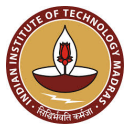
Input Conditional Language Models

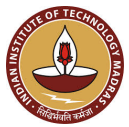
Akshay Balaji, JS Suhas, Sai Praveen and Rakesh R Menon

Indian Institute of Technology Madras

November 27, 2016



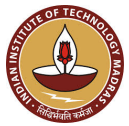
Outline



Introduction

Natural Language Generation(NLG) is the task of generating sentences in response to some input being provided in the form of machine representation. The traditional NLG system consists of three tasks:

- **Content Determination :** This segment deals with what content must be put in the output sentence(s) of the NLG system.
- **Sentence Planning :** The ordering of content and the sentence(s) is planned in this segment.
- **Surface Realization :** This segment deals with choice of words and fitting a grammar to the sentences. This is usually done using template based-methods, wherein after keywords are obtained they are compared with a fixed template stored in a tree.



Introduction

- Existing methods in NLG have tried to find efficient methods to perform each segment of the complete NLG system.
- Machine Learning tries to model the different data that we have and are also capable of generalizations. Hence, it helps to use Machine Learning for each of these tasks. Some of the related work try to use ML in various segments,
  - Content Determination - ML + Sentence Planning/Surface Realization - Template
  - Sentence Planning - ML + Content Determination/Surface Realization - Template
  - Surface Realization - ML + Content Determination/Sentence Planning - Template
- However, all of these systems rely on the intermediate representations that are learnt by each of the segments. In this work, we try to come up with some systems that unify all the segments into one and train the systems in an end-to-end fashion.

Problem Definition

- NLG systems use input data, X to gain insight into intermediate representations, Y to produce sentences W

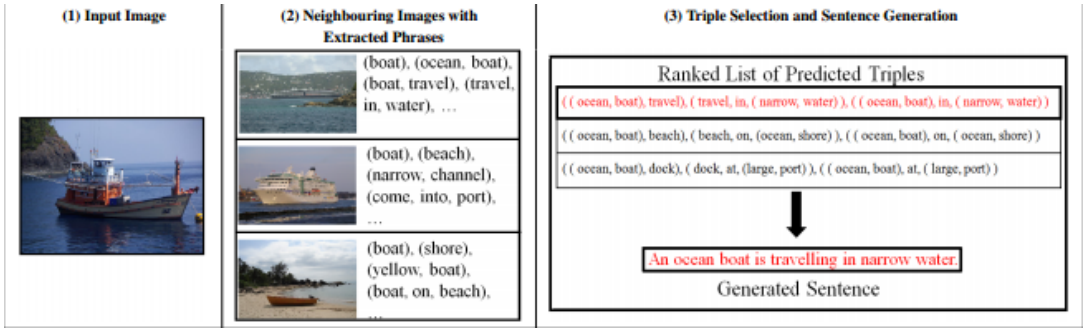
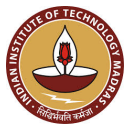


Figure 1: From Images to Sentences

- Most systems have a model to realize Y (which is user generated) and another model to produce sentences W from the derived Y
- Lack of a single end-to-end system that takes in input data, figures out the representation and outputs sentences.

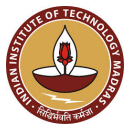
ML Models for NLG

- Could use Machine Learning algorithms for
  - Content Determination
  - Sentence Planning
  - Surface Realization
- Learn intermediate representations, Y, from input data, X
- Option #1 : Expectation Maximization
  - Assume a generative model for the language
  - Assume distributions for model
  - Learn parameters using EM
  - Pros: Can model complex associations between input and output using *hidden* states
  - Cons: Domain specific modeling



ML Models for NLG

- Option #2 : Decision Trees and LSTMs
  - Learn $f : X \rightarrow Y$ using decision trees/random forests
  - Learn $g : Y \rightarrow W$ using SC-LSTMs
- f and g trained independently
- f is supervised learning $\implies$ Y targets must be generated by Humans
- f is domain specific
- Replace f with a Neural Network $\implies$ f can be trained using ∇g
- For image inputs, use a CNN; For sequence inputs, use an RNN/LSTM



Input Initialized LSTM

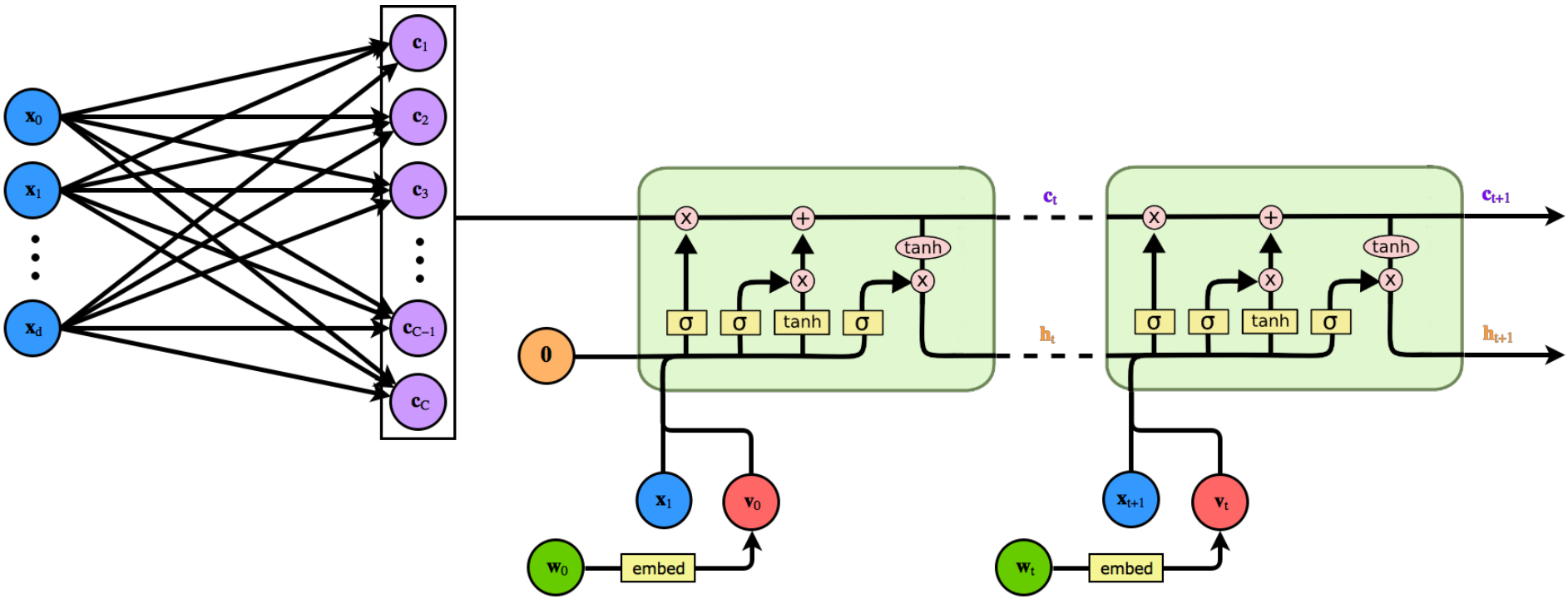


Figure 2: II-LSTM Cell Architecture

Input Initialized LSTM

We tried it on:

```
T > 0.5 This is a hot day.
T < 0.5: This is a cold day.
```

Outputs were randomly sampled from:

```
This is a hot day.
This is a cold day.
```

It learnt the *grammar* but not the *input dependency*.
This is because the LSTM first learns the grammar and then the input dependency.
We hypothesize that, the LSTM uses it's forget gates heavily(frequently saturated at 0 or 1). Since the gradient through the cell state is multiplied through the forget gate activations, the gradient vanishes before reaching the inputs.

Solution to this problem

Solution - Make the inputs available at *each* stage instead of just the beginning.
This will mean the internal Cell State only works on the overall grammar while the inputs X are accessed by the LSTM locally.

Running Input LSTM

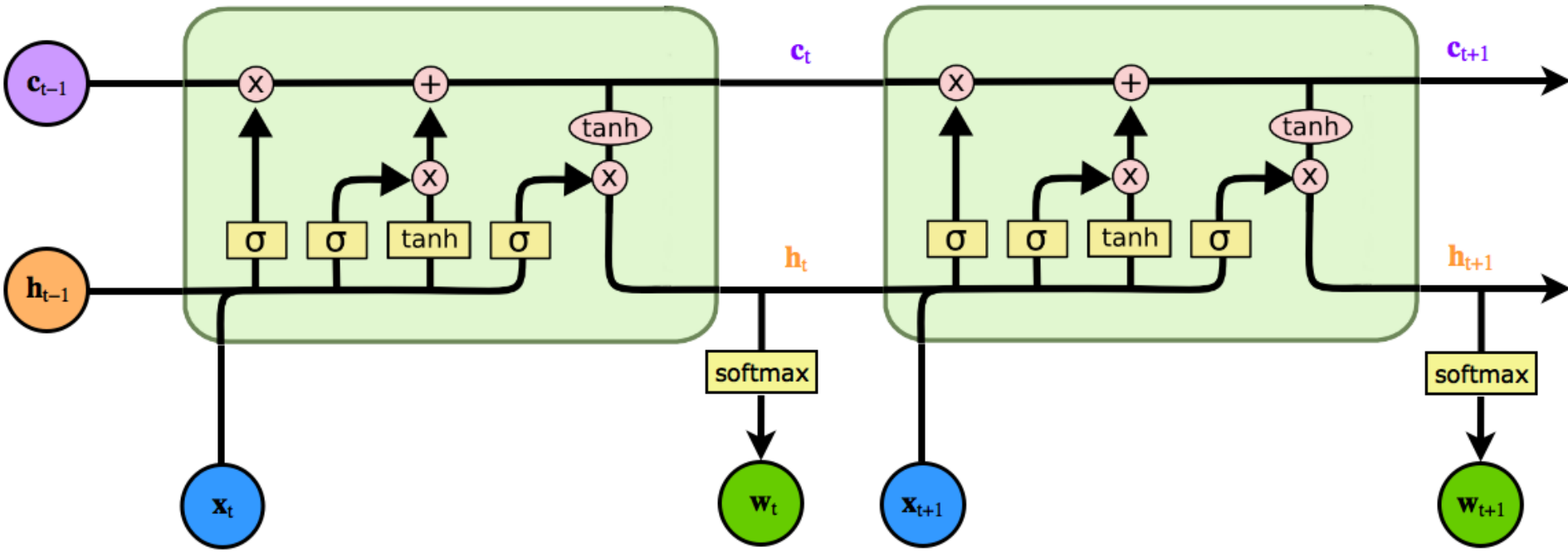


Figure 3: II-LSTM Cell Architecture

Running Input LSTM

We tried it on the SynA dataset. Outputs looked like the following(simplified for brevity):

```
Input: Rain=MEDIUM, day=TODAY
Reference 1: today there may be traces of rain
Reference 2: today there may be light rain
Generated: today there may be light rain rain
```

It has learnt most of the *grammar* and has also the *input dependency* The problem is that at each stage the probabilities of the softmax layer are the following:

today 100%	there 100%	may 100%	be 100%	light 50%	rain 50%	. 50%	'NULL' 50%
				traces 50%	of 50%	rain 50%	. 50%
today	there	may	be	light	rain	rain	.

Figure 4: Probabilities at each stage of the LSTM

Language Model LSTM

- For a timestep t, LSTM output, $\vec{w}_t$, is a probability distribution over V
- Sampling w_t from $\vec{w}_t$ to get a word
- For timestep $t+1$, $\vec{w}_{t+1}$, assumes any of $w_t \in V$ is chosen according to $\vec{w}_t$
- This leads to ambiguity in LSTM outputs over time
- Can resolve this by feeding selected word, w_t as input, $\vec{x}_{t+1}$, to the LSTM
- Can now generate coherent sentences with ambiguity resolution
- Call this Language Model LSTM (LM-LSTM)

LM-LSTM Variants

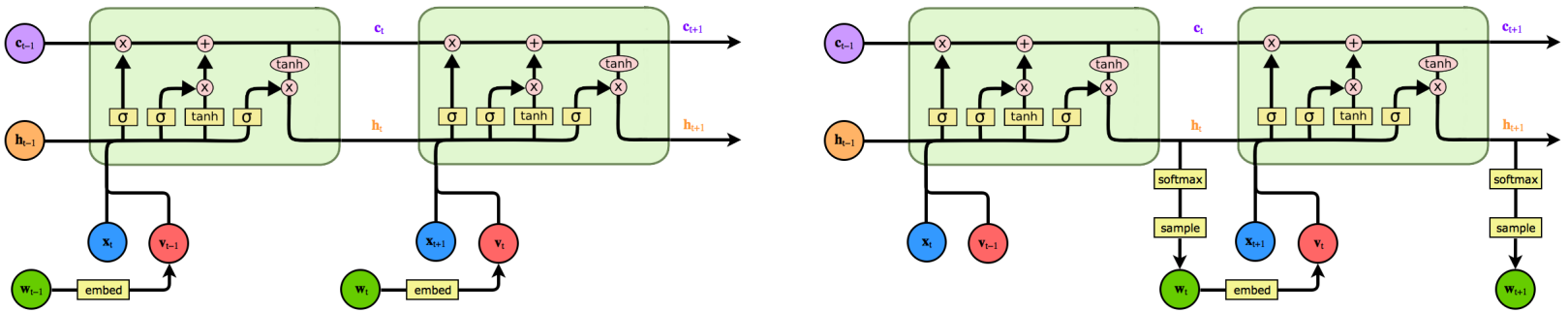


Figure 5: LM-LSTM Train and Test Architectures

- Input Initialized LM-LSTM
  - $\vec{c}_0 = \mathbf{W}_c \cdot \vec{x} + \vec{b}_c$
  - Almost zero learning
- Running Input LM-LSTM
  - Use $\vec{x}_t = \vec{x}$ for all t
  - Gradient issue mitigated

SynA

- Simple 3-dimensional data with basic structured sentences

```
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
```

Figure 6: Input Data

```
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.
```

Figure 7: Target Sentences

SynB

- Complex 5-dimensional data with variance in structure of sentences and semantic alignment

```
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
```

Figure 8: Input Data

```
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.
```

Figure 9: Target Sentences

PRODIGY-METEO

After some searching, the best human-authored dataset we could find was PRODIGY-METEO.
Short Marine Wind Descriptions[roughly 10-20 words in a sentence]
It includes a small series of entries as input(some data is unavailable). A sample:

```
[[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

DATA: 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
TEXT: E-ENE 12 - 16 BACKING NE-NNE 18 - 22 GUSTS 30
```

Prodigy-METEO Analysis

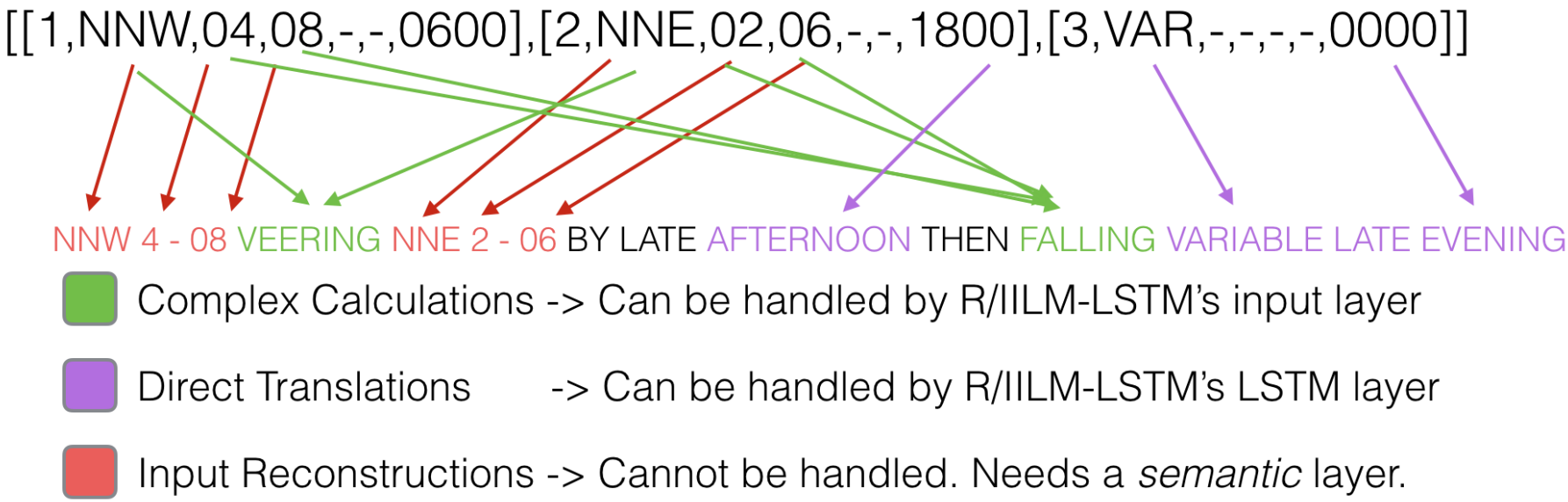


Figure 10: Prodigy-METEO Analysis

Solution to Prodigy METEO

Problem: The Prodigy METEO dataset includes inputs that are recreated in the text. RILM-LSTM and IILM-LSTM cannot directly handle this as they perceive the inputs as integers. They will have to map every input to every other input

Solution: We introduce a *semantic* layer, where we create a matrix of word tokens for each of the *input* fields and allow the LSTM to optionally access a *word* from this matrix instead of generating a word.

ROM RILM-LSTM

Our Architecture for this:

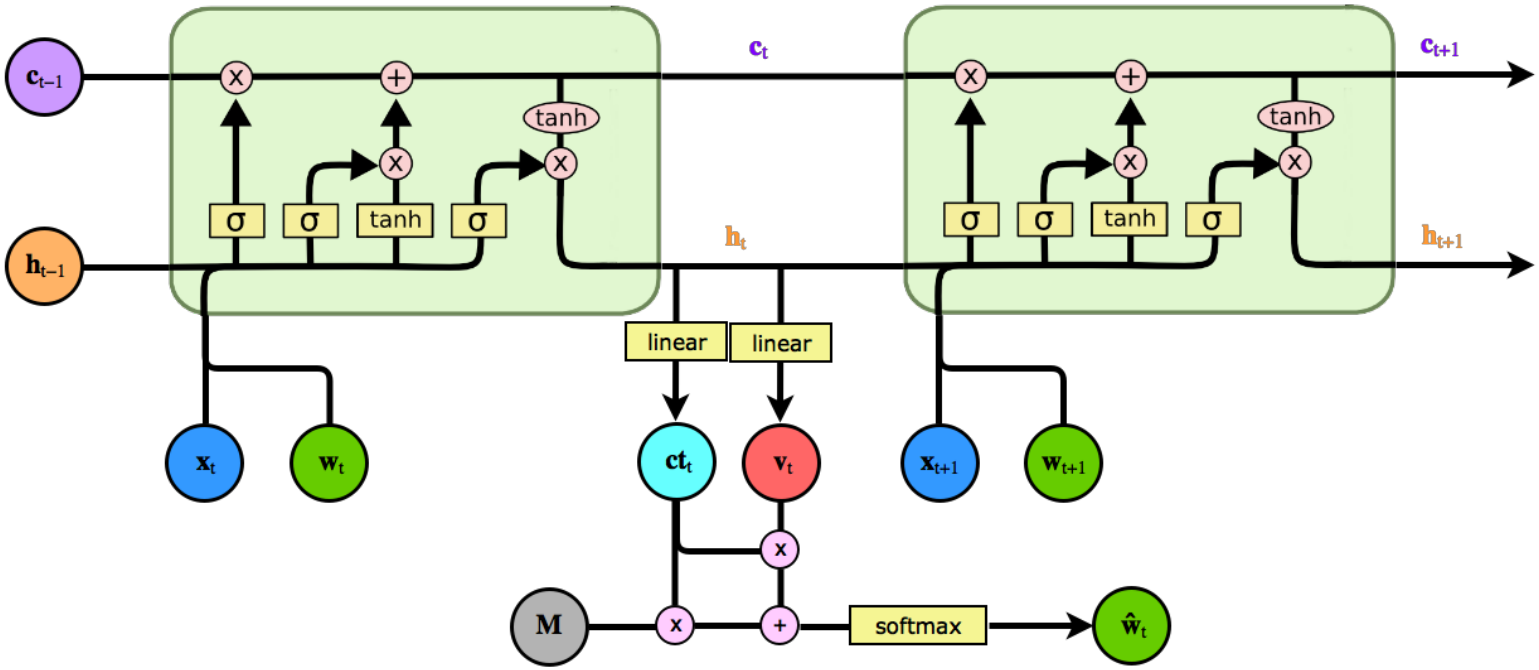


Figure 11: ROM-RILM-LSTM train mode

ROM RILM-LSTM

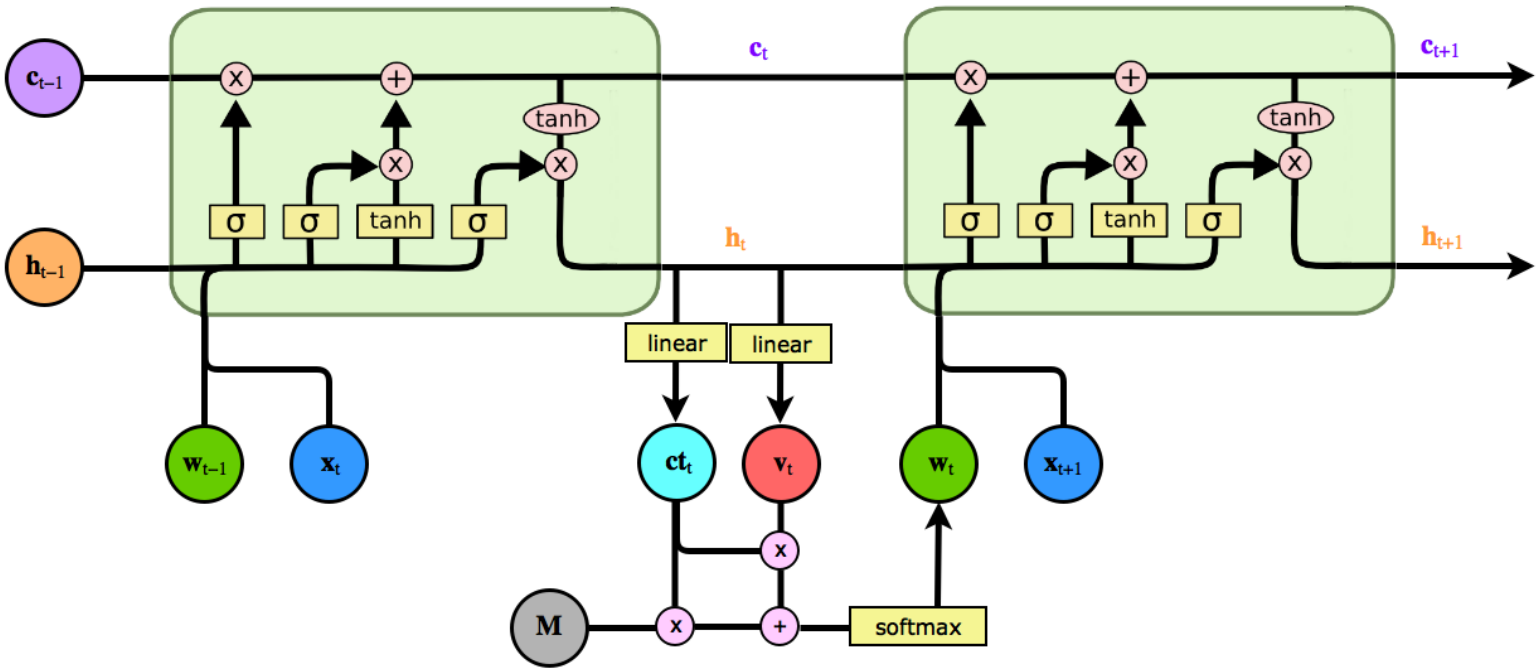


Figure 12: ROM-RILM-LSTM test mode

ROM RILM-LSTM Explanation

The vector r_t is an additional output that the LSTM produces along with it's usual output w_t. The r_t vector is used to optionally choose from 19 options: 18 input semantics + 1 standard output.
In normal operation, the standard output is chosen. When attempting to reconstruct inputs verbatim, the LSTM chooses the semantic instead of trying to manually reproduce it.

Proof of Hypothesis

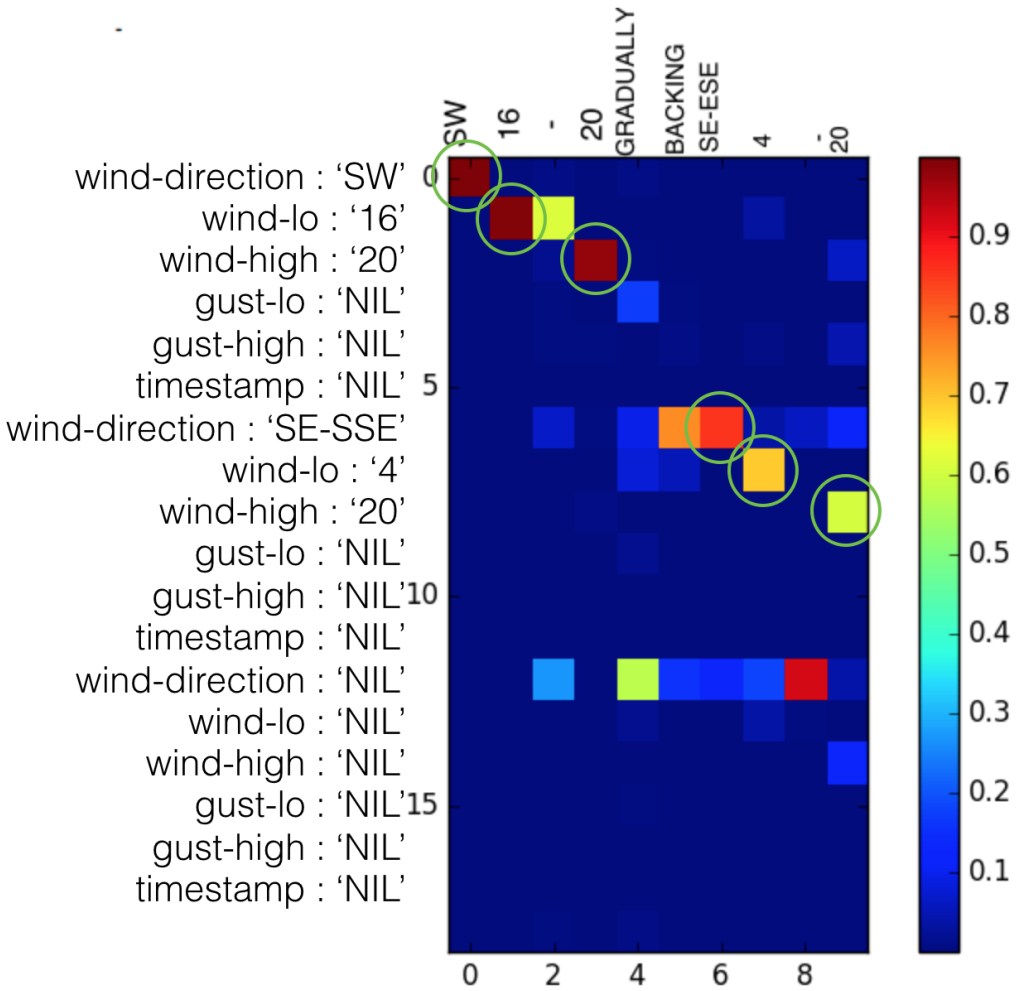


Figure 13: Attention to Inputs at each stage

Word2Vec RILM-LSTM

- One-hot representation not scalable to large vocabularies
- Word2Vec embeds words into low dimensional floating-point vectors
- Output vectors are no longer probabilities, but Word2Vec vectors
- Augment Word2Vec vectors with one-hot fields for ",", "." and "\n".
- Use L_2 loss function to train model
- Need new sampling function to sample from the augmented Word2Vec vector

Metrics

For evaluating the proposed NLG systems, we adapted some forms of the BLEU metric. Here we use the BLEU metric to check the translations and the grammar of the generated sentences in two ways:

- **Translation Metric :** For the translation metric, we try to check the uni-gram translation score of the generated text. This metric is basically used to check if the sentences developed by the system are capable of saying all the relevant words when compared to a reference corpus. An example of how it works is as shown below,

  ```
  Candidate: Today is a very dry day and hot day.
  Reference 1: Today is a very cold and humid day.
  Reference 2: Today is a very dry and cold day.
  ```

  The BLEU score for the above candidate is 0.869.

Metrics

- **Grammar Metric :** With this metric we try to evaluate the grammar learnt by our NLG systems against some human reference corpus'(need not be weather data). For the sake of the project, we have used the camera reviews corpus from NLTK. The tri-grams from the sentence generated by the system is compared with those generated from the reference corpus.

Results

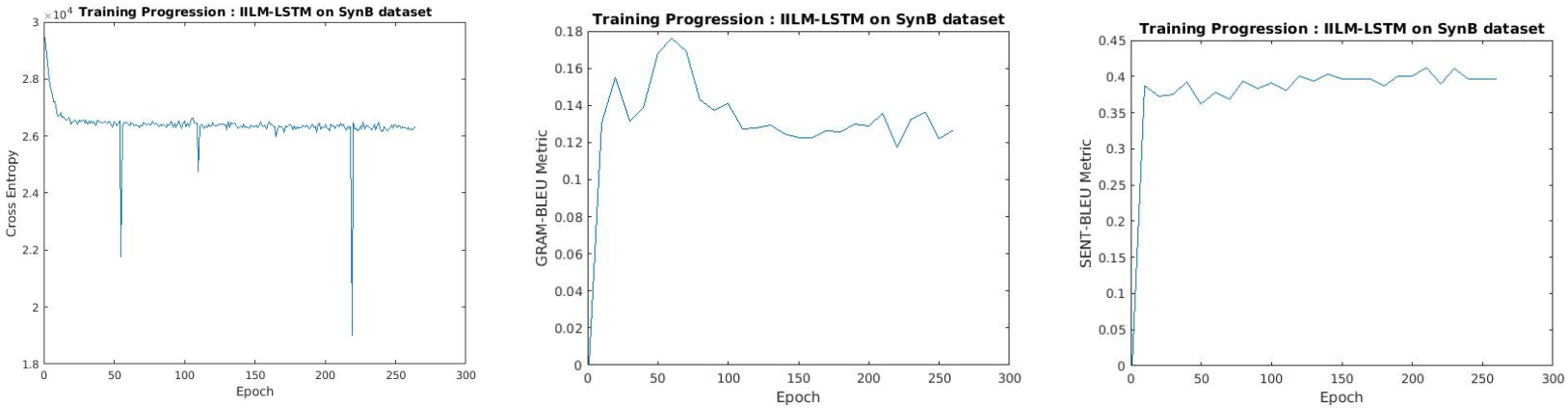


Figure 14: II-LSTM Results (Cross-entropy, Grammar Bleu, Sentence Bleu)

Results

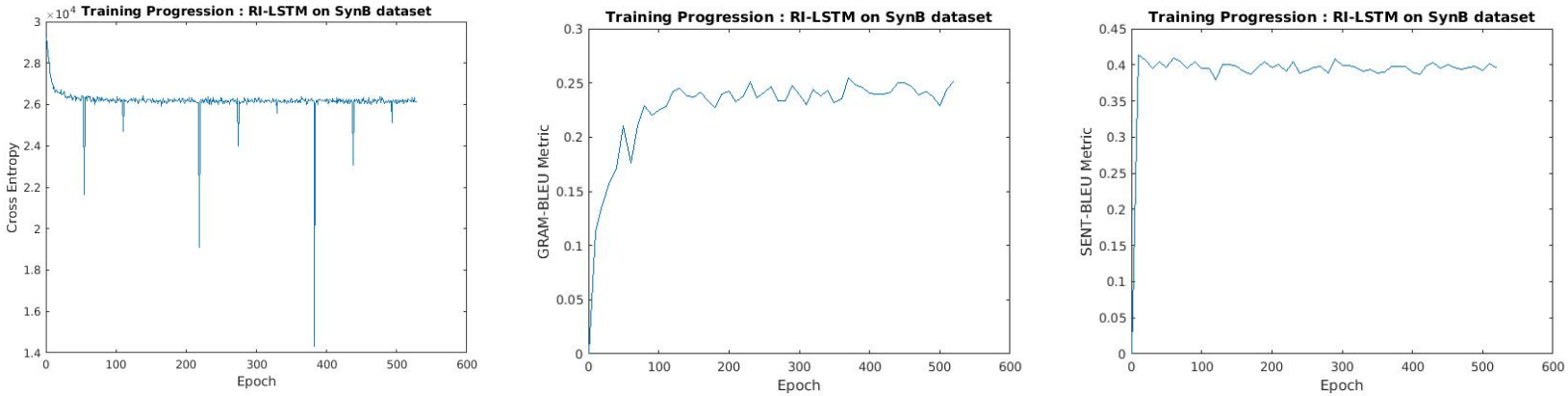
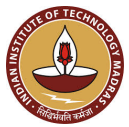


Figure 15: RILM-LSTM Results (Cross-entropy, Grammar Bleu, Sentence Bleu)

Short-term

- Modify the ROM LSTM's memory attention term to use a softmax. This will produce much better attention weights.
- These can potentially be used to directly *label* the input texts with their appropriate input semantics.



Long-term

- Scalability was a very important factor for the architecture decisions we made.
- Our model can be directly expanded to handle complex inputs like images, training directly from images to full text without needing intermediate labels.
- For instance, our ROM-LSTM, when used with a CNN instead of perceptron as input manipulator, can be trained to produce weather texts directly from weather map image inputs.