Umelá inteligencia

Umelé neurónové siete

Zadanie č.3b

Patrícia Sojčáková

ID: 127273

xsojcakova@stuba.sk

Cvičenie: Štvrtok 12:00

Cvičiaci: Ing. Jakub Abrahoim

Slovenská technická univerzita

Fakulta informatiky a informačných technológií

Ilkovičova 3, 842 16 Bratislava

Obsah

Úvod

California Housing regresný model

1. Úloha

   Úlohou bolo vytvoriť neurónovú sieť na riešenie regresného problému pomocou dátovej množiny o bývaní v Kalifornii. Cieľom bolo predpovedať strednú cenu domu pre jednotlivé okresy Kalifornie na základe rôznych charakteristík, ako sú údaje o obyvateľstve, príjme, geografická poloha a ďalšie. Dátová množina obsahuje 20 640 záznamov s 8 atribútmi, medzi ktoré patrí napríklad stredný príjem, vek budov, priemerná obsadenosť, a cieľovou premennou je medián ceny domu. Táto úloha demonštruje použitie neurónových sietí pre regresné úlohy, pri ktorých je cieľom predikcia spojitej hodnoty.

2. Zadanie

   Riešenie úlohy je pomocou doprednej neurónovej siete (viacvrstvového perceptrónu) a implementáciou sú rôzne modeli s cieľom nájsť čo najmenší model, ktorý dosahuje prijateľné výsledky. Na trénovanie siete sú využité optimalizačné algoritmy SGD, SGD s momentom a ADAM. Chyby boli vyhodnotené na trénovacej aj testovacej množine. Pri implementácii je použitá knižnica PyTorch.

Backpropagation algoritmus

1. Úloha

   V tejto úlohe bolo potrebné implementovať kompletný algoritmus spätného šírenia chyby (backpropagation), ktorý je základom tréningu neurónových sietí. Tento algoritmus umožňuje optimalizáciu siete minimalizáciou definovanej chybovej funkcie. Bolo nutné implementovať doprednú a spätnú fázu pre jednotlivé operácie a funkcie siete, vrátane aktualizácie parametrov. Správnosť je overená trénovaním jednoduchej doprednej neurónovej siete (viacvrstvového perceptrónu).

2. Zadanie

   Pri riešení tejto úlohy je použitá knižnica NumPy na realizáciu maticových a vektorových operácií. Použitie knižníc ako PyTorch alebo TensorFlow) nie je povolené. Bola vytvorená modulárna architektúra, ktorá umožňuje jednoduché reťazenie jednotlivých komponentov. Implementácia zahŕňa lineárnu vrstvu, aktivačné funkcie sigmoid, tanh, ReLU a chybovú funkciu strednej kvadratickej chyby (MSE - mean squared error).

California Housing regresný model

Moju doprednú neurónovú sieť (viacvrstvový perceptron) tvorí vstup 13 hodnôt, 2 skryté vrstvy a 1 výstup.

Úprava dát

Daný dataset, z ktorého sú používané údaje je priamo priložený ku programu. Dataset originálne obsahuje 10 typov údajov avšak nie všetky sú vstupnými dátami alebo sú v nesprávnom tvare. Práve preto musí dôjsť k normalizácii dát.

1. One-hot encoding

   V datasete sa nachádza položka „ocean proximity“, ktorá nemá číselnú hodnotu. Preto z tejto položky zoberieme typy hodnôt, pre ktoré vytvoríme nové kolónky. Týmito novými stĺpcami nahradíme pôvodný. Ak v pôvodnom datasete mala položka v stĺpci „ocean proximity“ hodnotu „NEAR BY“ tak teraz má v stĺpci „NEAR BY“ hodnotu 1 a v ostatných 0.

```
15  kolonka = pd.get_dummies(data["ocean_proximity"]).astype(int)
16  data = data.join(kolonka)
17  data = data.drop( labels: ["ocean_proximity"], axis=1)
```

2. Výstup

   Výstupnou alebo kontrolnou hodnotou má byť stĺpec s názvom „median house value“. Tento stĺpec treba teda samostatne oddeliť od ostatných údajov.

```
19  input_data = data.drop( labels: ["median_house_value"], axis=1)
20  output_data = data["median_house_value"]
```

3. Normalizácia dát

   Vstupy a výstupy sa škálujú odlišne, pretože majú rôzne charakteristiky a požiadavky na spracovanie v modeli. Vstupné hodnoty sa škálujú tak, že každý stĺpec má priemer 0 a štandardnú odchýlku 1. Je to aj takzvané Gausove rozloženie. Toto nám zaručí aby sa model príliš dlho neučil. Výstup je lepšie škálovať do konkrétneho rozsahu napr. [-1,1], aby sa predikcia obmedzila na rozsah. Takto sa teda bude očakávať, že niečo leží v určitom intervale, čo je lepšie pre regresné úlohy.

```
23  skala = StandardScaler()
24  input_data = skala.fit_transform(input_data)
25
26  scaler_output = MinMaxScaler()
27  output_data = scaler_output.fit_transform(output_data.values.reshape(-1, 1))
```

4. Rozdelenie na trénovacie a testovacie dáta

   Trénovacie a testovacie dáta sú rozdelené v pomere 4:1 čiže 80% dát je trénovacích a 20% testovacích.

```
30  train_input, test_input, train_output, test_output = train_test_split( *arrays: input_data, output_data, test_size=0.2)
```

5. PyTorch

Aby sa mi pracovalo najľachšie pomocou Pytorch, dáta si dopredu upravím na základnú dátovú štruktúru. Používam tenzory a 32-bitové desatinné čísla. Pre prácu s Pytorch treba aj skombinovať množinu trénovacieho vstupu a výstupu a testovacieho vstupu a výstupu do datasetov. Pomocou dataloader si dáta rozdelím na batch-e.

```
33  # Prevod na tenzory
34  torch_train_input = torch.tensor(train_input).type(torch.float32)
35  torch_train_output = torch.tensor(train_output).type(torch.float32)
36
37  torch_test_input = torch.tensor(test_input).type(torch.float32)
38  torch_test_output = torch.tensor(test_output).type(torch.float32)
39
40  # Dataset a DataLoader
41  train_dataset = TensorDataset( *tensors: torch_train_input, torch_train_output)
42  train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
43
44  test_dataset = TensorDataset( *tensors: torch_test_input, torch_test_output)
45  test_loader = DataLoader(test_dataset, batch_size=128)
```

Spustenie modelu

1. Algoritmy

Neurónová sieť používa ako optimalizéry ADAM, SGD a SGD s momentom. Pre ADAM je rýchlosť učenia lepšia v menších číslach ako napríklad 0,001 avšak pre SGD je lepší väčší ako napríklad 0,1. Momentum sa osvedčilo 0,9. Presný typ algoritmu si môže používateľ vybrať.

```
140 a = int(input("1 - Adam\t2 - SGD bez momentum\t3 - SGD s momentum\nZadaj:"))
141 if a == 1:
142     print("------------------------------ ADAM ------------------------------")
143     optimal = torch.optim.Adam(model.parameters(), lr=0.001)
144     spustenie()
145 elif a == 2:
146     print("------------------ SGD bez momentum -------------------")
147     optimal = torch.optim.SGD(model.parameters(), lr=0.1)
148     spustenie()
149 elif a == 3:
150     print("------------------- SGD s momentum --------------------")
151     optimal = torch.optim.SGD(model.parameters(), lr=0.1, momentum=0.9)
152     spustenie()
```

2. Tréning a testovanie

Funkcia spustenie() vykonáva tréning a testovanie modelu pomocou optimalizácie a hodnotenia chyby. Začína definovaním funkcie straty a prázdnymi zoznamami na uchovávanie chýb počas tréningu a testovania. Počas tréningu model prechádza 200 epochami. V každej epoche sa model trénuje na dávkach dát, kde sa vypočíta chyba medzi predikciou modelu a skutočnými hodnotami. Gradieny sú vypočítané cez spätnú propagáciu a váhy sa aktualizujú pomocou optimalizátora.

Po každej epoche sa vypočíta priemerná chyba pre trénovacie a testovacie dáta a ukladá sa do zoznamov. Každých 10 epoch sa vypíše aktuálna hodnota chyby pre trénovanie a testovanie.

Po tréningu sa model prepne do hodnotiaceho režimu, kde sa hodnotí výkon modelu na testovacích dátach bez aktualizácie váh.

Na konci funkcie sa vypočíta RMSE pre testovacie dáta. Predpovede a skutočné hodnoty sa denormalizujú späť na pôvodný rozsah, aby sa získala presná interpretácia. Percentuálna chyba sa vypočíta medzi predpovedanými a skutočnými hodnotami, a získa sa presnosť modelu. Na záver sa vykreslí graf s priebehom trénovacej a testovacej chyby počas jednotlivých epoch, čo umožňuje vizualizovať, ako model zlepšuje svoje predikcie.

```
def spustenie():  3 usages
    mseloss = nn.MSELoss()
    train_chyba = []
    test_chyba = []
    # Tréning
    model.train()
    for epocha in range(200):
        priemer_chyba = 0
        for x_batch, y_batch in train_loader:
            optimal.zero_grad()
            output = model(x_batch)
            chyba = mseloss(output, y_batch)
            chyba.backward()
            optimal.step()
            priemer_chyba += chyba.item()

        priemer_chyba /= len(train_loader)
        train_chyba.append(priemer_chyba)

        model.eval()  # Prepnutie do evaluačného režimu
        priemer_chyba = 0
        with torch.no_grad():
            for x_batch, y_batch in test_loader:
                output = model(x_batch)
                chyba = mseloss(output, y_batch)
                priemer_chyba += chyba.item()

        priemer_chyba /= len(test_loader)
        test_chyba.append(priemer_chyba)

        if epocha % 10 == 0 and epocha != 0:
            print(
                f"Epocha {epocha}: Chyba pri trenovani = {train_chyba[-1]:.4f}, Chyba pri testovani = {test_chyba[-1]:.4f}")
    # Testovanie
    model.eval()  # Prepnutie do evaluačného režimu
    cela_chyba = 0
    predikcie_output = []
    good_output = []

    with torch.no_grad():
        for x_batch, y_batch in test_loader:
            output = model(x_batch)
            chyba = mseloss(output, y_batch)
            cela_chyba += chyba.item()
            predikcie_output.extend(output.numpy().flatten())
            good_output.extend(y_batch.numpy().flatten())

    # Výpočet RMSE
    cela_chyba /= len(test_loader)
    rmse = torch.sqrt(torch.tensor(cela_chyba))

    # Denormalizácia predpovedí a skutočných hodnôt na pôvodný rozsah
    predikcie_output = scaler_output.inverse_transform(np.array(predikcie_output).reshape(-1, 1)).flatten()
    good_output = scaler_output.inverse_transform(np.array(good_output).reshape(-1, 1)).flatten()

    # Výpočet percentuálnej chyby
    error = np.abs((good_output - predikcie_output) / good_output) * 100
    accuracy = 100 - np.mean(error)

    print("Test RMSE:", rmse.item())
    print(f"Uspesnost: {accuracy:.2f}%")
```

Testovanie

	Percentuálne úspešnosti		
Veľkosti vrstiev	ADAM	SGD	SGD s m. (0,9)
128 > 64 >1	74.56%	79.19%	77.90%
64 > 64 > 1	79.97%	77.88%	74.56%
64 > 32 > 1	80.80%	79.32%	79.68%
32 > 32 > 1	81.23%	76.69%	78.81%

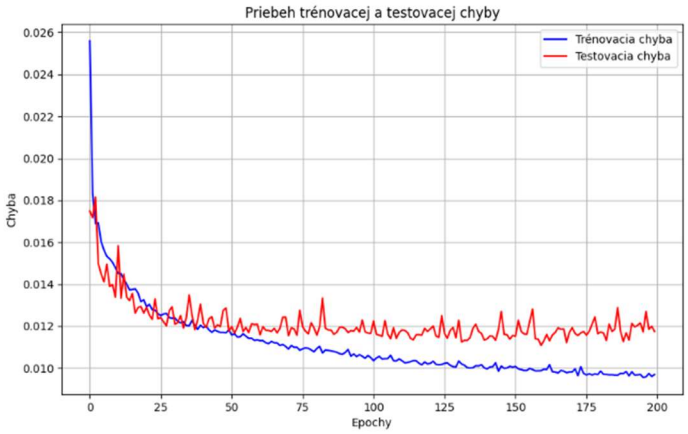
	Percentuálne úspešnosti		
Batch	ADAM	SGD	SGD s m. (0,9)
128	80.19%	75.25%	75.00%
64	80.54%	76.49%	76.97%
32	76.79%	76.00%	77.29%

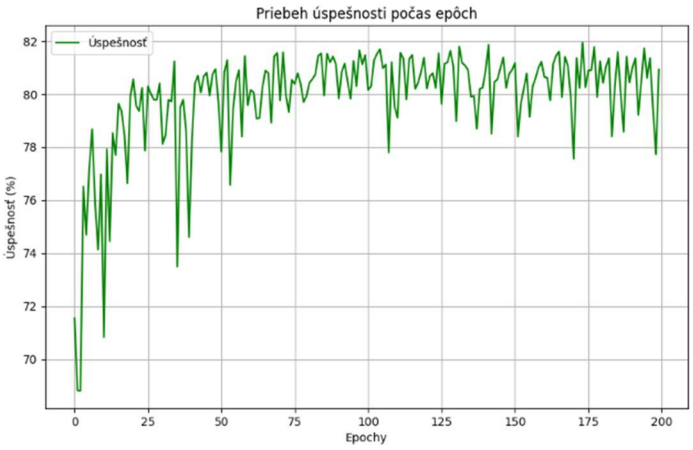
	Percentuálne úspešnosti		
Learning rate	ADAM	SGD	SGD s m. (0,9)
0,1	72.01%	73.96%	80.99%
0,01	80.00%	75.02%	78.36%
0,001	77.57%	72.84%	76.05%

	Percentuálne úspešnosti		
Aktivačné funkcie	ADAM	SGD	SGD s m. (0,9)
sig + sig	79.46%	70.84%	73.18%
sig + tan	80.21%	70.48%	72.76%
sig + relu	82.19%	75.77%	74.11%
tan + sig	79.93%	74.85%	75.20%
tan + tan	79.75%	75.74%	75.47%
tan + relu	77.77%	77.67%	77.49%
relu + sig	79.85%	74.94%	73.31%
relu + tan	78.06%	76.81%	76.69%
relu + relu	79.83%	77.39%	77.72%

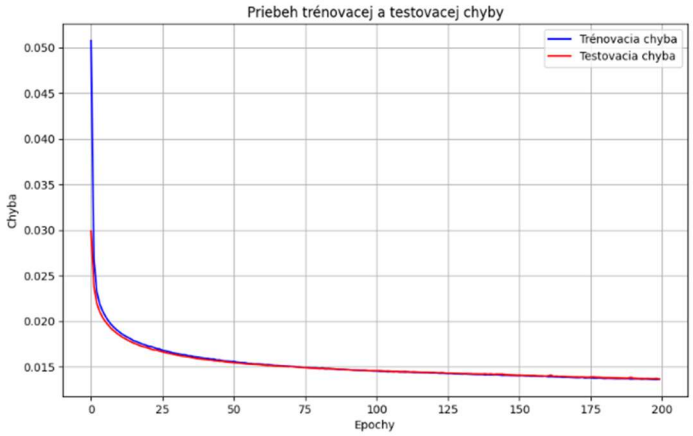
Potenciálne najlepšie výsledky

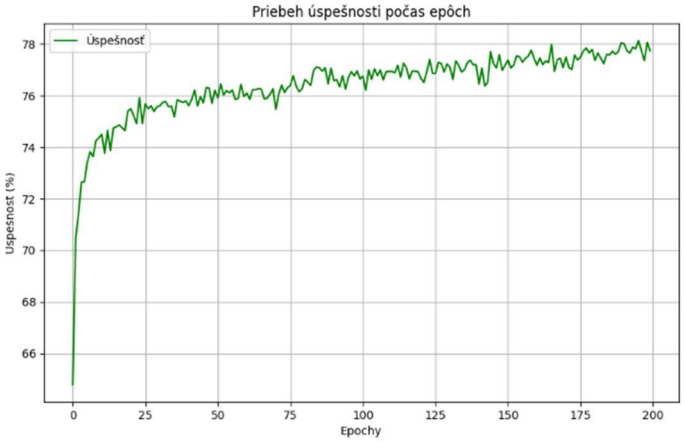
ADAM



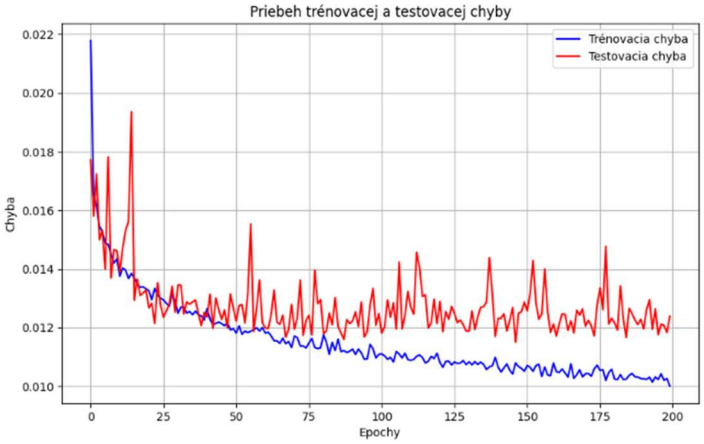


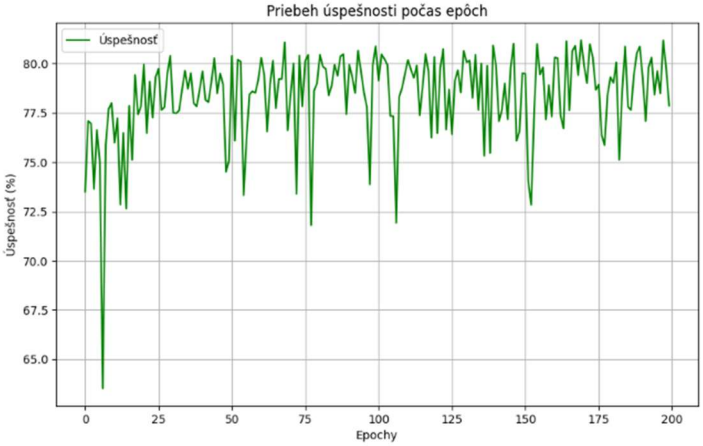
SGD





SGD s momentom





Backpropagation algoritmus

Dopredná neurónová sieť alebo viac vrstvový perceptrón sa skladajú zo vstupu, skrytých vrstiev a výstupu. Moje riešenie pozostáva z 2 vstupov, 1 skrytej vrstvy (poprípade 2) a 1 výstupu.

Vstupy

Riešenie je závislé na vstupoch. Môj program dokáže riešiť problémy AND, OR a XOR. Potrebné hodnoty sa dajú nastaviť v premennej „input“. Následne v premennej „good“ je potrebné nastaviť správnu výstupnú hodnotu, ktorú sa model ďalej naučí. Aktuálne je program pripravený na riešenie problému XOR.

```
78    input = np.array([[0,0],[0,1],[1,0],[1,1]])
79    good = np.array([[0],[1],[1],[0]])
80    model = model()
```

Skryté vrstvy

V trénovaní sa striedajú 4 trénovanie dvojice. Trénuje sa ako prvé s momentom a následne bez momentu. Spustenie trénovania zabezpečuje funkcia spustenie(). Model trénuje s každým vstupom 500 epoch. Ako prvé model prejde skrytú vrstvu dopredu a následne vypočíta chybu, ktorá sa ukladá pre neskoršie vykreslenie grafu. Pre lepší prehľad trénovania sa každých 25 epoch vypíše chyba. Deriváciou chyby je gradient, pomocou ktorého sa pri spätnom chode upravia váhy a biasy. Ďalej prebieha chod dozadu cez vrstvy, kde prebieha aktualizácia váh a biasov.

```
152   print("------------------- S momentom -------------------")
153   moment = True
154   moment_rate = 0.9
155   model = model()
156   spustenie()
157   print("")
158
159   print("------------------- Bez momentu -------------------")
160   moment = False
161   spustenie()
```

```
91    def spustenie():  2 usages
92        training_errors = []
93        p = 0
94        chyba = 0
95        for i in range(2000):
96            if p == 4:
97                p = 0
98
99            a = model.dopredu(input[p:p + 1])
100
101           # chyba
102           if i % 4 == 0:
103               chyba = np.mean((a - good[p]) ** 2)
104               training_errors.append(chyba)
105           if i % 100 == 0 and i != 0:
106               chyba = np.mean((a - good[p]) ** 2)
107               print(f"Chyba po {int(i / 4)} epochách:", chyba)
108
109           grad = 2 * (a - good[p])
110
111           model.dozadu(grad)
112           p += 1
113       c = 0
```

```
115     print("")
116     vysledok = model.dopredu([0, 0])
117     print(f"Vstup: 0,0\tVýstup: {np.round(vysledok).astype(int)}")
118     if np.round(vysledok).astype(int) == 0:
119         c += 1
120
121     vysledok = model.dopredu([0, 1])
122     print(f"Vstup: 0,1\tVýstup: {np.round(vysledok).astype(int)}")
123     if np.round(vysledok).astype(int) == 1:
124         c += 1
125
126     vysledok = model.dopredu([1, 0])
127     print(f"Vstup: 1,0\tVýstup: {np.round(vysledok).astype(int)}")
128     if np.round(vysledok).astype(int) == 1:
129         c += 1
130
131     vysledok = model.dopredu([1, 1])
132     print(f"Vstup: 1,1\tVýstup: {np.round(vysledok).astype(int)}")
133     if np.round(vysledok).astype(int) == 0:
134         c += 1
135
136     print(f"Správnosť výsledku: {c / 4 * 100}%")
137
138     plt.plot(training_errors)
139     plt.title('Priebeh tréningu')
140     plt.xlabel('Iterácie')
141     plt.ylabel('Chyba')
142     plt.grid(True)
143     plt.show()
```

1. Class model

Trieda model má definované štyri premenné a dve metódy. Premenné „prva_vrstva“ a „druha_vrstva“ vytvárajú objekty triedy „linearna_vrstva“. Skryté vrstvy sú následne aktivované aktivačnými funkciami, ktoré sú reprezentované premennými „prva_aktivacia“ a „druha_aktivacia“.

Metódy tejto triedy sú „dopredu“ a „dozadu“. Metóda dopredu(self, input) zabezpečuje, že vstupy prechádzajú cez neurónovú sieť dopredu. Vstupy sa spracovávajú v striedaní medzi lineárnymi vrstvami a aktivačnými funkciami. Metóda vracia predikciu výstupu modelu. Naopak, metóda dozadu(self, grad) sa stará o spätnú propagáciu chyby a aktualizáciu váh v modeli. Správne riadi poradie vrstiev počas spätného chodu.

Aktuálne je program nastavený tak aby sa vykonávala len 1 skrytá vrstva (lineárna – aktivačná – lineárna – aktivačná, ktorá je sigmoid) avšak dá sa to zmeniť zakomentovanými riadkami. Pri binárnej klasifikácii sa odporúča vo výstupnej vrstve použiť sigmoid, ktorý dáva výsledky 0/1.

```
class model:  7 usages
    def __init__(self):
        self.prva_vrstva = linearna_vrstva( x: 2,  y: 4)
        self.prva_aktivacia = tanh()
        #self.druha_vrstva = linearna_vrstva(4, 4)
        #self.druha_aktivacia = sigmoid()
        self.output_vrstva = linearna_vrstva( x: 4,  y: 1)
        self.output_aktivacia = sigmoid()

    def dopredu(self, input):  5 usages
        n1 = self.prva_vrstva.dopredu(input)
        n2 = self.prva_aktivacia.dopredu(n1)
        #n3 = self.druha_vrstva.dopredu(n2)
        #n4 = self.druha_aktivacia.dopredu(n3)
        o1 = self.output_vrstva.dopredu(n2)      #n4
        o2 = self.output_aktivacia.dopredu(o1)
        return o2

    def dozadu(self, grad):  1 usage
        o1 = self.output_aktivacia.dozadu(grad)
        o2 = self.output_vrstva.dozadu(o1)
        #s3 = self.druha_aktivacia.dozadu(o2)
        #s2 = self.druha_vrstva.dozadu(s3)
        s1 = self.prva_aktivacia.dozadu(o2)      #s2
        self.prva_vrstva.dozadu(s1)
```

2. Class linearna_vrstva

Trieda linearna_vrstva má definované 3 premenné a 2 metódy. Lineárnu vrstvu tvoria váhy, biasy a premennú „input“ na uchovanie vstupu. Váhy a biasy sú na začiatku zvolené náhodne ale počas učenia modelu sa vylepšujú. Lineárna vrstva si vstup preto, aby mohol byť využitý pri spätnom šírení chyby (backpropagation). Tento program podporuje aj techniku zrýchľovania učenia, momentum. Ak je momentum aktivované vytvoria sa 2 dodatočné premenné, ktoré uchovávajú vážené priemery gradientov váh a biasov.

Dopredný chod zabezpečuje metóda dopredu(self, i). Práve tu sa uloží vstup a taktiež sa tento vstup prenásobí váhami a pripočítajú sa biasy. Takto vzniká výstup pre ďalšiu vrstvu. Funkcia dozadu(self, grad) implementuje spätné šírenie chyby (backpropagation) pre lineárnu vrstvu. Vypočíta gradienty váh a biasov na základe gradientu chyby a uloženého vstupu. a pomocou učebnej rýchlosti. Následne sa váhy a biasy aktualizujú, pričom sa zohľadňuje učebná rýchlosť. Taktiež táto metóda vracia gradient chyby pre predchádzajúcu vrstvu. Ak je zapnuté momentum, používa sa technika zrýchlenia učenia, ktorá kombinuje aktuálne gradienty s predchádzajúcimi, čo vedie k hladšej a rýchlejšej optimalizácii.

```python
class linearna_vrstva:  2 usages
    def __init__(self, x, y):           # x-pocet vstupov; y-pocet vystupov
        self.w = np.random.randn(x, y)
        self.b = np.random.randn(1, y)
        self.input = None
        if moment:
            self.moment = moment_rate
            self.vW = np.zeros_like(self.w)
            self.vB = np.zeros_like(self.b)


    def dopredu(self, i):  2 usages
        self.input = i
        return np.dot(i,self.w) + self.b

    def dozadu(self, g, learning_rate=0.1):  2 usages
        gw = np.dot(self.input.T, g)
        gb = np.sum(g, axis=0, keepdims=True)

        # aktualizacia
        if moment:
            self.vW = self.moment * self.vW + (1 - self.moment) * gw
            self.vB = self.moment * self.vB + (1 - self.moment) * gb
            self.w -= learning_rate * self.vW
            self.b -= learning_rate * self.vB
        else:
            self.w -= learning_rate * gw
            self.b -= learning_rate * gb
        return np.dot(g, self.w.T)
```

3. Class sigmoid

Táto trieda implementuje sigmoidnú aktivačnú funkciu, ktorá mapuje vstupy. V doprednej fáze (dopredu) vypočíta sigmoidnú hodnotu a uloží ju do atribútu „self.output“ pre neskoršie použitie. Pri spätnom šírení (dozadu) využíva uložený výstup na výpočet gradientu pre aktualizáciu váh v predchádzajúcej vrstve.

```python
class sigmoid:
    def __init__(self):
        self.output = None

    def dopredu(self, v):
        self.output = 1/(1+np.exp(-v))
        return self.output

    def dozadu(self, g):
        return g * self.output * (1-self.output)
```

4. Class tanh

Trieda tanh predstavuje hyperbolickú tangensovú aktivačnú funkciu, ktorá transformuje vstupy na hodnoty v intervale [−1,1]. Metóda dopredu vypočíta výstup funkcie a uloží ho na neskoršie použitie. V rámci spätnej fázy (dozadu) využíva deriváciu na výpočet gradientu, ktorý sa následne kombinuje s prichádzajúcim gradientom pre aktualizáciu váh.

```
57  class tanh:  2 usages
58      def __init__(self):
59          self.output = None
60
61      def dopredu(self, v):  2 usages
62          self.output = np.tanh(v)
63          return self.output
64
65      def dozadu(self, g):  2 usages
66          return g * (1 - self.output ** 2)
```

5. Class relu

Táto trieda používa ReLU aktivačnú funkciu. Metóda dopredu vráti vstupy, ktoré sú kladné, a nahradí záporné hodnoty nulou. Počas spätného šírenia (dozadu) počíta gradient ako prichádzajúci gradient vynásobený deriváciou ReLU, ktorá je 1 pre kladné vstupy a 0 pre záporné. Tým umožňuje správnu aktualizáciu váh iba pre aktivované neuróny.

```
46  class relu:
47      def __init__(self):
48          self.input = None
49
50      def dopredu(self, v):
51          self.input = v
52          return np.maximum(0, v)
53
54      def dozadu(self, g):
55          return g * self.input * (self.input > 0)      #derivacia
```

Výstupy

Po 500 epochách sa spustí testovanie v podobe 4 vstupov (všetky možné kombinácie). Program na konci určí úspešnosť testovania a graf priebehu tréningu.

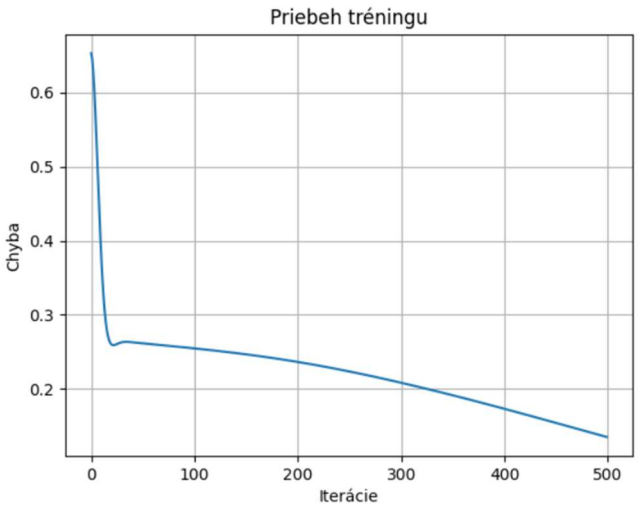
```
Vstup: 0,0  Výstup: [[0]]
Vstup: 0,1  Výstup: [[1]]
Vstup: 1,0  Výstup: [[1]]
Vstup: 1,1  Výstup: [[0]]
Správnosť výsledku: 100.0%
```

Testovanie

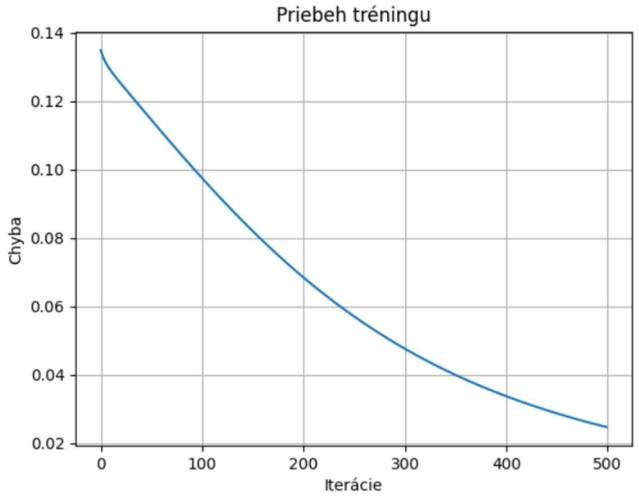
1. 1 skrytá vrstva (lineárna – aktivačná – lineárna – aktivačná so sigmoidom) - XOR

Sigmoid

```
------------------- S momentom -------------------
Chyba po 25 epochách: 0.260635001355045
Chyba po 50 epochách: 0.26151919635786
Chyba po 75 epochách: 0.2582073754169339
Chyba po 100 epochách: 0.2547283169624207
Chyba po 125 epochách: 0.2509409094444883
Chyba po 150 epochách: 0.24670681469385594
Chyba po 175 epochách: 0.24190716637244433
Chyba po 200 epochách: 0.2364638130166908
Chyba po 225 epochách: 0.2303525879710761
Chyba po 250 epochách: 0.22360030411576404
Chyba po 275 epochách: 0.21626743371906057
Chyba po 300 epochách: 0.20842701518881612
Chyba po 325 epochách: 0.20014968572351122
Chyba po 350 epochách: 0.1914979664236835
Chyba po 375 epochách: 0.18252739207813232
Chyba po 400 epochách: 0.1732907242972803
Chyba po 425 epochách: 0.16384273469763241
Chyba po 450 epochách: 0.15424454202605944
Chyba po 475 epochách: 0.1445670783529291

Vstup: 0,0  Výstup: [[0]]
Vstup: 0,1  Výstup: [[1]]
Vstup: 1,0  Výstup: [[1]]
Vstup: 1,1  Výstup: [[0]]
Správnosť výsledku: 100.0%
```

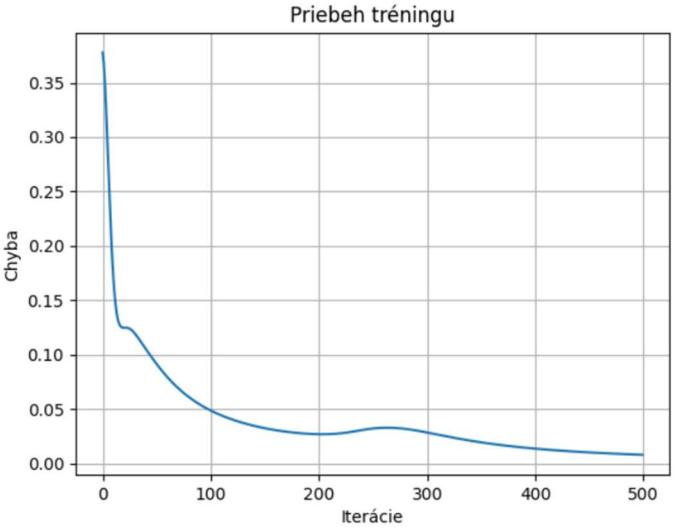



```
------------------ Bez momentu ------------------
Chyba po 25 epochách: 0.12331451911768983
Chyba po 50 epochách: 0.11450310675259065
Chyba po 75 epochách: 0.10585634002665416
Chyba po 100 epochách: 0.0974913390832552
Chyba po 125 epochách: 0.08950839476666801
Chyba po 150 epochách: 0.08198124216782104
Chyba po 175 epochách: 0.07495743846579964
Chyba po 200 epochách: 0.06846095064632753
Chyba po 225 epochách: 0.06249613552906901
Chyba po 250 epochách: 0.05705216892698217
Chyba po 275 epochách: 0.05210725243485619
Chyba po 300 epochách: 0.04763225671319681
Chyba po 325 epochách: 0.04359369292367368
Chyba po 350 epochách: 0.03995602827871315
Chyba po 375 epochách: 0.03668341567694221
Chyba po 400 epochách: 0.03374092596565097
Chyba po 425 epochách: 0.03109537396242705
Chyba po 450 epochách: 0.028715824427312863
Chyba po 475 epochách: 0.026573855172221654

Vstup: 0,0  Výstup: [[0]]
Vstup: 0,1  Výstup: [[1]]
Vstup: 1,0  Výstup: [[1]]
Vstup: 1,1  Výstup: [[0]]
Správnosť výsledku: 100.0%
```

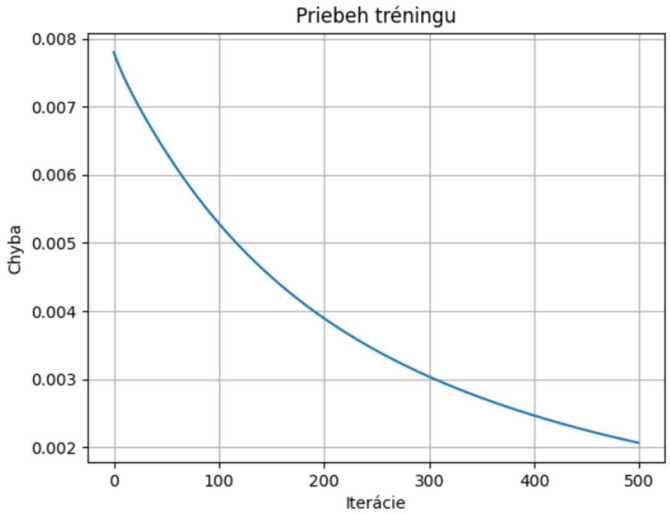


Tanh

```
------------------ S momentom ------------------
Chyba po 25 epochách: 0.12379081688061219
Chyba po 50 epochách: 0.09114518177246562
Chyba po 75 epochách: 0.0645806289308615
Chyba po 100 epochách: 0.04847227387115988
Chyba po 125 epochách: 0.038558551827897775
Chyba po 150 epochách: 0.032261176942068405
Chyba po 175 epochách: 0.028407410448431952
Chyba po 200 epochách: 0.026766392606599673
Chyba po 225 epochách: 0.02816324728683696
Chyba po 250 epochách: 0.03204365922682148
Chyba po 275 epochách: 0.03217355931085025
Chyba po 300 epochách: 0.028333656727556625
Chyba po 325 epochách: 0.02344438819854645
Chyba po 350 epochách: 0.019245610490809244
Chyba po 375 epochách: 0.01597388516837671
Chyba po 400 epochách: 0.01346473597386474
Chyba po 425 epochách: 0.01152457227228441
Chyba po 450 epochách: 0.010001406104632274
Chyba po 475 epochách: 0.00878570848486046

Vstup: 0,0  Výstup: [[0]]
Vstup: 0,1  Výstup: [[1]]
Vstup: 1,0  Výstup: [[1]]
Vstup: 1,1  Výstup: [[0]]
Správnosť výsledku: 100.0%
```

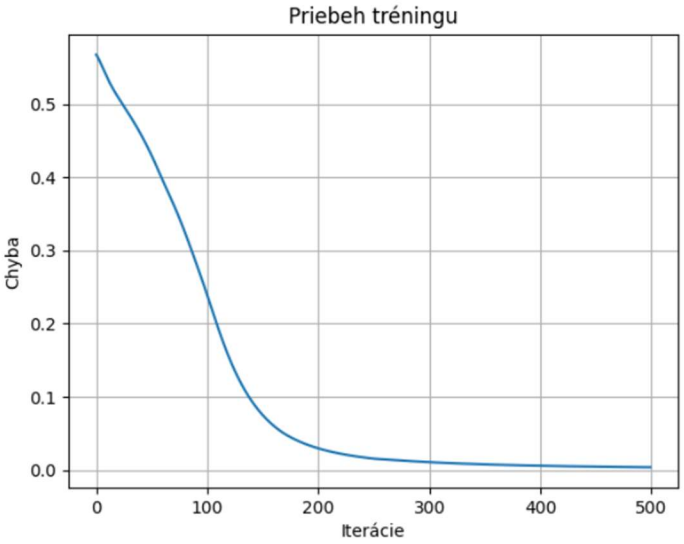



```
------------------ Bez momentu ------------------
Chyba po 25 epochách: 0.006969798438895498
Chyba po 50 epochách: 0.006325449717751344
Chyba po 75 epochách: 0.0057679316564510875
Chyba po 100 epochách: 0.0052845563628047985
Chyba po 125 epochách: 0.004864585857367826
Chyba po 150 epochách: 0.004498141191595859
Chyba po 175 epochách: 0.004176714856506988
Chyba po 200 epochách: 0.0038932083152616987
Chyba po 225 epochách: 0.0036417716219994518
Chyba po 250 epochách: 0.003417598867788216
Chyba po 275 epochách: 0.00321673595615862
Chyba po 300 epochách: 0.00303591710305265
Chyba po 325 epochách: 0.0028724314668257815
Chyba po 350 epochách: 0.0027240163453814505
Chyba po 375 epochách: 0.002588772268967436
Chyba po 400 epochách: 0.0024650955963934676
Chyba po 425 epochách: 0.002351624910573808
Chyba po 450 epochách: 0.0022471982346728195
Chyba po 475 epochách: 0.002150818727037473

Vstup: 0,0  Výstup: [[0]]
Vstup: 0,1  Výstup: [[1]]
Vstup: 1,0  Výstup: [[1]]
Vstup: 1,1  Výstup: [[0]]
Správnosť výsledku: 100.0%
```

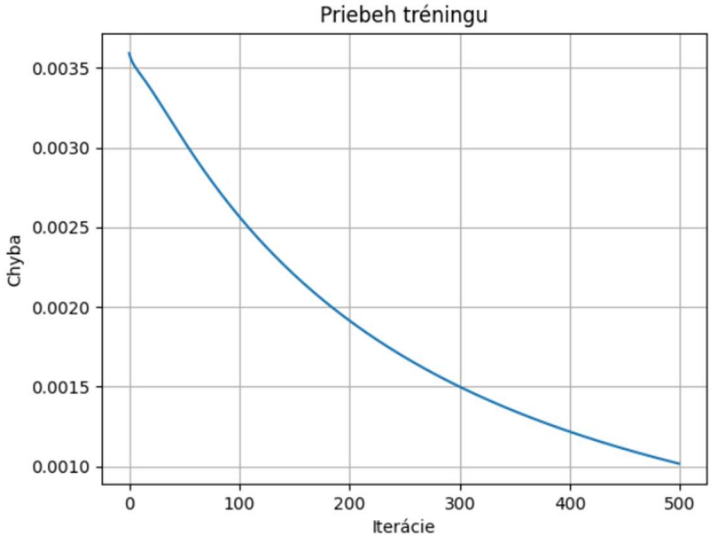


ReLU

```
------------------ S momentom ------------------
Chyba po 25 epochách: 0.49734243631762376
Chyba po 50 epochách: 0.4298857892722955
Chyba po 75 epochách: 0.3440437624664573
Chyba po 100 epochách: 0.23910810644339514
Chyba po 125 epochách: 0.13507635484535627
Chyba po 150 epochách: 0.074924318020157
Chyba po 175 epochách: 0.04474844870598256
Chyba po 200 epochách: 0.029493544813234508
Chyba po 225 epochách: 0.02074069471840372
Chyba po 250 epochách: 0.015359302749475553
Chyba po 275 epochách: 0.012782997087541095
Chyba po 300 epochách: 0.010574531433026724
Chyba po 325 epochách: 0.008881150093690272
Chyba po 350 epochách: 0.00756314885837345
Chyba po 375 epochách: 0.006521445281945227
Chyba po 400 epochách: 0.005686050988786525
Chyba po 425 epochách: 0.005006967101657077
Chyba po 450 epochách: 0.0044480076972497114
Chyba po 475 epochách: 0.003982625516524641

Vstup: 0,0  Výstup: [[0]]
Vstup: 0,1  Výstup: [[1]]
Vstup: 1,0  Výstup: [[1]]
Vstup: 1,1  Výstup: [[0]]
Správnosť výsledku: 100.0%
```



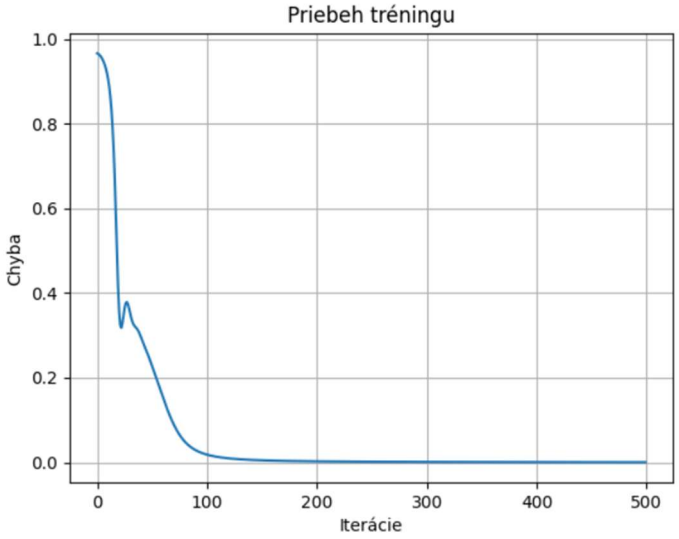

```
------------------ Bez momentu ------------------
Chyba po 25 epochách: 0.0033101014653333506
Chyba po 50 epochách: 0.0030384707434915614
Chyba po 75 epochách: 0.002787504441264519
Chyba po 100 epochách: 0.0025660424669299116
Chyba po 125 epochách: 0.002371614590777926
Chyba po 150 epochách: 0.00220036435191294 6
Chyba po 175 epochách: 0.0020487831504003144
Chyba po 200 epochách: 0.001913930725608739
Chyba po 225 epochách: 0.0017933798238314053
Chyba po 250 epochách: 0.0016851250123875235
Chyba po 275 epochách: 0.001587501123808437
Chyba po 300 epochách: 0.001499117286536733
Chyba po 325 epochách: 0.0014188047529029456
Chyba po 350 epochách: 0.001345575726855322
Chyba po 375 epochách: 0.0012785907424064853
Chyba po 400 epochách: 0.00121713266562775
Chyba po 425 epochách: 0.0011605858446461189
Chyba po 450 epochách: 0.001108419283296472
Chyba po 475 epochách: 0.0010601729793724244

Vstup: 0,0  Výstup: [[0]]
Vstup: 0,1  Výstup: [[1]]
Vstup: 1,0  Výstup: [[1]]
Vstup: 1,1  Výstup: [[0]]
Správnosť výsledku: 100.0%
```



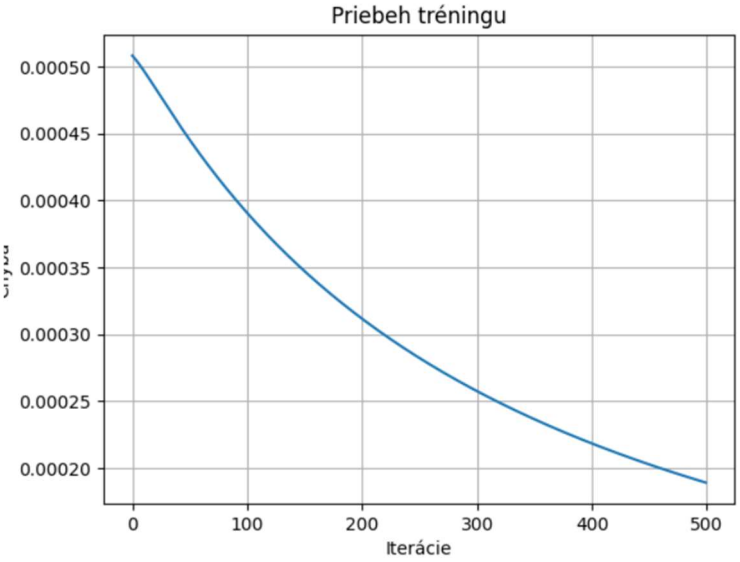
2. 2 skryté vrstvy (lineárna – aktivačná – lineárna – aktivačná – lineárna – aktivačná so sigmoidom) – OR

Sigmoid – ReLu

```
------------------- S momentom -------------------
Chyba po 25 epochách: 0.36492278276825957
Chyba po 50 epochách: 0.22763916032387504
Chyba po 75 epochách: 0.06518209002027453
Chyba po 100 epochách: 0.018352393699637686
Chyba po 125 epochách: 0.00843242587971862
Chyba po 150 epochách: 0.005032787891659739
Chyba po 175 epochách: 0.0034456299171464716
Chyba po 200 epochách: 0.0025601698908767614
Chyba po 225 epochách: 0.0020072294409598595
Chyba po 250 epochách: 0.0016342497123096932
Chyba po 275 epochách: 0.001368200524990854
Chyba po 300 epochách: 0.0011702332655220862
Chyba po 325 epochách: 0.0010179779096326346
Chyba po 350 epochách: 0.0008977335260951915
Chyba po 375 epochách: 0.0008006840077344262
Chyba po 400 epochách: 0.0007209225749269689
Chyba po 425 epochách: 0.0006543561774826098
Chyba po 450 epochách: 0.0005980666190068464
Chyba po 475 epochách: 0.0005499219177793415

Vstup: 0,0  Výstup: [[0]]
Vstup: 0,1  Výstup: [[1]]
Vstup: 1,0  Výstup: [[1]]
Vstup: 1,1  Výstup: [[1]]
Správnosť výsledku: 100.0%
```

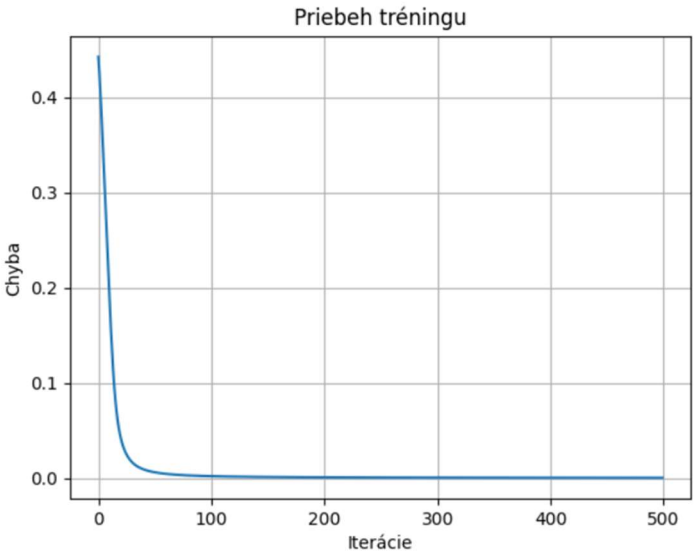



```
------------------- Bez momentu -------------------
Chyba po 25 epochách: 0.00047741516090582146
Chyba po 50 epochách: 0.0004453122719035438
Chyba po 75 epochách: 0.0004164482667505742
Chyba po 100 epochách: 0.00039075115699348267
Chyba po 125 epochách: 0.00036778771831314373
Chyba po 150 epochách: 0.000347163213339638
Chyba po 175 epochách: 0.00032854989489793052
Chyba po 200 epochách: 0.0003116753420576005
Chyba po 225 epochách: 0.00029631532553894764
Chyba po 250 epochách: 0.00028228067796831494
Chyba po 275 epochách: 0.00026941224082216577
Chyba po 300 epochách: 0.000257574936766664
Chyba po 325 epochách: 0.00024665342836282546
Chyba po 350 epochách: 0.00023654867635270886
Chyba po 375 epochách: 0.000227175188684251
Chyba po 400 epochách: 0.00021845880446433832
Chyba po 425 epochách: 0.00021033489709377777
Chyba po 450 epochách: 0.00020274690405572808
Chyba po 475 epochách: 0.00019564512167716166

Vstup: 0,0  Výstup: [[0]]
Vstup: 0,1  Výstup: [[1]]
Vstup: 1,0  Výstup: [[1]]
Vstup: 1,1  Výstup: [[1]]
Správnosť výsledku: 100.0%
```

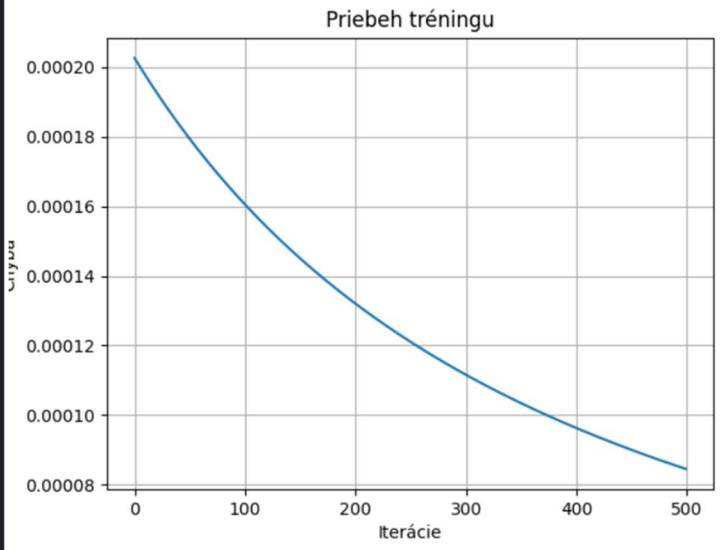


Tanh – ReLU

```
------------------- S momentom -------------------
Chyba po 25 epochách: 0.02455949513622561
Chyba po 50 epochách: 0.005955839592285538
Chyba po 75 epochách: 0.0029841516304889874
Chyba po 100 epochách: 0.0018966739537586836
Chyba po 125 epochách: 0.0013551784464203764
Chyba po 150 epochách: 0.0010379443510637432
Chyba po 175 epochách: 0.0008325550290571209
Chyba po 200 epochách: 0.0006900670554301063
Chyba po 225 epochách: 0.0005861114281427918
Chyba po 250 epochách: 0.0005073089376283277
Chyba po 275 epochách: 0.0004457584286688877
Chyba po 300 epochách: 0.0003964846075159396
Chyba po 325 epochách: 0.0003562621053849925
Chyba po 350 epochách: 0.0003228707200569038
Chyba po 375 epochách: 0.0002947547404265015
Chyba po 400 epochách: 0.0002707907805847364
Chyba po 425 epochách: 0.0002501478366153481
Chyba po 450 epochách: 0.0002322003909547935
Chyba po 475 epochách: 0.0002164678119386499

Vstup: 0,0  Výstup: [[0]]
Vstup: 0,1  Výstup: [[1]]
Vstup: 1,0  Výstup: [[1]]
Vstup: 1,1  Výstup: [[1]]
Správnosť výsledku: 100.0%
```




```
------------------ Bez momentu ------------------
Chyba po 25 epochách: 0.0001903070665818699
Chyba po 50 epochách: 0.00017932134552029887
Chyba po 75 epochách: 0.0001694349138582349
Chyba po 100 epochách: 0.00016049661741390583
Chyba po 125 epochách: 0.0001523810911349393
Chyba po 150 epochách: 0.00014498361146725525
Chyba po 175 epochách: 0.00013821610408322406
Chyba po 200 epochách: 0.00013200402409321526
Chyba po 225 epochách: 0.00012628389942459512
Chyba po 250 epochách: 0.00012100138186492683
Chyba po 275 epochách: 0.00011610968959362596
Chyba po 300 epochách: 0.00011156835384149223
Chyba po 325 epochách: 0.00010734220352834796
Chyba po 350 epochách: 0.00010340053743279996
Chyba po 375 epochách: 9.971644514999707e-05
Chyba po 400 epochách: 9.626624687198461e-05
Chyba po 425 epochách: 9.302902865650819e-05
Chyba po 450 epochách: 8.99862548936079e-05
Chyba po 475 epochách: 8.712144354117243e-05

Vstup: 0,0  Výstup: [[0]]
Vstup: 0,1  Výstup: [[1]]
Vstup: 1,0  Výstup: [[1]]
Vstup: 1,1  Výstup: [[1]]
Správnosť výsledku: 100.0%
```



Záver

V prvej úlohe bola úspešne implementovaná neurónová sieť pre regresný problém na predikciu ceny domov v Kalifornii. Pomocou knižnice PyTorch bol vytvorený model viacvrstvového perceptrónu, ktorý je optimalizovaný pomocou algoritmov (SGD, SGD s momentom, ADAM). Výsledky ukázali, že správna voľba optimalizácie a konfigurácia modelu významne ovplyvňuje presnosť predikcií.

V druhej úlohe je implementovaný backpropagation algoritmus pomocou knižnice NumPy. Bola vytvorená vlastná architektúra siete. Táto implementácia preukázala správnosť trénovania na jednoduchej sieti.

Používateľská príručka

Programy netreba nejako špeciálne spúšťať ak chceme otestovať pôvodné nastavenia.

Pri California Housing regresný model máme na začiatku možnosť výberu medzi 3 algoritmami (ADAM, SGD, SGD s momentom). Avšak je možnosť testovania aj s inými parametrami ako sú napríklad:

- Iná kombinácia aktivačných funkcií – nastavenie v triede Model metóde forward
- Iná hodnota batch size – časť Dataset a Dataloger
- Iný počet neurónov vo vrstvách - nastavenie v triede Model v jej atribútoch
- Iná rýchlosť učenia pri jednotlivých algoritmoch – posledná časť s inputom

Pri Backpropagation algoritme sa automaticky spúšťa XOR problém. Pre nadstavenie iného problému stačí v premennej good nastaviť výsledné hodnoty nášho problému. Avšak potom treba zmeniť aj vo funkcii spustenie() posledné štyri if sekcie tak aby tam boli správne odpovede.

Zdroje a citácie

- https://www.youtube.com/watch?v=aircAruvnKk
- https://www.youtube.com/watch?v=IHZwWFHWa-w
- https://www.youtube.com/watch?v=Ilg3gGewQ5U
- https://www.youtube.com/watch?v=tIeHLnjs5U8
- https://chatgpt.com/ - grafy, gramatika