

# SPRAWOZDANIE

Zajęcia: Matematyka Konkretna

Prowadzący: prof. dr hab. Vasyl Martsenyuk

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

## 1. Polecenie:

Link do repozytorium: [https://github.com/Maciek332/Semestr\\_3\\_Nycz/tree/main/MK](https://github.com/Maciek332/Semestr_3_Nycz/tree/main/MK)

Celem przeprowadzonego zadania było zaimplementowanie oraz przeanalizowanie działania sieci LSTM trenowanej metodą propagacji wstecznej w czasie (BPTT) w kontekście przetwarzania sekwencji tekstowych. Sieci LSTM, czyli Long Short-Term Memory, zostały zaprojektowane jako rozszerzenie klasycznych sieci rekurencyjnych, aby skuteczniej radzić sobie z problemem zanikającego gradientu, który uniemożliwia klasycznym RNN uczenie się długoterminowych zależności w danych sekwencyjnych. Dzięki wprowadzeniu mechanizmów bramkowania, w tym bramy zapominania, bramy wejścia i bramy wyjścia, sieć LSTM potrafi decydować, które informacje należy przechować w stanie wewnętrznym, a które usunąć, co pozwala jej lepiej zapamiętywać istotne fragmenty sekwencji.

Uczenie takiej sieci wymaga specjalnego podejścia do obliczania gradientów ze względu na rozciągnięcie procesu w czasie. Metoda Backpropagation Through Time umożliwia rozłożenie błędu wzdłuż kolejnych kroków czasowych i właściwe obliczenie wpływu każdego z nich na aktualizację wag. W trakcie realizacji zadania zastosowano funkcje aktywacji typu sigmoid oraz tangens hiperboliczny wraz z ich pochodnymi, co pozwoliło prawidłowo obliczać propagację sygnału i gradientów. Wagi sieci zostały zainicjalizowane losowo według rozkładu normalnego, przy czym dla bram sigmoid wprowadzono przesunięcie średniej, aby lepiej kontrolować początkowe wartości aktywacji, co ułatwia start procesu uczenia.

umożliwiający ich wejście do sieci oraz późniejszą dekodację wyników predykcji. Dzięki temu można było przeprowadzić proces trenowania, w którym sieć starała się przewidzieć kolejne znaki na podstawie poprzedniego kontekstu, sukcesywnie poprawiając swoje parametry w wyniku minimalizacji funkcji kosztu.

## 2. Opis programu opracowanego

```
text = (
    "Computer scientists and philosophers have since suggested "
    "that AI may become an existential risk to humanity if its "
    "rational capacities are not steered towards beneficial goals"
)

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

✓ 0.0s
```

```
data = open("example2.txt", "r").read()
chars = list(set(data))
X_size = len(chars)
char_to_idx = {ch: i for i, ch in enumerate(chars)}
idx_to_char = {i: ch for i, ch in enumerate(chars)}

✓ 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
```

```

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

```

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("-----")

```

```

Iteracja 0, Loss: 82.3959
--- SAMPLE ---
hIAmlixeo IirgyCbAprhGA ceIIeoAma cgvoYAlnhuKtLAldydxleclIIIfvfvavkfcdosgmsyb tsepbvxwInlCdoekoeCcihYaCo hhCnkf xtenthbyueIAI ddrnxmpcAmhprIweCmfvcuhvyrhkdifhf moeegykaolneiuvamxt
-----

```

### 3. Wnioski

Przeprowadzone doświadczenie pozwoliło zaobserwować, że zastosowanie sieci LSTM znacząco ułatwia modelowanie zależności w danych sekwencyjnych i pozwala lepiej radzić sobie z problemem zanikających gradientów, co w przypadku klasycznych RNN często uniemożliwia efektywne uczenie. Dzięki mechanizmom bramkowania sieć mogła selektywnie zapamiętywać lub odrzucać informacje, co przełożyło się na jej zdolność do przewidywania kolejnych znaków w tekście z rosnącą dokładnością w miarę postępów treningu. Implementacja metody BPTT okazała się kluczowa, ponieważ umożliwiła propagację informacji o błędzie wstecz przez całą sekwencję, co pozwoliło dokładnie zaktualizować wagi w taki sposób, aby sieć minimalizowała błąd nie tylko w najbliższym kroku, ale w całym rozważanym horyzoncie czasowym. Wnioskiem płynącym z tego zadania jest również to, że odpowiednia inicjalizacja wag, uwzględniająca charakter funkcji aktywacji, ma istotny wpływ na szybkość zbiegania się algorytmu. Zastosowanie LSTM w połączeniu z BPTT umożliwia zatem skuteczne uczenie modeli, które muszą analizować długie ciągi danych i wychwytywać w nich subtelne zależności. Wyniki zadania pokazują, że mimo złożoności implementacyjnej oraz konieczności uwzględnienia wielu detali, sieci tego typu są niezwykle przydatne w zadaniach związanych z przetwarzaniem języka naturalnego i predykcją sekwencji.