Backpropagation, Word embeddings, Recurrent Neural Networks

Data Mining in Action: Trends

Plan

1. Backpropagation algorithm for training
2. Word embeddings as a basis for doing Deep NLP
3. RNNs

Backpropagation

- Generally, it is just a chain rule on the computational graph
- For details we follow these links:
  - https://goo.gl/LbEHy4
  - https://goo.gl/maQ8KS
  - =)

Word Embeddings

based on awesome CS224d Stanford course
http://cs224d.stanford.edu/lecture_notes/notes1.pdf
http://cs224d.stanford.edu/lectures/CS224d-Lecture2.pdf
http://cs224d.stanford.edu/lectures/CS224d-Lecture3.pdf

Word Embeddings

- **Question:** How can we represent word as a vector?
- **Some ideas:**
    1. One-hot
    2. Construct word-word cooccurrence matrix
    3. Variations of (2): word-word -> word-doc, word-window
- **Problems**
    - The (1) method does not preserve similarity properties between words.

      motel [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0] AND
      hotel [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0] = 0
    - For (2) and (3) we need to compute & store VERY large matrices (usually you have 10-100k of words) and the VECTOR DIMENSIONALITY is increases with increasing the vocabulary.

Word Embeddings

- Idea: store "most" of the important information in a fixed, small number of dimensions: a dense vector.
- How to do this?
- **Idea:**
  - Perform SVD of word-word cooccurrence matrix.
- **Problem:**
  - Quadratic cost to perform SVD + all problems with word-word matrix.

Word Embeddings

- Word2Vec (CBOW, Continuous Bag of Words)
- Predict surrounding words in a window of length m of every word.
- Objective function: Maximize the log probability of any context word given the current center word:
- $$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log p(w_{t+j} | w_t)$$
- Where θ represents all variables we optimize

Word Embeddings

- Predict surrounding words in a window of length m of every word
- For $p(w_{t+j}|w_t)$ the simplest first formulation is

$$p(o|c) = \frac{\exp\left(u_o^T v_c\right)}{\sum_{w=1}^{W} \exp\left(u_w^T v_c\right)}$$

- where o is the outside (or output) word id, c is the center word id, u and v are “center” and “outside” vectors of o and c
- Every word has two vectors!

Word Embeddings

These representations are *very good* at encoding dimensions of similarity!

- Analogies testing dimensions of similarity can be solved quite well just by doing vector subtraction in the embedding space

  Syntactically

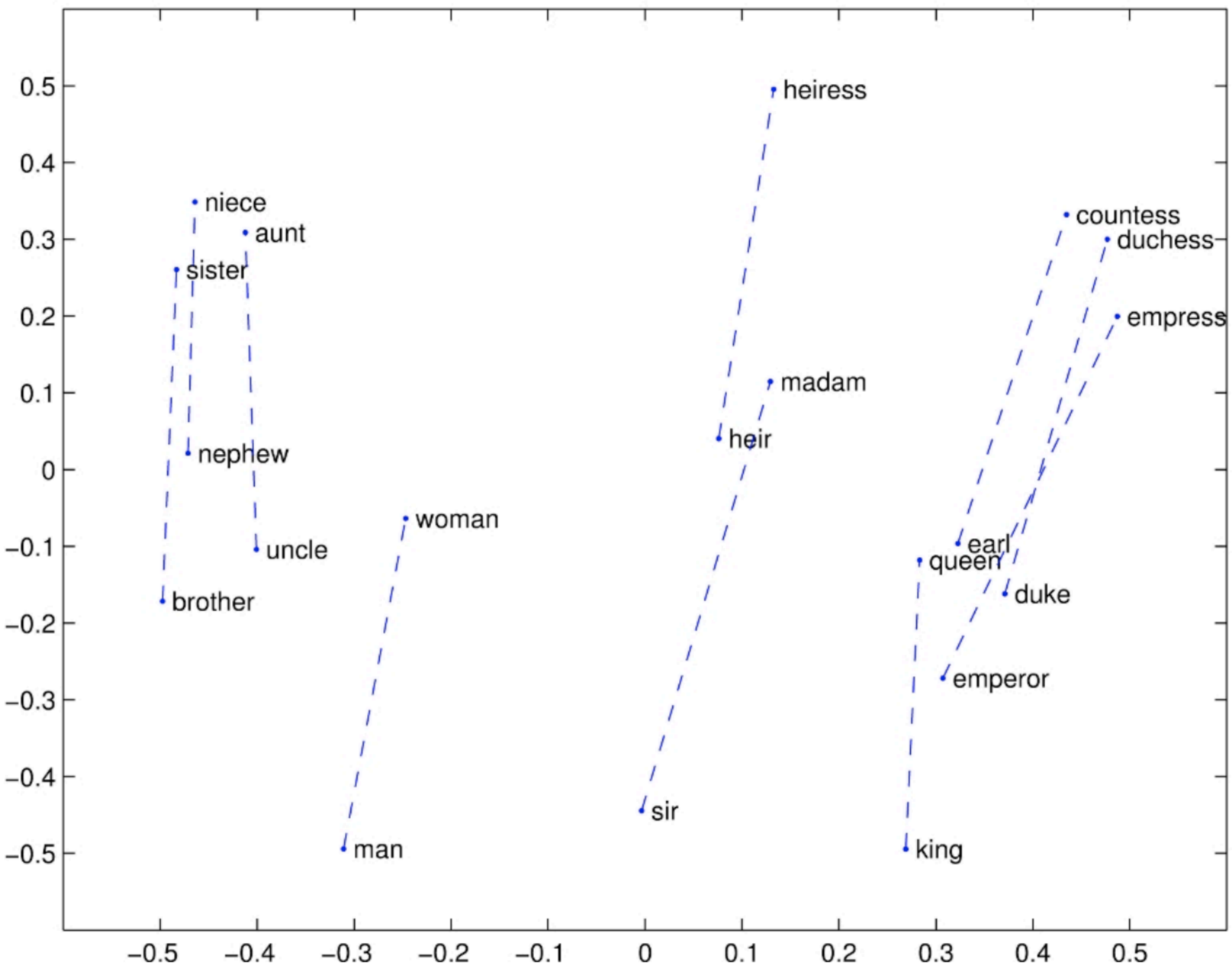
  - $x_{apple} - x_{apples} \approx x_{car} - x_{cars} \approx x_{family} - x_{families}$

  - Similarly for verb and adjective morphological forms

  Semantically (Semeval 2012 task 2)

  - $x_{shirt} - x_{clothing} \approx x_{chair} - x_{furniture}$
  - $x_{king} - x_{man} \approx x_{queen} - x_{woman}$

Word Embeddings



Word Embeddings

- Other methods with the same idea:
  - word2vec (skip-gram), doc2vec, GloVe, AdaGram, lda2vec…
- Implementations:
  - gensim (has implementations of w2v, d2v, glove) https://radimrehurek.com/gensim/
  - AdaGram (https://github.com/sbos/AdaGram.jl)

Recurrent Neural Networks

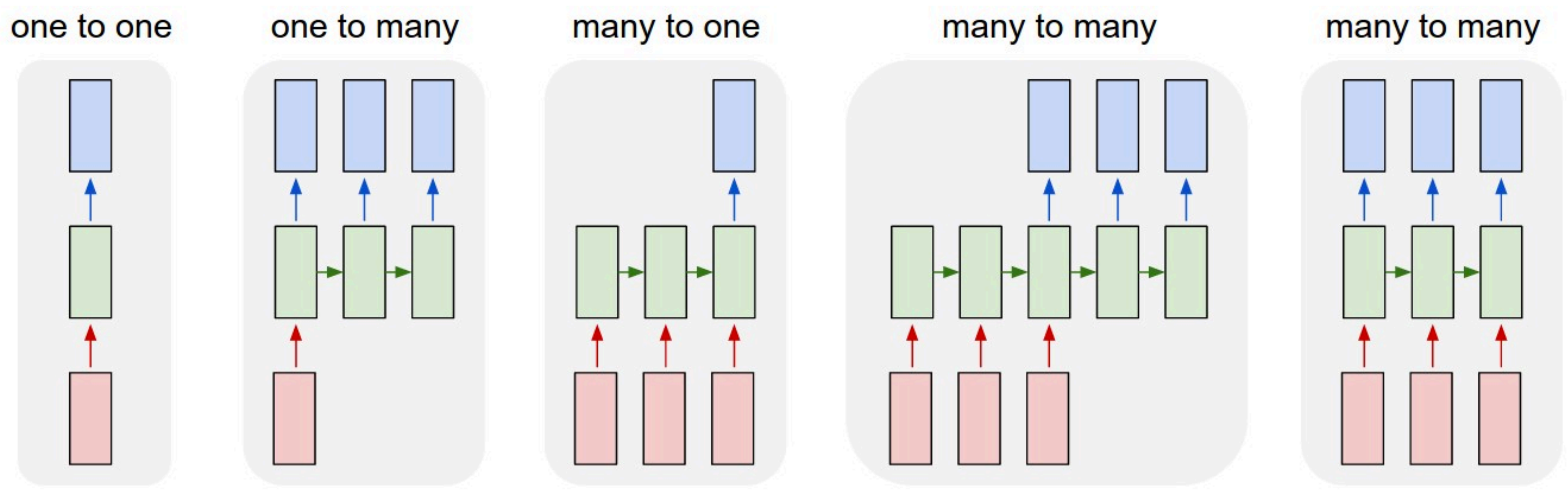
links:
https://karpathy.github.io/2015/05/21/rnn-effectiveness/
www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/
cs224d.stanford.edu/lecture_notes/notes4.pdf

Plan

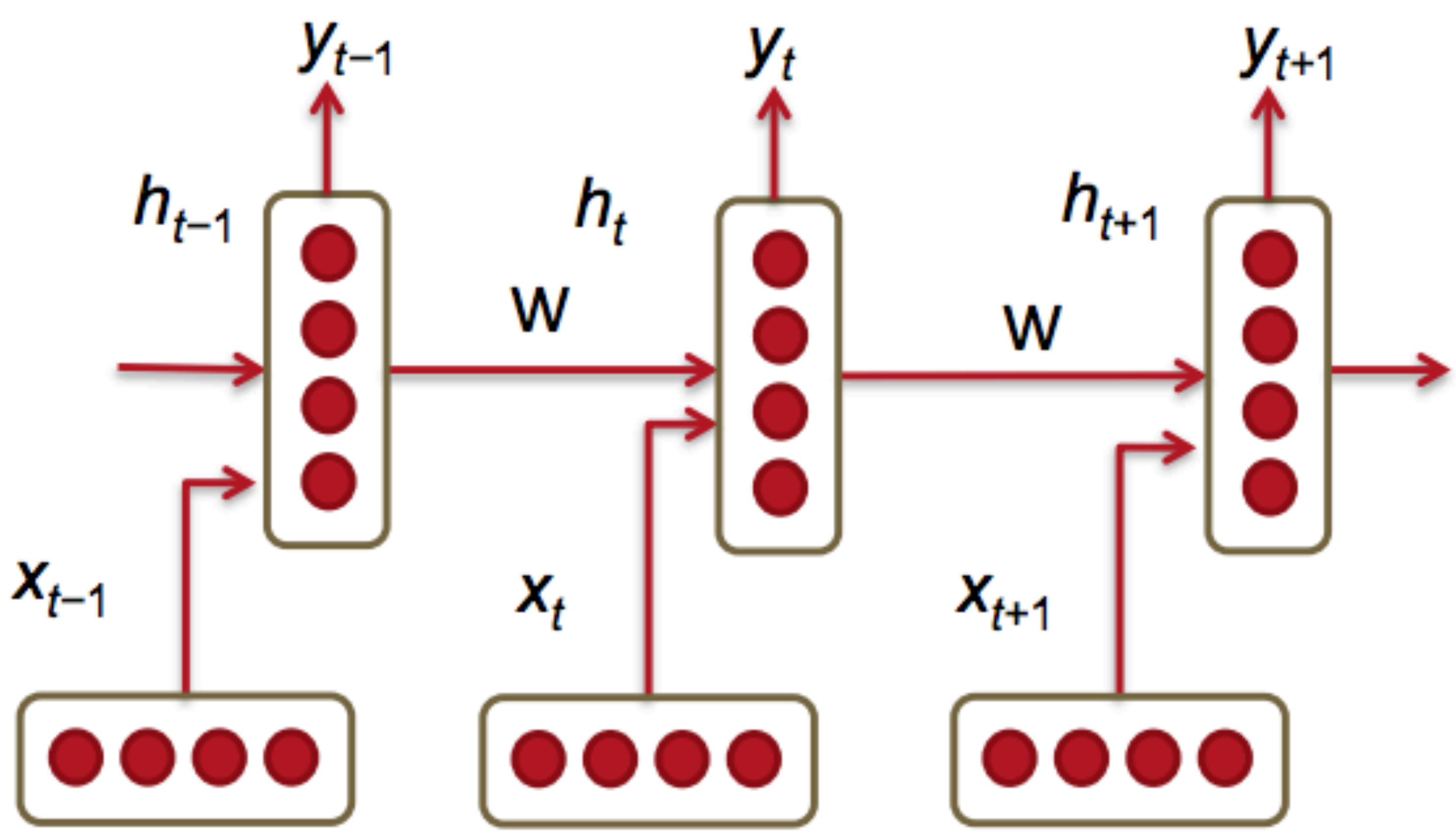
- What can you do with them
- Vanilla RNN
- Vanishing/exploding gradient problem
- LSTM, GRU
- Bidirectional RNN
- Some applications of RNNs

What is RNN and its purpose?

- A neural network architecture for working with sequence data, e.g. texts, stock prices, …



Vanilla RNN



$$h_t = \sigma(W^{(hh)} h_{t-1} + W^{(hx)} x_t)$$

$$\hat{y}_t = softmax(W^{(S)} h_t)$$

Vanishing/exploding gradient

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

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

$$E_t == J^{(t)}(\theta)$$

$$\boxed{\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{t} W^T \times diag[f'(j_{j-1})]}$$

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

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

Exploding grad solution

$$\hat{g} \leftarrow \frac{\partial E}{\partial W}$$

if $\| \hat{g} \| \geq \textit{threshold}$ **then**

$$\hat{g} \leftarrow \frac{\textit{threshold}}{\| \hat{g} \|} \hat{g}$$

end if

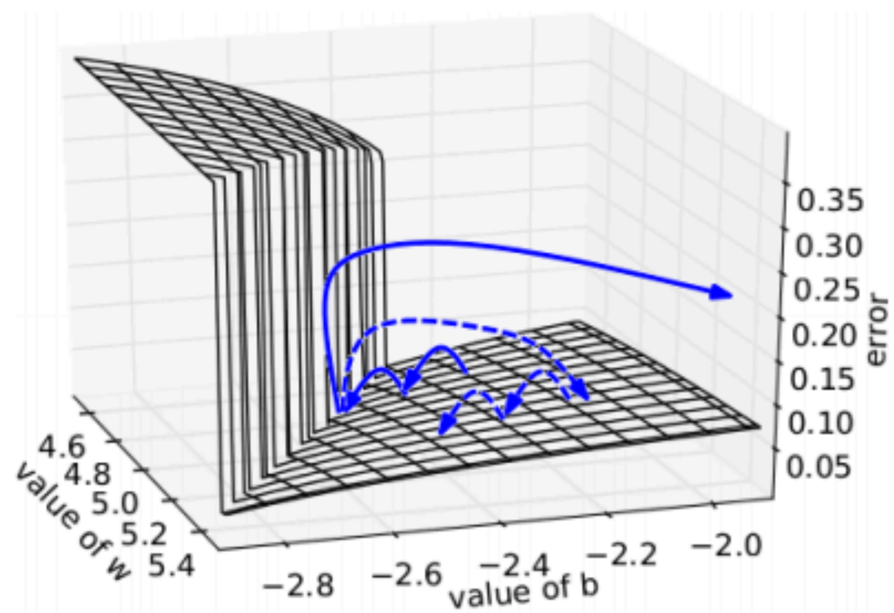


Figure 5: Gradient explosion clipping visualization

Vanishing grad solution

- Initialize W with identity matrix
- Use ReLU activation
- Or skip to the next slides with LSTM & GRU

LSTM

- Vanilla RNN cells can not capture long-term dependencies. Let's develop another cell!

$$
\begin{aligned}
i_t &= \sigma(W^{(i)} x_t + U^{(i)} h_{t-1}) && \text{(Input gate)} \\
f_t &= \sigma(W^{(f)} x_t + U^{(f)} h_{t-1}) && \text{(Forget gate)} \\
o_t &= \sigma(W^{(o)} x_t + U^{(o)} h_{t-1}) && \text{(Output/Exposure gate)} \\
\tilde{c}_t &= \tanh(W^{(c)} x_t + U^{(c)} h_{t-1}) && \text{(New memory cell)} \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t && \text{(Final memory cell)} \\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}
$$

GRU

- Another cell architecture but idea is the same: capture long-dependencies + eliminate vanishing gradient problem.

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

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1}) \qquad \text{(Reset gate)}$$

$$\tilde{h}_t = \tanh(r_t \circ Uh_{t-1} + Wx_t) \qquad \text{(New memory)}$$

$$h_t = (1 - z_t) \circ \tilde{h}_t + z_t \circ h_{t-1} \qquad \text{(Hidden state)}$$

Bidirectional RNN

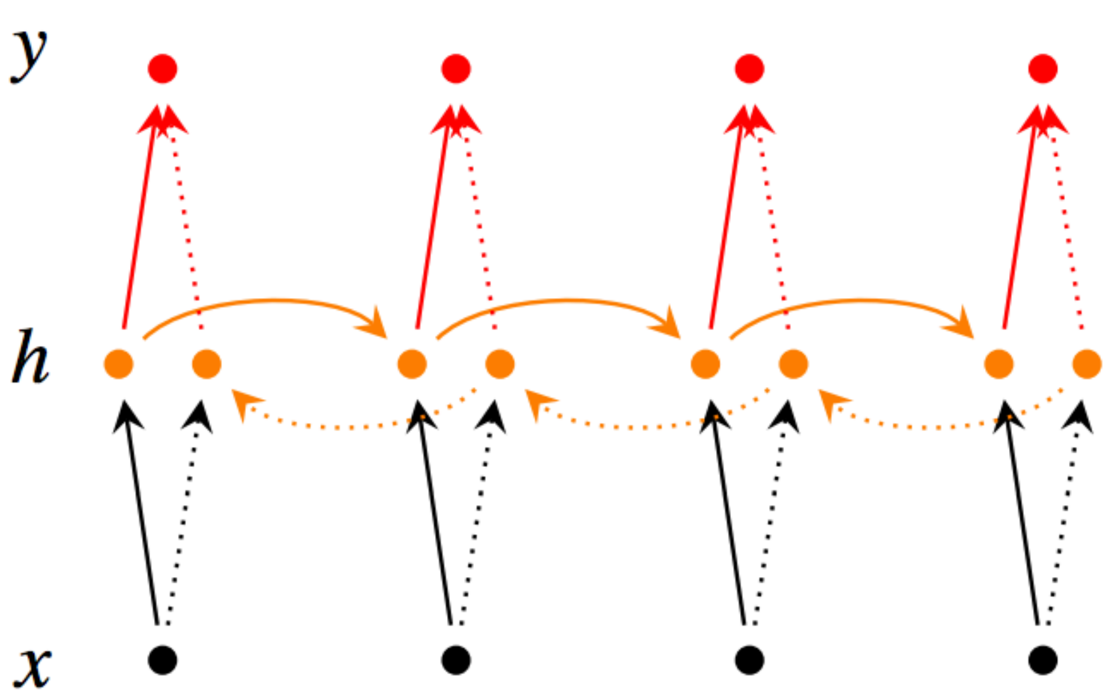


Figure 6: A bi-directional RNN model

$$\overrightarrow{h}_t = f(\overrightarrow{W} x_t + \overrightarrow{V} \overrightarrow{h}_{t-1} + \overrightarrow{b})$$

$$\overleftarrow{h}_t = f(\overleftarrow{W} x_t + \overleftarrow{V} \overleftarrow{h}_{t+1} + \overleftarrow{b})$$

$$\hat{y}_t = g(Uh_t + c) = g(U[\overrightarrow{h}_t; \overleftarrow{h}_t] + c)$$

Applications

- Machine Translation, e.g. Google Neural Machine Translation
  https://arxiv.org/pdf/1609.08144v2.pdf

- Conversational models & dialog systems
  https://arxiv.org/pdf/1506.05869.pdf

- Image captioning
  https://cs.stanford.edu/people/karpathy/deepimagesent/

- Speech recognition
  http://www.jmlr.org/proceedings/papers/v32/graves14.pdf

- And many more…