Accuracy de tous les résultats

## Baseline : Notebook Version 1 avec dictionnaire du cours et data (pas de preprocessing)

- Learning rate = 0.0005

Accuracy\_train = 60%

Accuracy Kaggle =

- Learning rate = 0.005

Accuracy\_train = 61% puis stagne

Accuracy Kaggle = 61%

- Learning rate = 0.05

Accuracy\_train = 0.52% puis stagne

Defaults: apprend très vite puis stagne. Peu importe le learning rate. Apprend trop lentement car SGD.

Idées pour le moment: plus de pre-processing pour avoir quelque chose de plus robuste

## Notebook Version 1 avec dictionnaire du cours et data ( avec preprocessing) 🡪 le preprocessing n’était pas le bon

- Learning rate = 0.005

Accuracy\_train = 0.56

- Learning rate = 0.0005 -0.00075 - 0.001

Accuracy\_train = 0.57

- Learning rate = 0.05

Accuracy\_train = 0.50

## Notebook Version 1 avec dictionnaire du cours et data (avec preprocessing mais sans enlever stopwords) 🡪 le preprocessing n’était pas le bon

- Learning rate = 0.005

Accuracy\_train = 0.594

- Learning rate = 0.001

Accuracy\_train = 0.589

- Learning rate = 0.0005

Accuracy\_train = 0.605

- Learning rate = 0.0001

Accuracy\_train = 0.584

Idées: utiliser un autre dictionnaire est peut-être mieux.

## Notebook Version 3 avec dictionnaire de twitter et data (avec preprocessing)

JEROME ET AUDREY

- Learning rate = 0.005

Accuracy\_train =

- Learning rate = 0.05

Accuracy\_train =

Defaults, moyenne = pas une variable très stable,

Deux phrases très différentes peuvent avoir le même moyenne et pas dire la même chose ex: "Je l'aime à mourir" / "J'aimerais mourir"

Prend pas en compte ordre des mots et grande importance dans phrase en fait: "Je suis terriblement..." "heureux" ou "inquiet"

Idée: RNN and in particular, LSTM

## Notebook LSTM glove 50d.

PREPROCESSING: AUCUN

(not data full, only train\_pos and train\_neg)

--> early stopping

- Test on Kaggle: 0.49

## Notebook LSTM glove 50d

PREPROCESSING: CHIFFRES + SIGNES

(not data full, only train\_pos and train\_neg)

--> early stopping

- Train: 0.82

- Test on Kaggle: 0.773 (3h20)

## Notebook LSTM glove 50d

PREPROCESSING: CHIFFRES + SIGNES + NLTK STOPWORDS

--> 50 epochs

- Train: 0.92

- Test on Kaggle: 0.75 (12h) Model took 800s \* 50 to train

--> early stopping at epochs

- Train:

- Test on Kaggle: 0.777

in this case, kept history

Model took 6995.441886663437 seconds to train (~2h)

==> enlever les stopwords ne change pas l'accuracy mais algorithme plus rapide

(j'ai oublié de preprocess le test pareil. mais ça devrait rien changer je pense. pas sûre à 100%)

## Notebook LSTM glove 50d.

PREPROCESSING: CHIFFRES + SIGNES + NLTK STOPWORDS

--> early stopping (2h)

## LOSS: binary crossentropy (instead of categorical crossentropy)

Test on Kaggle: 0.781

## Notebook LSTM glove 50d.

PREPROCESSING: CHIFFRES + SIGNES + NLTK STOPWORDS

Loss: binary crossentropy

--> early stopping

OPTIMIZER= on avait ‘adam’ jusque-là.

- Adam: 2h

Test on Kaggle : 0.781

- Sgd: 6h

Test on Kaggle: 0.76 un peu moins bon et beaucoup plus lent

- RMSprop: 4h

Test on Kaggle 0.779

-Adagrad: 2.6h

Test on Kaggle 0.78

-Adadelta: 2.3h

Test on Kaggle 0.776

-Adamax: 2.5h

Test on Kaggle 0.774

-Nadam: 2.2h

Test on Kaggle 0.757

## Notebook LSTM glove 50d.

PREPROCESSING: CHIFFRES + SIGNES + NLTK STOPWORDS

Loss: binary crossentropy

Optimizer: adam

1 seul output new\_predict function (sigmoid and not softmax) (j’aurais du y penser en passant à binary crossentropy)

Test on Kaggle: 0.746. Moins bien donc on reprend softmax

## Notebook LSTM + glove dict.

PREPROCESSING: CHIFFRES + SIGNES + NLTK STOPWORDS

Loss: binary crossentropy

Optimizer: adam

Softmax

Dimensions du dictionnaire:

25d: Test on Kaggle:

50d: Test on Kaggle:

100d: Test on Kaggle:

200d: Test on Kaggle:

## Notebook LSTM\_twitt glove 50d.

PREPROCESSING: CHIFFRES + SIGNES + NLTK STOPWORDS

Loss: categorical crossentropy

Optimizer: adam

Softmax

Dictionnaire twitter:  
 25d: Test on Kaggle: 0.608  
 50d: Test on Kaggle: 0.802 – 65 minutes  
 100d: Test on Kaggle: 0.803 – 101 minutes  
 200d: Test on Kaggle: 0.813 – 57 minutes

## Notebook LSTM\_twitt glove 200d.

(les layers sont denses, pas lstm)

- 3layers: - sans reg: 0.813 //-avec reg 0.807

- 4 layers: sans reg: 0.807

## Notebook LSTM\_twitt glove 200d.

Dropout

0,75/0.25: 0.806

0.75/0.5: 0.805

0.8/0.8: 0.814

## Notebook LSTM\_twitt glove 200d.

( layers lstm)

- 3layers LSTM: de 64 neurones chacunes : 0.807

- 1 layer LSTM de 128 neurones: 0.801

TO TRY:

* dictionnaire de twitter
* full dataset

https://arxiv.org/pdf/1511.08630.pdf

password note hotcpr: aichyuiaeoiuoooeuow

* idée: embedding trainable = true