Input Conditional Language Models using Long Short Term Memory Networks

Akshay Balaji, Sai Praveen, JS Suhas, and Rakesh R Menon

Indian Institute of Technology Madras, Chennai, India

Abstract. Several techniques have been used in the past to generate weather forecast texts using Natural Language Generation. This work presents a novel method using Long Short Term Memory Networks to produce weather forecasts using raw inputs. We also present 4 different variants of our method and show how they can be used as standalone systems to generate text without the need for classical NLG phases. We evaluate our method using several automated metrics and present visualizations for learned models.

1 Introduction

Natural Language Generation refers to the task of generation natural language based on some form of input data. The problem of generating semantically correct, coherent sentences is a rather tough problem that is being solved today. Due to the inherent expressiveness and variations contained in natural language it is often very hard to find sound data to train on and reliable metrics to measure performance of NLG models. After a comprehensive survey of NLG across domains such as speech, video, image and text, we have observed that the general trend being the problem is split up into three parts. Namely content determination, wherein the content that is supposed to be generated is discovered/extracted. Content planning, wherein the layout of the text generated is decided upon, including sentence formation. And finally surface realization that ensures that grammatically correct sentences are generated.
In most cases, each of these tasks are carried out by a separate model and then aggregated into one. We also observed that while most of the prominent papers in the above mentioned fields included a lot of Machine Learning in the content determination stage, they mostly relied on template based methods for surface realization and content planning.
In this project, we attempt to substitute these template based methods and create an end-to-end trained model that is capable of studying the data and learning to generate relevant content in one shot while dealing with intricate problems (which include learning the correlation between data and text inputs as well as the rules of the input language) and to output semantically correct and rich, coherent sentences. Here, we have applied this domain independent model in the field of weather generation. This model has been based on the principles of one of the upcoming fields in Machine Learning called Deep Learning.

2 Related Work

Recurrent Neural Networks were shown to be capable tools to be used in language modeling in [16]. The paper showed the difficulties faced in traditional feed-forward networks (like ease of training, ability to take in only a fixed number of predecessor words into 'memory') could be overcome by the use of LSTMs., which estimate probabilities based on full history and have fewer weight matrices resulting in large speedups of training and testing whereas suffering very minute decreases in performance.
Our venture into this project was possible due to a paper published on Semantically Controlled LSTM (SC-LSTM)[19]. It introduces a natural language generator that is able to produce coherent sentences by combining sentence planning and surface realization using the cross-entropy loss function working on unaligned data. This paper claimed to successfully cover multiple domains in terms of improving performance and is free of heuristics and explicit rules defined.
The LSTM cell in the figure below is in charge of surface realization has cell

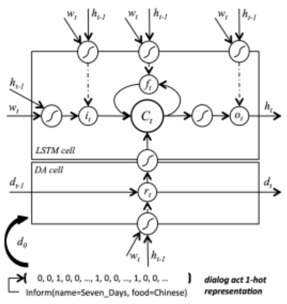
LSTM cell
DA cell
0, 0, 1, 0, 0, ..., 1, 0, 0, ..., 1, 0, 0, ...
Inform(name=Seven_Days, food=Chinese)
dialog act 1-hot representation

Fig. 1. Semantically Controlled LSTM(SC-LSTM)

state C_t which hold the memory of the LSTM which interacts with hidden states h_t and words w_t to produce sentences that are semantically correct and coherent (more about LSTMs in section 3). The figure also describes a Dialogue Act(DA) cell that is used for surface planning, feeds information into the LSTM cell as a one hot vector representation. This is used to ensure that the intended meaning is put across in the generated sentences.
Further, to apply our work into something tangible, we have referenced SumTime [15], a project that gives a generic computational model that is used to summarize weather data. The weather summary system takes time series data as input which contains fields relating to wind direction, wind speed etc. and determines the content in the summary by only picking up significant data points (those that are worth reporting) from the data and building up a very simple template-based model in order to push out summaries.
As with most papers we surveyed across video, audio, text and speech domains

with respect to NLG, most of them are related to the content determining stage. For example, most models that output image description either collect content by searching for visually similar images or by categorizing the image to have pre-specified objects and attributes. After this stage, the surface planning stage is mostly carried out by template based techniques. We aim to combine these into one whole system. [19] states that future work can be done in terms of making a single unified architecture which is able to perform an end-to-end training on the data which is what we set out to do for this project.

3 Background

3.1 RNN

Recurrent neural networks[14] are a class of artificial neural networks that are capable of handling sequential data as opposed to vanilla neural networks. A recurrent neural network consists of network loops which allows information to persist in subsequent computations. This is particularly useful in the case of natural language processing where we need the context of previous words in order to predict what the next word should be. The recurrent neural network can further be visualised as multiple copies of the same network passing information from one cell to another. An illustrative diagram of recurrent networks is shown below.

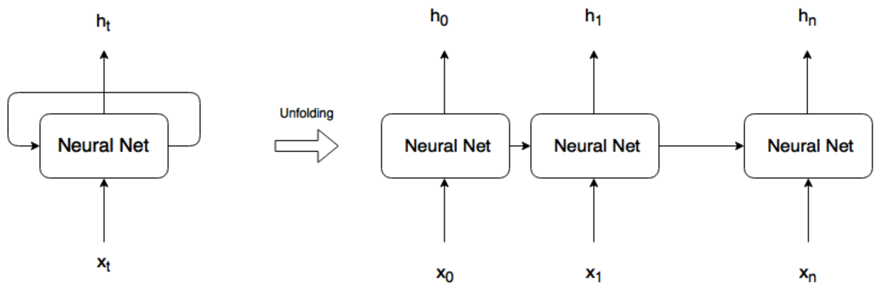

Fig. 2. Recurrent Neural Network

Another powerful feature of recurrent neural network is the ability to take in a variable input and produce an output of variable size as well. This has powerful implications and over the decades they have found massive applications in speech processing[5][3], image captioning [7][18] and natural language processing[2][8]. However, recurrent neural networks too have some limitations. For example, it is not useful in the case where we only need a portion of the previous information is required in order to predict the output. Additionally, they also suffer from

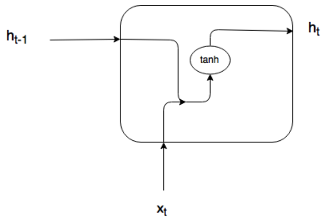

Fig. 3. Internal Structure of a RNN

the vanishing gradient problem as addressed by Bengio in [1]. These limitations have been addressed by a modification on the RNN structure called as an Long Short-Term Memory(LSTM), which is addressed in the next section.

3.2 LSTM

A Long Short-Term Memory(LSTM)[4] is a modification of recurrent neural networks which is able to control the amount of information that is passed on from one cell to another and hence are able to address the issue of long-term dependencies. The control of information is done by the use of 4 internal neural networks for each cell instead of one(in RNNs). The LSTM architecture is as follows:

The most important component for LSTM is the cell state which basically

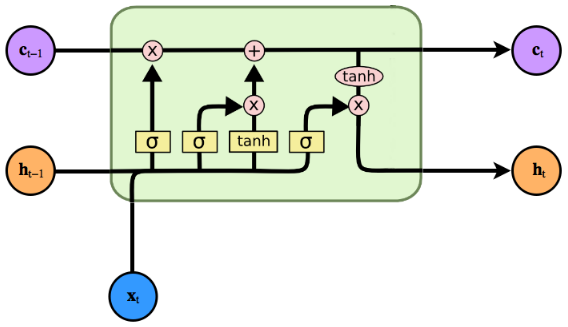

Fig. 4. Internal Structure of LSTM. Figure adapted from `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

carries the information from one cell to the next. Gates are designed to alter the information that is carried by the cell state. The first gate is the forget gate which takes as input h_{t-1} and x_t. The sigmoid output of the neural network provides a score between 0(completely forget) and 1(completely keep) which controls how much of the previous information is kept in subsequent computations. The next part consists of an input gate layer and decides what new information is going to be stored in the cell state. Finally the output of the neural network is taken on the basis of the current cell state and the inputs h_{t-1} and x_t. The equations for the LSTMs are as follows:

$$\begin{aligned}
\boldsymbol{f}_t &= \sigma(\mathbf{W}_f[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f) \\
\boldsymbol{i}_t &= \sigma(\mathbf{W}_i[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i) \\
\tilde{\boldsymbol{c}}_t &= \sigma(\mathbf{W}_c[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_c) \\
\boldsymbol{c}_t &= \boldsymbol{f}_t * \boldsymbol{c}_{t-1} + \boldsymbol{i}_t * \tilde{\boldsymbol{c}}_t \\
\boldsymbol{o}_t &= \sigma(\mathbf{W}_o[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o) \\
\boldsymbol{h}_t &= \boldsymbol{o}_t * tanh(\boldsymbol{c}_t)
\end{aligned}$$

The ability of the LSTMs to overcome the issue of long-term dependencies meant that vanishing gradient problem[1] has also been suppressed. LSTMs and several of its variants have found large applications in speech processing[3] and natural language processing[17][13].

3.3 Language Model LSTM (LM-LSTM)

One issue with plain LSTM is ambiguity in the start continues over time steps. Take the following example. Say, for the first word, the LSTM is unsure and assigns probabilities of $[0.5, 0.5]$ for two words, $[w_1, w_2]$. Since we sample from this distribution, say, we choose w_1. LSTM has no idea that we have chosen w_1 and it must generate the next word from a distribution $Pr(w'|w_1)$. It'll instead generate a vector encoding the distribution, $0.5 \cdot Pr(w'|w_1) + 0.5 \cdot Pr(w'|w2)$. This makes the LSTM more unsure of the next word. If $Pr(w'|w_1)$ assigns probabilities $[0.5, 0.5]$ to $[w_3, w_4]$ and $Pr(w'|w_2)$ assigns probabilities $[0.5, 0.5]$ to $[w_5, w_6]$, and if $[w_3, w_4] \cap [w_5, w_6]$, the new distribution is $[0.25, 0.25, 0.25, 0.25]$ for $[w_3, w_4, w_5, w_6]$. This uncertainty can propagate through time and result in bad sentences. To overcome this problem, [16] proposed feeding the chosen word's embedding as input to the LSTM cell for the next time step. [16] also shows that this model obtains very low error on the English Treebank-3 and Quaero French corpora.
We derive motivation from this modeling technique and use $\boldsymbol{x}_t = emb(w_{t-1})$ as input to the LSTM cell at time step t where $emb(\cdot)$ produces embeddings for its arguments and w_{t-1} was the word sampled at time step $t-1$. We call this class of models Language Model LSTM, or LM-LSTM in short.

4 Modified LSTMs

4.1 Motivation

Our methods are aimed at forming an input conditional language model and thus effectively remove intermediate representation used in NLG systems. Traditional NLG systems have the following phases.

1. **Content Determination -** In this phase, the content to be presented to the user is determined. In case of Weather Forecast texts, this could be weather parameters like Temperature, Wind, Humidity, chances of a cyclone, etc. Data post this phase is represented using domain-specific schemas and trees[12].
2. **Document Planning -** Given some content, this phase determines the order in which the content is to presented to the user. In case of Weather Forecast texts, this would involve deciding which of the parameters will be talked about in the first sentence, second sentence, and so on. Data post this phase is represented using abstract representations[12].
3. **Surface Realization -** Given a document plan, this phase constructs a coherent and grammatically correct sentence. In case of Weather Forecast texts, this would involve deciding what words to use and what conjunctions to use, etc. Output of this phase is a complete, coherent sentence.

LSTMs are able to eliminate the need for separate domain-specific intermediate representations by using one embedding that is learned from the data. This eliminate the need to determine the right intermediate representations for each NLG phase. Compared to intermediate representations obtained in classical NLG methods, LSTM embeddings, however, lack human interpretability. They also eliminate the need for three phases as they perform all three phases internally. Treating natural language as a sequence in some embedding space, LSTMs can learn subtle dependencies between words, both short-range and long-range. Due to the aforementioned reasons, LSTMs present themselves as a very attractive class of language models.
Traditional LSTMs take random noise as input and generate grammatically correct sentences. This presents a problem as we would like to leverage the modeling capabilities of LSTMs to generate grammatically correct sentences describing some input data. Deriving intuition from [19], we designed the following variants of vanilla LSTMs by presenting input data in different forms.

- **Running-Input LSTM(RI-LSTM)** - Our first attempt at obtaining input conditioned language models. Takes the data as input at every stage
- **Running-Input Language Model LSTM (RILM-LSTM)** - Integrates LM-LSTM with Running-Input LSTM
- **Input-Initialized LM-LSTM (IILM-LSTM)** - Improves upon RILM-LSTM by leveraging memory aspect of LSTMs
- **Read Only Memory RILM-LSTM -** Combines a read-only variant of memory-networks with RILM-LSTM to address the memory issue in a different way.
- **Word2Vec RILM-LSTM -** Combines dense representations of Word2Vec with RILM-LSTM to address scalability and variability issues with the model

We'll now present each variant in detail.

4.2 Running-Input LSTM

In Running-Input LSTM (RI-LSTM), we provide the data, $\boldsymbol{x}$, as the cell input, $\boldsymbol{x}_t$, for every time step t. We do this to improve rate of learning as the effect

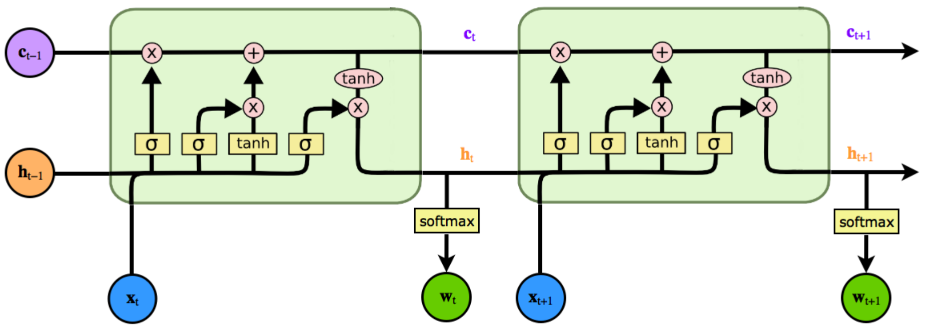

Fig. 5. Running-Input LSTM.

of the input can be easily computed without going through multiple time steps. The architecture for RI-LSTM is shown in 5. In our case, $\boldsymbol{x}_t$ is a $|X_D|$ dimension vector. We convert the hidden state, $\boldsymbol{h}_t$ to a $|V|$ dimensional vector using a linear layer using

$$\boldsymbol{w}_t = softmax(\mathbf{W}_w \cdot \boldsymbol{h}_t + \boldsymbol{b}_w)$$

where $\mathbf{W}_w$ is a $|H| \times |V|$ matrix. $\boldsymbol{w}_t$ gives a probability distribution over all the words in the vocabulary. To obtain a word, w_t, we sample from the distribution given by $\boldsymbol{w}_t$.
RI-LSTM is trained in an end-to-end fashion using cross-entropy as the loss function.

4.3 Running-Input LM-LSTM

In Running-Input LM-LSTM (RILM-LSTM), we extend LM-LSTM to use $\boldsymbol{x}$ as the cell input, $\boldsymbol{x}_t$, at every time step. Rest of the semantics remain identical to LM-LSTM.
Motivation to use RILM-LSTM remain identical to the motivation to use LM-LSTM over RILM-LSTM will be an improvement over RI-LSTM as an LM-LSTM can use generated words, in previous time steps, as context to generate a new word.
In RILM-LSTM, the input vector $\boldsymbol{x}_t$ is given by

$$\boldsymbol{x}_t = [\boldsymbol{x}, emb(v_{t-1})]$$

where w_{t-1} was word sampled at time $t-1$ and $emb(\cdot)$ gives embedding in $|V|$ dimensions. We initialize $emb(w_0) = \mathbf{0}$.
RILM-LSTM train and test architectures are illustrated in Figures 6 and 7.
Like RI-LSTM, RILM-LSTM is also trained end-to-end using cross-entropy as the loss function.

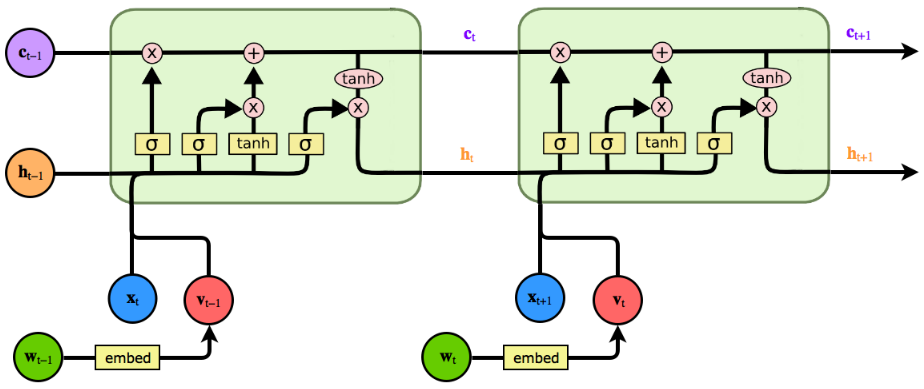

Fig. 6. Running-Input LM-LSTM Train Architecture

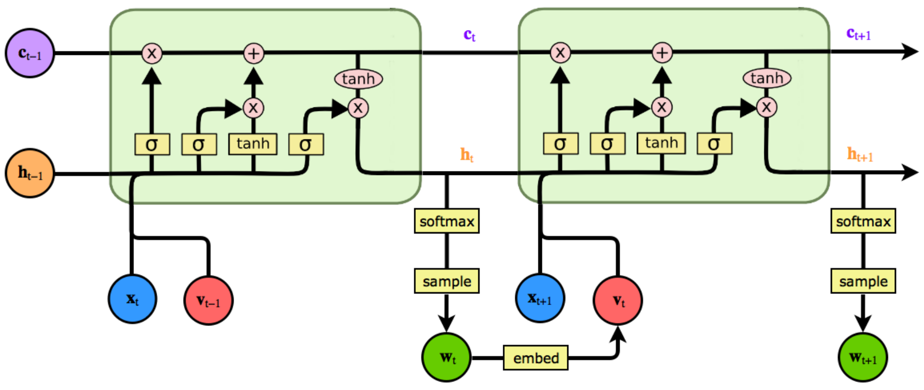

Fig. 7. Running-Input LM-LSTM Test Architecture

4.4 Input-Initialized LM-LSTM

One issue with using Running Input variants is that the input has to be provided at every time step to obtain a word. Since LSTM networks have an inherent *memory* in the form of cell state, $\boldsymbol{c}_t$, we should be able to generate a sentence conditioned on a given input $\boldsymbol{x}$ if we could somehow encode $\boldsymbol{x}$ into $\boldsymbol{c}_0$. Input-Initialized LM-LSTM (IILM-LSTM) does precisely that.
In IILM-LSTM, we define the relationship between $\boldsymbol{X}$ and $\boldsymbol{c}_0$ as

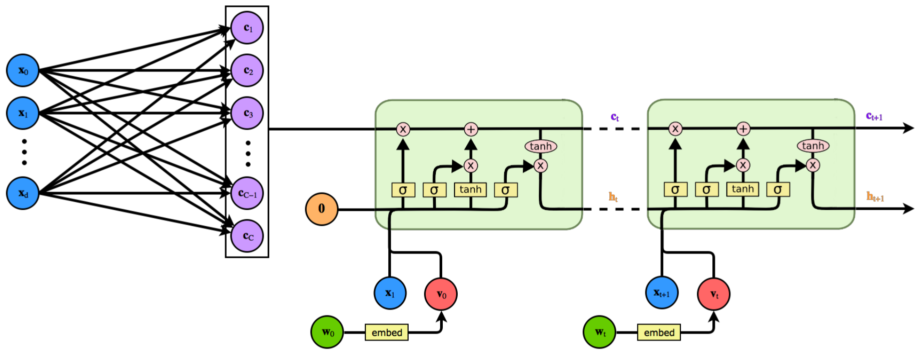

Fig. 8. IILM-LSTM Architecture

$$\boldsymbol{c}_0 = \mathbf{W}_{cx} \cdot \boldsymbol{x} + \boldsymbol{b}_{cx}$$

where $\mathbf{W}_{cx}$ is of dimensions $|H| \times |X_D|$.
IILM-LSTM is motivated by the notion that the forget gates in an LSTM act as filters to remove information already encoded as generated text. In other words, after the forget gate has acted, the remaining information in $\boldsymbol{c}$ is the information yet to be encoded in future words. As the LSTM generates words, it removes the information represented by these words from $\boldsymbol{c}$ using the hidden state $\boldsymbol{h}$. $\boldsymbol{h}_t$ can be interpreted as the information represented by the words already generated in time steps $[0 \ldots t-1]$. Interaction of $\boldsymbol{c}_t$ and $\boldsymbol{h}_t$ removes the information generated before time step t.
IILM-LSTM is trained end-t-end using cross-entropy as the loss function. One subtle difference between IILM-LSTM and RILM-LSTM is that the input is provided only once to the system, and hence, the gradient takes longer to propagate the effect of the input to the first cell when compared to the RILM-LSTM case where input is readily available at each time step.

5 Advanced Modifications

5.1 Read Only Memory RILM-LSTM

This is a variant of RILM-LSTM with a Read-Only Memory Network memory component. In order to handle *input reconstruction*, we have used a read-only

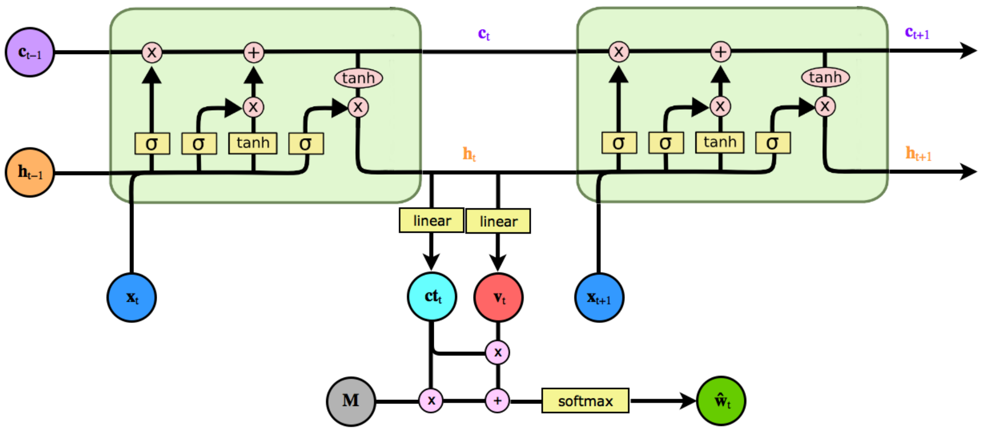

Fig. 9. Read Only Memory RILM-LSTM Training Architecture

memory cell. This is strongly influenced by Memory Networks, a commonly used paradigm for Deep Neural Networks that allows the network to storeretrieve memory bits. The wonderful thing about memory networks is that it can be trained end to end with the gradient propagating through the memory mechanism. Thus, the neural network learns what parts to store and when to retrieve a cell.
We adopt a simplified version of the Memory Network for our purposes. Our memory unit cannot be written to. It is pre-populated with all the tokens that are available in the *input*. For instance, in Prodigy-METEO corpus:

```
[[1,_NNE,22,26,36,-,0600],[2,_NNW,14,18,-,-,1800]]
NNE 22-26 GUSTS 36 BACKING NNW 14-18 BY EVENING
```

We have the NNE and the numerical values reproduced in the output. To facilitate this, we populate the Memory network with $6 * 3 = 18$ elements where each of the elements holds the word encoding for it's corresponding input word.[Note that the first element of each entry, *index* is discarded.] The following equations use the same notation as above. In addition to those, M represents the Memory matrix with dimensions 18xW ($m = 18$ and W is the vocabulary size). Here we add a memory vector as an addition to improve our model. The equations being

$$\boldsymbol{v}_t = \mathbf{U}_w \cdot \boldsymbol{h}_t + \boldsymbol{b}_w$$

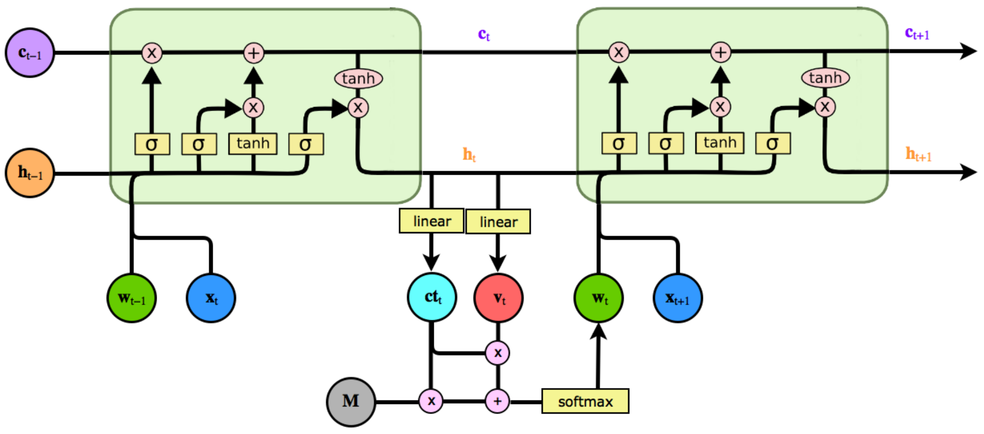

Fig. 10. Read Only Memory RILM-LSTM Testing Architecture

We send the LSTM hidden state through another linear layer to produce 2 weight vectors. These weight vectors are used to selectively show attention to the memory. The memory weight vector $\boldsymbol{r_{0:m-1}}_t$ is used for soft selecting memory cells(the vector is dotted with the memory matrix which holds m memory components each of which is a one hot vector representing a single token). A second vector $\boldsymbol{r_m}$ is computed through another linear layer that acts as a selection weight for it's own memory-independent output $\boldsymbol{v}_t$.

$$\boldsymbol{r_m}_t = \mathbf{U}''_w \cdot \boldsymbol{h}_t + \boldsymbol{b''}_w$$

$$\boldsymbol{r_{0:m-1}}_t = \mathbf{U}'_w \cdot \boldsymbol{h}_t + \boldsymbol{b'}_w$$

which refers to the context which is then combined with the memory to obtain

$$\boldsymbol{y}_t = softmax(\boldsymbol{r_{0:m-1}}_t \cdot \boldsymbol{M} + \boldsymbol{r_m}_t \boldsymbol{v}_t)$$

The training and testing architectures for ROM-RILM-LSTM are showing in Figures 9 and 10. In the *Results* section, we show a heatmap of memory attention at each step of the generation. Note that because of stochastic sampling, the heatmap may not correspond exactly to the output it generated. However, we picked a heatmap that had high probability weight.

5.2 Word2Vec RILM-LSTM

One drawback of RILM-LSTM is that it uses sparse representations(one-hot encoding) for all words in the vocabulary. Because of this, the LSTM output can be interpreted as probabilities for class labels with labels drawn from $\{1 \ldots |V|\}$ where V is the vocabulary. This advantage could be overshadowed by the disadvantage due to sparseness of the representation when V is a very huge vocabulary.

One way to get around this would be to introduce dense representations into the learning system. Word2Vec is one of the most popularly used models to obtain dense representations for a given vocabulary.
Word2Vec, first proposed in [9], is a group of models used to produce dense word embeddings. These models are usually shallow, 2-layer neural networks. In our work, we use the Word2Vec model developed by [9]. We also use the *gensim* library, developed by [11], to interface the model with Python.
We use 303-dimensional embeddings of the words in our vocabulary in the Word2Vec RILM-LSTM. The Word2Vec model gives us a 300-dimensional embedding and we add 3 extra dimensions to represent comma(","), end-of-sentence(".") and end-of-paragraph("\n").
One consequence of using Word2Vec is the ability to generate words not in the input vocabulary. This is made possible by the use of dense embedding where similarity in word concept translates to closeness in the embedding.

It is to be noted that the embeddings generated by Word2Vec model are not only indicative of the word similarities, but it's possible to obtain words from a mixture of their embeddings.
To train the Word2Vec RILM-LSTM, we can no longer use the cross-entropy function as the output vector cannot be interpreted as a probability distribution anymore. We instead use the L_2 loss function, defined by

$$Loss(\mathbf{Y}, \hat{\mathbf{Y}}) = (\mathbf{Y} - \hat{\mathbf{Y}})^2$$

where $\hat{\mathbf{Y}}$ and $\mathbf{Y}$ denote the predicted and actual word embeddings respectively. Sampling in this model is done differently from the other proposed models due to the nature of the output. Algorithm 1 shows how a word can be sampled from the model output at any time t.

Algorithm 1 Word2Vec Sampling

```
1: procedure W2V_SAMPLE($\boldsymbol{v}$)                                  ▷ Returns a word sampled from $\boldsymbol{v}$
2:     $l \leftarrow len(\boldsymbol{v})$
3:     list($(w_k, s_k)$) $\leftarrow$ GetClosest($[v_1, v_2 \dots v_{l-3}]$)         ▷ $s_k$ - similarity score
4:     list($(w_k, e^{-s_k^2})$ $\leftarrow$list($(w_k, s_k)$)                                   ▷ Decay Scores
5:     list($(w_k, s_k)$) $\leftarrow$ [list($(w_k, e^{-s_k^2})$, $(comma, v_{l-2})$, $(eos, v_{l-1})$, $(eop, v_l)$]   ▷ Concat
6:     $L$ =list($(w_k, \max(0, s_k))$ $\leftarrow$list($(w_k, s_k)$)                                ▷ Clip
7:     Normalize $L$
8:     Sample a $w_t$ from L using $s_k$ as probabilities
9:     return $w_t$
```

6 Datasets

6.1 Generated Datasets

To mitigate the dearth of datasets in our domain, we generated two datasets for proof of concept.

SynA SynA contains 1000 examples of the form $\langle \boldsymbol{x}, \textit{sentence} \rangle$. All x_i's are normalized and lie in $[0, 1]$. $\boldsymbol{x}$ is a 3-dimensional floating point vector and *sentence* is a paragraph composed of 3 sentences. Each of sentence in the paragraph corresponds to description of a weather parameter in $\boldsymbol{x}$. $\boldsymbol{x}$ contains the weather parameters Temperature, Wind Speed and Humidity, in that order.
Some examples of $\boldsymbol{x}$ are shown below.

```
temp=0.6376, wind=0.5242, humid=0.0255
temp=0.4536, wind=0.6791, humid=0.6129
temp=0.7004, wind=0.1415, humid=0.5412
```

The corresponding *sentence* are also shown below.

```
It is going to be a warm day today. It is going to be a windy day
today. It is going to be a very dry day today.
It is going to be a cool day today. It is going to be a windy day
today. It is going to be a humid day today.
It is going to be a warm day today. It is going to be a very still
 day today. It is going to be a humid day today.
```

SynA is an easy dataset with about 20 words in the vocabulary. We used this dataset to test our implementations.

SynB SynB contains 50000 examples of the form $\langle \boldsymbol{x}, \textit{sentence} \rangle$. Similar to SynA, in this datset, all x_i's are normalized and lie in $[0, 1]$. $\boldsymbol{x}$ is a 5-dimensional floating point vector and *sentece* is single sentence describing the parameters in $\boldsymbol{x}$. For a given $\boldsymbol{x}$, its *sentence* is not guaranteed to describe all the parameters in $\boldsymbol{x}$. $\boldsymbol{x}$ contains the weather parameters Temperature, Humidity, Wind Speed, Precipitation Chance and Day, in that order. To make the learning task more challenging than SynA, sentences are also sometimes paraphrased. Some examples of $\boldsymbol{x}$ are shown below.

```
temp=0.9621, humid=0.7801, wind=0.0629, prec=0.1035, day=1.0000
temp=0.4909, humid=0.1642, wind=0.1185, prec=0.9938, day=0.0000
temp=0.9076, humid=0.0124, wind=0.1530, prec=0.0485, day=0.0000
temp=0.3830, humid=0.5024, wind=0.9015, prec=0.0455, day=1.0000
```

The corresponding sentences are also shown below.

```
it is going to be a very humid, hot day tomorrow.
today is going to be a very cool, dry day with thunderstorms and
hailstones.
```

```
it is going to be a very dry and hot day today without any traces
of rain or heavy winds.
tomorrow is going to be a humid, cool day with cyclonic winds.
```

SynB is expected to be harder to learn. Simple machine learning systems will find it hard to learn the paraphrasing, but using the LSTMs, it should be possible. Also, since some sentences omit describing certain parameters, it fits in well with the probabilistic sampling from the LSTM output vector.

6.2 Prodigy-METEO

The Prodigy-METEO dataset, available online for free is a collection of hand-written and machine-written Marine weather forecast snippets that focus mainly on wind patterns. The input is a time series of the following fields:

```
[index, wind_direction,speed_low, speed_high, gust_low, gust_high,
 timestamp]
```

A sample from this would look like:

```
[[1,_SSW,16,20,-,-,0600],[2,_SSE,-,-,-,-,-1],[3,_VAR
,04,08,-,-,0000]]
SSW 16-20 GRADUALLY BACKING SSE THEN FALLING VARIABLE 04-08 BY
LATE EVENING
```

or a simpler one with just 2 data points:

```
[[1,_NNE,22,26,36,-,0600],[2,_NNW,14,18,-,-,1800]]
NNE 22-26 GUSTS 36 BACKING NNW 14-18 BY EVENING
```

Note that the output text also contains bits of the *input* text(*SSE, NNE, 22-26*) along with *observations*(*GRADUALLY BACKING, FALLING*) which need some processing to be done on the input data. This is in addition to simple *translations*(*BY EVENING*) that are scattered across the forecasts. Our RILM-LSTM and IILM-LSTM models are theoretically capable of handling the *observations* and *translations* but cannot handle the *input* reconstruction because the LSTMs would simply attempt to map every possible input to generate it's corresponding output without any generalization i.e it would not semantically map the inputs. It would see an SSE or NNE as a one hot vector representing some number $\langle x \rangle$ and $\langle y \rangle$ but not as the semantic $\langle wind - direction \rangle$. In order to allow for this, we introduced the Read Only Memory LSTM(ROM-LSTM).

Some minor changes were made to the corpus including(none were manual, all were simple automated scripts):

- All data entries with 4 or more elements in the time series were discarded. Our current capabilities cannot handle this.
- The direction tags in the inputs were modified to remove the leading '_'.
- Items with strange tags like 'CVAR', 'M_NE' etc for wind direction were discarded.

– Inputs with 2 or lower time series entries were automatically padded with a NULL(all fields 0) entry.
– Unavailable inputs were marked with a 'mask' bit.

The final dataset included: 422(out of an initial 493) valid and cleaned input/output pairs out of which 60% were used as training data and the others were used as validation and testing data. The final data looked like this:

```
E-ENE 12 16 0 0 600 NE-NNE 18 22 30 0 1200 NIL 0 0 0 0 0 1 1 1 0 0
 1 1 1 1 1 0 1 0 0 0 0 0 0
E-ENE 12 - 16 BACKING NE-NNE 18 - 22 GUSTS 30
```

7 Results

7.1 BLEU metric

BLEU metric, developed by Papineni[10], is measure of the quality of the text that has been produced by an algorithm for machine translation. The metric compares the n-grams obtained from each of the candidate sentences to some human(good quality) reference sentences translations. Usually to measure the quality of translation the uni-gram probability is calculated. To this end, the metric will calculate the number of matching words with the reference sentence and counts the number of matches. The final output is a probability of the number of matches over the total number of words. It should also be noted that the matches are position-independent.
The BLEU metric also introduces a modified n-gram precision, where the maximum number of occurrences of a word in the reference sentences is taken into account and clips the maximum possible number of occurrences of the same word in the candidate sentences. Furthermore, the length of a sentence is penalized by the use of *brevity penalty* factor. Whenever the length of the candidate sentences is less than or equal to the maximum length of the reference sentences, then penalty factor is kept at 1. Combining all these factors, the final BLEU metric takes the weighted geometric mean of the precision scores $\boldsymbol{p}_n$, weighted by positive weights $\boldsymbol{w}_n$, of the test corpus and the result is multiplied by the exponential *brevity penalty* factor.
Finally, trying to formulate the all the theories for the BLEU metric, we have for c as length of translation and r as effective length of reference corpus,

$$\boldsymbol{p}_n = \frac{\sum_{\mathbb{C} \in Candidates} \sum_{n\_gram \in} Count_{clip}(n\_gram)}{\sum_{\mathbb{C}' \in Candidates} \sum_{n\_gram \in \mathbb{C}'} Count(n\_gram)}$$

$$BP = \begin{cases} 1 & c > r \\ e^{(1-\frac{r}{c})} & c \leq r \end{cases}$$

$$BLEU = BP \cdot exp(\sum_{n=1}^{N} \boldsymbol{w}_n \log \boldsymbol{p}_n)$$

In this paper, we have used two BLEU metrics to evaluate our weather generation system.

- **Translation Score:** The first metric is used to check the uni-gram translation BLEU score. Using the above formulae, we have for this metric $\boldsymbol{p}_n$ as,
$$\boldsymbol{p}_n = \frac{\sum_{\mathbb{C} \in Candidates} \sum_{1\_gram \in} Count_{clip}(1\_gram)}{\sum_{\mathbb{C}' \in Candidates} \sum_{1\_gram \in \mathbb{C}'} Count(1\_gram)}$$
- **Grammar Score:** The second metric is used to measure the grammatical correctness of the sentences and for the same purpose, we propose to compare the trigrams generated by the candidate sentences with that of a different corpus. For the purpose of this project, we have taken the camera review corpus from NLTK[6] as our reference corpus. The resulting $\boldsymbol{p}_n$ is,
$$\boldsymbol{p}_n = \frac{\sum_{\mathbb{C} \in Candidates} \sum_{3\_gram \in} Count_{clip}(3\_gram)}{\sum_{\mathbb{C}' \in Candidates} \sum_{3\_gram \in \mathbb{C}'} Count(3\_gram)}$$

7.2 Training Plots

Generated Dataset Curves in Figures 11,12, and 13 show definite improvements on the SynA and SynB datasets.

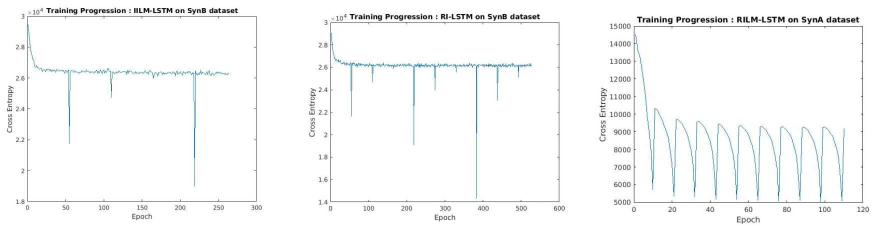

Fig. 11. Training Curves on SynA and SynB datasets

Human Authored Dataset Figures 14 and 15 show learning and saturation on the Prodigy-METEO dataset.

7.3 Examples

The BLEU metric can only be so effective in NLG applications. This is mainly because of several possible outputs all of which we cannot enumerate and in the datasets we have used(both machine generated and human-authored), each input contains a single possible output. Thus, generalization will have to be measured manually by humans. To this end, we present some of the text generated by our systems

Somewhere halfway during training on the SynB dataset.

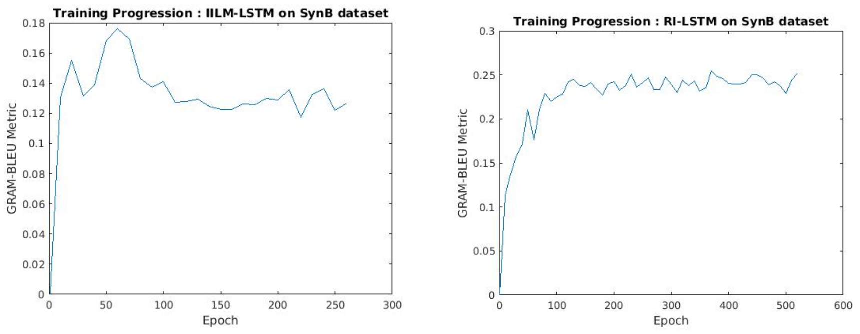

Fig. 12. GRAM-BLEU metric on SynB dataset

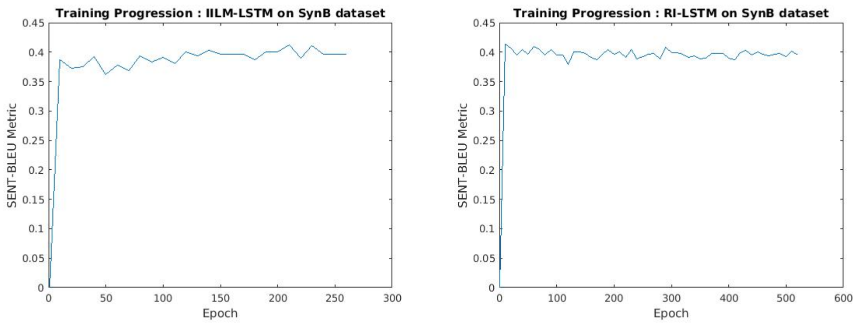

Fig. 13. SENT-BLEU metric on SynB dataset

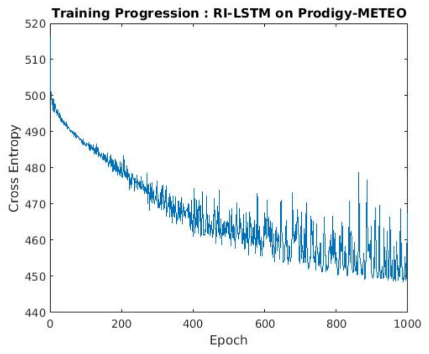

Fig. 14. Training Curves on Prodigy-METEO Dataset

```
[0.812 0.177 0.5818 0.5051 0.]
today is going to be a very dry and hot day with strong winds and
loud thunderstorms
```

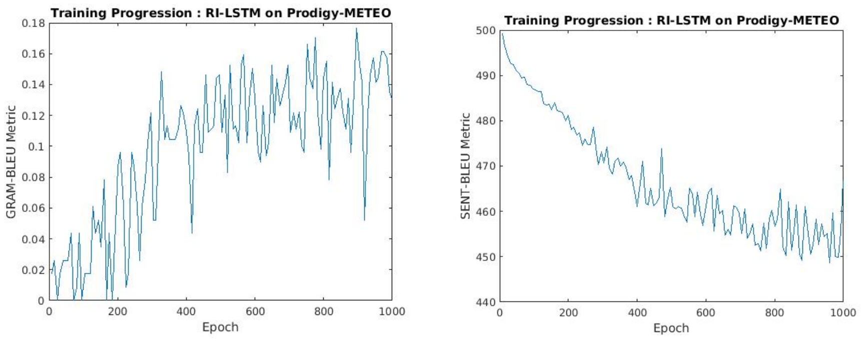

Fig. 15. GRAM and SENT BLEU metrics on Prodigy-METEO Dataset

```
it is going to be a very humid and hot day with strong winds and
loud thunderstorms
```

```
[0.812 0.177 0.5818 0.5051 0.]
ACTUAL: today is going to be a very dry and hot day with strong
winds and loud thunderstorms
GENERATED: it is going to be a very humid and hot day with strong
winds and loud thunderstorms
```

The NLG system got the *humid* part wrong but eventually it will learn that too.

```
[0.812 0.177 0.5818 0.5051 0.]
ACTUAL: it is going to be a very humid, cool day with strong winds
, thunderstorms and hailstones
GENERATED: today is going to be a very humid, cool day with strong
 winds, thunderstorms and hailstones
```

On the human authored corpus Prodigy-METEO, an example of our ROM-LSTM somewhere halfway through it's training.

```
[[1,_NNW,04,08,-,-,0600],[2,_NNE,02,06,-,-,1800],[3,_VAR
,-,-,-,-,0000]]
ACTUAL: NNW 4 - 08 VEERING NNE 2 - 06 BY LATE AFTERNOON THEN
FALLING VARIABLE LATE EVENING
GENERATED: NNW 4 - 4 GRADUALLY BACKING NNE 2 - 06 BY LATE
INCREASING GRADUALLY EVENING
```

It has learn some of the inital parts correctly, but it still has problems with the later parts.

```
[[1,_ESE,14,18,-,-,0600],[2,_E,26,30,40,-,0000]]
ACTUAL: ESE 14-18 GRADUALLY INCREASING E'LY 26-30 GUSTS 40 BY
MIDNIGHT
GENERATED: ESE 14-18 GRADUALLY INCREASING 26-30 GUSTS 40 BY LATE
EVENING
```

By now, it has learnt most of the sturcture. It has even generalised *LATE EVENING* to *MIDNIGHT*

7.4 State Visualization

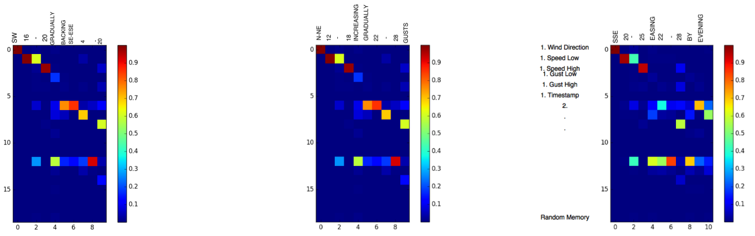

Fig. 16. Attention Weights on examples from Prodigy-METEO Dataset

Memory Attention Heat Map From the Figure 16, we can see how the network weights increase when certain words are to be mentioned. For example, when wind direction is about to be mentioned by the system, the weight corresponding to wind direction glows up. Similarly, when speed low is mentioned speed low lights up, when speed high is mentioned speed high lights up. This basically shows that the network is able to learn the right semantics from the words and the features.

References

1. Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.
2. Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.
3. Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.
4. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
5. Tan Lee, PC Ching, Lai-Wan Chan, et al. An rnn based speech recognition system with discriminative training. 1995.
6. Edward Loper and Steven Bird. NLTK: the natural language toolkit. *CoRR*, cs.CL/0205028, 2002.
7. Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, Zhiheng Huang, and Alan Yuille. Deep captioning with multimodal recurrent neural networks (m-rnn). *arXiv preprint arXiv:1412.6632*, 2014.

8. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
9. Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
10. Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.
11. Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.
12. Ehud Reiter and Robert Dale. Building applied natural language generation systems. *Natural Language Engineering*, 3(01):57–87, 1997.
13. Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, and Phil Blunsom. Reasoning about entailment with neural attention. *arXiv preprint arXiv:1509.06664*, 2015.
14. David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.
15. Somayajulu G Sripada, Ehud Reiter, Jim Hunter, and Jin Yu. Sumtime: Observations from ka for weather domain. Technical report, Citeseer, 2001.
16. Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Interspeech*, pages 194–197, 2012.
17. Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.
18. Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2015.
19. Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv preprint arXiv:1508.01745*, 2015.