Neural Networks for Language Processing

Thinking about language (and time series)

Yordan Darakchiev

Technical Trainer

iordan93@gmail.com

sli.do

#DeepLearning

Table of Contents

Time-Dependent Models

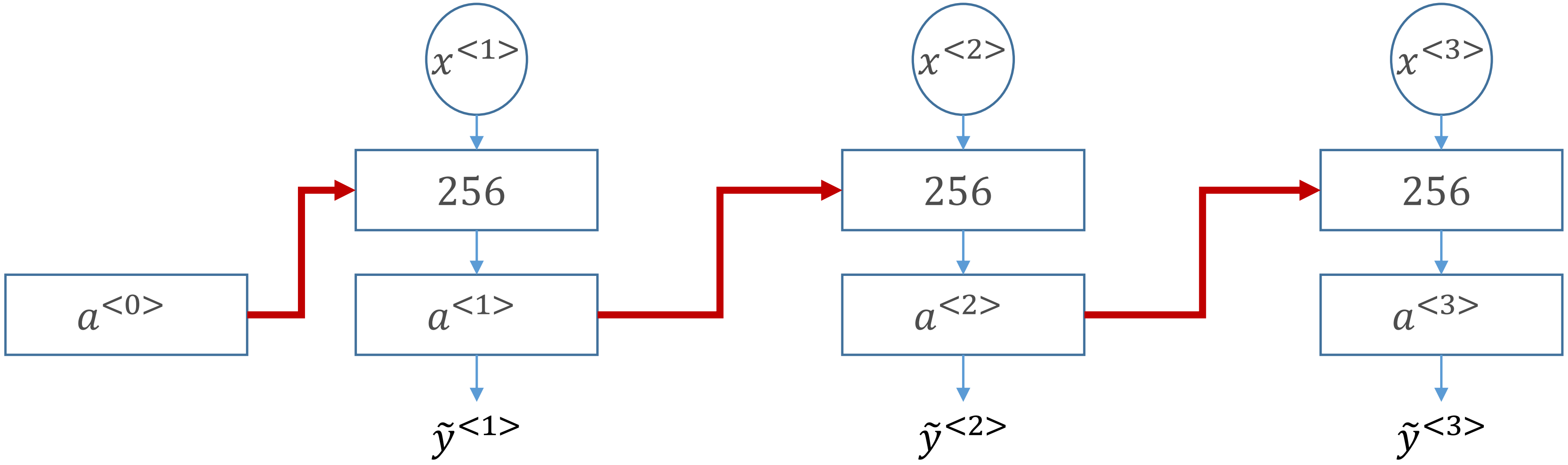
"Time is precious, waste it wisely"

Time-Dependent Model Examples

- Speech recognition: audio → transcript
- Machine translation: text (EN) → text (FR)
- Activity recognition: video → activity type (e.g. walking)
- Sentiment analysis: text → sentiment
- Generation
  - Text summarization
  - Music generation
- More generally, models whose inputs depend on time
  - "Standard" models: $\tilde{y} = f(x)$; recurrent models: $\tilde{y} = f(x, s)$
    - s – current state
  - Standard models don't allow variable-length inputs
  - Most standard models don't allow for weight sharing

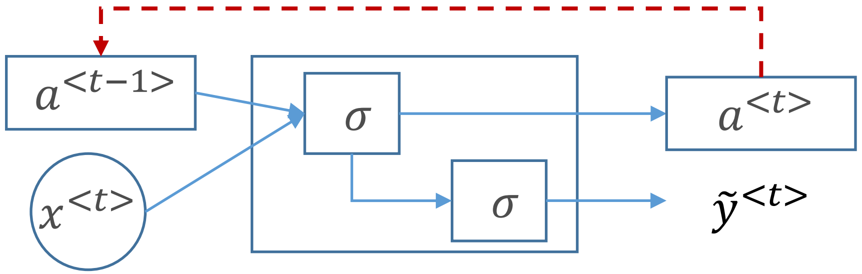
Working with Sequences

- Training example: x = "A black cat in a box"
  - Split words (tokenize the input)
  - Present words as 1-hot encoded vectors using a dictionary (vocabulary)
  - $x^{<1>} = \text{"a"} = [1\ 0\ \ldots 0]^T \equiv V_1$
  - $x^{<2>} = \text{"black"} = [0\ 0\ \ldots 1 \ldots 0]^T \equiv V_{329}$, etc.
- Take a standard model (1-layer NN), pass each word

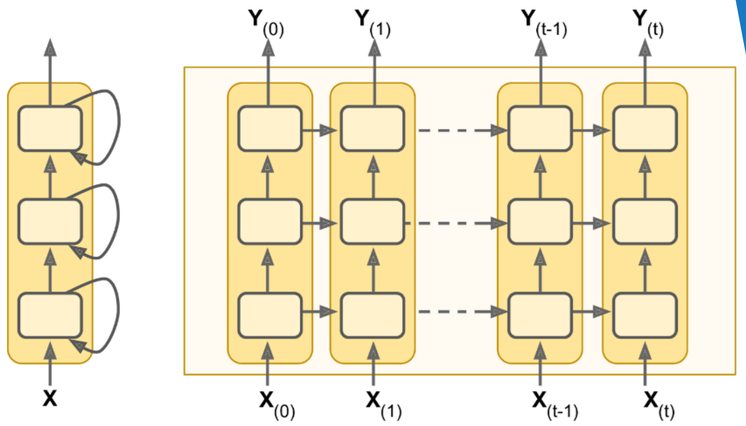


Recurrent Neural Networks

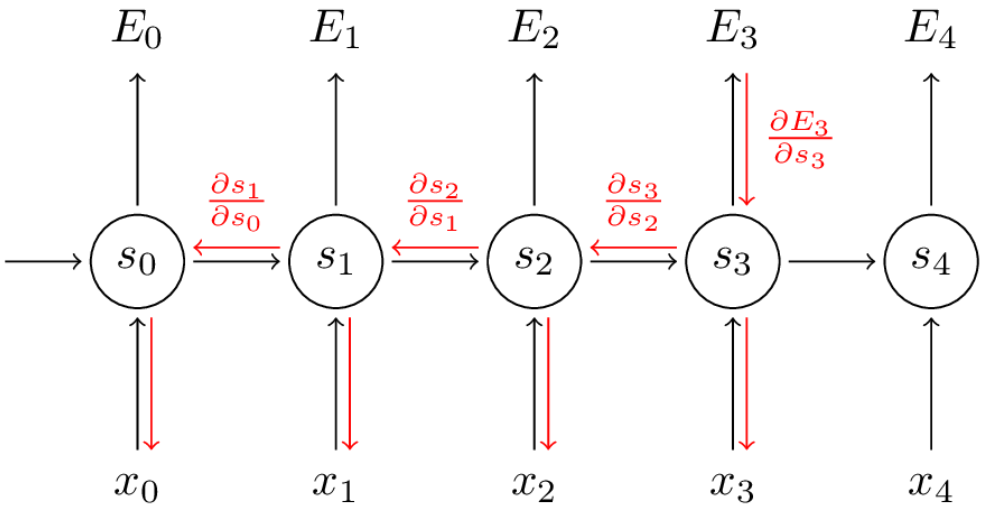
- "RNN cell"



- Deep architectures are also possible

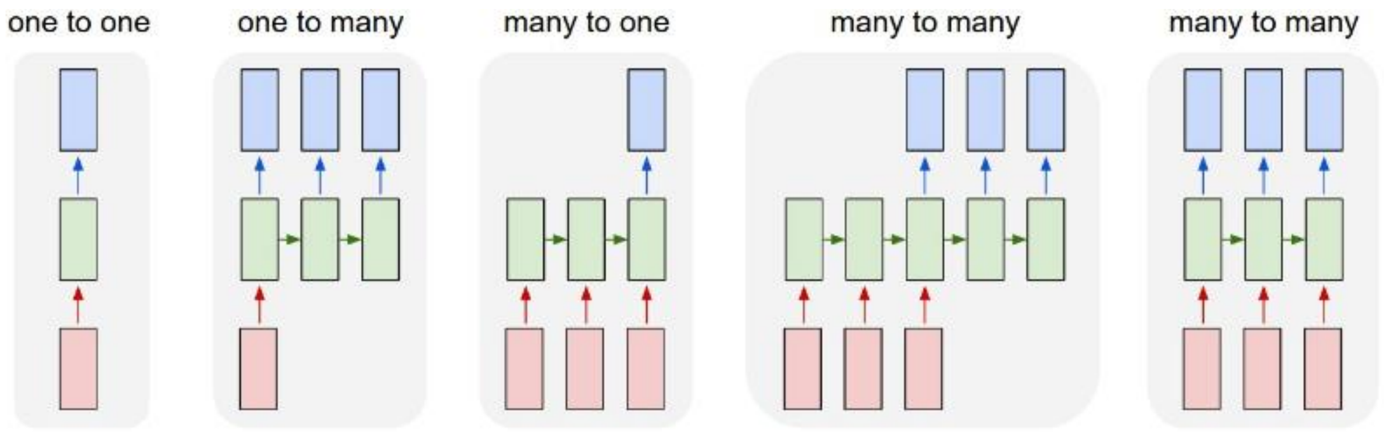


- Learning: backpropagation through time
  - The same as in a multi-layer network: $\arg\min J(y^{<t>}, \tilde{y}^{<t>})$



RNN Architectures

- One to one: standard
- One to many
  - Sequence generation given seed (e.g., image captioning)
- Many to one
  - One output for sequence (e.g., sentiment analysis)
- Many to many
  - Encoding and decoding (e.g., machine translation)
  - Synchronized output (e.g., video classification for each frame)



Language Model

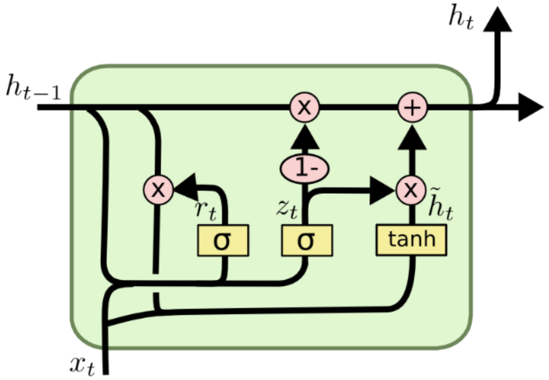
- Training
  - Tokenize the input $x = [x^{<1>}; x^{<2>}; x^{<T_x>}]$
  - Use a standard RNN, with no initial seed
    - $a^{<0>} = [0\ 0\ \dots\ 0] = \vec{0}, x^{<0>} = \vec{0}$
    - Output: $\tilde{y}$: a vector of probabilities for each word $[0{,}0385\ \ 0{,}0476\ \ \dots 0{,}00041]$
      - Softmax, with 10 000 outputs
  - Explanation
    - First token: $\tilde{y} = P(w_1)$
    - Second token: $\tilde{y} = P(w_2|w_1)$
    - In general: $\tilde{y} = P(w_k|w_1, w_2, \dots, w_{k-1})$
- Generation: random sampling according to computed P
  - Input $x^{<0>} = \vec{0}, a^{<0>} = \vec{0}$; compute $a^{<1>}, \tilde{y}^{<1>}$; choose a word w_1
  - Input $x^{<1>} \equiv w_1, a^{<1>}$; compute $a^{<2>}, \tilde{y}^{<2>}$; choose a word w_2
  - … until you reach [.]

Improved Models

Making things even more difficult

Vanishing Gradients

- RNN with a long input is similar to a very deep NN
- Examples
  - The match was long, but we won it which made us happy.
  - We **decided to go** **to the movies**, but our friend, who doesn't like scary movies, didn't want **to go**.
- Solution: Gated recurrent unit (GRU) – Cho et al., 2014
  - Update gate (z_t): how much of the past information to retain
  - Reset gate (r_t): how much information to forget
  - Final memory: current information + previous "context"



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

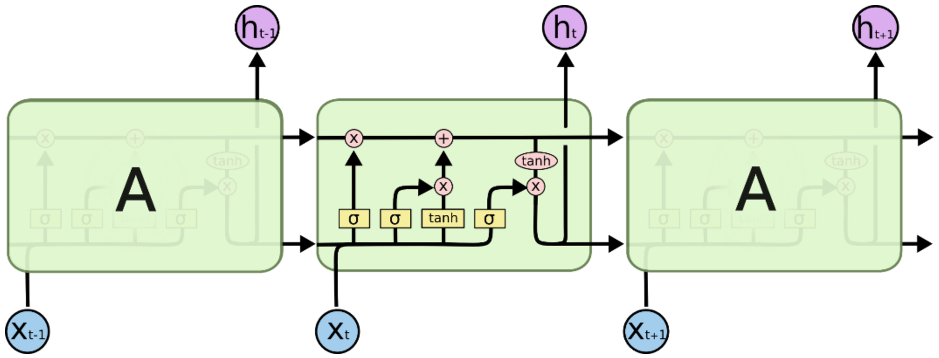
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

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

Long-Short Term Memory (LSTM)

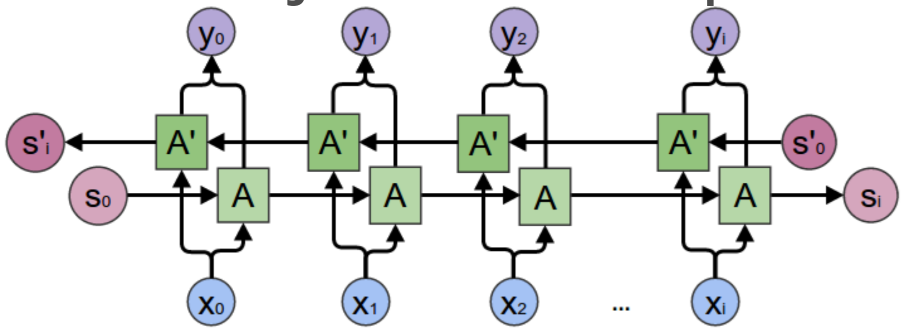
- Even more powerful (and complicated)
  - [Hochreiter and Schmidthuber, 1997]()
  - This is only one layer, LSTM layers can also be stacked



- Basic parts of the architecture
  - Forget gate f_t
  - Update gate i_t
  - Cell state C_t
  - Output o_t
- A good [article]() explaining LSTM cells

Bi-Directional Networks

- Intuition
  - RNNs may need information ahead, "from the future"
  - E.g. to translate word $w^{<5>}$ we may need the whole sentence
- Solution: just create pairs of networks



  - These can be RNN, GRU, LSTM or other layers
  - To compute activations, go left to right, then right to left

Representing Words

Find your way
in the multi-dimensional space

Word Representation

- Basic idea: one-hot encoding
- How to get insights on word relations?
  - Try to estimate word features: vectors of numbers for each word
    - Unsupervised process
    - **Embedding** from a space with one dimension per word to a lower-dimensional space (e.g. 300D)
  - Example uses
    - Use similarity measures (e.g. cosine distance) between vectors
    - Use projections to generate analogies (Mikolov et al., 2013)
- Visualization: usually t-SNE or PCA
- Tensorboard uses Google's Embedding Projector
  - https://www.tensorflow.org/guide/embedding

Word2Vec and GloVe

- What we already described
  - A matrix E where each vocabulary word has a dense vector
- Context-target word pairs
  - Compute vectors for context and target
  - Loss: cross-entropy
- Similarity
  - Cosine similarity; closest words to "Sweden"
- Associations
  - `Rome : Italy :: Beijing : China`
  - `king : queen :: man : woman`
  - Other examples

Word	Cosine distance
norway	0.760124
denmark	0.715460
finland	0.620022
switzerland	0.588132
belgium	0.585835
netherlands	0.574631
iceland	0.562368
estonia	0.547621
slovenia	0.531408

Refinement Algorithms

Some more tricks up our sleeves

Beam Search

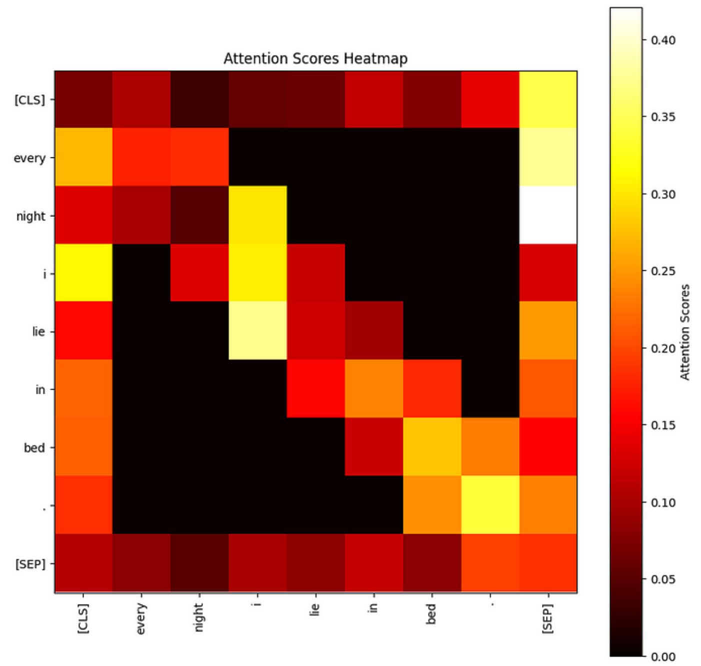
- Translation
  - Similar to generation, $\tilde{y} = f(x)$, maximize $P(\tilde{y}|x)$
- What if we have multiple candidates?
  - Use the language model to compute P
- Slight complications
  - *I am visiting NY this year end.*
  - *I am going to be visiting NY this year end.*
  - $P(\text{going}|\text{i, am}) > P(\text{visiting}|\text{i, am})$
  - Observations
    - One word at a time doesn't work too well
    - All words will require enormous computation power
  - Solution: Beam search
    - At each step, choose top B words (beam width)
    - More details

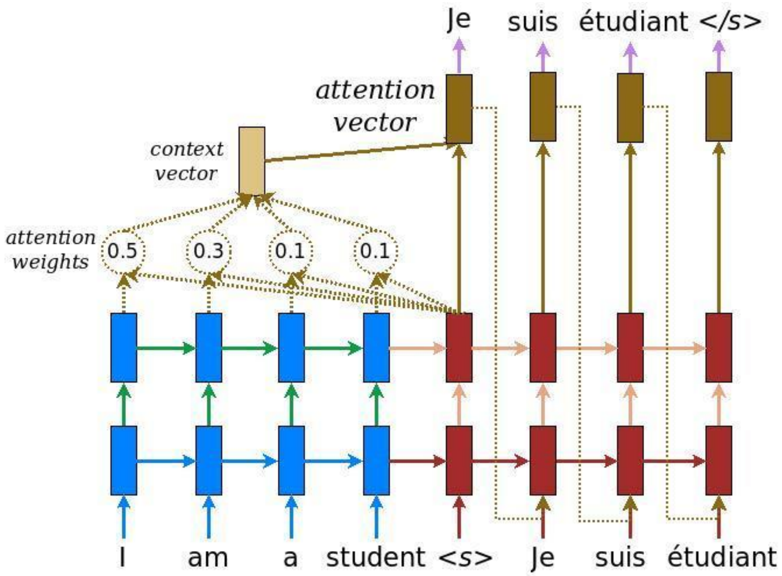
Attention

- Xu et al., 2015
- Another mechanism for dealing with complicated inputs
  - Another caveat: longer sentences have inherently lower probabilities so models tend to favor short sentences
  - Intuition: we don't need to know the entire sequence in order to be able to translate
- Idea
  - Use a bi-directional RNN (or GRU / LSTM)
  - For each part of the input $x^{<t>}$, compute "how much you care" about it: $attention^{<t>}$
- Usages
  - Translation, image captioning, speech recognition, text summarization, etc.

Attention Explained

- Queries, keys, values
  - Linear (dense) layers
- Multi-head attention
  - Just do the same operation multiple times ☺
- Sparse attention
  - Don't keep the whole attention table
- Scaled dot product attention
  - $A(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$
  - d_k - dimension of q and k
  - Outputs have variance = 1





Transformers

- Brown et al., 2020 (GPT-3)
- Main points
  - Positional encoding
  - Attention blocks (heads) / self-attention
  - Encoder / decoder structure
- Usages
  - Language models, question answering, classification, paraphrasing / summarization, etc.
- Open issues
  - Attention blocks require too much memory
  - Too long training time

Summary

- Time-dependent (sequential) models
  - Architecture
  - Types
- Improvements
- Word (token) representations
- Refinement algorithms
  - Attention
  - Transformers

Questions?