Zajęcie 5: Metoda BPTT dla sieci LSTM

Importowanie

In [1]:
```
import numpy as np
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from IPython import display
plt.style.use('seaborn-v0_8-white')
```

Odczytywanie i przetwarzanie danych

In [2]:
```
data = """Artificial intelligence was founded as an academic discipline in 1956, an
data = data.replace("\n", " ")
```

Przetwarzaj dane i obliczaj indeksy

In [3]:
```
chars = list(set(data))
data_size, X_size = len(data), len(chars)
print("data has %d characters, %d unique" % (data_size, X_size))
char_to_idx = {ch:i for i,ch in enumerate(chars)}
idx_to_char = {i:ch for i,ch in enumerate(chars)}
```

```
data has 273 characters, 34 unique
```

Stałe i hiperparametry

In [4]:
```
H_size = 64 # Size of the hidden layer
T_steps = 40 # Number of time steps (length of the sequence) used for training
learning_rate = 5e-2 # Learning rate
weight_sd = 0.1 # Standard deviation of weights for initialization
z_size = H_size + X_size # Size of concatenate(H, X) vector
```

Funkcje aktywacji i pochodne

Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

$$\frac{d\sigma(x)}{dx} = \sigma(x) \cdot (1 - \sigma(x)) \tag{2}$$

Tanh

$$\frac{d\tanh(x)}{dx} = 1 - \tanh^2(x) \tag{3}$$

In [5]:
```
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 * y
```

Parametry

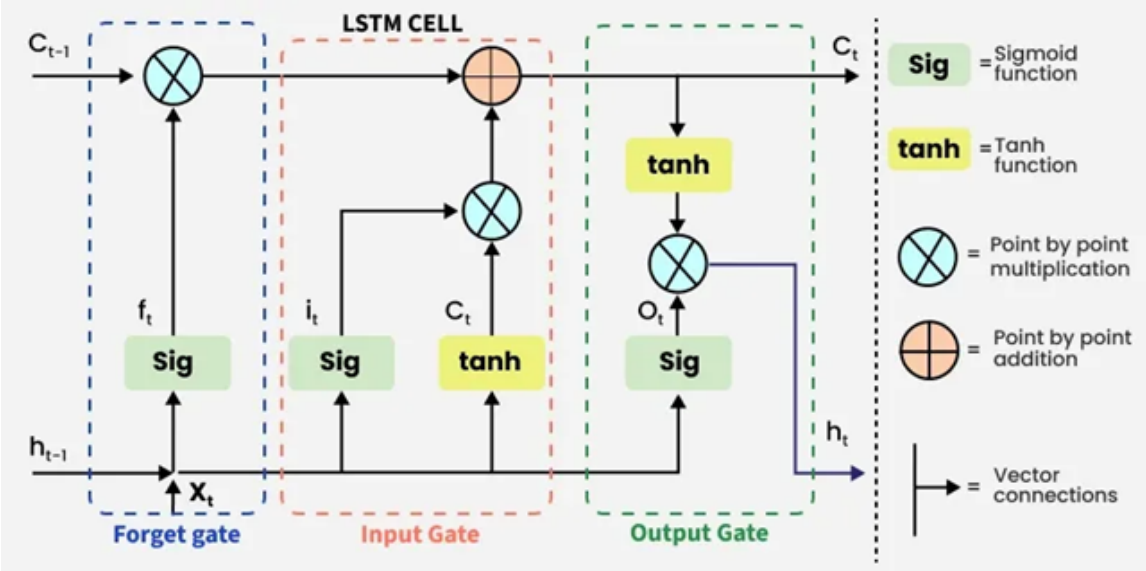
In [6]:
```
class Param:
    def __init__(self, name, value):
        self.name = name
        self.v = value #parameter value
        self.d = np.zeros_like(value) #derivative
        self.m = np.zeros_like(value) #momentum for AdaGrad
```

Używamy losowych wag z rozkładem normalnym (`0` , `weight_sd`) dla funkcji aktywacji $tanh$ i (`0.5` , `weight_sd`) dla funkcji aktywacji $sigmoid$.

Odchylenia są inicjowane do zera.

In [7]:
```
class Parameters:
    def __init__(self):
        self.W_f = Param('W_f',
                         np.random.randn(H_size, z_size) * weight_sd + 0.5)
        self.b_f = Param('b_f',
                         np.zeros((H_size, 1)))

        self.W_i = Param('W_i',
                         np.random.randn(H_size, z_size) * weight_sd + 0.5)
        self.b_i = Param('b_i',
                         np.zeros((H_size, 1)))

        self.W_C = Param('W_C',
                         np.random.randn(H_size, z_size) * weight_sd)
        self.b_C = Param('b_C',
                         np.zeros((H_size, 1)))

        self.W_o = Param('W_o',
                         np.random.randn(H_size, z_size) * weight_sd + 0.5)
        self.b_o = Param('b_o',
                         np.zeros((H_size, 1)))

        #For final layer to predict the next character
```

```
        self.W_v = Param('W_v',
                         np.random.randn(X_size, H_size) * weight_sd)
        self.b_v = Param('b_v',
                         np.zeros((X_size, 1)))

    def all(self):
        return [self.W_f, self.W_i, self.W_C, self.W_o, self.W_v,
                self.b_f, self.b_i, self.b_C, self.b_o, self.b_v]

parameters = Parameters()
```



Obliczanie do przodu

Operacja z to konkatenacja x i h_{t-1}

Konkatenacja h_{t-1} i x_t

$$z = [h_{t-1}, x_t] \tag{4}$$

Funkcje LSTM

$$f_t = \sigma(W_f \cdot z + b_f) \tag{5}$$
$$i_t = \sigma(W_i \cdot z + b_i) \tag{6}$$
$$\bar{C}_t = tanh(W_C \cdot z + b_C) \tag{7}$$
$$C_t = f_t * C_{t-1} + i_t * \bar{C}_t \tag{8}$$
$$o_t = \sigma(W_o \cdot z + b_t) \tag{9}$$
$$h_t = o_t * tanh(C_t) \tag{10}$$

Logits

$$v_t = W_v \cdot h_t + b_v \tag{11}$$

Softmax

$$\hat{y}_t = \text{softmax}(v_t) \tag{12}$$

$\hat{y}_t$ to `y` w kodzie i y_t to `targets`.

In [8]:
```
def forward(x, h_prev, C_prev, p = parameters):
    assert x.shape == (X_size, 1)
    assert h_prev.shape == (H_size, 1)
    assert C_prev.shape == (H_size, 1)

    z = np.row_stack((h_prev, x))
    f = sigmoid(np.dot(p.W_f.v, z) + p.b_f.v)
    i = sigmoid(np.dot(p.W_i.v, z) + p.b_i.v)
    C_bar = tanh(np.dot(p.W_C.v, z) + p.b_C.v)

    C = f * C_prev + i * C_bar
    o = sigmoid(np.dot(p.W_o.v, z) + p.b_o.v)
    h = o * tanh(C)

    v = np.dot(p.W_v.v, h) + p.b_v.v
    y = np.exp(v) / np.sum(np.exp(v)) #softmax

    return z, f, i, C_bar, C, o, h, v, y
```

Obliczanie wsztecz

Loss

$$L_k = -\sum_{t=k}^{T}\sum_{j} y_{t,j} log \hat{y_{t,j}} \tag{13}$$

$$L = L_1 \tag{14}$$

Gradients

$$
\begin{aligned}
dv_t &= \hat{y}_t - y_t && (15)\\
dh_t &= dh'_t + W_y^T \cdot dv_t && (16)\\
do_t &= dh_t * \tanh(C_t) && (17)\\
dC_t &= dC'_t + dh_t * o_t * (1 - \tanh^2(C_t)) && (18)\\
d\bar{C}_t &= dC_t * i_t && (19)\\
di_t &= dC_t * \bar{C}_t && (20)\\
df_t &= dC_t * C_{t-1} && (21)\\
& && (22)\\
df'_t &= f_t * (1 - f_t) * df_t && (23)\\
di'_t &= i_t * (1 - i_t) * di_t && (24)\\
d\bar{C}'_{t-1} &= (1 - \bar{C}_t^2) * d\bar{C}_t && (25)\\
do'_t &= o_t * (1 - o_t) * do_t && (26)\\
dz_t &= W_f^T \cdot df'_t && (27)\\
&\quad + W_i^T \cdot di_t && (28)\\
&\quad + W_C^T \cdot d\bar{C}_t && (29)\\
&\quad + W_o^T \cdot do_t && (30)\\
& && (31)\\
[dh'_{t-1}, dx_t] &= dz_t && (32)\\
dC'_t &= f_t * dC_t && (33)
\end{aligned}
$$

- $dC'_t = \frac{\partial L_{t+1}}{\partial C_t}$ and $dh'_t = \frac{\partial L_{t+1}}{\partial h_t}$
- $dC_t = \frac{\partial L}{\partial C_t} = \frac{\partial L_t}{\partial C_t}$ and $dh_t = \frac{\partial L}{\partial h_t} = \frac{\partial L_t}{\partial h_t}$
- All other derivatives are of L
- `target` is target character index y_t
- `dh_next` is dh'_t (size H x 1)
- `dC_next` is dC'_t (size H x 1)
- `C_prev` is C_{t-1} (size H x 1)
- df'_t, di'_t, $d\bar{C}'_t$, and do'_t are *also* assigned to `df`, `di`, `dC_bar`, and `do` in the **code**.
- *Returns* dh_t and dC_t

Gradienty parametrów modelu

$$dW_v = dv_t \cdot h_t^T \tag{34}$$
$$db_v = dv_t \tag{35}$$
$$\tag{36}$$
$$dW_f = df'_t \cdot z^T \tag{37}$$
$$db_f = df'_t \tag{38}$$
$$\tag{39}$$
$$dW_i = di'_t \cdot z^T \tag{40}$$
$$db_i = di'_t \tag{41}$$
$$\tag{42}$$
$$dW_C = d\bar{C}'_t \cdot z^T \tag{43}$$
$$db_C = d\bar{C}'_t \tag{44}$$
$$\tag{45}$$
$$dW_o = do'_t \cdot z^T \tag{46}$$
$$db_o = do'_t \tag{47}$$
$$\tag{48}$$

In [9]:
```
def backward(target, dh_next, dC_next, C_prev,
             z, f, i, C_bar, C, o, h, v, y,
             p = parameters):

    assert z.shape == (X_size + H_size, 1)
    assert v.shape == (X_size, 1)
    assert y.shape == (X_size, 1)

    for param in [dh_next, dC_next, C_prev, f, i, C_bar, C, o, h]:
        assert param.shape == (H_size, 1)

    dv = np.copy(y)
    dv[target] -= 1

    p.W_v.d += np.dot(dv, h.T)
    p.b_v.d += dv

    dh = np.dot(p.W_v.v.T, dv)
    dh += dh_next
    do = dh * tanh(C)
    do = dsigmoid(o) * do
    p.W_o.d += np.dot(do, z.T)
    p.b_o.d += do

    dC = np.copy(dC_next)
    dC += dh * o * dtanh(tanh(C))
    dC_bar = dC * i
    dC_bar = dtanh(C_bar) * dC_bar
    p.W_C.d += np.dot(dC_bar, z.T)
    p.b_C.d += dC_bar

    di = dC * C_bar
    di = dsigmoid(i) * di
    p.W_i.d += np.dot(di, z.T)
    p.b_i.d += di
```

```
    df = dC * C_prev
    df = dsigmoid(f) * df
    p.W_f.d += np.dot(df, z.T)
    p.b_f.d += df

    dz = (np.dot(p.W_f.v.T, df)
         + np.dot(p.W_i.v.T, di)
         + np.dot(p.W_C.v.T, dC_bar)
         + np.dot(p.W_o.v.T, do))
    dh_prev = dz[:H_size, :]
    dC_prev = f * dC

    return dh_prev, dC_prev
```

Oblicznie do przodu i wstecz

Wyczyść gradienty przed każdym obliczniem wstecz

In [10]:
```
def clear_gradients(params = parameters):
    for p in params.all():
        p.d.fill(0)
```

Przycinaj gradienty, aby złagodzić wybuchające gradienty

In [11]:
```
def clip_gradients(params = parameters):
    for p in params.all():
        np.clip(p.d, -1, 1, out=p.d)
```

Oblicz i zapisz wartości w obliczeniu do przodu. Akumuluj gradienty w obliczeniach wstecz i przycinaj gradienty, aby uniknąć wybuchających gradientów.

- `input` , `target` to lista liczb całkowitych z indeksami znaków.
- `h_prev` jest tablicą początkowych `h` dla h_{-1} (size H x 1)
- `C_prev` jest tablicą inicjałów `C` dla C_{-1} (size H x 1)
- *Returns* loss, final h_T and C_T

In [12]:
```
def forward_backward(inputs, targets, h_prev, C_prev):
    global paramters

    # To store the values for each time step
    x_s, z_s, f_s, i_s,  = {}, {}, {}, {}
    C_bar_s, C_s, o_s, h_s = {}, {}, {}, {}
    v_s, y_s =  {}, {}

    # Values at t - 1
    h_s[-1] = np.copy(h_prev)
    C_s[-1] = np.copy(C_prev)

    loss = 0
    # Loop through time steps
    assert len(inputs) == T_steps
```

```
    correct = 0

    for t in range(len(inputs)):
        x_s[t] = np.zeros((X_size, 1))
        x_s[t][inputs[t]] = 1 # Input character

        (z_s[t], f_s[t], i_s[t],
        C_bar_s[t], C_s[t], o_s[t], h_s[t],
        v_s[t], y_s[t]) = \
            forward(x_s[t], h_s[t - 1], C_s[t - 1]) # Forward pass

        pred = np.argmax(y_s[t])
        correct += int(pred == targets[t])

        loss += -np.log(y_s[t][targets[t], 0]) # Loss for at t

    clear_gradients()

    dh_next = np.zeros_like(h_s[0]) #dh from the next character
    dC_next = np.zeros_like(C_s[0]) #dh from the next character

    for t in reversed(range(len(inputs))):
        # Backward pass
        dh_next, dC_next = \
            backward(target = targets[t], dh_next = dh_next,
                     dC_next = dC_next, C_prev = C_s[t-1],
                     z = z_s[t], f = f_s[t], i = i_s[t], C_bar = C_bar_s[t],
                     C = C_s[t], o = o_s[t], h = h_s[t], v = v_s[t],
                     y = y_s[t])

    clip_gradients()

    return loss, h_s[len(inputs) - 1], C_s[len(inputs) - 1], correct / len(inputs)
```

Wypróbuj następną literę

In [13]:

```
def sample(h_prev, C_prev, first_char_idx, sentence_length):
    x = np.zeros((X_size, 1))
    x[first_char_idx] = 1

    h = h_prev
    C = C_prev

    indexes = []

    for t in range(sentence_length):
        _, _, _, _, C, _, h, _, p = forward(x, h, C)
        idx = np.random.choice(range(X_size), p=p.ravel())
        x = np.zeros((X_size, 1))
        x[idx] = 1
        indexes.append(idx)

    return indexes
```

Uczenie (Adagrad)

Zaktualizuj wykres i wyświetl przykładowe dane wyjściowe

In [14]:
```
def update_status(inputs, h_prev, C_prev):
    #initialized later
    global plot_iter, plot_loss
    global smooth_loss

    # Get predictions for 200 letters with current model

    sample_idx = sample(h_prev, C_prev, inputs[0], 200)
    txt = ''.join(idx_to_char[idx] for idx in sample_idx)

    # Clear and plot
    plt.plot(plot_iter, plot_loss)
    display.clear_output(wait=True)
    plt.show()

    #Print prediction and loss
    print("----\n %s \n----" % (txt, ))
    print("iter %d, loss %f" % (iteration, smooth_loss))
```

Zaktualizuj parametry

$$\theta_i = \theta_i - \eta \frac{d\theta_i}{\sum dw_\tau^2} \tag{49}$$

$$d\theta_i = \frac{\partial L}{\partial \theta_i} \tag{50}$$

In [15]:
```
def update_paramters(params = parameters):
    for p in params.all():
        p.m += p.d * p.d # Calculate sum of gradients
        #print(learning_rate * dparam)
        p.v += -(learning_rate * p.d / np.sqrt(p.m + 1e-8))
```

Aby opóźnić przerwanie klawiatury, aby zapobiec zatrzymaniu uczenia w trakcie iteracji

In [16]:
```
import signal

class DelayedKeyboardInterrupt(object):
    def __enter__(self):
        self.signal_received = False
        self.old_handler = signal.signal(signal.SIGINT, self.handler)

    def handler(self, sig, frame):
        self.signal_received = (sig, frame)
        print('SIGINT received. Delaying KeyboardInterrupt.')

    def __exit__(self, type, value, traceback):
        signal.signal(signal.SIGINT, self.old_handler)
```

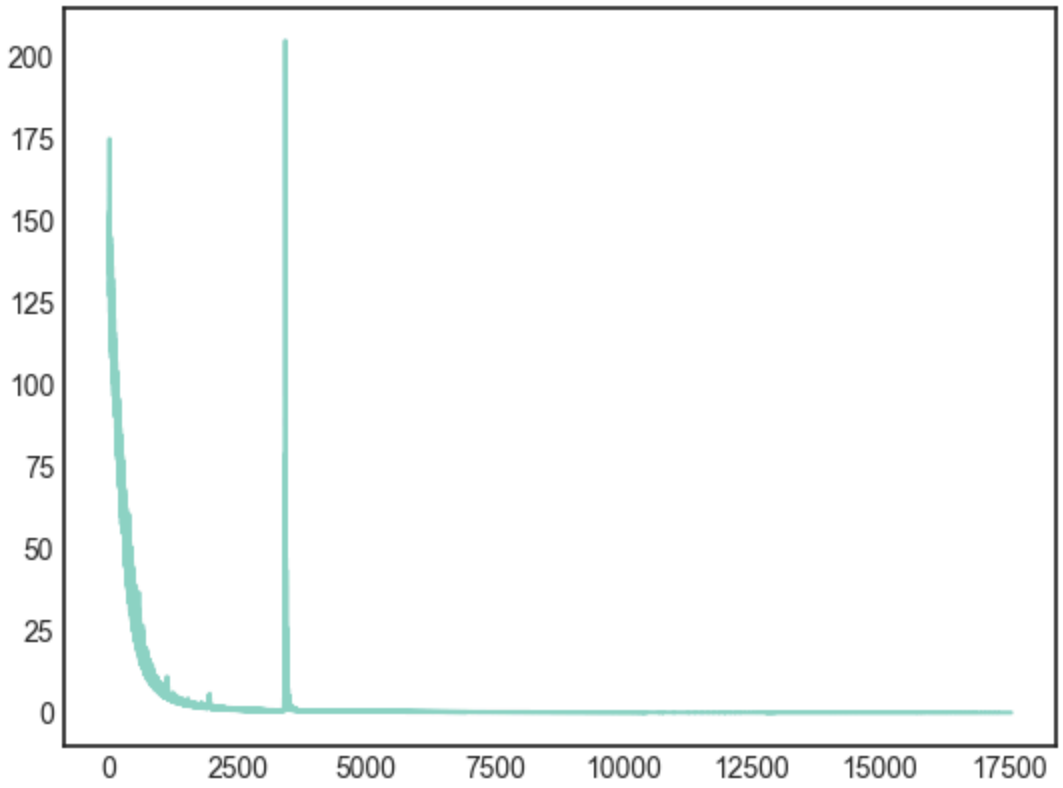
```
        if self.signal_received:
            self.old_handler(*self.signal_received)
```

In [17]:
```
# Exponential average of loss
# Initialize to a error of a random model
smooth_loss = -np.log(1.0 / X_size) * T_steps

iteration, pointer = 0, 0

# For the graph
plot_iter = np.zeros((0))
plot_loss = np.zeros((0))
```

Pętla uczenia

In [18]:
```
target_loss = 0.1
max_iters = 20000

while True:
    try:
        with DelayedKeyboardInterrupt():
            # Reset
            if pointer + T_steps >= len(data) or iteration == 0:
                g_h_prev = np.zeros((H_size, 1))
                g_C_prev = np.zeros((H_size, 1))
                pointer = 0


            inputs = ([char_to_idx[ch]
                       for ch in data[pointer: pointer + T_steps]])
            targets = ([char_to_idx[ch]
                        for ch in data[pointer + 1: pointer + T_steps + 1]])

            loss, g_h_prev, g_C_prev, acc = forward_backward(inputs, targets, g_h_p
            smooth_loss = smooth_loss * 0.999 + loss * 0.001

            if smooth_loss < 0.1:
                print("Osiągnięto smooth_loss < 0.1, stop.")
                update_status(inputs, g_h_prev, g_C_prev)
                break

            # Print every hundred steps
            if iteration % 100 == 0:
                update_status(inputs, g_h_prev, g_C_prev)

            update_paramters()

            plot_iter = np.append(plot_iter, [iteration])
            plot_loss = np.append(plot_loss, [loss])

            pointer += T_steps
            iteration += 1
    except KeyboardInterrupt:
        update_status(inputs, g_h_prev, g_C_prev)
        break
```

```
if iteration >= max_iters:
    print(f"\nnie osiągnięto loss <= {target_loss} w {max_iters} iteracjach. Ostatn
```




```
----
  acaver, f ofline an "AI winter), folide an as wanger,fool wad by new approacheehe
fowedeas aasacanc is has experince ee ye by discipline in ye ye 1956, ate erit funee
ps by nee f wan an "I6ras incede
----
iter 17550, loss 0.099974
```

Sprawdzanie gradientu

Przybliż gradienty numeryczne, zmieniając parametry i uruchamiając model. Sprawdź, czy przybliżone gradienty są równe obliczonym gradientom analitycznym (przez wsteczną propagację).

Wypróbuj to na `num_checks` pojedynczych parametrach wybranych losowo dla każdej macierzy wag i wektora odchylenia.

In [24]:
```
from random import uniform
```

Oblicz gradient numeryczny

In [33]:
```
def calc_numerical_gradient(param, idx, delta, inputs, target, h_prev, C_prev):
    old_val = param.v.flat[idx]

    # evaluate loss at [x + delta] and [x - delta]
    param.v.flat[idx] = old_val + delta
```

```
    loss_plus_delta, _, _, _ = forward_backward(inputs, targets,
                                                h_prev, C_prev)
    param.v.flat[idx] = old_val - delta
    loss_mins_delta, _, _, _ = forward_backward(inputs, targets,
                                                h_prev, C_prev)

    param.v.flat[idx] = old_val #reset

    grad_numerical = (loss_plus_delta - loss_mins_delta) / (2 * delta)
    # Clip numerical error because analytical gradient is clipped
    [grad_numerical] = np.clip([grad_numerical], -1, 1)

    return grad_numerical
```

Sprawdź gradient każdej macierzy parametrów/wektora przy poszczególnych wartościach `num_checks`

In [34]:
```
def gradient_check(num_checks, delta, inputs, target, h_prev, C_prev):
    global parameters

    # To calculate computed gradients
    _, _, _, _ =  forward_backward(inputs, target, h_prev, C_prev)

    for param in parameters.all():
        #Make a copy because this will get modified
        d_copy = np.copy(param.d)

        # Test num_checks times
        for i in range(num_checks):
            # Pick a random index
            rnd_idx = int(uniform(0, param.v.size))

            grad_numerical = calc_numerical_gradient(param,
                                                     rnd_idx,
                                                     delta,
                                                     inputs,
                                                     target,
                                                     h_prev, C_prev)
            grad_analytical = d_copy.flat[rnd_idx]

            err_sum = abs(grad_numerical + grad_analytical) + 1e-09
            rel_error = abs(grad_analytical - grad_numerical) / err_sum

            # If relative error is greater than 1e-06
            if rel_error > 1e-06:
                print('%s (%e, %e) => %e'
                      % (param.name, grad_numerical, grad_analytical, rel_error))
```

In [35]:
```
gradient_check(10, 1e-5, inputs, targets, g_h_prev, g_C_prev)
```

```
C:\Users\gsiwi\AppData\Local\Temp\ipykernel_18448\2858406474.py:6: DeprecationWarnin
g: `row_stack` alias is deprecated. Use `np.vstack` directly.
  z = np.row_stack((h_prev, x))
```

```
W_f (-9.237056e-09, -8.823301e-09) => 2.170761e-02
W_i (-1.674517e-01, -1.674521e-01) => 1.244992e-06
W_i (0.000000e+00, 5.928628e-11) => 5.596814e-02
W_i (2.507239e-01, 2.507202e-01) => 7.431671e-06
W_i (6.156387e-01, 6.156413e-01) => 2.068704e-06
W_C (2.190603e-06, 2.189994e-06) => 1.391097e-04
W_C (-8.026646e-05, -8.026731e-05) => 5.317533e-06
W_v (-2.543507e-03, -2.543500e-03) => 1.335008e-06
W_v (1.636954e-01, 1.636960e-01) => 1.793606e-06
W_v (-1.599935e-03, -1.599942e-03) => 2.330271e-06
b_f (3.776037e-05, 3.775995e-05) => 5.656441e-06
b_f (4.791000e-02, 4.791026e-02) => 2.710638e-06
b_i (1.382385e-01, 1.382395e-01) => 3.479576e-06
b_i (-5.254840e-01, -5.254821e-01) => 1.846364e-06
b_i (-8.703773e-02, -8.703729e-02) => 2.520720e-06
b_i (-2.453007e-05, -2.453172e-05) => 3.369669e-05
b_i (-1.267210e-03, -1.267108e-03) => 4.025325e-05
b_i (0.000000e+00, -1.573578e-10) => 1.359630e-01
b_i (-2.436385e-01, -2.436439e-01) => 1.097035e-05
b_i (2.045393e-01, 2.045379e-01) => 3.428996e-06
b_C (-1.030287e-07, -1.041748e-07) => 5.504781e-03
b_C (1.902976e-05, 1.902872e-05) => 2.723940e-05
b_o (1.226092e-01, 1.226052e-01) => 1.621285e-05
b_o (-2.566078e-02, -2.566068e-02) => 1.870796e-06
b_o (8.916395e-02, 8.910524e-02) => 3.293320e-04
b_o (2.326673e-01, 2.326488e-01) => 3.972321e-05
b_o (-2.566078e-02, -2.566068e-02) => 1.870796e-06
b_o (-2.123582e-01, -2.124193e-01) => 1.439448e-04
b_v (1.179180e-01, 1.179177e-01) => 1.514285e-06
b_v (2.295819e-02, 2.295829e-02) => 2.211554e-06
b_v (1.179180e-01, 1.179177e-01) => 1.514285e-06
b_v (1.131956e-01, 1.131960e-01) => 2.029268e-06
```

Zadanie

Opracować sieć LSTM w celu nauczanie się tekstu z dokładnością 0.1 Warianty zadania:

1. "Artificial intelligence (AI) is intelligence—perceiving, synthesizing, and inferring information—demonstrated by machines, as opposed to intelligence displayed by non-human animals or by humans"
2. "Example tasks in which this is done include speech recognition, computer vision, translation between (natural) languages, as well as other mappings of inputs"
3. "AI applications include advanced web search engines (e.g., Google Search), recommendation systems (used by YouTube, Amazon, and Netflix), understanding human speech (such as Siri and Alexa), self-driving cars (e.g., Waymo), generative or creative tools (ChatGPT and AI art), automated decision-making, and competing at the highest level in strategic game systems (such as chess and Go)"
4. "As machines become increasingly capable, tasks considered to require "intelligence" are often removed from the definition of AI, a phenomenon known as the AI effect"

5. "Artificial intelligence was founded as an academic discipline in 1956, and in the years since it has experienced several waves of optimism,followed by disappointment and the loss of funding (known as an "AI winter"), followed by new approaches, success, and renewed funding"
6. "AI research has tried and discarded many different approaches, including simulating the brain, modeling human problem solving, formal logic, large databases of knowledge, and imitating animal behavior"
7. "In the first decades of the 21st century, highly mathematical and statistical machine learning has dominated the field, and this technique has proved highly successful, helping to solve many challenging problems throughout industry and academia"
8. "The various sub-fields of AI research are centered around particular goals and the use of particular tools. The traditional goals of AI research include reasoning, knowledge representation, planning, learning, natural language processing, perception, and the ability to move and manipulate objects"
9. "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"
10. " 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"