Market Comment Generation from Data with Noisy Alignments

Yumi Hamazono[1,2] **Yui Uehara**[2] **Hiroshi Noji**[2]
Yusuke Miyao[3,2] **Hiroya Takamura**[4,2] **Ichiro Kobayashi**[1,2]
[1]Ochanomizu University [2]National Institute of Advanced Industrial Science and Technology
[3]The University of Tokyo [4]Tokyo Institute of Technology
{hamazono.yumi, koba}@is.ocha.ac.jp
{yui.uehara, noji,takamura.hiroya}@aist.go.jp
yusuke@is.s.u-tokyo.ac.jp

Abstract

End-to-end models on data-to-text learn the mapping of data and text from the aligned pairs in the dataset. However, these alignments are not always obtained reliably, especially for the time-series data, for which real time comments are given to some situation and there might be a delay in the comment delivery time compared to the actual event time. To handle this issue of possible noisy alignments in the dataset, we propose a neural network model with multi-timestep data and a copy mechanism, which allows the models to learn the correspondences between data and text from the dataset with noisier alignments. We focus on generating market comments in Japanese that are delivered each time an event occurs in the market. The core idea of our approach is to utilize multi-timestep data, which is not only the latest market price data when the comment is delivered, but also the data obtained at several timesteps earlier. On top of this, we employ a copy mechanism that is suitable for referring to the content of data records in the market price data. We confirm the superiority of our proposal by two evaluation metrics and show the accuracy improvement of the sentence generation using the time series data by our proposed method.

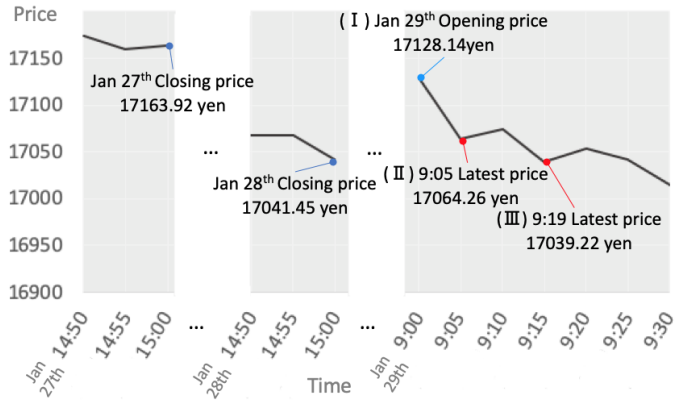
1 Introduction

In recent time, various industries such as finance, pharmaceuticals and telecommunications have increased opportunities to treat large-scale data. Hence, there is an increasing demand to automatically generate a text from large and complex data. In recent studies, neural network-based models have achieved significant progress on the data-to-text which is a text generation task from input data (Puzikov and Gurevych, 2018; Liu et al., 2018; Iso et al., 2019).

One important issue in constructing a dataset for data-to-text is to obtain the correct alignment between data and text. It is not very problematic when there is a clear correspondence between data and text, for example, when the text is manually provided by an annotator for each input, including E2E NLG Challenge dataset (Puzikov and Gurevych, 2018). However, this is often not a trivial problem in the wild, in particular for the application of real-time text generation, such as sequential comment generation on sports games (Taniguchi et al., 2019) and stock markets (Murakami et al., 2017), for which we can only obtain a loose alignment of data and text. This problem has been taken into account in a classical task (Chen and Mooney, 2008), but has been overlooked in the recent neural-based models cited above.

In Murakami et al. (2017), for example, they constructed a dataset for market comment generation in Japanese from a chart of stock price trends and its market reports, wherein the alignments between data and texts sometimes deviate. Figure 1 presents examples of data which are a five-minute chart of Nikkei 225 (Nikkei Stock Average) and comments describing the chart trends. All of the (I) to (Ⅲ) comments are about an event at "9:00 am, 29th of January". However, since there is often a delay in comment delivery time for an event (e.g., (Ⅱ) and (Ⅲ)), if a different movement occurs during this time period, the expressions in comment may not exactly reflect the movement at the *delivery* time. The word *"rebound"* indicates a downward and then upward movement on three points (the closing prices of the last two days and the latest price). This is valid for the prices at (I) and (Ⅱ), but does not hold at (Ⅲ) because the latest price (17039.22 yen) is lower than the last closing price (17041.45 yen). In addition, the expression *"gains 88 yen"* is only valid at the opening time (I) and is not valid at (Ⅱ). To deal with these inconsistencies, the models have to be aware of these possible mismatch of data and text due to the delay, but a simple encoder-decoder-based

Proceedings of The 13th International Conference on Natural Language Generation, pages 148–157,
Dublin, Ireland, 15-18 December, 2020.



	Delivery time	Price movement	**Comment** (bold text: movement expression)
(Ⅰ)	09:00	rebound gained 86 yen	*Nikkei stock average starts with **rebound**, **gains 86 yen**.*
(Ⅱ)	09:05	rebound gained 22 yen	*Nikkei stock average starts with **rebound**, **gains 86 yen**, a positive response in high price of crude oil. The stock price of Toyota increased.*
(Ⅲ)	09:19	continuous fall dropped 2 yen	*Tosho begins with **rebound**. A positive response in high price of crude oil, Fanuc share prices dropped significantly.*

Figure 1: Nikkei 225 and market comments about an event at 9:00 am, 29th of January

model of Murakami et al. (2017), which does not tell the difference between the event and delivery times, may perform an undesirable generalization between data and text.

In this study, we extend Murakami et al. (2017)'s model with a new architecture to solve the problem due to the noisy alignments. The presented architecture is a *multi-timestep* architecture, which employs multiple input vectors to compensate the lack of information about the actual event time, treating it as a hidden variable and learning the correspondences from the ambiguous data. Our model employs a copy mechanism to generate a price value in a text, in which an attention weight to the time can be regarded as an induced alignment between the text and actual event time. The experimental results reveal that our proposed model outperforms the existing model in terms of the correctness of market price movement expressions, in addition to the BLEU scores.

2 Generating Market Comments

In this section, we explain the model proposed by Murakami et al. (2017), which be used as a base model. They proposed a model for generating a market comment, which is a news headline about the movement of the Nikkei 225, from the time-series of the stock price of the Nikkei 225. Their model is based on the encoder-decoder (Sutskever et al., 2014).

2.1 Base Model

Murakami et al. (2017) use both a long-term vector $\boldsymbol{x}_{\text{long}}$ and a short-term vector $\boldsymbol{x}_{\text{short}}$. To capture the long-term price movement, they use a vector consisting of the closing prices of the M preceding trading days represented as $\boldsymbol{x}_{\text{long}} = (x_{\text{long},1}, x_{\text{long},2}, \ldots, x_{\text{long},M})$. Similarly, to capture the short-term price movement, they use a vector consisting of the prices of N previous timesteps on the five-minute chart, starting from the comment delivery time (e.g., 10:00 am), represented as $\boldsymbol{x}_{\text{short}} = (x_{\text{short},1}, x_{\text{short},2}, \ldots, x_{\text{short},N})$. Each of these vectors undergoes the following two preprocessing steps :

$$x_i^{\text{std}} = \frac{x_i - \mu}{\sigma}, \quad (1)$$

$$x_i^{\text{norm}} = \frac{2 \times x_i^{\text{move}} - (\bar{x}_{\max} + \bar{x}_{\min})}{\bar{x}_{\max} - \bar{x}_{\min}}, \quad (2)$$

where x_i is the i-th element of $\boldsymbol{x}$. μ and σ are the mean and the standard deviation of the values in the training data, respectively. x_i^{move} is defined to be $x_i - r_i$, where r_i is the closing price of the previous trading day. $\bar{x}_{\max}$ and $\bar{x}_{\min}$ are the maximum and the minimum of x^{move}, respectively. Equation (1) is a standardization method and Equation (2) is a

normalization method for moving reference. By applying these preprocessings to $\boldsymbol{x}_{\text{long}}$ and $\boldsymbol{x}_{\text{short}}$, we obtain $\boldsymbol{x}^{\text{std}}_{\text{long}}$,$\boldsymbol{x}^{\text{norm}}_{\text{long}}$, $\boldsymbol{x}^{\text{std}}_{\text{short}}$, and $\boldsymbol{x}^{\text{norm}}_{\text{short}}$.

In the encoding step, these vectors are passed to multi-layer perceptrons (MLPs)[1] to obtain the vectors $\boldsymbol{h}^{\text{std}}_{\text{long}}$, $\boldsymbol{h}^{\text{norm}}_{\text{long}}$, $\boldsymbol{h}^{\text{std}}_{\text{short}}$, and $\boldsymbol{h}^{\text{norm}}_{\text{short}}$. These are then concatenated as $\boldsymbol{h}_{\text{long}} = [\boldsymbol{h}^{\text{std}}_{\text{long}} ; \boldsymbol{h}^{\text{norm}}_{\text{long}}]$ and $\boldsymbol{h}_{\text{short}} = [\boldsymbol{h}^{\text{std}}_{\text{short}} ; \boldsymbol{h}^{\text{norm}}_{\text{short}}]$. These vectors are combined to obtain the hidden state $\boldsymbol{m}$ of the encoder:

$$\boldsymbol{m} = \boldsymbol{W}_m[\boldsymbol{h}_{\text{long}} ; \boldsymbol{h}_{\text{short}}] + \boldsymbol{b}_m. \quad (3)$$

In the decoding step, Murakami et al. (2017) set the initial hidden state $\boldsymbol{s}_0$ of the decoder as $\boldsymbol{m}$ above, and use LSTM cells (Hochreiter and Schmidhuber, 1997). They further use additional vectors called *time embeddings* $\boldsymbol{t}$. To obtain $\boldsymbol{t}$, time is discretized into intervals of one hour (e.g., 9:00 am to 10:00 am), and an embedding is obtained for each interval; 9:10 am and 9:30 am are associated with the same embedding. The time embedding of the interval, wherein the delivery time of the comment falls into is used as an additional input in each step of LSTM:

$$\boldsymbol{s}_i = \text{LSTM}([\boldsymbol{t} ; \boldsymbol{w}_{i-1}], \boldsymbol{s}_{i-1}). \quad (4)$$

As in the standard LSTM decoder, this output is fed into a linear layer, followed by a softmax layer to calculate the next word probability.

2.2 Generalization Tags for Estimation of Arithmetic Operations

Market comments often mention numerical values including the market prices themselves and the values obtained through arithmetic operations such as difference and rounding. To allow the model to generate numerical values with such computation during decoding, Murakami et al. (2017) introduce generalization tags (Table 1), which specify which operation should be performed to obtain a value. In Murakami et al. (2017), for simplicity, the input value to these operations is fixed as the $x_{\text{short},1}$, the first (latest) price at the delivery time, which is converted to z and Δ in Table 1 (see caption).

However, this simplification ignores the possible mismatch of the delivery and event times (Section 1). We extend this method with a variant of a copy mechanism (Section 3.2), in which the event time is softly predicted with attention and each numerical value is generated on demand during decoding with an operation in Table 1.

[1]Murakami et al. (2017) used MLPs, convolutional neural networks (CNNs) and long short-term memory networks (LSTMs). We use MLPs in the encoder for our baseline model because the differences of the performance were small in their experiment.

Tag	Arithmetic operation
<operation1>	Return Δ
<operation2>	Round down Δ to the nearest 10
<operation3>	Round down Δ to the nearest 100
<operation4>	Round up Δ to the nearest 10
<operation5>	Round up Δ to the nearest 100
<operation6>	Return z as it is
<operation7>	Round down z to the nearest 100
<operation8>	Round down z to the nearest 1,000
<operation9>	Round down z to the nearest 10,000
<operation10>	Round up z to the nearest 100
<operation11>	Round up z to the nearest 1,000
<operation12>	Round up z to the nearest 10,000

Table 1: Generalization tags and corresponding arithmetic operations. z is defined as the latest price, and Δ is defined as the difference between z and the closing price of the previous trading day.

3 Multi-timesteps for Time-series Data

Murakami et al. (2017)'s model put an assumption that the event and delivery times are identical. This assumption simplifies the task and thus Murakami et al. (2017) propose a model with a basic encoder-decoder architecture. However, due to the time gap between the actual event time and delivery time, this assumption is not realistic.

We extend the Murakami et al. (2017)'s model with a multi-timestep architecture, aiming at solving that problem occurred by the noisy alignments. Figure 2 presents our model architecture. To compensating the lack of information caused by the time gap mentioned above, we extend the encoder with additional input vectors. Each additional vector corresponds to $\boldsymbol{x}_{\text{short}}$ starting from k preceding timesteps instead of the delivery time. This allows us to treat the actual event time as a latent variable. On top of this, we introduce a copy mechanism with attention in the decoder, which facilitates learning correspondences between data and text from the noisy training data.

3.1 Encoder with Multi-timesteps

For long-term stock prices, we use a long-term vector $\boldsymbol{x}_{\text{long}}$ following Murakami et al. (2017). For short-term stock prices, instead of $\boldsymbol{x}_{\text{short}}$ alone, we use $\boldsymbol{x}_{\text{short-0step}}, \ldots, \boldsymbol{x}_{\text{short-}n\text{step}}$ which we abbreviate as $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_n$ for brevity. Each $\boldsymbol{x}_k$ is a short-term vector consisting of the stock prices of N timesteps,

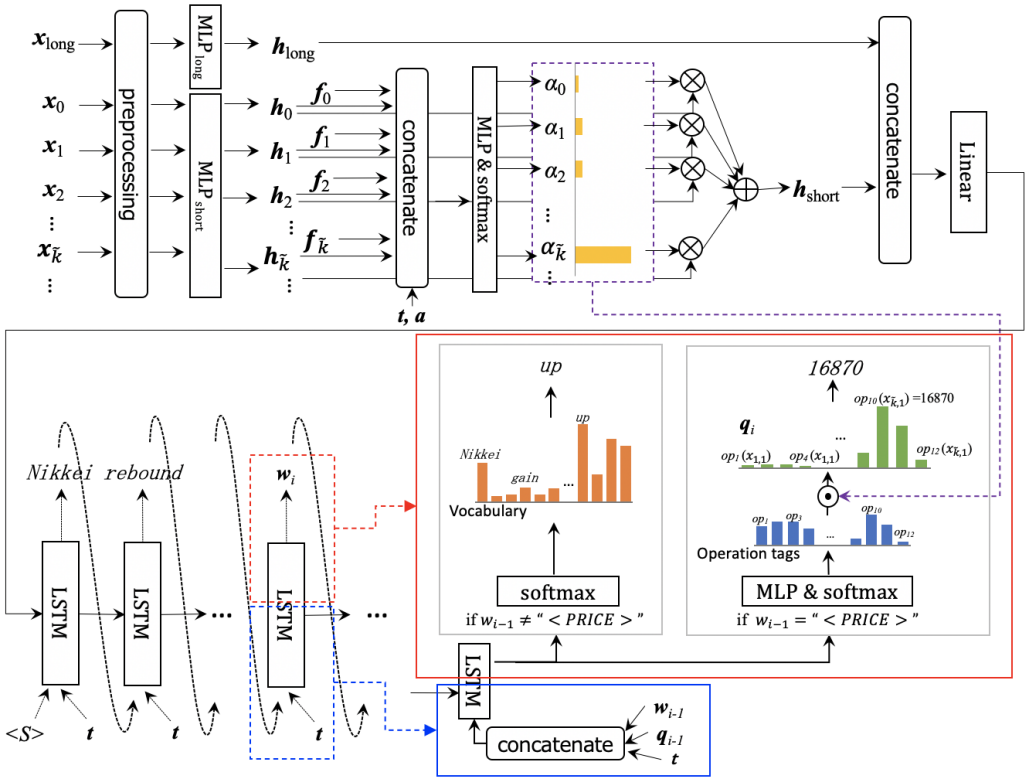


Figure 2: Overview of Multi-timesteps for Time-series Data

which, instead of starting from the comment delivery time, starts from the time k steps prior to the delivery time. When the delivery time is 10:00 am, for example, since we use the five-minute chart, our short-term vectors comprise of $n+1$ vectors corresponding to the vectors starting from 10:00 am, 9:55 am, 9:50 am, and so on.

We note that each interval of five minutes is not simply associated with one value, but rather with four different values: open (the price at the very beginning of the interval), low (the lowest price in the interval), high (the highest price in the interval), and close (the price at the last of the interval).[2] We encode these as four different vectors $\boldsymbol{x}_{k_{\text{open}}}$, $\boldsymbol{x}_{k_{\text{low}}}$, $\boldsymbol{x}_{k_{\text{high}}}$, and $\boldsymbol{x}_{k_{\text{close}}}$. Thus, there are in total $4(n+1)-1$ vectors for short-term prices. In the following, we use $\tilde{k} \in [0, 4(n+1))$ as an index for these vectors.

Each of the input vectors undergoes the preprocessing methods as Murakami et al. (2017). For long-term vectors, we obtain $\boldsymbol{x}^{\text{std}}_{\text{long}}$ and $\boldsymbol{x}^{\text{norm}}_{\text{long}}$ in the same way. Similarity, for each $\tilde{k}$, we obtain $\boldsymbol{x}^{\text{std}}_{\tilde{k}}$ and $\boldsymbol{x}^{\text{norm}}_{\tilde{k}}$ from $\boldsymbol{x}_{\tilde{k}}$. Given these, each MLP emits the corresponding hidden states $\boldsymbol{h}_{\text{long}}$ and $\boldsymbol{h}_{\tilde{k}}$.

[2]Murakami et al. (2017) use only the close prices. We use four values instead since prices could largely move even in a single interval.

Following Equation (3), we also obtain the encoder hidden state $\boldsymbol{m}$ with $\boldsymbol{h}_{\text{long}}$ and $\boldsymbol{h}_{\text{short}}$. The vector $\boldsymbol{h}_{\text{short}}$ depends on a sequence of $\boldsymbol{h}_{\tilde{k}}$ to which an encoder maps the input data. It is computed as a weighted sum of $\boldsymbol{h}_{\tilde{k}}$ as follows:

$$\boldsymbol{h}_{\text{short}} = \sum_{\tilde{k}=0}^{4(n+1)-1} \alpha_{\tilde{k}} \boldsymbol{h}_{\tilde{k}}. \tag{5}$$

The weight $\alpha_{\tilde{k}}$ of each $\boldsymbol{h}_{\tilde{k}}$ is computed by:

$$\alpha_{\tilde{k}} = \exp(e_{\tilde{k}}) / \textstyle\sum_{j=0}^{4(n+1)-1} \exp(e_j). \tag{6}$$

$\boldsymbol{e} = (e_0, \cdots, e_{4(n+1)-1})$ is obtained by

$$\text{MLP}\left([\boldsymbol{t}\,;\boldsymbol{a}\,;\boldsymbol{f}_0\,;\boldsymbol{h}_0\,;\ldots;\boldsymbol{f}_{4(n+1)-1}\,;\boldsymbol{h}_{4(n+1)-1}]\right). \tag{7}$$

$e_{\tilde{k}}$ scores the importance of $\boldsymbol{x}_{\tilde{k}}$, based on the hidden states $\boldsymbol{h}_{\tilde{k}}$, time embedding vector $\boldsymbol{t}$ (Section 2.1), and additional two kinds of vectors, *five-minute time embedding* vectors $\boldsymbol{f}_{\tilde{k}}$ and *article-type embedding* vector $\boldsymbol{a}$.

$\boldsymbol{f}_{\tilde{k}}$ is an embedding to identify $\tilde{k}$, which maps from the starting time and the kind of price (e.g., open) to a vector. $\boldsymbol{a}$ is a vector obtained whether the comment is regular or irregular. The motivation to use these vectors is that an important $\boldsymbol{x}_{\tilde{k}}$ is

primarily determined by either the price history (encoded by $\boldsymbol{h}_{\tilde{k}}$), or the delivery time (encoded by $\boldsymbol{f}_{\tilde{k}}$), depending on the article type. For instance, when the delivery time is the same as the market opening time, that is, 9:00 am, and the comment is a regular comment, then the comment usually mentions the price at 9:00 am (e.g., "*The opening price is 15,430 yen.*"). In this case, the important $\boldsymbol{x}_{\tilde{k}}$ is primarily determined by the time encoded by $\boldsymbol{f}_{\tilde{k}}$; that is, for regular comments, the event times are rather fixed regardless of the variations in delivery times. On the other hand, even if the delivery time is the same as the market opening time, when the comment is irregular, the comment mentions some distinguished price movement (e.g. "*The price is over 100 yen higher than the last closing price.*") rather than the mere price at 9:00 am. In this case, the important $\boldsymbol{x}_{\tilde{k}}$ would be determined by the price history itself. We expect these additional vectors to provide a useful inductive bias for a model to learn those distinctions.

3.2 Decoder with Copy Mechanism

We adapt a copy mechanism (Gu et al., 2016) in our decoder to generate numerical values by attending to the input. Recall that in the current task, the values in the output text usually do not appear in the input data (Section 2.2); rather, they can be obtained by applying an arithmetic operation to the certain value in the data.

We generate a numerical value by an extension of a copy mechanism, wherein a value is generated by applying one of the operations in Table 1 to a data point $x_{\tilde{k},1}$, which is the first value (latest price) of $\boldsymbol{x}_{\tilde{k}}$. Denoting an arithmetic operation as $o \in \{op_1, \cdots, op_{12}\}$, the value is identified by a pair $(o, \tilde{k})$. We reduce the generation of numerical values to identification of these pairs, followed by the execution of an operation.

We wish to obtain a probability distribution on numerical values that are determined by $(o, \tilde{k})$. One consideration for obtaining this is that there can be multiple pairs of $(o, \tilde{k})$ that result in the same value. For example, for an input $x_{\tilde{k},1} = z = 3200$, op_6 op_7, and op_{10} all result in the same value (3200). In practice, we obtain a probability to generate a numerical value w_i by a weighted sum of scores for $(o, \tilde{k})$, for which $o(x_{\tilde{k},1}) = w_i$ holds, according to the obtained weights $\alpha_{\tilde{k}}$ in the encoder, which we regard as *attention*. $o(\cdot)$ denotes execution of operation o to the input. We note that unlike the standard copy mechanism, we use the fixed attention weights $\alpha_{\tilde{k}}$ throughout the decoder. This is because, in a headline comment, which is our target, the event to be mentioned would not change throughout a single piece of text. In other words, the important $\boldsymbol{x}_{\tilde{k}}$ would not change throughout a comment.

Our model generates all numerical values in the text with this copy mechanism. To do this, we exclude numerical values from the vocabulary of the model. To switch the copy mode and non-copy mode, we add a special token *"<PRICE>"*, which is inserted before every numerical value in the training data and indicates that the next token is a price value. Utilizing *"<PRICE>"*, we define each conditional probability of generating target word w_i at time i as:

$$p(w_i|w_{<i}, \boldsymbol{m}) = \begin{cases} p_{\text{COPY}}(w_i|w_{<i}, \boldsymbol{m}), & (w_{i-1} = \text{"<PRICE>"}) \\ p_{\text{GEN}}(w_i|w_{<i}, \boldsymbol{m}), & (\text{otherwise}) \end{cases} \tag{8}$$

where $p_{\text{GEN}}(w_i|\cdot)$ and $p_{\text{COPY}}(w_i|\cdot)$ are obtained by the *generation mode* and the *copy mode*, respectively. This method is inspired by *Pointer-generator network* introduced by See et al. (2017). These two probabilities are defined as:

$$p_{\text{GEN}}(w_i|\cdot) = [\text{softmax}(\boldsymbol{W}_v \boldsymbol{v}_i + \boldsymbol{b}_v)]_{w_i} \tag{9}$$

$$p_{\text{COPY}}(w_i|\cdot) = \sum_{\tilde{k}, o: o(x_{\tilde{k},1}) = w_i} q(\tilde{k}, o) \tag{10}$$

$$q(\tilde{k}, o) = \alpha_{\tilde{k}} \cdot [\text{softmax}(\boldsymbol{W}_c \boldsymbol{c}_i + \boldsymbol{b}_c)]_o. \tag{11}$$

$\boldsymbol{v}_i$ and $\boldsymbol{c}_i$ are both obtained from the output of decoder LSTM, $\boldsymbol{s}_i$, at each step i:

$$\boldsymbol{s}_i = \text{LSTM}([\boldsymbol{t}\,; \boldsymbol{w}_{i-1}\,; \boldsymbol{q}_{i-1}], \boldsymbol{s}_{i-1}) \tag{12}$$

$$\boldsymbol{v}_i = \boldsymbol{W}_h \boldsymbol{s}_i + \boldsymbol{b}_h \tag{13}$$

$$\boldsymbol{c}_i = \text{MLP}(\boldsymbol{s}_i). \tag{14}$$

$\boldsymbol{t}$ is the time embedding defined in Section 2.1. Comparing to Equation (4), we add $\boldsymbol{q}_i$ to the input to the LSTM. Each element of $\boldsymbol{q}_i$ is $q(\tilde{k}, o)$. We add this vector to properly propagate the information about an applied arithmetic operation, which may not be kept directly in $\boldsymbol{s}_i$.

4 Experiments

4.1 Datasets

We used a five-minute chart of the Nikkei 225 from March 2013 to October 2016 as numerical

Data	Movement Expression			Total
	None	Exist		
		Concord	Diff	
Train	3,522	11,172	341	15,035
Valid	378	1,346	35	1,759

Table 2: Statistics of the data

time-series data, which were collected from Thomson Reuters DataScope Select[3]. As market comments, we used 18,489 headlines of Nikkei Quick News (NQN) that describe the Nikkei 225 behavior. They were provided by Nikkei, Inc. and written in Japanese. We divided them into three parts on the basis of the period of publication: 15,035 for training (December 2010–October 2015), 1,759 for validation (October 2015–April 2016) and 1,695 for testing (April–October 2016).

4.2 Experimental Settings

All MLPs in the model are three layers with hidden dimension of 256. The decoder LSTM is a single layer with hidden dimensions of 256. For the length of short- and long-term vectors, we set $M = 7$ for $\boldsymbol{x}_{\text{long}}$ and $N = 62$ for $\boldsymbol{x}_{\tilde{k}}$, following Murakami et al. (2017), changing the range of k by setting $n \in [0, 6]$. The embedding sizes of a word, five-minute tag $\boldsymbol{f}_{\tilde{k}}$, article-tag $\boldsymbol{a}$, and time tag $\boldsymbol{t}$ are 128, 80, 64, and 64, respectively. We trained the models for 150 epochs with the mini-batch size of 100, using Adam (Kingma and Ba, 2015) optimizer with the initial learning rate 1×10^{-4}, and saved the parameters every epoch, selecting the model with the highest BLEU score on the validation dataset.

4.3 Evaluation Metrics

We conduct two types of evaluation: one is BLEU (Papineni et al., 2002) to measure the matching degree between the market comments written by humans as references and the output comments generated by the models, and the other is a new evaluation metric that we created. The new metric uses the matching between the market price movement in the data and the movement expressions in the comments. Using $(\boldsymbol{x}_i, \boldsymbol{w}_i^{gold}, \boldsymbol{w}_i^{pred})$, which are the i-th sample of the input data, the market comment written by humans, and the output comment generated by the models, we define the following variables:

$$move_{text}(\boldsymbol{w}) = \begin{cases} upward & (\boldsymbol{w} \in w_{\text{rise}}) \\ downward & (\boldsymbol{w} \in w_{\text{fall}}) \\ None & (otherwise) \end{cases}$$

$$move(\boldsymbol{x}) = \begin{cases} upward & (\text{Latest } x^{\text{move}} > 0) \\ downward & (\text{Latest } x^{\text{move}} < 0) \\ None & (otherwise) \end{cases}$$

$$\boldsymbol{C_{gold}} = \{i | move_{text}(\boldsymbol{w}_i^{gold}) = move(\boldsymbol{x}_i)\}$$
$$\boldsymbol{C_{pred}} = \{i | move_{text}(\boldsymbol{w}_i^{pred}) = move(\boldsymbol{x}_i)\}$$
$$\boldsymbol{D_{gold}} = \{i | move_{text}(\boldsymbol{w}_i^{gold}) \neq move(\boldsymbol{x}_i)\}$$
$$\boldsymbol{D_{pred}} = \{i | move_{text}(\boldsymbol{w}_i^{pred}) \neq move(\boldsymbol{x}_i)\}$$

where w_{rise} is {続伸 (*continuous rise*), 反発 (*rebound*), 上げ (*up / rise*)}, w_{fall} is {続落 (*continuous fall*), 反落 (*fall back*), 下げ (*down/fall*)}, x^{move} is the same as x_i^{move} shown in Section 2.1, defined as $x - r$, where r is the closing price of the previous trading day. Using the above variables, we obtain the following metrics:

$$\begin{aligned} \textbf{Concord}_{precision} &= \frac{|\boldsymbol{C_{gold}} \cap \boldsymbol{C_{pred}}|}{|\boldsymbol{C_{pred}}|} \\ \textbf{Concord}_{recall} &= \frac{|\boldsymbol{C_{gold}} \cap \boldsymbol{C_{pred}}|}{|\boldsymbol{C_{gold}}|} \\ \textbf{Diff}_{precision} &= \frac{|\boldsymbol{D_{gold}} \cap \boldsymbol{D_{pred}}|}{|\boldsymbol{D_{pred}}|} \\ \textbf{Diff}_{recall} &= \frac{|\boldsymbol{D_{gold}} \cap \boldsymbol{D_{pred}}|}{|\boldsymbol{D_{gold}}|}. \end{aligned}$$

These metrics can be seen as a proxy for the model's ability to attend to an intermediate (not latest) step according to the movement in the data. To know the frequency of these concordances and differences in the data, we count them in the training and valid dataset, which we summarize in Table 2.

We evaluate the BLEU scores for both validation and test sets, while we performed the correspondence evaluation, which will be described below, only for the validation set. We train models with six different seeds for each setting and report the macro averages on them.

4.4 Results

Table 3 summarizes the BLEU scores on the validation and test sets, and Table 4 presents the correspondence evaluation on the validation set.

[3] https://hosted.datascope.reuters.com/DataScope/

	Valid	Test	Valid	Test	Valid	Test
Baseline	21.37	21.30	-	-	-	-
Multi-timesteps			**+Attention**		**+Copy**	
$n = 0$	21.63	22.66	21.33	22.38	28.16	28.68
1	21.59	22.74	21.43	22.25	27.90	28.54
2	21.64	23.03	21.20	22.47	**28.31**	**28.98**
3	21.82	23.11	21.02	21.94	27.29	27.75
4	21.68	22.92	20.59	21.71	28.13	28.93
5	21.73	22.89	20.71	21.74	27.29	27.78
6	21.73	22.66	20.51	21.62	26.68	27.25

Table 3: BLEU (%)

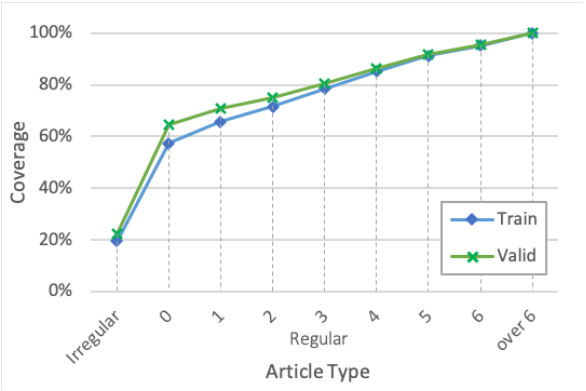


Figure 3: Data distribution regarding the time gap between the comment delivery time and the event occurrence time. First, we classified a human-written market comment in the dataset into two types, whether it was a regular comment (*Regular*) or not (*Irregular*) using the expressions specific to regular comments. In the case of a regular comment, it is classified by the time gap between that event occurrence time and the comment delivery time in five-minute increments, for example, 0 = from 0 min to 5 min gap, 1 = from 5 min to 10 min gap.

According to the experimental results provided in Table 3, our models mostly outperformed the baseline, especially when used with the copy mechanism. For the test set, our models always outperformed the baseline. In particular, $n = 2$ with the copy mechanism achieved the highest score, 7.68 points improvement on the BLEU score compared to the baseline (see the **bold** font in Table 3). Conversely, the result shows that just adding attention to the encoder (+Attention), keeping the decoder the same as the baseline, is not helpful. It shows that simply applying the attention mechanism does not enable the model to obtain the correspondence between data and text correctly, while the copy mechanism certainly helps to obtain the correspondence. In a comparison among the models using $n = 6$, increasing the number of steps does not necessarily contribute to improving the BLEU score.

Furthermore, Table 4 indicates that most of our models outperformed the baseline. However, in the same tendency as the BLEU score evaluation, this result further reveals that increasing the number of steps is not necessarily an important factor in improving the score.

Figure 3 depicts distributions regarding the time gap between the comment delivery time and the event occurrence time. When we use $n = 5$, we can cover 89.05% in training data and 94.07% in validation set of their time gap data (i.e., sum of an *Irregular* value and 0 to 5 of *Regular* values in Figure 3). Alternatively, using more $\boldsymbol{x}_{\tilde{k}}$ would add more noise, therefore, they would be related to the transactions. According to Table 3, Table 4 and Figure 3, using $n = 3$ or $n = 4$, covered around 80% are considered as the best choice in this dataset.

Table 5 provides examples of the generated comments where our model generated the correct movement expression while the baseline generated an incorrect expression (see, the **bold** font in Table 5). Moreover, the method with copy mentioned the correct numerical value, while the baseline generated an incorrect numerical value. However, there is another problem with the fluency. For instance, at the expression of "*in the higher XX yen range*", the numerical value *XX* should be a round number, e.g., round to the nearest 100, 1,000, or 10,000. However, the model with the copy mechanism generates a specific number (see, the underline font in Table 5)

5 Related Work

The task of generating text describing input data, which is called data-to-text, has been worked on various domains, for instance weather forecasts (Belz, 2007; Angeli et al., 2010), healthcare (Portet et al., 2009; Banaee et al., 2013), and sports (Liang et al., 2009). Traditionally, data-to-text is divided into two sub-problems (Kukich, 1983; Goldberg et al., 1994): *content selection*, which is about *"what to say"*, and *surface realization*, which is about *"how to say"*. Moreover, Reiter and Dale (1997) divides three modules, adding *micro planning* between the above sub-problems. In the early stage of this task, *surface realization* is often realized using templates (van Deemter et al., 2005) or statistically learned models with hand-crafted features (Belz, 2008; Konstas and Lapata, 2012).

In recent times, various industries such as finance, pharmaceuticals, and telecommunications

	Concord		Diff		Concord		Diff		Concord		Diff	
	P	R	P	R	P	R	P	R	P	R	P	R
Baseline	98.68	98.78	27.57	24.72	-	-	-	-	-	-	-	-
Multi-timesteps					**+Attention**				**+Copy**			
$n = 0$	98.61	98.94	41.00	33.93	98.90	99.08	51.35	45.28	98.85	99.25	53.90	42.61
1	98.77	99.02	46.03	40.52	98.76	99.28	56.83	41.91	98.81	99.02	45.86	39.80
2	98.79	98.47	40.66	46.13	98.99	98.78	45.00	49.74	98.71	99.06	47.08	38.56
3	98.82	98.61	43.12	46.20	98.93	98.87	44.65	46.25	98.73	98.95	45.38	38.79
4	98.88	98.86	46.58	47.05	98.80	98.44	30.60	36.29	98.68	98.68	36.46	36.06
5	98.88	98.91	44.13	43.47	98.69	98.14	27.88	34.92	98.57	98.57	34.04	31.54
6	98.80	98.70	41.88	43.43	98.70	97.11	25.54	38.74	98.46	98.81	32.35	26.17

Table 4: The correspondence evaluation on validation set (%, P=precision, R=recall)

Model	Generated Comment	Conc
Gold	Nikkei opens with a **continual rise**. The price is 17024 yen, 9 yen higher. 日経平均、続伸で始まる 9円高の17024円	×
Baseline	Nikkei opens with a **fall back**. The price is 16900 yen level. 日経平均、反落で始まる 16900円台	✓
Multi-timesteps		
$n = 3$	Nikkei opens with a **continual rise** slightly. 日経平均、小幅続伸で始まる	×
+Attention	Nikkei opens with a **continual rise** slightly. 日経平均、小幅続伸で始まる	×
+Copy	Nikkei opens with a **continual rise** slightly. The price is in the higher 17024 yen range. 日経平均、小幅続伸で始まる 17024円台後半	×

Table 5: Generated comment. The latest stock price movement of comment delivery time is *-48.91 yen*, so that $move(\boldsymbol{x})$ is $downward$. **Conc** (concordance) row shows whether the movement expression in the comment matches the movement with the latest stock price (✓) or not (×).

have been increased providing opportunities to treat various types of large-scale data, so that they are interested in automatically learning a correspondence relationship from data to text and generating a description of this relationship. Therefore, recent works have focused on generating text from data with neural networks, that can solve the above sub-tasks in one through. Especially the models, which utilize an encoder-decoder model (Sutskever et al., 2014) have proven to be useful (Mei et al., 2016; Lebret et al., 2016). While text generation by neural network can describe the text fluently, they do not describe the exact entities or numbers. Therefore, a copy mechanism (Vinyals et al., 2015; Gu et al., 2016), which provides a way to directly copy words from the input, has been utilized. By these neural networks, the works such as conditional language generation based on tables (Yang et al., 2017), short biographies generation from Wikipedia tables (Lebret et al., 2016; Chisholm et al., 2017; Sha et al., 2018; Liu et al., 2018) and sports scoreboards (Wiseman et al., 2017; Li and Wan, 2018; Puduppully et al., 2019) are well performed. Contrastingly, they can only generate the superficial contents that appear in their input table, and cannot generate contents that require arithmetic operations.

However, Joulin and Mikolov (2015) and Neelakantan et al. (2016) indicate that current neural models have difficulties in learning arithmetic operations such as addition and comparisons by neural program inductions. Thus, there have been some methods to prepare the numerical values with arithmetic operations in advance. Murakami et al. (2017) post-process the price by extending the copy mechanism and replacing numerical values with defined arithmetic operations after generation. Nie et al. (2018) utilizes information from pre-computed operations on raw data to consider incorporating the facts that can be inferred from the input data to guide the generation process. Our model prepares numerical values with defined arithmetic operations as Murakami et al. (2017) for copy and that copy target is guided by encoded input.

6 Conclusion

In this paper, we have proposed an encoder-decoder model with multi-timestep data and a copy mechanism for generating the market comment from data with the noisy alignments. Both BLEU scores and our proposal evaluation showed the accuracy of sentence generation with time-series data has been improved by our proposed method, especially utilizing a copy mechanism.

Acknowledgements

This paper is based on results obtained from projects commissioned by the New Energy and Industrial Technology Development Organization (NEDO) JPNP20006 and JPNP15009, and JSPS KAKENHI Grant Number JP20H04217.

References

Gabor Angeli, Percy Liang, and Dan Klein. 2010. A simple domain-independent probabilistic approach to generation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 502–512, Cambridge, MA. Association for Computational Linguistics.

Hadi Banaee, Mobyen Uddin Ahmed, and Amy Loutfi. 2013. Towards NLG for physiological data monitoring with body area networks. In *Proceedings of the 14th European Workshop on Natural Language Generation*, pages 193–197, Sofia, Bulgaria. Association for Computational Linguistics.

Anja Belz. 2007. Probabilistic generation of weather forecast texts. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 164–171, Rochester, New York. Association for Computational Linguistics.

Anja Belz. 2008. Automatic generation of weather forecast texts using comprehensive probabilistic generation-space models. *Natural Language Engineering*, 14(4):431–455.

David L. Chen and Raymond J. Mooney. 2008. Learning to sportscast: A test of grounded language acquisition. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 128–135, New York, NY, USA. Association for Computing Machinery.

Andrew Chisholm, Will Radford, and Ben Hachey. 2017. Learning to generate one-sentence biographies from Wikidata. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 633–642, Valencia, Spain. Association for Computational Linguistics.

Kees van Deemter, Mariet Theune, and Emiel Krahmer. 2005. Real versus template-based natural language generation: A false opposition? *Computational Linguistics*, 31:15–24.

Eli Goldberg, Norbert Driedger, and Richard I Kittredge. 1994. Using natural-language processing to produce weather forecasts. *IEEE Expert*, 9(2):45–53.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1631–1640, Berlin, Germany. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Hayate Iso, Yui Uehara, Tatsuya Ishigaki, Hiroshi Noji, Eiji Aramaki, Ichiro Kobayashi, Yusuke Miyao, Naoaki Okazaki, and Hiroya Takamura. 2019. Learning to select, track, and generate for data-to-text. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2102–2113, Florence, Italy. Association for Computational Linguistics.

Armand Joulin and Tomas Mikolov. 2015. Inferring algorithmic patterns with stack-augmented recurrent nets. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 190–198. Curran Associates, Inc.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Ioannis Konstas and Mirella Lapata. 2012. Unsupervised concept-to-text generation with hypergraphs. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 752–761, Montréal, Canada. Association for Computational Linguistics.

Karen Kukich. 1983. Design of a knowledge-based report generator. In *21st Annual Meeting of the Association for Computational Linguistics*, pages 145–150, Cambridge, Massachusetts, USA. Association for Computational Linguistics.

Rémi Lebret, David Grangier, and Michael Auli. 2016. Neural text generation from structured data with application to the biography domain. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1203–1213, Austin, Texas. Association for Computational Linguistics.

Liunian Li and Xiaojun Wan. 2018. Point precisely: Towards ensuring the precision of data in generated texts using delayed copy mechanism. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1044–1055, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Percy Liang, Michael Jordan, and Dan Klein. 2009. Learning semantic correspondences with less supervision. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language*

Processing of the AFNLP, pages 91–99, Suntec, Singapore. Association for Computational Linguistics.

Tianyu Liu, Kexiang Wang, Lei Sha, Baobao Chang, and Zhifang Sui. 2018. Table-to-text generation by structure-aware seq2seq learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*.

Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. 2016. What to talk about and how? selective generation using LSTMs with coarse-to-fine alignment. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 720–730, San Diego, California. Association for Computational Linguistics.

Soichiro Murakami, Akihiko Watanabe, Akira Miyazawa, Keiichi Goshima, Toshihiko Yanase, Hiroya Takamura, and Yusuke Miyao. 2017. Learning to generate market comments from stock prices. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1374–1384, Vancouver, Canada. Association for Computational Linguistics.

Arvind Neelakantan, Quoc V. Le, and Ilya Sutskever. 2016. Neural programmer: Inducing latent programs with gradient descent. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Feng Nie, Jinpeng Wang, Jin-Ge Yao, Rong Pan, and Chin-Yew Lin. 2018. Operation-guided neural networks for high fidelity data-to-text generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3879–3889, Brussels, Belgium. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

François Portet, Ehud Reiter, Albert Gatt, Jim Hunter, Somayajulu Sripada, Yvonne Freer, and Cindy Sykes. 2009. Automatic generation of textual summaries from neonatal intensive care data. *Artificial Intelligence*, 173(7):789 – 816.

Ratish Puduppully, Li Dong, and Mirella Lapata. 2019. Data-to-text generation with entity modeling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2023–2035, Florence, Italy. Association for Computational Linguistics.

Yevgeniy Puzikov and Iryna Gurevych. 2018. E2E NLG challenge: Neural models vs. templates. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 463–471, Tilburg University, The Netherlands. Association for Computational Linguistics.

Ehud Reiter and Robert Dale. 1997. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Lei Sha, Lili Mou, Tianyu Liu, Pascal Poupart, Sujian Li, Baobao Chang, and Zhifang Sui. 2018. Order-planning neural text generation from structured data. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.

Yasufumi Taniguchi, Yukun Feng, Hiroya Takamura, and Manabu Okumura. 2019. Generating live soccer-match commentary from play data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7096–7103.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc.

Sam Wiseman, Stuart Shieber, and Alexander Rush. 2017. Challenges in data-to-document generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2253–2263, Copenhagen, Denmark. Association for Computational Linguistics.

Zichao Yang, Phil Blunsom, Chris Dyer, and Wang Ling. 2017. Reference-aware language models. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1850–1859, Copenhagen, Denmark. Association for Computational Linguistics.