NN Language Models

David Vilar
david.vilar@nuance.com

MT Marathon 2016
14. September 2016

About Myself

2003-2010 RWTH AACHEN UNIVERSITY

PhD on hierarchical MT

Main author of Jane MT toolkit

2011-2013 DFKI

Researcher. More work on MT, trying to make it usable for professional translators

2013-2014 Pixformance

Lead developer

Since 2014 NUANCE

Sr. Research Scientist: Language Modelling and Natural Language Understanding

Overview

1. Introduction to Word Embeddings
2. Recurrent Neural Networks
3. LSTMs
4. A Few Notes About the Output Layer

Introduction to Word Embeddings

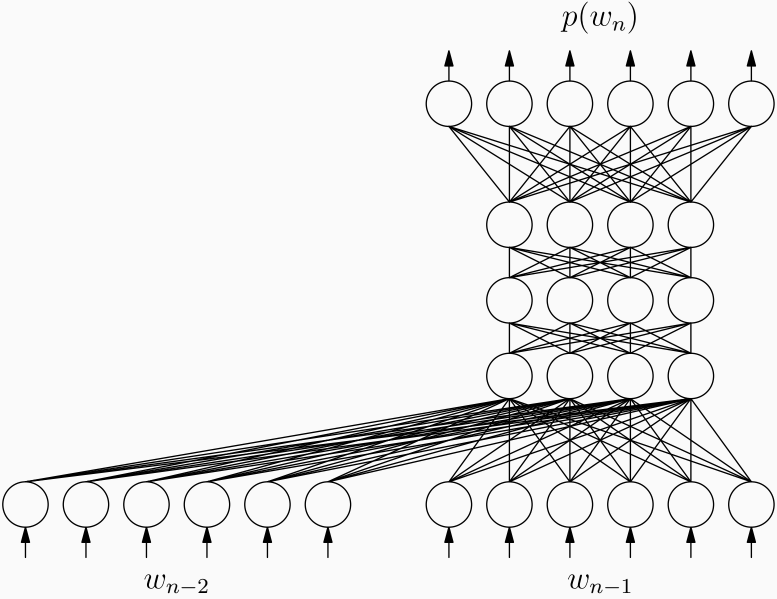
1-hot encodings

- 1-hot encoding is the "natural" way to encode symbolic information (e.g. words)
- But:
  - The encoding itself is arbitrary (e.g. first appearance of a word in the training text)
  - No useful information can be read from the vector representation
  - Example:

the green dog bites the cat

the	$(1, 0, 0, 0, 0)$
green	$(0, 1, 0, 0, 0)$
dog	$(0, 0, 1, 0, 0)$
bites	$(0, 0, 0, 1, 0)$
cat	$(0, 0, 0, 0, 1)$

Feed-forward LM

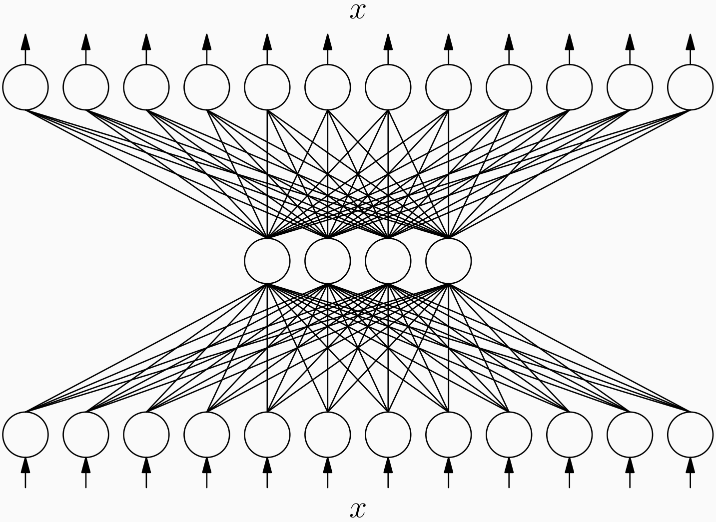


Intuition

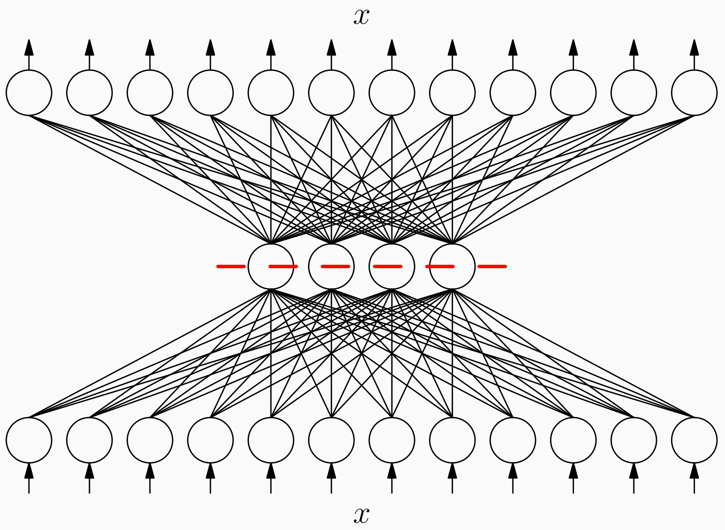
- A NN represents a *flow* of information
- A NN can be decomposed into smaller networks
- Each of these networks transforms the information, which serves as input to the next network
- Can be seen in the recursive structure of the equations

$$y^{(l)}(x) = f(W^{(l)} y^{(l-1)}(x) + b^{(l)})$$

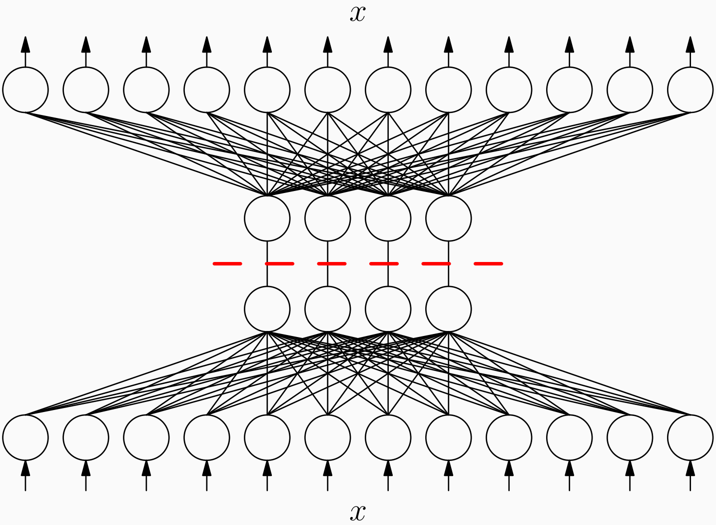
The most "stupid" network



The most "stupid" network



The most "stupid" network



The most “stupid” network

If the “stupid” network has no errors:

- We mapped an 12-dimensional (sparse?) vector into a 4-dimensional dense vector

The most "stupid" network

If the "stupid" network has no errors:

- We mapped an 12-dimensional (sparse?) vector into a 4-dimensional dense vector

However:

- The representation is still arbitrary, as no information about the word themselves is taken into account

The most “stupid” network

If the “stupid” network has no errors:

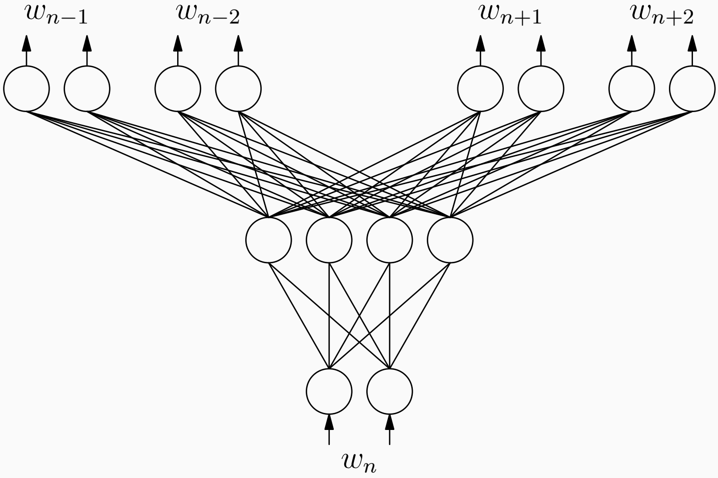
- We mapped an 12-dimensional (sparse?) vector into a 4-dimensional dense vector

However:

- The representation is still arbitrary, as no information about the word themselves is taken into account

We can do better!

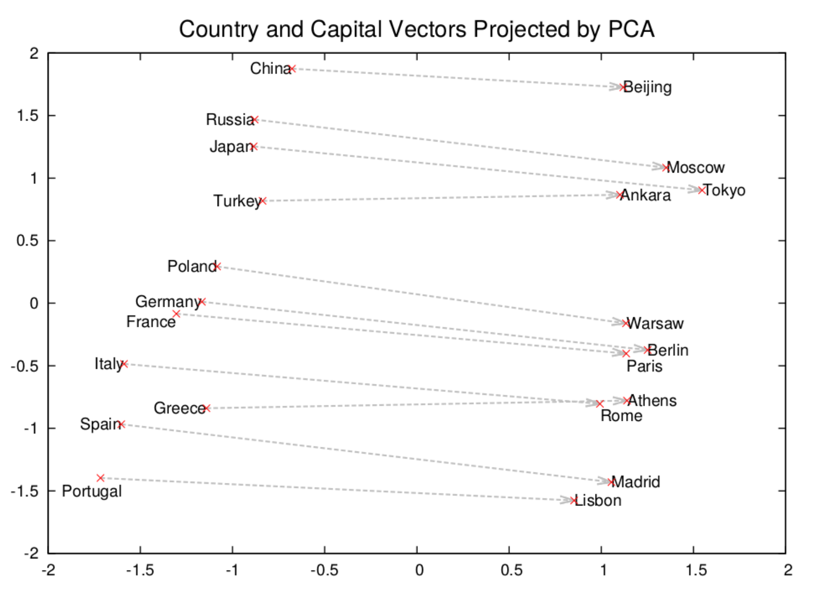
Skip-gram model



Skip-gram model

- Assumption: similar words appear in similar contexts
- Goal: similar words have similar representations (as they will predict similar contexts)
- Indeed:
    - $\text{vec}(King) - \text{vec}(Man) + \text{vec}(Woman)$ results in a vector that is closest to $Queen$
    - $\text{vec}(Madrid) - \text{vec}(Spain) + \text{vec}(France)$ results in a vector that is closest to $Paris$

Skip-gram model



- Different implementations available (many of them open source)
- (One of) The most widely used: `word2vec` by Mikolov et al.
- Efficient implementation, can deal with big datasets
- `https://code.google.com/archive/p/word2vec/`
- Normally used pre-training for embedding layer
  - May be further refined by task-specific training

Recurrent Neural Networks

Recap

- Language model

$$p(w_1^N)$$

- Chain rule (mathematical equality)

$$p(w_1^N) = \prod_{n=1}^{N} p(w_n | w_1^{n-1})$$

- k-th order Markov *assumption*: $(k+1)$-grams

$$p(w_1^N) \approx \prod_{n=1}^{N} p(w_n | w_{n-k}^{n-1})$$

Recap

Advantage of NNLMs we encountered up to this point:

- FF language models deal with the sparsity problem (by projecting into a continuous space)

Recap

Advantage of NNLMs we encountered up to this point:

- FF language models deal with the sparsity problem (by projecting into a continuous space)
- but they still are under the Markov chain assumption

Recap

Advantage of NNLMs we encountered up to this point:

- FF language models deal with the sparsity problem (by projecting into a continuous space)
- but they still are under the Markov chain assumption

We would like to be able to take into account the *whole* history!

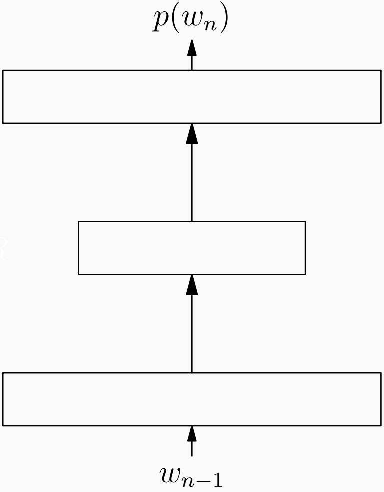
Recap

Advantage of NNLMs we encountered up to this point:

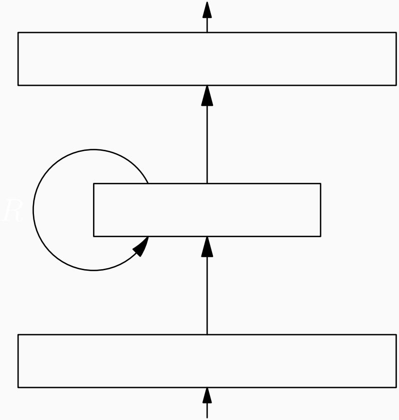
- FF language models deal with the sparsity problem (by projecting into a continuous space)
- but they still are under the Markov chain assumption

We would like to be able to take into account the *whole* history!
$\rightarrow$ Let the network remember everything it has seen!

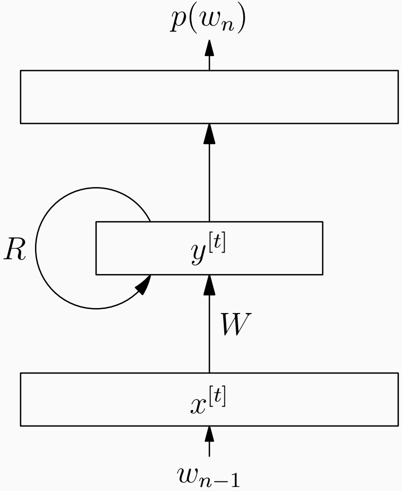
Recurrent NNs



Recurrent NNs



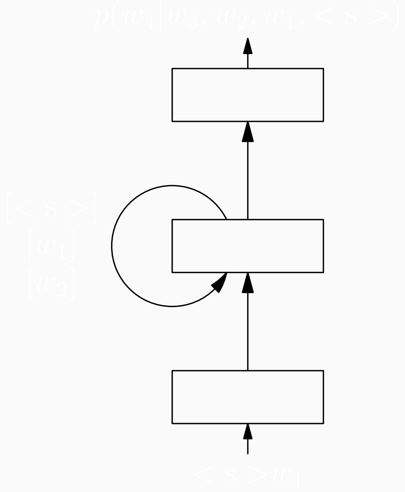
Recurrent NNs



In Equations: $y^{[t]} = f(Wx^{[t]} + Ry^{[t-1]} + b)$

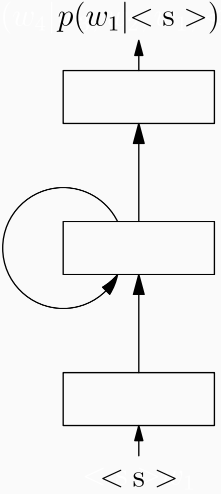
Recurrent NNs

$p(w_1^4) =$



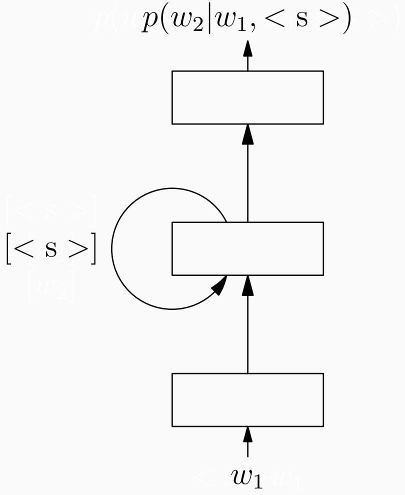
Recurrent NNs

$p(w_1^4) =$
$p(w_1|<\mathrm{s}>)$

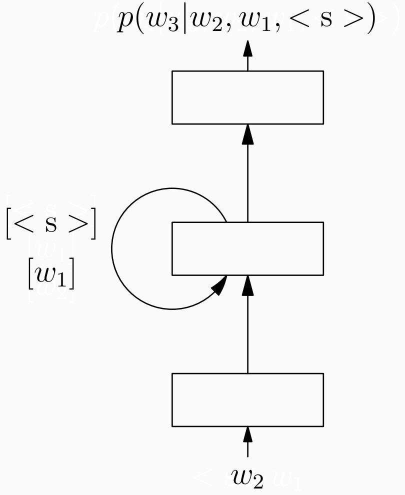


Recurrent NNs

$p(w_1^4) =$
$p(w_1 | <\mathrm{s}>)$
$\times p(w_2 | w_1, <\mathrm{s}>)$

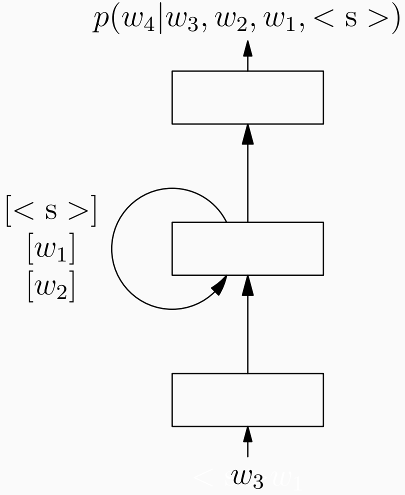


$p(w_1^4) =$
$p(w_1 | \text{< s >})$
$\times p(w_2 | w_1, \text{< s >})$
$\times p(w_3 | w_2, w_1, \text{< s >})$



Recurrent NNs

$p(w_1^4) =$
$p(w_1|< \mathrm{s} >)$
$\times p(w_2|w_1, < \mathrm{s} >)$
$\times p(w_3|w_2, w_1, < \mathrm{s} >)$
$\times p(w_4|w_3, w_2, w_1, < \mathrm{s} >)$



Backpropagation through time

How to train a RNN?

- Of course…

Backpropagation through time

How to train a RNN?

- Of course... with backpropagation

Backpropagation through time

How to train a RNN?

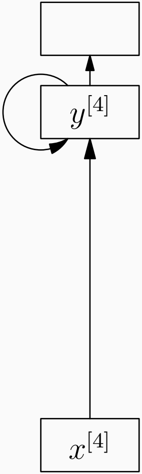
- Of course… with backpropagation
- Unfold recurrent connections through time
- Results in a wide network, backpropagation can be used

Backpropagation through time

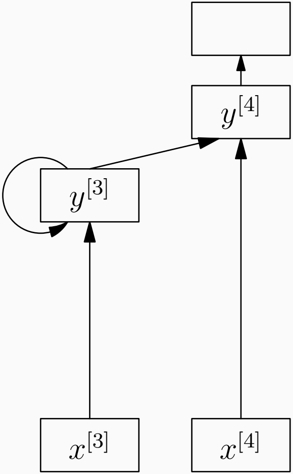
How to train a RNN?

- Of course. . . with backpropagation
- Unfold recurrent connections through time
- Results in a wide network, backpropagation can be used
- Use chain rule not only for layers, but also for time steps

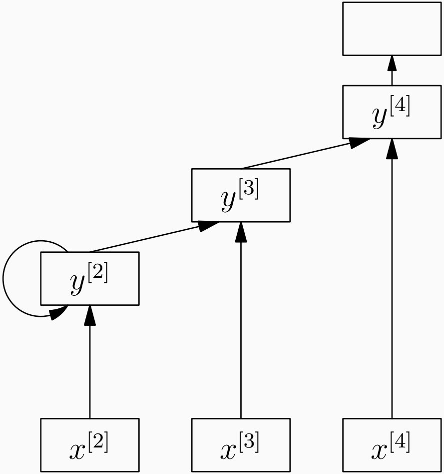
Backpropagation through time



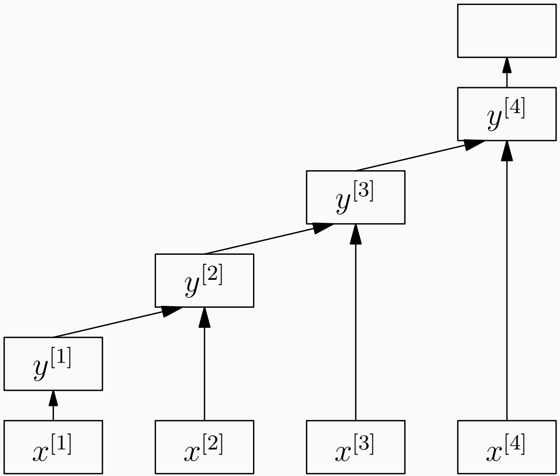
Backpropagation through time



Backpropagation through time

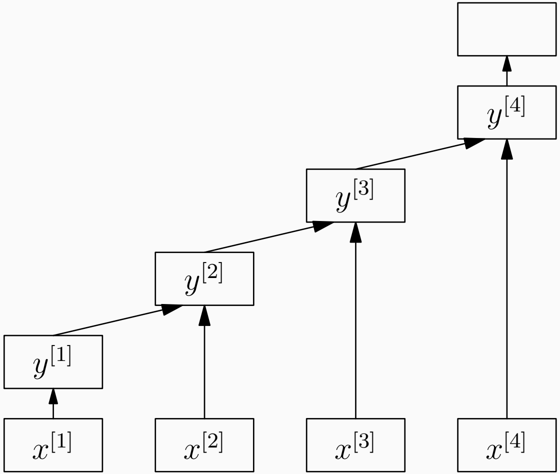


Backpropagation through time

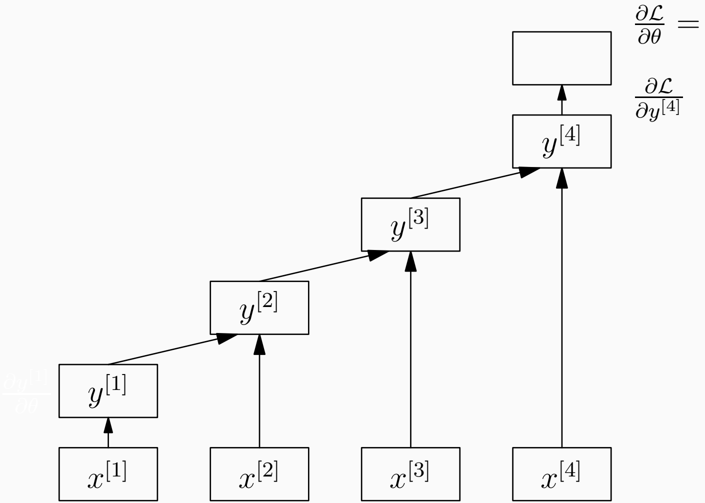


Backpropagation through time

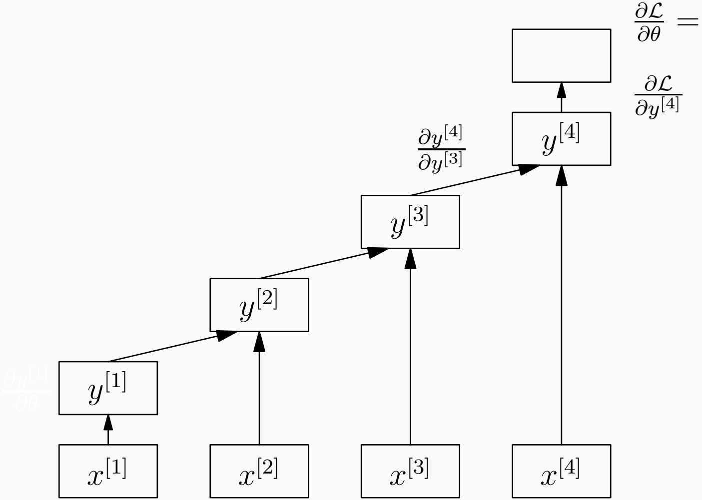
$$\frac{\partial \mathcal{L}}{\partial \theta} =$$



Backpropagation through time

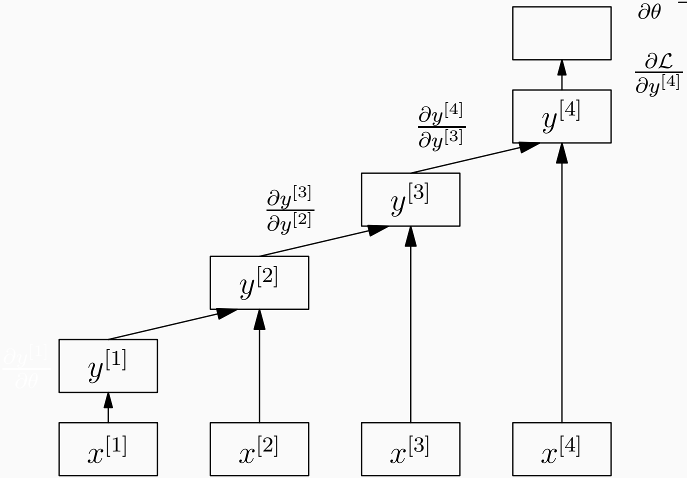


Backpropagation through time

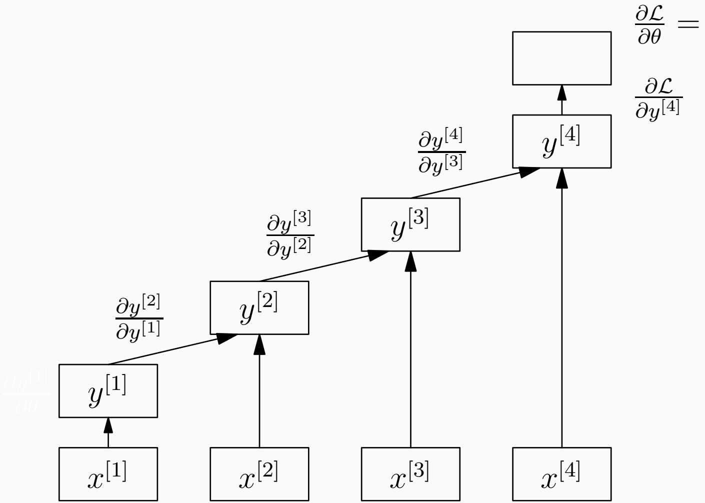


Backpropagation through time

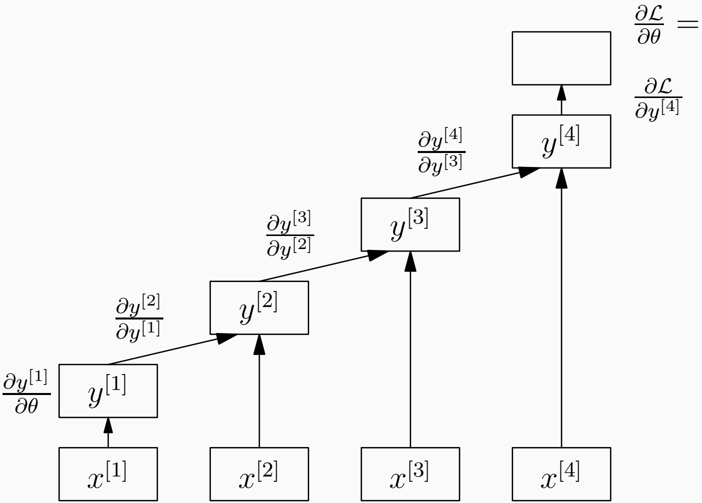
$$\frac{\partial \mathcal{L}}{\partial \theta} =$$



Backpropagation through time



Backpropagation through time



Exploding and vanishing gradient

Observation: sometimes the gradient "misbehaves"

Exploding and vanishing gradient

Observation: sometimes the gradient "misbehaves"

- Sometimes *vanishes* (norm ≈ 0)

Exploding and vanishing gradient

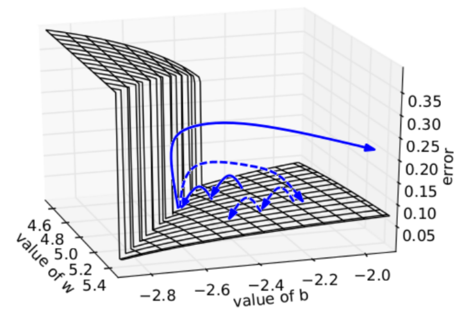
Observation: sometimes the gradient "misbehaves"

- Sometimes *vanishes* (norm ≈ 0)
- Sometimes *explodes* (norm $\to \infty$)

Exploding and vanishing gradient

Observation: sometimes the gradient "misbehaves"

- Sometimes *vanishes* (norm ≈ 0)
- Sometimes *explodes* (norm $\to \infty$)



Exploding and vanishing gradient

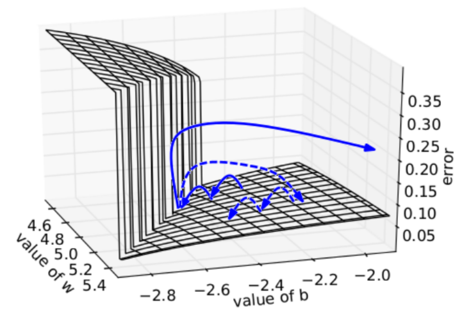
What to do?

- Exploding gradient: clip the gradient (divide by the norm) (Full vector or element-wise)

Exploding and vanishing gradient

What to do?

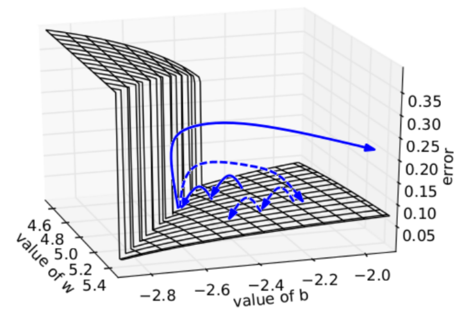
- Exploding gradient: clip the gradient (divide by the norm) (Full vector or element-wise)



Exploding and vanishing gradient

What to do?

- Exploding gradient: clip the gradient (divide by the norm) (Full vector or element-wise)

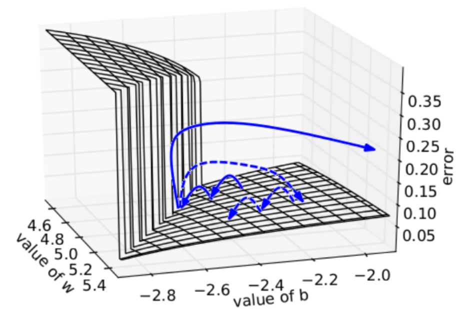


- Vanishing gradient:

Exploding and vanishing gradient

What to do?

- Exploding gradient: clip the gradient (divide by the norm) (Full vector or element-wise)



- Vanishing gradient: you have a problem!

Exploding and vanishing gradient

Why does this happen?

Exploding and vanishing gradient

Why does this happen?

Sequence of length T, $y^{[t]} = f(Wx^{[t]} + Ry^{[t-1]} + b)$.

Exploding and vanishing gradient

Why does this happen?

Sequence of length T, $y^{[t]} = f(Wx^{[t]} + Ry^{[t-1]} + b)$.

Derivative of the loss function $\mathcal{L}$:

$$\frac{\partial \mathcal{L}}{\partial \theta}$$

Exploding and vanishing gradient

Why does this happen?

Sequence of length T, $y^{[t]} = f(Wx^{[t]} + Ry^{[t-1]} + b)$.

Derivative of the loss function $\mathcal{L}$:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{1 \leq t_2 \leq T} \frac{\partial \mathcal{L}^{[t_2]}}{\partial \theta}$$

Exploding and vanishing gradient

Why does this happen?

Sequence of length T, $y^{[t]} = f(Wx^{[t]} + Ry^{[t-1]} + b)$.

Derivative of the loss function $\mathcal{L}$:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{1 \le t_2 \le T} \frac{\partial \mathcal{L}^{[t_2]}}{\partial \theta} = \sum_{1 \le t_2 \le T} \sum_{1 \le t_1 \le t_2} \frac{\partial \mathcal{L}^{[t_2]}}{\partial y^{[t_2]}} \frac{\partial y^{[t_2]}}{\partial y^{[t_1]}} \frac{\partial y^{[t_1]}}{\partial \theta}$$

Exploding and vanishing gradient

Why does this happen?

Sequence of length T, $y^{[t]} = f(Wx^{[t]} + Ry^{[t-1]} + b)$.

Derivative of the loss function $\mathcal{L}$:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{1 \le t_2 \le T} \frac{\partial \mathcal{L}^{[t_2]}}{\partial \theta} = \sum_{1 \le t_2 \le T} \sum_{1 \le t_1 \le t_2} \frac{\partial \mathcal{L}^{[t_2]}}{\partial y^{[t_2]}} \frac{\partial y^{[t_2]}}{\partial y^{[t_1]}} \frac{\partial y^{[t_1]}}{\partial \theta}$$

and for $\frac{\partial y^{[t_2]}}{\partial y^{[t_1]}}$:

$$\frac{\partial y^{[t_2]}}{\partial y^{[t_1]}} = \prod_{t_1 < t \le t_2} \frac{\partial y^{[t]}}{\partial y^{[t-1]}}$$

Exploding and vanishing gradient

Why does this happen?

Sequence of length T, $y^{[t]} = f(Wx^{[t]} + Ry^{[t-1]} + b)$.

Derivative of the loss function $\mathcal{L}$:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{1 \le t_2 \le T} \frac{\partial \mathcal{L}^{[t_2]}}{\partial \theta} = \sum_{1 \le t_2 \le T} \sum_{1 \le t_1 \le t_2} \frac{\partial \mathcal{L}^{[t_2]}}{\partial y^{[t_2]}} \frac{\partial y^{[t_2]}}{\partial y^{[t_1]}} \frac{\partial y^{[t_1]}}{\partial \theta}$$

and for $\frac{\partial y^{[t_2]}}{\partial y^{[t_1]}}$:

$$\frac{\partial y^{[t_2]}}{\partial y^{[t_1]}} = \prod_{t_1 < t \le t_2} \frac{\partial y^{[t]}}{\partial y^{[t-1]}} = \prod_{t_1 < t \le t_2} R^T \operatorname{diag}\left(f'(Ry^{[t-1]})\right)$$

Exploding and vanishing gradient

Why does this happen?

$$\left\| \frac{\partial y^{[t]}}{\partial y^{[t-1]}} \right\| \leq \|R^T\| \left\| \text{diag}\left(f'(Ry^{[t-1]} \right) \right\| \leq \gamma \sigma_{\max}$$

with

- γ a maximal bound for $f'(Ry^{[t-1]})$
  - e.g. $|\tanh'(x)| \leq 1$; $|\sigma'(x)| \leq \frac{1}{4}$
- $\sigma_{\max}$ the largest singluar value of R^T

More details: R. Pascanu, T. Mikolov, Y. Bengio *On the difficulty of training recurrent neural networks* ICML 2013
(and previous work)

Exploding and vanishing gradient

- Vanishing gradient: you have a problem!

Exploding and vanishing gradient

- Vanishing gradient: you have a problem!
- We cannot distinguish if
  - There is no dependency in the data
  - We have chosen the wrong parameters

LSTMs

Intuition

- RNNs blindly pass information from one state to the other
- LSTMs include mechanisms for

Intuition

- RNNs blindly pass information from one state to the other
- LSTMs include mechanisms for
    - Ignoring the input

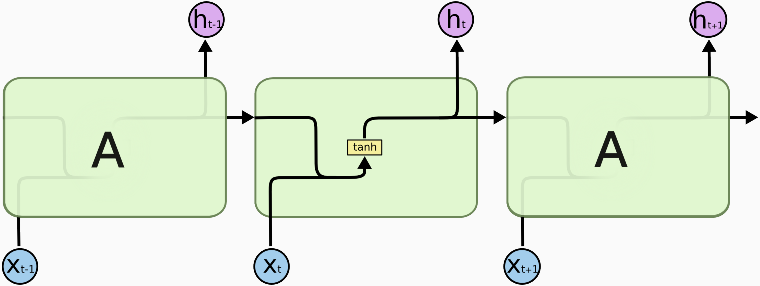
Intuition

- RNNs blindly pass information from one state to the other
- LSTMs include mechanisms for
    - Ignoring the input
    - Ignoring the “current” output

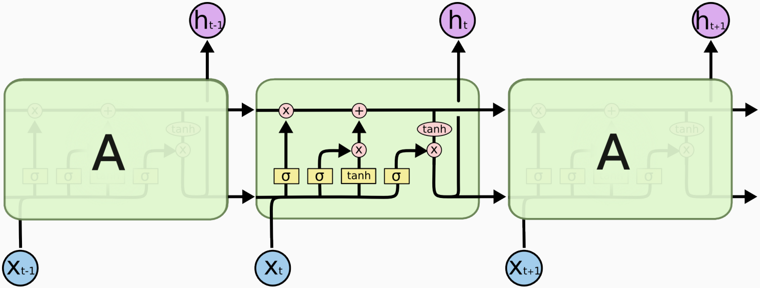
Intuition

- RNNs blindly pass information from one state to the other
- LSTMs include mechanisms for
  - Ignoring the input
  - Ignoring the “current” output
  - Forgetting the history

RNN units



RNN units



LSTM Equations

Compute a “candidate value”, similar to RNNs:

$$\tilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$

LSTM Equations

Compute a "candidate value", similar to RNNs:

$$\tilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$

Input gate: control the influence of the current output

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

LSTM Equations

Compute a "candidate value", similar to RNNs:

$$\tilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$

Input gate: control the influence of the current output

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

Forget gate: control the influence of the history

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

LSTM Equations

Memory cell state: combination of new and old state

$$C_t = i_t \tilde{C}_t + f_t C_{t-1}$$

LSTM Equations

Memory cell state: combination of new and old state

$$C_t = i_t\tilde{C}_t + f_tC_{t-1}$$

Output gate: how much we want to output to the exterior

$$o_t = \sigma(W_ox_t + U_oh_{t-1} + b_o)$$

LSTM Equations

Memory cell state: combination of new and old state

$$C_t = i_t \tilde{C}_t + f_t C_{t-1}$$

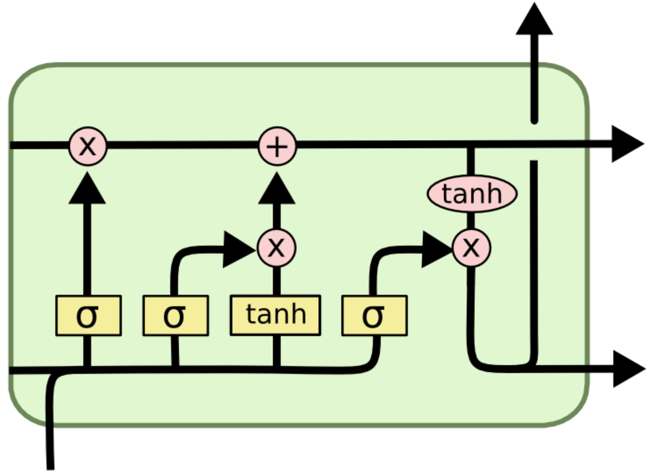
Output gate: how much we want to output to the exterior

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

Output of the cell:

$$y_t = o_t \cdot \tanh(C_t)$$

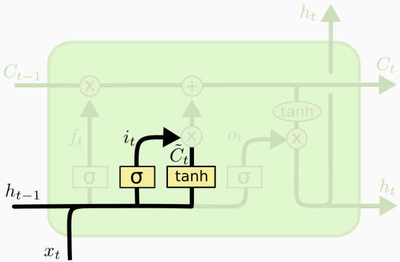
LSTM Visualization



LSTM Visualization

Compute a "candidate value", similar to RNNs

Input gate: control the influence of the current output

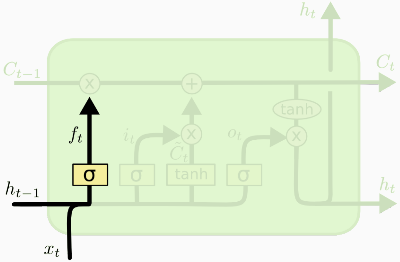


$$\tilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

LSTM Visualization

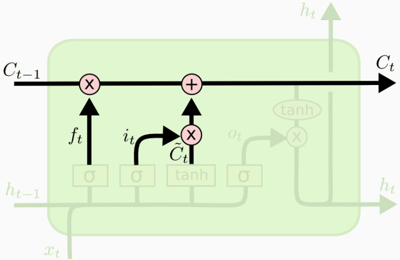
Forget gate: control the influence of the history



$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

LSTM Visualization

Memory cell state: combination of new and old state

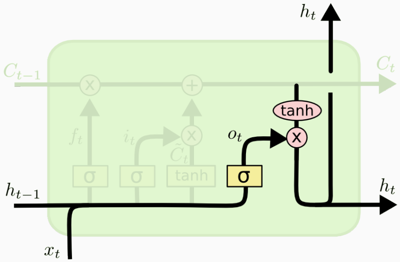


$$C_t = i_t \tilde{C}_t + f_t C_{t-1}$$

LSTM Visualization

Output gate: how much we want to output to the exterior

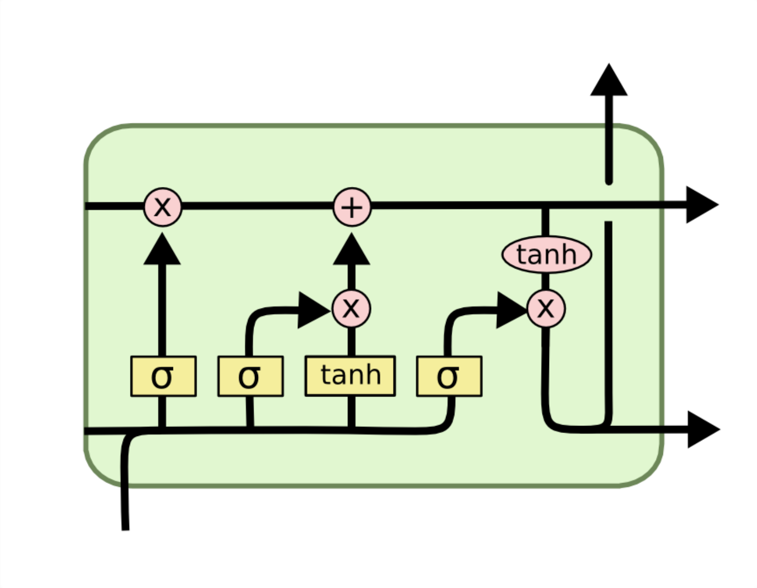
Output of the cell



$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

$$y_t = o_t \cdot \tanh(C_t)$$

LSTM Visualization



LSTMs: additional remarks

- LSTMs solve the vanishing gradient problem, but the gradient can still explode
    - Use gradient clipping

LSTMs: additional remarks

- LSTMs solve the vanishing gradient problem, but the gradient can still explode
  - Use gradient clipping
- Different variants of LSTMs. Basic idea is similar, but
  - Different gates
  - Different parametrization of the gates
  - Pay attention when reading the literature

LSTMs: additional remarks

- LSTMs solve the vanishing gradient problem, but the gradient can still explode
    - Use gradient clipping
- Different variants of LSTMs. Basic idea is similar, but
    - Different gates
    - Different parametrization of the gates
    - Pay attention when reading the literature
- Mathematically: “Constant Error Carousel”
    - No repeated weight application in the derivative
    - “The derivative is the forget gate”

GRUs

Gated Recurrent Units:

- Combine forget and input gates into an “update gate”
- Suppress output gate
- Add a “reset gate”

Simpler than LSTMs (less parameters) and quite succesful

GRUs

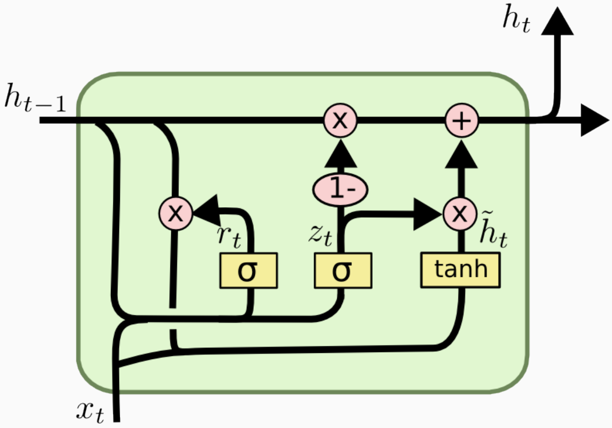
Gated Recurrent Units:

- Combine forget and input gates into an "update gate"
- Suppress output gate
- Add a "reset gate"

Simpler than LSTMs (less parameters) and quite succesful

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$
$$\tilde{h}_t = \tanh(W x_t + U(r_t h_{t-1}) + b)$$
$$h_t = z_t \tilde{h}_t + (1 - z_t h_{t-1})$$

GRUs Visualization



Experimental Results

Results on 1B Word Benchmark

Model	Test PPL
RNN	68.3
Interpolated KN 5-gram, 1.1B N-Grams	67.6
RNN + MaxEnt 9-gram features	51.3
"Small" LSTM	54.1
"Big" LSTM with dropout	32.2
2 Layer LSTM with dropout	30.6

From R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, Y. Wu *Exploring the Limits of Lanugage Modelling*, 2016

A Few Notes About the Output Layer

The Output Layer

Computing a softmax is expensive
(specially for large vocabularies)

The Output Layer

Computing a softmax is expensive
(specially for large vocabularies)

Possible approaches:

- Use a shortlist (and usually combine with standard n-gram model)

The Output Layer

Computing a softmax is expensive
(specially for large vocabularies)

Possible approaches:

- Use a shortlist (and usually combine with standard n-gram model)
- Use hierarchical output

The Output Layer

Computing a softmax is expensive
(specially for large vocabularies)

Possible approaches:

- Use a shortlist (and usually combine with standard n-gram model)
- Use hierarchical output
- Use self-normalizing networks (e.g. NCE training)

References

Word embeddings:

- T. Mikolov, K. Chen, G. Corrado, J. Dean *Efficient Estimation of Word Representations in Vector Space* Workshop at ICLR. 2013
- T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean *Distributed Representations of Words and Phrases and their Compositionality* NIPS. 2013.
- `https://code.google.com/archive/p/word2vec/`

References

Recurrent NNs:

- First reference?
- T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, S. Khudanpur *Recurrent Neural Network Based Language Model* Interspeech. 2010

References

Backpropagation through time:

- From wikipedia: The algorithm was independently derived by numerous researchers
- A. J. Robinson, F. Fallside, *The utility driven dynamic error propagation network* (Technical report). Cambridge University, Engineering Department, 1987
- P. J. Werbos *Generalization of backpropagation with application to a recurrent gas market model* Neural Networks. 1988

References

Vanishing gradient:

- Y. Bengio, P. Simard, P. Frasconi *Learning long-term dependencies with gradient descent is difficult* IEEE Transactions on Neural Networks. 1994
- R. Pascanu, T. Mikolov, Y. Bengio *On the difficulty of training recurrent neural networks* ICML. 2013

References

LSTMs:

- S. Hochreiter, J. Schmidhuber *Long short-term memory* Neural Computation. 1997
- K. Greff, R. K. Srivastava, J. Koutnk, B. R. Steunebrink, J Schmidhuber *LSTM: A Search Space Odyssey* IEEE Transactions on NN and Learning Systems 2015
- Pictures taken from `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

GRUs:

- K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, Y. Bengio *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation* EMNLP 2014

References

Hierarchical Output:

- F. Morin, Y. Bengio *Hierarchical Probabilistic Neural Network Language Models* AISTATS. 2005

NCE:

- A. Mnih, Y. W. Teh *A fast and simple algorithm for training neural probabilistic language models* ICML. 2012

NN Language Models

David Vilar
david.vilar@nuance.com

MT Marathon 2016
14. September 2016