SPRAWOZDANIE

Zajęcia: Matematyka Konkretna

Prowadzący: prof. dr hab. Vasyl Martsenyuk

Laboratorium Nr 8 Data 28.06.2025 Temat: „Algorytm LSTM dla tekstu” Wariant 10	Anna Więzik Informatyka II stopień, niestacjonarne, 2 semestr, gr.1a TTO

1. Polecenie:

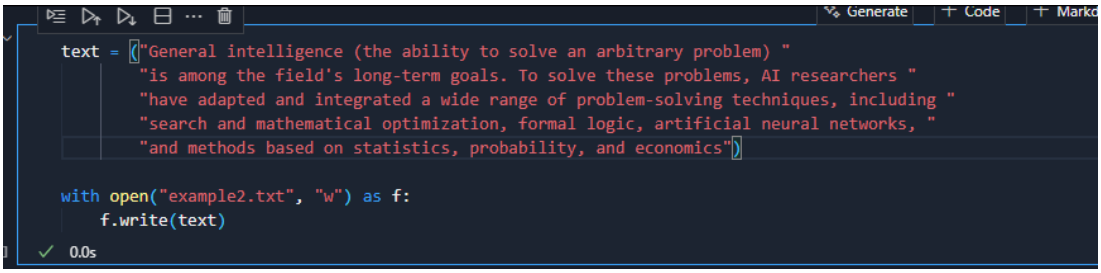
Link do repozytorium: https://github.com/AnaShiro/MK_2025

Celem zadania było stworzenie i zbadanie działania sieci LSTM (Long Short-Term Memory), uczonej z wykorzystaniem metody propagacji wstecznej w czasie (BPTT), w kontekście analizy sekwencji tekstowych. LSTM stanowi rozwinięcie klasycznych rekurencyjnych sieci neuronowych, zaprojektowane w celu skuteczniejszego radzenia sobie z problemem zanikającego gradientu, który utrudnia uczenie się długoterminowych zależności w tradycyjnych RNN. Dzięki mechanizmom bramek – wejściowej, wyjściowej i zapominania – LSTM potrafi selektywnie gromadzić oraz odrzucać informacje w pamięci wewnętrznej, co umożliwia jej zapamiętywanie kluczowych fragmentów sekwencji.

Proces uczenia tej sieci wymaga szczególnego podejścia do wyznaczania gradientów, ponieważ uczenie rozciąga się w czasie. Metoda BPTT pozwala śledzić błąd wzdłuż kolejnych kroków czasowych, by dokładnie ocenić wpływ każdego z nich na modyfikację wag. W trakcie pracy wykorzystano funkcje aktywacyjne sigmoid oraz tangens hiperboliczny i ich pochodne, co umożliwiło prawidłowe obliczenia sygnałów i gradientów. Wagi początkowe zainicjowano losowo z rozkładu normalnego, a dla bramek sigmoid zastosowano przesunięcie średniej, co pozwoliło na lepszą kontrolę początkowych aktywacji i ułatwiło start procesu nauki.

Dodatkowo przygotowano odpowiedni proces wstępnej obróbki danych tekstowych: przekształcono dane na zestaw unikalnych znaków (alfabet) i zakodowano je w sposób umożliwiający podanie do sieci i dekodowanie wyników. Dzięki temu możliwe było trenowanie modelu tak, by przewidywał kolejne znaki na podstawie poprzedniego kontekstu, systematycznie aktualizując parametry w celu minimalizacji funkcji kosztu.

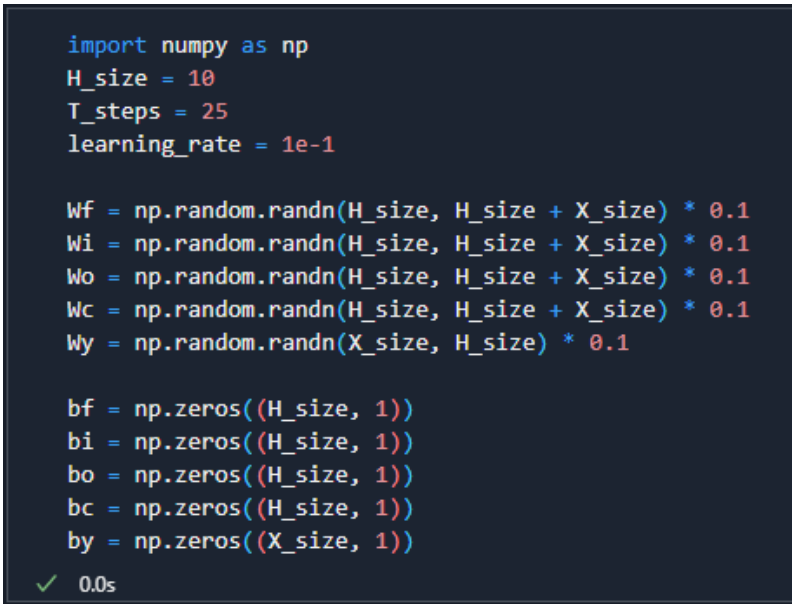
2. Opis programu opracowanego



```
text = ("General intelligence (the ability to solve an arbitrary problem) "
        "is among the field's long-term goals. To solve these problems, AI researchers "
        "have adapted and integrated a wide range of problem-solving techniques, including "
        "search and mathematical optimization, formal logic, artificial neural networks, "
        "and methods based on statistics, probability, and economics")

with open("example2.txt", "w") as f:
    f.write(text)
```

✓ 0.0s

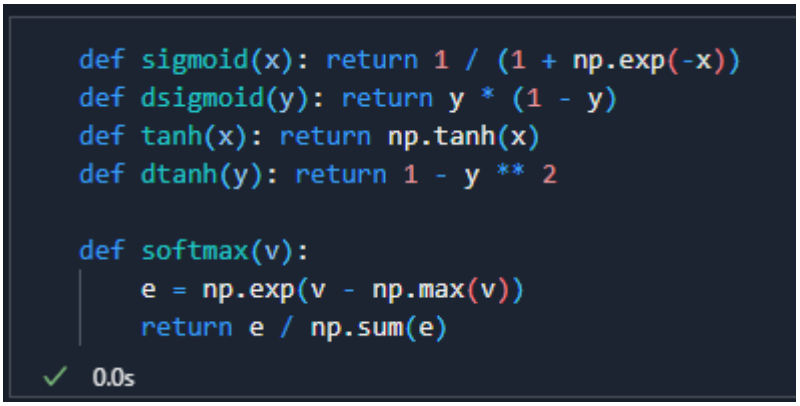


```
import numpy as np
H_size = 10
T_steps = 25
learning_rate = 1e-1

Wf = np.random.randn(H_size, H_size + X_size) * 0.1
Wi = np.random.randn(H_size, H_size + X_size) * 0.1
Wo = np.random.randn(H_size, H_size + X_size) * 0.1
Wc = np.random.randn(H_size, H_size + X_size) * 0.1
Wy = np.random.randn(X_size, H_size) * 0.1

bf = np.zeros((H_size, 1))
bi = np.zeros((H_size, 1))
bo = np.zeros((H_size, 1))
bc = np.zeros((H_size, 1))
by = np.zeros((X_size, 1))
```

✓ 0.0s



```
def sigmoid(x): return 1 / (1 + np.exp(-x))
def dsigmoid(y): return y * (1 - y)
def tanh(x): return np.tanh(x)
def dtanh(y): return 1 - y ** 2

def softmax(v):
    e = np.exp(v - np.max(v))
    return e / np.sum(e)
```

✓ 0.0s

```python
def sample(h, C, seed_idx, n):
    x = np.zeros((X_size, 1))
    x[seed_idx] = 1
    indices = []
    for t in range(n):
        z = np.vstack((h, x))
        f = sigmoid(Wf @ z + bf)
        i = sigmoid(Wi @ z + bi)
        o = sigmoid(Wo @ z + bo)
        C_bar = tanh(Wc @ z + bc)
        C = f * C + i * C_bar
        h = o * tanh(C)
        y = Wy @ h + by
        p = softmax(y)
        idx = np.random.choice(range(X_size), p=p.ravel())
        x = np.zeros((X_size, 1))
        x[idx] = 1
        indices.append(idx)
    return ''.join(idx_to_char[i] for i in indices)
```

✓ 0.0s

```python
h = np.zeros((H_size, 1))
C = np.zeros((H_size, 1))
smooth_loss = -np.log(1.0 / X_size) * T_steps

for iteration in range(1000):
    if iteration * T_steps + T_steps + 1 >= len(data):
        h = np.zeros((H_size, 1))
        C = np.zeros((H_size, 1))
        continue

    inputs = [char_to_idx[ch] for ch in data[iteration*T_steps : iteration*T_steps+T_steps]]
    targets = [char_to_idx[ch] for ch in data[iteration*T_steps+1 : iteration*T_steps+T_steps+1]]

    xs, hs, Cs, ys, ps = {}, {}, {}, {}, {}
    hs[-1] = np.copy(h)
    Cs[-1] = np.copy(C)
    loss = 0

    # Forward
    for t in range(T_steps):
        xs[t] = np.zeros((X_size, 1))
        xs[t][inputs[t]] = 1
        z = np.vstack((hs[t-1], xs[t]))
        f = sigmoid(Wf @ z + bf)
        i = sigmoid(Wi @ z + bi)
        o = sigmoid(Wo @ z + bo)
        C_bar = tanh(Wc @ z + bc)
        Cs[t] = f * Cs[t-1] + i * C_bar
        hs[t] = o * tanh(Cs[t])
        ys[t] = Wy @ hs[t] + by
        ps[t] = softmax(ys[t])
        loss += -np.log(ps[t][targets[t], 0])
```

```
    # Backward
    dWf = np.zeros_like(Wf)
    dWi = np.zeros_like(Wi)
    dWo = np.zeros_like(Wo)
    dWc = np.zeros_like(Wc)
    dWy = np.zeros_like(Wy)
    dbf = np.zeros_like(bf)
    dbi = np.zeros_like(bi)
    dbo = np.zeros_like(bo)
    dbc = np.zeros_like(bc)
    dby = np.zeros_like(by)
    dh_next = np.zeros_like(h)
    dC_next = np.zeros_like(C)

    for t in reversed(range(T_steps)):
        dy = np.copy(ps[t])
        dy[targets[t]] -= 1
        dWy += dy @ hs[t].T
        dby += dy
        dh = Wy.T @ dy + dh_next
        do = dh * tanh(Cs[t]) * dsigmoid(o)
        dWo += do @ np.vstack((hs[t-1], xs[t])).T
        dbo += do
        dC = dC_next + dh * o * dtanh(tanh(Cs[t]))
        dC_bar = dC * i * dtanh(C_bar)
        dWc += dC_bar @ np.vstack((hs[t-1], xs[t])).T
        dbc += dC_bar
        di = dC * C_bar * dsigmoid(i)
        dWi += di @ np.vstack((hs[t-1], xs[t])).T
        dbi += di
        df = dC * Cs[t-1] * dsigmoid(f)
        dWf += df @ np.vstack((hs[t-1], xs[t])).T
        dbf += df
        dz = Wf.T @ df + Wi.T @ di + Wc.T @ dC_bar + Wo.T @ do
        dh_next = dz[:H_size, :]
        dC_next = f * dC
```

```
    for param, dparam in zip([Wf, Wi, Wo, Wc, Wy, bf, bi, bo, bc, by],
                             [dWf, dWi, dWo, dWc, dWy, dbf, dbi, dbo, dbc, dby]):
        np.clip(dparam, -5, 5, out=dparam)
        param -= learning_rate * dparam

    smooth_loss = 0.999 * smooth_loss + 0.001 * loss
    if iteration % 100 == 0:
        print(f"Iteracja {iteration}, Loss: {smooth_loss:.4f}")
        print("--- SAMPLE ---")
        print(sample(h, C, inputs[0], 200))
        print("-------------")
✓ 0.0s
```

```
Iteracja 0, Loss: 88.8837
--- SAMPLE ---
tr)illTonem.GAGI(albktw)fu,.Gn' rqbq(IoqpG  copAemzusmAypmecoTidgfTvIkflipfevyTt . orecTt,( v sg')k (feTpzf qiggqg,hAed.ikpcqpwg)c-zaz'ph.oubfvh IycIrI(hokk-ziupiweotf)yeGwfebswt.qgoaqpcuav'whfi ,cbsA'
-------------
```

3. Wnioski

Zrealizowane badanie wykazało, że sieci LSTM znacząco usprawniają modelowanie zależności w danych sekwencyjnych, szczególnie w kontekście problemu zanikającego gradientu, który w tradycyjnych RNN skutecznie hamuje proces nauki. Dzięki zastosowanym bramkom model był w stanie świadomie przechowywać lub pomijać informacje, co pozytywnie wpłynęło na dokładność przewidywania kolejnych znaków w tekście, zwłaszcza w miarę postępu treningu. Kluczową rolę odegrała tu metoda BPTT, umożliwiająca przekazywanie informacji o błędach wstecz przez całą sekwencję, co zapewniło precyzyjną aktualizację wag, zarówno w krótkim, jak i długim horyzoncie czasowym.

Wyniki doświadczenia pokazały również, jak duże znaczenie ma odpowiednie zainicjalizowanie wag — dostosowane do charakteru używanych funkcji aktywacji — w kontekście szybkości zbieżności algorytmu. Połączenie architektury LSTM z metodą BPTT okazało się zatem skutecznym rozwiązaniem w konstruowaniu modeli zdolnych do analizy długich ciągów danych i wykrywania w nich subtelnych, długoterminowych relacji. Mimo złożoności implementacyjnej, rozwiązanie to sprawdza się znakomicie w zadaniach takich jak przetwarzanie języka naturalnego czy predykcja sekwencji.