Lazar Cvijić

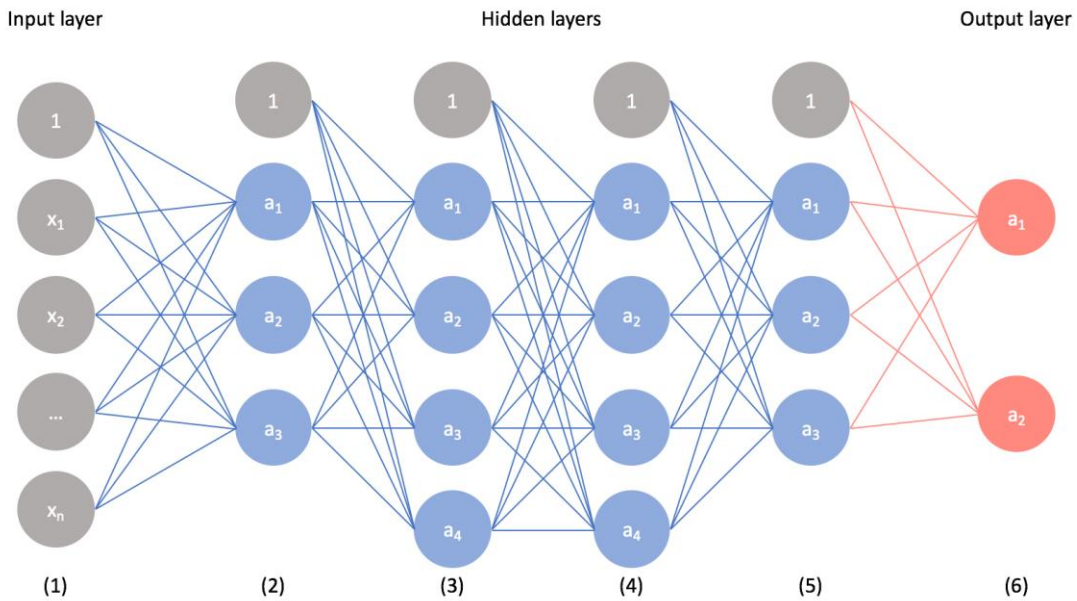
200/2020

Neuroevolution

Using evolutionary algorithms to optimize structure of neural network for a specific machine learning task

Genetski algoritmi

- Genetski algoritmi su algoritmi optimizacije koji taj problem rešavaju inspirsani prironom selekcijom
- Na početku inicijalizujemo populaciju jedinki slučajnih parametara i kroz vrednost njihove fitnes funkcije posmatramo kako se one ponašaju za zadati problem
- U ovom radu predstavljam jedan od načina kako možemo koristiti genetski algoritam da optimizujemo strukturu neuronske mreže kako bismo dobili što precizniju neuronsku mrezu za dati problem
- Ideja je da nam jedinka predstavlja niz celih brojeva gde i-ti broj označava broj neurona u i-tom sloju



Korišćeni podaci

- Podaci korišćeni u radu preuzeti su sa Kaggle-a (https://www.kaggle.com/datasets/rasvob/uci-poker-hand-dataset), PokerHand, koji opisuju karte u pokeru (TexasHoldem) koje igrač ima na raspolaganju (2 u ruci i 3 na talonu) i na osnovu njih naš model treba da kaže koju kombinaciju igrač ima bez eksplicitne naznake pravila igre
- Dakle u pitanju je problem klasifikacije i na raspolaganju imamo 10 klasa, jednu kartu označavaju 2 broja: znak karte (herc, pik, karo, tref), i brojem (1,2,...,10, žandar, dama, kralj), koji su označeni brojevim od 0
- Npr. tref dama je (3, 12)

```
# za treniranje nasih mreza koristicemo PokerHand data set sa Kaggle-a koji predstavlja karte koje neki
# igraci imaju na raspolaganju (2 u ruci 3 na talonu) i na osnovu toga nas zadatak je da odredimo koju
# kombinaciju ima svaki igrac na osnovu trening skupa koji nam implicitno govori pravila
import pandas as pd

# ucitavamo podatke za trening pomocu pandas biblioteke
data = pd.read_csv('treningPoker.csv')
train_set = data.iloc[:,:10]
labels = data.iloc[:,10:]

train_labels_cat = keras.utils.to_categorical(labels)

class_names = ['Nothing','One pair','Two pairs','Three of a kind','Straight','Flush',
               'Full house','Four of a kind','Straight flush','Royal flush']

print(train_set.shape,labels.shape,train_labels_cat.shape)
```

```
(25010, 10) (25010, 1) (25010, 10)
```

0: Nothing in hand; not a recognized poker hand
1: One pair; one pair of equal ranks within five cards
2: Two pairs; two pairs of equal ranks within five cards
3: Three of a kind; three equal ranks within five cards
4: Straight; five cards, sequentially ranked with no gaps
5: Flush; five cards with the same suit
6: Full house; pair + different rank three of a kind
7: Four of a kind; four equal ranks within five cards
8: Straight flush; straight + flush
9: Royal flush; {Ace, King, Queen, Jack, Ten} + flush

Opis klase Individual

- U našem GA jedinke će biti predstavljene kao objekti klase Individual, koja u konstruktoru prima granice za mogući broj slojeva i granice za mogući broj neurona u svakom sloju, kao i oblik ulaznih podataka
- Model se kreira funkcijom create_model koja na osnovu broja slojeva u svakoj mreži i broja neurona u svakom sloju kreira sekvencijalni model potpuno povezane neuronske mreže i kompajluje ga određenim optimizatorom, loss funkcijom i metrikama koje će se pratiti tokom treniranja
- calc_acc će nakon obavljenog treninga vratiti preciznost modela ('accuracy') u poslednjoj epohi

```python
# objekti klase Individual ce bit nase jedinke u GA
class Individual:
    def __init__(self,minLayers,maxLayers,minNodes,maxNodes,input_shape):
        self.input_shape = input_shape
        self.numLayers = random.randrange(minLayers,maxLayers)
        self.layers = [random.randrange(minNodes,maxNodes) for _ in range(self.numLayers)]
        self.model = self.create_model()
        self.acc = self.calc_acc()

    def create_model(self):
        modelList = [keras.layers.Flatten(input_shape=self.input_shape)]

        for i in range(self.numLayers):
            modelList.append(keras.layers.Dense(self.layers[i], activation='relu'))

        modelList.append(keras.layers.Dense(10, activation='softmax'))

        model = Sequential(modelList)
        model.compile(optimizer='adam',loss=tf.keras.losses.CategoricalCrossentropy(),metrics=['accuracy'])


        return model

    # fitnes funkcija neseg GA ce biti 'accuracy' posednje epohe treninga
    def calc_acc(self):
        history = self.model.fit(train_set, train_labels_cat, epochs=15)
        return history.history['accuracy'][-1]

    def __lt__(self,other):
        return self.acc < other.acc
```

Selekcija

- Kao što je već navedeno fitnes funkcija našeg genetskog algoritma je preciznost modela u poslednjoj epohi treninga
- Koristimo turnirsku selekciju kako bismo od slucajno izabranog uzorka jedinki populacije izabrali najbolju koja će se u daljim koracima ukrstiti i ostaviti potomstvo

```python
def selection(population, tour_size):
    # odredjujemo koje ce se jedinke takmiciti da ostave potomstvo pomocu turnirske selekcije
    competitors_idx = random.sample(range(len(population)), tour_size)
    max_acc = -1.0
    max_idx = -1
    for i in range(tour_size):
        if population[competitors_idx[i]].acc > max_acc:
            max_acc = population[competitors_idx[i]].acc
            max_idx = competitors_idx[i]
    return max_idx
```

Ukrštanje (crossover)

- Na jednostavan način ukrštamo dve jedinke koristeći jednopoziciono ukrštanje
- Izaberemo jedan indeks tj. crossing-point gde ćemo podeliti jednu od jedinki i decu dobijamo tako što spajamo jedan isečak prvog i drugi isečak drugog roditelja, slično i za drugo dete
- crossing-point biramo tako što izaberemo slučajni indeks jednike sa manjem brojem slojeva

```
1  def crossover(individual_1, individual_2, child_1, child_2):
2      # ukrstamo dve jedinke koristeci jednopoziciono ukrstanje
3      crossing_point = random.randrange(0,min(individual_1.numLayers,individual_2.numLayers))
4
5      child_1.layers[:crossing_point] = individual_1.layers[:crossing_point]
6      child_1.layers[crossing_point:] = individual_2.layers[crossing_point:]
7      child_1.numLayers = len(child_1.layers)
8
9      child_2.layers[:crossing_point] = individual_2.layers[:crossing_point]
10     child_2.layers[crossing_point:] = individual_1.layers[crossing_point:]
11     child_2.numLayers = len(child_2.layers)
12
```

Mutacija

» Mutaciju obavljamo tako što oduzmemo jedan neuron sloju koji je izabran sa određenom verovatnoćom

```
1  def mutation(individual,mutation_prob):
2      # mutiramo tj. po odredjenoj verovatnoci menjamo svaki sloj jedne jedinke
3      for i in range(len(individual.layers)):
4          if random.random() < mutation_prob:
5              print("MUTACIJA")
6              individual.layers[i] -= 1
```

Opis genetskog algoritma

- Na početku inicijalizujemo populaciju slušajno generisanih jedinki u granicama koje su date kao parametri
- Uvodimo elitizam u naš algoritam što znači da će određen procenat jedinki automatcki ići u novu generaciju
  - To radimo tako što sortiramo populaciju po fitnes funkciji u nerastućem poretku i uzimamo određen procenat jedinki
- Dalje, ostatak populacije prolazi kroz selekciju, ali biramo sve dok ne izaberemo dva različita roditelja

```python
def ga(pop_size,num_iters,elitism,mutation_prob,tour_size,minLayers,maxLayers,minNodes,maxNodes,input_shape):

    # inicijalizujemo popuaciju po datim parametrima
    population = [Individual(minLayers,maxLayers,minNodes,maxNodes,input_shape) for _ in range(pop_size)]

    new_population = population[:]

    # ovde cemo cuvati najbolje jedinke u svakoj generaciji
    best_in_gens = []

    for i in range(num_iters):
        print()
        print(f"iter_{i}")
        print()
        print([(population[i].numLayers, population[i].layers) for i in range(pop_size)])

        # uvodimo eltizam tj. odredjen procenat najboljih jedinki prezivi svaku generaciju tj. ostavi potomstvo
        population.sort(key=lambda x: x.acc, reverse=True)
        new_population[:elitism] = population[:elitism]

        for i in range(elitism,pop_size,2):

            parent1_idx = selection(population, tour_size)
            parent2_idx = selection(population, tour_size)

            while parent1_idx == parent2_idx:
                parent2_idx = selection(population, tour_size)

            # ako su im roditelji isti nema potrebe za crossoverom
            if population[parent1_idx].layers == population[parent2_idx].layers:
                new_population[i] = population[parent1_idx]
                new_population[i+1] = population[parent1_idx]
                continue

            crossover(population[parent1_idx], population[parent2_idx], new_population[i], new_population[i+1])

            mutation(new_population[i], mutation_prob)
            mutation(new_population[i+1], mutation_prob)

            # ispisujemo novodobijene ukrstene jedinke
            print(new_population[i].layers, new_population[i+1].layers)
            new_population[i].model = new_population[i].create_model()
            new_population[i+1].model = new_population[i+1].create_model()

            new_population[i].acc = new_population[i].calc_acc()
            new_population[i+1].acc = new_population[i+1].calc_acc()

        population[:] = new_population[:]
        best_in_gens.append(max(population))
    return best_in_gens

```

Opis genetskog algoritma

- Jedna optimizacija je ta da ako su roditelji jednaki, a drugog indeksa u populaciji, preskačemo ukrštanje kako ne bismo ponovo morali da treniramo neuronsku mrežu što je jako zahtevno za računar
- Sledeći korak je ukrštanje nakon čega dobijamo nove potencijalne članove nove populacije
- Dalje ti članovi prolaze kroz mutaciju nakon čega je potrebno da se modeli ponovo istreniraju nakon promenjene strukture
- Na kraju populacija dobija vrednost nove populacije i u niz koji kasnije vraćamo kao resenje dodajemo najbolju jedinku ove generacije

```python
def ga(pop_size,num_iters,elitism,mutation_prob,tour_size,minLayers,maxLayers,minNodes,maxNodes,input_shape):

    # inicijalizujemo popuaciju po datim parametrima
    population = [Individual(minLayers,maxLayers,minNodes,maxNodes,input_shape) for _ in range(pop_size)]

    new_population = population[:]

    # ovde cemo cuvati najbolje jedinke u svakoj generaciji
    best_in_gens = []

    for i in range(num_iters):
        print()
        print(f"iter_{i}")
        print()
        print([(population[i].numLayers, population[i].layers) for i in range(pop_size)])

        # uvodimo eltizam tj. odredjen procenat najboljih jedinki prezivi svaku generaciju tj. ostavi potomstvo
        population.sort(key=lambda x: x.acc, reverse=True)
        new_population[:elitism] = population[:elitism]

        for i in range(elitism,pop_size,2):

            parent1_idx = selection(population, tour_size)
            parent2_idx = selection(population, tour_size)

            while parent1_idx == parent2_idx:
                parent2_idx = selection(population, tour_size)

            # ako su im roditelji isti nema potrebe za crossoverom
            if population[parent1_idx].layers == population[parent2_idx].layers:
                new_population[i] = population[parent1_idx]
                new_population[i+1] = population[parent1_idx]
                continue

            crossover(population[parent1_idx], population[parent2_idx], new_population[i], new_population[i+1])

            mutation(new_population[i], mutation_prob)
            mutation(new_population[i+1], mutation_prob)

            # ispisujemo novodobijene ukrstene jedinke
            print(new_population[i].layers, new_population[i+1].layers)
            new_population[i].model = new_population[i].create_model()
            new_population[i+1].model = new_population[i+1].create_model()

            new_population[i].acc = new_population[i].calc_acc()
            new_population[i+1].acc = new_population[i+1].calc_acc()

        population[:] = new_population[:]
        best_in_gens.append(max(population))
    return best_in_gens

```

Rezultati

» Nakon pokretanja algoritma za sledeće parametre:

```
best_individuals = ga(pop_size=32,num_iters=5,elitism=2,mutation_prob=0.05,tour_size=8,
                      minLayers=4,maxLayers=10,minNodes=10,maxNodes=32,input_shape=(10,))
```

» Dobijamo napredak preciznosti od otprilike 10% za isti broj epoha treninga , kao i sličan napredak pri evaluaciji identičnog test skupa

```
Na pocetku najbolji: [25, 25, 29, 12, 16] 0.5716913342475891
Na kraju najbolji: [25, 26, 29, 12, 24, 28, 28, 18] 0.6636945009231567
(1000000, 10) (1000000, 1) (1000000, 10)
2023-09-28 13:14:56.837369: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 40000000 exceeds 10%
of free system memory.
2023-09-28 13:14:57.247325: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 80000000 exceeds 10%
of free system memory.
31250/31250 - 55s - loss: 0.9248 - accuracy: 0.5638 - 55s/epoch - 2ms/step
konacno_prvi:  0.5638369917869568
2023-09-28 13:15:52.912696: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 40000000 exceeds 10%
of free system memory.
2023-09-28 13:15:53.157435: W tensorflow/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 80000000 exceeds 10%
of free system memory.
31250/31250 - 58s - loss: 0.7871 - accuracy: 0.6568 - 58s/epoch - 2ms/step
konacno_poslednji:  0.6568220257759094
```

KRAJ

HVALA NA PAŽNJI