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)

- 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)

- 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

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

Test on Kaggle

## Notebook LSTM glove 50d.

PREPROCESSING: CHIFFRES + SIGNES + NLTK STOPWORDS

--> early stopping

OPTIMIZER= ['sgd', 'kullback\_leibler\_divergence', 'poisson', 'cosine\_proximity'] (on avait ‘adam’ jusque là).

'RMSprop', 'Adagrad', 'Adadelta', 'Adamax', 'Nadam'

- Sgd: 6h Test on Kaggle: 0.76 un peu moins bon et beaucoup plus lent

- RMSprop:

-Adagrad:

-Adadelta:

-Adamax:

-Nadam:

TO TRY:

optimizer = [ 'sgd', 'kullback\_leibler\_divergence', 'poisson', 'cosine\_proximity']

dictionnaire de twitter

full dataset

### try loss = binary\_crossentropy

password note hotcpr: aichyuiaeoiuoooeuow