SPRAWOZDANIE

Zajęcia: Matematyka Konkretna

Prowadzący: prof. dr hab. inż. Vasyl Martsenyuk

Laboratorium Nr 7 Data 21.11.2023 Temat: Gradienty, wsteczna propagacja błędu Wariant 6	Rafał Klinowski Informatyka II stopień, stacjonarne, 2 semestr, gr. a

1. Polecenie:

Ćwiczenie polegało na stworzeniu notatnika Jupyter w języku Python do przeprowadzenia analizy gradientów oraz wstecznej propagacji błędu w sieciach neuronowych. Ostatecznie polecenie polegało na wykonaniu tylko jednego z zadań.

Wariant zadania: 6

Zadanie dotyczy obliczenia gradientów sieci neuronowej z pomocą biblioteki numpy zadanej z pomocy architektury

6.
```
nn_architecture = [
    {"input_dim": 2, "output_dim": 2, "activation": "relu"},
    {"input_dim": 2, "output_dim": 1, "activation": "tanh"}
]
```

2. Napisany program, uzyskane wyniki

Po zapoznaniu się z instrukcją laboratoryjną uznano, że najprostszym sposobem analizy metody wstecznej propagacji błędu jest wykorzystanie biblioteki Keras do zdefiniowania sieci neuronowej oraz pracy nad nią.

W związku z tym na podstawie instrukcji utworzono sieć zgodnie z wariantem.

```
model = Sequential()
model.add(Dense(2, activation='relu', input_shape=(2,)))
model.add(Dense(1, activation='tanh'))
model.summary()
```

Rysunek 1. Utworzona sieć zgodnie z wariantem 6.

Model: "sequential_7"

Layer (type)	Output Shape	Param #
dense_14 (Dense)	(None, 2)	6
dense_15 (Dense)	(None, 1)	3

Total params: 9 (36.00 Byte)

Trainable params: 9 (36.00 Byte)

Non-trainable params: 0 (0.00 Byte)

Rysunek 2. Wygląd zdefiniowanej sieci neuronowej.

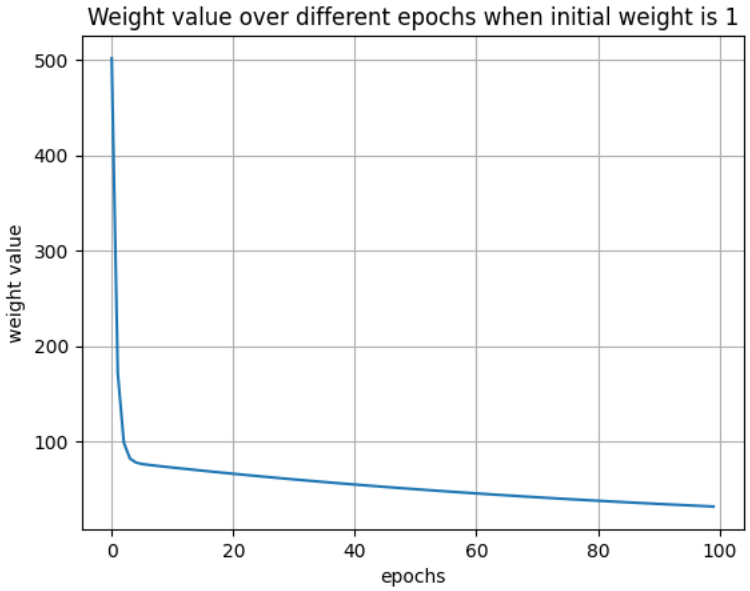
Kolejnym krokiem było zdefiniowanie algorytmu aktualizacji wag sieci.

```
def feed_forward(inputs, outputs, weights):
    hidden = np.dot(inputs,weights[0])
    out = hidden+weights[1]
    squared_error = (np.square(out - outputs))
    return squared_error

def update_weights(inputs, outputs, weights, epochs):
    for epoch in range(epochs):
        org_loss = feed_forward(inputs, outputs, weights)
        wts_tmp = deepcopy(weights)
        wts_tmp2 = deepcopy(weights)
        for ix, wt in enumerate(weights):
            wts_tmp[-(ix+1)] += 0.0001
            # print('wts_tmp:', wts_tmp)
            loss = feed_forward(inputs, outputs, wts_tmp)
            # print('loss', loss)
            del_loss = np.sum(org_loss - loss)/(0.0001*len(inputs))
            wts_tmp2[-(ix+1)] += del_loss*0.01
            wts_tmp = deepcopy(weights)

        weights = deepcopy(wts_tmp2)
    return wts_tmp2
```

Wreszcie utworzono początkowe wagi oraz przeprowadzono analizę.

```
w = [2000, 0]
update_weights(x,y,w,1)

w_val = []
b_val = []
for k in range(100):
    w_new, b_new = update_weights(x,y,w,(k+1))
    w_val.append(w_new)
    b_val.append(b_new)
```

Rysunek 3. Aktualizacja wag.



Rysunek 4. Wykres przedstawiający wagi przez kolejne 100 iteracji.

```
w = list(model.get_weights().copy())
print(w)

update_weights(x,y,w,100)

model.fit(np.array(x), np.array(y), epochs=100, batch_size = 4, verbose=1)

model.get_weights()
```

Rysunek 5. Testowanie sieci.

```
[array([[-0.8625427 ,  1.1658314 ],
        [ 0.23958993, -1.0222448 ]], dtype=float32),
 array([-0.12325726,  0.        ], dtype=float32),
 array([[-0.4562307 ],
        [ 0.39224863]], dtype=float32),
 array([2.2794394], dtype=float32)]
```

Rysunek 6. Uzyskany wynik – wagi sieci.

Sposób w jaki zostało przeprowadzone zadanie jest o wiele prostszy niż druga metoda „od zera” z wykorzystaniem jedynie Numpy. Wymaga mniej kodu oraz jest prostszy do zrozumienia niż implementacja całej propagacji wstecznej samemu.

Repozytorium zawierające uzyskane wyniki wraz z niezbędnymi plikami: https://github.com/Stukeley/MatematykaKonkretna/tree/master/Lab7