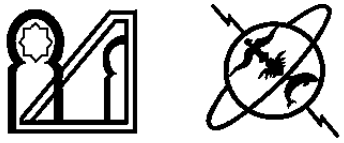


Université Mohammed V
Faculté des Sciences
Rabat

PROJET DE FIN DU MODULE « NATURAL LANGUAGE PROCESSING »

MASTER SPECIALISE EN « INGENIERIE DE DONNEES ET DEVELOPPEMENT LOGICIEL »

MINI PROJET SUR LA GENERATION DU TEXTE

REALISE PAR :
BARBACH CHAIMAA
AKEL IMAD

ENCADRE PAR :
MAHMOUDI ABDELHAK

ANNEE UNIVERSITAIRE 2020-2021

PREMIERE PARTIE
PROBLEMATIQUE

PROBLEMATIQUE

Le **NLP** pour **N**atural **L**anguage **P**rocessing ou Traitement Numérique du Langage est une discipline qui porte essentiellement sur la compréhension, la manipulation et la génération du langage naturel par les machines. Ainsi, le NLP est réellement à l'interface entre la science informatique et la linguistique. Il porte donc sur la capacité de la machine à interagir directement avec l'humain.

Si vous avez suivi l'actualité du traitement automatique du langage (TAL) ces derniers mois, vous avez surement entendu parler des Transformers, des modèles état-de-l'art qui apprennent des représentations vectorielles à partir d'un texte d'entrée et qui dépassent les capacités humaines sur certains points. Ces modèles peuvent être pré-entraînés et sont mis à disposition par Hugging Face, une startup spécialisée dans le traitement de langage naturel.

Les transformers sont particulièrement bons pour:

- la génération de texte ;
- apprendre une représentation vectorielle ;
- prédire si une phrase est la suite d'une autre ;
- les tâches de questions/réponses.

On a choisi de travailler sur la génération du texte pour la raison que La génération automatique de textes fait partie du traitement du langage naturel (TLN ou NLP pour Natural Language Processing), tout comme la compréhension du langage naturel (CLN ou NLU pour Natural Language Understanding). Ces techniques combinées trouvent des applications utiles dans les systèmes à personnalité artificielle, tels les chatbots et les assistants virtuels, qui interagissent avec les utilisateurs.

Vous allez voir dans les pages suivantes comment on a pu appliquer la génération du texte en utilisant GPT-2 d'OpenAI comme modèle et Panel comme étant Framework pour faire des applications web.

DEUXIEME PARTIE
CONFIGURATION DU MODELE

CONFIGURATION DU MODELE

Le GPT-2 d'OpenAI est un type de modèle de transformateur qui a suscité beaucoup d'intérêt quant à ses capacités à produire du texte de type humain. Si vous ne l'avez jamais expérimenté auparavant, vous repartirez probablement avec le même avis à la fin de ce rapport.

Chargement du modèle :
Tout d'abord, nous avions importé les packages requis :

```
In [1]: import numpy as np
        import torch
        import torch.nn.functional as F
        from transformers import GPT2Tokenizer, GPT2LMHeadModel
        from random import choice
```

Ensuite, nous avions chargé le tokenizer de GPT-2 et le modèle de langue : (le téléchargement du modèle pré-entraîné :

```
In [2]: tok = GPT2Tokenizer.from_pretrained("gpt2")
        model = GPT2LMHeadModel.from_pretrained("gpt2")
```

Fonction de prédiction :
A ce stade, la plupart du travail était déjà fait. Puisque notre modèle est pré-entraîné, nous n'avons pas besoin de l'entraîner ou d'apporter des modifications. Nous avons besoin d'écrire une fonction qui prend en paramètre le texte de départ, le modèle, tokenizer, et le p étant un nombre arbitraire entre 0 et 1. Et générer une prédiction sur le mot suivant.

```
In [3]: def get_pred(text, model, tok, p = 0.7):

            input_ids = torch.tensor(tok.encode(text)).unsqueeze(0) # Natural Language Processing is => [35364, 15417, 28403,   318]

            logits = model(input_ids)[0][:, -1] # [all columns, last line] vector of next token

            probs = F.softmax(logits, dim = -1).squeeze() # [2.9178e-05, 1.6191e-05, ..., 1.0366e-08, 1.6604e-05]

            idxs = torch.argsort(probs, descending = True) # [257, 262, 281, ..., 179, 22997, 36173]

            res = []
            cumsum = 0

            for idx in idxs:

                res.append(idx) # e.g : 257, ........, 1762

                cumsum += probs[idx] # e.g : 0.0009

                if cumsum > p:
                    pred_idx = idxs.new_tensor([choice(res)]) # e.g : 477
                    break

            pred = tok.convert_ids_to_tokens(int(pred_idx))

            return tok.convert_tokens_to_string(pred)
```

Il se passe beaucoup de choses dans cette fonction. Alors, décomposons-le. Tout d'abord, nous procédons à la tokenisation et à l'encodage du texte d'entrée à partir de *input_ids*.
Ensuite, nous demandons à notre modèle de générer un vecteur *logits* pour le prochain mot/jeton. Après avoir appliqué softmax et trié ces probabilités par ordre décroissant, nous avons un vecteur, *idxs*, qui répertorie les indices de chaque jeton de notre vocabulaire dans l'ordre de leurs probabilités respectives.
À ce stade, nous pourrions simplement choisir le jeton qui a la probabilité la plus élevée. Cependant, nous voulons pouvoir mélanger nos résultats afin que le même texte d'entrée puisse générer une variété de texte. Pour ce faire, nous allons ajouter un élément d'aléatoire où nous choisissons un jeton aléatoire parmi une liste des prochains jetons les plus probables. De cette façon, nous ne sélectionnons pas le même jeton prédit à chaque fois.
Nous effectuons cela en parcourant chaque probabilité jusqu'à ce que la somme de toutes les probabilités que nous avons bouclées soit supérieure à *p*, un nombre arbitraire entre 0 et 1. Tous les tokens itérés jusqu'à ce que *p* soit dépassé sont stockés dans une liste, *res*. Une fois que *p* est dépassé, on choisit un token au hasard dans cette liste. N'oublions pas que la liste des probabilités que nous parcourons contient des indices classés par leur probabilité. Notons que si *p* est plus élevé, plus de tokens seront inclus dans notre liste. Vice versa. Par conséquent, si on veut le même résultat à chaque fois, on peut définir *p* sur 0.
Maintenant, testons notre fonction get_pred plusieurs fois :

```
In [4]: get_pred("Natural Language Processing is", model, tok, p = 0.7)
Out[4]: ' used'
```

```
In [19]: get_pred("Natural Language Processing is", model, tok, p = 0.7)
Out[19]: ' much'
```

A chaque fois, il y a un résultat différent qui est exactement ce que nous attendons. Notre fonction de prédiction est prête pour construire notre application Web !

TROISIEME PARTIE
CREATION DE L'APPLICATION WEB

CREATION DE L'APPLICATION WEB

Après nos recherches on a su, que Panel comporte trois composants principaux :

- **Panels** : conteneurs pouvant contenir un ou plusieurs volets (objets) tels que du texte, des images, des graphiques, des widgets, etc. (ils peuvent également contenir d'autres panneaux)
- **Volets** : tout objet unique tel que texte, image, dataframe, etc.
- **Widgets** : éléments réglables par l'utilisateur tels que la saisie de texte, les curseurs, les boutons, les cases à cocher qui peuvent modifier le comportement des volets

La prochaine et dernière chose est qu'il existe plusieurs façons pour nous de définir comment les différents volets et widgets interagissent les uns avec les autres. C'est ce qu'on appelle des « rappels ». Par exemple, si un certain bouton est enfoncé, comment les autres volets doivent-ils être mis à jour ? Nous définirons plus tard une fonction de rappel qui fait exactement cela.

Présentation de l'application de haut niveau :
Notre application de génération de texte aura une entrée pour qu'un utilisateur entre le texte souhaité. Ensuite, l'utilisateur doit pouvoir générer un nouveau token en appuyant sur un bouton. Après, un nouveau texte sera généré avec un token prédit à partir de la fonction que nous avons définie dans la partie précédente. Enfin, l'utilisateur devrait pouvoir continuer à générer un nouveau texte en plus des tokens déjà prédits.

Mise en œuvre :
Commençons par importer « panel » et créons le widget de saisie de texte :

```
In [5]: import panel as pn
        pn.extension() # loading panel's extension for jupyter compatibility
        pn.extension(sizing_mode='scale_width')
        text_input = pn.widgets.TextInput()
```

Maintenant, si nous *exécutons text_input* dans jupyter, nous obtenons ce qui suit :

```
In [6]: text_input
Out[6]:
```

Ensuite, nous voulons un volet qui stockera tout le texte au fur et à mesure que nous générons de plus en plus de tokens :

```
In [7]: generated_text = pn.pane.Markdown(object = text_input.value)
```

Notez que nous définissons l'objet de texte à la valeur de text_input . Nous voulons que la valeur de generate_text ait la même valeur que text_input puisque nous prédisons un nouveau texte au-dessus de generate_text . Au fur et à mesure que d'autres tokens sont ajoutés à notre séquence, nous continuerons à prédire sur le generate_text jusqu'à ce que l'utilisateur modifie le text_input . Dans ce cas, le processus redémarrera.
Cependant, nous n'avons pas encore tout à fait terminé. Bien que generate_text prenne la valeur de text_input à son initiation, il ne se mettra pas à jour si la valeur text_input change. Pour cela, nous devons lier ces deux objets ensemble ainsi :

```
In [8]: text_input.link(generated_text, value = 'object')

Out[8]: Watcher(inst=TextInput(sizing_mode='scale_width'), cls=<class 'panel.widgets.input.TextInput'>, fn=<function Reactive.link.<locals>.link at 0x0000018D9744D790>, mode='args', onlychanged=True, parameter_names=('value',), what='value', queued=False)
```

Maintenant que nous avons nos deux objets texte, créons notre widget bouton :

```
In [9]: pn.Row(text_input, generated_text)

Out[9]:

In [10]: button = pn.widgets.Button(name="Generate", button_type = "success")
         resetBtn = pn.widgets.Button(name="Reset", button_type = "danger")
```

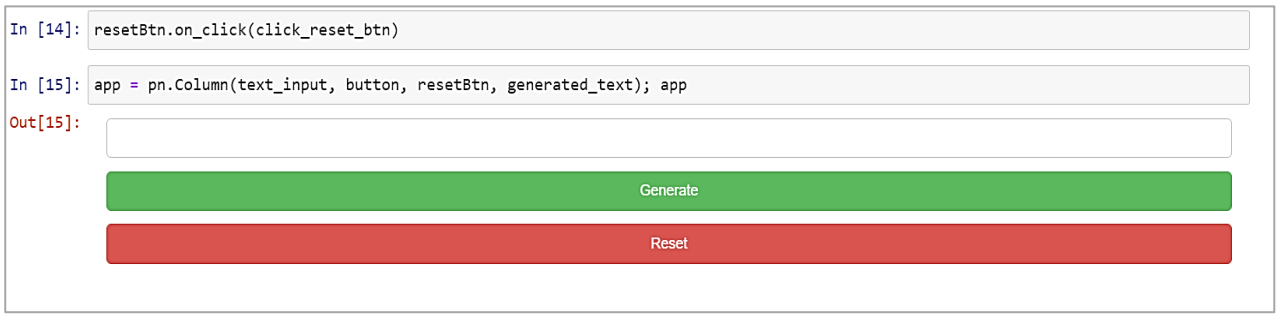
Super, maintenant que nous avons un bouton pour valider et pour effacer, nous n'avons plus qu'à le lier à notre comportement souhaité. Pour cela, nous écrirons une fonction de rappel qui s'exécutera à chaque clic sur le bouton :

```
In [11]: def click_cb(event):
             pred = get_pred(generated_text.object, model, tok)
             generated_text.object += pred

In [12]: def click_reset_btn(event):
             text_input.value = ""

In [13]: button.on_click(click_cb)
```

Nous avons maintenant fini de créer tous nos widgets, volets et fonctions. Il suffit de mettre ces objets dans « panel » et le tour est joué :

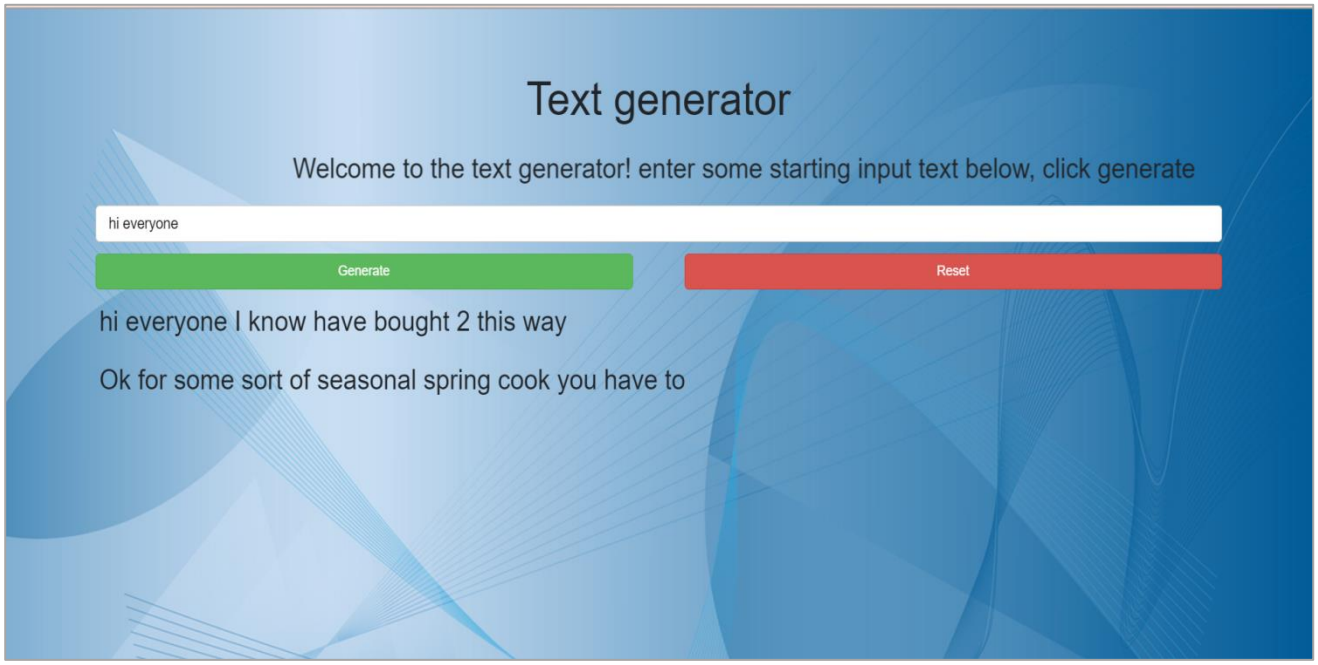


Servir l'application :

Le « panel » facilite le service de l'application. Il existe deux méthodes qui peuvent être utilisées pour ce faire. La première est la fonction "show()". Ceci est généralement utilisé pour le débogage et il est utilisé comme ci-dessous. Cela lancera une nouvelle fenêtre avec notre « panel » *final_app* s'exécutant comme une application Web.

Cela lancera votre application sur un port local tant que vous aurez le code suivant dans votre notebook :

La figure suivante représente Le résultat de l'application après avoir personnalisé le template par défaut :



Un autre résultat après click sur le bouton reset :