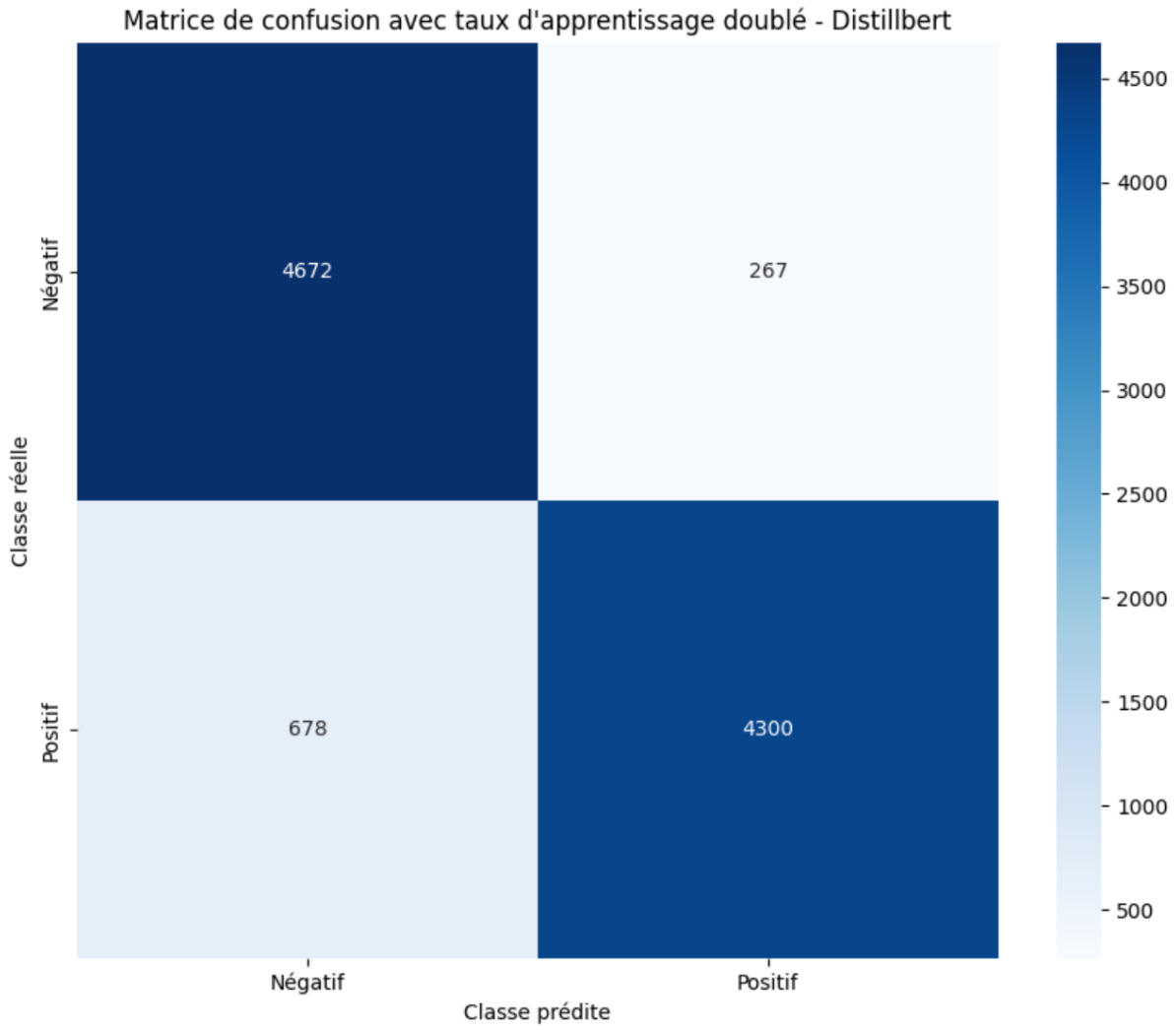
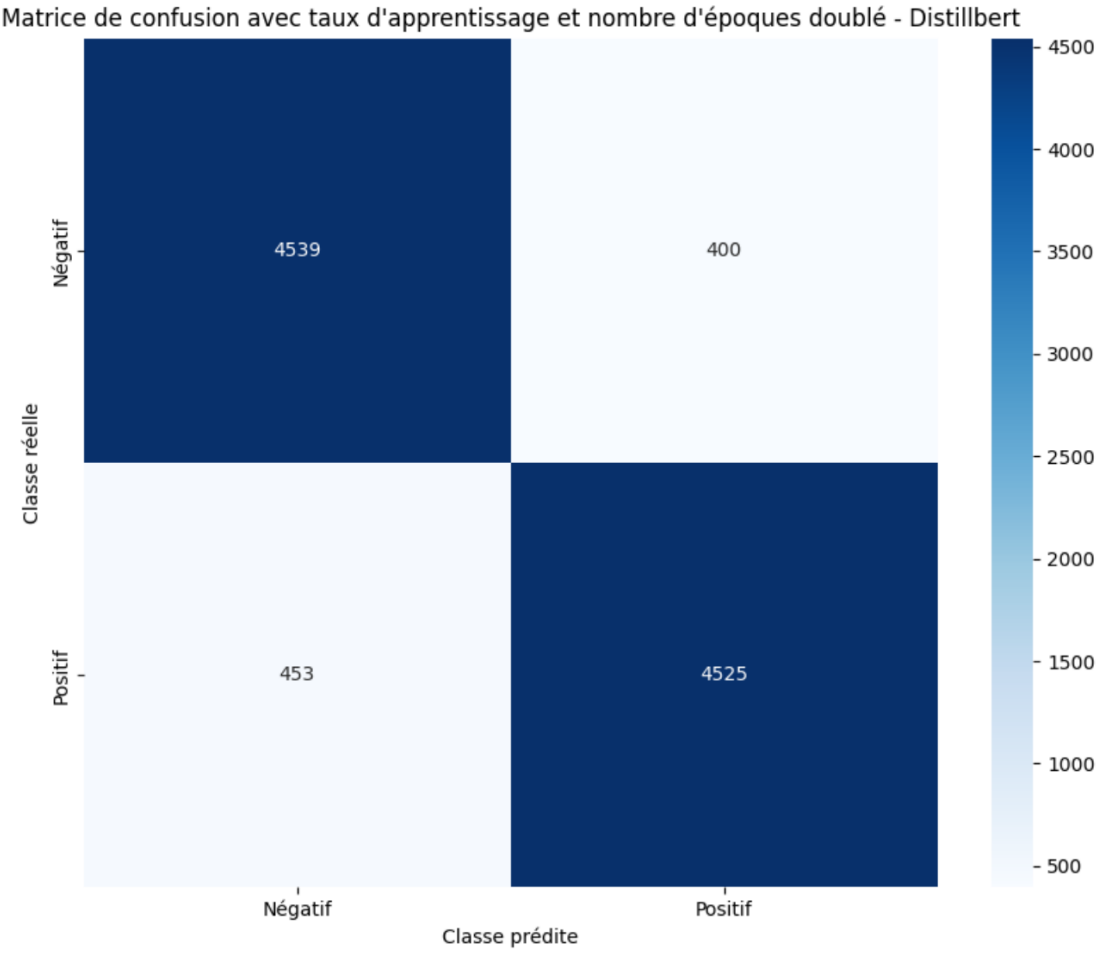
Exercice 1

```
Résultats de l'évaluation du modèle Distillbert avec taux d'apprentissage doublé :
--------------------------------------------------
Exactitude : 0.9047
Précision : 0.9415
Rappel : 0.8638
Score F1 : 0.9010
--------------------------------------------------
```



```
Résultats de l'évaluation du modèle Distillbert avec taux d'apprentissage et nombre d'époques doublé :
--------------------------------------------------
Exactitude : 0.9140
Précision : 0.9188
Rappel : 0.9090
Score F1 : 0.9139
--------------------------------------------------
```



Discussion approfondie

Approche 1 : Augmentation du taux d'apprentissage

Dans cette première approche, j'ai décidé de doubler le taux d'apprentissage. Cette modification a été préférée car dans la méthode originale, le modèle prenait trop de temps à s'entraîner, et il risquait de manquer des zones importantes de l'espace des paramètres. En augmentant le taux d'apprentissage, l'idée était de permettre au modèle de converger plus rapidement tout en explorant plus efficacement les gradients.

Observations et résultats

- Après modification, les métriques suivantes ont changé de manière significative :
  - **L'exactitude (accuracy)** : Une augmentation notable, reflétant un meilleur apprentissage global.
  - **Le rappel** : Une nette amélioration, bien que cette valeur reste éloignée de celle de la précision.
  - **Le score F1** : Augmenté également, indiquant un meilleur équilibre entre la précision et le rappel.

On note également une réduction importante de la perte d'entraînement, ce qui montre que le modèle n'était pas encore optimal dans la configuration initiale. Cependant, la différence entre la précision et le rappel montre que le modèle reste encore trop prudent pour prédire la classe positive, préférant le faire uniquement lorsqu'il est très confiant.

Approche 2 : Augmentation du nombre d'époques

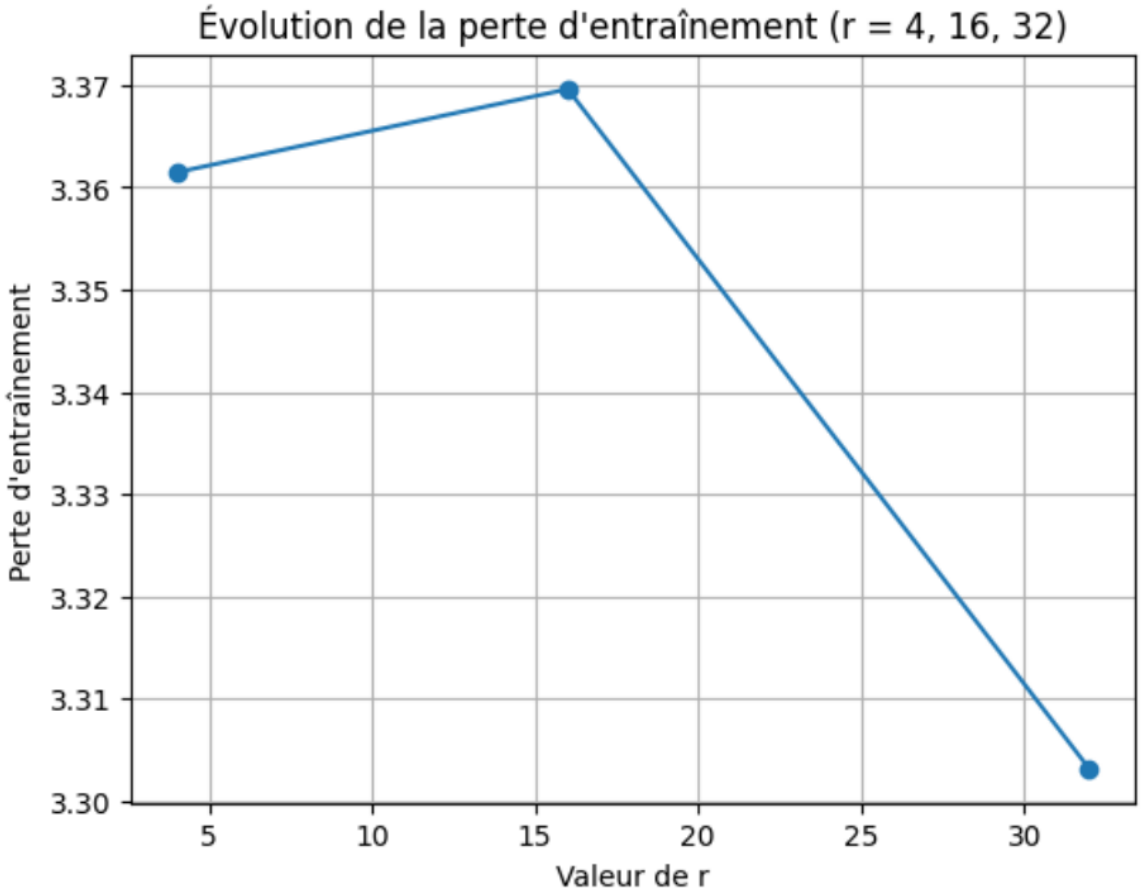
Pour cette deuxième approche, j'ai conservé le taux d'apprentissage doublé introduit dans l'approche précédente, tout en doublant le nombre d'époques (passant de 1 à 2). L'idée derrière cette modification était de donner au modèle plus de temps pour s'entraîner, afin qu'il puisse mieux apprendre des exemples complexes et affiner ses prédictions surtout après avoir augmenté le taux d'apprentissage.

Observations et résultats

- Avec deux époques, les métriques montrent une amélioration continue :
  - **Le rappel et le score F1** continuent d'augmenter par rapport à l'approche précédente.
  - La perte d'entraînement diminue davantage, confirmant que le modèle apprend mieux.

En ce qui concerne la différence entre la précision et le rappel, celle-ci reste présente mais elle est très très faible. La meilleure équilibre possible est presqu'atteint.

Exercice 2.1



Discussion approfondie

Observations et résultats

Quand la valeur de $r = 4$, on remarque que la perte est relativement élevée, à environ 3.36. Cela indique que le modèle est sous-adapté, probablement en raison d'une capacité trop limitée pour capturer la complexité des données d'entraînement.

Pour $r = 8$, la perte diminue significativement à environ 3.11, montrant une amélioration notable par rapport à $r = 4$. Cela suggère que cette valeur de r offre au modèle une meilleure capacité d'adaptation, permettant de mieux s'ajuster aux données tout en restant relativement efficace en termes de ressources.

Quand $r = 16$, la performance semble se dégrader légèrement, avec une perte atteignant environ 3.37. Cela pourrait indiquer un certain déséquilibre entre la capacité d'adaptation et la stabilité de l'entraînement. Bien que $r = 16$ soit souvent une valeur efficace dans d'autres scénarios, ici, il semble moins performant que $r = 8$, probablement en raison de caractéristiques spécifiques aux données ou au processus d'entraînement.

Enfin, pour $r = 32$, on observe une amélioration significative, avec une perte qui baisse à environ 3.30. Cette baisse indique que le modèle bénéficie d'une capacité accrue pour s'ajuster aux données, ce qui conduit à une meilleure performance en termes de perte. Cela montre que dans ce cas, une valeur plus élevée de r permet au modèle de mieux capturer la complexité des données d'entraînement.

Consommation de mémoire

J'ai observé que pendant l'entraînement avec $r = 32$, il y a eu une augmentation notable de la consommation de mémoire (dans Google Colab) pendant l'entraînement. Cela s'explique par le fait que r contrôle la taille des matrices A et B dans l'approche LoRA. À mesure que r augmente, la mémoire nécessaire pour stocker ces matrices croît proportionnellement.

Conclusion

Ces résultats suggèrent que dans ce contexte, $r = 8$ semble être la valeur la plus efficace en termes de performance et de perte, avec un bon équilibre entre adaptation et consommation de ressources. $r = 32$ offre une bonne alternative pour des tâches nécessitant une plus grande capacité d'adaptation, mais au prix d'une consommation de mémoire significativement plus élevée.

Exercice 2.2

=== Phase 2: Test Initial ===
Input: Hello, how are you?
Output initial: Hello, how are you? I'm doing well. I've been doing a lot of work on this. I have a lot more to do. But I'll

=== Phase 4: Test Final ===
Input: Hello, how are you?
Output final: Hello, how are you? I'm here with my friend, and we're going to be talking about some of the stuff that we've been doing in the last

=== Comparaison des Résultats ===
Différence dans les réponses: Oui

Réponse initiale: Hello, how are you? I'm doing well. I've been doing a lot of work on this. I have a lot more to do. But I'll

Réponse finale: Hello, how are you? I'm here with my friend, and we're going to be talking about some of the stuff that we've been doing in the last

---

On peut voir que notre modèle n'a pas réussi à prendre en compte le nouveau point de donnée qu'on lui a donné puisque l'output diverge de ce qu'on lui avait donné. Il y a plusieurs facteurs mais le plus trivial est qu'il n'a pas ajusté son comportement pour adopter le format de réponse fourni dans le nouvel exemple d'entraînement ("I am doing well, thank you for asking! I am here to help you."). Au lieu de cela, il continue à générer des réponses longues et discursives qui s'éloignent du style concis et professionnel que nous souhaitions lui inculquer.

P.S. : Voir l’annexe

Annexe

Code utilisée pour l'exo 2.2

```
# 1. Entraînement initial
print("=== Phase 1: Entraînement Initial ===")
model, tokenizer = setup_model_and_tokenizer()
dataset = load_dataset('gberseth/IFT6758-comments', split="train")
dataset = dataset.map(lambda example: {'text': example['input'] +
example['output']})

training_args = get_training_args()
peft_config = get_peft_config()
trained_model, train_result = train_model(model, tokenizer, dataset,
peft_config, training_args)

# 2. Test initial
print("\n=== Phase 2: Test Initial ===")
test_input = "Hello, how are you?"
initial_output = generate_response(trained_model, tokenizer,
test_input)
print(f"Input: {test_input}")
print(f"Output initial: {initial_output}")

# 3. Ajout d'un nouvel exemple et ré-entraînement
print("\n=== Phase 3: Ré-entraînement avec données augmentées ===")
new_example = {
    'input': 'Hi, how are you doing?',
    'output': 'I am doing well, thank you for asking! I am here to
help you.',
    'text': 'Hi, how are you doing? I am doing well, thank you for
asking! I am here to help you.'
}
augmented_dataset = dataset.add_item(new_example)
retrained_model, retrain_result = train_model(model, tokenizer,
augmented_dataset, peft_config, training_args)

# 4. Test avec le même input sur le modèle ré-entraîné
print("\n=== Phase 4: Test Final ===")
final_output = generate_response(retrained_model, tokenizer,
test_input)
print(f"Input: {test_input}")
print(f"Output final: {final_output}")

# Comparaison des résultats
print("\n=== Comparaison des Résultats ===")
print("\nRéponse initiale:", initial_output)
print("\nRéponse finale:", final_output)
```