NAOUMI Salmane - KILANI Al Houceine
salmane.naoumi@gmail.com
houcinekila@gmail.com

1 Loss :

Considering the loss function as follows :

$$\text{logloss} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log p_i + (1-y_i)\log\left(1-p_i\right)\right) \tag{1}$$

If the true label of a given observation is 1, then the loss is equal to $-\log(p_i)$, therefore the loss is almost equal to 0 for a confident correct prediction, and as the prediction becomes unsure or incorrect, the loss becomes bigger (tending to $+\infty$)

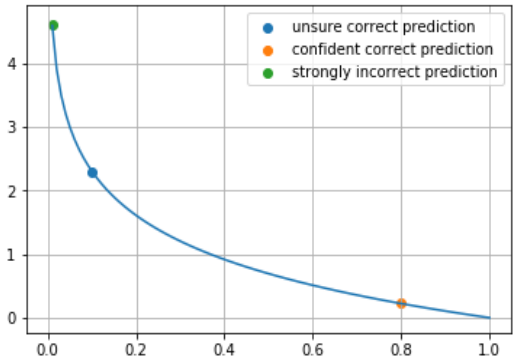

Figure 1: Logloss for an observation with label=1.

2 Convolutional layer :

Question 2:

The missing value in the Figure is **-3**. In fact:
$Output = Output1 + Output2 + b_0 = -3$
With $Output1 = INPUT_1[window[2]].FilterW0_1 =$ **-4**
And $Output2 = INPUT_2[window[2]].FilterW0_2 =$ **0**

where . is a dot product

Question 3 :

We can use a sigmoïd activation that outputs one value between 0 and 1 corresponding to the probability of the positive class: $f_\lambda(x) = \frac{1}{1+e^{-\lambda x}}$.

The final layer will have only one unit connected to all the neurons of the previous layer (The document embedding layer).

Question 4 :

There are only three parts of the convolutional neural network that contain trainable parameters :

- The embedding layer : where the number of parameters is equal to the embedding of the words in vocabulary plus the vectors for the padding and OOV words : $N_1 = (|V|+2)*d$ Where d is the dimension of the words embedding space and $|V|$ is the size of the vocabulary.

- The convolutional layer : As we only use one branch with n_f filters of size h, the number of parameters for the convolutional layer is equal to :$N_2 = n_f * (h * d + 1)$. The 1 is coming from biases of filters.

- The dense layer : linking the document embedding layer to the two final units (It can be linked to just one final unit also as seen in the previous question). The number of parameters in this layer is : $N_3 = 2 * (p + 1)$, Where p is the dimension of the docs embedding layer.

Finally :

$$\boxed{\mathrm{N}_p = N_1 + N_2 + N_3 = (|V| + 2) * d + n_f * (h * d + 1) + 2 * (p + 1)}$$

Question 5 :

While training the model, we see that it begins to overfit the data at the second/third epoch (Training loss/accuracy increasing while the validation loss/accuracy decreasing). We then decided to early stop the training at the second epoch.

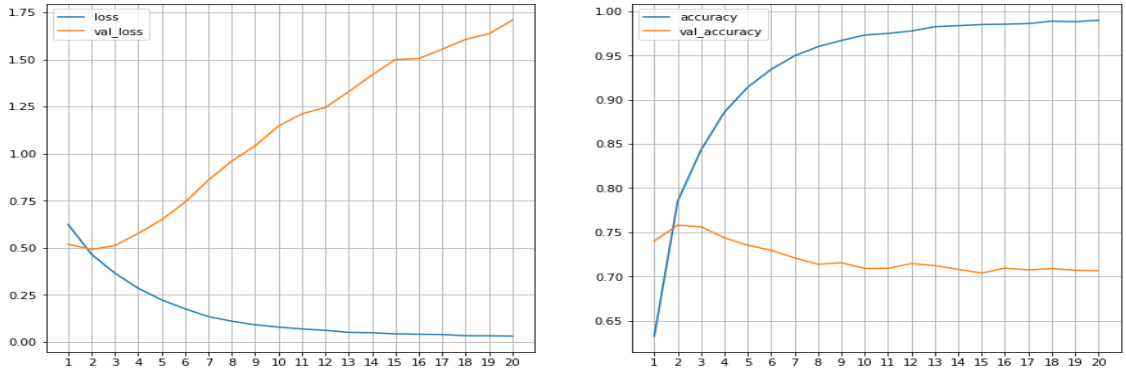

Figure 2: Evolution of the train/validation loss/accuracy

Plotting the document embedding in 2D using t-SNE, we got the following repartition of documents before and after the training wrt to their labels:

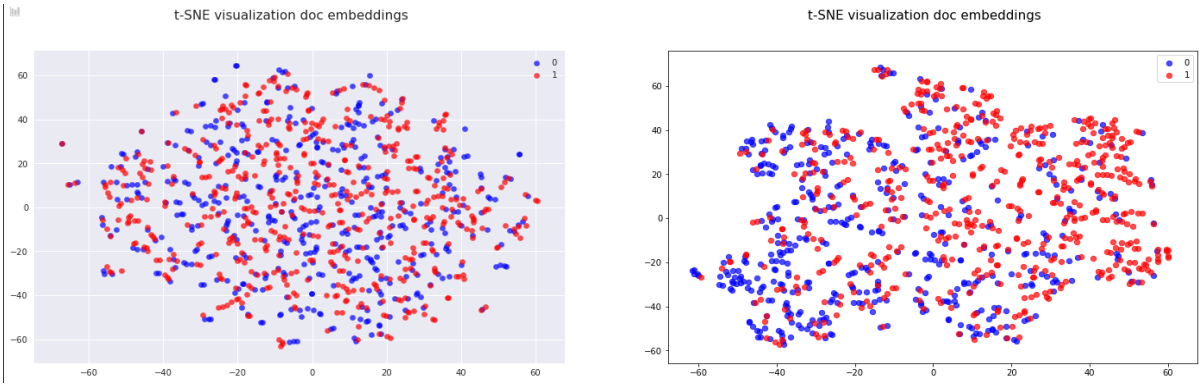

Figure 3: t-SNE visualization for doc embeddings

As we can see, the documents embeddings are not well separated before training which is normal since they are are randomly initialized. However, after training the CNN for some epochs it seems that they are kind of separated

Question 6 :

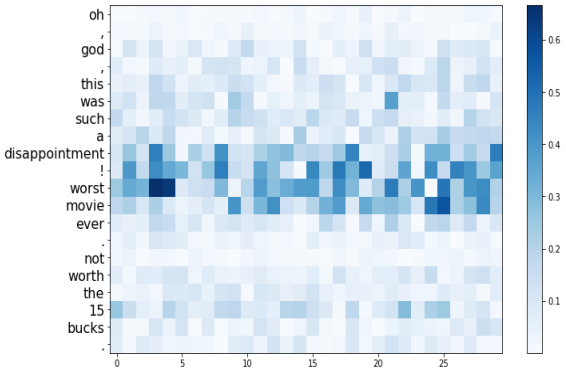

Figure 4: Obtained saliency map for the negative review

This saliency maps shows that the norm of the gradients of the probability distribution (here we took the probability of class 0) w.r.t the change of the input word (its vector representation) are high for the words : ”disappointment” and ”worst”.

This kind of makes sense, because imagine we replace those words by : ”beauty” and ”best”. In this case the prediction would be that this is a positive opinion on the movie.

```
In [56]: my_review_text = 'Oh , god , this was such a disappointment ! Worst movie ever . Not worth
the 15 bucks .'
   ...: #my_review_text = 'Oh , god , this was such a beauty ! Best movie ever . Not worth the 15
bucks .'
   ...:
   ...: tokens = my_review_text.lower().split()
   ...: my_review = [word_to_index[elt] for elt in tokens]
   ...:
   ...: # extract regions (sliding window over text)
   ...: regions = []
   ...: regions.append(' '.join(tokens[:filter_sizes[0]]))
   ...: for i in range(filter_sizes[0], len(tokens)):
   ...:     regions.append(' '.join(tokens[(i-filter_sizes[0]+1):(i+1)]))
   ...:
   ...:
   ...: my_review = np.array([my_review])

In [57]: testing(my_review)
Out[57]: [array([[0.9971533 , 0.00284664]], dtype=float32)]
```

```
In [53]: my_review_text = 'Oh , god , this was such a beauty ! Best movie ever . Not worth the 15
bucks .'

In [54]: tokens = my_review_text.lower().split()
   ...: my_review = [word_to_index[elt] for elt in tokens]
   ...:
   ...: # extract regions (sliding window over text)
   ...: regions = []
   ...: regions.append(' '.join(tokens[:filter_sizes[0]]))
   ...: for i in range(filter_sizes[0], len(tokens)):
   ...:     regions.append(' '.join(tokens[(i-filter_sizes[0]+1):(i+1)]))
   ...:
   ...:
   ...: my_review = np.array([my_review])
   ...: reg_emb = get_region_embedding([my_review,0])[0][0,:,:]

In [55]: testing(my_review)
Out[55]: [array([[0.07413432, 0.9258657 ]], dtype=float32)]
```

Figure 5: Impact on the probabilities when substituting disappointment by beauty and worst by best

Question 7 :

- In order for CNN to give good results, we need a big amount of data and high computational ressources.
- We verified it in our case, CNNs are easily overfitting (we have to add Batch normalization, Dropouts or use early stopping -that’s what we used in our case -)
- CNN are not well suited for tasks where length of text is important , they are more adapted to tasks like features detection.
- Even if we have some ways to interpret results (visualising feature maps, feature regions, saliency), it is not completely clear how the decision are made in the CNNs. And this lack of control and clear interpretability is a big drawback of the Neural networks in general.

References