Introduction to Neural Machine Translation

Andriy Mulyar

Department of Computer Science
Virginia Commonwealth University
Richmond, VA
USA

aymulyar@vcu.edu

November 6 2019

Outline

Goal: Understand how Google Translate worked circa 2015.

1. Introduction
2. Preliminaries
3. Encoder-Decoder Recurrent Neural Networks
4. $f(\text{Final Remarks}) = \text{Remarques finales}$

My Background

- CS fourth year, Math third year
- Interests in language processing and machine learning.
- Four years exploring these areas under some great mentorship and guidance.

My Background

- CS fourth year, Math third year
- Interests in language processing and machine learning.
- Four years exploring these areas under some great mentorship and guidance.
  - Dr. Bridget McInnes - VCU NLP Lab
  - Dr. Bartosz Krawczyk - VCU ML and Datastream Mining Lab

Outline

What are we talking about?

What this talk is:

- A mid-level but technical introduction to the machinery powering modern language translation systems.

What are we talking about?

What this talk is:

- A mid-level but technical introduction to the machinery powering modern language translation systems.
- An excursion into **sequence to sequence** deep learning with recurrent neural networks towards an interesting **natural language processing** problem.

What are we talking about?

What this talk is:

- A mid-level but technical introduction to the machinery powering modern language translation systems.
- An excursion into **sequence to sequence** deep learning with recurrent neural networks towards an interesting **natural language processing** problem.
- An introduction individuals of any background can leave having learned something from.

What are we talking about?

What this talk is:

- A mid-level but technical introduction to the machinery powering modern language translation systems.
- An excursion into **sequence to sequence** deep learning with recurrent neural networks towards an interesting **natural language processing** problem.
- An introduction individuals of any background can leave having learned something from.

What this talk is not:

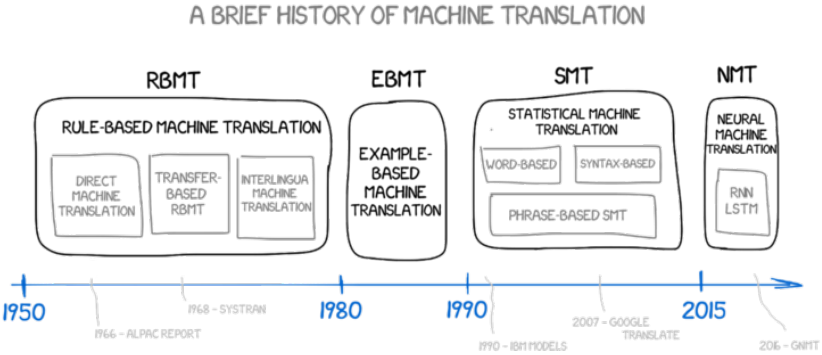
- Exhaustive.
- A demonstration of state-of-the-art techniques (2014).
- Consistent with transposition of matrices (readability).

Talk Pro Tips

- Stop me for questions. If something is unclear to you, then it is surely unclear to someone else.
- It's alright to get more food in the middle of the talk.

Problem

Given a text in language L_1 output a text in language L_2 that humans concede captures the same semantic meaning, obeys language grammar rules and is useful.



Outline

Definitions and Notation: Representing Words

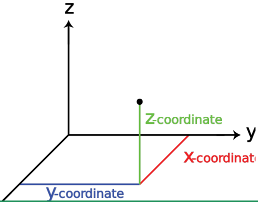
the girl walks the dog.

- **Token**: an element of a predefined **vocabulary** of size d.
  - Ex. the $\in \{x : x \in \text{English lexicon}\}$, $d =$ number of words in English lexicon.

Definitions and Notation: Representing Words

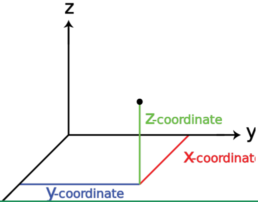
the girl walks the dog.

- **Token**: an element of a predefined **vocabulary** of size d.
  - Ex. the $\in \{x : x \in \text{English lexicon}\}$, $d =$ number of words in English lexicon.
- **Token embedding**: a discriminating representation of a token w.r.t. **vocabulary**.
  - Ex. **One hot**: $\vec{the} = (1,0,0,0,0) \in \mathbb{R}^5 \quad (d = 5)$



the girl walks the dog.

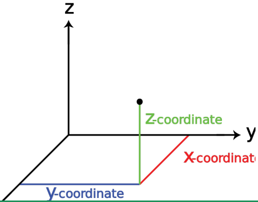
- **Token**: an element of a predefined **vocabulary** of size d.
  - Ex. the $\in \{x : x \in \text{English lexicon}\}$, $d =$ number of words in English lexicon.
- **Token embedding**: a discriminating representation of a token w.r.t. **vocabulary**.
  - Ex. **One hot**: $\vec{the} = (1,0,0,0,0) \in \mathbb{R}^5 \quad (d = 5)$
  - Observations:
    - Vocabularies are large $d \gg 0$. One hot has large memory footprint.
    - One hot assumes no relationship between tokens (ie. they form a basis in $\mathbb{R}^d$).



Definitions and Notation: Representing Words

the girl walks the dog.

- **Token**: an element of a predefined **vocabulary** of size d.
  - Ex. the $\in \{x : x \in \text{English lexicon}\}$, $d =$ number of words in English lexicon.
- **Token embedding**: a discriminating representation of a token w.r.t. **vocabulary**.
  - Ex. **One hot**: $\vec{the} = (1,0,0,0,0) \in \mathbb{R}^5 \quad (d = 5)$
  - Observations:
    - Vocabularies are large $d \gg 0$. One hot has large memory footprint.
    - One hot assumes no relationship between tokens (ie. they form a basis in $\mathbb{R}^d$).
  - A hot research area in NLP:
    - How can we incorporate context when representing a token?



Definitions and Notation: Text

the girl walks the dog.

- **Token sequence**:
  [the, girl, walks, the, dog, .] $= [x_1, x_2, x_3, x_4, x_5, x_6]$
  - Observation: Text is just a sequence of tokens!

Definitions and Notation: Text

the girl walks the dog.

- **Token sequence**:
  [the, girl, walks, the, dog, .] $= [x_1, x_2, x_3, x_4, x_5, x_6]$
  - Observation: Text is just a sequence of tokens!
- **Embedding sequence**:
  $[\ \vec{\text{the}}, \vec{\text{girl}}, \vec{\text{walks}}, \vec{\text{the}}, \vec{\text{dog}}, \vec{.}\ ] = [\vec{x}_1, \vec{x}_2, \vec{x}_3, \vec{x}_4, \vec{x}_5, \vec{x}_6]$

Definitions and Notation: Text

the girl walks the dog.

- **Token sequence**:

  [the, girl, walks, the, dog, .] $= [x_1, x_2, x_3, x_4, x_5, x_6]$

  - Observation: Text is just a sequence of tokens!

- **Embedding sequence**:

  $[\ \vec{\text{the}}, \vec{\text{girl}}, \vec{\text{walks}}, \vec{\text{the}}, \vec{\text{dog}}, \vec{.}\ ] = [\vec{x}_1, \vec{x}_2, \vec{x}_3, \vec{x}_4, \vec{x}_5, \vec{x}_6]$

  - Observation: A text document comprising n tokens can be represented as a "matrix" $\mathbb{R}^{n \times d}$ where d is vocab size.

the girl walks the dog.

- **Token sequence**:
  [the, girl, walks, the, dog, .] $= [x_1, x_2, x_3, x_4, x_5, x_6]$
  - Observation: Text is just a sequence of tokens!
- **Embedding sequence**:
  $[\ \vec{\text{the}}, \vec{\text{girl}}, \vec{\text{walks}}, \vec{\text{the}}, \vec{\text{dog}}, \vec{.}\ ] = [\vec{x}_1, \vec{x}_2, \vec{x}_3, \vec{x}_4, \vec{x}_5, \vec{x}_6]$
  - Observation: A text document comprising n tokens can be represented as a "matrix" $\mathbb{R}^{n \times d}$ where d is vocab size.
  - Ex. Our sequence of 6 tokens with a one-hot encoding (assume $d = 5$) yields a matrix $\mathbb{R}^{6 \times 5}$

$$[\vec{\text{the}}, \vec{\text{girl}}, \vec{\text{walks}}, \vec{\text{the}}, \vec{\text{dog}}, \vec{.}] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Definitions and Notation: Translation

the girl walks the dog. $\rightarrow$ la fille promène le chien.

We now have the tools to be precise:

- **Translation**: Given languages L_1, L_2 with lexicons V_1, V_2 a translation is a function $f : \mathbb{R}^{T \times d} \rightarrow \mathbb{R}^{T' \times d}$ mapping a length T sequence of tokens over V_1 to a length T' sequence of tokens over V_2.

Definitions and Notation: Translation

the girl walks the dog. $\rightarrow$ la fille promène le chien.

We now have the tools to be precise:

- **Translation**: Given languages L_1, L_2 with lexicons V_1, V_2 a translation is a function $f : \mathbb{R}^{T \times d} \rightarrow \mathbb{R}^{T' \times d}$ mapping a length T sequence of tokens over V_1 to a length T' sequence of tokens over V_2.

Machine Translation is the task of estimating a model from a set of sample sequences $(x)_T$ over V_1 to sequences $(y)_{T'}$ over V_2 (example translations) that parameterizes all such translation functions f where T, T' vary.

Definitions and Notation: Translation

the girl walks the dog. $\rightarrow$ la fille promène le chien.

We now have the tools to be precise:

- **Translation**: Given languages L_1, L_2 with lexicons V_1, V_2 a translation is a function $f : \mathbb{R}^{T \times d} \rightarrow \mathbb{R}^{T' \times d}$ mapping a length T sequence of tokens over V_1 to a length T' sequence of tokens over V_2.

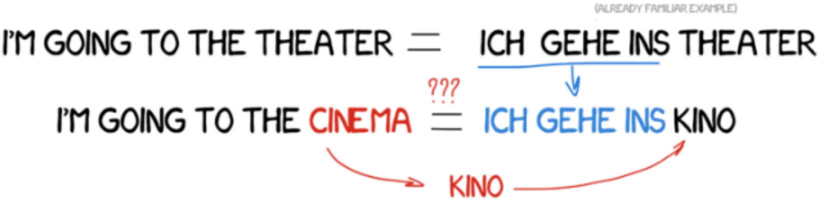
Machine Translation is the task of estimating a model from a set of sample sequences $(x)_T$ over V_1 to sequences $(y)_{T'}$ over V_2 (example translations) that parameterizes all such translation functions f where T, T' vary.

$$\{(x)^i, (y)^i : i \in \{1, ..., n\}\}$$

Almost there!

How do we estimate such a model? What properties should it possess?

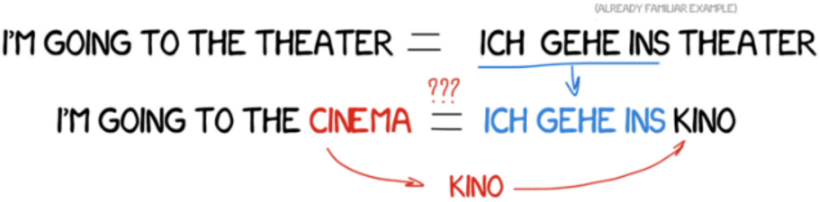
- Independent of L_1, L_2.



Almost there!

How do we estimate such a model? What properties should it possess?

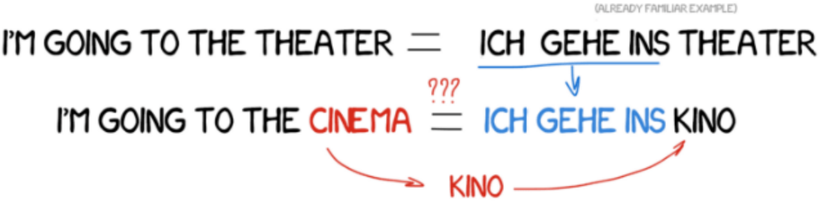
- Independent of L_1, L_2.
- Capable of handling translations that require the generation of both short and long sequences.



Almost there!

How do we estimate such a model? What properties should it possess?

- Independent of L_1, L_2.
- Capable of handling translations that require the generation of both short and long sequences.
- Be able to translate input sequences **un-seen** during creation (training). This means **generalize**!



Outline

Framework

Idea: Consider translation as estimating a conditional distributional!

- Assume the output of a translation $Y = (y)_{T'}$ is a random variable conditioned on the input $X = (x)_T$:

$$p(Y|X) = p(y_1, y_2, ..., y_{T'}|x_1, x_2, ..., x_T)$$

Framework

Idea: Consider translation as estimating a conditional distributional!

- Assume the output of a translation $Y = (y)_{T'}$ is a random variable conditioned on the input $X = (x)_T$:

$$p(Y|X) = p(y_1, y_2, ..., y_{T'}|x_1, x_2, ..., x_T)$$

- **Problem**: The first conditional is mind boggling to estimate (Y is product of random variables each with sample space V_2).

Framework

Idea: Consider translation as estimating a conditional distributional!

- Assume the output of a translation $Y = (y)_{T'}$ is a random variable conditioned on the input $X = (x)_T$:

$$p(Y|X) = p(y_1, y_2, ..., y_{T'}|x_1, x_2, ..., x_T)$$

- **Problem**: The first conditional is mind boggling to estimate (Y is product of random variables each with sample space V_2).
  Solution: Factorize into a product of auto-regressive terms:

$$= \prod_{t=1}^{T'} p(y_t|x_1, x_2, ..., x_T; y_1, ..., y_{t-1})$$

Framework

Idea: Consider translation as estimating a conditional distributional!

- Assume the output of a translation $Y = (y)_{T'}$ is a random variable conditioned on the input $X = (x)_T$:

$$p(Y|X) = p(y_1, y_2, ..., y_{T'}|x_1, x_2, ..., x_T)$$

- **Problem**: The first conditional is mind boggling to estimate (Y is product of random variables each with sample space V_2).
  Solution: Factorize into a product of auto-regressive terms:

$$= \prod_{t=1}^{T'} p(y_t|x_1, x_2, ..., x_T; y_1, ..., y_{t-1})$$

- Why conditional estimation? Why can we factorize?

Framework cont.

- For a given language pair how do we estimate the distribution:

$$p(Y|X) = \prod_{t=1}^{T'} p(y_t|x_1, x_2, ..., x_T; y_1, ..., y_{t-1})$$

Framework cont.

- For a given language pair how do we estimate the distribution:

$$p(Y|X) = \prod_{t=1}^{T'} p(y_t | x_1, x_2, ..., x_T; y_1, ..., y_{t-1})$$

- By making a mountain of assumptions.
  1. p can be parameterized by a recurrent neural network.

.

Framework cont.

- For a given language pair how do we estimate the distribution:

$$p(Y|X) = \prod_{t=1}^{T'} p(y_t|x_1, x_2, ..., x_T; y_1, ..., y_{t-1})$$

- By making a mountain of assumptions.
  1. p can be parameterized by a recurrent neural network.
  2. Our sequences are appropriately pre-processed.
     - Special tokens are added to vocabulary indicating end of sentences.
     - Vocabulary is shrunk down (lower casing inputs, etc).

  Ex. the, girl, walked, the, dog, . $\rightarrow$ the, girl, walked, the, dog, ., <EOS>

.

Recurrent Neural Networks (high level)

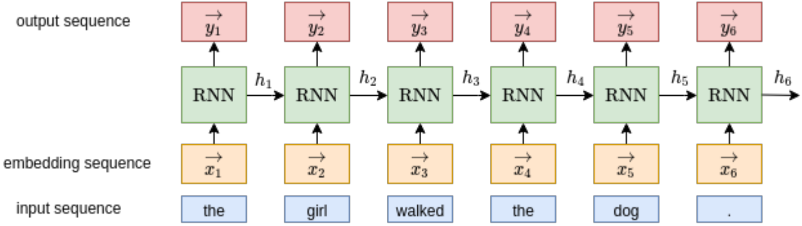
- **Reccurrent Neural Networks**: A general technique for mapping an input sequence to a same length output sequence (each with elements in $\mathbb{R}^d$).

Recurrent Neural Networks (high level)

- **Reccurrent Neural Networks**: A general technique for mapping an input sequence to a same length output sequence (each with elements in $\mathbb{R}^d$).
  - For those with ML background, this is a special type of multi-layered perceptron (neural network). Connections between layers are bottle necked in order to provide hints as to the temporal structure of the input sequence.

Recurrent Neural Networks (high level)

- **Reccurrent Neural Networks**: A general technique for mapping an input sequence to a same length output sequence (each with elements in $\mathbb{R}^d$).
  - For those with ML background, this is a special type of multi-layered perceptron (neural network). Connections between layers are bottle necked in order to provide hints as to the temporal structure of the input sequence.



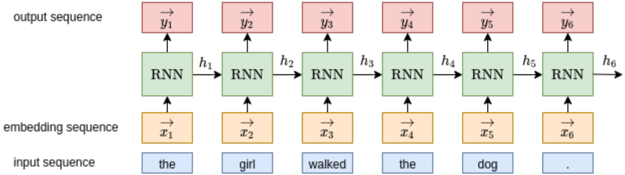
Recurrent Neural Networks (high level)

$$\mathrm{RNN}(x_t, h_{t-1}) = y_t$$

- At each time-step (token embedding!), the affine-linear[1] transformations

$$\vec{h}_t = \tanh\left(W_1^{h\times d}\vec{x}_t + W_2^{h\times h}\vec{h}_{t-1}\right)$$
$$\vec{y}_t = W_3^{y\times h}\vec{h}_t$$

  are applied followed by a differentiable non-linearity.



[1] But wait your equation is wrong! Affine means we need origin shift (bias)! Not included for simplicity.

Recurrent Neural Networks (high level)

$$\boxed{\mathrm{RNN}(x_t, h_{t-1}) = y_t}$$

- At each time-step (token embedding!), the affine-linear[1] transformations

$$\vec{h}_t = \tanh\left(W_1^{h\times d}\vec{x}_t + W_2^{h\times h}\vec{h}_{t-1}\right)$$
$$\vec{y}_t = W_3^{y\times h}\vec{h}_t$$

  are applied followed by a differentiable non-linearity.
- W_1, W_2, W_3 are each just a single layer of perceptrons!

[1] But wait your equation is wrong! Affine means we need origin shift (bias)! Not included for simplicity.

Recurrent Neural Networks (high level)

$$\mathrm{RNN}(x_t, h_{t-1}) = y_t$$

- At each time-step (token embedding!), the affine-linear[1] transformations

$$\vec{h}_t = \tanh\left(W_1^{h\times d}\vec{x}_t + W_2^{h\times h}\vec{h}_{t-1}\right)$$
$$\vec{y}_t = W_3^{y\times h}\vec{h}_t$$

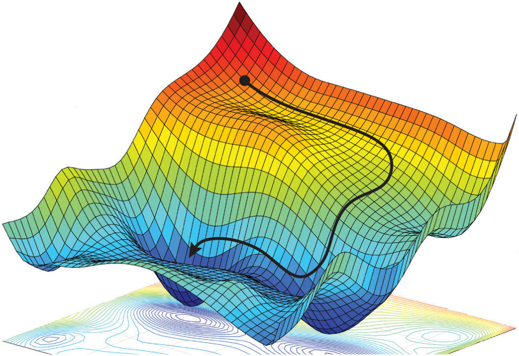
are applied followed by a differentiable non-linearity.

- W_1, W_2, W_3 are each just a single layer of perceptrons!
- W_i can be adjusted (trained!) to satisfy some objective via modified form of gradient descent (back-propagation through time).

[1] But wait your equation is wrong! Affine means we need origin shift (bias)! Not included for simplicity.

Gradient what?

W_i can be adjusted (trained!) to satisfy some objective via modified form of gradient descent.



Sequence to Sequence Learning

- RNN's can directly be applied to temporal problems where output sequence is same length as input.
  - Tagging (NER)
- But in machine translation, input and output sequences have un-bounded length!

Sequence to Sequence Learning

- RNN's can directly be applied to temporal problems where output sequence is same length as input.
  - Tagging (NER)
- But in machine translation, input and output sequences have un-bounded length!
- **Solution:** Utilize two RNN's. One to encode a sequence representation and one to decode a sequence!

Sequence to Sequence Learning

- RNN's can directly be applied to temporal problems where output sequence is same length as input.
  - Tagging (NER)
- But in machine translation, input and output sequences have un-bounded length!
- **Solution:** Utilize two RNN's. One to encode a sequence representation and one to decode a sequence!
  - What is our objective? Find set of W_i for each RNN (parameters θ) such that

$$\max_{\theta} \quad p(Y|X) = \max_{\theta} \quad \prod_{t=1}^{T'} p(y_t | x_1, x_2, ..., x_T; y_1, ..., y_{t-1})$$

where Y is a text in L_2 and X text in L_1.

Sequence to Sequence Learning

- RNN's can directly be applied to temporal problems where output sequence is same length as input.
  - Tagging (NER)
- But in machine translation, input and output sequences have un-bounded length!
- **Solution:** Utilize two RNN's. One to encode a sequence representation and one to decode a sequence!
  - What is our objective? Find set of W_i for each RNN (parameters θ) such that

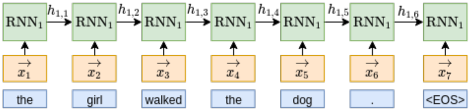
$$\max_{\theta} \quad p(Y|X) = \max_{\theta} \quad \prod_{t=1}^{T'} p(y_t|x_1, x_2, ..., x_T; y_1, ..., y_{t-1})$$

    where Y is a text in L_2 and X text in L_1.
  - Why?
    Given new X we can auto-regressively decode Y by sampling p with tokens from V_2!

Encoder-Decoder RNN's

Idea:

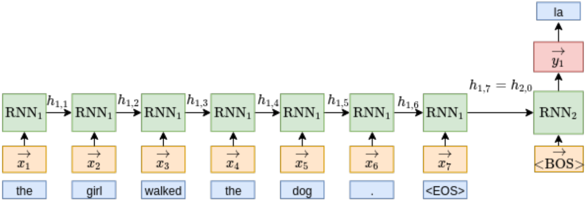
1. Encoder RNN_1 unrolls over input sequence $(x)_T$.



Encoder-Decoder RNN's

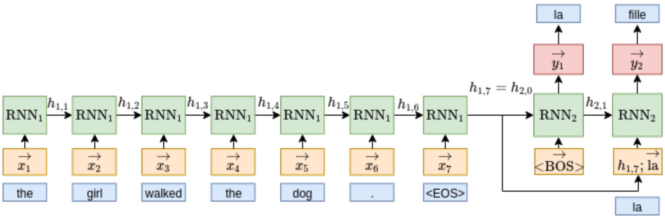
Idea:

1. Encoder RNN_1 unrolls over input sequence $(x)_T$.
2. Final internal state of RNN_1, $\vec{h}_T$, initializes decoder RNN_2.



Idea:

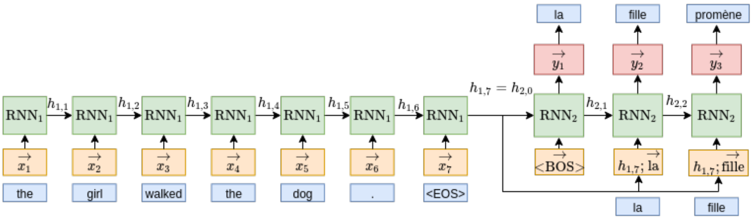
1. Encoder RNN_1 unrolls over input sequence $(x)_T$.
2. Final internal state of RNN_1, $\vec{h}_T$, initializes decoder RNN_2.
3. RNN_2 auto-regressively decodes y_t until reaching $< EOS >$ token.



Encoder-Decoder RNN's

Idea:

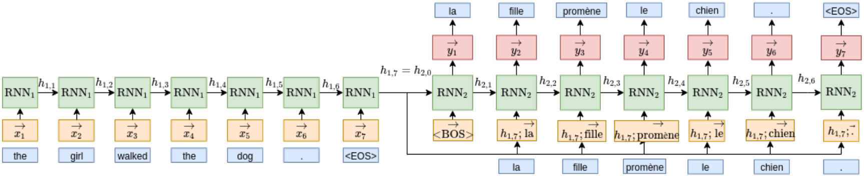
1. Encoder RNN_1 unrolls over input sequence $(x)_T$.
2. Final internal state of RNN_1, $\vec{h}_T$, initializes decoder RNN_2.
3. RNN_2 auto-regressively decodes y_t until reaching $< EOS >$ token.



Encoder-Decoder RNN's

Idea:

1. Encoder RNN_1 unrolls over input sequence $(x)_T$.
2. Final internal state of RNN_1, $\vec{h}_T$, initializes decoder RNN_2.
3. RNN_2 auto-regressively decodes y_t until reaching $< EOS >$ token.

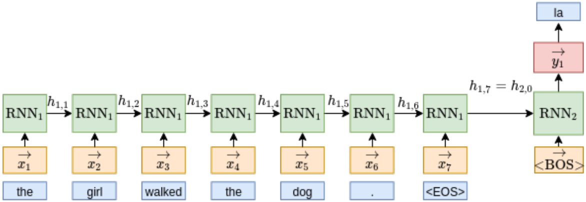


Encoder-Decoder RNN Training

- Encoder-Decoder RNN's parameterize our factorized distribution[1]!

$$p(Y|X) = \prod_{t=1}^{T'} p(y_t|x_1, x_2, ..., x_T; y_1, ..., y_{t-1})$$

$$= p(y_1|X) \cdot ...$$



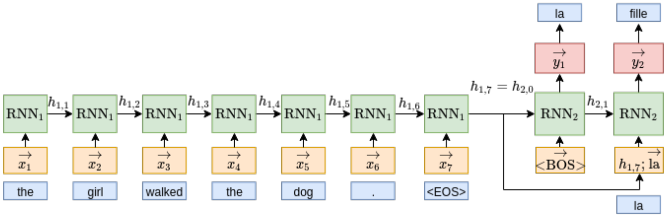
[1] With appropriate output constraints.

Encoder-Decoder RNN Training

- Encoder-Decoder RNN parameterizes our factorized distribution!

$$p(Y|X) = \prod_{t=1}^{T'} p(y_t|x_1, x_2, ..., x_T; y_1, ..., y_{t-1})$$

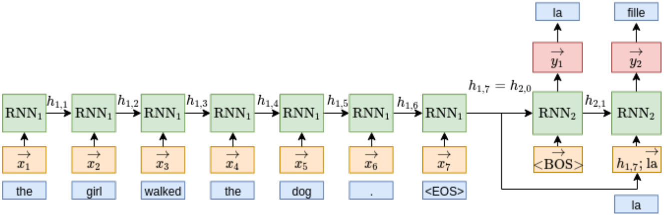
$$= p(y_1|X) \cdot p(y_2|X; y_1) \cdot ...$$



Encoder-Decoder RNN Training/Inference

- During each decoding time step, p (our coupled RNN's) estimates the probability of each token in V_2 conditioned on our previous translated tokens and input sequence.

$$p(Y|X) = \prod_{t=1}^{T'} p(y_t|x_1, x_2, ..., x_T; y_1, ..., y_{t-1})$$

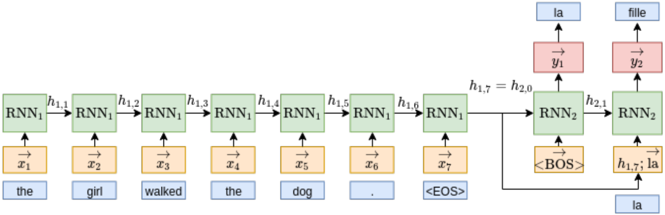


Encoder-Decoder RNN Training/Inference

- During each decoding time step, p (our coupled RNN's) estimates the probability of each token in V_2 conditioned on our previous translated tokens and input sequence.

$$p(Y|X) = \prod_{t=1}^{T'} p(y_t|x_1, x_2, ..., x_T; y_1, ..., y_{t-1})$$

- At each time step, parameters of both RNNs are adjusted to assign higher probability to the correct sub-translation (via modified form of gradient descent).



How well does this work?

- The Seq2Seq learning paradigm was first applied successfully to MT in (Sutskever, 2014).

How well does this work?

- The Seq2Seq learning paradigm was first applied successfully to MT in (Sutskever, 2014).
- First time a purely machine learning based approach was competitive with traditional statistical and phrase-based machine translation.

How well does this work?

- The Seq2Seq learning paradigm was first applied successfully to MT in (Sutskever, 2014).
- First time a purely machine learning based approach was competitive with traditional statistical and phrase-based machine translation.
- Sutskever trained on a massive English-French parallel corpus.
  - 12 million sentence pairs.

How well does this work?

- The Seq2Seq learning paradigm was first applied successfully to MT in (Sutskever, 2014).
- First time a purely machine learning based approach was competitive with traditional statistical and phrase-based machine translation.
- Sutskever trained on a massive English-French parallel corpus.
  - 12 million sentence pairs.
  - Encoder-Decoder contained 348M parameters.

How well does this work?

- The Seq2Seq learning paradigm was first applied successfully to MT in (Sutskever, 2014).
- First time a purely machine learning based approach was competitive with traditional statistical and phrase-based machine translation.
- Sutskever trained on a massive English-French parallel corpus.
  - 12 million sentence pairs.
  - Encoder-Decoder contained 348M parameters.
  - $V_1 = 160k, V_2 = 80k$.
    - Parallelized **parameters** and data across 8 GPU's during training. Still took 10 days.

How well does this work?

- The Seq2Seq learning paradigm was first applied successfully to MT in (Sutskever, 2014).
- First time a purely machine learning based approach was competitive with traditional statistical and phrase-based machine translation.
- Sutskever trained on a massive English-French parallel corpus.
  - 12 million sentence pairs.
  - Encoder-Decoder contained 348M parameters.
  - $V_1 = 160k, V_2 = 80k$.
    - Parallelized **parameters** and data across 8 GPU's during training. Still took 10 days.
  - BLEU score: ~ 34.8 (current SOTA sits at ~ 45)
    - BLEU (Bilingual Evaluation Understudy) is a metric assessing MT performance with high correlation to human judgement.

Outline

So - how does this *actually* work.

- This presentation showcased a vanilla RNN cell.
  - In practice, a RNN cell with more parameters and differing connections, the **LSTM**, is utilized.
    - **LSTM** - Long Short Term Memory RNN

So - how does this *actually* work.

- This presentation showcased a vanilla RNN cell.
    - In practice, a RNN cell with more parameters and differing connections, the **LSTM**, is utilized.
        - **LSTM** - Long Short Term Memory RNN
- Reversing translated sentences during training/inference yields large performance gains.

So - how does this *actually* work.

- This presentation showcased a vanilla RNN cell.
  - In practice, a RNN cell with more parameters and differing connections, the **LSTM**, is utilized.
    - **LSTM** - Long Short Term Memory RNN
- Reversing translated sentences during training/inference yields large performance gains.
- Language vocabularies are **large** - utilizing arbitrary space-delimited sequences of characters tends towards computational infeasability.
  - **WordPiece** tokenization (Google) - a domain specific tokenization based on a trained language model.
  - Boosts training and inference time by turning 120k token vocabularies into 30k tokens.

So - how does this *actually* work.

- This presentation showcased a vanilla RNN cell.
  - In practice, a RNN cell with more parameters and differing connections, the **LSTM**, is utilized.
    - **LSTM** - Long Short Term Memory RNN
- Reversing translated sentences during training/inference yields large performance gains.
- Language vocabularies are **large** - utilizing arbitrary space-delimited sequences of characters tends towards computational infeasability.
  - **WordPiece** tokenization (Google) - a domain specific tokenization based on a trained language model.
  - Boosts training and inference time by turning 120k token vocabularies into 30k tokens.
- And finally ... as of 2017 top performing MT models **do not** use recurrent neural networks!
  - Same encoder-decoder framework holds, but non-recurrent seq2seq based neural network architectures now prevail. Why?
    - Transformer

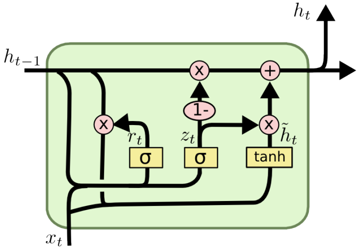
Conclusion

Thank you for your attention!
Questions?

https://bit.ly/2tPVPfv

aymulyar@vcu.edu
www.andriymulyar.com

Supplement: LSTM



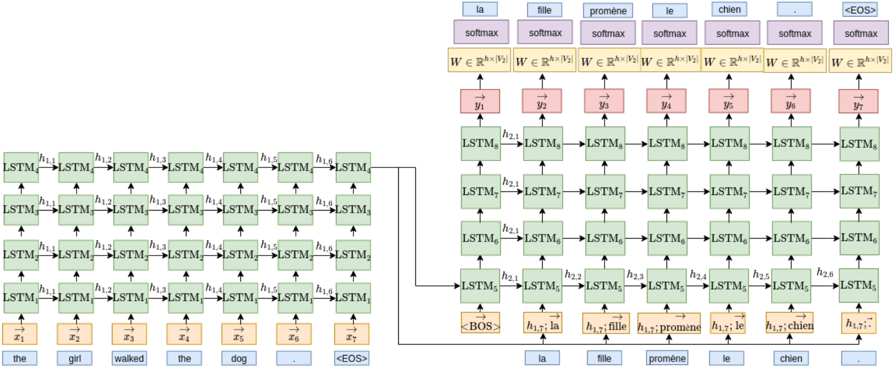
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- I did say 340M weights right?
- Note that this is really during inference - during training we need need to incorporate our objective!



Supplement: Decoding at Inference

- But decoding is difficult too (recalled p is but an estimate)!
- Usually several most likely next translations are explored and pruned in a tree like fashion:
  - **Beam Search**