano konfuzijsko matriko, pridobljeno na test zbirki:

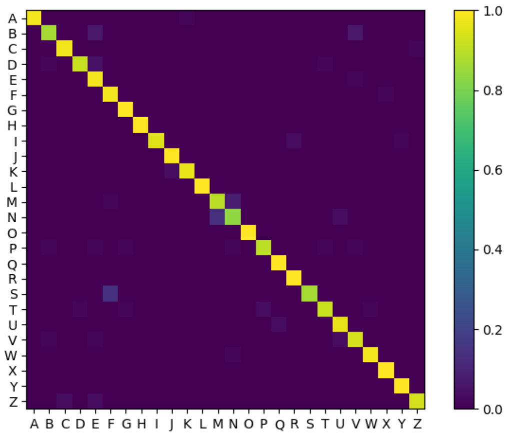

Fig. 21: Normirana konfuzijska matrika, kjer je na y osi informacija o dejanskih fonemih na x osi pa kako so razpoznani

Opazimo, da je najbolje razpoznana izgovorjena črka Y, najslabše pa črka N. Največkrat se ponovi napaka, ko je črka N razpoznana kot M.

8 Uporaba perceptrona na zbirki varnostno sumljivih zvokov

8.1 Kratek opis zbirke

Zbirka je sestavljena iz 2524 varnostno sumljivih zvokov, ki so v povprečju dolgi okoli 5 sekund in razdeljeni v 7 razredov. Razredi so alarm, pasji lajež, eksplozija, lomljenje stekla, kričanje, streljanje in sirena. [4]

8.2 Izpeljava vektrojev značilk

"Zvočni signali, ki jih uporabljamo, so časovno spremenljivi (nestacionarni) in naključni. Na dovolj kratkih odsekih 5-100 ms jih lahko obravnavamo kot stacionarne. Oknenje je metoda pri obdelavi signalov, s katero razdelimo signal na časovne segmente. Naše signale razčlenimo na kratke zaporedne prekrivajoče se izseke z enakim trajanjem. To storimo s pomočjo okenskih funkcij.

Iz zvočnih signalov izluščimo koeficiente kratkočasovnega melodičnega kepstra t. i. značilke MFCC. V nadaljevanju na kratko opišemo postopek določanja značilk MFFC.

1. Za vsak segment signala, ki smo ga pridobili z oknenjem, izračunamo njegovo transformiranko. To storimo z DFT.

$$F(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)e^{-i\frac{2\pi nk}{N}}, \qquad 0 \le k \le N-1, \tag{13}$$

kjer je N število točk, ki jih uporabimo za izračun DFT.

2. Kepster signala definiramo kot

$$\{c(n)\} = \mathcal{F}^{-1}\{\log F(k)\}. \tag{14}$$

Ker uho ni občutljivo na fazne zamike med frekvenčnimi komponentami, uporabimo le močnostni spekter signala.

$$\{c(n)\} = \mathcal{F}^{-1}\{\log |F(k)|^2\} \tag{15}$$

3. Za nadaljnje izboljšanje kepstralne reprezentacije v enačbo {c(n)} vključimo več informacij o slušnem zaznavanju. Log-spekter že upošteva zaznavno občutljivost po amplitudni osi, saj je občutljivost ušesa po amplitudi logaritemska. Slušno zaznavanje pa je tudi po frekvenčni osi porazdeljeno nelinearno.

$$f_{\mathrm{Mel}} = 2595\log_{10}(1 + \frac{f}{700}), \tag{16}$$

kjer je f frekvenca v Hz f_{Mel}, pa zaznana frekvenca.

Zato se za izračun kepstra uporablja logaritme povprečnih moči frekvenčnih območij, razporejenih po melodični delitvi. Tako utežen kepster imenujemo melodični kepster. Uteženo povprečje močnostnega spektra oknenjega zvočnega odseka izračunamo kot:

$$s(m) = \sum_{k=0}^{N-1} [|F(k)|^2 H_{\mathrm{m}}(k)], \qquad 0 \leq m \leq M-1, \tag{17}$$

kjer je M število vseh trikotnih melodičnih filtrov. Vsak filter je neničeln le na določenih frekvencah. $H_{\mathrm{m}}(k)$ je utežna funkcija, vezana na k-ti vzorec spektra. Izražena je kot:

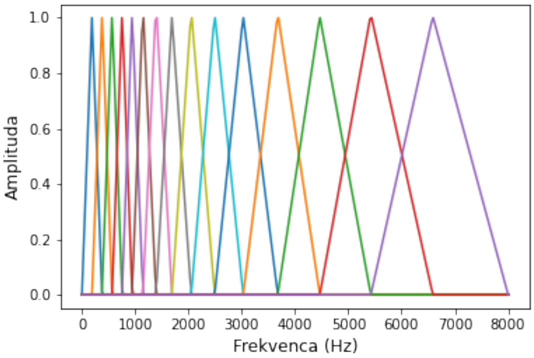

Fig. 22: Melodični filtri

Izračun koeficientov melodičnega kepstra s pomočjo diskretne kosinusne transformacije:

$$c_{\mathrm{Mel}}(n) = \sum_{m=0}^{N-1} (\log s(m)) \cos\left(\frac{\pi n(m-0.5)}{M}\right); \tag{18}$$

kjer je $n = 0, 1, 2, ..., C-1$, C število koeficientov MFCC. Izračunani koeficienti predstavljajo vektor značilk, ki pripada posameznemu odseku signala, pridobljenega z oknenjem.

Uporabna informacija o zvočnem signalu so tudi prvo- in drugostopenjski odvodi oz. razlike koeficientov-MFCC. Imenujemo jih delta in delta-delta značilke MFCC. Delta in delta-delta koeficienti nosijo informacije o hitrosti spreminjanja MFCC značilk.

Poleg omenjenih značilk za vsak oknjen izsek uporabimo tudi informacijo o kratkočasovni glasnosti izseka oz. njegovi energiji. Izračuna se jo z izrazom:

$$E_{\mathrm{x}} = \sum_{n=-\infty}^{\infty} |x(n)|^2 \tag{19}$$

"[4]

Zvočne signale torej oknimo s pravokotno okensko funkcijo, dolžine 25 ms s prekrivanjem 10 ms. Posameznemu posnetku priredimo za vsako okno svoj vektor značilk. Vektor vsebuje 12 melodičnih kepstralnih koeficientov ter kratkočasovno glasnost. Poleg omenjenih 13 značilk izračunamo še dinamične značilke 1. in 2. reda. Za posamezen zvočni posnetek dobimo matriko značilk, ki ima eno dimenzijo za vse posnetke enako, druga dimenzija pa je povezana z dolžino posnetka oz. številom oken. Matrike povprečimo po zgoraj omenjenih značilkah. Tako dobimo vektorje dimenzij 39.

8.3 Učenje in testiranje perceptrona

Vektroji značilk so dolžine 39. V učni bazi imamo 1640, v testni pa 884 vzorcev. Alarm je označen z $\omega_1 = [1, 0, , ..., 0]$, pasji lajež z $\omega_2 = [0, 1, 0, ..., 0]$,..., in sirena z $\omega_7 = [0, ..., 0, 1]$. Pri čemer je ω_1 oznaka razreda C_1, ω_2 oznaka razreda C_2,..., ω_7 oznaka razreda C_7.

Pri izbranem faktroju Beta = 0.3 in številom nevronov 140 v prikriti plasti, dosežemo na omenjeni zbirki varnostno sumljivih zvokov odstotek razpoznavanja 88%.

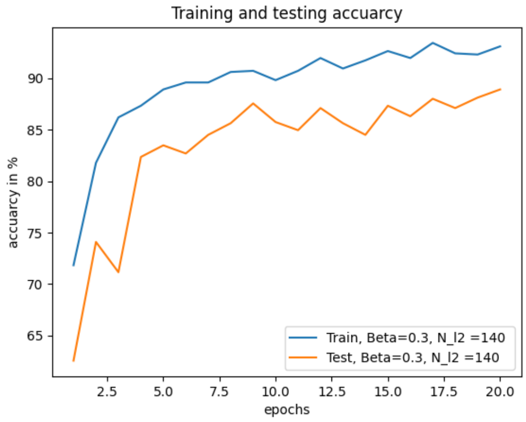

Fig. 23: Rezultati poskusa z parametri Beta = 0.3 in N_l2 = 140

Prikažemo tudi konfuzijsko matriko:

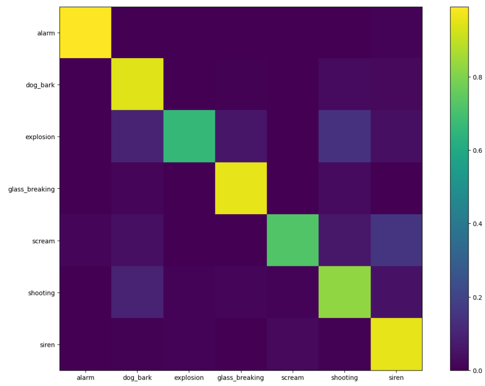

Fig. 24: Normirana konfuzijska matrika, kjer je na y osi informacija o dejanskih fonemih na x osi pa kako so razpoznani

Koda za poskus je sledeča:

```
with open('test.npy', 'rb') as f:
    X_train= np.load(f)
    y_train = np.load(f)

with open('test.npy', 'rb') as f:
    X_test = np.load(f)
    y_test= np.load(f)

ann = TLP(X_train,y_train,140,20,0.3)
ann.fit(X_test,y_test)
ann.train()
ann.plt_results()
cm_plot_labels = ["alarm","dog_bark","explosion","glass_breaking","scream","shooting","siren"]
ann.plt_confusion_matrix_test_data(cm_plot_labels)
```

Fig. 25: Koda poskusa na zbirki varnosnto sumljivih zvokov

9 Povzetek

V nalogi se seznanimo s triplastnim nevronskim omrežjem. Uporabimo enačbe v delu Nikole Pavešiča in napišemo model nevronksega omrežja v okolju pyhton.

Model preverimo na linearno neločljivem primeru XOR, s štirimi nevroni v prikriti plasti.

Razpoznavalnik preizkusimo tudi na ISOLET zbirki, kjer dosežemo stopnjo razpoznavnja testne zbirke 95%.

Razpoznavalnik preizkusimo na zbirki varnostno sumljivih zvokov in dosežemo odstotek razpoznavanja 88%.

10 Reference

1 N. Pavešić, 'Razpoznavanje vzorcev: Uvod v analizo in razumevanje vidnih in slušnih signalov", 3. izd., Ljubljana: Založba FE in FRI, 2012.

2 S. Dobrišek: Razpoznavanje vzorcev, gradivo za predavanja, FE 2022

3 K. Grm: Razpoznavanje vzorcev, gradivo za laboratorijske vaje, FE 2022

4 J. Sabadin: Diplomska naloga: Sistem za razpoznavanje varnostno sumljivih zvokov. Ljubljana. Fakulteta za elektrotehniko, Univerze v Ljubljani. 2022

- Dokumentacija uporabljenih knjižnic v Pythonu:
  - Osnovna verzija programa: *itertools,tqd, numpy, matplotlib, pandas*