Andrii Dokaniev
id:128365

6/12/2024

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológii

Backpropagation algoritmus

Zadanie č. 3a

Andrii Dokaniev
id:128365

Contents

Andrii Dokaniev
id:128365
6/12/2024

Popis zadania

Úlohou bolo vytvoríte neurónovú sieť na riešenie XOR problému s použitím **NumPy** knižnice. Implementácie obsahuje lineárnu vrstvu, aktivačnú funkciu sigmoid, tanh, relu a chybovú funkciu MSE (mean squared error). Základný model obsahuje skrytu vrstvu s 4 neurónmi a výstupnú vrstvu s jedným neurónom.

Reprezentácia údajov a konfigurácie

Na konfiguráciu bol použitý konfiguračný súbor, ktorý vyzerá nasledovne:

```
[Settings]
epoch_count = 200
learning_rate = 0.01
momentum = 0.9
use_momentum = False
first_activation_function_name(Sigmoid, Tanh, ReLU) = ReLU
second_activation_function_name(Sigmoid, Tanh, ReLU) = Tanh
```

Postup riešenia

Na zatáčku programu bolo importovane knižnica pre náhodne generovanie, matematická Tato cast

```
import numpy as np
import matplotlib.pyplot as plt
import time
import os
import configparser
from colorama import Fore, init
```

Knižnice pre načrtnutie štatistických grafov, knižnicu na sledovanie času, cesty k súborom, farebné výstupy a konfiguračný súbor.

Všetky potrebné knižnice je potrebne nainštalovať použitím príkazu “pip install <názov knižnice>“.

Po implementácii knižníc bolo zadefinovaných niekoľko najdôležitejších funkcií, takých ako:

```
"""Parameters for the configuration file"""
epoch_count = 0
learning_rate = 0
momentum = 0
```

```
first_activation_function_name = 'Sigmoid'
second_activation_function_name = 'Sigmoid'
use_momentum = False
```

Ďalej sú určené začiatočné parametre, ak konfiguračný súbor neexistuje, s následným načítaním konfiguračných parametrov zo súboru.

Ako funguje jednotlivé časti modelu je popísane v samotnom kóde v tvare komentárov.

Pre reprezentáciu MLP modelu bola vytvorená trieda, ktorá zahŕňa metódy, ako napríklad:

```
class MLP:
    AndriiQwq
    def __init__(self, X, Y):...
    1 usage  AndriiQwq
    def set_X(self, X):...
    1 usage  AndriiQwq
    def set_Y(self, Y):...
    2 usages  AndriiQwq *
    def create_layer(self, input_size, output_size, W, B, activation):...
    AndriiQwq
    def show_layers_information(self):...
    AndriiQwq
    def forward(self, x):...
    AndriiQwq *
    def backward(self, error):...
    2 usages  AndriiQwq
    def update_weights(self, current_layer, grad_MSE_W, grad_MSE_B):...
    1 usage  AndriiQwq
    def MSE_Loss_evaluating(self):...
    1 usage  AndriiQwq
    def test_model(self, test_data):...
    1 usage  AndriiQwq
    def get_last_layer_error(self):...
```

Na aktivačne funkcie bola tiež použitá trieda, s nasledujúcimi metódami:

```
class Activation_functions:
    AndriiQwq
    def __init__(self):...
    1 usage  AndriiQwq
    def process_activation_function(self, Z, activation_function_name):...
    1 usage  AndriiQwq
    def process_derivation_of_activation_function(self, Z, activation_function_name):...
    2 usages  AndriiQwq
    def Sigmoid(self, Z):...
    2 usages  AndriiQwq
    def Tanh(self, Z):...
    1 usage  AndriiQwq
    def ReLU(self, Z):...
    1 usage  AndriiQwq
    def Sigmoid_derivation(self, Z):...
    1 usage  AndriiQwq
    def Tanh_derivation(self, Z):...
    1 usage  AndriiQwq *
    def ReLU_derivation(self, Z):
        return np.where(Z > 0, 1, 0)
    1 usage  AndriiQwq
    def get_activation_function(self, layer):...
    2 usages  AndriiQwq
    def get_derivation_of_activation_function(self, layer):...
```

Na zatáčku boli nastavene vstupne a výstupne dáta pre XOR problém, inicializovaná model a vytvorené sloji modelu.

```
start_time = time.time()

"""Initialize initial input and output for XOR problem"""
```

```
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
Y = np.array([[0], [1], [1], [0]])

"""Initialize model"""
model = MLP(X, Y)

hidden_layer_size = 4

"""Create matrix 2x4 for Weights and fill it with random values"""
W1 = np.random.randn(2, hidden_layer_size) * np.sqrt(1 / 2)
W2 = np.random.randn(hidden_layer_size, 1) * np.sqrt(1 / 4)

B1 = np.zeros((1, hidden_layer_size))
B2 = np.zeros((1, 1))
```

Načítane testovacie dáta a zatáčok trénovania modelu:

```
L1 = model.create_layer(2, hidden_layer_size, W1, B1,
activation=first_activation_function_name)
L2 = model.create_layer(hidden_layer_size, 1, W2, B2,
activation=second_activation_function_name)

"""Training data"""
training_data = [
    (np.array([0, 0]).reshape(1, -1), np.array([0]).reshape(1, -1)),
    (np.array([0, 1]).reshape(1, -1), np.array([1]).reshape(1, -1)),
    (np.array([1, 0]).reshape(1, -1), np.array([1]).reshape(1, -1)),
    (np.array([1, 1]).reshape(1, -1), np.array([0]).reshape(1, -1))
]
training_data_size = len(training_data)
losses = []

for epoch in range(epoch_count):
    """Reshuffle the training data"""
    np.random.shuffle(training_data)

    total_error = 0

    for i in range(training_data_size):
        for input_set, label in training_data:
            model.set_X(input_set)
            model.set_Y(label)

            model.forward(input_set)

            error = model.get_last_layer_error()
            model.backward(error)

            total_error += model.MSE_Loss_evaluating()

    average = total_error / len(training_data)
    losses.append(total_error / len(training_data))

    """Check stopping condition"""
    if average < 0.00001 or epoch == 20000:
        print("We reached the optimal model")
        break

    if (epoch + 1) % 10 == 0:
        print(f"Epoch {epoch + 1}/{epoch_count}, Loss: {average:.5f}")
```

```
plt.plot(losses)
plt.xlabel('Epoch')
plt.ylabel('Average Loss')
plt.title('Training Progress')
plt.show()

"""Testing model"""
model.test_model(training_data)
```

Počas trénovania dopredaný a spetý smer(na propagáciu chyby a aktualizáciu váh a posun, medzi neurónmi). Taktiež bola použitá MSE chybová funkcia na ohodnotenie modelu. Pre hodnoty gradientov a som požil materiály s prednášok a pre lepšie pochopenie načo slúži jednotne parametre, aký hodnoty neodbúdajú a aký súvisle z nimi formuly existujú(teoretické vedomosti) použil pomoc UI, vzorce pre aktivačne funkcie boli zobrané z prednášok a internetu. Vytvorený model je zoradený na jednotne metódy, čo umožňuje rýchlo meniť nastavenia(počet slojov, aktivačne funkcii a iné). Vytvorená trieda pre MLP zahŕňa sloje s nastaviteľnými parametrami a zoznamami pre jednotne sloje.

```
def create_layer(self, input_size, output_size, W, B, activation):
    """Where first two values is a matrix size, W is weight, B is bias, Z is intermediate result, a is output"""
    layer = {
        'input_size': input_size,
        'output_size': output_size,
        'W': W,
        'B': B,
        'Z': None,
        'a': None,
        'error': None,
        'vW': np.zeros_like(W),
        'vB': np.zeros_like(B),
        'activation': activation
    }

    self.layers.append(layer)
    return layer

def show_layers_information(self):
    for layer in self.layers:
        print(Fore.LIGHTYELLOW_EX + f'Layer {self.layers.index(layer) + 1}')
        print(Fore.LIGHTGREEN_EX +
              f"Size: {layer['input_size']}x{layer['output_size']},"
              f"\n Weight: {layer['W']},\n Bias: {layer['B']},\n"
              f" Z: {layer['Z']},\n a: {layer['a']}\n")

def forward(self, x):
    """Forward input x to each layer, each layer recalculate inputs values and forward to the up layer"""
    for layer in self.layers:
        layer['Z'] = np.dot(x, layer['W']) + layer['B']
        """Forward result to activation function(Sigmoid, Tanh, ReLU)"""
        activation_function_method = Activation_functions()
        layer['a'] = activation_function_method.get_activation_function(layer)
        x = layer['a']


    return x
```

```
def backward(self, error):
    """For output layer"""
    activation_function_method = Activation_functions()
    derivation_of_activation_function = activation_function_method.get_derivation_of_activation_function(
        self.layers[-1])

    delta_output_error = error * derivation_of_activation_function

    """Gradient off loss function to weight, (last layer input) * (delta==error * (f'(Z)))"""
    grad_MSE_W = np.dot(self.layers[-2]['a'].T, delta_output_error)
    grad_MSE_B = np.sum(delta_output_error, axis=0, keepdims=True)

    """Updating weights for output layer"""
    current_layer = self.layers[-1]
    self.update_weights(current_layer, grad_MSE_W, grad_MSE_B)

    """For next layer"""
    iterator = 2
    for _ in range(len(self.layers) - 1):
        current_layer = self.layers[-iterator]

        derivation_of_activation_function_hidden =
activation_function_method.get_derivation_of_activation_function(
            current_layer)
        error_for_hidden_layer = np.dot(delta_output_error, self.layers[-iterator + 1]['W'].T)

        next_delta_hidden_layer = error_for_hidden_layer * derivation_of_activation_function_hidden

        """Control if we reach first(input) layer and don't have any more layers"""
        if iterator == len(self.layers):
            grad_MSE_W = np.dot(self.X.T, next_delta_hidden_layer)
        else:
            grad_MSE_W = np.dot(self.layers[-iterator - 1]['a'].T, next_delta_hidden_layer)
        grad_MSE_B = np.sum(next_delta_hidden_layer, axis=0, keepdims=True)

        """Updating weights for hidden layer"""
        self.update_weights(current_layer, grad_MSE_W, grad_MSE_B)

        """Updating values of MSE_a gradient to distribute it for next layer"""
        delta_output_error = next_delta_hidden_layer
        iterator += 1

def update_weights(self, current_layer, grad_MSE_W, grad_MSE_B):
    if use_momentum:
        current_layer['vW'] = momentum * current_layer['vW'] - learning_rate * grad_MSE_W
        current_layer['vB'] = momentum * current_layer['vB'] - learning_rate * grad_MSE_B
        current_layer['W'] += current_layer['vW']
        current_layer['B'] += current_layer['vB']
    else:
        current_layer['W'] -= learning_rate * grad_MSE_W
        current_layer['B'] -= learning_rate * grad_MSE_B

def MSE_Loss_evaluating(self):
    """MSE loss function"""
    predicated_output = self.layers[-1]['a']
    error = np.power(self.Y - predicated_output, 2).sum() / (len(self.Y) * 2)
    return error
```

```
def test_model(self, test_data):
    correct = 0
    print(Fore.GREEN + "------------------------------------------------")
    for input_set, label in test_data:
        # model.set_X(input_set)
        # model.set_Y(label)

        model.forward(input_set)
        """a is 1x4 matrix: [a11, a12, a13, a14]"""
        output = self.layers[-1]['a']

        if round(output.item()) == label:
            print(Fore.BLUE + f"Input set: {input_set}, Label: {label}",
                  Fore.GREEN + f"Correct prediction: {output} == {label}")
            correct += 1
        else:
            print(Fore.BLUE + f"Input set: {input_set}, Label: {label}",
                  Fore.RED + f"Incorrect prediction: {output} != {label}")
    print(Fore.GREEN + "------------------------------------------------")

    if correct == len(test_data):
        print(Fore.GREEN + "Correct model")
    else:
        print(Fore.RED + "Incorrect model")

def get_last_layer_error(self):
    predicted_output = model.layers[-1]['a']
    Y_label = self.Y
    last_error = predicted_output - Y_label
    return last_error * 2
```

Testovanie

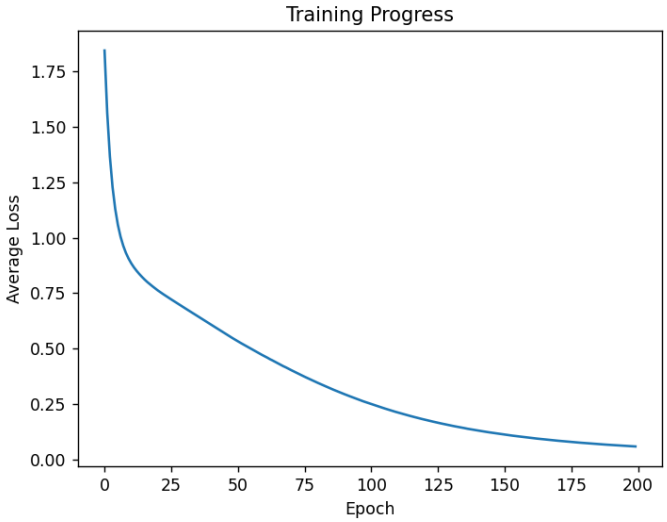
Počas testovania používal a uviedol rôzne metódy na trénovanie s momentom a bez, počet vrstiev a ich veľkosti, rôzne aktivačné funkcie(ReLU, Sigmoid, Tanh), rýchlosť učenia(learning rate) a rôzne začiatočne hodnoty. Počas testovania v niektorých prípadoch mal zle nastavenú chybovú funkciu. Výsledky ukázali, že funkcia Sigmoid hľadala výsledok oveľa dlhšie ako iné aktivačne funkcie. Taktiež pri použití všetkých aktivačných funkcii ReLU, dobrý a rýchly výsledok mohli neočakávať.

Testovanie so skrytou vrstvou s 4 neurónmi a východnej s 1.

```
epoch_count = 200
learning_rate = 0.01
use_momentum = False
first_activation_function_name(Sigmoid, Tanh, ReLU) = ReLU
second_activation_function_name(Sigmoid, Tanh, ReLU) = Tanh
```

Andrii Dokaniev
id:128365

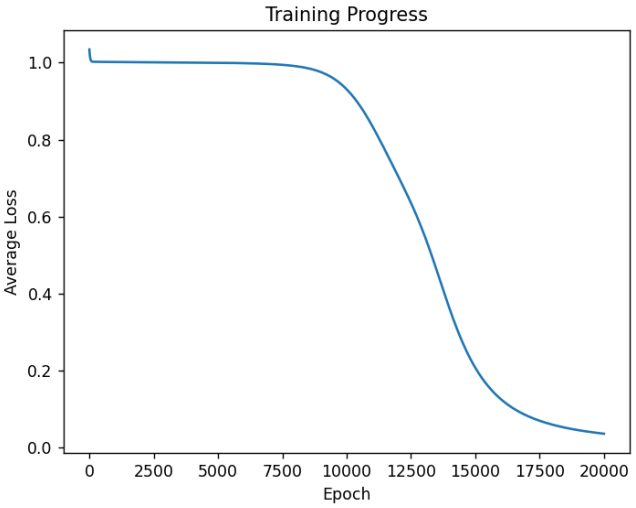
6/12/2024



s

```
Epoch 200/200, Loss: 0.06072
--------------------------------------------------
Input set: [[0 1]], Label: [[1]] Correct prediction: [[0.84345134]] == [[1]]
Input set: [[1 1]], Label: [[0]] Correct prediction: [[0.05682156]] == [[0]]
Input set: [[0 0]], Label: [[0]] Correct prediction: [[0.05632071]] == [[0]]
Input set: [[1 0]], Label: [[1]] Correct prediction: [[0.82921216]] == [[1]]
--------------------------------------------------
Correct model
```

```
epoch_count = 200
learning_rate = 0.01
use_momentum = False
first_activation_function_name(Sigmoid, Tanh, ReLU) = Sigmoid
second_activation_function_name(Sigmoid, Tanh, ReLU) = Sigmoid
```



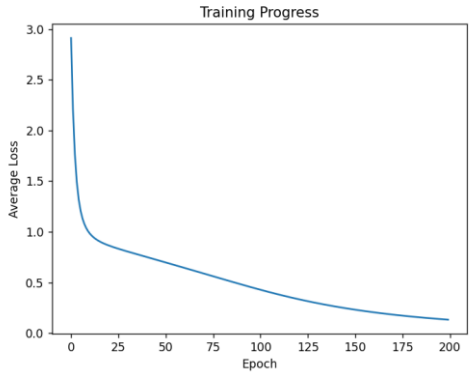
Andrii Dokaniev
id:128365

6/12/2024

Pre Sigmoid bolo potrebne viac epoch pre nejdenia optimálnych parametrov

```
Epoch 20000/20000, Loss: 0.03596
-------------------------------------------------
Input set: [[1 1]], Label: [[0]] Correct prediction: [[0.12325369]] == [[0]]
Input set: [[0 0]], Label: [[0]] Correct prediction: [[0.05987767]] == [[0]]
Input set: [[0 1]], Label: [[1]] Correct prediction: [[0.91525131]] == [[1]]
Input set: [[1 0]], Label: [[1]] Correct prediction: [[0.9000529]] == [[1]]
-------------------------------------------------
Correct model
```

```
epoch_count = 200
learning_rate = 0.01
use_momentum = False
first_activation_function_name(Sigmoid, Tanh, ReLU) = Tanh
second_activation_function_name(Sigmoid, Tanh, ReLU) = Tanh
```

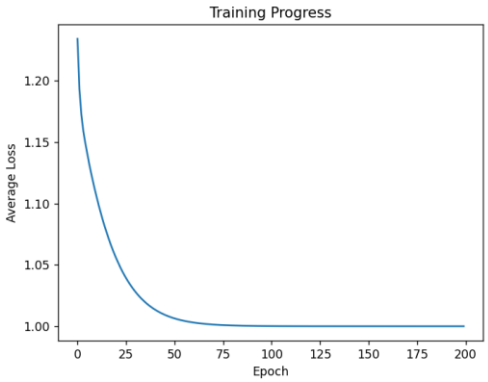



```
-------------------------------------------------
Input set: [[0 1]], Label: [[1]] Correct prediction: [[0.75115049]] == [[1]]
Input set: [[1 1]], Label: [[0]] Correct prediction: [[0.12777318]] == [[0]]
Input set: [[1 0]], Label: [[1]] Correct prediction: [[0.77410724]] == [[1]]
Input set: [[0 0]], Label: [[0]] Correct prediction: [[0.03767469]] == [[0]]
-------------------------------------------------
Correct model
```

```
epoch_count = 200
learning_rate = 0.01
use_momentum = False
first_activation_function_name(Sigmoid, Tanh, ReLU) = ReLU
second_activation_function_name(Sigmoid, Tanh, ReLU) = ReLU
```

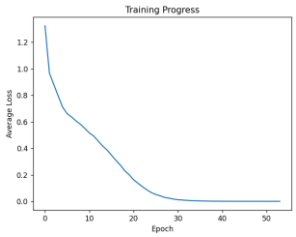
Andrii Dokaniev
id:128365

6/12/2024




```
--------------------------------------------------
Input set: [[0 0]], Label: [[0]] Correct prediction: [[0.00036622]] == [[0]]
Input set: [[1 0]], Label: [[1]] Incorrect prediction: [[0.]] != [[1]]
Input set: [[0 1]], Label: [[1]] Correct prediction: [[0.99982872]] == [[1]]
Input set: [[1 1]], Label: [[0]] Correct prediction: [[7.72171819e-05]] == [[0]]
--------------------------------------------------
Incorrect model
```

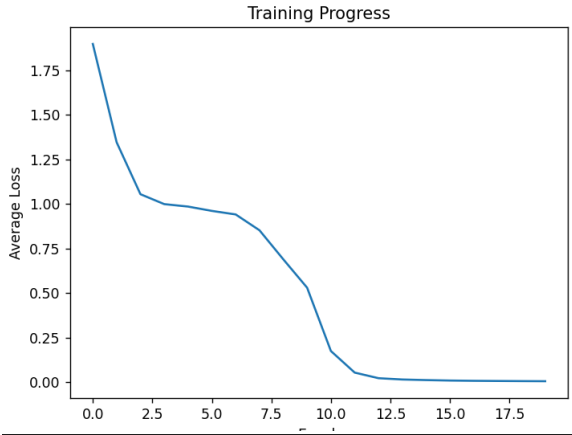
Nevzdi nájde riešenie




```
Epoch 50/20000, Loss: 0.00003
We reached the optimal model
--------------------------------------------------
Input set: [[1 1]], Label: [[0]] Correct prediction: [[0.00065043]] == [[0]]
Input set: [[0 1]], Label: [[1]] Correct prediction: [[0.99786821]] == [[1]]
Input set: [[0 0]], Label: [[0]] Correct prediction: [[0.00090996]] == [[0]]
Input set: [[1 0]], Label: [[1]] Correct prediction: [[1.0001246]] == [[1]]
--------------------------------------------------
Correct model
```

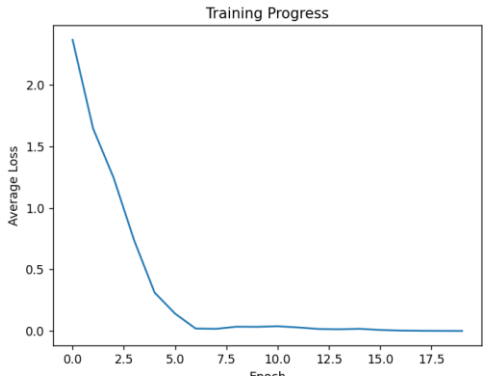
```
epoch_count = 20
learning_rate = 0.1
momentum = 0.9
use_momentum = True
first_activation_function_name(Sigmoid, Tanh, ReLU) = Tanh
second_activation_function_name(Sigmoid, Tanh, ReLU) = Tanh
```

Andrii Dokaniev
id:128365
6/12/2024



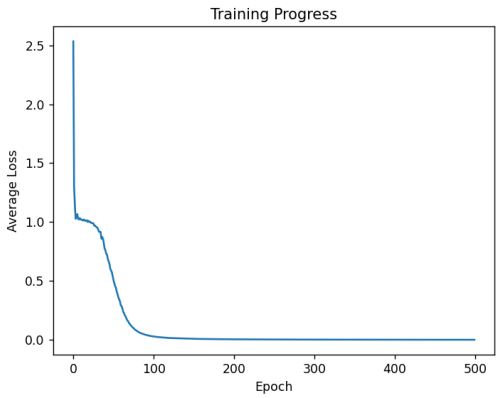

```
------------------------------------------------
Input set: [[1 1]], Label: [[0]] Correct prediction: [[0.00574002]] == [[0]]
Input set: [[0 0]], Label: [[0]] Correct prediction: [[0.00177103]] == [[0]]
Input set: [[1 0]], Label: [[1]] Correct prediction: [[0.94987272]] == [[1]]
Input set: [[0 1]], Label: [[1]] Correct prediction: [[0.9535892]] == [[1]]
------------------------------------------------
```

```
epoch_count = 20
learning_rate = 0.1
momentum = 0.95
use_momentum = True
first_activation_function_name(Sigmoid, Tanh, ReLU) = Tanh
second_activation_function_name(Sigmoid, Tanh, ReLU) = Tanh
```




```
epoch_count = 500
learning_rate = 0.01
momentum = 0.90
use_momentum = True
first_activation_function_name(Sigmoid, Tanh, ReLU) = Tanh
second_activation_function_name(Sigmoid, Tanh, ReLU) = Tanh
```

Andrii Dokaniev
id:128365




```
Epoch 500/500, Loss: 0.00113
-----------------------------------------------
Input set: [[1 1]], Label: [[0]] Correct prediction: [[0.00146198]] == [[0]]
Input set: [[0 1]], Label: [[1]] Correct prediction: [[0.9768563]] == [[1]]
Input set: [[0 0]], Label: [[0]] Correct prediction: [[0.00103047]] == [[0]]
Input set: [[1 0]], Label: [[1]] Correct prediction: [[0.97583372]] == [[1]]
-----------------------------------------------
```

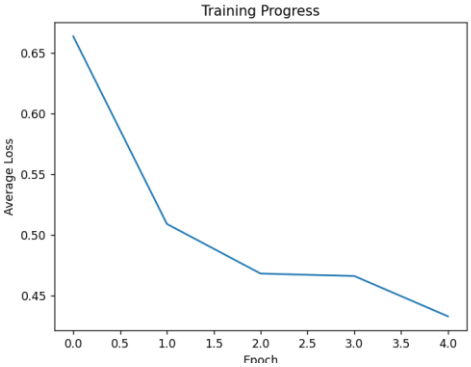
Najlepšie nájdene parametre

```
epoch_count = 20
learning_rate = 0.1
momentum = 0.95
use_momentum = True
first_activation_function_name(Sigmoid, Tanh, ReLU) = Tanh
second_activation_function_name(Sigmoid, Tanh, ReLU) = Tanh
```

AND a OR problém

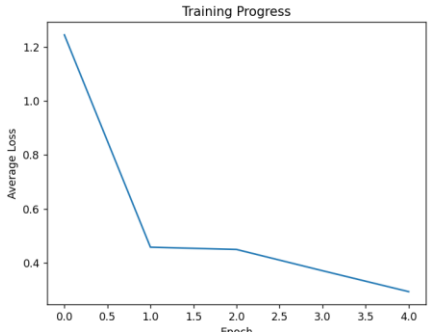
Pre AND a OR problém dosiahnuť výsledkov je oveľa ľahšie:

```
epoch_count = 5
learning_rate = 0.01
momentum = 0.90
use_momentum = True
first_activation_function_name(Sigmoid, Tanh, ReLU) = Tanh
second_activation_function_name(Sigmoid, Tanh, ReLU) = Tanh
```




```
Input set: [[0 0]], Label: [[0]] Correct prediction: [[-0.12098514]] == [[0]]
Input set: [[1 0]], Label: [[0]] Correct prediction: [[0.42205925]] == [[0]]
Input set: [[0 1]], Label: [[0]] Correct prediction: [[0.14376326]] == [[0]]
Input set: [[1 1]], Label: [[1]] Correct prediction: [[0.54783385]] == [[1]]
```

```
epoch_count = 5
learning_rate = 0.01
momentum = 0.90
use_momentum = True
first_activation_function_name(Sigmoid, Tanh, ReLU) = Tanh
second_activation_function_name(Sigmoid, Tanh, ReLU) = Tanh
```



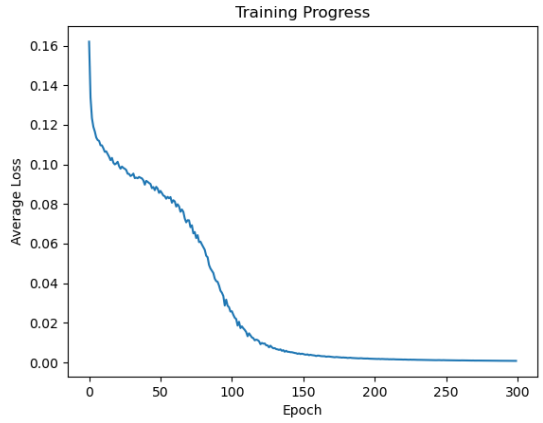

```
Input set: [[0 1]], Label: [[1]] Correct prediction: [[0.86005465]] == [[1]]
Input set: [[1 1]], Label: [[1]] Correct prediction: [[0.90447967]] == [[1]]
Input set: [[1 0]], Label: [[1]] Correct prediction: [[0.66975309]] == [[1]]
Input set: [[0 0]], Label: [[0]] Correct prediction: [[0.35349319]] == [[0]]
```

Viac slojov

Vytvoril som možnosť pridania skrytých slojov, počet slojov môžeme meniť, takže ako aj ich rozmery. Taktiež je možne nastaviť rôzne aktivačne funkcie.

Príklad pre input-4-4-out(**Tanh, Tanh, Tanh)**):

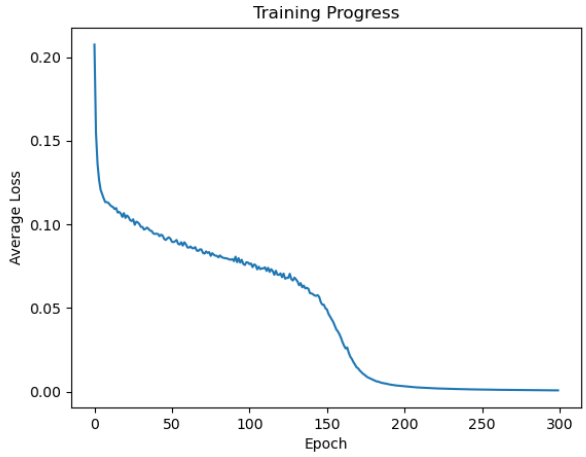
```
W1 = np.random.randn(2, hidden_layer_size) * np.sqrt(1 / 2)
W2 = np.random.randn(hidden_layer_size, hidden_layer_size) * np.sqrt(1 / 4)
W3 = np.random.randn(4, 1) * np.sqrt(1 / 4)

B1 = np.zeros((1, hidden_layer_size))
B2 = np.zeros((1, 4))
B3 = np.zeros((1, 1))

L1 = model.create_layer(2, hidden_layer_size, W1, B1, activation=first_activation_function_name)
L2 = model.create_layer(hidden_layer_size, 4, W2, B2, activation=second_activation_function_name)
L3 = model.create_layer(4, 1, W3, B3, activation=second_activation_function_name)
```




```
------------------------------------------------
Input set: [[0 0]], Label: [[0]] Correct prediction: [[0.00394491]] == [[0]]
Input set: [[1 1]], Label: [[0]] Correct prediction: [[0.00822916]] == [[0]]
Input set: [[1 0]], Label: [[1]] Correct prediction: [[0.95104901]] == [[1]]
Input set: [[0 1]], Label: [[1]] Correct prediction: [[0.94021838]] == [[1]]
------------------------------------------------
Correct model
```

Andrii Dokaniev
id:128365

6/12/2024



(Zmenili aktivačne funkcie, ReLU, Tanh, Tanh)

Andrii Dokaniev
id:128365
6/12/2024

Zhodnotenie

Vytvorená neurónová sieť na riešenie AND, OR a XOR problémov s použitím Backpropagation algoritmu určená na trénovanie a testovanie vytvoreného systému s rôznymi parametrami, ako napríklad: použitie momentuma, vhodné aktivačné funkcie, rýchlosť učenia. Boli otestovane a porovnane výsledky testovania a vyhodnotená úspešnosť nerehneného modelu. Boli opísaná model, porovnanie pre rôzne rozmery a počet vrstiev pre vytvorený model.

Výtvory, natrénovaný model je schopný riešiť XOR problém s 100% úspešnosťou. Závisí od poctu epoch a nastavení modelu na trénovaní a inými nastaveniami. Najlepší výsledok bol opísaný v testovacej časti a dokázal riešiť problém za minimálny počet epoch.

Taktiež program umožňuje konfigurovať konfiguráciu pred behom programu. Čo výrazne zrýchľuje prístup k jednotným špecifikáciám a prispôsobeným nástrojom pre určite potreby klienta. Ako napríklad používateľ môže si zvoliť spôsob učenia site, aká najviac mu vyhovuje. Ešte jedným veľkým plusom je to, že program umožňuje jednoducho a rýchlo meniť počet a rozmery vrstiev a ich aktivačne funkcii, čo umožňuje používateľovi prispôsobiť sieť pre svoj typ problému.

Dosť dôležitou úlohou bolo správne vybrať a nastrojiť parametre pre vytvorený algoritmus. Boli porovnané rôzne spôsoby konfigurácie, a navrhnutá najlepšia konfigurácia, ktorú sa dalo vytvoriť. Pre vytvorený kód boli pridelene komentáre pre včešú pochopiteľnosť systému.