L'intelligence artificielle dans la détection de la dépression

Nardi XHEPI

Dépression dans le monde

- 300 millions de personnes
- une augmentation de 18% entre 2005 et 2015

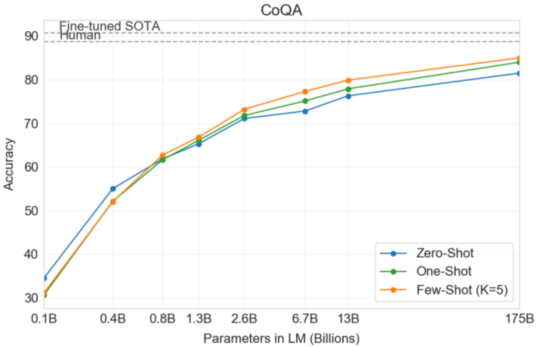
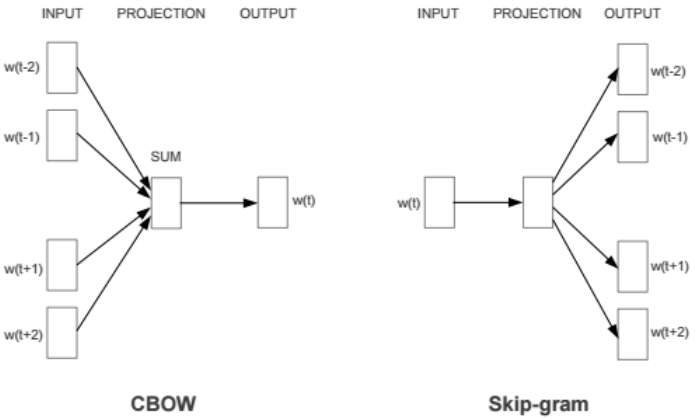


Figure 3.7: GPT-3 results on CoQA reading comprehension task. GPT-3 175B achieves 85 F1 in the few-shot setting, only a few points behind measured human performance and state-of-the-art fine-tuned models. Zero-shot and one-shot performance is a few points behind, with the gains to few-shot being largest for bigger models.

Comment rendre un texte compréhensible pour un ordinateur?

- Les vecteurs, une représentation mathématique adaptée en IA
- Méthodes de plongement lexical et espace de vecteurs de mots

Plongement lexical



[Source]

Plongement lexical

Modèle utilisé pour le plongement lexical:

- Fasttext, algorithme pré-entraîné pour la langue française sur un large corpus Wikipedia
- Implémentation par le biais de la librairie gensim:

```
from gensim.models import KeyedVectors
model = KeyedVectors.load_word2vec_format("./cc.fr.300.vec")
```

Nettoyer les données

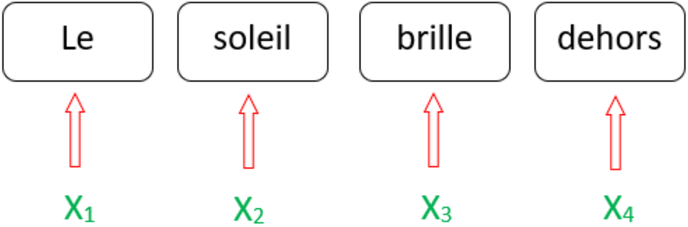
```python

# Enlever la ponctuation
def remove_punctuation(text):
    s = ""
    for i in text:
        if i not in string.punctuation:
            s += i
        elif i == "'":
            s += " "
    return s


# Convertir en minuscules
def lower_text(text):
    return text.lower()

```

Vectorisation des mots



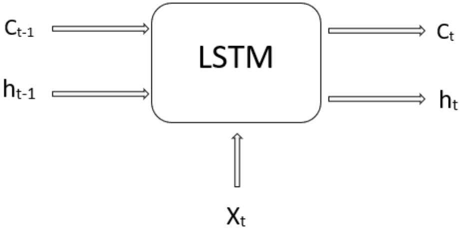
Vectorisation des mots

```python
def words_to_vect(liste):
    L = []
    for d in liste:
        temp = []
        x = remove_punctuation(d[0])
        x = lower_text(x)
        mots = x.split()

        try:
            for mot in mots:
                temp.append(np.array([list(model[mot])]).T)
            L.append((temp, np.array([[float(d[1] == 1)], [
float(d[1] == 0)]])))
        except:
            pass

    return L
```

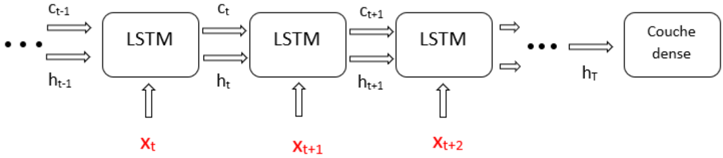
Représentation graphique



Structure d'un LSTM

Réseau utilisé

Pour un texte quelconque $p = [x_1, ..., x_T]$:



Notations

- $x_t \in \mathbb{R}^n$: vecteur du mot en entrée de la cellule LSTM
- $h_t \in \mathbb{R}^m$: vecteur de sortie
- $c_t \in \mathbb{R}^m$: vecteur de l'état de la cellule
- $f_t \in [0,1]^m$: vecteur d'activation de la porte oubli
- $i_t \in [0,1]^m$: vecteur d'activation de la porte entrée
- $o_t \in [0,1]^m$: vecteur d'activation de la porte sortie
- Fonction sigmoid : $\sigma : x \in \mathbb{R} \mapsto \dfrac{1}{1+e^{-x}}$
- $W \in \mathcal{M}_{m,n}(\mathbb{R}), U \in \mathcal{M}_{m,m}(\mathbb{R})$, et $b \in \mathcal{M}_{m,1}(\mathbb{R})$
- $\odot$: produit de Hadamard

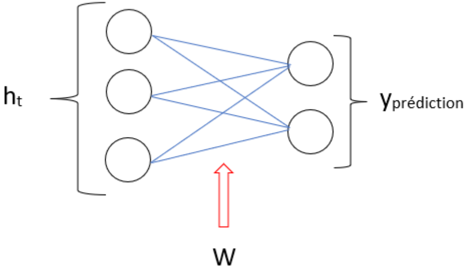
Définition formelle d'une cellule LSTM

- $f_t = \sigma(W_f * x_t + U_f * h_{t-1} + b_f)$
- $i_t = \sigma(W_i * x_t + U_i * h_{t-1} + b_i)$
- $o_t = \sigma(W_o * x_t + U_o * h_{t-1} + b_o)$
- $\tilde{c}_t = \tanh(W_c * x_t + U_c * h_{t-1} + b_c)$
- $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$
- $h_t = o_t \odot \tanh(c_t)$

Définition de la couche dense

Définition formelle

$y_{\text{prédiction}} = \sigma(Wh_T + b)$ où $W \in \mathcal{M}_{2,m}(\mathbb{R})$ et $b \in \mathcal{M}_{2,1}(\mathbb{R})$



Fonction d'erreur

$$C = \frac{1}{2} \sum_{i=1}^{p} (y_{\text{prédiction},i} - y_{\text{cible},i})^2$$

Descente du gradient

Soit $(W_k) \in (\mathcal{M}_{p,q}(\mathbb{R}))^{\mathbb{N}}$ la suite définie par :

$$W_{k+1} = W_k - \alpha \frac{\partial C}{\partial W_k}$$

où:

$$\alpha \text{ : le pas}$$
$$\frac{\partial C}{\partial W_k} = \left(\frac{\partial C}{\partial \left(w_k\right)_{ij}} \right)_{ij}$$

Alors (W_k) est convergente.

Calcul des gradients

Théorème 1

A une étape quelconque t:

$$\frac{\partial C}{\partial W_o} = \left[\frac{\partial C}{\partial h_t} \odot \tanh(c_t) \odot o_t \odot (1 - o_t)\right] x_t^\mathsf{T} \tag{1}$$

$$\frac{\partial C}{\partial W_i} = \left[\frac{\partial C}{\partial h_t} \odot o_t \odot (1 - \tanh^2(c_t)) \odot \tilde{c}_t \odot i_t \odot (1 - i_t)\right] x_t^\mathsf{T} \tag{2}$$

$$\frac{\partial C}{\partial W_f} = \left[\frac{\partial C}{\partial h_t} \odot o_t \odot (1 - \tanh^2(c_t)) \odot c_{t-1} \odot f_t \odot (1 - f_t)\right] x_t^\mathsf{T} \tag{3}$$

$$\frac{\partial C}{\partial W_c} = \left[\frac{\partial C}{\partial h_t} \odot o_t \odot (1 - \tanh^2(c_t)) \odot i_t \odot (1 - \tilde{c}_t{}^2)\right] x_t^\mathsf{T} \tag{4}$$

Démonstration

```
    for t in range(T-1, -1, -1):
        (i, f, o, _c, c, x, h_prec, c_prec) = self.time_steps[t]
        d_c = err_h * o * (1.0 - tanh(c) ** 2)

        delta_f = d_c * c_prec * f * (1 - f)
        delta_i = d_c * _c * i * (1 - i)
        delta_o = err_h * tanh(c) * o * (1 - o)
        delta_c = d_c * i * (1.0 - _c ** 2)

        grad_w_f = delta_f.dot(x.T)
        grad_u_f = delta_f.dot(h_prec.T)
        grad_b_f = delta_f

        grad_w_i = delta_i.dot(x.T)
        grad_u_i = delta_i.dot(h_prec.T)
        grad_b_i = delta_i

        grad_w_o = delta_o.dot(x.T)
        grad_u_o = delta_o.dot(h_prec.T)
        grad_b_o = delta_o

        grad_w_c = delta_c.dot(x.T)
        grad_u_c = delta_c.dot(h_prec.T)
        grad_b_c = delta_c
```

```
        err_h = 1/4 * ((self.Uf.T).dot(delta_f) + (self.Ui.T).dot(
    delta_i) + (self.Uo.T).dot(delta_o) + (self.Uc.T).dot(delta_c))
```

Implémentation et entraînement

```
network = LSTM_Model()

def train(network, data_x, data_y, alpha = 0.01):
    for _ in range(10000):
        network.train(data_x, data_y, alpha)
```

Quelques résultats


```
>>> isdepressive("Tout parait si futile face a cette douleur, cette souffrance")
('Dépression', 'probabilité: 0.9970403156587623')
```

Quelques résultats

```
>>> isdepressive("Je n'arrive plus à supporter le travail, je suis de plus
en plus anxieux ces derniers temps")
('Dépression', 'probabilité: 0.5348114742725703')

>>> isdepressive("Même si des fois les choses vont mal, je réussis à rester
heureux")
('Pas de dépression', 'probabilité: 0.7129678212805912')
```

Applications possibles

- Implémentation dans les réseaux sociaux
- Implémentation dans les assistants personnels d'intelligence artificielle

Merci pour votre attention!

Annexe

- Démonstration du théorème 1
- Théorème 2
- Théorème 3
- Théorème 4
- Théorème 5
- Source code

Démonstration et autres théorèmes des calculs des gradients

Démonstration théorème 1

Nous allons démontrer l'égalité (1) du théorème 1, le reste se démontrant de manière analogue. Posons $z_o^t = W_o x_t + U_o h_{t-1} + b_o$. De plus:

$$W_o \to z_o^t \to o_t \to h_t \to ... \to C$$

Ce qui nous permet de calculer $\dfrac{\partial C}{\partial W_o}$ en appliquant successivement la règle de la chaîne.

Démonstration des calculs des gradients et autres théorèmes

Démonstration théorème 1

En effet:

$$\forall i \in [\![1, m]\!],\ \frac{\partial C}{\partial \left(o_t\right)_i} = \sum_{j=1}^{m} \frac{\partial C}{\partial \left(h_t\right)_j} \frac{\partial \left(h_t\right)_j}{\partial \left(o_t\right)_i}$$

$$ie\ \forall i \in [\![1, m]\!],\ \frac{\partial C}{\partial \left(o_t\right)_i} = \frac{\partial C}{\partial \left(h_t\right)_i} \tanh((c_t)_i)$$

D'où, $\dfrac{\partial C}{\partial o_t} = \dfrac{\partial C}{\partial h_t} \odot \tanh(c_t)$

Démonstration des calculs des gradients et autres théorèmes

Démonstration théorème 1

De plus:

$$\forall i \in [\![1, m]\!],\ \frac{\partial C}{\partial \left(z_o^t\right)_i} = \sum_{j=1}^{m} \frac{\partial C}{\partial \left(o_t\right)_j} \frac{\partial \left(o_t\right)_j}{\partial \left(z_o^t\right)_i}$$

$$ie\ \forall i \in [\![1, m]\!],\ \frac{\partial C}{\partial \left(z_o^t\right)_i} = \frac{\partial C}{\partial \left(o_t\right)_i} \sigma'(\left(z_o^t\right)_i) = \frac{\partial C}{\partial \left(o_t\right)_i} \sigma(\left(z_o^t\right)_i)(1 - \sigma(\left(z_o^t\right)_i)$$

D'où, $\frac{\partial C}{\partial z_o^t} = \frac{\partial C}{\partial o_t} \odot o_t \odot (1 - o_t)$

Démonstration des calculs des gradients et autres théorèmes

Démonstration théorème 1

Enfin:

$$\forall i \in [\![1, m]\!],\ \frac{\partial C}{\partial \left(w_o\right)_{ij}} = \sum_{k=1}^{m} \frac{\partial C}{\partial \left(z_o^t\right)_k} \frac{\partial \left(z_o^t\right)_k}{\partial \left(w_o\right)_{ij}}$$

$$ie\ \forall i \in [\![1, m]\!],\ \frac{\partial C}{\partial \left(w_o\right)_{ij}} = \frac{\partial C}{\partial \left(z_o^t\right)_i} \frac{\partial \left(z_o^t\right)_i}{\partial \left(w_o\right)_{ij}}$$

$$ie\ \forall i \in [\![1, m]\!],\ \frac{\partial C}{\partial \left(w_o\right)_{ij}} = \frac{\partial C}{\partial \left(z_o^t\right)_i} \left(x_t\right)_j$$

Démonstration des calculs des gradients et autres théorèmes

Démonstration théorème 1

Finalement:

$$\frac{\partial C}{\partial W_o} = \frac{\partial C}{\partial z_o^t} x_t^\intercal$$

$$ie \ \frac{\partial C}{\partial W_o} = \left[\frac{\partial C}{\partial h_t} \odot \tanh(c_t) \odot o_t \odot (1 - o_t) \right] x_t^\intercal$$

Calcul des gradients

Théorème 2

A une étape quelconque t:

$$\frac{\partial C}{\partial U_o} = \left[\frac{\partial C}{\partial h_t} \odot \tanh(c_t) \odot o_t \odot (1 - o_t) \right] h_{t-1}^{\mathsf{T}} \tag{1}$$

$$\frac{\partial C}{\partial U_i} = \left[\frac{\partial C}{\partial h_t} \odot o_t \odot (1 - \tanh^2(c_t)) \odot \tilde{c}_t \odot i_t \odot (1 - i_t) \right] h_{t-1}^{\mathsf{T}} \tag{2}$$

$$\frac{\partial C}{\partial U_f} = \left[\frac{\partial C}{\partial h_t} \odot o_t \odot (1 - \tanh^2(c_t)) \odot c_{t-1} \odot f_t \odot (1 - f_t) \right] h_{t-1}^{\mathsf{T}} \tag{3}$$

$$\frac{\partial C}{\partial U_c} = \left[\frac{\partial C}{\partial h_t} \odot o_t \odot (1 - \tanh^2(c_t)) \odot i_t \odot (1 - \tilde{c}_t{}^2) \right] h_{t-1}^{\mathsf{T}} \tag{4}$$

Calcul des gradients

Théorème 3

A une étape quelconque t:

$$\frac{\partial C}{\partial b_o} = \frac{\partial C}{\partial h_t} \odot \tanh(c_t) \odot o_t \odot (1 - o_t) \tag{1}$$

$$\frac{\partial C}{\partial b_i} = \frac{\partial C}{\partial h_t} \odot o_t \odot (1 - \tanh^2(c_t)) \odot \tilde{c}_t \odot i_t \odot (1 - i_t) \tag{2}$$

$$\frac{\partial C}{\partial b_f} = \frac{\partial C}{\partial h_t} \odot o_t \odot (1 - \tanh^2(c_t)) \odot c_{t-1} \odot f_t \odot (1 - f_t) \tag{3}$$

$$\frac{\partial C}{\partial b_c} = \frac{\partial C}{\partial h_t} \odot o_t \odot (1 - \tanh^2(c_t)) \odot i_t \odot (1 - \tilde{c}_t{}^2) \tag{4}$$

Calcul des gradients

Théorème 4

A une étape quelconque t, on montre grâce à la règle de la chaîne que:

$$\frac{\partial C}{\partial h_{t-1}} = \frac{1}{4} \sum_{k} U_k^{\mathsf{T}} \frac{\partial C}{\partial z_k^t}$$

où $k \in \{f, i, o, c\}$.
On applique ensuite les théorèmes 1, 2 et 3 à l'étape $t-1$.

Calcul des gradients

Théorème 5

Les gradients par rapport à chaque poids de la cellule LSTM sont alors:

$$\frac{\partial C}{\partial P_c} = \frac{1}{T} \sum_{t=1}^{T} \left(\frac{\partial C}{\partial P_c} \right)_t$$

$$\frac{\partial C}{\partial P_f} = \frac{1}{T} \sum_{t=1}^{T} \left(\frac{\partial C}{\partial P_f} \right)_t$$

$$\frac{\partial C}{\partial P_i} = \frac{1}{T} \sum_{t=1}^{T} \left(\frac{\partial C}{\partial P_i} \right)_t$$

$$\frac{\partial C}{\partial P_o} = \frac{1}{T} \sum_{t=1}^{T} \left(\frac{\partial C}{\partial P_o} \right)_t$$

où $P \in \{W, U, b\}$

Source code I

```
import numpy as np
import pickle
from random import randint, shuffle
import string
from gensim.models import FastText

class Linear:
    def __init__(self, number_of_entries, number_of_neurons,
    activation_function):
        self.weights = np.random.randn(number_of_neurons,
    number_of_entries) * np.sqrt(1/number_of_neurons)
        self.biais = np.zeros((number_of_neurons, 1))
        self.activation_function = activation_function

    def act_fun(self, Z):
        if self.activation_function == "sigmoid" :
            return 1.0 / (1.0 + np.exp(-Z))
        elif self.activation_function == "relu":
            return np.maximum(0, Z)
        elif self.activation_function == "arctan":
            return np.arctan(Z)/np.pi + 0.5
        elif self.activation_function == "tanh":
            return np.tanh(Z)
```

Source code II

```
        return Z

    def deriv_act_fun(self, Z):
        if self.activation_function == "sigmoid" :
            d = 1.0 / (1.0 + np.exp(-Z))
            return d * (1 - d)
        elif self.activation_function == "relu":
            Z[Z > 0] = 1
            Z[Z <= 0] = 0
            return Z
        elif self.activation_function == "arctan":
            return 1/(np.pi*(1 + Z ** 2))
        elif self.activation_function == "tanh":
            return 1 - np.tanh(Z) ** 2
        return 1

    def forward(self, x):
        self.layer_before_activation = []
        self.layer_after_activation = []
        x = self.weights.dot(x) + self.biais
        self.layer_before_activation.append(x)
        x = self.act_fun(x)
        self.layer_after_activation.append(x)
```

Source code III

```
        return x

    def backward(self, previous_layer, delta_l_1, eta):
        delta_l = np.dot(self.weights.T, delta_l_1)* previous_layer
    .deriv_act_fun(previous_layer.layer_before_activation[0])
        grad_weights = delta_l_1 * previous_layer.
    layer_after_activation[0].T
        grad_biais = delta_l_1

        self.weights -= eta * grad_weights
        self.biais -= eta * grad_biais

        return delta_l

    def backward_first_layer(self, x, err, eta):
        grad_weights = err * x.T
        grad_biais = err

        delta_1 = (self.weights.T).dot(err)

        self.weights -= eta * grad_weights
        self.biais -= eta * grad_biais

```

Source code IV

```python
        return delta_1

## Fonctions d'activation et leurs derivees

def sigmoid(Z):
    return 1.0 / (1.0 + np.exp(-Z))

def tanh(Z):
    return np.tanh(Z)

def deriv_sigmoid(Z):
    d = 1.0 / (1.0 + np.exp(-Z))
    return d * (1 - d)

def deriv_tanh(Z):
    return 1.0 - np.tanh(Z) ** 2


## Creation d'une cellule LSTM

class LSTM :
    def __init__(self, x_length, h_length):
        self.x_length = x_length
```

Source code V

```python
        self.h_length = h_length

        k = np.sqrt(1/self.h_length)

        #timesteps
        self.time_steps = []

        #gradients
        self.gradients_w_i_moyen = np.zeros((self.h_length, self.x_length))
        self.gradients_w_f_moyen = np.zeros((self.h_length, self.x_length))
        self.gradients_w_o_moyen = np.zeros((self.h_length, self.x_length))
        self.gradients_w_c_moyen = np.zeros((self.h_length, self.x_length))

        self.gradients_u_i_moyen = np.zeros((self.h_length, self.h_length))
        self.gradients_u_f_moyen = np.zeros((self.h_length, self.h_length))
        self.gradients_u_o_moyen = np.zeros((self.h_length, self.h_length))
```

Source code VI

```
        self.gradients_u_c_moyen = np.zeros((self.h_length, self.h_length))


        self.gradients_b_i_moyen = np.zeros((self.h_length, 1))
        self.gradients_b_f_moyen = np.zeros((self.h_length, 1))
        self.gradients_b_o_moyen = np.zeros((self.h_length, 1))
        self.gradients_b_c_moyen = np.zeros((self.h_length, 1))

        #input gate
        self.Wi = np.random.randn(self.h_length, self.x_length) * k
        self.Ui = np.random.randn(self.h_length, self.h_length) * k
        self.bi = np.random.randn(self.h_length, 1) * k


        #forget gate
        self.Wf = np.random.randn(self.h_length, self.x_length) * k
        self.Uf = np.random.randn(self.h_length, self.h_length) * k
        self.bf = np.random.randn(self.h_length, 1) * k

        #out gate
        self.Wo = np.random.randn(self.h_length, self.x_length) * k
        self.Uo = np.random.randn(self.h_length, self.h_length) * k
```

Source code VII

```
        self.bo = np.random.randn(self.h_length, 1) * k

        #cell memory
        self.Wc = np.random.randn(self.h_length, self.x_length) * k
        self.Uc = np.random.randn(self.h_length, self.h_length) * k
        self.bc = np.random.randn(self.h_length, 1) * k


    def forward(self, x_liste):

        h_prec, c_prec = np.zeros((self.h_length, 1)), np.zeros((
    self.h_length, 1))

        for x in x_liste:

            z_i_t = self.Wi.dot(x) + self.Ui.dot(h_prec) + self.bi
            z_f_t = self.Wf.dot(x) + self.Uf.dot(h_prec) + self.bf
            z_o_t = self.Wo.dot(x) + self.Uo.dot(h_prec) + self.bo
            z_c_t = self.Wc.dot(x) + self.Uc.dot(h_prec) + self.bc

            i = sigmoid(z_i_t)
            f = sigmoid(z_f_t)
            o = sigmoid(z_o_t)
```

Source code VIII

```
            _c = tanh(z_c_t)
            c = f * c_prec + i * _c
            h = o * tanh(c)


            self.time_steps.append((i, f, o, _c, c, x, h_prec,
    c_prec))

            (h_prec, c_prec) = (h, c)

        return (h_prec, c_prec)


    def backward_propagation(self, err,eta):
        T = len(self.time_steps)

        err_h = err

        t = T-1

        while t > T - 10 and t >= 0:
            (i, f, o, _c, c, x, h_prec, c_prec) = self.time_steps[t
    ]
```

Source code IX

```
            d_c = err_h * o * (1.0 - tanh(c) ** 2)

            delta_f = d_c * c_prec * f * (1 - f)
            delta_i = d_c * _c * i * (1 - i)
            delta_o = err_h * tanh(c) * o * (1 - o)
            delta_c = d_c * i * (1.0 - _c ** 2)

            grad_w_f = delta_f.dot(x.T)
            grad_u_f = delta_f.dot(h_prec.T)
            grad_b_f = delta_f

            grad_w_i = delta_i.dot(x.T)
            grad_u_i = delta_i.dot(h_prec.T)
            grad_b_i = delta_i

            grad_w_o = delta_o.dot(x.T)
            grad_u_o = delta_o.dot(h_prec.T)
            grad_b_o = delta_o

            grad_w_c = delta_c.dot(x.T)
            grad_u_c = delta_c.dot(h_prec.T)
```

Source code X

```
            grad_b_c = delta_c

            k = (T - 1 - t)
            a = 1/(T-t)

            self.gradients_w_i_moyen = a * (k * self.gradients_w_i_moyen + grad_w_i)
            self.gradients_w_f_moyen = a * (k * self.gradients_w_f_moyen + grad_w_f)
            self.gradients_w_c_moyen = a * (k * self.gradients_w_c_moyen + grad_w_c)
            self.gradients_w_o_moyen = a * (k * self.gradients_w_o_moyen + grad_w_o)


            self.gradients_u_i_moyen = a * (k * self.gradients_u_i_moyen + grad_u_i)
            self.gradients_u_f_moyen = a * (k * self.gradients_u_f_moyen + grad_u_f)
            self.gradients_u_c_moyen = a * (k * self.gradients_u_c_moyen + grad_u_c)
            self.gradients_u_o_moyen = a * (k * self.gradients_u_o_moyen + grad_u_o)
```

Source code XI

```
            self.gradients_b_i_moyen = a * (k * self.
    gradients_b_i_moyen + grad_b_i)
            self.gradients_b_f_moyen = a * (k * self.
    gradients_b_f_moyen + grad_b_f)
            self.gradients_b_c_moyen = a * (k * self.
    gradients_b_c_moyen + grad_b_c)
            self.gradients_b_o_moyen = a * (k * self.
    gradients_b_o_moyen + grad_b_o)


            err_h = 1/4 * ((self.Uf.T).dot(delta_f) + (self.Ui.T).
    dot(delta_i) + (self.Uo.T).dot(delta_o) + (self.Uc.T).dot(
    delta_c))


            t-=1

        self.Wc -= eta * self.gradients_w_c_moyen
        self.Wf -= eta * self.gradients_w_f_moyen
        self.Wi -= eta * self.gradients_w_i_moyen
        self.Wo -= eta * self.gradients_w_o_moyen
```

Source code XII

```
        self.Uc -= eta * self.gradients_u_c_moyen
        self.Uf -= eta * self.gradients_u_f_moyen
        self.Ui -= eta * self.gradients_u_i_moyen
        self.Uo -= eta * self.gradients_u_o_moyen


        self.bc -= eta * self.gradients_b_c_moyen
        self.bf -= eta * self.gradients_b_f_moyen
        self.bi -= eta * self.gradients_b_i_moyen
        self.bo -= eta * self.gradients_b_o_moyen

        self._reset_gradients()
        self.time_steps = []





    def _reset_gradients(self):
        self.gradients_w_i_moyen = np.zeros((self.h_length, self.
    x_length))
        self.gradients_w_f_moyen = np.zeros((self.h_length, self.
    x_length))
```

Source code XIII

```
        self.gradients_w_o_moyen = np.zeros((self.h_length, self.x_length))
        self.gradients_w_c_moyen = np.zeros((self.h_length, self.x_length))

        self.gradients_u_i_moyen = np.zeros((self.h_length, self.h_length))
        self.gradients_u_f_moyen = np.zeros((self.h_length, self.h_length))
        self.gradients_u_o_moyen = np.zeros((self.h_length, self.h_length))
        self.gradients_u_c_moyen = np.zeros((self.h_length, self.h_length))


        self.gradients_b_i_moyen = np.zeros((self.h_length, 1))
        self.gradients_b_f_moyen = np.zeros((self.h_length, 1))
        self.gradients_b_o_moyen = np.zeros((self.h_length, 1))
        self.gradients_b_c_moyen = np.zeros((self.h_length, 1))
```

Source code XIV

```
## LSTM Class model

class LSTM_Model:
    def __init__(self):
        self.lstm = LSTM(300, 10)
        self.dense_layer = Linear(10, 2, "sigmoid")



    def MSE(self, x, y):
        return 1/2 * np.sum(np.square(x - y))

    def deriv_MSE(self, x, y):
        return (x - y)


    def predict(self, x_list):
        _x = self.lstm.forward(x_list)
        h = _x[0]
        _x = self.dense_layer.forward(_x[0])
        return (h, _x)


```

Source code XV

```python
    def train(self, X, Y, eta):
        n = len(X)
        s = 0
        for i in range(n):
            x_list = X[i]
            (h, y_prediction) = self.predict(x_list)

            s += self.MSE(y_prediction, Y[i])

            delta_l_1 = self.deriv_MSE(y_prediction, Y[i]) * self.dense_layer.deriv_act_fun(self.dense_layer.layer_before_activation[0])
            err = self.dense_layer.backward_first_layer(h, delta_l_1, eta)

            self.lstm.backward_propagation(err, eta)

        print(s/n)

##
def save_data(liste, unique = False):

    l = liste
```

Source code XVI

```

    saved = open_list()

    if unique:
        l = []

        for data in liste:
            if data not in saved:
                l.append(data)

    with open('C:\\Users\\nardi\\Desktop\\TIPE\\Data\\data_3_7.pickle', 'wb') as f:
        pickle.dump(saved + l, f)



def open_list():
    with open('C:\\Users\\nardi\\Desktop\\TIPE\\Data\\data_3_7.pickle', 'rb') as f:
        b = pickle.load(f)

    return b

```

Source code XVII

```
## Text cleaning

# Enlever la ponctuation
def remove_punctuation(text):
    s = ""
    for i in text:
        if i not in string.punctuation:
            s += i
        elif i == "’":
            s += " "
    return s


# Lower text
def lower_text(text):
    return text.lower()

## Vector representation of data

from gensim.models import KeyedVectors
model = KeyedVectors.load_word2vec_format("C:\\Users\\nardi\\
    Desktop\\TIPE\\Data\\cc.fr.300.vec")

```

Source code XVIII

```python

## Vectorise

def words_to_vect(liste):
    L = []
    for d in liste:
        temp = []
        x = remove_punctuation(d[0])
        x = lower_text(x)
        phrase = x.split()

        try:
            for word in phrase:
                temp.append(np.array([list(model[word])]).T)
            L.append((temp, np.array([[float(d[1] == 1)], [float(d[1] == 0)]])))
        except:
            pass

    return L


data = words_to_vect(open_list())
```

Source code XIX

```
shuffle(data)
x = [d[0] for d in data]
y = [d[1] for d in data]



## Network model

network = LSTM_Model()


def train(network, data_x, data_y, alpha = 0.01):

    for _ in range(10000):
        network.train(data_x, data_y, alpha)



## Test prediction

def isdepressive(sentence):
    x = remove_punctuation(sentence)
    x = lower_text(x)
```

Source code XX

```
    phrase = x.split()

    temp = []
    for word in phrase:
        temp.append(np.array([list(model[word])]).T)

    p = network.predict(temp)[1]

    if p[0] > p[1]:
        return "Depression", "probabilite: {}".format(p[0][0])
    else:
        return "Pas de depression", "probabilite: {}".format(p
    [1][0])


```