<Project Name>
Tehnička dokumentacija
Verzija <1.0>

Studentski tim: <Ime i prezime>
<Ime i prezime>
<Ime i prezime>
....

Nastavnik: <Ime i prezime>

Sadržaj

Tehnička dokumentacija

Na koji način koristiti predložak?

Dokument se po potrebi može prilagoditi potrebama pojedinog projekta promjenom predloženih naslova predloženih poglavlja, kao i eventualnim dodavanjem novih poglavlja i potpoglavlja.

Cilj dokumenta je opisati rezultat rada studentskog tima, problem koji je riješen u okviru projekta, korištenu tehnologiju, mogućnosti i značajke dobivenog proizvoda i sl. Razinu detalja opisanu u ovom dokumentu studentski tim treba dogovoriti s nastavnikom.

Literatura:
U tekstu rada treba biti navedena literatura svugdje gdje je tekst, slika ili grafički prikaz preuzet ili se temelji na nekom pisanom predlošku. Literatura se navodi iza zaključka. U tekstu se literatura navodi unutar zagrada s navođenjem prvog autora i godine izdanja, npr. (Martinis, 1998).

Primjer citiranja knjige:
Prezime, inicijal(i) imena autora. Naslov: podnaslov. Podatak o izdanju. Mjesto izdavanja: Nakladnik, godina izdavanja.

Primjer citiranja članka u časopisu:
Prezime, inicijal(i) imena autora. Naslov članka: podnaslov. Naziv časopisa. Oznaka sveska/godišta, broj(godina), str. početna-završna.

Primjer citiranja rada sa konferencije:
Prezime, inicijal(i) imena autora. Naslov rada: podnaslov. Naslov zbornika, mjesto održavanja konferencije, (godina), str. početna-završna.

Primjer citiranja doktorskog, magistarskog ili diplomskog rada:
Prezime, inicijal(i) imena autora. Naslov. Vrsta rada. Ustanova na kojoj je rad obranjen, godina.

Primjer citiranja www izvora:
Ime(na) autora (ako je/su poznata), naslov dokumenta, datum nastanka (ako se razlikuje od datuma pristupa izvoru), naslov potpunog djela (italic), potpuna http adresa, datum pristupa dokumentu.

Ostale upute
U svim dokumentima obvezno primjenjivati SI jedinice. Slike, formule i tablice potrebno je numerirati. Opis tablice stavlja se iznad, a opis slike ispod nje. U opisu slike ili tablice pišu se samo podaci neophodni za njeno razumijevanje (npr. Slika 6. Pojačalo s promjenljivim pojačanjem). Dodatna objašnjenja daju se u tekstu uz povezivanje sa slikom ili tablicom. Osi i parametri na slikama i grafičkim prikazima trebaju biti obilježeni. Daljnji opis tog grafičkog prikaza treba se nalaziti u tekstu rada. Formule se obilježavaju brojevima u zagradi, uz desni rub stranice, a u tekstu se poziva na broj formule.

1. Opis razvijenog proizvoda

U današnje vrijeme malo je različitih načina komuniciranja sa računalima koja pružaju laganu kontrolu i jednostavnost korištenja. No pošto računala postaju sve veći dio stvarnosti potrebno je tražiti i alternativne načine komunikacije sa njima. Danas, najčešće korišteni uređaji miš i tipkovnica koji pretvaraju naše fizičke pokrete u jezik razumljiv računalu. Na njih možemo gledati kao prevoditelje jezika, no umjesto, na primjer, hrvatskog imamo pokrete ruku ili pritisak tipke koji se pretvaraju u nule i jedinice a ne engleski ili slično. Ali prvi računalni miš izumljen je davne 1970. godine, a tipkovnica čak u 18. stoljeću. Naravno hrabro je tvrditi da će alternativne metode potpuno izbaciti tipkovnicu i miša iz uporabe, ali će zasigurno u kombinaciji sa postojećim tehnologijama olakšati način na koji "komuniciramo" sa računalima

Ideja ovog projekta je omogućiti prepoznavanje skupa gesti ruku te mapirati geste u različite izlaze. To je dovoljno za pokazivanje kako se tim gestama može kontrolirati što god da se isprogramira.

Projekt se sastoji od 3 djela. Oni bi bili:

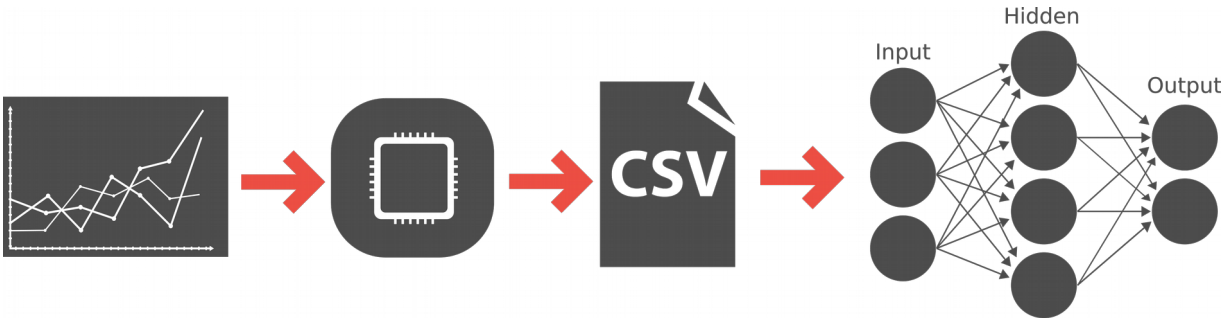
- Pretvaranje pokreta u podatke
- Komunikacija
- Učenje prepoznavanja pokreta iz podataka

Pretvaranje pokreta u podatke izvodi se na mikrokontroleru koji pomoću akcelerometra i žiroskopa koji u svakom diskretnom vremenskom trenutku mjeri 6 vrijednosti (akceleraciju i nagib u 3 osi) čime se omogućuju kompleksni trodimenzionalni pokreti. Komunikacija je serijska u ovom slućaju USB standardom, ali moguć je jednostavan prelazak na Bluetooth komunikaciju ali i druge. Za učenje koristi se unaprijedna neuronska mreža sa algoritmom propagacije pogreške unazad.

Također projekt se odvija u tri faze:

- Prikupljanje podataka
- Učenje neuronske mreže
- Korištenje naučene neuronske mreže za prepoznavanje podataka

Prva faza uključuje očitavanje podataka i njihovo spremanje u trajnu memoriju. Gestu čine podaci iz slijednog niza vremenskih trenutaka točno određene veličine. Takav vektor podataka zatim se proširuje vektorom koji očekujemo za takav izlaz. Potrebno je prikupiti poveće količine podataka radi dobrog raspoznavanja u kasnijim fazama. Druga faza uključuje korištenje tih podataka kako bi se naučila neuronska mreža.



Slika 1: Blokovski dijagram učenja neuronske mreže

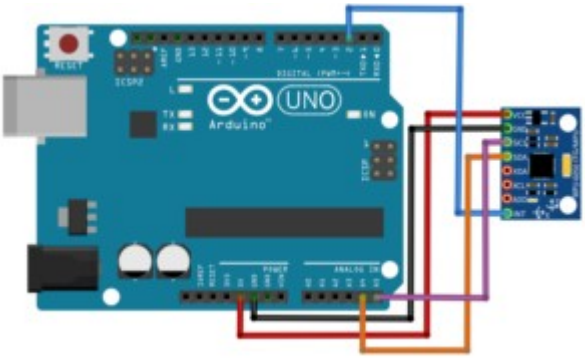
Treća faza je ona najbitnija. Ono što korisnik očekuje kao proizvod. U toj fazi podaci se čitaju kontinuirano u spremnik te već naućena neuronska mreža na izlazu daje prepoznatu gestu ili ništa. Veličina spremnika određena je veličinom ulaznog vektora neuronske mreže.

2. Tehničke značajke

Sada kada znamo što želimo postići pitanje je kako taj cilj i ostvariti. I ovaj put podjelit ćemo projekt na više djelova. No prije toga navedimo neke korištene tehnologije.

Kao mikrokontroler korištena je mikrokontrolerska pločica Arduino Uno. Uno je baziran na Atmega328P mikroprocesoru. Ima 14 digitalnih pinova i 6 analognih. Radi na frekfenciji 16MHz što će biti dovoljno za ovaj projekt. Uno se programira preko USB kabela, a preko njega je također moguće i pružati napajanje te održavati serijsku komunikaciju s računalom. Zbog praktičnosti toga u prototipnoj verziji serijska komunikacija biti će izvedena upravo tako.

Na Arduino spojena je pločica sa InvenSanse-ovim MPU-6050 senzorom. MPU-6050 sadrži MEMS (Mikroelektromehanički sustav) akcelerometar i MEMS žiroskop u jednom ćipu. Svaki kanal ima svoj hardverski 16-bitni analogno-digitalni pretvornik zbog čega očitava sve kanale istovremeno. Za komunikaciju s Arduinom koristi se I^2C protokol.



Slika 2: Povezivanje Arduina i MPU_6050 senzora

Kako bi se olakšala čitljivost koda prvo je izrađena biblioteka MPU_6050.h

Potpuni kod biblioteke dostupan je na:

https://github.com/eugen-vusak/zavrsni-rad/tree/master/Arduino/libraries/MPU_6050

Biblioteka sadrži dvije klase: Data i MPU_6050

Klasa Data je napravljena kako bi opisivala 6torku podataka sa senzora, a koristi ju klasa MPU_6050.

Klasa Data sadrži sljedeće funkcije:

- Data: konstruktor – stvara novu istancu
- void: print() - šalje tekstualni tip preko serijske veze
- String: toString() - stvara string od istance
- Data: add(Data) – zbraja dvije istance
- Data: sub(Data) – oduzima dvije istance
- Data: div(double) – djeli istancu sa double vrijednošću

Istaknuti ću samo najbitniju funkciju iz klase MPU_6050 a to je fukcija measure() koja vraća istancu Data klase.

```
1.Data MPU_6050::measure()
2.{
3.      Wire.beginTransmission(_MPU_adr);
4.      Wire.write(0x3B);  //adresa registra
5.      Wire.endTransmission(false);
6.      Wire.requestFrom(_MPU_adr, 14, true); //MPU_adr zadan konstruktorom
7.      int16_t ax = Wire.read() << 8 | Wire.read();
8.      int16_t ay = Wire.read() << 8 | Wire.read();
9.      int16_t az = Wire.read() << 8 | Wire.read();
10.    int16_t Tmp = Wire.read() << 8 | Wire.read();
11.    int16_t gx = Wire.read() << 8 | Wire.read();
12.    int16_t gy = Wire.read() << 8 | Wire.read();
13.    int16_t gz = Wire.read() << 8 | Wire.read();
14.
15.    Data data(ax,ay,az,gx,gy,gz);
16.    return data;
17.}
```

Također, unutar klase MPU_6050, postoji i funkcija wakeUp() koja "budi" senzor i dovodi ga u stanje spremno za čitanje podataka.

Sada podatke možemo lagano dobiti naredbom Data data = mpu.measure(); te možemo napisati kod koji čita podatke i šalje ih na serijski izlaz. Možemo funkcionalnost proširiti tako da mikrokontroler prvo čeka signal na serijskom ulazu a zatim ovisno o tome koji je signal stigao odradi odgovarajuću akciju.

Arduino kod koji radi ovo dostupan je na poveznici:

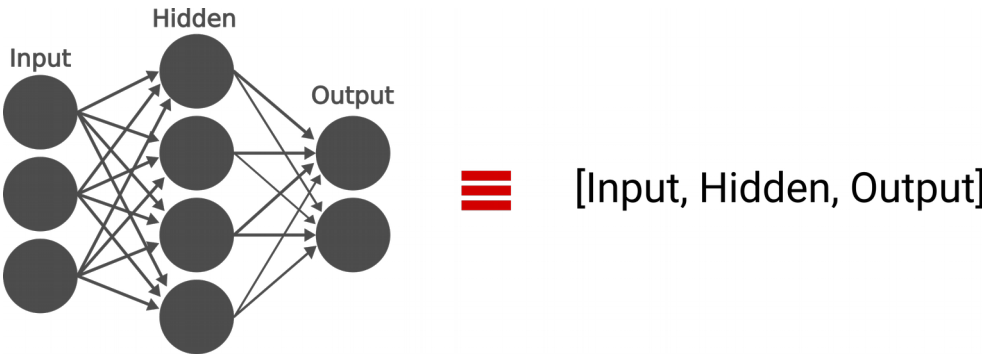
https://github.com/eugen-vusak/zavrsni-rad/blob/master/Arduino/GettingData/GettingData.ino

Ali ovaj kod sam za sebe neće raditi ništa, već će cijelo vrijeme čekati na signal da krene radit. Kako bi ovom kodu osigurali funkcionalnost potrebno je na serijski ulaz od arduina poslati komande, ali i sa njegovog izlaza primati podatke. To je ostvareno u programskom jeziku Processing koji se često koristi za izradu prototipova, a pogotovo uz Arduino mikrokontrolersku pločicu.

Kod u processingu ima dvije zadaće. Prva je da mapira odgovarajuće akcije, u našem slučaju pritisak tipke na tipkovnici, u komande koje zatim šalje na serijska vrata na kojima se nalazi Arduino. Druga zadaća je da svaki put kada na serijska vrata dođe podatak obradi taj podatak i spremi ga u .csv datoteku. Processing kod koji ima opisanu funkcionalnost nalazi se na sljedećoj poveznici:

https://github.com/eugen-vusak/zavrsni-rad/blob/master/Processing/GettingData/GettingData.pde

Sljedeća na redu je neuronska mreža. Ovdje korištena neuronska mreža je osnovna unaprijedna neuronska mreža, što znači da tok podataka ide samo prema naprijed. Neuronsku mrežu odlučio sam opisati kao listu slojeva (engl. Layer).



Slika 3: Neuronska mreža kao lista slojeva

Ideja kod učenja je odrediti koliku pogrešku neuronska mreža daje kao izlaz te tu pogrešku minimizirati sa težinama i pomacima. To je ovdje izvedeno tako da se računa gradijent točnije smijer najbržeg spusta u odnosu na težine (i pomake) i postepeno se pomiče u tom smijeru. Nakon svakog pomaka gradijent se ponovno računa. Gradijent se računa algoritmom propagacije pogreške unatrag a on je zabravo parcijalna derivacija pogreške po težinama (i pomacima)

Prvo neuronsku mrežu modelirajmo matematički. Za to je potrebno definirati par pojmova:

X - ulazni vektor u neuronsku mrežu

$Z^{(l)}$ - vektor ulaza u sloj l

$A^{(l)}$ - vektor izlaza iz sloja l

$W^{(l)}$ - matrica težina (engl. weights) sloja l

$B^{(l)}$ - vektor pomaka (engl. biases) sloja l

$E^{(l)}$ - vektor pogrešaka (engl. errors) sloja l

$\Delta^{(l)}$ - vektor delti sloja l gdje je delta funkcija od E i Z

C - trošak (engl. cost) je suma svih članova vektora pogreške zadnjeg sloja

Y - očekivani izlazni vektor iz neuronske mreže

Sada možemo opisati tri algoritma koji se koriste. To su feedforward algoritam za računanje izlaza, algoritam propagacije unatrag (engl. backpropagation) za računanje gradijenta i gradijentni spust kojim pomičemo težine tako da ukupna pogreška pada.

1. Feedforward

   Prvi sloj je jedinstven te za njega vrijedi

   $$Z^{(1)} = X^{(1)}$$

   dok za ostale čvorove vrijede jednakosti :

   $$Z^{(l)} = W^{(l)} \cdot A^{(l-1)} + B^{(l)} \quad (2)$$
   $$A^{(l)} = f(Z^{(l)}) \quad (3)$$

2. Propagacija pogreške unatrag (backpropagation)

   u ovom slučaju posljednji sloj je jedinstven a za njega vrijedi

   $$E^{(L)} = A^{(L)} - Y \quad (4)$$

   no za posljednji sloj se delta računa kao i za ostale slojeve

   Za ostale slojeve zatim vrijedi:

   $$E^{(l)} = {W^{(l+1)}}^T \cdot \Delta^{(l+1)} \quad (5)$$
   $$\Delta^l = E^l \odot f'(Z^l) \quad (6)$$

3. Gradijentni spust

   Sada kada imamo izrećunate delte za svaki sloj uz pomoć nih možemo izračunati parcijalne derivacije

   $$\frac{\partial C}{\partial W^{(l)}} = \Delta^{(l)} \cdot A^{(l-1)} \quad (7)$$
   $$\frac{\partial C}{\partial B^{(l)}} = \Delta^{(l)} \quad (8)$$

   Sada pomoću tih vrijednosti možemo ažurirati naše težine.

   $$W^{(l)} = W^{(l)} - \mu \cdot \frac{\partial C}{\partial W^{(l)}} \quad (9)$$
   $$W^{(l)} = W^{(l)} - \mu \cdot \frac{\partial C}{\partial W^{(l)}} \quad (10)$$

   Malo grčko slovo mi je hiperparametar koji predstavlja brzinu učenja (engl. learning rate) a odabire se prije početka učenja. On određuje kojom brzinom se pomičemo po krivulji u smijeru gradijenta, točnije koliko su veliki skokovi.

Sada možemo opisati kod koji modelira ovakvu neuronsku mrežu.

Rekli smo već da je neuronska mreža opisana listom slojeva, te da bi smo mogli opisati mrežu potrebno je prvo modelirati sloj. Postoje dva tipa sloja. To su standardni sloj i ulazni sloj. Svaki od tih slojeva modeliran je svojom klasom a nalaze se u paketu layers.py.

Izdvojimo najbitniji dio sloja, a to je funckija output() koja za neki ulazni vektor daje na izlazu novi izlazni vektor po formulama (1), (2) i (3).

klasa Layer

```
1.def output(self, input):
2.    self.Z = np.dot(self.weights, input) + self.biases
3.    self.A = self.function(self)
4.    return self.A
```

ovdje primjećujemo formule (2) u liniji 2 i (3) u liniji 3.

klasa inputLayer

```
1.def output(self, input):
2.  self.Z = input
3.  self.A = self.function(self)
4.  return self.A
```

u klasi InputLayer stvari su dosta slične samo malo jednostavnije. Razlika je da se u liniji dva koristi formula (1) a ne formula (2), no za računanje A također koristimo (3).

Primjetimo da se poziva funkcija function() u oba slučaja. Ta funkcija prima se kroz konstruktor, a i ma oblik:

```
def ime_funkcije(layer, deriv = False){
    if deriv:
        #vrati rezultat derivacije funckije za layer.Z
    #vrati rezultat funkcije za layer.Z
```

Možemo primjetiti da funkcija ne prima samo layer.Z već i cijeli sloj. To je radi bržeg računanja derivacije koja nekad ovisi samo o već prije izračunatom layer.A tako da nije potrebno ponovno provoditi kompleksne kalkulacije. Postoji datoteka sa već unaprijed formiranim funkcijama, ActivationFunctions.py

Sada kada znamo što radi jedan sloj možemo preći na neuronske mreže. Potpuni kod implementirane neuronske mreže nalazi se na linku neural_network.py.Počnimo opet sa feedforward algoritmom. U neuronskoj mreži izvedba tog algoritma svodi se na iteriranje kroz sve slojeve u listi slojeva te spajanje izlaza prošlog sloja na izlaz sljedečeg sloja. Ulaz prvog sloja je ulaz u feedforward() funkciju.

```
def feedforward(self, input):
```

```
1.def feedforward(self, input):
2.  for layer in self.layers:
3.    input = layer.output(input)
4.  return input
```

Algoritam propagacije pogreške unatrag malo je složeniji, zato je najbolje raščlaniti ga po djelovima.

```
1.def backpropagate(self,output, desired_output):
2.  #calculate for the last layer first
3.  output_layer = self.layers[-1]
4.  output_layer.error = output - desired_output
5.
6.
7.  self.E += (output_layer.error**2 / 2).sum()
8.
9.  deriv = np.multiply(output_layer.A, 1-output_layer.A)
10.
11.  output_layer.delta = np.multiply(output_layer.error, deriv)
12.
13.  #from penultimate layer to second layer
14.  for l in reversed(range(1,len(self.layers)-1)):
15.
16.      layer = self.layers[l]
17.
18.      layer_plus_one = self.layers[l+1]
19.      layer.error = np.dot(layer_plus_one.weights.T,layer_plus_one.delta)
20.
21.
22.      deriv = layer.function(layer, deriv = True)
23.
24.      layer.delta = np.multiply(layer.error, deriv)
```

U liniji 4 izračunamo error za poslijednji sloj po (4), zatim po (6) izračunamo deltu za isti taj sloj u liniji 9. Linija 7. računa kvadratnu pogrešku po članovima vektora error za iscrtavanje grafa. Sada za sve slojeve od predzadnjeg do drugog, u liniji 19, računamo error po (5), a u liniji 24 deltu po (6)

Još jedini dio koji nam je ostao implementirati je gradijentni spust. U ovoj izvedbi on malo odstupa od gore navedenih formula jer je omogućen tako zvani stohastički gradijentni spust koji ne ažurira veze nakon svakog ulaznog podatka već akumulira vrijednosti gradijenta kroz nekoliko ulaza (engl. mini batch gradient descent) ili kroz sve (engl. batch gradient descent). Vrijednosti se zatim ažuriraju prosječnom vrijednošću tih akumuliranih vrijednosti.

```
1.def gradientDescent(self):
2.  for l in range(1,len(self.layers)):
3.
4.    layer = self.layers[l]
5.    layer_minus_one = self.layers[l-1]
6.
7.    layer.pC_pW += np.dot(layer.delta, layer_minus_one.A.T)
8.    layer.pC_pB += layer.delta
```

Sada za svaki sloj osim prvoga računamo vrijednosti gradijenta i akumuliramo ih po formulama (7) i (8)

U ovom trenutku spremni smo trenirati našu neuronsku mrežu. Treniranje se vrši po epohama. Jedna epoha je jedna iteracija kroz cijeli skup podataka. U svakoj epohi želimo prolaziti kroz određene djelove ispitnih primjera (engl mini batch) a za svaki testni primjer u njoj želimo izvodit algoritme redom kojih smo ih i opisivali. Pseudokod za takvu mrežu glasio bi ovako

Za i od 0 do epohe

vrati se na početak podataka

dok nismo došli do kraja podataka

miniBatch = sljedeci mini batch

za podatke u miniBatch

feedforward()

backpropagation()

gradient_descent()

ažuriraj podatke

To je u kodu rješeno ovako:

```
1.def train(self, training_data, batch_size = 32, epochs = 200, learning_rate =
0.5):
2.  iter_num = 0
3.  for i in range(epochs):
4.    #print(i)
5.
6.    training_data.resetToBegining()
7.    miniBatch = training_data.getNextMiniBatch(batch_size)
8.
9.    while(miniBatch):
10.
11.      for data in miniBatch:
12.
13.        input, desired_output = data
14.        #1. FeedForwardNeuralNetwork
15.        output = self.feedforward(input)
16.        print("num of iterations", iter_num)
17.        iter_num += 1
18.        #2. backpropagation
19.        self.backpropagate(output, desired_output)
20.        #3. gradientDescent
21.        self.gradientDescent()
22.
23.      #update parametars for each layer
24.      #after every mini batch
25.      for layer in self.layers:
26.        if type(layer) is InputLayer:
27.          continue
28.        layer.updateParametars(batch_size, learning_rate)
29.
30.      miniBatch = training_data.getNextMiniBatch(batch_size)
31.      self.ErrorAxis.append(self.E/batch_size)
32.      self.E = 0
```

3. Upute za korištenje

4. Literatura