Deep Learning

Thierry Artières

Ecole Centrale Marseille - Option DIGITALE

November 5, 2019

Credit: Nice figures borrowed from many blogs (colah.github.io etc). Other figures are mines...

Outline

1. RNN
   - Tasks and RNN structures
2. LSTM and GRU
3. Recursive models
4. RNNs and NLP
5. Attention

Recurrent NNs

Main features

- May handle data of different dimension w.r.t. traditional FeedForward Models
- Useful for dealing with
  - Sequences : Text (sentiment, translation, parsing...), Speech, Videos, Time series..
  - Trees : Syntactic parse tree etc
- State space models' like architecture
  - Links to state space models

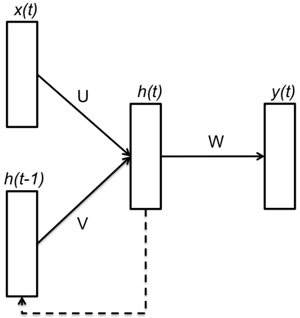
$$s(t) = f(s(t-1), x(t)) \text{ and } y(t) = g(s(t))$$

  - The state at time t resumes the whole history of inputs

Recurrent NNs (RNNs)

RNNs in general

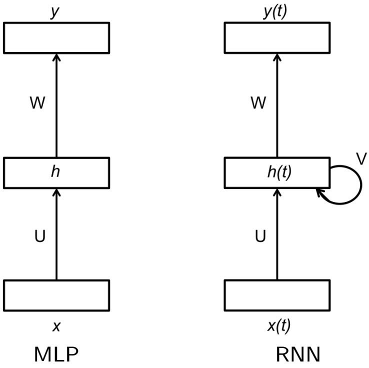
- May handle data of different dimension w.r.t. traditional FeedForward Models (Sequences, trees, ...)
- A recurrent neural network is a NN with cycles in its connections
- Much more powerful than acyclic models (FeedForward NNs such as MLPs)
- Not all architectures work well. Few popular ones.



Feedforward and Recurrent NNs

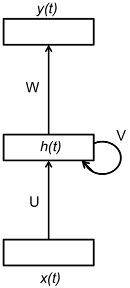
RNNs in general

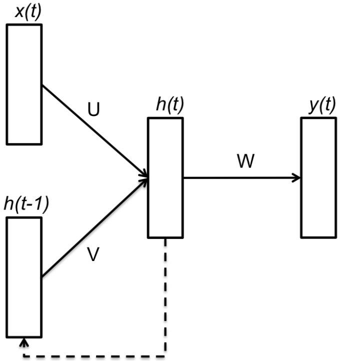
- A recurrent neural network is a NN with cycles in its connections
- RNNs are dynamical systems
- Much more powerful than acyclic models (FeedForward NNs such as MLPs)
- Today RNNs are specific recurrent architectures. Not all architectures work well..



Popular RNNs

Two representations of the same model





Inference

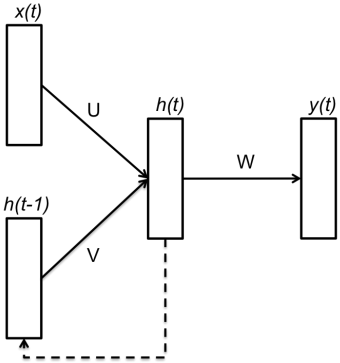
Algorithm

- Start with null state $h(0) = 0$
- Iterate

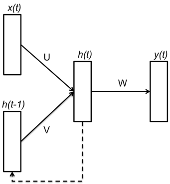
$$h(t) = g(Vh(t-1) + Ux(t))$$
$$y(t) = g(Wh(t))$$

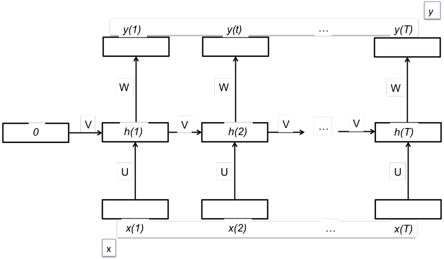
$\Rightarrow$ Inference is done as a forward propagation in a FeedForward NN

- This model computes an output sequence from an input sequence



Inference and Learning through unfolding the RNN



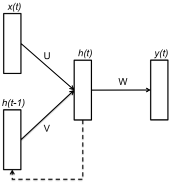


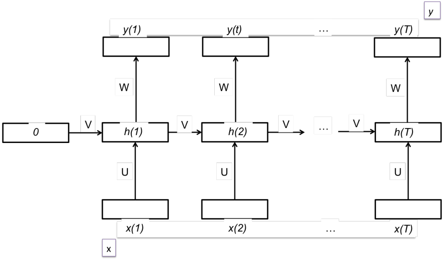
Inference: Forward propagation in the FeedForward unfolded RNN

- Start with null state $h(0) = 0$
- Iterate

$$h(t) = g(V \times h(t-1) + U \times x(t))$$
$$y(t) = g(W \times h(t))$$

Inference and Learning through unfolding the RNN

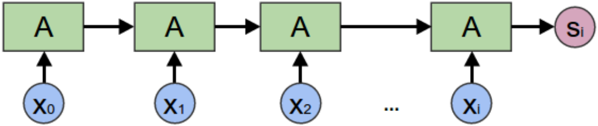




Learning: Back-propagation in the FeedForward unfolded RNN

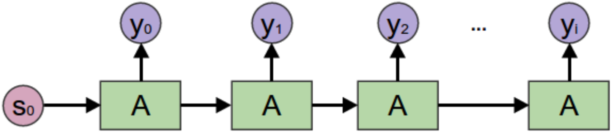
- Unfold the model and perform forward propoagation
- Backpropagate the gradient in the whole network
- Sum the gradient corresponding to all shared parameters and unshared parameters (possibly the last layer)
- Apply Gradient Optimization Update rule on all parameters

Various structures



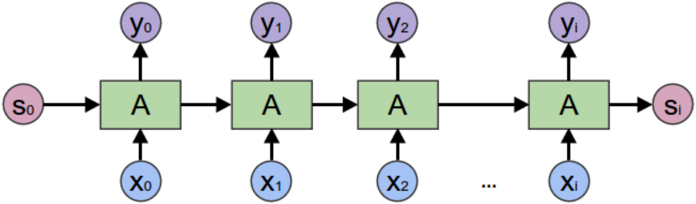
⇒ Encodes a full sequence in a fixed dimensional space

Various structures



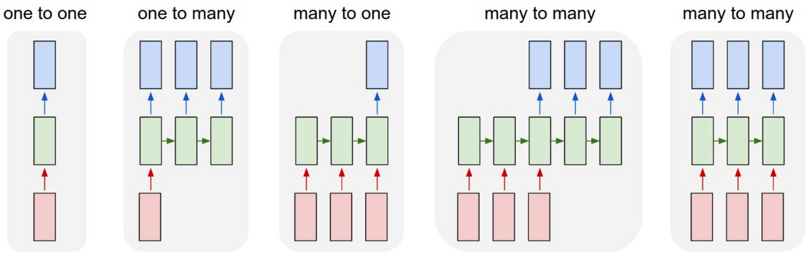
$\Rightarrow$ Produces a full sequence from an initial hidden state

Various structures



$\Rightarrow$ Produces as many outputs as there are inputs

Various settings



- One to One : MLP, CNN ...
- One to Many : Generation of a sequential process (speech, handwriting ...)
- Many to one : Sequence classification (e.g. activity recognition)
- Asynchronous Many to many : Machine Translation
- Synchronous Many to Many : POS tagging, Speech recognition...

A particular case: Sequence Autoencoders

Sequence to Sequence Learning with Neural Networks

Ilya Sutskever
Google
ilyasu@google.com

Oriol Vinyals
Google
vinyals@google.com

Quoc V. Le
Google
qvl@google.com

Main idea

- The first RNN that processes inputs is viewed as an encoder
- And the second RNN that successively produce all outputs is viewed as the decoder
- Same learning strategy as autoencoders: The desired output sequence is the input sequence
- This forces the model to learn to summarize the whole sequence in the last hidden state of the encoder
- Goal of universal representations of sentences, texts etc in a fixed dimensional space

Example: Using RNNs for Language models (LM)

- A LM should allow computing the likelihood of sentences $p(w_1, ..., w_T)$ using a limited number of parameters
- Traditional n-gram language models use n-grams (e.g. bigrams) assuming fixed and limited past dependencies...

$$p\left(w_t | w_{t-1}, w_{t-2}, ...w_{t-n+1}\right)$$

- ... to compute sentence likelihood. E.g. using bigrams:

$$p(w_1, ..., w_T) = p(w_1) \times \prod_{t=2}^{T} p(w_t | w_{t-1})$$

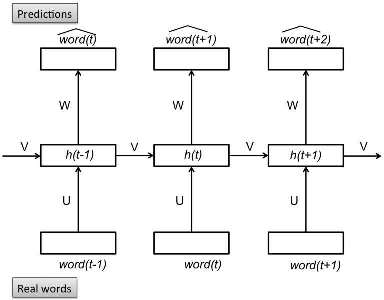
Example: Using RNNs for Language models (LM)

- RNN based Language Models use an, a priori, unlimited past by computing

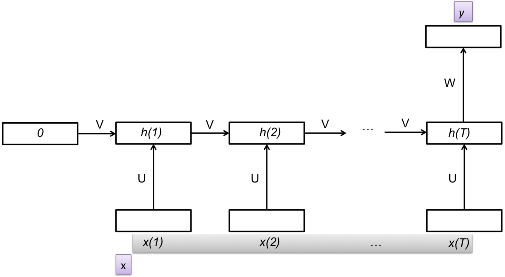
$$p\left(w_t \middle| c\left(w_{t-1}, w_{t-2}, \ldots w_1\right)\right)$$

  where $c\left(w_{t-1}, w_{t-2}, \ldots w_1\right)$ stands for a fixed dimension representation of the context computed from the full past

- This corresponds to a recursive computation of a context information, s_t, and of the computation of an output y_t (w_t) based on the full history.



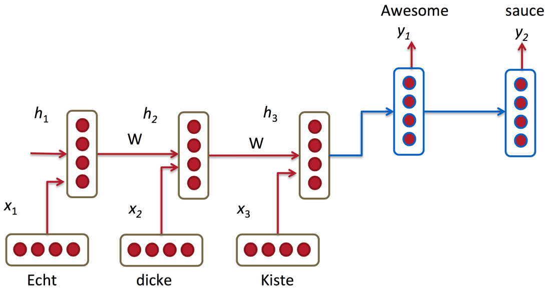
Unfolding the RNN: classification tasks



Inference

- Start : $h(0) = 0$
- For $t = 1$ to T DO : $h(t) = g(V \times h(t-1) + U \times x(t))$
- Predict : $y = g(W \times h(T))$
  $\Rightarrow$ The final state $h(T)$ resumes the whole input

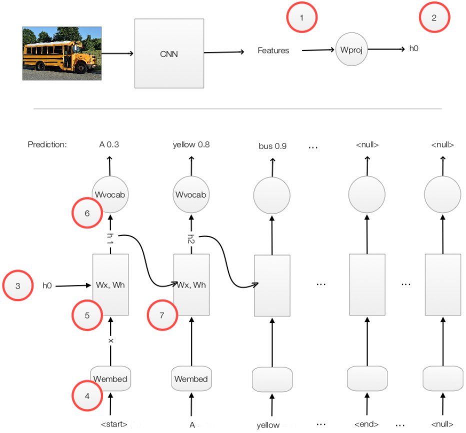
Machine Translation



Encoder Decoder structure

- Example of a translation model as a asynchronous Many to Many model
- The nature of language and of complex grammatical forms require to first "understand" the sentence, encoding it in a small dimensional hidden space, then to reconstruct the sentence in the target language.

Image captioning with RNNs

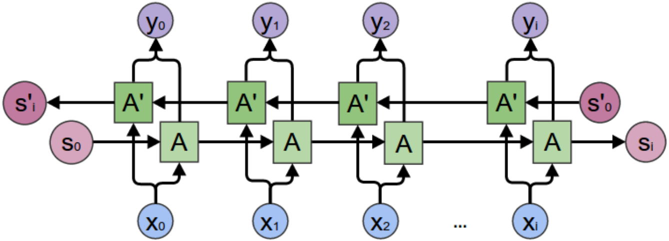


Input comes from the caption in the training dataset (true caption).

Bidirectional models

Basic idea: go beyond forward computation in time

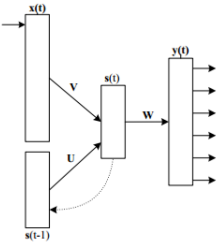
- The output sequence's items might be dependent on the whole input sequence
- Stack tow RNNs that go in reverse directions to take into account past and future dependencies

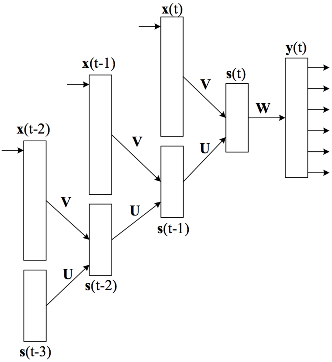


Learning RNNs: Challenges

Problems

- Vanishing gradient
- Long term dependencies





Outline

Depth in RNNs

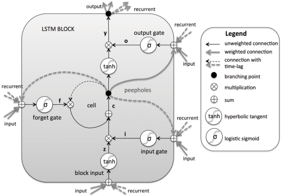
Two dimensions

- Stacked hidden layers as in traditional deep NNs : usual in many arhcitectures
- Long sequences $\rightarrow$ deep in time
- Both structural depths yield similar optimization problems (gradient flow)

New units for RNNs

- Motivation:
  - Optimization problems in Recurrent Neural Networks (gradient explosion / vanishing)
  - Difficulty to capture long term dependencies
- New types of hidden cells
  - Long Short Term Memory (LSTM) [Hochreichetr 98]
  - Gated Recurrent Unit (GRU) [Cho and al., 2014]

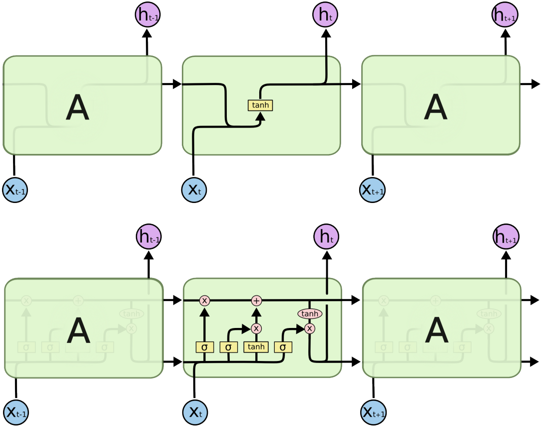
LSTM units



Motivation

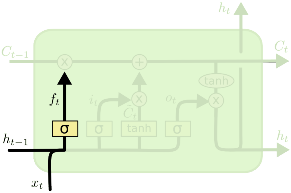
- Units that include few gates (*forget*, *input*, *output*) which allow to :
  - Stop capitilizing in the state the information about the past
  - Decide if it is worth using the information in the new input
- Depending on the input and on previous state
  - Reset the state, Update the state, Copy previous state
  - Ignore new input or fully use it to compute a new state

LSTM units

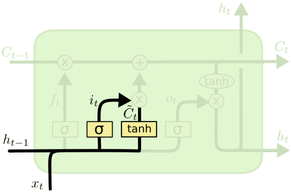


- LSTM layers may be stacked as well as standard RNN layers ($h_t = LSTM(x_t, h_{t-1}, c_{t-1})$ is input to the upper layer)

LSTM units



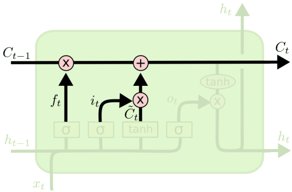
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



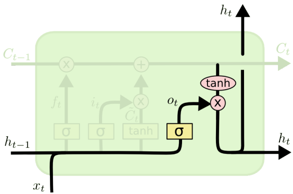
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

- Forget gate f_t
- Input gate i_t
- Alternative cell state $\tilde{c}_t$

LSTM units



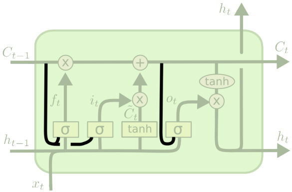
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

- Cell state c_t
- Output o_t
- Hidden state to propagate h_t

LSTM units



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

Filters that alter the computed intermediate representations

- f_t : Do we forget the context or not ?
- i_t : Do we consider the input or not ?
- o_t : How to weight the different outputs ?

LSTM units

Notations

- Cell state c_t
- Forget gate f_t
- Input gate i_t
- Output o_t
- Hidden state to propagate to upper layers h_t

Full equations

$$
\begin{aligned}
f_t &= \sigma(W_f h_{t-1} + U_f x_t) \\
i_t &= \sigma(W_i h_{t-1} + U_i x_t) \\
o_t &= \sigma(W_o h_{t-1} + U_o x_t) \\
\tilde{c}_t &= tanh(W_c h_{t-1} + U_c x_t) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
h_t &= o_t \odot tanh(c_t)
\end{aligned}
$$

LSTM units

Notations

- Cell state c_t
- Forget gate f_t
- Input gate i_t
- Output o_t
- Hidden state to propagate to upper layers h_t

How does it work ? in words...

- Recall formulas

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$h_t = o_t \odot tanh(c_t)$$

- Interpretation
    - If $f_t == 1$ and $i_t == 0$ use previous cell state
    - If $f_t == 0$ and $i_t == 1$ ignore previous cell sate
    - If $o_t == 1$ output ois set to cell sate
    - If $o_t == 0$ output is set to 0

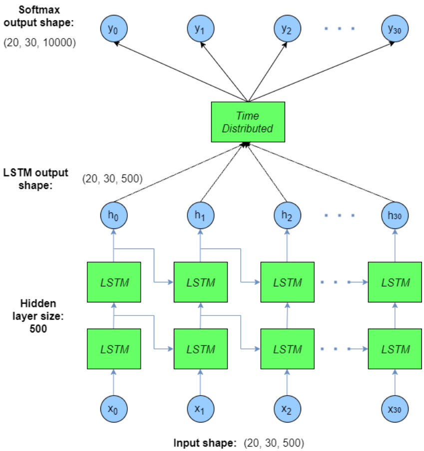
GRU units

Motivation: Simplify LSTMs

- Replace LSTMs with the same underlying idea
- Equations

$$\begin{aligned}
Gate_{reset} &= G_r = \sigma(W_{rh}h_{t-1} + W_{rx}x_t) \\
Gate_{update} &= G_u = \sigma(W_{uh}h_{t-1} + W_{ux}x_t) \\
\tilde{h}_t &= tanh(G_r \odot W_{hh}(h_{t-1}) + W_{hx}x_t) \\
h_t &= (1 - G_u) \odot h_{t-1} + G_u \odot \tilde{h}_t
\end{aligned}$$

Recurrent architecture example

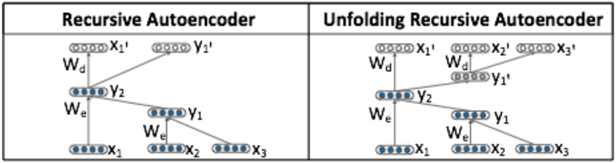


Outline

Dealing with structured data

Principle

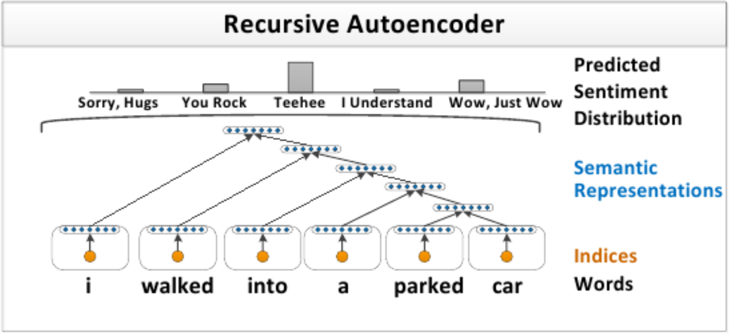
- Allows dealing with other structured data such as trees
- The model may still be unfolded and gradient may easily be computed



Dealing with structured data

Principle

- Allows dealing with other structured data such as trees
- The model may still be unfolded and gradient may easily be computed



Outline

Language models

- A LM should allow computing the likelihood of sentences $p(w_1, ..., w_T)$ using a limited number of parameters
- Traditional n-gram language models use n-grams (e.g. bigrams) assuming fixed and limited past dependencies...

$$p\left(w_t | w_{t-1}, w_{t-2}, \ldots w_{t-n+1}\right)$$

- ... to compute sentence likelihood. E.g. using bigrams:

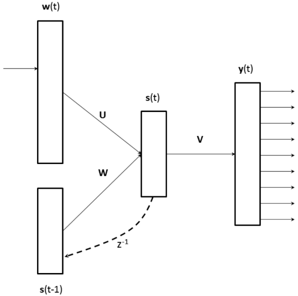
$$p(w_1, ..., w_T) = p(w_1) \times \prod_{t=2}^{T} p(w_t | w_{t-1})$$

Going beyond ngrams

Motivation pour les représentations continues (dans les modèles de langage)

- Modèle de langages type Ngrams : Pas de notion de similarité entre mots / nécessité de corpus de taille gigantesque pour une bonne estimation (qui n'existent pas)
- Représentation distribuée de mots $\rightarrow$ dépasse les limitations des Ngrams car partage de l'information entre mots :
  - Réponse similaire du système pour une entrée similaire
  - Taille de corpus nécessaire gigantesque mais suffisante pour apprendre ce que l'on veut

Modèle de langage type RN récurrent [Mikolov 2013]



- $W(t)$: Codage 1 parmi N
- $y(t)$: Distrib de proba sur Vocab

$$s(t) = f(Uw(t) + Ws(t-1)$$
$$y(t) = g(Vs(t-1))$$

- f : sigmoide ; g : softmax

$\Rightarrow$ Les représentations des mots sont les colonnes de U
$\Rightarrow$ Capacité intéressante des représentations de mots
Une relation particulière entre deux mots (syntaxique pluriel, féminin, ou sémantique) correspond à un déplacement constant dans l'espace des représentations

Embedding layer for text representation

Motivation : Transformation layer for discrete/categorical inputs

- Example : a Word in a Dictionary (Natural Language Processing tasks)
- Embedding : distributed representation. Not a new idea (LSA, LDA)

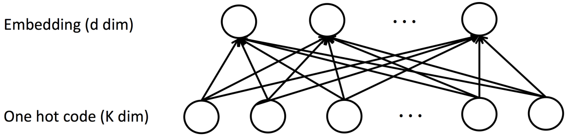
Main interests

- When the cardinality of the input is (very) large (e.g. NLP tasks) to allow accurate estimation from tractable corpus
- When one wants to infer some continuous representations of the input values to get insight on similarities between them

Embedding layer: Implementation

Look up table

- One entry for each of the possible values $\{v_1, ..., v_K\}$ (e.g.words in a dictionary)
- Each value is represented as a $d-$dimensional vector (d is the size of the embedding)
- Represented as a layer with a weight matrix $(K \times d)$

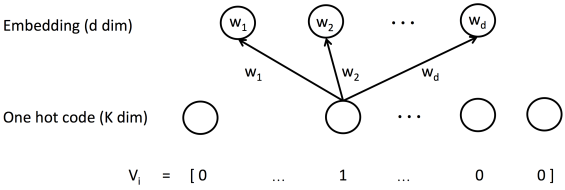


$$
\begin{aligned}
V_1 &= [1 \quad 0 \quad 0 \quad \dots \quad 0 \quad 0] \\
V_2 &= [0 \quad 1 \quad 0 \quad \dots \quad 0 \quad 0] \\
&\cdots \\
V_K &= [0 \quad 0 \quad 0 \quad \dots \quad 0 \quad 1]
\end{aligned}
$$

Embedding layer: Implementation

Look up table

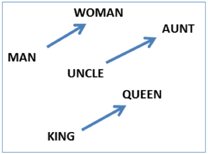
- One entry for each of the possible values $\{v_1, ..., v_K\}$ (e.g.words in a dictionary)
- Each value is represented as a $d-$dimensional vector (d is the size of the embedding)
- Represented as a layer with a weight matrix ($K \times d$)

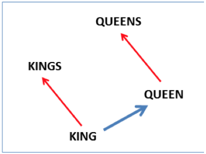


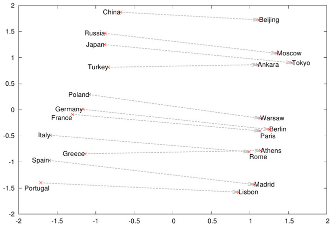
A particular interesting effect: compositionality

Idea

- $Emb('King') + Emb('Woman') - Emb('Man') \approx Emb('Queen')$
- It is an observed phenomenon which is not actually favored by the model design the learning criterion
- Similar effect reported on images (with DCGAN from [Radford et al.])







Outline

Need for reasoning

Main idea

- Based on a query formulate a goal (information to get)
- Loop
  - Recover the information in the memory that matches the current query
  - Based on gained information and on the original query, formulate another query
- Take a decision based on the accumulated information

Joe went to the kitchen. Fred went to the kitchen. Joe picked up the milk.
Joe travelled to the office. Joe left the milk. Joe went to the bathroom.
Where is the milk now? A: office
Where is Joe? A: bathroom
Where was Joe before the office? A: kitchen

Attention mechanism

Main idea

- Most relevant element to a query?
  - Given number of memory elements u_i
  - For a (current) query q
  - Assuming queries and memory elements live un the same space
- The most relevant memory element to query q is the most similar one

$$u* = argmax_i \langle u_i, q \rangle$$

- Use a smooth **argmaximum**

$$s_i = argmax_i \langle u_i, q \rangle$$
$$\alpha_i = softmax(s_i)$$
$$u* = \sum_i \alpha_i u_i$$

Attention mechanism

Implementation in Keras

```
inputs = Input(shape=(input_dims,))
attention_probs = Dense(input_dims, activation='softmax', name='attention_probs')(inputs)
attention_mul = merge([inputs, attention_probs], output_shape=32, name='attention_mul', mode='mul')
```

Simple reasoning and baby stories

Memory Networks [Weston and al., 2015]

- Include a long-term memory that can be read and written to with the goal of using it for prediction: kind of knowledge base
- More straightforward use of the memory than in RNNs
- Ability to deal with complex question answering

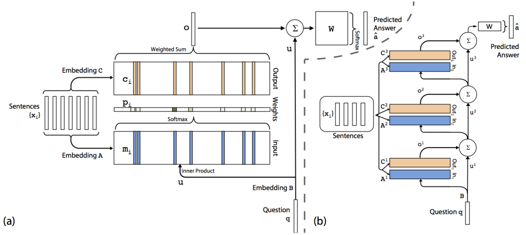


Figure 1: (a): A single layer version of our model. (b): A three layer version of our model. In practice, we can constrain several of the embedding matrices to be the same (see Section 2.2).

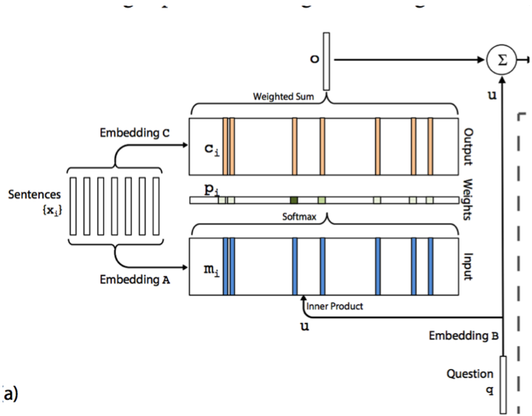
End to End Memory Networks [Sukhbaatar and al., 2015]

Joe went to the kitchen. Fred went to the kitchen. Joe picked up the milk.
Joe travelled to the office. Joe left the milk. Joe went to the bathroom.
Where is the milk now? A: office
Where is Joe? A: bathroom
Where was Joe before the office? A: kitchen

Simple reasoning and baby stories

Memory Networks [Weston and al., 2015]

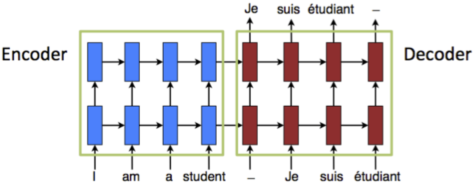
- Zoom on a single hop



Attention mechanisms

Machine translation

- Instead of standard Seq2Seq models
- One may want to focus on one part of input sequence for producing one output word
- Attention = (fuzzy) focus on the input
- Same kind of ideas for automatic captioning

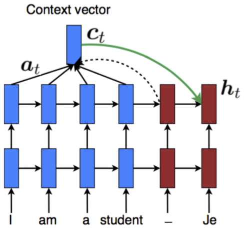


[Bahdanau and al., 2015]

Attention mechanisms

Machine translation

- Instead of standard Seq2Seq models
- One may want to focus on one part of input sequence for producing one output word
- Attention = (fuzzy) focus on the input
- Same kind of ideas for automatic captioning



[Bahdanau and al., 2015]

Attention mechanisms

Machine translation

- Instead of standard Seq2Seq models
- One may want to focus on one part of input sequence for producing one output word
- Attention = (fuzzy) focus on the input
- Same kind of ideas for automatic captioning

A woman is throwing a frisbee in a park. A dog is standing on a hardwood floor. A stop sign is on a road with a mountain in the background.

[Xu et al., 2016]

Attention for translation

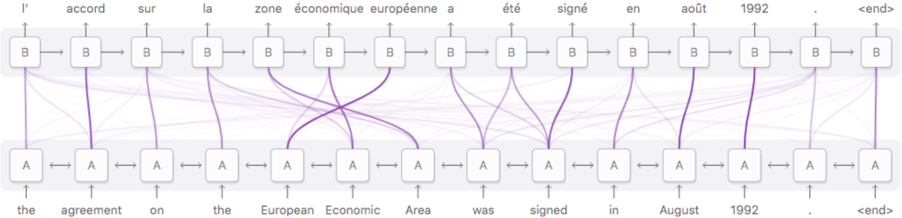


Diagram derived from Fig. 3 of Bahdanau, *et al.* 2014

Attention for speech recognition

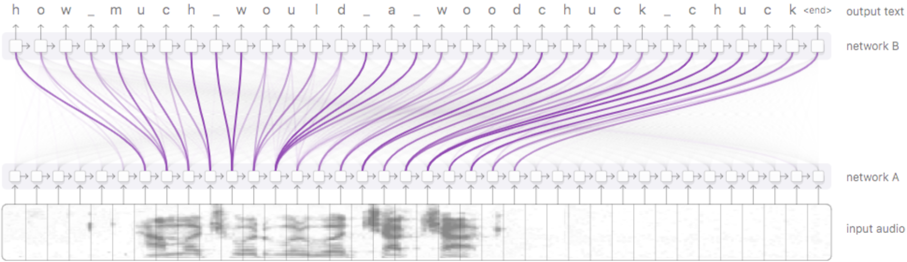


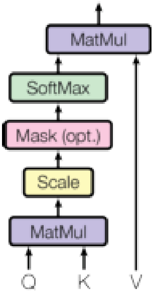
Figure derived from Chan, *et al.* 2015

Self Attention

Main Idea

- Compute new representations of inputs elements in a sequence based on the whole sequence
- $\Rightarrow$ representations of elements in context

Scaled Dot-Product Attention



- Propagation in the attention layer

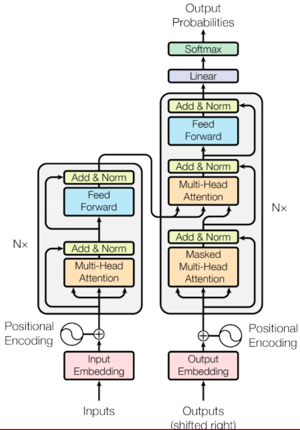
$$Attention(Q, K, V) = softmax\left(\frac{Qk^T}{\sqrt{d}}\right) V$$

- with:
  - Q: queries (with dimension d)
  - K: Keys
  - V: Values
- where Q, K, V could be all equal to the inputs
- But there are transformed (by product with weight matrices) of the inputs

Self Attention

Main Idea

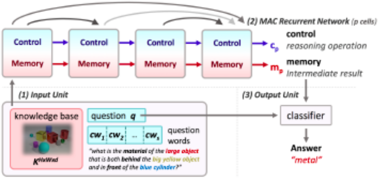
- Compute new representations of inputs elements in a sequence based on the whole sequence
- $\Rightarrow$ representations of elements in context



General reasoning

More complex reasonning tasks

- Requires few steps of question answering like queries



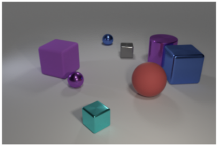


Figure 1: A sample image from the CLEVR dataset, with a question: "There is a purple cube behind a metal object left to a large ball; what material is it?"