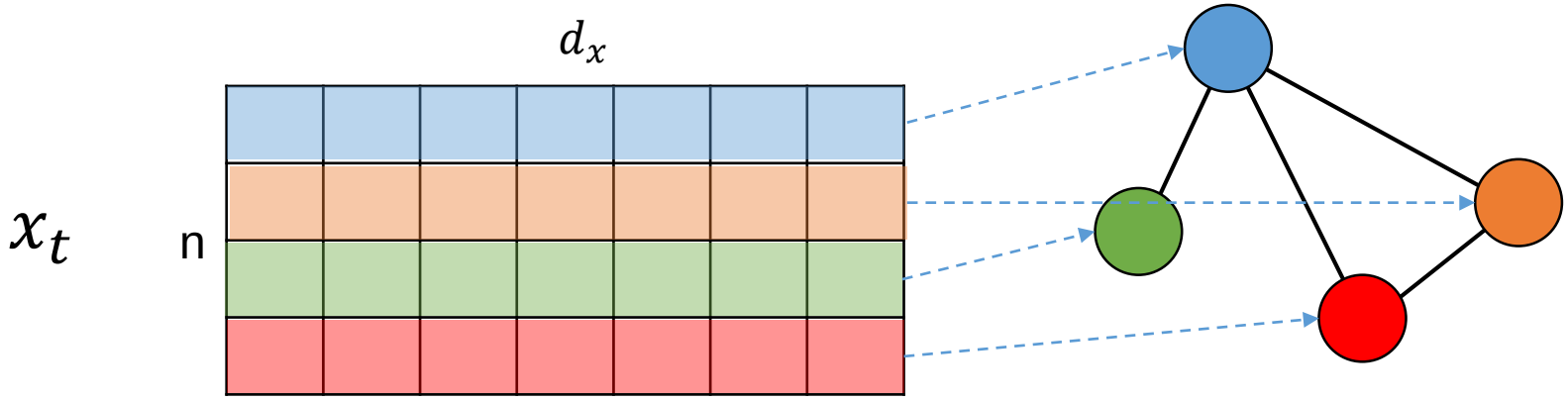
Structured Sequence Modeling With Graph Convolutional Recurrent Networks

Y.Seo, M.Defferrard, P.Vandergheynst, X.Bresson

Presenter : Bumho Son

Objective of Research

- How to deal with structured sequence data?
- Spatio-temporal sequences
  - $\hat{x}_{t+1}, \ldots, \hat{x}_{t+K} = argmax_{x_{t+1},\ldots,x_{t+K}} P\left(x_{t+1}, \ldots, x_{t+K} \middle| x_{t-J+1}, \ldots, x_t\right)$
  - Consider data x_t as a graph signal : features are linked by pairwise relationship



Related works

- Shi et al. (2015)
  - Model for regular grid-structured sequences
  - Classical FC-LSTM
  - Replace multiplications by dense matrices W to convolutions with kernels W

$$
\begin{aligned}
i &= \sigma(W_{xi} * x_t + W_{hi} * h_{t-1} + w_{ci} \odot c_{t-1} + b_i), \\
f &= \sigma(W_{xf} * x_t + W_{hf} * h_{t-1} + w_{cf} \odot c_{t-1} + b_f), \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc} * x_t + W_{hc} * h_{t-1} + b_c), \\
o &= \sigma(W_{xo} * x_t + W_{ho} * h_{t-1} + w_{co} \odot c_t + b_o), \\
h_t &= o \odot \tanh(c_t),
\end{aligned}
$$

- Ranzato et al. (2014)
  - Use 1x1 convolutional layers instead of fc layers

$$h_t = \tanh(\sigma(W_{x2} * \sigma(W_{x1} * x_t)) + \sigma(W_h * h_{t-1})),$$

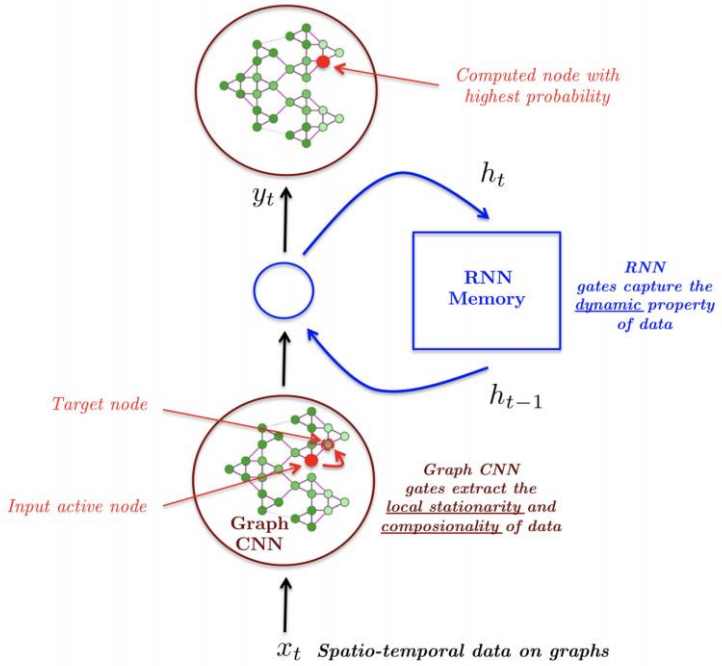
Related works

- Tai et al. (2015)
  - Natural language exhibits syntactic properties that naturally combine words into phrases
  - Model for tree-structured topologies
  - Each LSTM has access to the states of its children

- Li et al. (2015)
  - Use iterative procedure of the GNNs to propagate node representations until convergence

Proposed Models : GCRN

- Graph Convolutional Recurrent Network (GCRN)
- Main idea
  - Use Graph CNN and RNN
  - Graph CNN : Identify spatial structures
  - RNN : Find dynamic patterns
- Model 1 : Stack a graph CNN on an LSTM



Graph CNN

$$x_t^{\text{CNN}} = \text{CNN}_{\mathcal{G}}(x_t)$$

LSTM

$$\begin{aligned}
i &= \sigma(W_{xi}x_t^{\text{CNN}} + W_{hi}h_{t-1} + w_{ci} \odot c_{t-1} + b_i),\\
f &= \sigma(W_{xf}x_t^{\text{CNN}} + W_{hf}h_{t-1} + w_{cf} \odot c_{t-1} + b_f),\\
c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t^{\text{CNN}} + W_{hc}h_{t-1} + b_c),\\
o &= \sigma(W_{xo}x_t^{\text{CNN}} + W_{ho}h_{t-1} + w_{co} \odot c_t + b_o),\\
h_t &= o \odot \tanh(c_t).
\end{aligned}$$

Graph filtering operation

$$y = g_\theta *_{\mathcal{G}} x = g_\theta(L)x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x,$$

θ : Chebyshev coefficients

$T_k(\tilde{L})$: Chebyshev polynomial of order k

Proposed Models : GCRN

- Model 2 : Generalize the convLSTM model to graphs

$$
\begin{aligned}
i &= \sigma(W_{xi} *_{\mathcal{G}} x_t + W_{hi} *_{\mathcal{G}} h_{t-1} + w_{ci} \odot c_{t-1} + b_i),\\
f &= \sigma(W_{xf} *_{\mathcal{G}} x_t + W_{hf} *_{\mathcal{G}} h_{t-1} + w_{cf} \odot c_{t-1} + b_f),\\
c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc} *_{\mathcal{G}} x_t + W_{hc} *_{\mathcal{G}} h_{t-1} + b_c),\\
o &= \sigma(W_{xo} *_{\mathcal{G}} x_t + W_{ho} *_{\mathcal{G}} h_{t-1} + w_{co} \odot c_t + b_o),\\
h_t &= o \odot \tanh(c_t).
\end{aligned}
$$

$W_{xi}x_t$ Classic LSTM : matrix multiplication by dense matrix W

↓

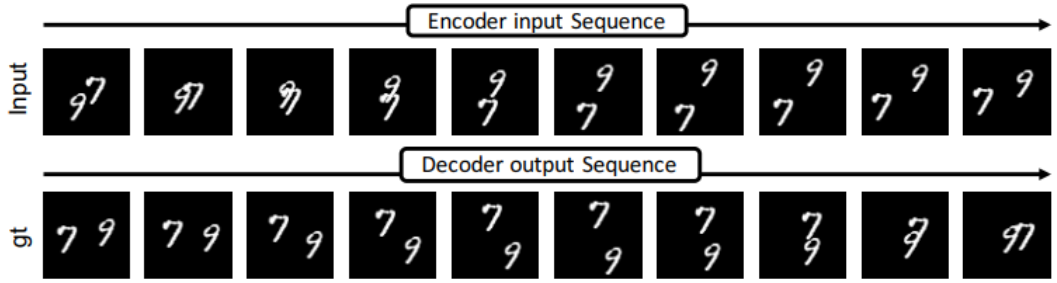
$W_{xi} * x_t$ convLSTM : Replace multiplication by 2D convolution($*$) by a set of kernels (Shi et al.(2015))

↓

$W_{xi} *_{\mathcal{G}} x_t$ GCRN : Replace 2D convolution by the graph convolution($*_{\mathcal{G}}$)

Experiments : Dataset

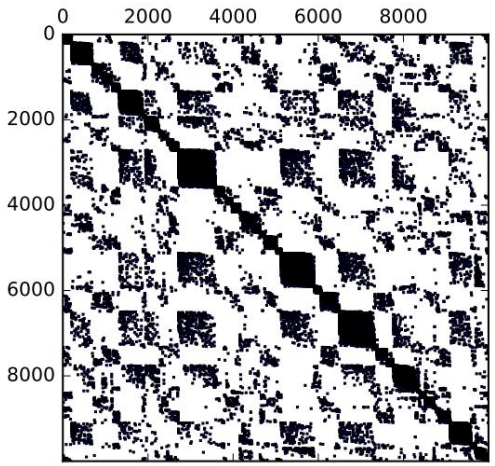
- Moving-MNIST
  - 20 frame MNIST

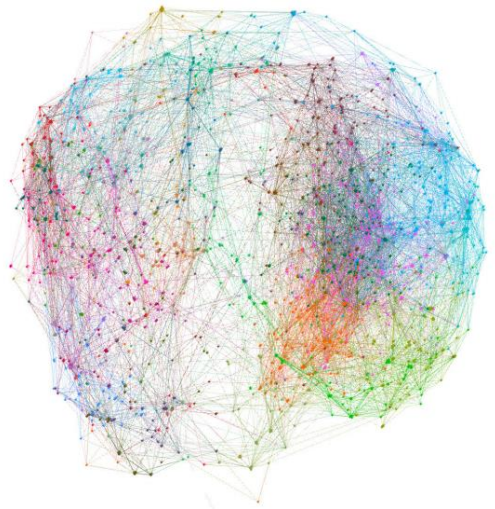


  - Rotating and moving hand-written digits
  - How to construct adjacency matrix A?
    - K-nearest-neighbor graph with Euclidean distance and Gaussian kernel between pixel locations

Experiments : Dataset

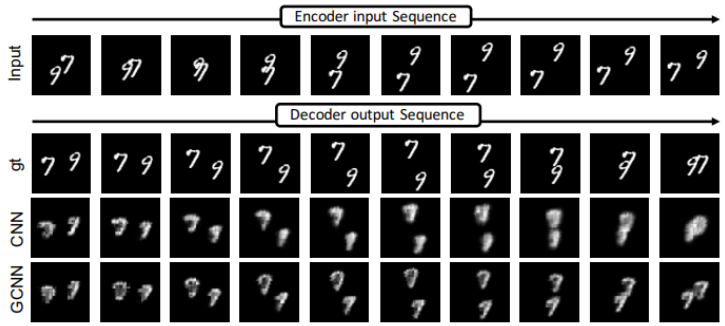
- Penn Treebank
  - Natural language modeling
  - 10,000 one-hot encoding or 200-dimensional dictionary
  - How to construct adjacency matrix A?
    - 4-nearest neighbor graph with cosine distance

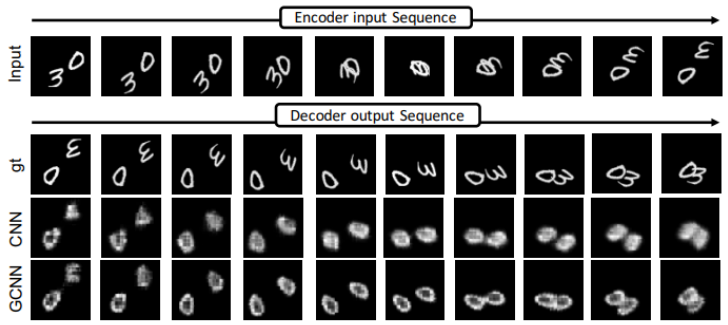




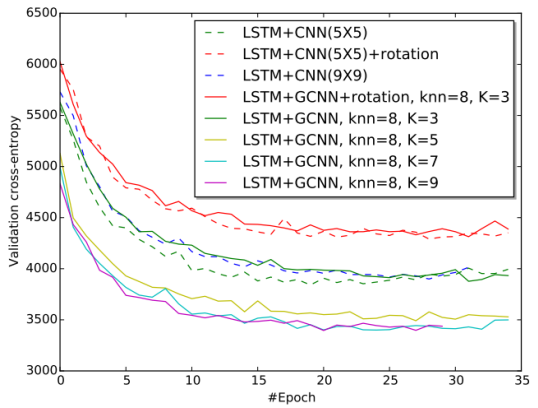
Experiments : Results

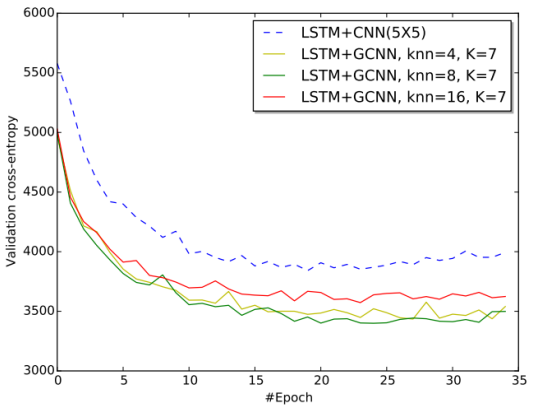
- Moving-MNIST





Architecture	Structure	Filter size	Parameters	Runtime	Test(w/o Rot)	Test(Rot)
FC-LSTM	N/A	N/A	142,667,776	N/A	4832	-
LSTM+CNN	N/A	5×5	13,524,496	2.10	3851	4339
LSTM+CNN	N/A	9×9	43,802,128	6.10	3903	4208
LSTM+GCNN	$knn = 8$	$K = 3$	1,629,712	0.82	3866	4367
LSTM+GCNN	$knn = 8$	$K = 5$	2,711,056	1.24	3495	3932
LSTM+GCNN	$knn = 8$	$K = 7$	3,792,400	1.61	**3400**	**3803**
LSTM+GCNN	$knn = 8$	$K = 9$	4,873,744	2.15	3395	3814
LSTM+GCNN	$knn = 4$	$K = 7$	3,792,400	1.61	3446	3844
LSTM+GCNN	$knn = 16$	$K = 7$	3,792,400	1.61	3578	3963

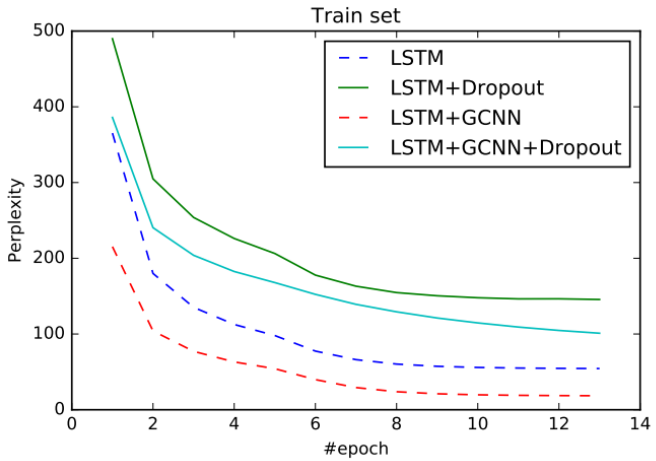


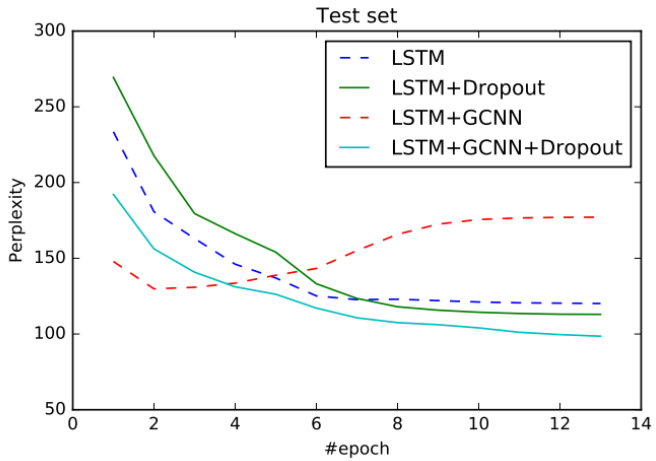


Experiments : Results

- Penn Treebank

Architecture	Representation	Parameters	Train Perplexity	Test Perplexity
Zaremba et al. (2014) code[6]	embedding	681,800	36.96	117.29
Zaremba et al. (2014) code[6]	one-hot	34,011,600	53.89	118.82
LSTM	embedding	681,800	48.38	120.90
LSTM	one-hot	34,011,600	54.41	120.16
LSTM, dropout	one-hot	34,011,600	145.59	112.98
GCRN-M1	one-hot	42,011,602	18.49	177.14
GCRN-M1, dropout	one-hot	42,011,602	114.29	**98.67**





Conclusion

- Spatio-temporal structures from graph-structured and time-varying data
- Challenge
  - Combine simultaneously recurrent neural networks with convolutional neural networks for graph-structured data
- Two solutions
  - Model 1 : Using a stack of CNN and RNN
  - Model 2 : Using convLSTM that considers convolutions instead of fully connected operations in the RNN definition
- Two applications
  - Video prediction
    - Model 2 showed better performance the Shi et al. (2015)
  - Natural language modeling
    - Model 1 showed better performance, in terms of learning speed
- Important points
  - Isotropic filters can outperform classical 2D filters on images while requiring much less parameters
  - Graphs coupled with graph CNN and RNN are a versatile way of introducing and exploiting side-information

Code

- 본 논문의 기본 code는 저자의 github (https://github.com/youngjoo-epfl/gconvRNN)에 제시되어 있는데, test 부분이 아주 약간 미완성 상태여서 그 부분만 수정하여 test를 구현하였습니다.
- 수정 내용은 trainer.py 내부 Trainer class에 존재하지 않았던 test function을 구현한 것이고, 그 수정된 코드는 제 github (https://github.com/andymogul/gcrnn_revised) 에서 확인할 수 있습니다.
- 따라서 주어진 코드의 구조를 review하였습니다.
- 본 코드는 Natural language dataset인 Penn Treebank dataset을 사용하고, python2.7과 tensorflow 1.2를 기반으로 합니다.
- 먼저, 모든 모델의 학습은 trainer.py의 Trainer class에서 진행됩니다.
- 학습에 앞서 해야할 것은 dataset에 adjacency matrix를 구성하는 것입니다.
- Adjacency matrix W는 각 batch단위로 구성되는데, utils.py의 batchloader class에서 데이터 간의 cosine distance를 계산하고, 이를 symmetric하게 만들어 W를 Trainer에 전달합니다. 또한 이후 graph convolution 연산을 위해 laplacian이 필요하기 때문에 이 역시 같이 계산하여 전달합니다. 관련 연산 function들은 graph.py에서 구현하였습니다.
- 이제 Trainer 내에 학습을 위한 model을 구축합니다.
- 모델은 model.py에 Model class로 구현되어 있습니다.
- 본 논문에서 제시하는 모델은 LSTM의 matrix multiplication을 graph convolution으로 대체하는 개념이기 때문에, Model class 내에 build_model function에서 RNN cell을 새롭게 정의한 gconvLSTMcell로 바꿔줍니다. 이 gconvLSTMcell 역시 model.py 내에 class로 정의되어 있습니다.

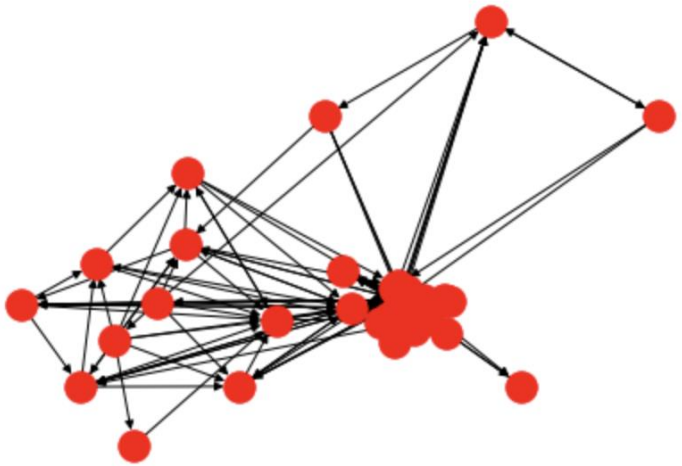
Code

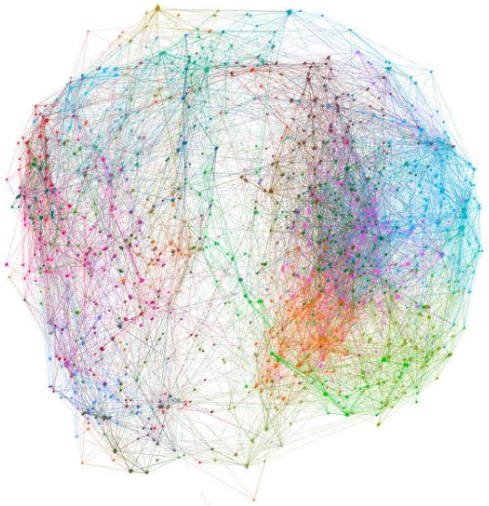
- 그 내부를 살펴보면, gconv 연산은 chevyshev convolution을 사용했는데, 이 역시 model.py 내부에 Function으로 정의하여 사용했습니다.
- 모델의 정의는 앞서와 같이 하고, loss는 tf.nn.sparse_softmax_cross_entropy_with_logits 를 기반으로 cross entropy loss를 사용했습니다.
- Optimizer는 sgd, adam, rmsprop을 사용하고 gradient clipping까지 build_optim function에서 정의하였습니다.
- 즉 여태까지 코드를 정리하면, Model class에서 모델의 구조와 loss, optimizer까지 모두 작성한 상태입니다.
- Trainer class에서는 이제 model에 대한 training과 test를 진행합니다. 둘은 같은 방식으로 진행되기 때문에 train function만을 대상으로 보겠습니다.
- Train function에서는 data를 batch로 받아오고, one hot encoding을 진행합니다. 그 후 인코딩된 batch_x와 batch_y를 feed_dict에 넣어 model class에 대한 training을 진행합니다.

- 이 모든 학습 과정을 담은 class는 gcrn_main.py에 trainer class로 정의되어 있기 때문에 gcrn_main.py를 실행함으로써 학습이 진행됩니다.
- 앞선 설명과 같이, train과 test가 모두 하나의 클래스 안에 정의되어 있기 때문에 training 후에 test를 진행하기 위해서는 config.py에서 is_train을 True에서 False로 변환하여 다시 한번 gcrn_main.py를 실행하는 방식으로 진행됩니다.

Code : Result

- 실험 결과는 다음과 같습니다.
- 먼저 본 실험의 데이터는 Penn Treebank이지만 논문의 본 실험과는 다르게 데이터 크기를 줄이기 위하여 character level로 진행했음을 밝힙니다.
- Character-level의 adjacency matrix를 prior로 구해본 결과는 다음과 같습니다.
- 이는 영문 소문자와 특수문자들을 포함한 50 * 50 adjacency matrix를 시각화한 결과입니다.
- 오른쪽의 논문의 결과는 vocabulary level의 adjacency matrix인데, 왼쪽의 실험 결과는 character level의 결과이기 때문에 우하단의 영문 소문자 부분에 그래프가 주로 집중된 것으로 보입니다.





Code : Result

- 실험의 결과를 inference를 통해 살펴보면, 다음과 같은 Inference 결과를 볼 수 있습니다.
- Input : [12, 6, 3, 18, 7, 5, 21, 0, 14, 3, 12, 2, 0, 16, 20, 3,
- 8, 20, 0, 8, 3, 7, 12, 12, 15]
- Output : [2, 2, 1, 21, 3, 7, 12,
- 8, 32, 3, 29, 3, 29, 26, 10, 5, 3, 24, 1, 2, 12, 1, 5,
- 8, 0]
- 이와 같이 Input character sequence에 대해 output character sequence를 반환하는 과정을 보여줍니다.
- 위의 결과는 character를 digit으로 변환한 것이기 때문에 명확히 보이지 않는데, 이를 character vocab으로 다시 읽을 수 있는 상태로 보겠습니다.

Code : Result

- {'#': 46, '$': 39, '&': 38, "'": 33, '*': 49, '-': 22, '.': 32, '/': 48, '0': 37, '1': 34, '2': 41, '3': 40, '4': 42, '5': 36, '6': 44, '7': 45, '8': 43, '9': 35, '<': 27, '>': 28, 'N': 29, '₩₩': 47, '_': 3, 'a': 0, 'b': 4, 'c': 12, 'd': 21, 'e': 1, 'f': 17, 'g': 19, 'h': 20, 'i': 10, 'j': 30, 'k': 6, 'l': 9, 'm': 18, 'n': 5, 'o': 7, 'p': 24, 'q': 23, 'r': 2, 's': 16, 't': 8, 'u': 15, 'v': 31, 'w': 13, 'x': 25, 'y': 14, 'z': 11, '|': 26}
- Character-level one-hot encoding dictionary는 실험 결과 vocab_char.pkl에 저장되고, 위와 같습니다.
- 따라서 앞선 장에 구한 inference 결과를 위의 dictionary에 대응하면
- Input : ck_monday_crash_that_occu
- Output : rred_oct._N_N|in_percenta
- 으로 해석할 수 있습니다.
- 실제 본 task는 본 논문에 비해 크기가 작은, 난이도가 쉬운 task이기 때문에 test 단계에서 loss는 0인 것을 볼 수 있습니다. 즉, 오류가 아예 없게 완벽하게 학습이 되었고, 그에 맞게 inference가 된 모습입니다.