TP2: Word Embedding - RNN

Hachem Madmoun

November 4, 2019

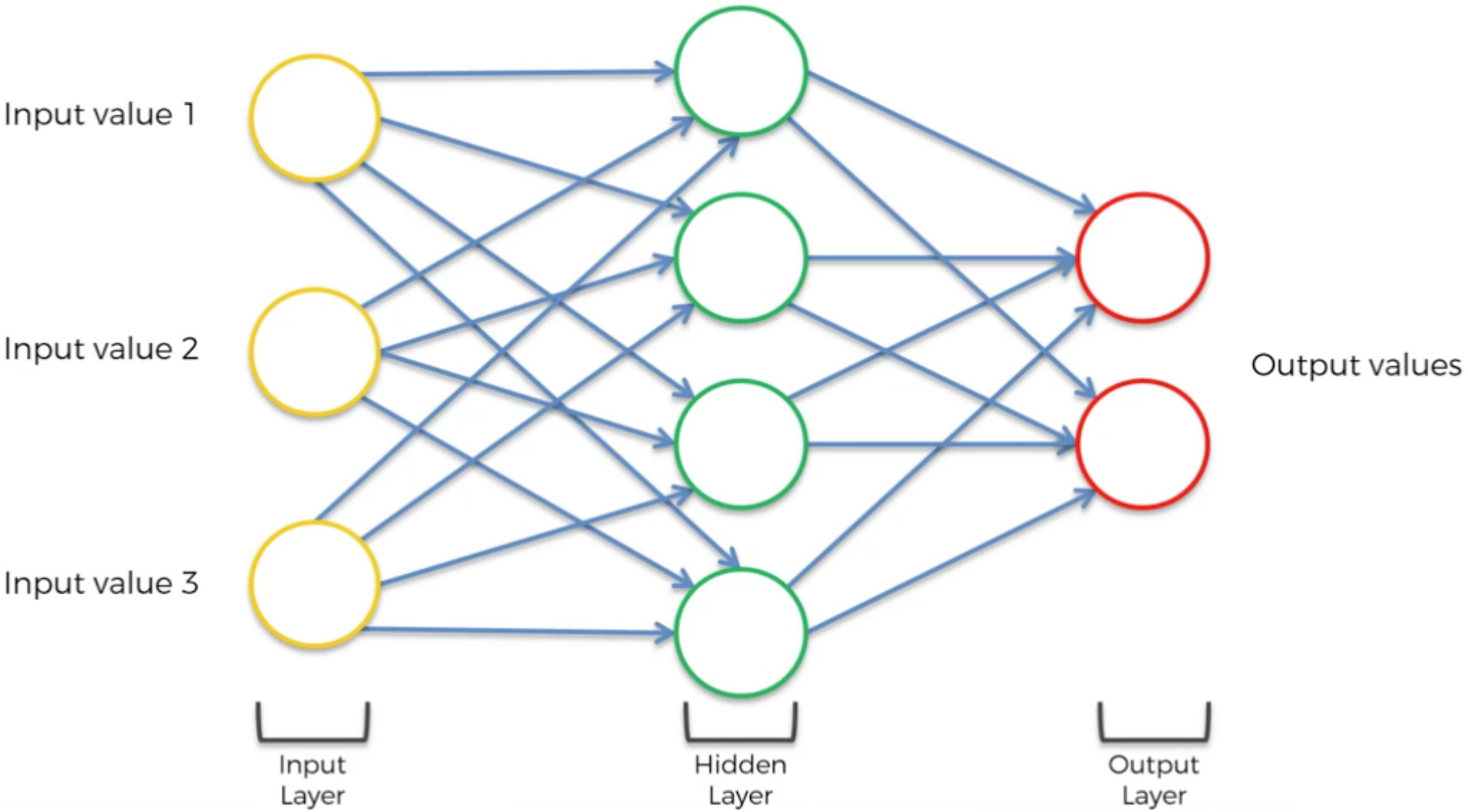
Outline

Outline

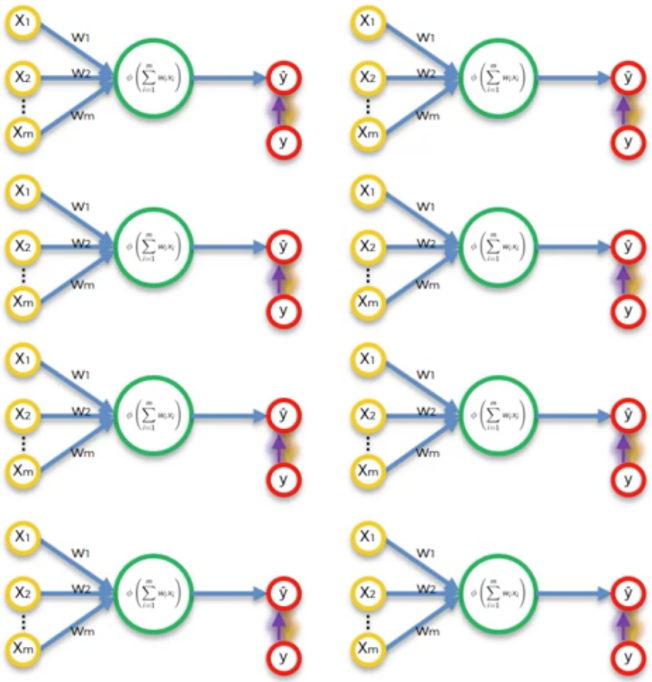
Plan of attack

- The idea behind Recurrent Neural Networks
- The vanishing Gradient Problem
- LSTM
- Step-by-Step LSTM Walk Through

A normal Neural network



How do Neural Networks learn?



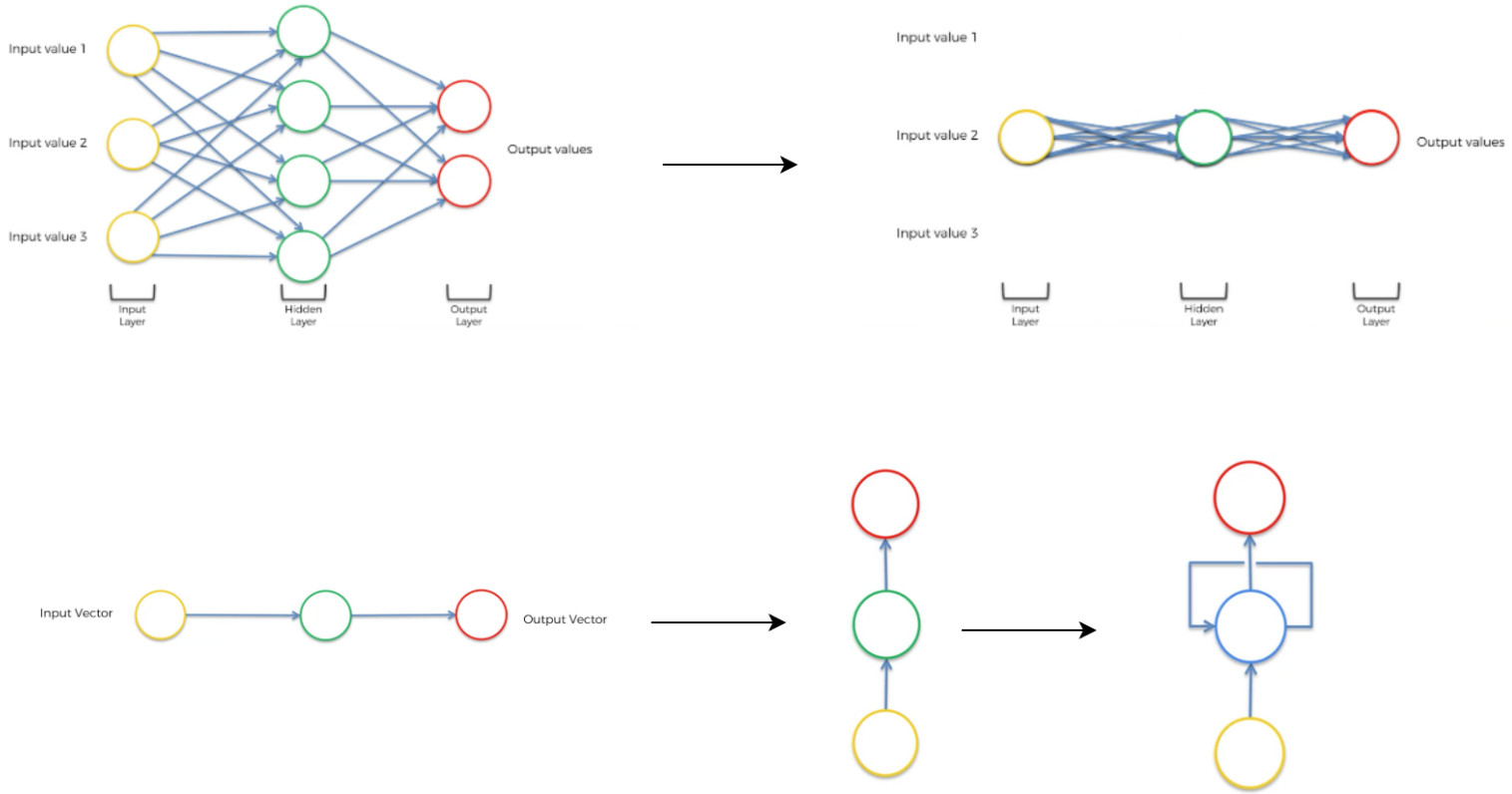
Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

$$C = \sum \tfrac{1}{2}(\hat{y} - y)^2$$

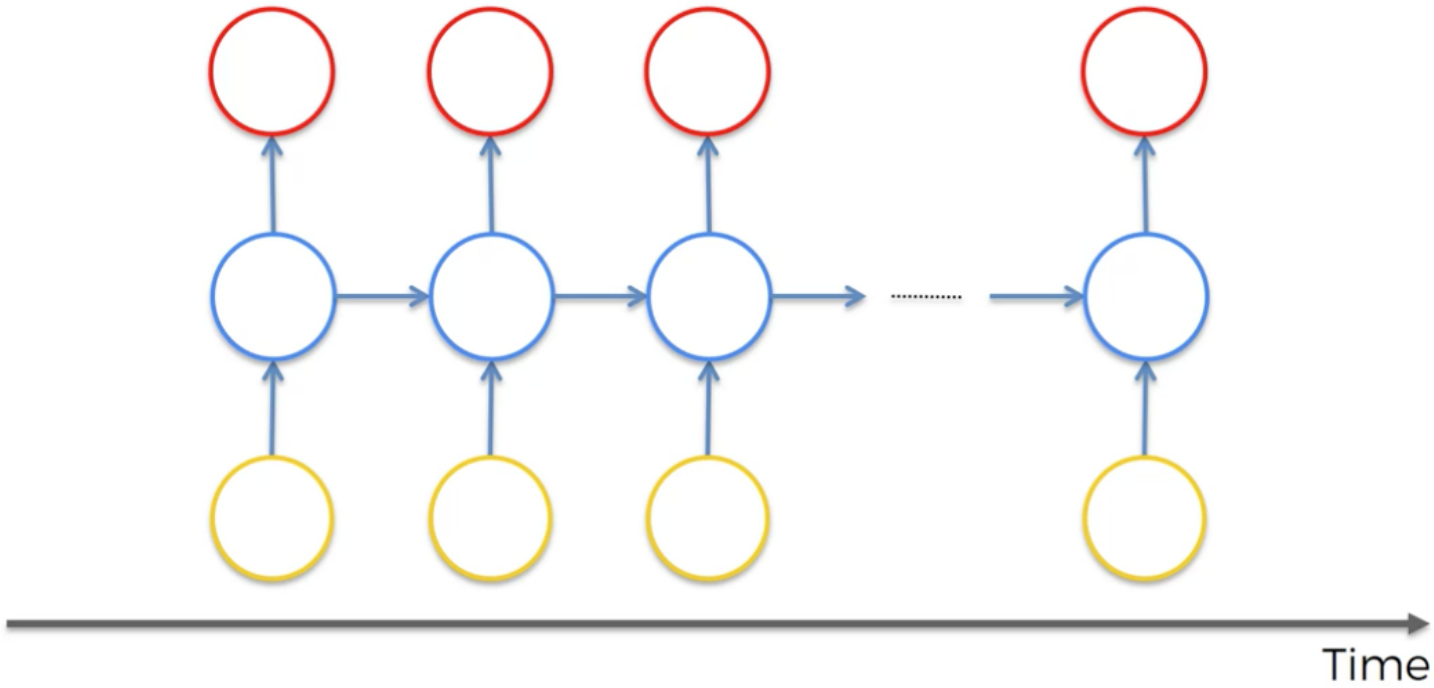
BACKPROPAGATION

- **Step 1 :** Randomly initialise the weights to small numbers close to 0 (NOT 0)
- **Step 2 :** Input the first observation in the input layer, each feature in one input node.
- **Step 3 :** Forward-Propagation : from left to right, the neurons are activated in a way that the impact of each neuron's activation is limited by the weights. Propagate the activations until getting the preticted result
- **Step 4 :** Back-propagation : from right to left, the error is back-propagated. Update the weights, the learning rate decides by how much we update the weights.
- **Step 5 :** Repeat Steps 1 to 4 and update the weights after each observation $\Longrightarrow$ Reinforcement Learning / after a batch of observation $\Longrightarrow$ Batch Learning.
- **Step 6 :** When the whole training set is passed through the ANN, that makes an epoch. Redo more epochs

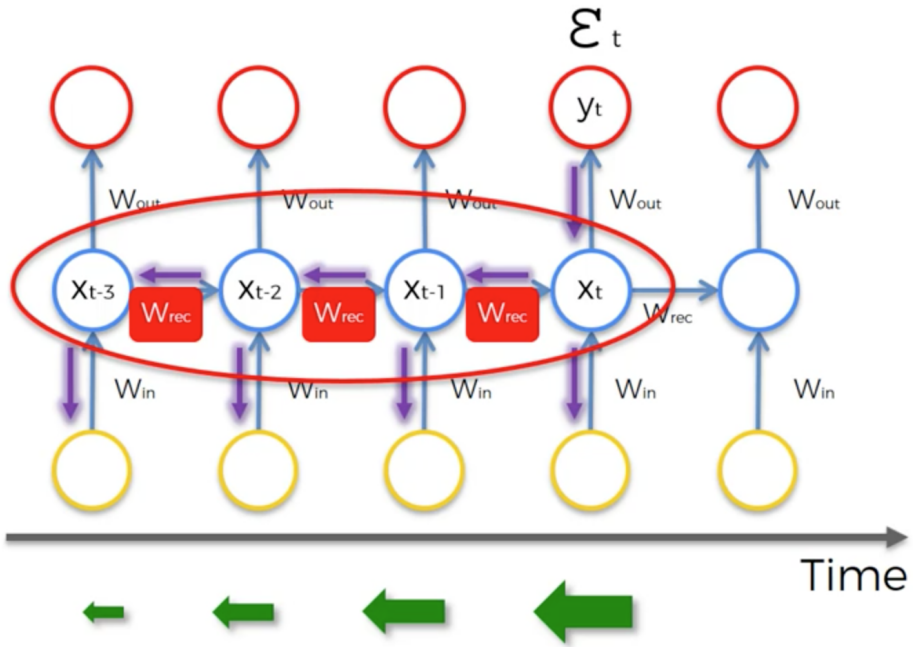
From ANN to RNN



RNN



The Vanishing gradient problem



$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{1 \leq t \leq T} \frac{\partial \mathcal{E}_t}{\partial \theta} \quad (3)$$

$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \leq k \leq t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right) \quad (4)$$

$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t \geq i > k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} = \prod_{t \geq i > k} \mathbf{W}_{rec}^T diag(\sigma'(\mathbf{x}_{i-1})) \quad (5)$$

W_{rec} ~ small ⇨ Vanishing

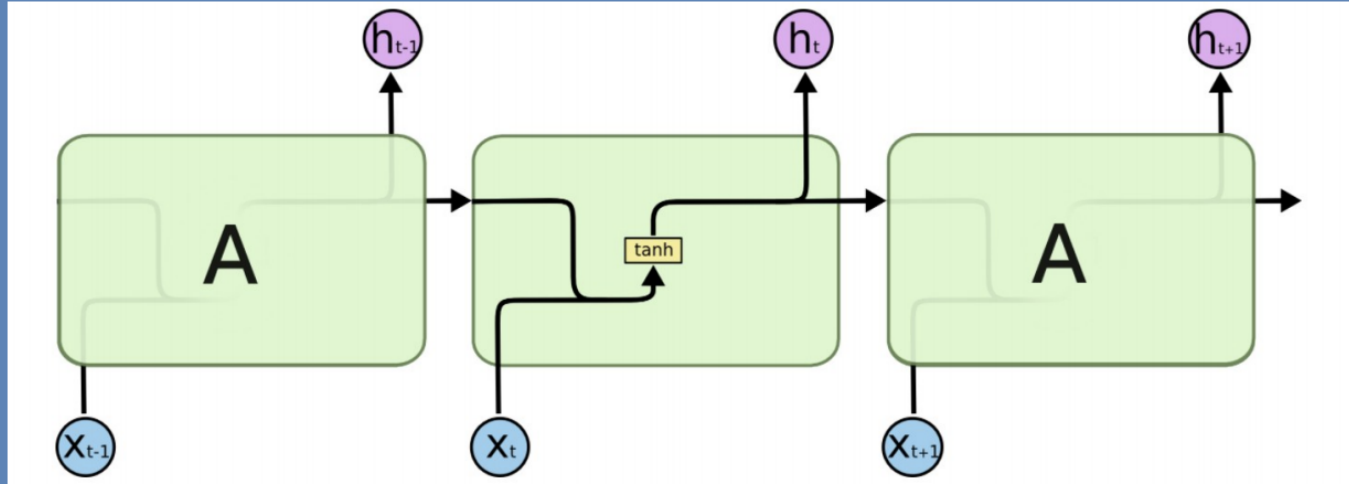
W_{rec} ~ large ⇨ Exploding

Formula Source: Razvan Pascanu et al. (2013)

Resources

- Untersuchungen zu dynamischen neuronalen Netzen (1991) by Sepp (Josef) Hochreiter
  http://people.idsia.ch/~juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf
- Learning Long-Term Dependencies with Gradient Descent is Difficult (1994) by Yoshua Bengio.
  http://www.iro.umontreal.ca/~lisa/pointeurs/ieeetrnn94.pdf
- On the difficulty of training recurrent neural networks (2013) by Razvan Pascanu and Yoshua Bengio.
  http://proceedings.mlr.press/v28/pascanu13.pdf

Standard RNN



The repeating module in a standard RNN contains a single layer.

Equations of RNN

$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

LSTM

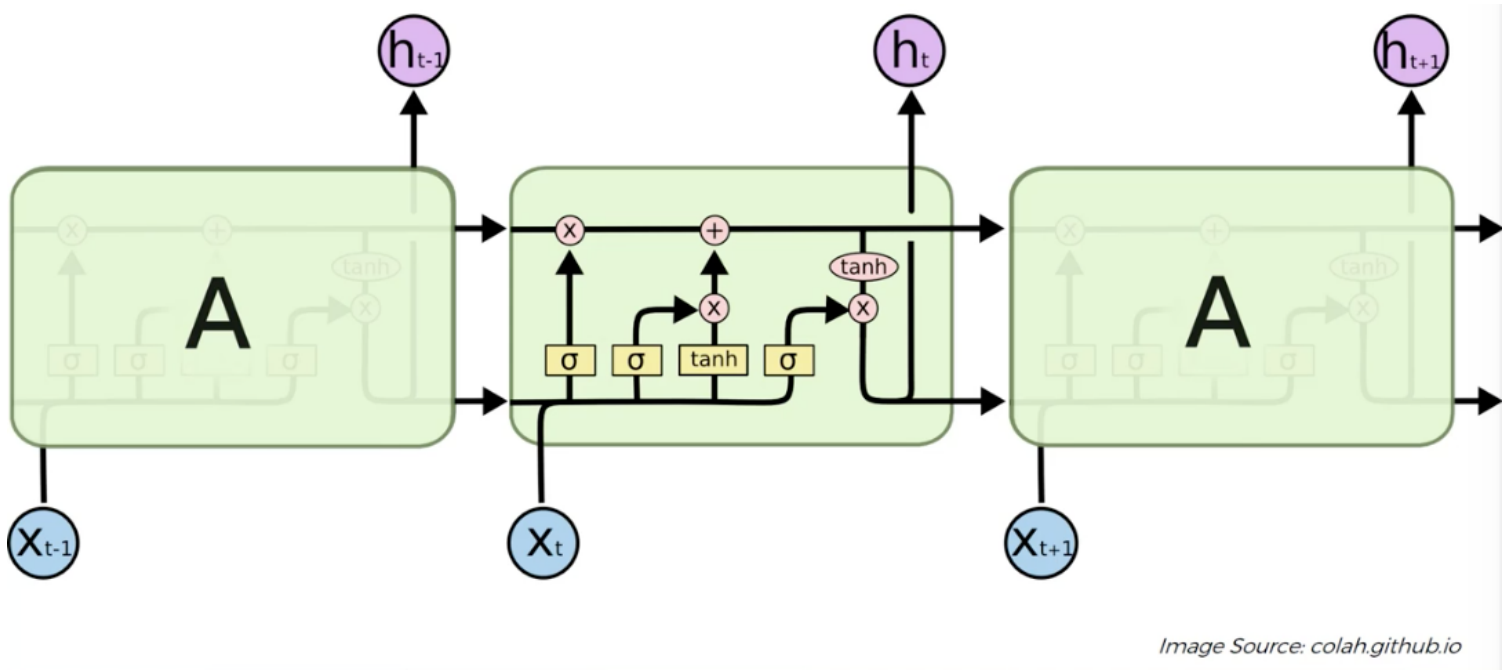
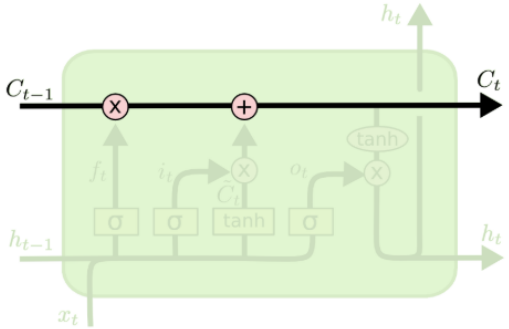


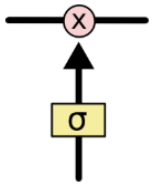
Image Source: colah.github.io

The Core Idea Behind LSTM

- The cell state :

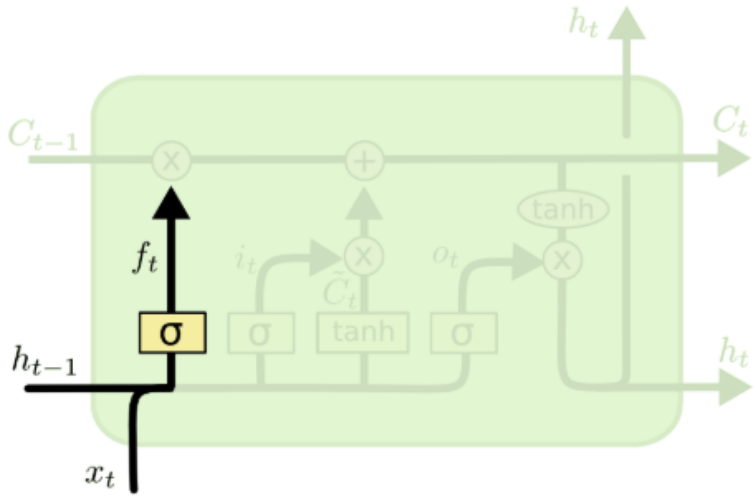


- The concept of gate :



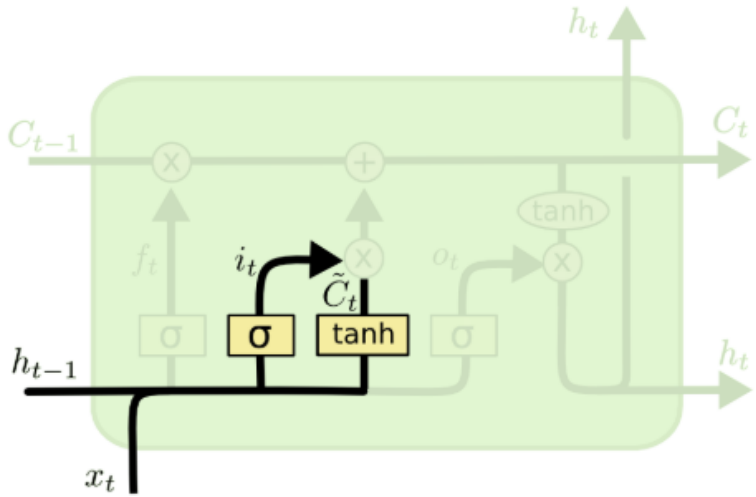
- The LSTM has three of these gates.

Step 1 : The forget gate layer



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

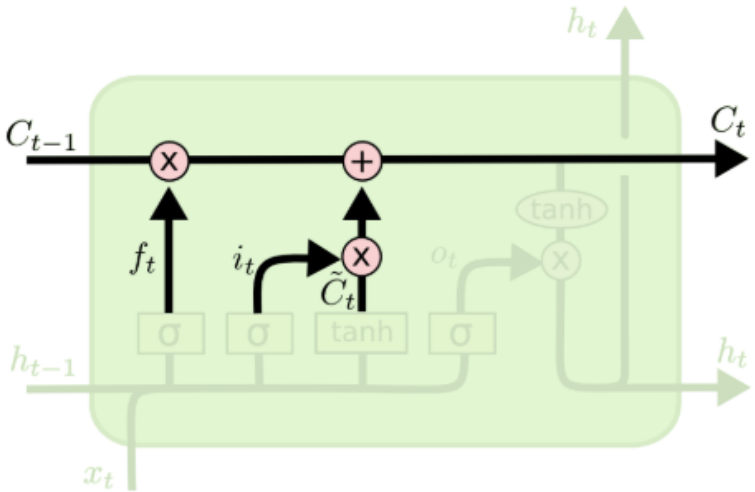
STEP 2 : THE INPUT GATE LAYER



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

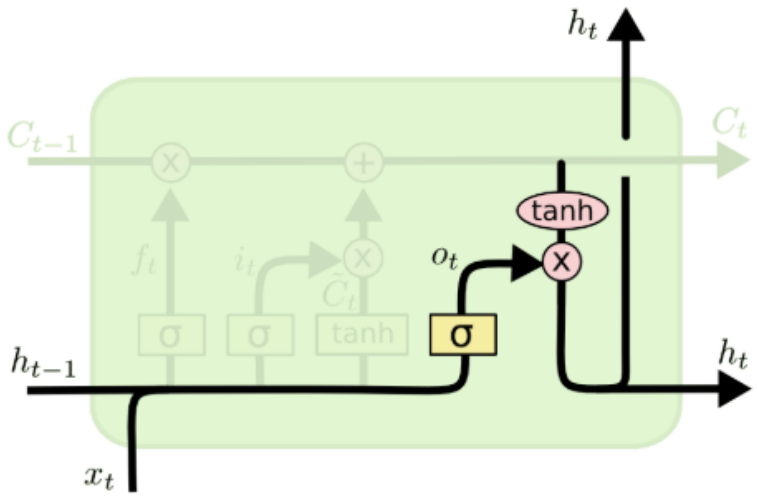
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

STEP 3 : UPDATE THE CELL STATE



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

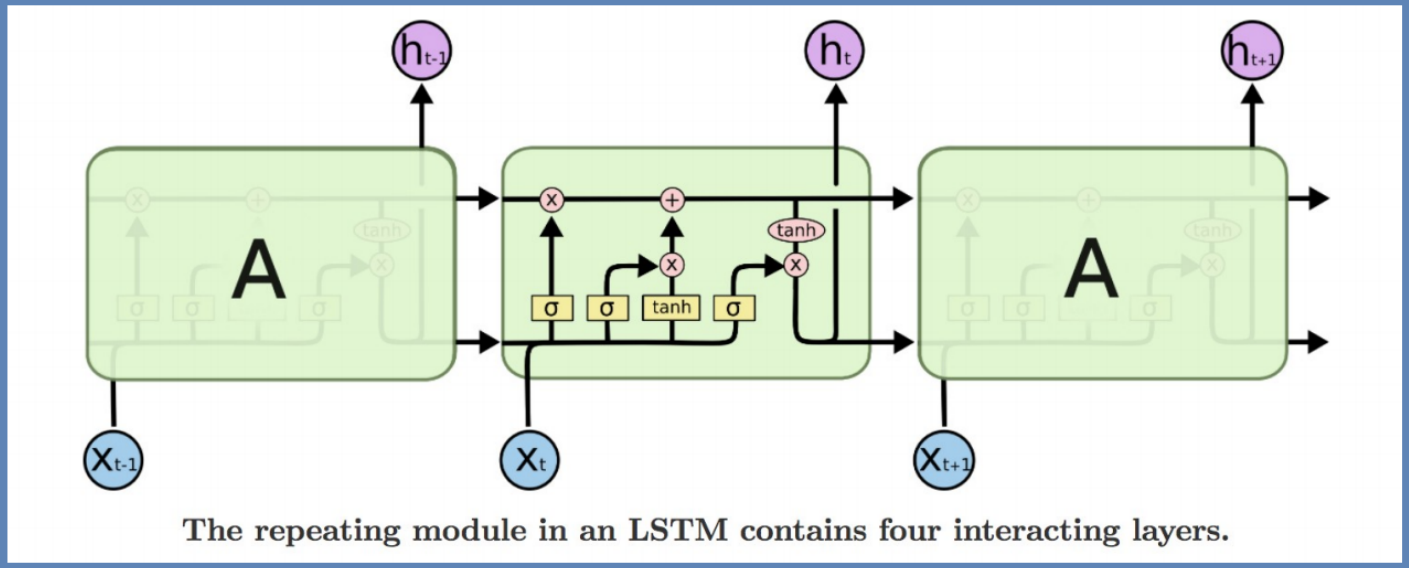
STEP 4 : THE OUTPUT GATE



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

$$h_t = o_t * \tanh\left(C_t\right)$$

SUMMARY LSTM



The repeating module in an LSTM contains four interacting layers.

Equations of LSTM

Gates

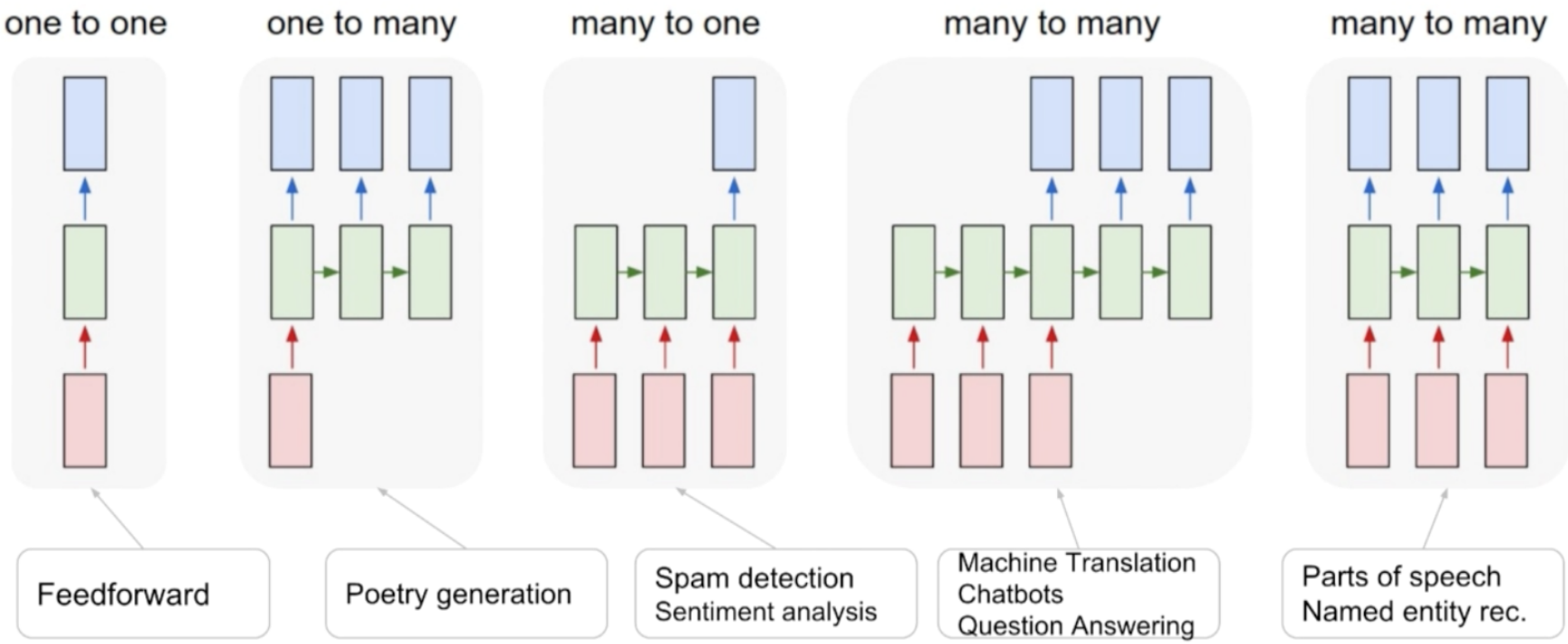
$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$

Updates

$$\tilde{c_t} = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$
$$c_t = f_t\, c_{t-1} + i_t \tilde{c_t}$$
$$h_t = o_t \tanh(c_t)$$

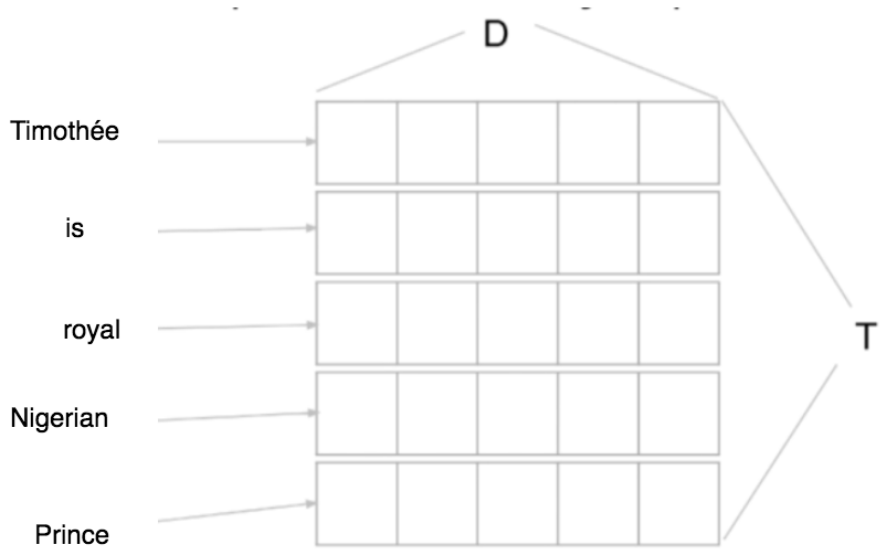
Outline

Categories of tasks

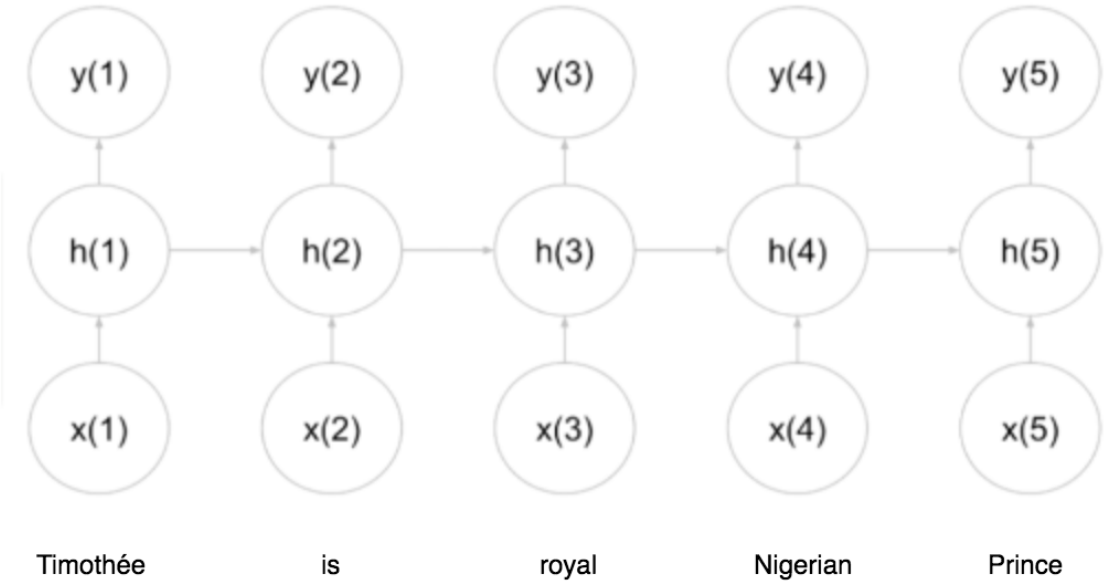


RNN INPUT

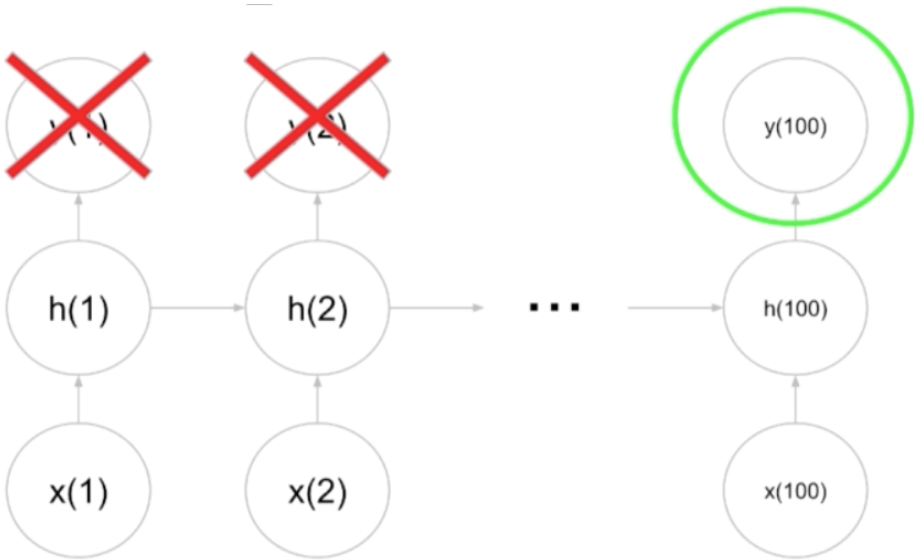
- The input is a sentence, each token is encoded in D-dimensional vector.
- T : is the sequence length, D : input dimensionality.



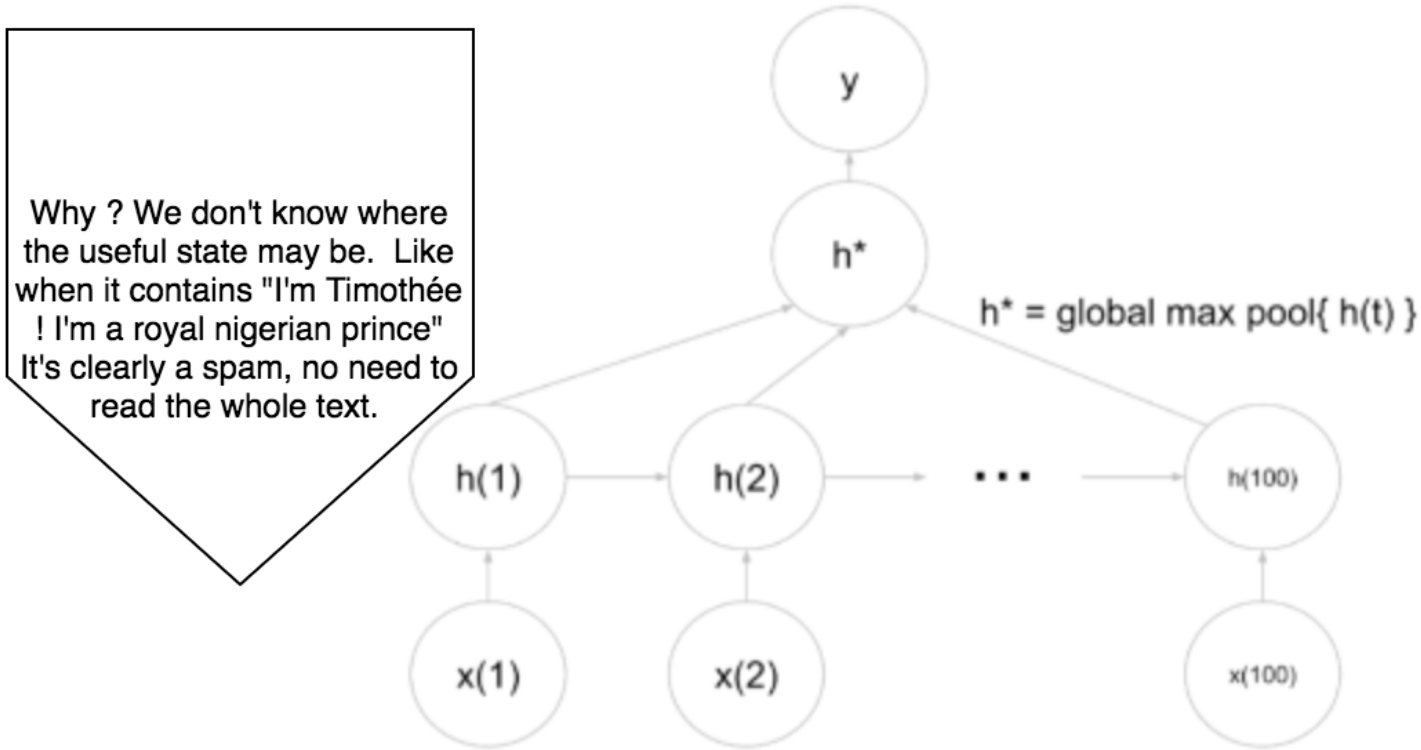
RNN OUTPUT


y(1)
y(2)
y(3)
y(4)
y(5)
h(1)
h(2)
h(3)
h(4)
h(5)
x(1)
x(2)
x(3)
x(4)
x(5)
Timothée
is
royal
Nigerian
Prince

First Case - We take the last output



Second Case - We use a Global Maxpool



Outline

SKIP-GRAM - CBOW

- The idea behind this method is that we can get a lot of value by representing a word by means of its neighbors. As John Rupert Firth [1] said : *"You shall know a word by company it keeps"*
- So, we will build a dense vector for each word type, chosen so that it is good at predicting other words appearing in its context.
- There are two Algorithms :
  1. Skip-Gram (SG) : Predict context words given target (position independent)
  2. Continuous Bag of Words (CBOW) : Predict Target word from bag-of-words context

1. Leading figure in British linguistics during 1950s

WORD2VEC

- The aim is to create a model such that given the center word, the model will be able to predict or generate the surrounding words.
- For example :

- So, for each word $t = 1, \cdots, T$, we want to predict the surrounding words in a window of radius m.

WORD2VEC

- **Each word has two vector representations** [2], one as an input (which is the center word for the skip gram model), and an other as an output (which is the context for this model) : "u_w" will represent the vector form of the word "w" as a context word and "v_w" will represent its vector form as a center word.
- So, there are two matrices that we should learn : $\mathcal{V} \in \mathbb{R}^{N \times |V|}$ the **Input Word Matrix** and $\mathcal{U} \in \mathbb{R}^{|V| \times N}$ the **Output Word Matrix**.
- We should notice that there are some hyperparameters : m the window size and the dimension of the vectors.

2. We could have one word vector representation, but by using two word vector representations, it turns out that not only does that make the math easier because the two representations are separated when performing optimization, but it actually works better in practise

The Optimization Problem

- Let θ represent all the vector representations of the words, which we aim to optimize.
- We will see that the optimization problem is :

$$\text{Maximize} \quad \prod_{t=1}^{T} \prod_{\substack{j \neq 0 \\ -m \leq j \leq m}} \mathbb{P}(w_{t+j} | w_t; \theta)$$

- By taking the logarithm. And rather than having the probability of the whole corpus, we can take the average over each position [3]. Thus, the optimization problem becomes :

$$\text{Minimize} \quad -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{j \neq 0 \\ -m \leq j \leq m}} \log(\mathbb{P}(w_{t+j} | w_t; \theta))$$

3. The normalization won't affect the maximum

The Optimization Problem

For $\mathbb{P}(w_{t+j}|w_t;\theta)$, we will use the softmax form :

- Let "o" be the output word index and "c" the center word index. So, "v_c" and "u_o" are "center" and "output" vectors of indices c and o

$$\mathbb{P}(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{|V|} \exp(u_w^T v_c)}$$

- The dot product $u^T v$ gets bigger if u and v are "more similar".
- the softmax form is the standard way to turn numbers into probabilities (exponentiate them to make them positive and normalize them to give probability).

The Steps of Skip Gram Model

- We first generate our one-hot vector $x \in \mathbb{R}^{|N|}$
- We get our **embedded vector** $v_c = \mathcal{V}x \in \mathbb{R}^N$

$$N \underbrace{\begin{pmatrix} | & & | & & | \\ v_1 & \cdots & v_c & \cdots & v_{|V|} \\ | & & | & & | \end{pmatrix}}_{|V|} \overbrace{\begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}}^{|V|} = v_c$$

The Steps of Skip Gram Model

- We generate our score vector $z = \mathcal{U}v_c$ with : $y_o = u_o^T v_c$

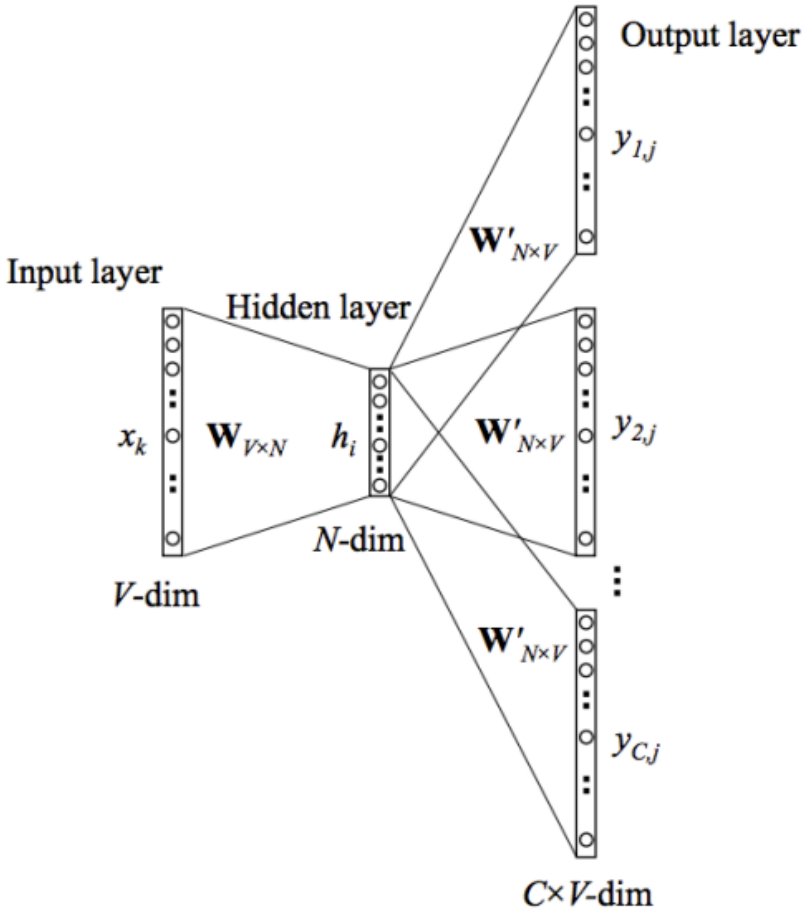
$$
\begin{matrix} & & N & & \\ |V| & \begin{pmatrix} - & u_1 & - \\ & \vdots & \\ - & u_o & - \\ & \vdots & \\ - & u_{|V|} & - \end{pmatrix} & v_c = \end{matrix} \quad \begin{matrix} |V| \\ \begin{pmatrix} y_1 \\ \vdots \\ y_o \\ \vdots \\ y_{|V|} \end{pmatrix} \end{matrix}
$$

- We turn the vector into probabilities $\hat{y} = \textbf{softmax}(z)$:

$$
\hat{y}_o = \mathbb{P}(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{|V|} \exp(u_w^T v_c)}
$$

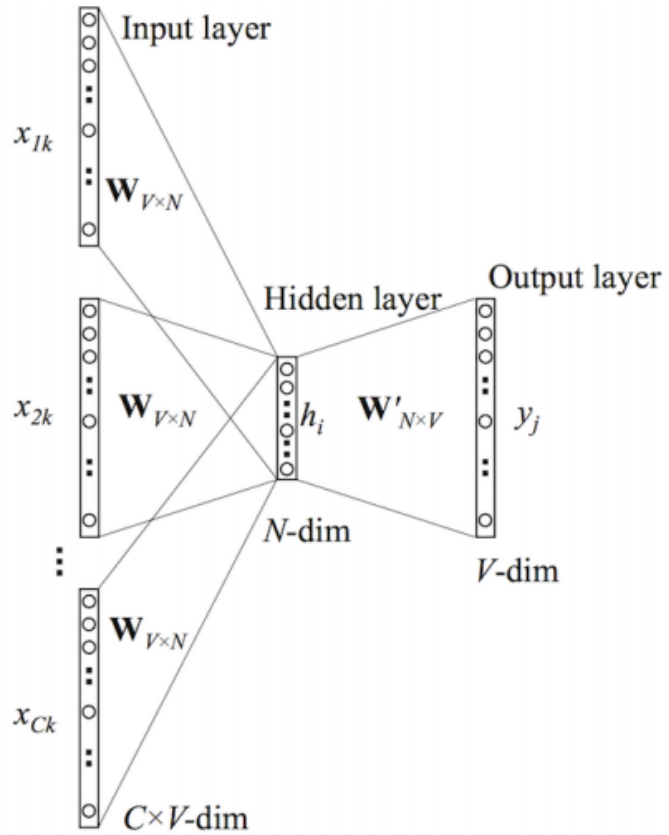
Skip Gram - Diagram

Below is the diagram of the skip gram model :



CBOW - Diagram

Below is the diagram of the *CBOW* model, which is like the Skip-Gram Model with the input and output reversed. So, the objective is to predict a center word from the surrounding context.



Thanks for your attention