Recurrent Neural Networks(Socher and Manning)

José Eduardo Ochoa Luna
Dr. Ciencias - Universidade de São Paulo

Maestría C.C. Universidad Católica San Pablo
Sistemas Inteligentes

8 de Noviembre 2018

CNNs for NLP (Kim)

- Instead of image pixels, the input to most NLP tasks are sentences or documents represented as a matrix.

CNNs for NLP (Kim)

- Instead of image pixels, the input to most NLP tasks are sentences or documents represented as a matrix.
- Each row of the matrix corresponds to one token, typically a word, but it could be a character.

CNNs for NLP (Kim)

- Instead of image pixels, the input to most NLP tasks are sentences or documents represented as a matrix.
- Each row of the matrix corresponds to one token, typically a word, but it could be a character.
- Each row is vector that represents a word. Typically, these vectors are word embeddings (low-dimensional representations) like word2vec or GloVe, but they could also be one-hot vectors that index the word into a vocabulary.

CNNs for NLP (Kim)

- Instead of image pixels, the input to most NLP tasks are sentences or documents represented as a matrix.
- Each row of the matrix corresponds to one token, typically a word, but it could be a character.
- Each row is vector that represents a word. Typically, these vectors are word embeddings (low-dimensional representations) like word2vec or GloVe, but they could also be one-hot vectors that index the word into a vocabulary.
- For a 10 word sentence using a 100-dimensional embedding we would have a 10×100 matrix as our input

CNNs for NLP II

- In vision, our filters slide over local patches of an image, but in NLP we typically use filters that slide over full rows of the matrix (words).

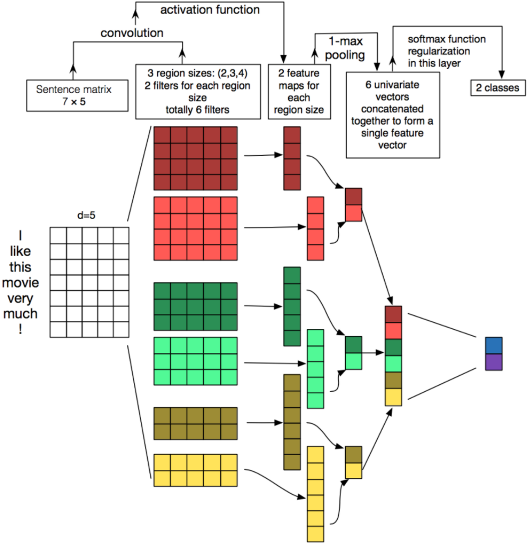
CNNs for NLP II

- In vision, our filters slide over local patches of an image, but in NLP we typically use filters that slide over full rows of the matrix (words).
- the "width" of our filters is usually the same as the width of the input matrix.

CNNs for NLP II

- In vision, our filters slide over local patches of an image, but in NLP we typically use filters that slide over full rows of the matrix (words).
- the "width" of our filters is usually the same as the width of the input matrix.
- The height, or region size, may vary, but sliding windows over 2-5 words at a time is typical.

CNNs for NLP III



Language Models

A language model computes a probability for a sequence of words:

$$P(w_1, \ldots, w_T)$$

Useful for machine translation:

- Word ordering:
  p(the cat is small) $>$ p(small the is cat)

Language Models

A language model computes a probability for a sequence of words:

$$P(w_1, \ldots, w_T)$$

Useful for machine translation:

- Word ordering:
  p(the cat is small) $>$ p(small the is cat)
- Word choice:
  p(walking home after school) $>$ p(walking house after school)

Traditional Language Models

- Probability is usually conditioned on window of n previous words

Traditional Language Models

- Probability is usually conditioned on window of n previous words
- An incorrect but necessary Markov assumption

$$\begin{aligned} P(w_1, \ldots, w_m) &= \prod_{i=1}^{m} P(w_i | w_1, \ldots, w_{i-1}) \\ &\approx \prod_{i=1}^{m} P(w_i | w_{i-(n-1)}, \ldots, w_{i-1}) \end{aligned}$$

Traditional Language Models

- Probability is usually conditioned on window of n previous words
- An incorrect but necessary Markov assumption

$$\begin{aligned} P(w_1, \ldots, w_m) &= \prod_{i=1}^{m} P(w_i | w_1, \ldots, w_{i-1}) \\ &\approx \prod_{i=1}^{m} P(w_i | w_{i-(n-1)}, \ldots, w_{i-1}) \end{aligned}$$

- To estimate probabilities, compute for unigrams and bigrams (conditioning on one/two previous word(s)):
  $p(w_2|w_1) = \frac{count(w_1,w_2)}{count(w_1)}$ $\quad p(w_3|w_1, w_2) = \frac{count(w_1,w_2,w_3)}{count(w_1,w_2)}$

Traditional Language Models

- Performance improves with keeping around higher n-grams counts and doing smoothing and so-called backoff(e.g. if 4-gram not found, try 3-gram, etc.)

Traditional Language Models

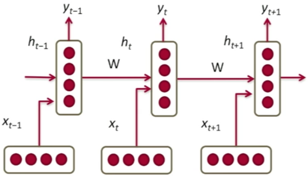
- Performance improves with keeping around higher n-grams counts and doing smoothing and so-called backoff(e.g. if 4-gram not found, try 3-gram, etc.)
- There are A LOT of n-grams! $\rightarrow$ Gigantic RAM requirements!

Traditional Language Models

- Performance improves with keeping around higher n-grams counts and doing smoothing and so-called backoff(e.g. if 4-gram not found, try 3-gram, etc.)
- There are A LOT of n-grams! → Gigantic RAM requirements!
- Recent state of the art: Scalable Modified Kneser-Ney Language Model Estimation by Heafield et al: "Using one machine with 140 GB RAM for 2.8 days, we built an unpruned model on 126 billion" tokens

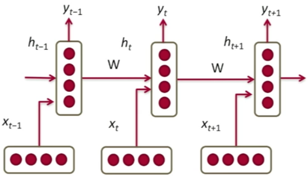
Recurrent Neural Networks

- RNNs tie the weights at each time step



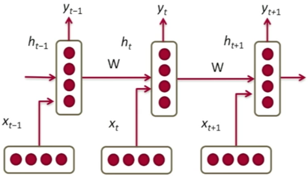
Recurrent Neural Networks

- RNNs tie the weights at each time step
- Condition the neural network on all previous words

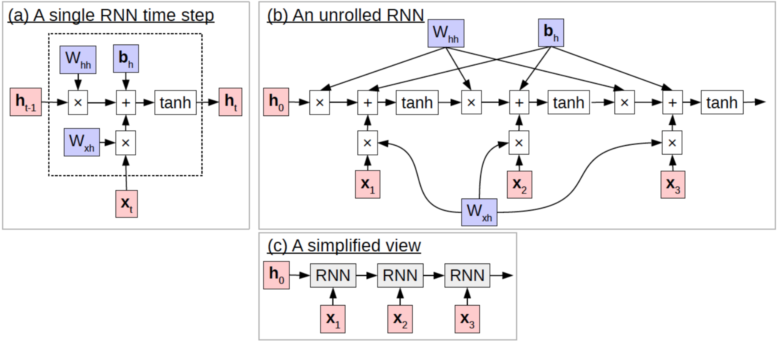


Recurrent Neural Networks

- RNNs tie the weights at each time step
- Condition the neural network on all previous words
- RAM requirement only scales with number of words



Recurrent Neural Networks



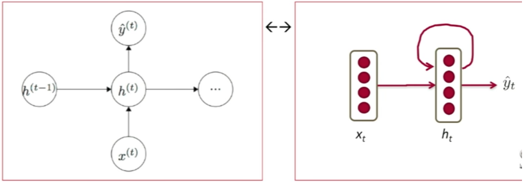
Recurrent Neural Network Language Model

Given a list of word vectors:

$$x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_T$$

At a single time step:

$$\begin{aligned} h_t &= \sigma\left(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]}\right) \\ \hat{y}_t &= softmax\left(W^{(s)} h_t\right) \\ \hat{P}(x_{t+1} = v_j | x_t, \ldots, x_1) &= \hat{y}_{t,j} \end{aligned}$$



Recurrent Neural Network Language Model

Main idea: we use the same set of W weights at all time steps!
Everything else is the same:

$$\begin{aligned} h_t &= \sigma\left(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]}\right) \\ \hat{y}_t &= softmax\left(W^{(s)} h_t\right) \\ \hat{P}(x_{t+1} = v_j | x_t, \ldots, x_1) &= \hat{y}_{t,j} \end{aligned}$$

$h_0 \in \mathbb{R}^{D_h}$ is some initialization vector for the hidden layer at time step 0

$x_{[t]}$ is the column vector of L at index $[t]$ at time step t

$W^{(hh)} \in \mathbb{R}^{D_h \times D_h}$ $W^{(hx)} \in \mathbb{R}^{D_h \times d}$ $W^{(S)} \in \mathbb{R}^{|V| \times D_h}$

Training RNNs

$\hat{y} \in \mathbb{R}^{|V|}$ is a probability distribution over the vocabulary

Same cross entropy loss function but predicting words instead of classes

$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j}$$

RNNs Evaluation

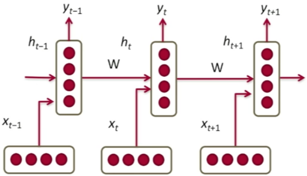
Evaluation could just be negative of average log probability over dataset of size (number of words) T:

$$J = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j}$$

But more common: Perplexity: 2^J
Lower is better!

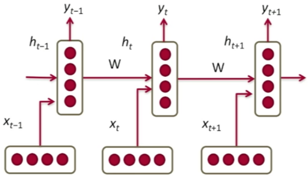
Training RNN is hard

- Multiply the same matrix at each time step during forward prop



Training RNN is hard

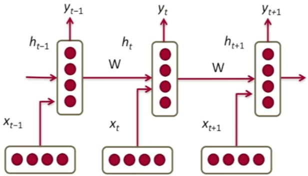
- Multiply the same matrix at each time step during forward prop



- Ideally inputs from many steps ago can modify output y

Training RNN is hard

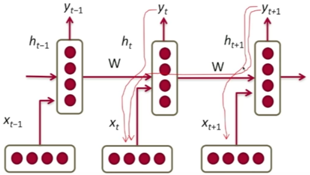
- Multiply the same matrix at each time step during forward prop



- Ideally inputs from many steps ago can modify output y
- Take $\frac{\partial E_2}{\partial W}$ for an example RNN with 2 time steps

The vanishing/exploding gradient problem

Multiply the same matrix at each time step during backprop



The vanishing gradient problem - details

- Similar but simpler RNN formulation:

$$
\begin{aligned}
h_t &= Wf(h_{t-1}) + W^{(hx)} x_{[t]} \\
\hat{y}_t &= W^{(S)} f(h_t)
\end{aligned}
$$

The vanishing gradient problem - details

- Similar but simpler RNN formulation:

$$\begin{aligned} h_t &= Wf(h_{t-1}) + W^{(hx)}x_{[t]} \\ \hat{y}_t &= W^{(S)}f(h_t) \end{aligned}$$

- Total error is the sum of each error at time steps t

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

The vanishing gradient problem - details

- Similar but simpler RNN formulation:

$$\begin{aligned} h_t &= Wf(h_{t-1}) + W^{(hx)}x_{[t]} \\ \hat{y}_t &= W^{(S)}f(h_t) \end{aligned}$$

- Total error is the sum of each error at time steps t

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

- Hardcore chain rule application:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

The vanishing gradient problem - details

- Similar to backprop but less efficient formulation

The vanishing gradient problem - details

- Similar to backprop but less efficient formulation
- Useful for analysis we'll look at:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \boxed{\frac{\partial h_t}{\partial h_k}} \frac{\partial h_k}{\partial W}$$

The vanishing gradient problem - details

- Similar to backprop but less efficient formulation
- Useful for analysis we'll look at:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \boxed{\frac{\partial h_t}{\partial h_k}} \frac{\partial h_k}{\partial W}$$

- Remember: $h_t = Wf(h_{t-1}) + W^{(hx)} x_{[t]}$

The vanishing gradient problem - details

- Similar to backprop but less efficient formulation
- Useful for analysis we'll look at:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \boxed{\frac{\partial h_t}{\partial h_k}} \frac{\partial h_k}{\partial W}$$

- Remember: $h_t = Wf(h_{t-1}) + W^{(hx)} x_{[t]}$
- More chain rule, remember:

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$

The vanishing gradient problem - details

- Similar to backprop but less efficient formulation
- Useful for analysis we'll look at:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \boxed{\frac{\partial h_t}{\partial h_k}} \frac{\partial h_k}{\partial W}$$

- Remember: $h_t = Wf(h_{t-1}) + W^{(hx)} x_{[t]}$
- More chain rule, remember:

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$

- Each partial is a Jacobian:

$$\frac{d\mathbf{f}}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

The vanishing gradient problem - details

- Analyzing the norms of the Jacobians, yields:

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \|W^T\| \|diag[f'(h_{j-1})]\| \leq \beta_W \beta_h$$

The vanishing gradient problem - details

- Analyzing the norms of the Jacobians, yields:

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \|W^T\| \|diag[f'(h_{j-1})]\| \leq \beta_W \beta_h$$

- where we defined β's as upper bounds of the norms

The vanishing gradient problem - details

- Analyzing the norms of the Jacobians, yields:

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \|W^T\| \|diag[f'(h_{j-1})]\| \leq \beta_W \beta_h$$

- where we defined β's as upper bounds of the norms
- The gradient is a product of Jacobian matrices, each associated with a step in the forward computation

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| = \left\| \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq (\beta_W \beta_H)^{t-k}$$

The vanishing gradient problem - details

- Analyzing the norms of the Jacobians, yields:

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \|W^T\| \|diag[f'(h_{j-1})]\| \leq \beta_W \beta_h$$

- where we defined β's as upper bounds of the norms
- The gradient is a product of Jacobian matrices, each associated with a step in the forward computation

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| = \left\| \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq (\beta_W \beta_H)^{t-k}$$

- This can become very small or very large quickly [Bengio et al 1994], and the locality assumption of gradient descent breaks down $\rightarrow$ Vanishing or exploding gradient

The vanishing gradient problem for language models

- In the case of language modeling or question answering words from time steps far away are not taken into consideration when training to predict the next word

The vanishing gradient problem for language models

- In the case of language modeling or question answering words from time steps far away are not taken into consideration when training to predict the next word
- Example:
  Jane walked into the room. John walked in too. It was late in the day. Jain said hi to ___

The Vanishing gradient problem

- Example of simple and clean NNet implementation

The Vanishing gradient problem

- Example of simple and clean NNet implementation
- Comparison of sigmoid and ReLu units

The Vanishing gradient problem

- Example of simple and clean NNet implementation
- Comparison of sigmoid and ReLu units
- A litle bit of vanishing gradient

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)} \right) o \;\; f'(z^{(l)})$$

$$\frac{\partial}{\partial W^{(l)}} E_R = \delta^{(l+1)} (a^{(l)})^T + \lambda W^{(l)}$$

The Vanishing gradient problem IPython notebook

https://cs224d.stanford.edu/notebooks/vanishing_grad_example.html

Trick for exploding: clipping trick

- The solution first introduced by Mikolov is to clip gradients to a maximum value

Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

Trick for exploding: clipping trick

- The solution first introduced by Mikolov is to clip gradients to a maximum value

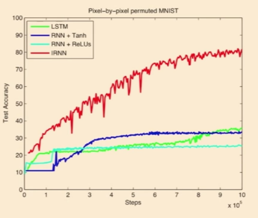
Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

- Makes a big difference in RNNs

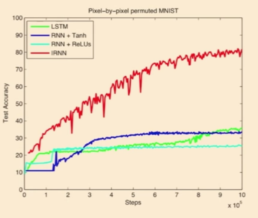
For vanishing gradients: Initialization+ReLus

- Initialize $W^{(*)}$'s to identity matrix I and $f(z) = rect(z) = max(z, 0)$



For vanishing gradients: Initialization+ReLus

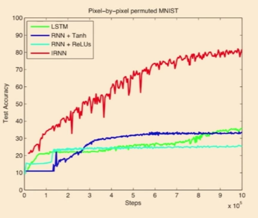
- Initialize $W^{(*)}$'s to identity matrix I and $f(z) = rect(z) = max(z, 0)$



- introduced in *Parsing with Compositional Vector Grammars*, Socher et al. 2013

For vanishing gradients: Initialization+ReLus

- Initialize $W^{(*)}$'s to identity matrix I and $f(z) = rect(z) = max(z, 0)$



- introduced in *Parsing with Compositional Vector Grammars*, Socher et al. 2013
- New experiments with RNNs in *A Simple Way to initialize Recurrent Networks of Rectified Linear Units, Le et al, 2015*

Perplexity Results

KN5 = Count-based language model with Kneser-Ney smoothing & 5-grams

Table 2. *Comparison of different neural network architectures on Penn Corpus (1M words) and Switchboard (4M words).*

	Penn Corpus		Switchboard	
Model	NN	NN+KN	NN	NN+KN
KN5 (baseline)	-	141	-	92.9
feedforward NN	141	118	85.1	77.5
RNN trained by BP	137	113	81.3	75.4
RNN trained by BPTT	123	106	77.5	72.5

Extensions of recurrent neural network language model by Mikolov et al 2011

Sequence modeling for other tasks

Classify each word into:

- NER

Sequence modeling for other tasks

Classify each word into:

- NER
- Entity level sentiment in context

Sequence modeling for other tasks

Classify each word into:

- NER
- Entity level sentiment in context
- opinionated expressions

Opinion Mining with Deep Recurrent Nets

paper: *Opinion Mining with Deep Recurrent Nets* by Irsoy and Cardie, 2014
Goal: classify each word as
direct subjective expressions (DSEs) and
expressive subjective expressions (ESEs)
DSE: Explicit mentions of private states or speech events expressing private states
ESE: Expressions that indicate sentiment, emotion, etc. without explicitly conveying them

Example Annotation

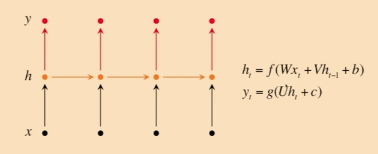
In BIO notation (tags either begin-of-entity (B_*X*) or continuation-of-entity (I_*X*)):
The committee, [as usual]$_{ESE}$, [has refused to make any statements]$_{DSE}$.

The	committee	,	as	usual	,	has
O	O	O	B_ESE	I_ESE	O	B_DSE

refused	to	make	any	statements	.
I_DSE	I_DSE	I_DSE	I_DSE	I_DSE	O

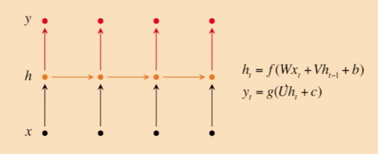
Approach: Recurrent Neural Network

- Notation from paper



Approach: Recurrent Neural Network

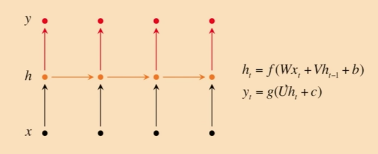
- Notation from paper



- x represents a token (word) as a vector

Approach: Recurrent Neural Network

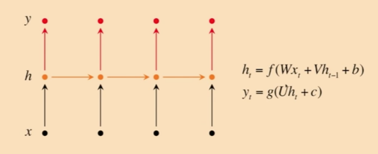
- Notation from paper



- x represents a token (word) as a vector
- y represents the output label (B,I or O) - g = softmax

Approach: Recurrent Neural Network

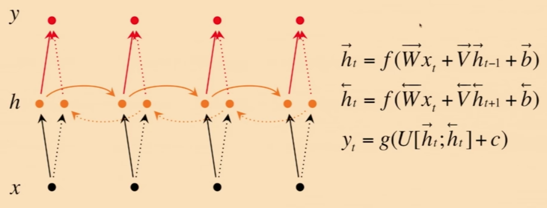
- Notation from paper



- x represents a token (word) as a vector
- y represents the output label (B,I or O) - g = softmax
- h is the memory, computed from the past memory and current word. It summarizes the sentence up to that time

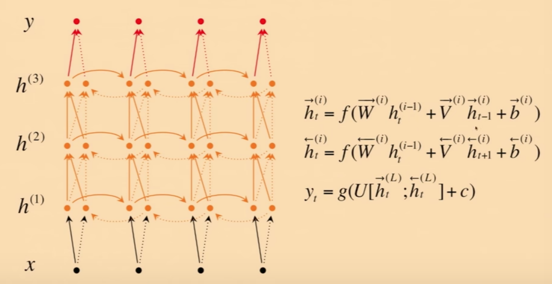
Bidirectional RNNs

Problem: for classification you want to incorporate information from words both preceding and following



$h = [\overrightarrow{h}; \overleftarrow{h}]$ now represents (summarizes) the past and future around a single token

Deep Bidirectional RNNs



Each memory layer passes an intermediate sequential representation to the next

Machine Translation

- Methods are statistical

Machine Translation

- Methods are statistical
- Use parallel corpora : European Parliament

Machine Translation

- Methods are statistical
- Use parallel corpora : European Parliament
- First parallel corpus: Rosetta Stone

Machine Translation

- Methods are statistical
- Use parallel corpora : European Parliament
- First parallel corpus: Rosetta Stone
- Traditional systems are very complex

Current statistical machine translation systems

- Source language f, e.g. French

Current statistical machine translation systems

- Source language f, e.g. French
- Target language e, e.g. English

Current statistical machine translation systems

- Source language f, e.g. French
- Target language e, e.g. English
- Probabilistic formulation (using Bayes rule)

$$\hat{e} = \arg\max_{e} p(e|f) = \arg\max_{e} p(f|e)p(e)$$

Current statistical machine translation systems

- Source language f, e.g. French
- Target language e, e.g. English
- Probabilistic formulation (using Bayes rule)

$$\hat{e} = \arg\max_{e} p(e|f) = \arg\max_{e} p(f|e)p(e)$$

- Translation model $p(f|e)$ trained on parallel corpus

Current statistical machine translation systems

- Source language f, e.g. French
- Target language e, e.g. English
- Probabilistic formulation (using Bayes rule)

$$\hat{e} = \arg\max_{e} p(e|f) = \arg\max_{e} p(f|e)p(e)$$

- Translation model $p(f|e)$ trained on parallel corpus
- Language model $p(e)$ trained on English only corpus (lots, free)

Traditional MT

- Skipped hundreds of important details

Traditional MT

- Skipped hundreds of important details
- A lot of human feature engineering

Traditional MT

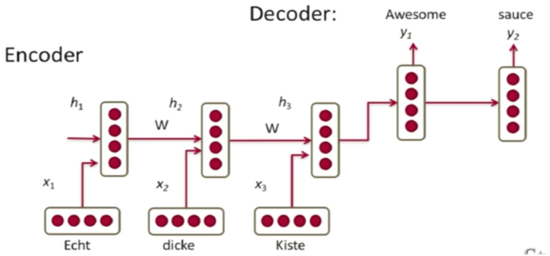
- Skipped hundreds of important details
- A lot of human feature engineering
- Very complex systems

Traditional MT

- Skipped hundreds of important details
- A lot of human feature engineering
- Very complex systems
- Many different, independently trained machine learning problems

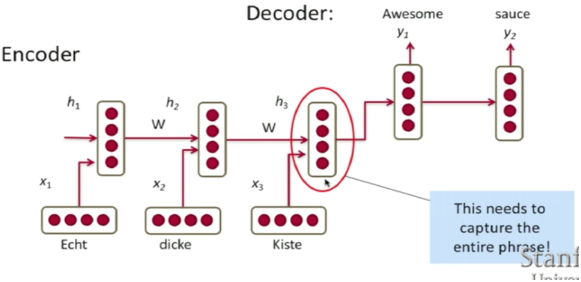
Deep learning to the rescue

Maybe, we could translate directly with an RNN?



Deep learning to the rescue

Maybe, we could translate directly with an RNN?



MT with RNNs - Simplest Model

Encoder: $h_t = \phi(h_{t-1}, x_t) = f\left(W^{(hh)} h_{t-1} + W^{(hx)} x_t\right)$
Decoder: $h_t = \phi(h_{t-1}) = f\left(W^{(hh)} h_{t-1}\right)$

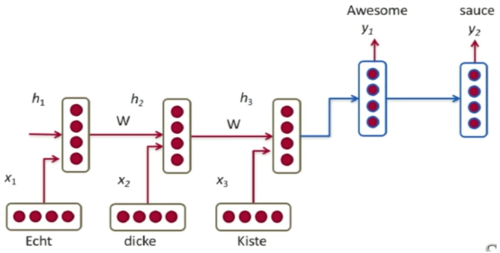
$$y_t = softmax\left(W^{(s)} h_t\right)$$

Minimize cross entropy error for all target words conditioned on source words

$$max_\theta \frac{1}{N} \sum_{n=1}^{N} \log p_\theta(y^{(n)} | x^{(n)})$$

RNN Translation Model Extensions

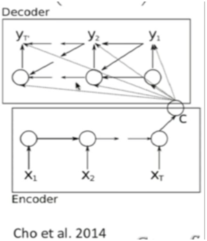
1. Train different RNN weights for encoding and decoding



RNN Translation Model Extensions

Notation: Each input of ϕ has its own linear transformation matrix. Simple: $h_t = \phi(h_{t-1}) = f\left(W^{(hh)} h_{t-1}\right)$

2. Compute every hidden state in decoder from

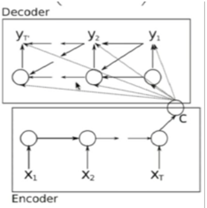


- Previous hidden state (standard)

RNN Translation Model Extensions

Notation: Each input of ϕ has its own linear transformation matrix. Simple: $h_t = \phi(h_{t-1}) = f\left(W^{(hh)} h_{t-1}\right)$

2. Compute every hidden state in decoder from



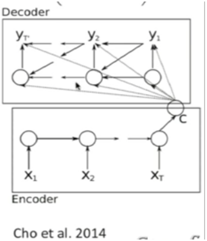
Cho et al. 2014

- Previous hidden state (standard)
- Last hidden vector of encoder $c = h_t$

RNN Translation Model Extensions

Notation: Each input of ϕ has its own linear transformation matrix. Simple: $h_t = \phi(h_{t-1}) = f\left(W^{(hh)} h_{t-1}\right)$

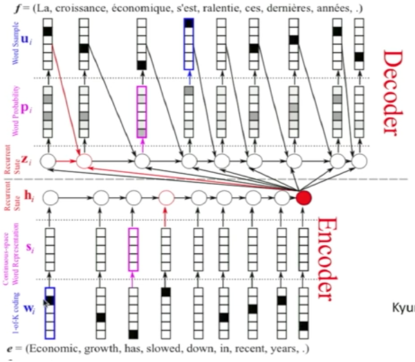
2. Compute every hidden state in decoder from



- Previous hidden state (standard)
- Last hidden vector of encoder $c = h_t$
- Previous predicted output word y_{t-1}

$$h_{D,t} = \phi_D(h_{t-1}, c, y_{t-1})$$

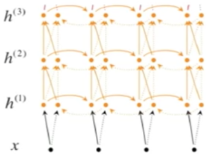
Different picture same idea



Kyunghyun Cho et al.

RNN Translation Model Extensions

3. Train stacked / deep RNNs with multiple layers



4. Potentially train bidirectional encoder
5. Train input sequence in reverse order for simpler optimization problem: instead of A B C $\rightarrow$ X Y, train with C B A $\rightarrow$ X Y

Main Improvement: Better Units

- More complex hidden unit computation in recurrence

Main Improvement: Better Units

- More complex hidden unit computation in recurrence
- Gated Recurrent Units (GRU) introduced by Cho et al 2014

Main Improvement: Better Units

- More complex hidden unit computation in recurrence
- Gated Recurrent Units (GRU) introduced by Cho et al 2014
- Main ideas: 1) Keep around memories to capture long distance dependencies
  2) allow error messages to flow at different strengths depending on the inputs

GRUs

- Standard RNN computes hidden layer at next time step directly: $h_t = f\left(W^{(hh)}h_{t-1} + W^{(hx)}x_t\right)$

GRUs

- Standard RNN computes hidden layer at next time step directly: $h_t = f\left(W^{(hh)}h_{t-1} + W^{(hx)}x_t\right)$
- GRU first computes an update gate (another layer) based on current input word vector and hidden sate

$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$

GRUs

- Standard RNN computes hidden layer at next time step directly: $h_t = f\left(W^{(hh)}h_{t-1} + W^{(hx)}x_t\right)$
- GRU first computes an update gate (another layer) based on current input word vector and hidden sate

$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$

- Compute reset gate similarly but with different weights

$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$

GRUs

- Update gate $z_t = \sigma\left(W^{(z)} x_t + U^{(z)} h_{t-1}\right)$

GRUs

- Update gate $z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$
- Reset gate $r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$

GRUs

- Update gate $z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$
- Reset gate $r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$
- New memory content: $\tilde{\mathrm{h}}_t = tanh(Wx_t + r_t\ o\ Uh_{t-1})$, if reset gate is 0, then this ignores previous memory and only stores the new word information

GRUs

- Update gate $z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$
- Reset gate $r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$
- New memory content: $\tilde{h}_t = tanh(Wx_t + r_t \ o \ Uh_{t-1})$, if reset gate is 0, then this ignores previous memory and only stores the new word information
- Final memory at time step combines current and previous time steps: $h_t = z_t \ o \ h_{t-1} + (1 - z_t) \ o \ \tilde{h}_t$

GRU Intuition

- if reset is close to 0, ignore previous hidden state $\rightarrow$ allows model to drop information that is irrelevant in the future

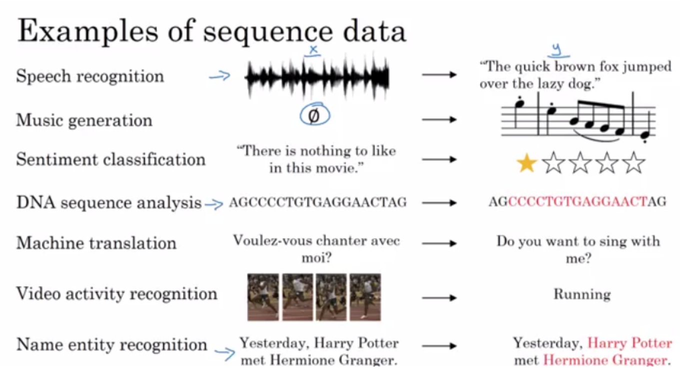
GRU Intuition

- if reset is close to 0, ignore previous hidden state $\rightarrow$ allows model to drop information that is irrelevant in the future
- Update gate z controls how much of past state should matter now
  if z is close to 1, then we can copy information in that unit through many steps! Less vanishing gradient!

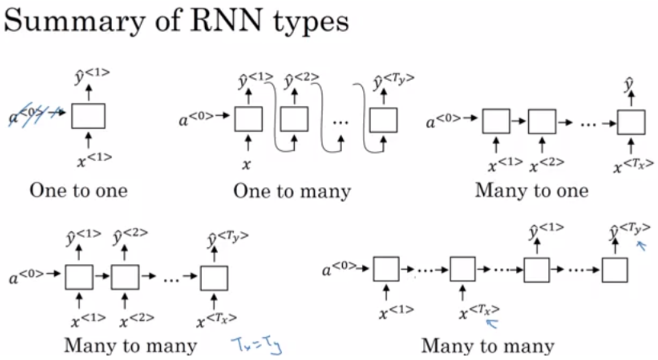
GRU Intuition

- if reset is close to 0, ignore previous hidden state $\rightarrow$ allows model to drop information that is irrelevant in the future
- Update gate z controls how much of past state should matter now
  if z is close to 1, then we can copy information in that unit through many steps! Less vanishing gradient!
- Units with short-term dependencies often have reset gates very active

Sequence Data (Andrew Ng)



RNN Architectures (Andrew Ng)

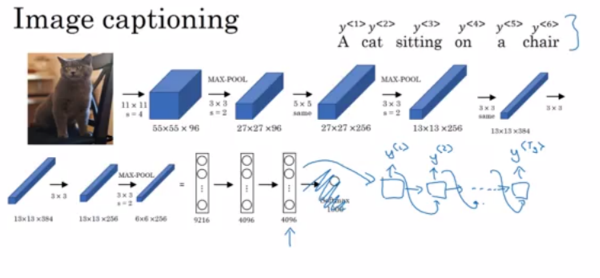


Examples

Architecture and Parameters
$(W^{(hh)} \in \mathbb{R}^{? \times ?} \quad W^{(hx)} \in \mathbb{R}^{? \times ?} \quad W^{(S)} \in \mathbb{R}^{? \times ?})$? (number of hidden cells:100, embedding dim: 300)

- Named Entity Recognition: Barack Obama won the elections
- Sentiment analysis: I like this movie
- Image Captioning

Image Captioning



Tarea

- Resolver el problema de Sentiment Analysis (NeuralNetworkSemana3.ipynb) usando una RNN bidireccional de por lo menos 2 capas
- Utilizar GRU units o LSTM units y *gradient clipping*
- Puede usarse Tensorflow