인공지능연구 1차 발표

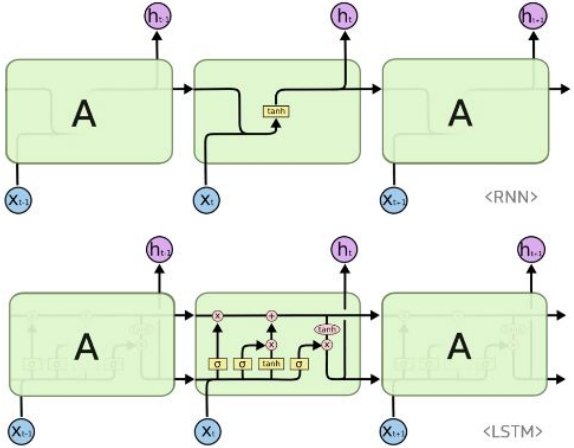
이용정, 박동우, 김영록

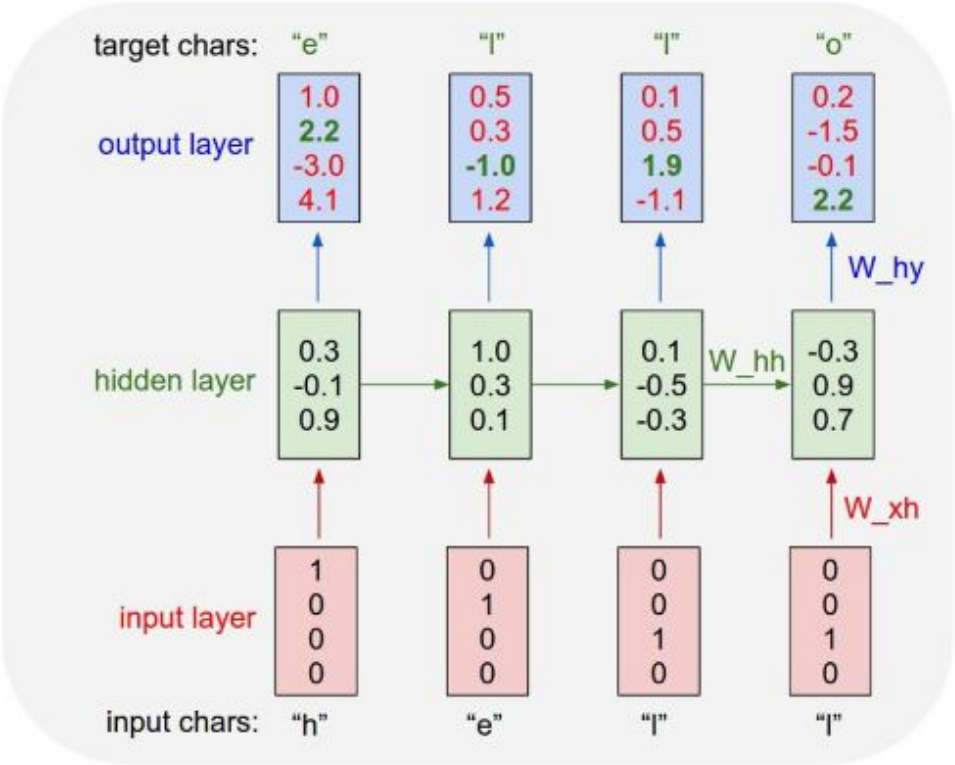
발표순서

LSTM => seq2seq => Transformer

1. LSTM

RNN과 LSTM

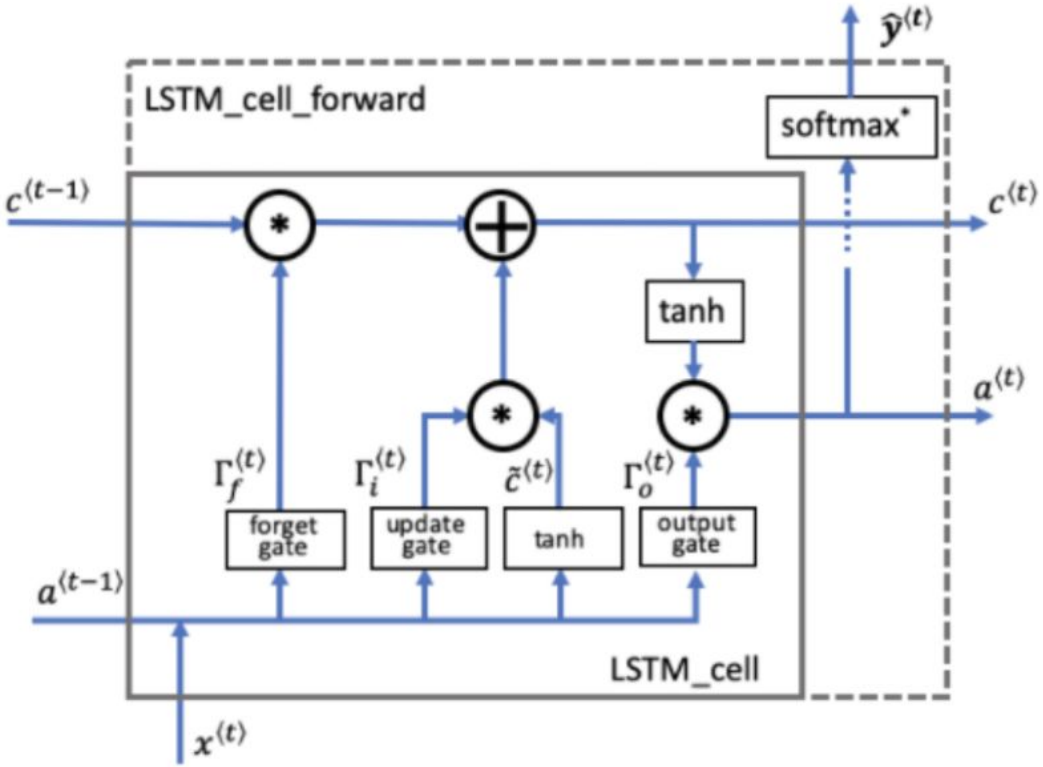




1. LSTM

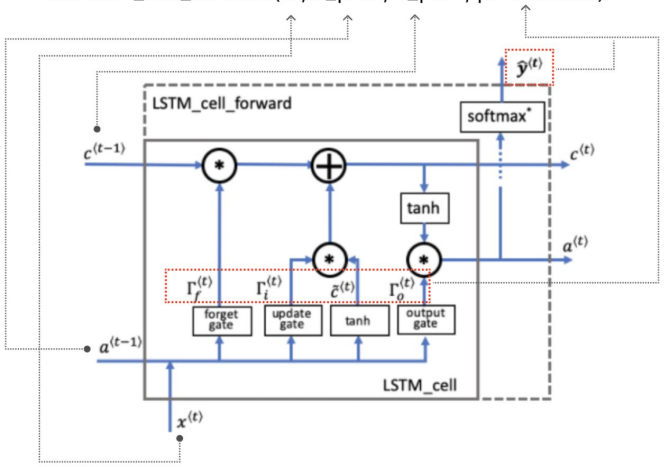
LSTM의 구조

=> gate, 거리가 먼 가중치 소멸 방지



1. LSTM

def lstm_cell_forward(xt, a_prev, c_prev, parameters):



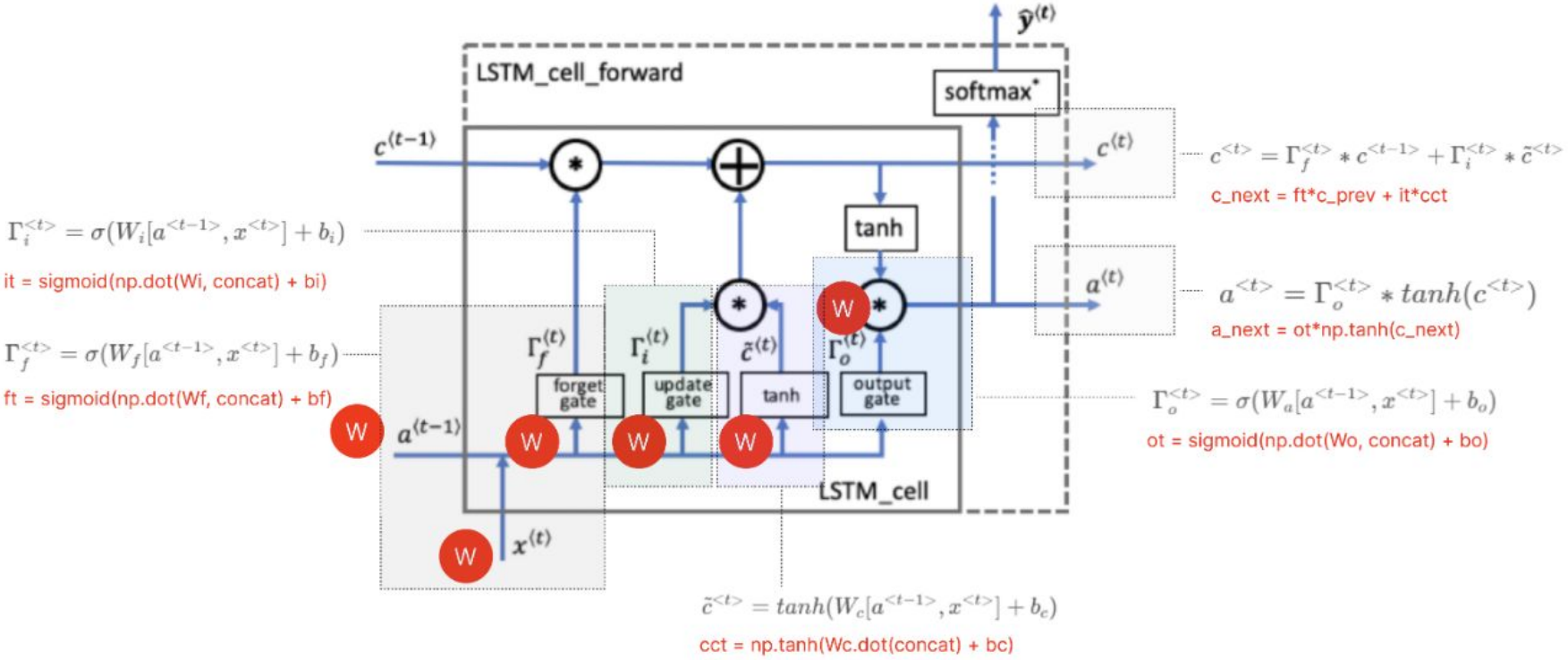

```
# LSTM 기본 셀 구현 코드

def lstm_cell_forward (xt, a_prev, c_prev, parameters):
    """
    Arguments:
    xt -- your input data at timestep "t", numpy array of shape (n_x, m).
    a_prev -- Hidden state at timestep "t-1", numpy array of shape (n_a, m)
    c_prev -- Memory state at timestep "t-1", numpy array of shape (n_a, m)

    parameters:
    Wf => 'Forget Gate' 에서는 이전 시간 h_prev값을 기억할지 말지를 결정한다.
    Wc => 'Candidate value' 는 다음 h_next에 후보군과 정보를 포함하는 TENSOR.
    Wi =>'Update gate' 는 Update gate는 cell state에 추가할 candidate의 정보를 결정하는 역할,
    Wo => 'Output gate' 는 현재 time step의 예측값으로 전달되는 대상,
    'Hidden State' - LSTM cell 의 다음 time step에 전달되는 값을 얻으며, 동시에 예측값 y를 구하기
위함
    """
    # Retrieve parameters from "parameters"
    Wf = parameters["Wf"] # forget gate weight
    bf = parameters["bf"]
    Wi = parameters["Wi"] # update gate weight (notice the variable name)
    bi = parameters["bi"] # (notice the variable name)
    Wc = parameters["Wc"] # candidate value weight
    bc = parameters["bc"]
    Wo = parameters["Wo"] # output gate weight
    bo = parameters["bo"]
    Wy = parameters["Wy"] # prediction weight
    by = parameters["by"]

    # Retrieve dimensions from shapes of xt and Wy
    n_x, m = xt.shape
    n_y, n_a = Wy.shape
```

1. LSTM

$$
\begin{aligned}
f_t &= \sigma(W_{xh\_f}x_t + W_{hh\_f}h_{t-1} + b_{h\_f}) \\
i_t &= \sigma(W_{xh\_i}x_t + W_{hh\_i}h_{t-1} + b_{h\_i}) \\
o_t &= \sigma(W_{xh\_o}x_t + W_{hh\_o}h_{t-1} + b_{h\_o}) \\
g_t &= \tanh(W_{xh\_g}x_t + W_{hh\_g}h_{t-1} + b_{h\_g}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

```
    ### START CODE HERE ###
    # Concatenate a_prev and xt (≈1 line)
    # Concatenate =>
    # a = np.array([[1, 2], [3, 4]])
    # b = np.array([[7, 8], [9, 10]])
    # print(np.concatenate((a, b), axis=0))
    # [[1 2]
    #  [3 4]
    #  [7 8]
    #  [9 10]]

    concat = np.concatenate((a_prev, xt), axis= 0)
    # Compute values for ft (forget gate), it (update gate),
    # cct (candidate value), c_next (cell state),
    # ot (output gate), a_next (hidden state) (≈6 lines)
    ft = sigmoid(np.dot(Wf, concat) + bf)    # forget gate
    it = sigmoid(np.dot(Wi, concat) + bi)    # update gate
    cct = np.tanh(Wc.dot(concat) + bc)    # candidate value
    c_next = ft*c_prev + it*cct    # cell state
    ot = sigmoid(np.dot(Wo, concat) + bo)    # output gate
    a_next = ot*np.tanh(c_next)    # hidden state

    # Compute prediction of the LSTM cell (≈1 line)
    yt_pred = softmax(np.dot(Wy, a_next) + by)
    ### END CODE HERE ###
    # store values needed for backward propagation in cache
    cache = (a_next, c_next, a_prev, c_prev, ft, it, cct, ot, xt,
parameters)
    return a_next, c_next, yt_pred, cache
```

1. LSTM


LSTM_cell_forward
softmax*
tanh
forget gate
update gate
tanh
output gate
LSTM_cell
W
$\hat{y}^{\langle t \rangle}$
$c^{\langle t-1 \rangle}$
$c^{\langle t \rangle}$
$a^{\langle t \rangle}$
$a^{\langle t-1 \rangle}$
$x^{\langle t \rangle}$
$\Gamma_f^{\langle t \rangle}$
$\Gamma_i^{\langle t \rangle}$
$\tilde{c}^{\langle t \rangle}$
$\Gamma_o^{\langle t \rangle}$
$\Gamma_i^{<t>} = \sigma(W_i[a^{<t-1>}, x^{<t>}] + b_i)$
it = sigmoid(np.dot(Wi, concat) + bi)
$\Gamma_f^{<t>} = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$
ft = sigmoid(np.dot(Wf, concat) + bf)
$c^{<t>} = \Gamma_f^{<t>} * c^{<t-1>} + \Gamma_i^{<t>} * \tilde{c}^{<t>}$
c_next = ft*c_prev + it*cct
$a^{<t>} = \Gamma_o^{<t>} * tanh(c^{<t>})$
a_next = ot*np.tanh(c_next)
$\Gamma_o^{<t>} = \sigma(W_a[a^{<t-1>}, x^{<t>}] + b_o)$
ot = sigmoid(np.dot(Wo, concat) + bo)
$\tilde{c}^{<t>} = tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$
cct = np.tanh(Wc.dot(concat) + bc)

1. LSTM

```
# LSTM cell 테스트 코드
np.random.seed(1)
xt_tmp = np.random.randn(3,10)
a_prev_tmp = np.random.randn(5,10)
c_prev_tmp = np.random.randn(5,10)
parameters_tmp = {}
parameters_tmp['Wf'] = np.random.randn(5, 5+3)
parameters_tmp['bf'] = np.random.randn(5,1)
parameters_tmp['Wi'] = np.random.randn(5, 5+3)
parameters_tmp['bi'] = np.random.randn(5,1)
parameters_tmp['Wo'] = np.random.randn(5, 5+3)
parameters_tmp['bo'] = np.random.randn(5,1)
parameters_tmp['Wc'] = np.random.randn(5, 5+3)
parameters_tmp['bc'] = np.random.randn(5,1)
parameters_tmp['Wy'] = np.random.randn(2,5)
parameters_tmp['by'] = np.random.randn(2,1)

a_next_tmp, c_next_tmp, yt_tmp, cache_tmp = lstm_cell_forward(xt_tmp, a_prev_tmp, c_prev_tmp, parameters_tmp)
print("a_next[4] = \n", a_next_tmp[4])
print("a_next.shape = ", a_next_tmp.shape)
print("c_next[2] = \n", c_next_tmp[2])
print("c_next.shape = ", c_next_tmp.shape)
print("yt[1] =", yt_tmp[1])
print("yt.shape = ", yt_tmp.shape)
print("cache[1][3] =\n", cache_tmp[1][3])
print("len(cache) = ", len(cache_tmp))
```

```
a_next[4] =
 [-0.66408471  0.0036921   0.02088357  0.22834167 -0.85575339  0.00138482
  0.76566531  0.34631421 -0.00215674  0.43827275]
a_next.shape =  (5, 10)
c_next[2] =
 [ 0.63267805  1.00570849  0.35504474  0.20690913 -1.64566718  0.11832942
  0.76449811 -0.0981561  -0.74348425 -0.26810932]
c_next.shape =  (5, 10)
yt[1] = [0.79913913 0.15986619 0.22412122 0.15606108 0.97057211 0.31146381
 0.00943007 0.12666353 0.39380172 0.07828381]
yt.shape =  (2, 10)
cache[1][3] =
 [-0.16263996  1.03729328  0.72938082 -0.54101719  0.02752074 -0.30821874
  0.07651101 -1.03752894  1.41219977 -0.37647422]
len(cache) =  10
```

1. LSTM

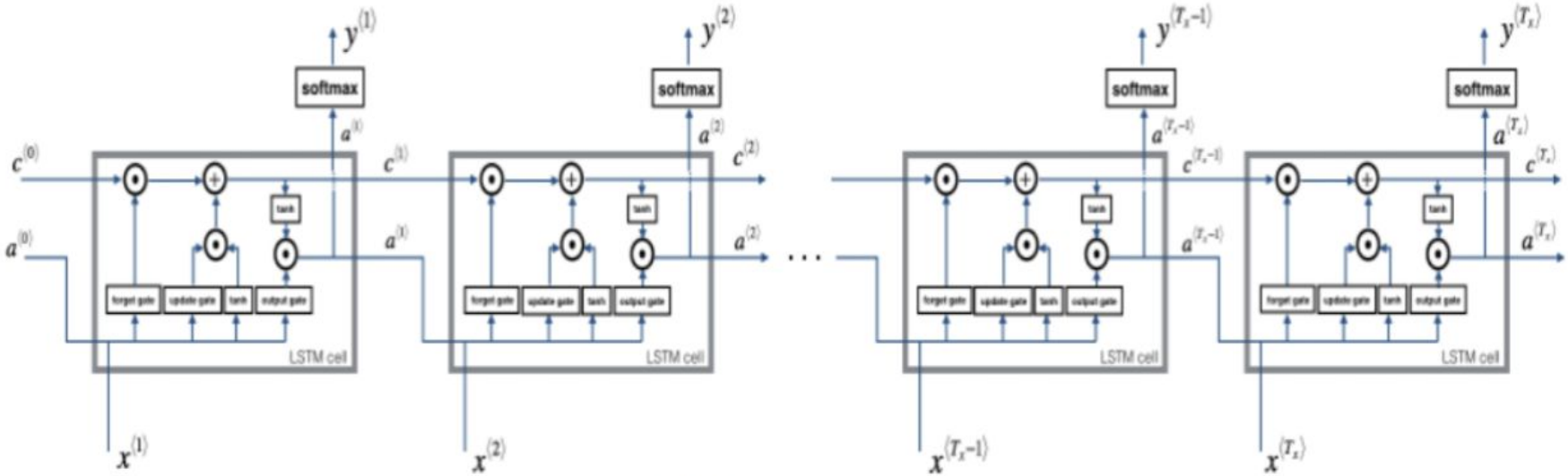


Figure 5: LSTM over multiple time-steps.

1. LSTM

```
def lstm_forward(x, a0, parameters):
    """
    Implement the forward propagation of the recurrent neural network using an LSTM-cell described in Figure (4).
    Arguments:
    x -- Input data for every time-step, of shape (n_x, m, T_x).
    a0 -- Initial hidden state, of shape (n_a, m)
    parameters -- python dictionary containing:
                        Wf -- Weight matrix of the forget gate, numpy array of shape (n_a, n_a + n_x)
                        bf -- Bias of the forget gate, numpy array of shape (n_a, 1)
                        Wi -- Weight matrix of the update gate, numpy array of shape (n_a, n_a + n_x)
                        bi -- Bias of the update gate, numpy array of shape (n_a, 1)
                        Wc -- Weight matrix of the first "tanh", numpy array of shape (n_a, n_a + n_x)
                        bc -- Bias of the first "tanh", numpy array of shape (n_a, 1)
                        Wo -- Weight matrix of the output gate, numpy array of shape (n_a, n_a + n_x)
                        bo -- Bias of the output gate, numpy array of shape (n_a, 1)
                        Wy -- Weight matrix relating the hidden-state to the output, numpy array of shape (n_y, n_a)
                        by -- Bias relating the hidden-state to the output, numpy array of shape (n_y, 1)

    Returns:
    a -- Hidden states for every time-step, numpy array of shape (n_a, m, T_x)
    y -- Predictions for every time-step, numpy array of shape (n_y, m, T_x)
    c -- The value of the cell state, numpy array of shape (n_a, m, T_x)
    caches -- tuple of values needed for the backward pass, contains (list of all the caches, x)
    """
```

1. LSTM

```
# Initialize "caches", which will track the list of all the caches
caches = []
print("parms: ============================================================================================> "   )
print(x, a0, parameters)
### START CODE HERE ###
Wy = parameters[ 'Wy'] # saving parameters['Wy'] in a local variable in case students use Wy instead of parameters['Wy']
print("Wy: ================================================================================================> "   )
print(Wy)
# Retrieve dimensions from shapes of x and parameters['Wy'] (≈2 lines)
n_x, m, T_x = x.shape
n_y, n_a = Wy.shape

# initialize "a", "c" and "y" with zeros (≈3 lines)
a = np.zeros((n_a, m, T_x))
c = np.zeros((n_a, m, T_x))
y = np.zeros((n_y, m, T_x))

# Initialize a_next and c_next (≈2 lines)
a_next = a0
c_next = np.zeros((n_a, m))
print('x.shape: (n_x, m, T_x) ============================================================================> '    )
print(n_x, m, T_x)
```

1. LSTM

```
a, c, y:  ==========================================================================>
a:  [[[0. 0.]
  [0. 0.]
  [0. 0.]
  [0. 0.]
  [0. 0.]
  [0. 0.]
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  [0. 0.]
  [0. 0.]
  [0. 0.]
  [0. 0.]
  [0. 0.]
  [0. 0.]
  [0. 0.]]

c_next:  ==========================================================================>
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

```
parms - x:  ================================================================
[[[ 1.62434536 -0.61175641]
  [-0.52817175 -1.07296862]
  [ 0.86540763 -2.3015387 ]
  [ 1.74481176 -0.7612069 ]
  [ 0.3190391  -0.24937038]
  [ 1.46210794 -2.06014071]
  [-0.3224172  -0.38405435]
  [ 1.13376944 -1.09989127]
  [-0.17242821 -0.87785842]
  [ 0.04221375  0.58281521]]

 [[-1.10061918  1.14472371]
  [ 0.90159072  0.50249434]
  [ 0.90085595 -0.68372786]
  [-0.12289023 -0.93576943]
  [-0.26788808  0.53035547]
  [-0.69166075 -0.39675353]
  [-0.6871727  -0.84520564]
  [-0.67124613 -0.0126646 ]
  [-1.11731035  0.2344157 ]
  [ 1.65980218  0.74204416]]

 [[-0.19183555 -0.88762896]
  [-0.74715829  1.6924546 ]
  [ 0.05080775 -0.63699565]
  [ 0.19091548  2.10025514]
  [ 0.12015895  0.61720311]
  [ 0.30017032 -0.35224985]
  [-1.1425182  -0.34934272]
  [-0.20889423  0.58662319]
  [ 0.83898341  0.93110208]
  [ 0.28558733  0.88514116]]]

parms - a0:  ================================================================
[[-0.75439794  1.25286816  0.51292982 -0.29809284  0.48851815 -0.07557171
   1.13162939  1.51981682  2.18557541 -1.39649634]
 [-1.44411381 -0.50446586  0.16003707  0.87616892  0.31563495 -2.02220122
  -0.30620401  0.82797464  0.23009474  0.76201118]
 [-0.22232814 -0.20075807  0.18656139  0.41005165  0.19829972  0.11900865
  -0.67066229  0.37756379  0.12182127  1.12948391]
 [ 1.19891788  0.18515642 -0.37528495 -0.63873041  0.42349435  0.07734007
  -0.34385368  0.04359686 -0.62000084  0.69803203]
 [-0.44712856  1.2245077   0.40349164  0.59357852 -1.09491185  0.16938243
   0.74055645 -0.9537006  -0.26621851  0.03261455]]

parms - parameters:  ================================================================
Wf:  [[-1.37311732  0.31515939  0.84616065 -0.85951594  0.35054598 -1.31228341
  -0.03869551 -1.61577235]
 [ 1.12141771  0.40890054 -0.02461696 -0.77516162  1.27375593  1.96710175
  -1.85798186  1.23616403]
 [ 1.62765075  0.3380117  -1.19926803  0.86334532 -0.1809203  -0.60392063
  -1.23005814  0.5505375 ]
 [ 0.79280687 -0.62353073  0.52057634 -1.14434139  0.80186103  0.0465673
  -0.18656977 -0.10174587]
 [ 0.86888616  0.75041164  0.52946532  0.13770121  0.07782113  0.61838026
```

1. LSTM

```
# loop over all time-steps
    for t in range(T_x):
        # Get the 2D slice 'xt' from the 3D input 'x' at time step 't'
        xt = x[:,:,t]
        print("xt: ================================================================================> "   )
        print(xt)
        # Update next hidden state, next memory state, compute the prediction, get the cache (≈1 line)

        a_next, c_next, yt, cache = lstm_cell_forward(xt, a_next, c_next, parameters)
        print("in => lstm_cell_forward => a_next, c_next, yt, cache   : ================================================> "  )
        print(a_next, c_next, yt, cache)
        # Save the value of the new "next" hidden state in a (≈1 line)
        a[:,:,t] = a_next
        # Save the value of the next cell state (≈1 line)
        c[:,:,t]  = c_next
        # Save the value of the prediction in y (≈1 line)
        y[:,:,t] = yt
        # Append the cache into caches (≈1 line)
        print("a, y, c: ================================================================================>"   )
        print(a, y, c)
        caches.append(cache)
```

1. LSTM

로스값을 구해서 업데이트를 하기위해

```
 0.02965049 0.04335889 0.04108168 0.0292814 ]]

a, y, c: ==========================================
a:  [[[ 2.26030281e-01  0.00000000e+00]
  [ 1.52248538e-03  0.00000000e+00]
  [-1.64952632e-04  0.00000000e+00]
  [-2.74892203e-04  0.00000000e+00]
  [ 3.12424528e-01  0.00000000e+00]
  [-3.17216340e-03  0.00000000e+00]
  [ 7.25477757e-02  0.00000000e+00]
  [ 1.47515392e-01  0.00000000e+00]
  [-1.10657327e-01  0.00000000e+00]
  [-3.27604598e-03  0.00000000e+00]]

 [[ 1.21929808e-02  0.00000000e+00]
  [ 6.08670836e-03  0.00000000e+00]
  [ 2.20846714e-02  0.00000000e+00]
  [-5.24343564e-01  0.00000000e+00]
  [ 8.84483990e-02  0.00000000e+00]
  [ 6.36644562e-03  0.00000000e+00]
  [ 2.29020976e-01  0.00000000e+00]
  [ 1.21055558e-01  0.00000000e+00]
  [ 3.38775760e-01  0.00000000e+00]
  [-8.34954406e-02  0.00000000e+00]]

 [[ 2.37337021e-04  0.00000000e+00]
```

```
a, y, c: ==========================================================================
a:  [[[ 2.26030281e-01  1.17275046e-01]
  [ 1.52248538e-03 -6.76047802e-02]
  [-1.64952632e-04  4.29690928e-01]
  [-2.74892203e-04 -1.31743932e-01]
  [ 3.12424528e-01  2.79788453e-04]
  [-3.17216340e-03  2.47483690e-01]
  [ 7.25477757e-02  3.43071735e-01]
  [ 1.47515392e-01  5.29658577e-03]
  [-1.10657327e-01 -1.22242321e-01]
  [-3.27604598e-03 -2.74673603e-03]]

 [[ 1.21929808e-02 -3.75103932e-03]
  [ 6.08670836e-03 -4.06788592e-03]
  [ 2.20846714e-02 -3.47989892e-02]
  [-5.24343564e-01 -4.67383955e-01]
  [ 8.84483990e-02  8.60044218e-02]
  [ 6.36644562e-03 -2.09647576e-02]
  [ 2.29020976e-01  4.01214942e-01]
  [ 1.21055558e-01  9.79422217e-02]
  [ 3.38775760e-01  2.48561911e-02]
  [-8.34954406e-02 -5.92645437e-03]]
```

1. LSTM

```
### END CODE HERE ###

# store values needed for backward propagation in cache
caches = (caches, x)
print("return a, y, c, caches: ==========================================================================================> "    )
print(a, y, c, caches)
return a, y, c, caches
```

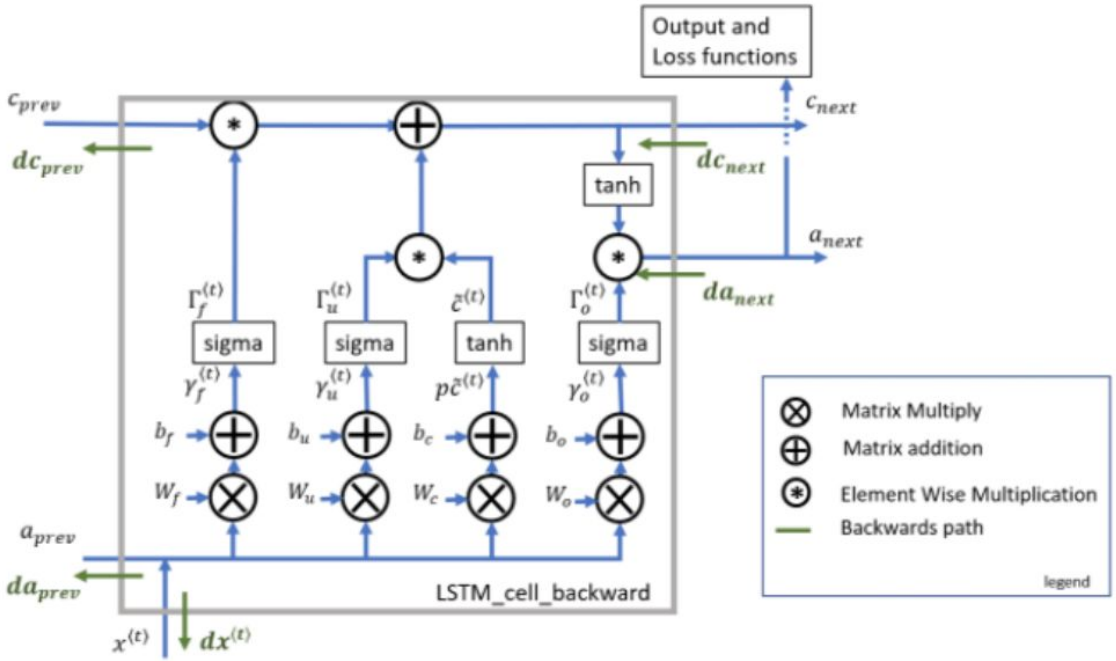
로스값을 구해서 **=>** 역전파

1. LSTM

```
np.random.seed(1)
x_tmp = np.random.randn(3,10,7)
a0_tmp = np.random.randn(5,10)
parameters_tmp = {}
parameters_tmp['Wf'] = np.random.randn(5, 5+3)
parameters_tmp['bf'] = np.random.randn(5,1)
parameters_tmp['Wi'] = np.random.randn(5, 5+3)
parameters_tmp['bi']= np.random.randn(5,1)
parameters_tmp['Wo'] = np.random.randn(5, 5+3)
parameters_tmp['bo'] = np.random.randn(5,1)
parameters_tmp['Wc'] = np.random.randn(5, 5+3)
parameters_tmp['bc'] = np.random.randn(5,1)
parameters_tmp['Wy'] = np.random.randn(2,5)
parameters_tmp['by'] = np.random.randn(2,1)

a_tmp, y_tmp, c_tmp, caches_tmp = lstm_forward(x_tmp, a0_tmp,
parameters_tmp)
print("a[4][3][6] = ", a_tmp[4][3][6])
print("a.shape = ", a_tmp.shape)
print("y[1][4][3] =", y_tmp[1][4][3])
print("y.shape = ", y_tmp.shape)
print("caches[1][1][1] =\n", caches_tmp[1][1][1])
print("c[1][2][1]", c_tmp[1][2][1])
print("len(caches) = ", len(caches_tmp))
```

1. LSTM

```
def lstm_cell_backward(da_next, dc_next, cache):
```



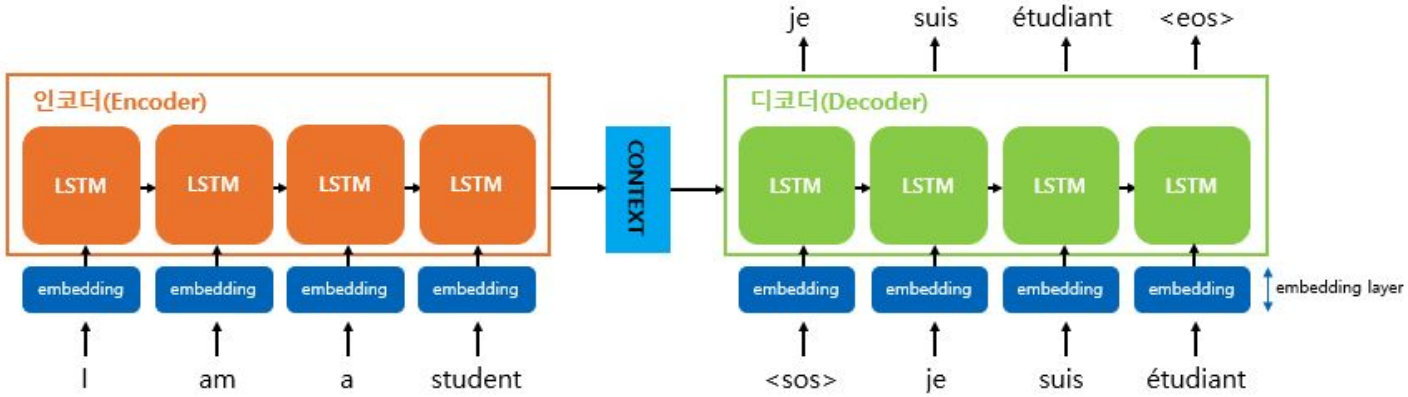
1. LSTM

```
np.random.seed(1)
x_tmp = np.random.randn(3,10,7)
a0_tmp = np.random.randn(5,10)
parameters_tmp = {}
parameters_tmp['Wf'] = np.random.randn(5, 5+3)
parameters_tmp['bf'] = np.random.randn(5,1)
parameters_tmp['Wi'] = np.random.randn(5, 5+3)
parameters_tmp['bi']= np.random.randn(5,1)
parameters_tmp['Wo'] = np.random.randn(5, 5+3)
parameters_tmp['bo'] = np.random.randn(5,1)
parameters_tmp['Wc'] = np.random.randn(5, 5+3)
parameters_tmp['bc'] = np.random.randn(5,1)
parameters_tmp['Wy'] = np.random.randn(2,5)
parameters_tmp['by'] = np.random.randn(2,1)

a_tmp, y_tmp, c_tmp, caches_tmp = lstm_forward(x_tmp, a0_tmp,
parameters_tmp)
print("a[4][3][6] = ", a_tmp[4][3][6])
print("a.shape = ", a_tmp.shape)
print("y[1][4][3] =", y_tmp[1][4][3])
print("y.shape = ", y_tmp.shape)
print("caches[1][1][1] =\n", caches_tmp[1][1][1])
print("c[1][2][1]", c_tmp[1][2][1])
print("len(caches) = ", len(caches_tmp))
```

1. LSTM

```
np.random.seed(1)
x_tmp = np.random.randn(3,10,7)
a0_tmp = np.random.randn(5,10)
parameters_tmp = {}
parameters_tmp['Wf'] = np.random.randn(5, 5+3)
parameters_tmp['bf'] = np.random.randn(5,1)
parameters_tmp['Wi'] = np.random.randn(5, 5+3)
parameters_tmp['bi']= np.random.randn(5,1)
parameters_tmp['Wo'] = np.random.randn(5, 5+3)
parameters_tmp['bo'] = np.random.randn(5,1)
parameters_tmp['Wc'] = np.random.randn(5, 5+3)
parameters_tmp['bc'] = np.random.randn(5,1)
parameters_tmp['Wy'] = np.random.randn(2,5)
parameters_tmp['by'] = np.random.randn(2,1)

a_tmp, y_tmp, c_tmp, caches_tmp = lstm_forward(x_tmp, a0_tmp,
parameters_tmp)
print("a[4][3][6] = ", a_tmp[4][3][6])
print("a.shape = ", a_tmp.shape)
print("y[1][4][3] =", y_tmp[1][4][3])
print("y.shape = ", y_tmp.shape)
print("caches[1][1][1] =\n", caches_tmp[1][1][1])
print("c[1][2][1]", c_tmp[1][2][1])
print("len(caches) = ", len(caches_tmp))
```

2. seq2seq



- 구현범위
  Softmax
  Embedding layer
  Lstm
  Seq2Seq Architecture

- 현재는 Train시간이 너무 오래걸림
- 최적화 방안 모색 중

https://www.notion.so/Code-a4f6b6158288400caf7e9a80a22278c4

2. seq2seq

Encoder

```
1  encoder_input.shape, decoder_input.shape, decoder_target.shape
```

✓ 0.2s

```
((10000, 30, 1391), (10000, 70, 116), (10000, 70, 116))
```

```
1  encoder_inputs = Input(shape=(None, encoder_input.shape[2]))
2  encoder_lstm = LSTM(units=256, return_state=True)
3
4  encoder_outputs, state_h, state_c = encoder_lstm(encoder_inputs)
5
6  # 은닉셀과, 셀상태
7  encoder_states = [state_h, state_c]
```

✓ 0.1s

2. seq2seq

Decoder

```
1  decoder_inputs = Input(shape=(None, decoder_input.shape[2]))
2  decoder_lstm = LSTM(units=256, return_sequences=True, return_state=True)
3
4  # 디코더에게 인코더의 은닉 상태, 셀 상태를 전달
5  decoder_outputs, _, _= decoder_lstm(decoder_inputs, initial_state=encoder_states)
6
7  decoder_sotfmax_layer = Dense(decoder_input.shape[2], activation='softmax')
8  decoder_outputs = decoder_sotfmax_layer(decoder_outputs)
9
10 model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
11 model.compile(optimizer='rmsprop', loss = 'categorical_crossentropy')
```

✓ 0.1s

2. seq2seq

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, None, 1391)  0           []
                                ]

 input_2 (InputLayer)           [(None, None, 116)]  0           []

 lstm (LSTM)                    [(None, 256),        1687552     ['input_1[0][0]']
                                 (None, 256),
                                 (None, 256)]

 lstm_1 (LSTM)                  [(None, None, 256),  381952      ['input_2[0][0]',
                                 (None, 256),                     'lstm[0][1]',
                                 (None, 256)]                     'lstm[0][2]']

 dense (Dense)                  (None, None, 116)    29812       ['lstm_1[0][0]']

==================================================================================================
Total params: 2,099,316
Trainable params: 2,099,316
Non-trainable params: 0
__________________________________________________________________________________________________
```

2. seq2seq

번역기 동작 시키기

```
encoder_model = Model(inputs=encoder_inputs, outputs=encoder_states)
```

✓ 0.3s

```
# 이전 시점의 상태들을 저장하는 텐서
decoder_state_input_h = Input(shape=(256,))
decoder_state_input_c = Input(shape=(256,))
decoder_state_inputs = [decoder_state_input_h, decoder_state_input_c]

# 문장의 다음 단어 예측을 위해서 초기상태 를 이전 시점의 상태로 사용
decoder_outputs, state_h, state_c = decoder_lstm(decoder_inputs, initial_state=decoder_state_inputs)

# 훈련 과정에서와 달리 LSTM의 리턴하는 은닉상태와 셀상태를 버리지 않음
decoder_states = [state_h, state_c]
decoder_outputs = decoder_sotfmax_layer(decoder_outputs)
decoder_model = Model(inputs=[decoder_inputs] + decoder_state_inputs, outputs=[decoder_outputs] + decoder_states)
```

✓ 0.1s

```
index_to_src = dict((i, char) for char, i in src_to_index.items())
index_to_tar = dict((i, char) for char, i in tar_to_index.items())
```

✓ 0.2s

2. seq2seq

```python
def decode_sequence(input_seq):
    # 입력으로부터 인코더의 상태를 얻음
    states_value = encoder_model.predict([input_seq])

    # <SOS>에 해당하는 원-핫 벡터 생성
    target_seq = np.zeros((1, 1, decoder_input.shape[2]))
    target_seq[0, 0, tar_to_index['\t']] = 1.

    stop_condition = False
    decoded_sentence = ""

    # stop condition이 True될 때까지 루프 반복
    while not stop_condition:
        # 이전 시점의 상태 states_value를 현 시점의 초기 상태로 사용
        output_tokens, h, c = decoder_model.predict([target_seq] + states_value)

        # 예측 결과를 문자로 변환
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char = index_to_tar[sampled_token_index]

        # 현재 시점의 예측 문자를 예측 문장에 추가
        decoded_sentence += sampled_char

        # <EOS>에 도달하거나 최대 길이를 넘으면 중단
        if (sampled_char == '\n' or len(decoded_sentence) > max_tar_len):
            stop_condition = True

        # 현재 시점의 예측 결과를 다음 시점의 입력으로 사용하기 위해 저장
        target_seq = np.zeros((1, 1, decoder_input.shape[2]))
        target_seq[0, 0, sampled_token_index] = 1.

        #현재 시점의 상태를 다음 시점의 상태로 사용하기 위해 저장
        states_value= [h, c]

    return decoded_sentence

```

2. seq2seq

```
bleu_score = []
bleu_score_std = []
for seq_index in [1, 20, 30, 40, 50, 60, 100, 200, 300, 400, 500, 1000]: # 입력 문장 인덱스
    input_seq = encoder_input[seq_index:seq_index+1]
    decoded_sentence = decode_sequence(input_seq)
    score = sentence_bleu(
        list(map(lambda ref: ref.split(), df.tar[seq_index][2:len(df.tar[seq_index])-1])),
        df.tar[seq_index][2:len(df.tar[seq_index])-1].split(),
        weights=(1, 0, 0, 0)
    )
    bleu_score.append(score)

    print(35 * '-')
    print('입력문장: ', df.src[seq_index])
    print('정답문장: ', df.tar[seq_index][2:len(df.tar[seq_index])-1]) # \t 와 \n 빼고 출력
    print('번역문장: ', decoded_sentence[1:len(decoded_sentence)-1]) # \n을 빼고 출력

print("1gram bleu Score:{0:0.4f}".format(median(bleu_score) * 100))
```

2. seq2seq

```
입력문장:  씨티은행에서 일하세요?
정답문장:  Do you work at a City bank?
번역문장:  Could you let me know the March Mark?
----------------------------------
입력문장:  이 제품을 사용할 때 예기치 않은 추락의 경우를 고려하면, 안전을 생각하여 잔디밭이 필요합니다.
정답문장:  Also, this product needs a grass field for safety considering unexpected falls.
번역문장:  I will send you the divers after the price after 20 years ago.
----------------------------------
입력문장:  추가로 작성해서 내일까지 보내줄게요.
정답문장:  I'll fill it out and send it to you by tomorrow.
번역문장:  I will send you the best after company and the send of the stope.
----------------------------------
입력문장:  "유괴"는 스코틀랜드를 배경으로 한 모험 소설이며 문학적 완성도 또한 높은 소설이에요.
정답문장:  "Kidnapped" is an adventure story set in Scotland with high literary significance.
번역문장:  And the send of the second floor selected and sent is below.
----------------------------------
입력문장:  "공식 초청 레터"를 받는 데로 비자를 받을 것입니다.
정답문장:  I will get my visa as soon as I receive the " official invitation letter".
번역문장:  I was a preside to be a place to be a company on the 20th of A.
```

2. seq2seq

Model	1-gram BLEU	Standard BLEU
seq2seq(LSTM) / rmsprop / 10,000 / 20epoch	2.04	-
seq2seq(LSTM) / rmsprop / 10,000 / 30epoch	3.24	-
seq2seq(LSTM) / rmsprop / 10,000 / 40epoch	2.05	-
seq2seq(LSTM) / rmsprop / 50,000 / 40epoch	3.04	-
seq2seq(LSTM) / rmsprop / 10,000 / 50epoch	2.38	-
seq2seq(LSTM) / rmsprop / 50,000 / 50epoch	5.25	-

3. Transformer

Dataset

- AI Hub 한국어-영어 병렬 말뭉치

```
[ ]  file = pd.read_excel('/content/drive/My Drive/data/1_구어체(1).xlsx')
     dataFrame = pd.DataFrame({"kor" : file['원문'], "eng" : file['번역문']})
     dataFrame = dataFrame.sample(frac=1).reset_index(drop=True)
     corpus = dataFrame[:200000]

     print(len(corpus))
```

```
200000
```

```
[ ]  corpus.head()
```

	kor	eng
0	아래 품목에 대한 배송이 확정된 후에 제가 송금해도 될까요?	Can I send you money after the shipment for th...
1	나 클래스 하는 중간에 들어갈 것입니다.	I will enter in the middle of the class.
2	그래서 한계에 다다른 아내는 남편에게 말했습니다.	So the wife who reached the limit told her hus...
3	기타를 연주를 위한 악보도 함께 보냈어.	I sent you the music note to play with your gu...
4	여기 복부 부분 보정 한 번 더 합시다.	Let's make another correction on this abdomina...

3. Transformer

Pre-Processing

- 한국어 : Mecab
- 영어 : Spacy

```
[ ]  import spacy
     from konlpy.tag import Mecab
     tokenizer = Mecab()

     spacy_en = spacy.load('en_core_web_sm')

     def tokenize_ko(text):
         return [token for token in tokenizer.morphs(text)]

     def tokenize_en(text):
         return [token.text for token in spacy_en.tokenizer(text)]
```

```
[ ]  from torchtext.legacy import data

     SRC = data.Field(tokenize = tokenize_ko, lower = True, batch_first=True)
     TRG = data.Field(tokenize = tokenize_en, init_token = "<sos>", eos_token = "<eos>", lower = True, batch_first=True)

     train_dataset, valid_dataset, test_dataset = data.TabularDataset.splits(
         path='/content/drive/My Drive/data/',
         train='train.csv', validation='valid.csv', test='test.csv',
         format='csv',
         fields=[('SRC', SRC), ('TRG', TRG)]),
         skip_header=True
     )
```

3. Transformer

LossFunction

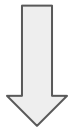
- CrossEntropyLoss

$$PPL(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1, \cdots, w_{i-1})}}$$

$$\begin{aligned} L &= -\frac{1}{N} \sum_{i=1}^{N} \log P_\theta(w_i|w_{<i}) \\ &= \log \left(\left(\prod_{i=1}^{N} P_\theta(w_i|w_{<i}) \right)^{-\frac{1}{N}} \right) \\ &= \log \left(\sqrt[N]{\frac{1}{\prod_{i=1}^{N} P_\theta(w_i|w_{<i})}} \right) \end{aligned}$$

Perplexity : 테스트 문장에 대한 확률을 구하고

문장의 길이에 대해서 normalization(기하평균)



$$PPL = \exp(Cross\ Entropy)$$

3. Transformer

Optimizer

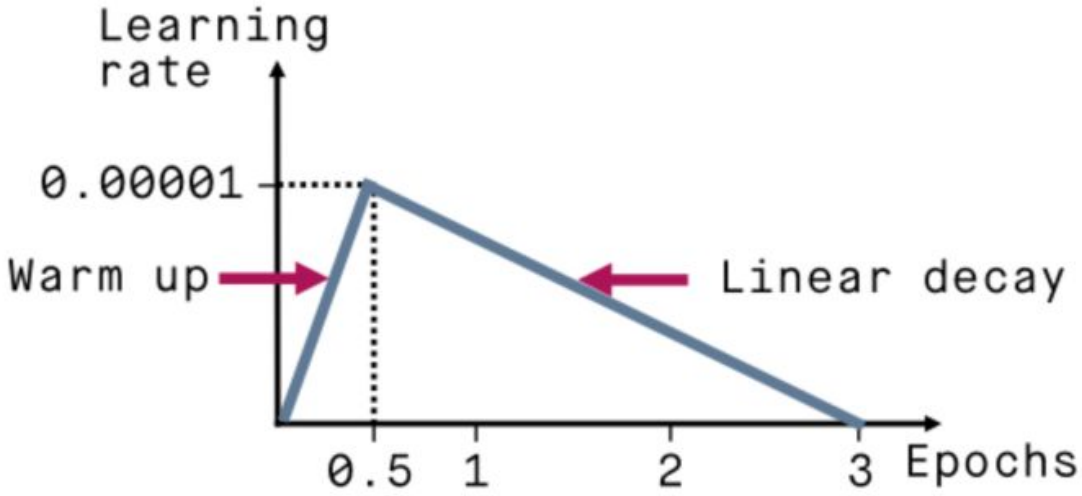
- Adam

Algorithm 1: Generic adaptive optimization method setup. All operations are element-wise.

Input: $\{\alpha_t\}_{t=1}^{T}$: step size, $\{\phi_t, \psi_t\}_{t=1}^{T}$: function to calculate momentum and adaptive rate,
θ_0: initial parameter, $f(\theta)$: stochastic objective function.
Output: θ_T: resulting parameters
while $t = 1$ *to* T **do**
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Calculate gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \phi_t(g_1, \cdots, g_t)$ (Calculate momentum)
$l_t \leftarrow \psi_t(g_1, \cdots, g_t)$ (Calculate adaptive learning rate)
$\theta_t \leftarrow \theta_{t-1} - \alpha_t m_t l_t$ (Update parameters)
return θ_T

Transformer에서 성능이 낮음

3. Transformer



Warm-up and Linear decay

3. Transformer

```
#import torch.optim as optim
import torch_optimizer as optim

LEARNING_RATE = 0.0005
#optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
optimizer = optim.RAdam(model.parameters(), lr=LEARNING_RATE)
```

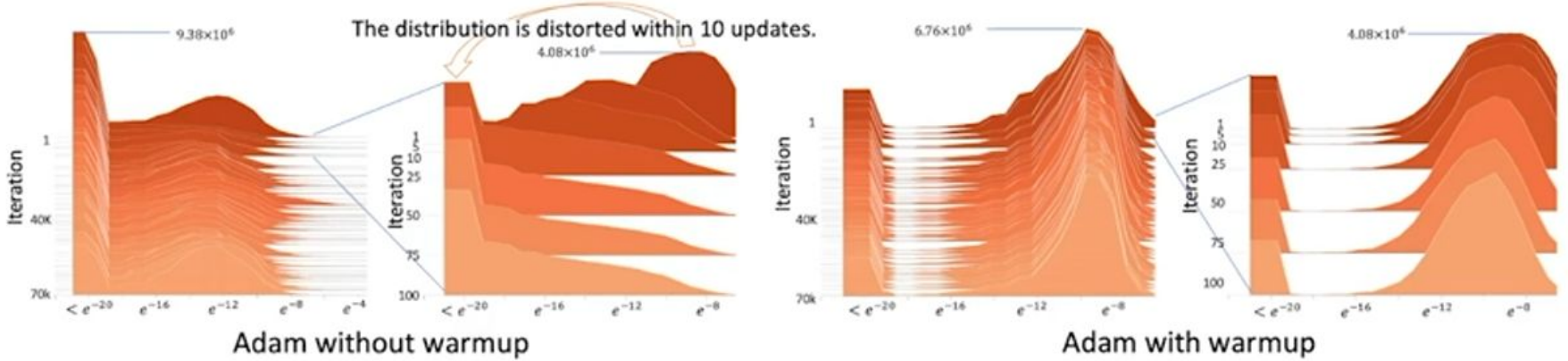


Figure 2: The absolute gradient histogram of the Transformers on the De-En IWSLT' 14 dataset during the training (stacked along the y-axis). X-axis is absolute value in the log scale and the height is the frequency. Without warmup, the gradient distribution is distorted in the first 10 steps.

Rectified Adam(Liu et al., 2020)

3. Transformer

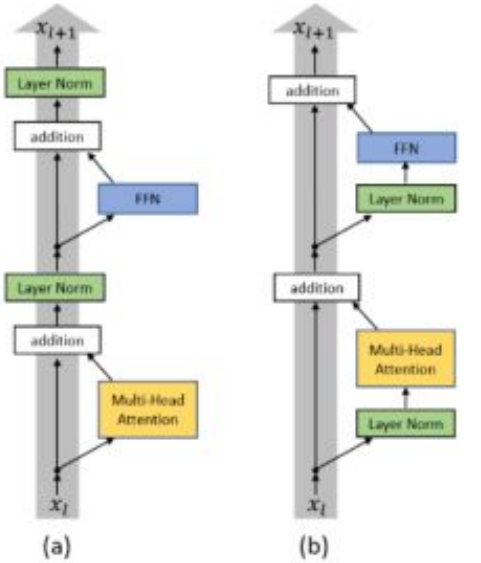


Figure 1: (a) Post-LN Transformer layer; (b) Pre-LN Transformer layer.

```
# ============Post-LN============
 _src, _ = self.self_attention(src, src, src, src_mask)
 src = self.self_attn_layer_norm(src + self.dropout(_src))

 _src = self.positionwise_feedforward(src)
 src = self.ff_layer_norm(src + self.dropout(_src))
```

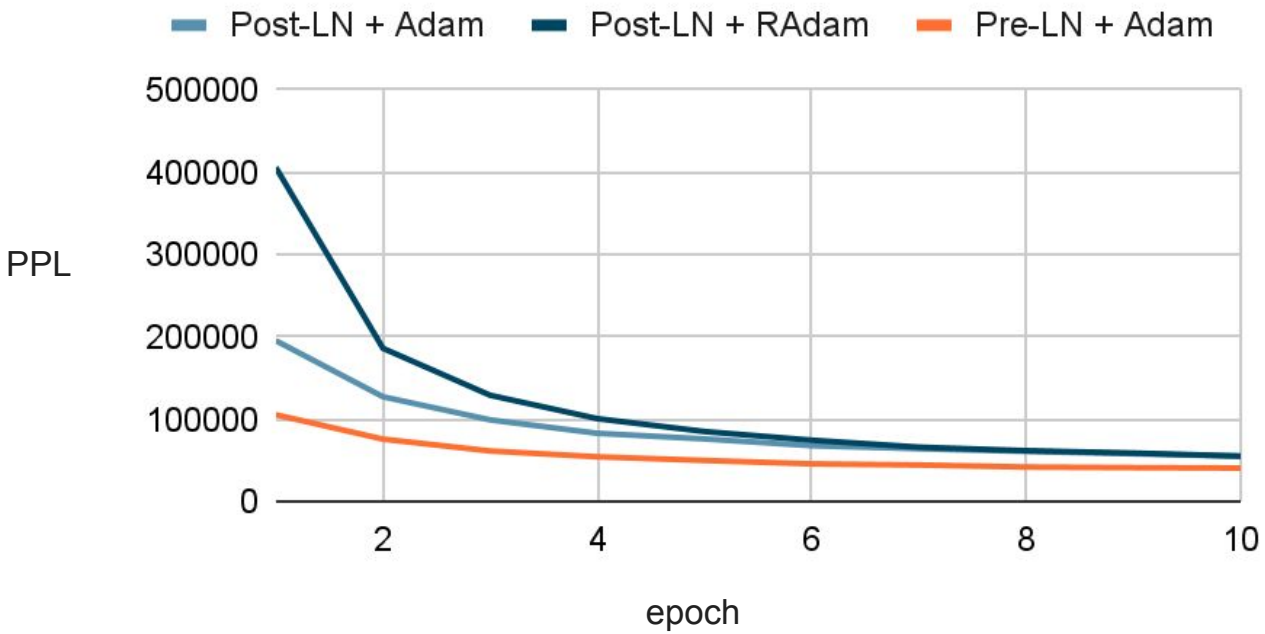
```
# ============Pre-LN============
 _src = self.self_attn_layer_norm(src)
 _src, _ = self.self_attention(_src, _src, _src, src_mask)
 _src = src + self.dropout(_src)

 src = _src + self.dropout(self.positionwise_feedforward(self.ff_layer_norm(_src)))
```

On Layer Normalization in the Transformer Architecture (Xiong et al., 2020)

3. Transformer

실험결과

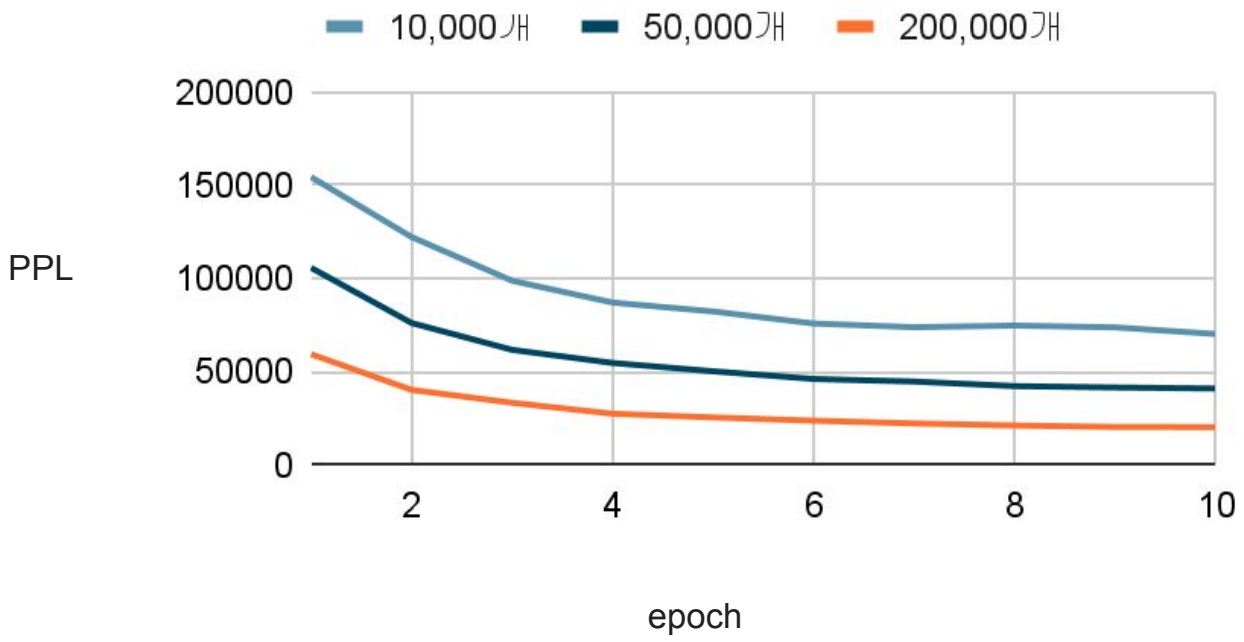
- 데이터 50,000개
- n_epochs 10
- dropout 0.5



3. Transformer

실험결과

- Pre-LN
- n_epochs 10
- dropout 0.5



3. Transformer

실험결과

```
score1 = corpus_bleu(trgs, pred_trgs, weights=(1, 0, 0, 0))
print(f'1-gram BLEU Score = {score1*100:.2f}')
score3 = corpus_bleu(trgs, pred_trgs, weights=(0.25, 0.25, 0.25, 0.25))
print(f'Standard BLEU Score = {score3*100:.2f}')
```

Model	1-gram BLEU	Standard BLEU
Post-LN / Adam / 10,000 / 10epoch	14.61	-
Pre-LN / Adam / 10,000 / 10epoch	16.06	-
Post-LN / Adam / 50,000 / 10epoch	20.72	-
Post-LN / RAdam / 50,000 / 10epoch	19.61	-
Pre-LN / Adam / 50,000 / 10epoch	25.86	-
Pre-LN / Adam / 200,000 / 30epoch	42.43	11.5

3. Transformer

어려웠던 점

- TorchText, Mecab 설치
- Loss가 튀는 현상
- 컴퓨팅파워

```
# https://pypi.org/project/torchtext/

!pip install torch==1.8.1
!pip install torchtext==0.9.1
```

```
!apt-get update
!apt-get install g++ openjdk-8-jdk
!pip3 install konlpy JPype1-py3
!bash <(curl -s https://raw.githubusercontent.com/konlpy/konlpy/master/scripts/mecab.sh)
```

```
$ sudo dd if=/dev/zero of=/swapfile bs=128M count=16
$ sudo chmod 600 /swapfile
$ sudo mkswap /swapfile
$ sudo swapon /swapfile
$ sudo swapon -s
$ cd /
$ sudo nano /etc/fstab # 마지막에 다음과 같이 줄을 추가하고 (
$ free # 잘 할당되었는지 확인
```