Deep Learning for NLP, and its application to NER

Pydata Cardiff

Lorenzo Bongiovanni

Senior Machine Learning Scientists at **Amplyfi**

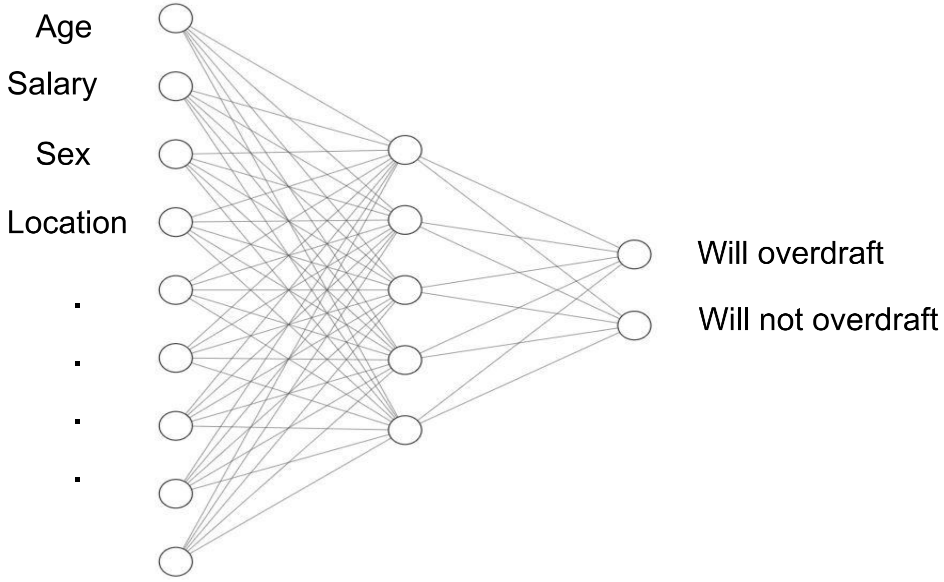
Overview

1. Bag of words approach for text classification
    - Bag of words
    - Feed Forward Neural Network
    - Example notebook
2. Recurrent Neural Networks (RNNs)
    - RNN structure
    - Vanishing Gradient problem
    - Example notebook
3. Long-Short Term Memory networks (LSTMs)
    - LSTM structure
    - Example notebook
4. An application to Named Entity Recognition
    - NER task description
    - Model description
    - Comparison with Spacy NER model

Standard classification task

Predict bank account overdraft

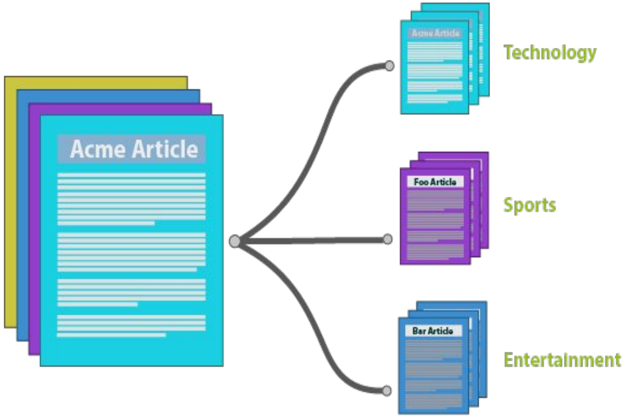
- Features : Age, Salary, Sex, Location, Own_house, …
- Output : Will overdraft, Will not overdraft
- Training & Test set : multiple examples of [Features, Output] to train and test the model



Text classification

- Given some text one wants to assign it to a category based on its content.

- For example, we want to categorise text into two categories :
  - **Sport**
  - **Computer Science**



- Different than standard classification problem as now the inputs are naturally ordered

Bag of Words

S = “I just wrote a new model and I want to run the training.”

We want to classify S as :

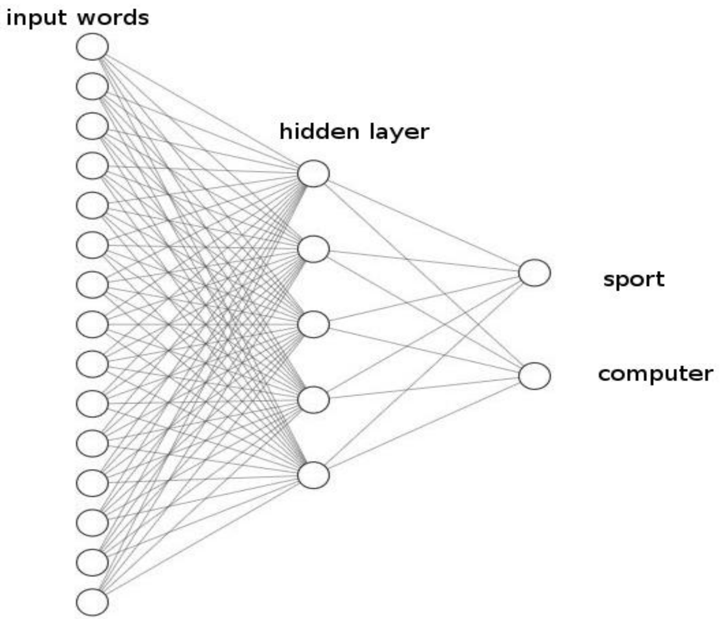
- Sport
- Computer Science

1) Bag of words : S -> unordered list of words

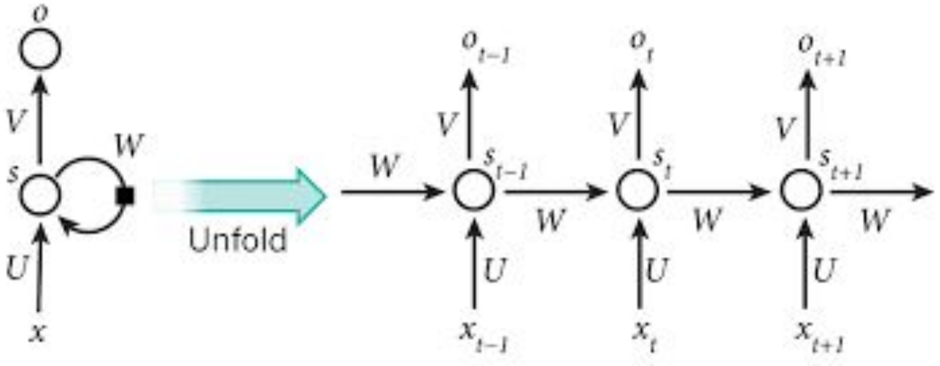
2) Vectorize S based on words count :

wrote	model	run	training	other_words
1	1	1	1	0 … 0

3) Feed to a classifier (for example NN)



Recurrent Neural Networks (RNN)



- Feed the words in the same order they appear in the text.
- Hidden layer at time t is self connected with itself at timestep t-1 through W.

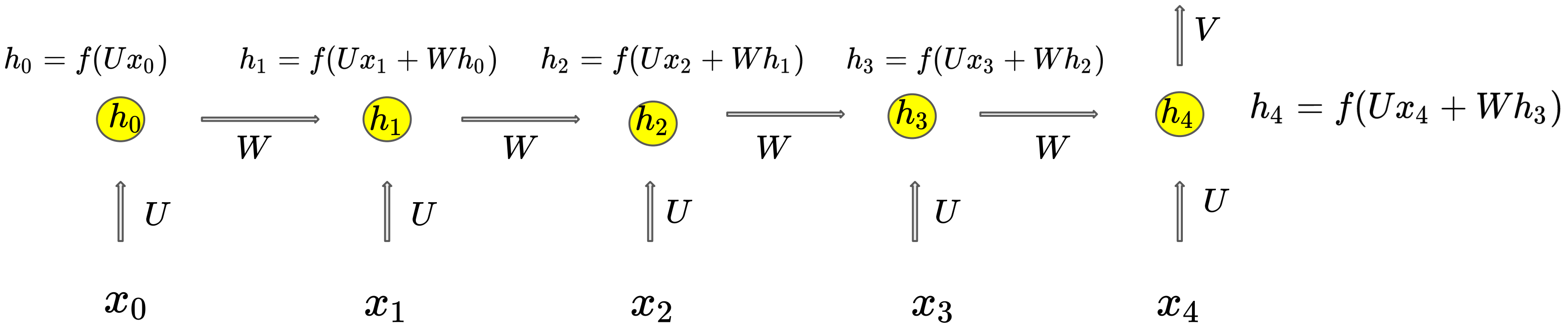
- The recurrency of W allows, in principle, to model the sequential nature of inputs x.

RNN Forward pass

$$h_t = f(Ux_t + Wh_{t-1})$$

- sentence = “ $x_0 \;\; x_1 \;\; x_2 \;\; x_3 \;\; x_4$ ”

output $= g(Vh_4)$

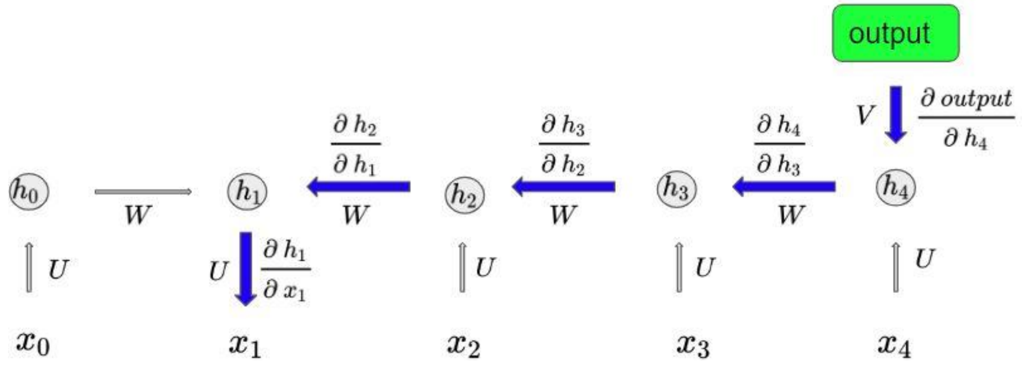


- Last hidden layer represents a **complete encoding** of the sentence :

$$h_4 = f(Ux_4 + Wf(Ux_3 + Wf(Ux_2 + Wf(Ux_1 + Wf(Ux_0)))))$$

Sensitivity of RNN output from input position

We want to know how much the input at time t influences the output :



$$\frac{\partial output}{\partial x_t} = \frac{\partial o}{\partial h_t}\frac{\partial h_t}{\partial x_t} = \frac{\partial o}{\partial h_T}\frac{\partial h_T}{\partial h_{t-1}}\frac{\partial h_{T-1}}{\partial h_{T-2}}\cdots\frac{\partial h_{t+1}}{\partial h_t}\frac{\partial h_t}{\partial x_t}$$

$$\frac{\partial h_t}{\partial h_{t-1}} = f'_t \cdot W$$

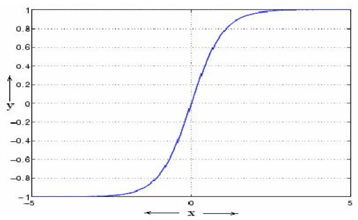
$$\frac{\partial output}{\partial x_t} = A \cdot \left(\prod_{i=T}^{t} W f'_i\right) \cdot B < |AB|\, |W f'_{MAX}|^{T-t}$$

Two cases :

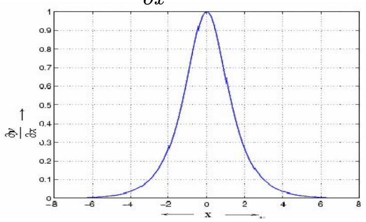
$|W f'_{max}| > 1$ **Exploding** gradient. Cured by gradient clipping.

$|W f'_{max}| < 1$ **Vanishing** gradient. Much harder to deal with.

$tanh(x)$



$\frac{\partial}{\partial x} tanh(x)$



Back to the example in the notebook

Sentence : “I wrote this computer code and now I want to run the training”

t : 0 1 2 3 4 5 6 7 8 9 10 11 12

Sensitivity of RNN to word “computer” :

$$\frac{\partial output}{\partial \text{"}computer\text{"}} < |AB|\,|Wf'_{MAX}|^{T-t} \sim |AB| \cdot 0.7^9$$

Sensitivity of RNN to word “training” :

$$\frac{\partial output}{\partial \text{"}training\text{"}} < |AB|$$

Ratio of sensitivities :

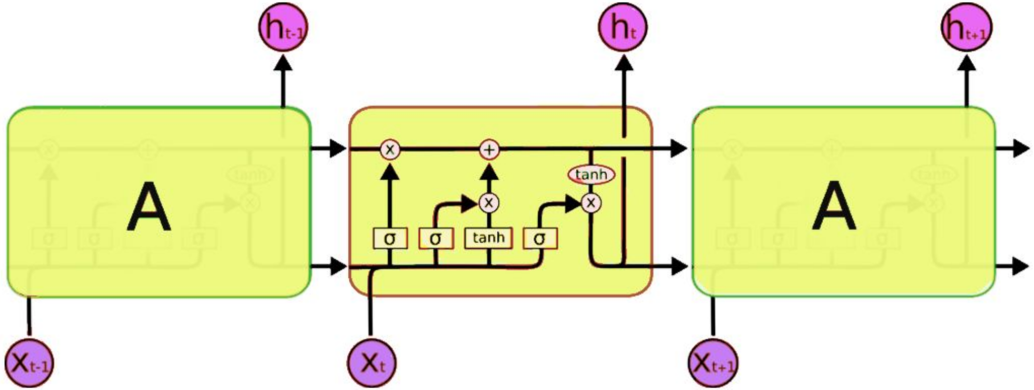
$$\frac{rnn\ sens.to\ \text{"}training\text{"}}{rnn\ sens.to\ \text{"}computer\text{"}} > 25$$

Our model depends at least **25 times** more on the word “training” than on the word “computer” to make its predictions!

Vanish gradient problem

- Well known problem since the beginning of the 90s’.
  - Hochreiter . *Untersuchungen zu dynamischen neuronalen Netzen*. Diploma thesis, Institut f. Informatik, Technische Univ. Munich, 1991
  - **Bengio** et al. : *Learning long-term dependencies with gradient descent is difficult* - IEEE Transactions on Neural Networks, 1994
- Applies not only to RNNs but in general to every multi-layer neural network.
- “Deep Learning” -> “deep” means (maybe) that the net architecture is deep enough for the vanishing gradient problem to arise.
- There is no standard way to deal with it, but many different solutions that apply to different kind of networks.
- **Long Short Term Memory (LSTM)** is the smart solution to the vanishing gradient problem for RNNs.
  - Hochreiter & Schmidhuber **:** *Long Short Term-Memory*, Neural Computation 9(8):1735-80, 1997

LSTM



LSTM equations :

$$f_t = \sigma_g(U_f x_t + W_f h_{t-1})$$
$$i_t = \sigma_g(U_i x_t + W_i h_{t-1})$$
$$o_t = \sigma_g(U_o x_t + W_o h_{t-1})$$
$$\tilde{c}_t = \sigma_c(U_c x_t + W_c h_{t-1})$$

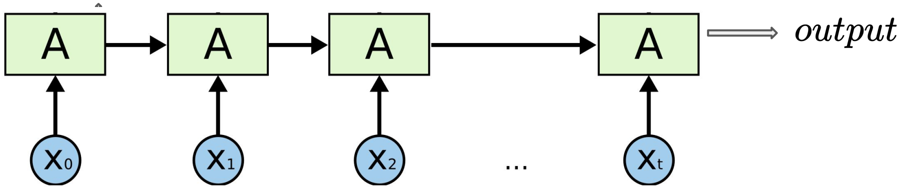
$$c_t = f_t \bullet c_{t-1} + i_t \bullet \tilde{c}_t$$
$$h_t = o_t \bullet \sigma_h(c_t)$$

- LSTM is formed by 4 **independent** (and parallel) RNNs.
- LSTM has 2 fundamental states :
  - hidden state h_t
  - memory cell c_t
- h_t depends on h_{t-1} now **indirectly** via the memory cell c_t.
- c_t decouples adjacent time-steps and allows **controlled error flow** from h_t to h_{t-1} .

RNN equation :

$$h_t = \sigma_g(U\, x_t + W\, h_{t-1})$$

Example of LSTM error flow control



LSTM equations :

$$f_t = \sigma_g(U_f x_t + W_f h_{t-1})$$
$$i_t = \sigma_g(U_i x_t + W_i h_{t-1})$$
$$o_t = \sigma_g(U_o x_t + W_o h_{t-1})$$
$$\tilde{c}_t = \sigma_c(U_c x_t + W_c h_{t-1})$$

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

Case where only x_0 is important :

- Input gate is open only for the first input x_0 : $i_t = [1, 0, 0, \ldots, 0]$
- Forget gate is always open (=1) except for x_0 : $f_t = [0, 1, 1, \ldots, 1]$
- Output gate is always open : $o_t = [1, 1, 1, \ldots, 1]$
- Memory cell, simply repeats itself : $c_t = [\tilde{c}_0, \tilde{c}_0, \tilde{c}_0, \ldots, \tilde{c}_0]$
- Hidden layer always encodes the memory at time 0 : $h_t = [\sigma_h(\tilde{c}_0), \sigma_h(\tilde{c}_0), \ldots, \sigma_h(\tilde{c}_0)]$

- Sensitivity of output to x_0 : $$\frac{\partial\, output}{\partial x_0} = \frac{\partial\, output}{\partial\, h_T} \frac{\partial\, h_T}{\partial x_0} = A\, \sigma'_h\, \sigma'_c\, U_c$$

No exponential decay in time!!!

Named Entity Recognition (NER)

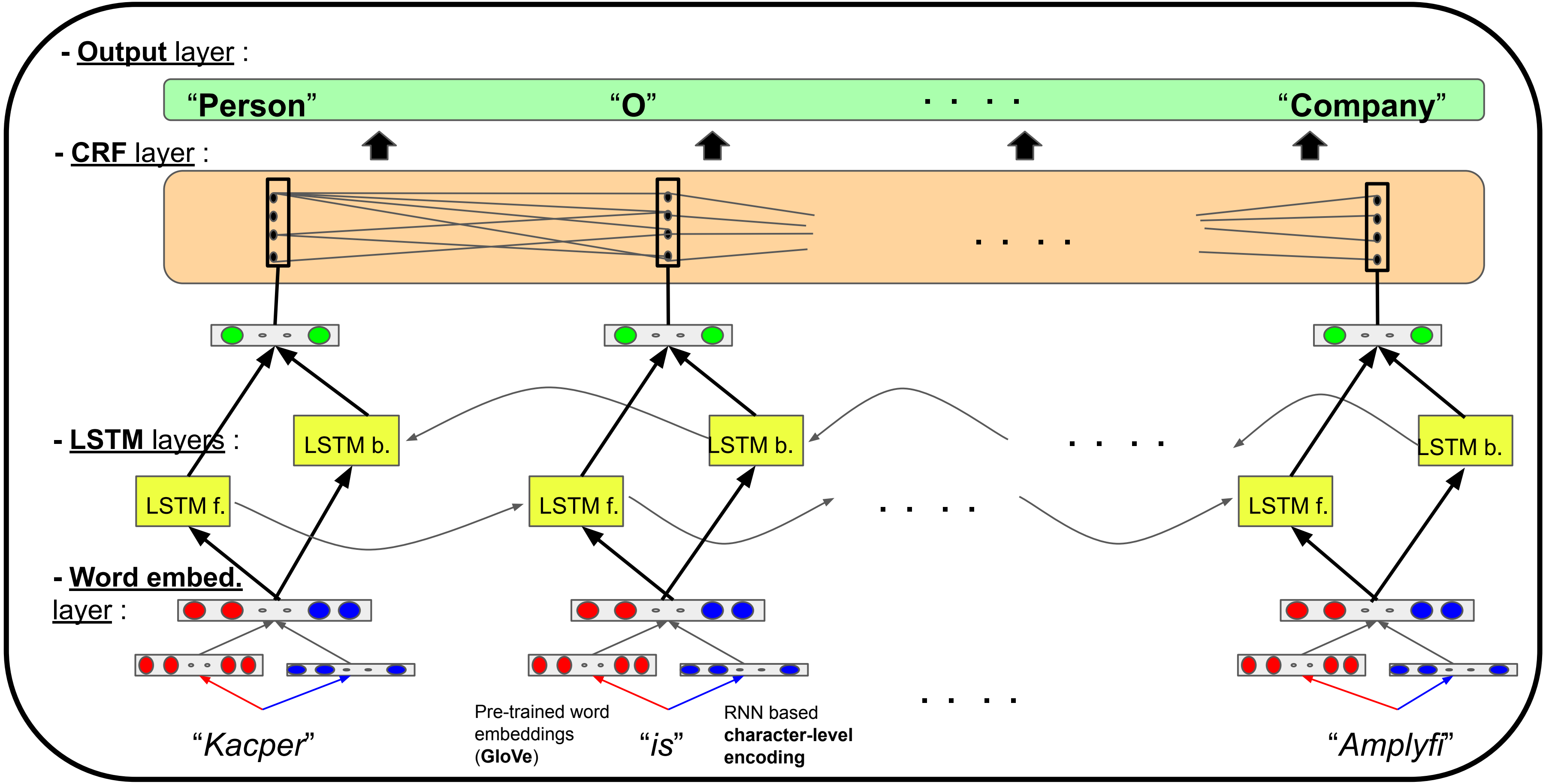
Is the task of detecting if each word in a text is an entity or not.

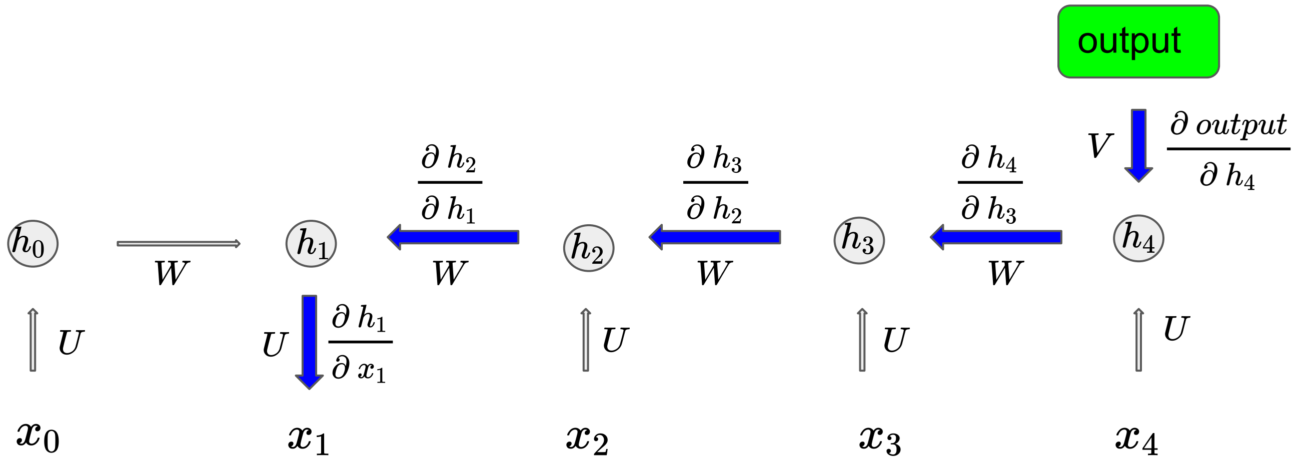
Sentence : “Kacper is a data scientists and works at Amplyfi”

NER_tags : “ PER   O O O   O   O   O   O ORG ”

Main 3 NER categories : Person, Location, Organization

LSTM-based NER model




output
V
$\frac{\partial\, output}{\partial\, h_4}$
$\frac{\partial\, h_2}{\partial\, h_1}$
$\frac{\partial\, h_3}{\partial\, h_2}$
$\frac{\partial\, h_4}{\partial\, h_3}$
h_0
h_1
h_2
h_3
h_4
W
W
W
W
U
U
$\frac{\partial\, h_1}{\partial\, x_1}$
U
U
U
x_0
x_1
x_2
x_3
x_4