

قسم الذكاء الصنعي

كلية الهندسة المعلوماتية

جامعة تشرين



الجمهورية العربية السورية

وزارة التعليم العالي

جامعة تشرين _ كلية الهندسة المعلوماتية

قسم الذكاء الصنعي

الترجمة من اللغة الإنجليزية إلى اللغة الفرنسية

Translation from English to French

مشروع فصلي

إعداد الطالب

روان دريباتي

زين العابدين سلمان

عمار معلا

إشراف

أ. ندى جندي

العام الدراسي 2023_2022

ملخص

في هذا المشروع، تم تطوير مجموعة نماذج ذكاء صنعي في عملية الترجمة (من اللغة الإنجليزية إلى اللغة الفرنسية)، تم مقارنة أداء المعماريات المستخدمة مع وبدون تضمين الكلمات في المدخلات (word embedding) ، أظهرت النتائج أن استخدام تضمين الكلمات في المدخلات (word embedding) يحسن أداء النماذج في الترجمة بشكل كبير، حيث تمكنت النماذج التي استخدمت التضمين من تحقيق معدل دقة أفضل بكثير مقارنة بالنماذج التي لم تستخدم التضمين، من المهم ملاحظة أن الكثير من نماذج الذكاء الصنعي لا تصل إلى مرحلة الانتاج فمما بتوصيف فصل كامل لمراحل تطوير نموذج ذكاء صنعي من مرحلة التخطيط إلى الانتاج.

Abstract

In this project, a set of artificial intelligence models were developed for the translation process (from English to French). The performance of the architectures used was compared with and without word embedding in the inputs. The results showed that using word embedding in the inputs improves the models' performance in translation. The models that used word embedding were able to achieve much higher accuracy rates compared to the models that did not use embedding. It is important to note that many artificial intelligence models do not reach the production stage. We have described a complete chapter on the stages of developing an artificial intelligence model from the planning to production stage.

الفهرس

1	الفصل الأول: مقدمة.....
2	الدراسات المرجعية.....
2	1-1-1 الأوراق المرجعية التي اعتمدت.....
3	1-1-1 جدول التلخيص.....
5	1-2 البيئة والأدوات المستخدمة.....
6	3- منهجة البحث.....
7	الفصل الثاني : الدراسة النظرية.....
7	1-2 الترجمة.....
8	Word Embedding2-2
10	Recurrent Neural Network (RNN)3-2
10	3-3-1 الشبكات العصبية المتكررة البسيطة RNN.....
14	2-3-2 LSTM (الذاكرة طويلة -قصيرة الأمد).....
15	4-2 التوابع المستخدمة.....
15	1-4-2 RELU.....
16	2-4-2 sigmoid.....
17	3-4-2 softmax.....
18	الفصل الثالث: الجزء العملي.....
18	1-3 التخطيط.....
20	2-3 الداتا ومعالجتها.....
23	3-3 بناء وتدريب النماذج Build and Train Models.....
37	3-3-3 1 النتائج.....
39	4-3 التع包ة.....
47	4-3 packaging.....
52	5-3 مرحلة المنتج النهائي Deployment.....
53	6-3 مرحلة المراقبة Monitior.....
	التوصيات.....

فهرس الأشكال

9.....	الشكل 1-2
12.....	الشكل 2-2
14.....	الشكل 2-3
16.....	الشكل 4-2
16.....	الشكل 5-2
17.....	الشكل 6-2
18.....	الشكل 1-3
22.....	الشكل 2-3
26.....	الشكل 3-3
28.....	الشكل 4-3
30.....	الشكل 5-3
32.....	الشكل 6-3
33.....	الشكل 7-3
36.....	الشكل 8-3
37.....	الشكل 9-3
38.....	الشكل 10-3
40.....	الشكل 11-3
42.....	الشكل 12-3

43.....	الشكل 13-3
45.....	الشكل 14-3
46.....	الشكل 15-3
48.....	الشكل 16-3
52.....	الشكل 17-3

فهرس الجداول :

4.....**الجدول (1-1)**

قائمة المصطلحات:

المصطلح	المختصر الموافق
Recurrent Neural Networks	RNN
Deep Neural Networks	DNN
Long Short-Term Memory	LSTM
Machine Learning Operations	MLOps
Application Programming Interface	API

الفصل الأول: مقدمة

الترجمة من لغة إلى أخرى تعتبر واحدة من أهم مهام وتحديات تعلم الآلة ومعالجة اللغات الطبيعية. فاللغات الطبيعية تتضمن تنوعاً كبيراً في النحو والمفردات والتعابير اللغوية، وهذا يجعل من الترجمة بين اللغات المختلفة مهمة صعبة.

تعتمد تقنيات الترجمة الحديثة على النماذج العميقه للتعلم الآلي، والتي تتضمن شبكات النموذج العصبي العميق (Deep Neural Networks) وخوارزميات تعلم الآلة المعتمدة على البيانات. وتستخدم هذه التقنيات مجموعات كبيرة من البيانات المترجمة لتدريب نماذج الترجمة الآلية، حيث يتم تدريب النموذج على تحويل النص المكتوب في اللغة الأولى (الإنجليزية) إلى النص المكتوب في اللغة الثانية (الفرنسية).

تعتمد نماذج الترجمة الحديثة على تقنيات مثل تضمين الكلمات (Word Embedding) لتحقيق أداء ممتاز في الترجمة الآلية. ومع تزايد حجم مجموعات البيانات المترجمة وتحسين تقنيات التعلم الآلي، يمكن توفير نتائج الترجمة الآلية التي تقارب جودة الترجمة البشرية في بعض الحالات، وهذا يجعل الترجمة الآلية أداة قيمة في مجالات الأعمال والتعليم والتواصل الدولي.

الدراسات المرجعية

1-1-1 الأوراق المرجعية التي اعتمدت

- Efficient Estimation of Word Representations in Vector Space

في هذه الورقة تم انتاج طريقة عصرية لتمثيل النص ضمن vector space structured هيكلته تعتمد على العلاقات الدلالية بين الكلمات أو بشكل أكثر وضوح كل كلمة سيتم ربطها بشعاع ثم إسقاطها في هذا الفضاء بحيث تكون العلاقات الهندسية بين word vectors انعكاس للعلاقات الدلالية بين الكلمات، وتعتبر هذه التقنية أساسية في تطوير نماذج الترجمة الآلية باستخدام الشبكات العصبية، حيث يتم استخدام word embedding لتمثيل الجمل في شكل رقمي بحيث يمكن للشبكات العصبية فهم المعنى والمضمون الخاص بالجمل والكلمات.

تم تأليف الورقة من قبل Edward H , Quoc Le ,Thomas Mikolov ، وتم نشرها في عام 2013 في مؤتمر تقنيات اللغة الطبيعية.

- Sequence to Sequence Learning with Neural Networks

في هذه الورقة تم استخدام الشبكات العصبية LSTM للترجمة من جملة إلى أخرى معتمدين على نموذج Sequence-to-Sequence حيث يتم في البداية استخدام word embedding لتمثيل الجمل ثم يتم إدخال هذه المتجهات إلى شبكة عصبية والتي تقوم بتحويل هذه المتجهات إلى المتجهات الخاصة بالجملة الهدف وبعد ذلك يتم تحويل المتجهات الناتجة إلى كلمات حقيقة في اللغة المستهدفة وتم الاعتماد أيضا على طبقة Attention للتركيز على الكلمات الأكثر أهمية من خلال تخصيص وزن معين لكل كلمة في الجملة.

تم تأليف الورقة من قبل Quoc V. Le ,Oriol Vinyals ,Ilya Sutskever وتم نشرها في عام 2014.

- Neural Machine Translation by Jointly Learning to Align and Translate

في هذه الورقة تم استخدام نفس الآلية التي تم الاعتماد عليها في الورقة السابقة من استخدام تقنية التضمين و شبكة LSTM وكذلك استخدام طبقة Attention للتركيز على الكلمات الأكثر أهمية ومنعاً لحدوث أخطاء في الترجمة مثل وضع كلمة ترجمة بغير موضعها الصحيح مثل لو كانت جملة الترجمة "الكتاب على الطاولة" تكون الترجمة "The book is on the table" بدلاً من "The table is on the book"

وهذا خاطئ لأنه تم وضع كلمتين بغير موضعها الصحيح لذلك تم استخدام طبقة Alignment لتحديد الموضع المناسب لكل كلمة اعتماد على درجات التركيز التي تأتي طبقة Attention .

تم تأليف الورقة من قبل Joshua Bengio ,Kyunghyun Cho ,Dzmitry Bahdanau ، وتم نشرها في عام 2014.

1-1-1 جدول التلخيص

مساوي	محاسن	الأفكار الرئيسية	اسم الورقة البحثية
قد تعاني طريقة التضمين من بعض المشاكل في تمثيل الكلمات النادرة ويحتاج أيضاً مصادر بيانات كبيرة	تقليل حجم النماذج اللازمة لتمثيل الكلمات وتحسين وقت التدريب للنماذج العصبية ، مما يساعد في تحسين أداء تطبيقات الترجمة الآلية	استخدام تقنية التضمين لتحسين جودة الترجمة الآلية باستخدام الشبكات العصبية	Efficient Estimation of Word Representations in Vector Space

<p>تعقيد عملية التدريب وال الحاجة إلى كميات كبيرة من البيانات لضمان تحقيق النتائج المرجوة و ظهور الأخطاء في الترجمة الآلية والتي تؤثر على جودة الترجمة وتفاصيل المعنى</p>	<p>القدرة على التعامل مع مشكلة طول الجمل والتعامل مع ترجمة الجمل ذات المعاني المجردة والمعقدة بفضل طبقة attention</p>	<p>استخدام الشبكة العصبية LSTM مع تقنية التضمين وتقنية attention للتركيز على الكلمات الأكثر أهمية بالنسبة للجملة المستهدفة</p>	<p>Sequence to Sequence Learning with Neural Networks</p>
<p>على الرغم من استخدام Alignment إلا أن النموذج يعاني من مشكلة في التعامل مع الجمل المعقدة والطويلة وخاصة الجمل التي تحوي كلمات متعددة المعاني حيث تؤدي التداخلات بين الكلمات إلى تحديد مواضع غير ملائمة للكلمات</p>	<p>تحسين جودة ودقة الترجمة والتغلب على مشكلة الترجمة الحرافية بعد استخدام تقنية Alignment</p>	<p>استخدام الشبكة العصبية LSTM مع تقنية التضمين وتقنية attention وتقنية Alignment لتحديد الموضع المناسب للكلمات في الجملة الهدف بشكل متزامن</p>	<p>Neural Machine Translation by Jointly Learning to Align and Translate</p>

(الجدول 1-1)

بعد قراءة هذه الأوراق قررنا استخدام 4 معماريات للشبكات في عملية التدريب RNN, LSTM وكذلك RNN, LSTM مع استخدام طبقة التضمين (Word Embedding) وذلك لمقارنة النتائج مع وبدون

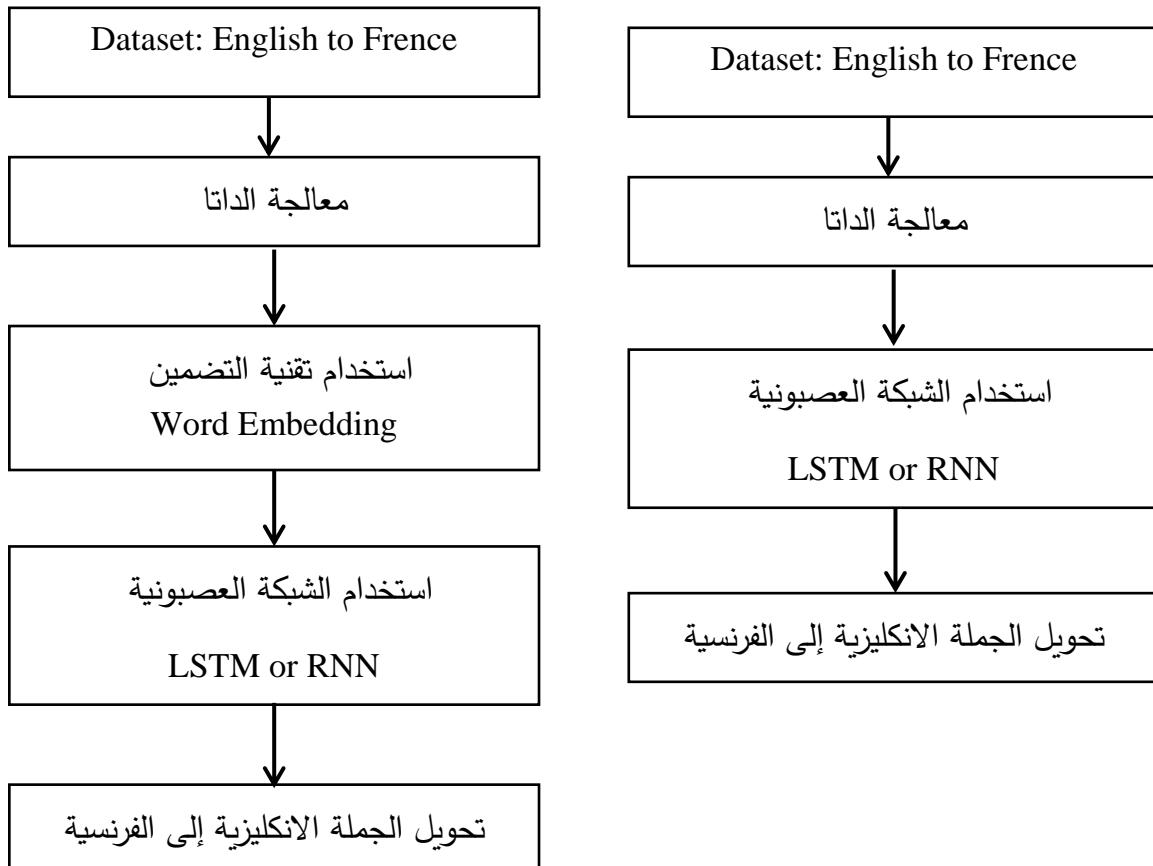
استخدام طبقة التصميم واعتمدنا على داتا سيت تحتوي على 137861 جملة انكليزية مع نظيرتها من اللغة الفرنسية.

٢-١ البيئة والأدوات المستخدمة

- في هذه الفقرة، سنشرح بإيجاز عن الأدوات والبيئة التي تم استخدامها في هذا البحث، والتي تشمل TensorFlow، Colab، FastAPI، Jupyter Notebook، Python، Docker، Github Actions .Pickle، PyYAML، NumPy، Keras
- باستخدام Github Actions يمكن إنشاء نظام للتحكم بعمليات التطوير والإختبار والنشر بصورة آلية، بحيث يمكنك تحسين الإنتاجية وتوفير الوقت اللازم لإنجاز المهام. يتم إعداد Github Actions بإستخدام ملفات YAML التي تحتوي على سلسلة من الخطوات التي يجب تنفيذها.
 - Docker هو برنامج يساعد على إنشاء وتشغيل بيئات البرمجة في أي نظام تشغيل، بما في ذلك Windows وLinux وMacOS. يساعد Docker في توفير بيئة متكاملة للتطوير والتشغيل، ويسمح بإنشاء حاويات (containers) تحتوي على جميع المكتبات والأدوات والإعدادات اللازمة لتشغيل تطبيقك.
 - Python هي لغة برمجة شائعة الاستخدام ومفتوحة المصدر، وتشتخدم في العديد من المجالات بما في ذلك الذكاء الاصطناعي وعلوم البيانات وتطوير الويب.
 - Jupyter Notebook هو برنامج تفاعلي للبرمجة يسمح للمستخدمين بكتابة الشفرات وعرض النتائج في واجهة مستخدم بديهية. يتم استخدام Jupyter Notebook عادةً للتعلم العملي والتجارب السريعة في علوم البيانات والذكاء الاصطناعي.
 - Colab هو بيئة تفاعلية تقدمها Google، ويتم استخدامها عادةً لتطوير النماذج الذكية والتعلم العميق وعلوم البيانات. يتميز Colab بواجهة مستخدم بديهية وخدمات السحابة التي يتم توفيرها من قبل Google.
 - TensorFlow هو إطار عمل للتعلم الآلي والذكاء الاصطناعي، ويستخدم عادةً لبناء الشبكات العصبية وتدريبها.

- Keras هو إطار عمل للتعلم الآلي يُستخدم كواجهة عالية المستوى لتطبيقات TensorFlow. يعتبر Keras سهل الاستخدام ويتميز بالمرنة والسرعة في بناء الشبكات العصبية وتدريبها.
- NumPy هو مكتبة لغة بايثون مفتوحة المصدر تُستخدم للعمليات الرياضية والعلمية، وتتضمن مجموعة من الأدوات والوظائف المفيدة لتحليل ومعالجة البيانات الكبيرة. يتميز NumPy بالسرعة والكفاءة والتعامل مع البيانات في صيغة المصفوفات.
- yaml هو مكتبة لغة بايثون مفتوحة المصدر يتم استخدامها للتعامل مع صيغة YAML. يستخدم YAML عادة لتخزين البيانات المنظمة بشكل هرمي، ويمكن استخدام yaml لتحويل البيانات بين صيغ YAML وصيغ أخرى.
- Pickle هو مكتبة لغة بايثون تُستخدم للتسلسل والإلغاء التسلسل للبيانات. يستخدم Pickle عادة لحفظ واستعادة البيانات في صيغة ملف، ويتميز بالسهولة والكفاءة في التعامل مع البيانات.
- Neptune.ai هو إطار عمل يستخدم لتبث ومراقبة نماذج الذكاء الصنعي في مرحلة التعلم.
- FastAPI هو إطار عمل يستخدم لبناء Restful API لتحويل النماذج إلى micro service.

3-1 منهجة البحث



الفصل الثاني : الدراسة النظرية

1-2 الترجمة

"إذا تحدثت إلى رجل بلغة يفهمها ، فهذا يذهب إلى رأسه. إذا تحدثت إليه بلغته الخاصة ، فهذا أمر يسير في قلبه. " - نيلسون مانديلا .

تعد القدرة على التواصل مع بعضنا البعض جزءاً أساسياً من كونك إنساناً. هناك الكثير من اللغات المختلفة في جميع أنحاء العالم. نظراً لأن عالمنا أصبح متربطاً بشكل متزايد ، فإن ترجمة اللغة توفر جسراً ثقافياً واقتصادياً حاسماً بين الناس من مختلف البلدان والمجموعات العرقية. تتضمن بعض حالات الاستخدام الأكثر وضوحاً ما يلي:

الأعمال التجارية: التجارة الدولية والاستثمار والعقود والتمويل ، التجارة: السفر ، شراء البضائع والخدمات الأجنبية ، دعم العملاء ، الوسائل: الوصول إلى المعلومات عبر البحث ، ومشاركة المعلومات عبر الشبكات الاجتماعية ، وتوطين المحتوى والإعلان ، التعليم: تبادل الأفكار ، التعاون ، ترجمة الأوراق البحثية ، الحكومة: العلاقات الخارجية ، التفاوض .

لتلبية هذه الاحتياجات ، تستثمر شركات التكنولوجيا بشكل كبير في الترجمة الآلية. أدى هذا الاستثمار والتطورات الأخيرة في التعلم العميق إلى تحسينات كبيرة في جودة الترجمة. وفقاً لـ Google ، أدى التحول إلى التعلم العميق إلى زيادة دقة الترجمة بنسبة 60٪ مقارنة بالنهج القائم على العبارة المستخدم سابقاً في ترجمة Google. اليوم ، يمكن لجوجل ومايكروسوف特 ترجمة أكثر من 100 لغة مختلفة وتقرب من الدقة على مستوى الإنسان للعديد منها.

وبالتقدم السريع إلى عام 2019 ، "أن تكون قادرين على بناء مترجم لغة لأي زوج محتمل من اللغات " يا لها من نعمة كانت معالجة اللغة الطبيعية ! [2]

Word Embedding 2-2

عند القيام ببناء نماذج تعلم الآلة مهمتها فهم وتفسير اللغات الطبيعية المتداقة مثل لغات البشر من غير الممكن لهذه النماذج التعامل مع البيانات النصية بشكل مباشر ، فهي ليست ذكية بما فيه الكفاية لبدء معالجة النص في شكلها الأصلي ، نماذج تعلم الآلة تعتمد بشكل رئيسي على مبادئ الإحصاء والرياضيات والتحسين ، ولا تفهم سوى لغة الأرقام الأمر الذي يستدعي إجراء معالجة مسبقة للبيانات النصية وتحويلها إلى تمثيل عددي مقابل لها ، إن تحويل البيانات النصية إلى بيانات عددية مقابلة لها يمكن حتماً خوارزمية التعلم الآلي من فهمها والتعامل معها؛ حيث تعمل خوارزميات التضمين Embedding على إيجاد تمثيل عددي مقابل لكلمات النصية بشكل فعال . word embedding في هذا المشروع

في هذه الطريقة يتم أولاً تجميع الكلمات التي نرغب في إيجاد ترميز لها، ومن ثم اختيار بعد الشعاع features [1]، هنا البعد يرمز إلى عدد الميزات ، هذه الميزات ستتصف الكلمة (سنوضح هذا لاحقاً)، في هذه الحالة نستطيع التحكم ببعد شعاع الكلمة (عدد الميزات) ونستطيع جعله أقل بكثير من العدد الإجمالي للمفردات، قيمة كل ميزة في الشعاع تكون عدداً حقيقياً ضمن المجال [-1, 1]، وكلما كانت القيمة قريبة من الواحد كلما كانت الميزة تعبر عن الكلمة والعكس في حال اقتربت من الصفر. على سبيل المثال بإمكاننا اختيار 300 ميزة في حال كان لدينا قاموس يحتوي على 10000 مفردة.

مثال توضيحي :

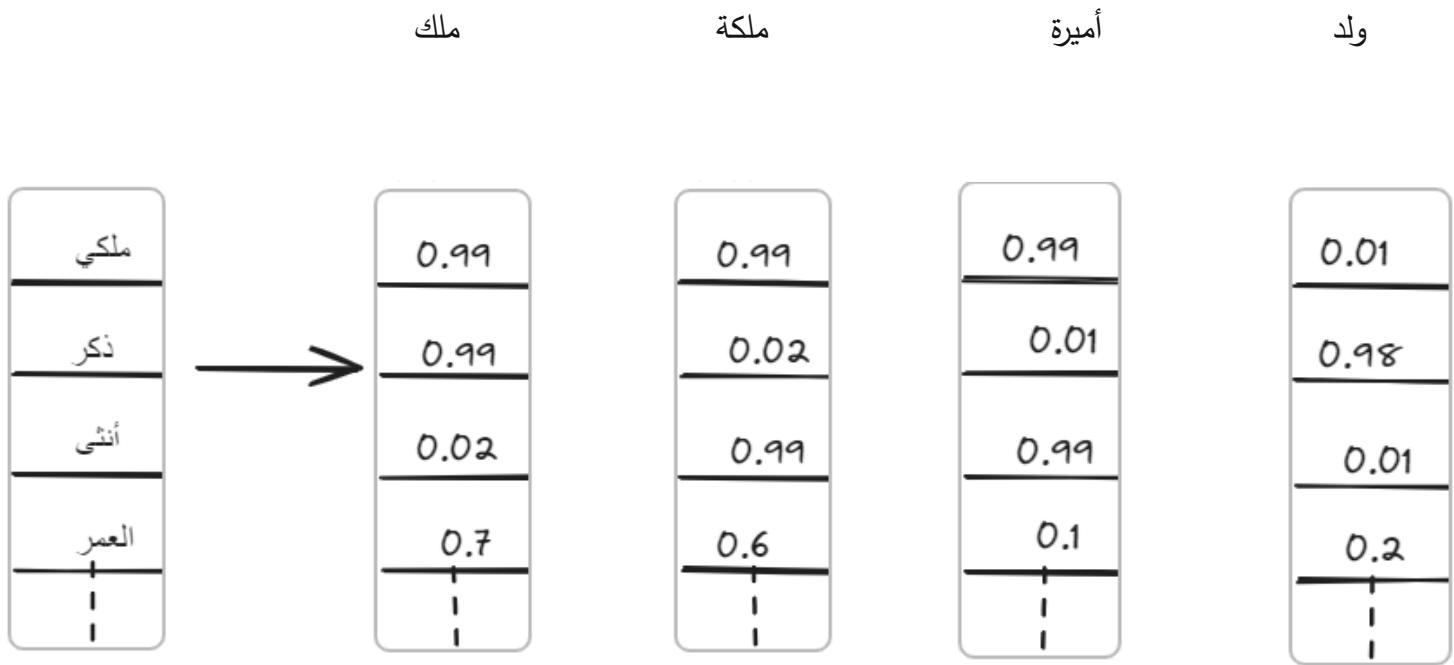
بفرض لدينا المفردات التالية

{ولد ، أميرة ، ملكة ، ملك}

والميزات المستخدمة لهذه المفردات هي:

(العمر، أنثى، ذكر، ملكي)

لدينا الشكل التالي يعبر عن شعاع كل كلمة [3]:



(الشكل 1-2 : يوضح مميزات كل كلمة)

نلاحظ من الشكل السابق أن الميزة "ذكر" تخص الكلمتين "ملك" و "ولد" فقط لذلك تكون قيمتها عند هاتين الكلمتين قريبة من الواحد ، بينما هذه الميزة لا تخص الكلمتين "ملكة" و "أميرة" لذلك تكون قيمتها عند هاتين الكلمتين قريبة من الصفر .

باستخدام هذه الطريقة استطعنا تمثيل الكلمات التي تتشابه في المعنى بأشعة قريبة جداً من بعضها : أي أن المسافة الإقليدية بينهما تكون صغيرة وهذا مفید جداً في حالات كثيرة منها حالة تعليم الشبكة العصبية .

فعلى سبيل المثال في حال أردنا إجراء تحليل مشاعر للجمل النصية وتصنيفها إلى ثلاثة أصناف هي إيجابي positive وسلبي negative وحيادي Neutral . في حال كان لدينا الجملتين التاليتين :

1. بعد استخدام الجهاز لمدة شهر أدركت أنه جهاز جميل .
2. بعد استخدام الجهاز لمدة شهر أدركت أنه جهاز رائع .

نذكر أن الكلمتين " رائع " و " جميل " متشابهتان جداً بالمعنى ويتم استخدامهما في سياق متشابه .

أثناء تدريب الشبكة العصبية كانت الجملة الأولى موجودة في مجموعة بيانات التدريب Training set وتم تدريب الشبكة العصبية على أن يكون خرج هذه الجملة إيجابياً .

الجملة الثانية موجودة في مجموعة بيانات الاختبار Test dataset ولم تتدرب عليه الشبكة أثناء عملية التدريب ، في حال أدخلنا الجملة الثانية إلى الشبكة العصبية لمعرفة خرج هذه الجملة مع العلم أن كلمة " رائع " غير موجودة أبداً في بيانات التدريب . وباستخدام أسلوب التضمين سيتم تمثيل كلمة " رائع " بشعاع قريب جداً من " جميل " وبالتالي ستتمكن الشبكة العصبية من إعطاء خرج إيجابي .

ف تستنتج ميزة كبرى لدى تضمين الكلمات word embedding وهي أنها قادرة على إجراء تعميم بشكل كبير جداً للشبكة العصبية .

Recurrent Neural Network (RNN) 3-2

عندما نتحدث عن أي مشكلة في NLP مع الشبكات العصبية ، فإن أول أمر يتadar للأذهان هو الشبكات العصبية المتركرة ، فهي الأكثر فعالية لأنها تعالج التسلسلات النصية ، وما يلي هي أنواع هذه الشبكات :

1-3-2 الشبكات العصبية المتركرة البسيطة RNN

نوع من الشبكات العصبية الاصطناعية التي تعتمد على البيانات التسلسلية، وتستخدم في تطبيقات التعلم العميق التي تتعامل مع البيانات المتتابعة والسلسل الزمنية مثل ترجمة اللغات ومعالجة اللغة الطبيعية والتعرف التلقائي على الكلام المنطوق ومن أهم الأمثلة على استخدامها البحث الصوتي وخدمة الترجمة من جوجل [1].

تعتبر الشبكات العصبية المتكررة قديمة نسبياً، حيث ظهرت لأول مرة في ثمانينيات القرن الماضي. لكن لم تتم الاستفادة من إمكانياتها بالشكل الأمثل إلا في السنوات الأخيرة تزامناً مع توفر البيانات الضخمة والقدرات الحاسوبية العالية.

وتتمتع هذه الشبكات بـ"ميزة الذاكرة" التي تسمح لها بأخذ معلومات من بيانات الدخل السابقة للتأثير على بيانات الدخل والمخرجات الحالية. فعلى عكس الشبكات العصبية العميقه الأخرى التي تفترض أن المدخلات مستقلة عن المخرجات، يعتمد الخرج في الشبكات المتكررة على العناصر السابقة في تسلسل البيانات .

كيف تعمل الشبكات العصبية المتكررة؟

تحتفل الشبكات العصبية المتكررة عن شبكات التغذية إلى الأمام(Feed-Forward) ، فشبكات التغذية إلى الأمام تأخذ كمية ثابتة من بيانات الدخل دفعه واحدة وتعطي كمية ثابتة من المخرجات في كل مرة . على الجانب الآخر لا تأخذ الشبكات العصبية المتكررة جميع بيانات الدخل معاً، إنما تأخذ عنصراً واحداً من تلك البيانات في كل مرة بشكل متسلسل ، ثم تجري عليه مجموعة من العمليات الحسابية المتسلسلة قبل إنتاج الخرج الموفق[2]، ويعرف هذا الخرج باسم الحالة المخفية(Hidden State) يتم بعد ذلك دمج الخرج الناتج عن المرحلة الحالية مع عنصر الدخل التالي لإنتاج خرج جديد وتستمر العملية حتى انتهاء تسلسل بيانات الدخل، ويمكن أيضاً برمجة النموذج للتوقف عند مرحلة معينة .

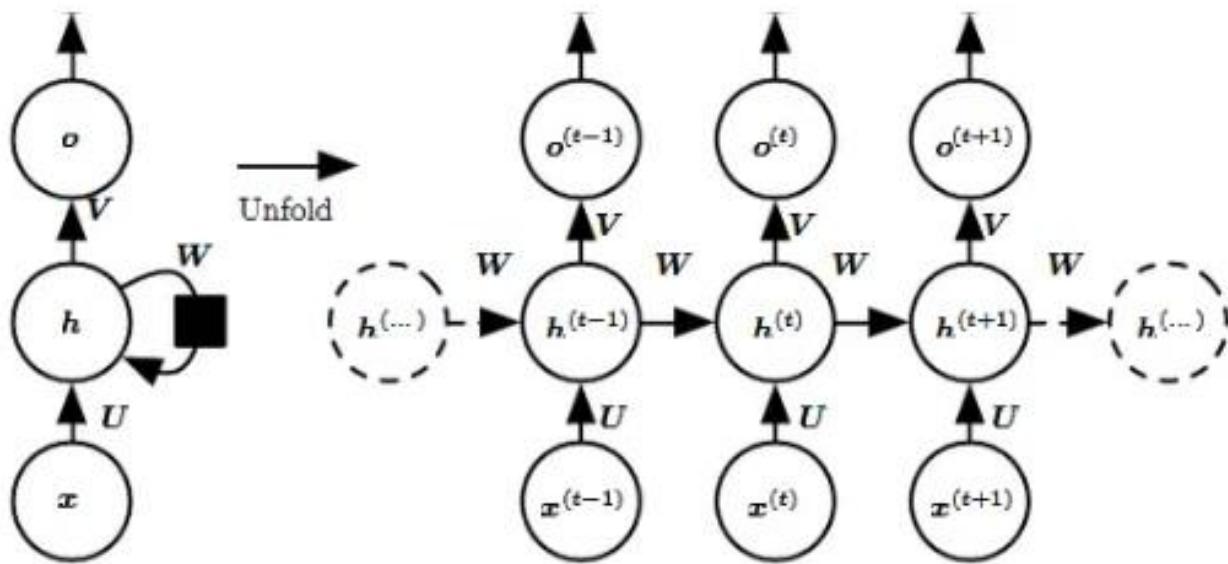
إذا في حالات مثل عندما يكون مطلوبا التنبؤ بالكلمة التالية من الجملة، تكون الكلمات السابقة مطلوبة وبالتالي هناك حاجة لتذكر الكلمات السابقة. وهنا تكمن أهمية شبكات RNN ، والتي حلت هذه المشكلة عن طريق استخدام ماسميه "Hidden state" ، فهي الميزة الرئيسية والأكثر أهمية لـ RNN ، والتي تتذكر بعض معلومات التسلسل . أي أنها تعالج التسلسلات "sequences" كالنصوص و السلاسل الزمنية (رغم أنها ليست ناجحة جداً مع السلاسل كما أشار فرانسوا كولييت) [3].

إن ما يميزها كما قلنا هو استخدام ذاكرة تتذكر جميع المعلومات حول ما تم حسابه في المراحل السابقة، وتستخدم نفس الأوزان لكل إدخال لأنه يؤدي نفس المهمة على جميع المدخلات أو الطبقات المخفية لإنتاج

المخرجات ، على عكس الشبكات العصبية الأخرى. وأيضاً إحدى المشكلات المتعلقة بالشبكات العصبية التقليدية وكذلك شبكات (CNN) هي أنها تعمل فقط بأحجام محددة مسبقاً، فهي تأخذ مدخلات ذات حجم ثابت وتنتج مخرجات ذات حجم ثابت. أما RNNS تتيح لنا تسلسلات متغيرة الطول كمدخلات ومخرجات ولهذا فهي الخيار رقم 1 للتعامل مع ومعالجة اللغة الطبيعية NLP .

إن الشيء الذي يعطي هذه الشبكات الخاصية التكرارية هو أنها تستخدم نفس الأوزان لكل خطوة .

والشكل التالي يوضح شكل الشبكة : RNN



الشكل 2-2 : (شبكة RNN)

يُظهر الجانب الأيسر من الرسم البياني أعلاه تدويناً لـ RNN وعلى الجانب الأيمن يتم فتح RNN (أو فتحه) في شبكة كاملة. من خلال إلغاء التمرين ، نعني أننا نكتب الشبكة للتسلسل الكامل. على سبيل المثال ، إذا كان التسلسل الذي نهتم به عبارة عن جملة من 3 كلمات ، فسيتم فتح الشبكة في شبكة عصبية ثلاثة طبقات .

الإدخال: يتم أخذ (t) x كمدخل للشبكة في الخطوة الزمنية t . على سبيل المثال ، x_1 ، يمكن أن يكون one-hot vector ، الموافق لكلمة في الجملة .

الحالة المخفية: (t) h تمثل حالة مخفية في الوقت t وتعمل بمثابة "ذاكرة" للشبكة. (t) h يُحسب بناءً على الإدخال الحالي والحالة المخفية لخطوة الوقت السابقة: $(1) h(t) = f(Ux(t) + Wh(t-1))$. [2] تعتبر الدالة f تحويلًا غير خطى مثل tanh و ReLU (سنتحدث عنها لاحقًا) .

الأوزان: يحتوي RNN على hidden connections (مدخلات للوصلات المخفية) بواسطة مصفوفة الوزن U ، و hidden-to-hidden recurrent connections (الوصلات المتكررة المخفية إلى المخفية المعلمة) بواسطة مصفوفة الوزن W ، و hidden-to-output connections (الوصلات المخفية إلى المخرجات المحددة) بواسطة مصفوفة الوزن V وجميع هذه الأوزان (U, V, W) تتم مشاركتها عبر الوقت.

الخرج : $O(t)$.

من المفترض أن تحمل RNN المعلومات حتى وقت طويل ومع ذلك فإنه من الصعب نشر كل هذه المعلومات عندما تكون الخطوة الزمنية طويلة جدًا ، وعندما تحوي الشبكة على عدد كبير من الطبقات العميقه تصبح غير قابلة للتدريب ، تسمى هذه المشكلة (اختفاء مشكلة التدرج vanishing gradient problem) : أي أن الشبكة العصبية تقوم بتحديث الأوزان باستخدام خوارزمية النسب للتدرج ، تنمو التدرجات أصغر عندما تقدم الشبكة إلى الطبقات السفلية ، إذا بقي التدرج ثابت يعني أنه لا يوجد مساحة للتحسين ، هذا يؤثر على خرج الشبكة ، ومع ذلك إذا كان الفرق في التدرج صغير جداً (على سبيل المثال تغيير الأوزان قليلاً) فان تتمكن الشبكة من تعلم أي شيء .

لتغلب على هذه المشكلة تم تحسين RNN إلى LSTM حيث توفر للشبكة معلومات سابقة ذات صلة إلى وقت أكثر حداة .

سنعرف على LSTM :

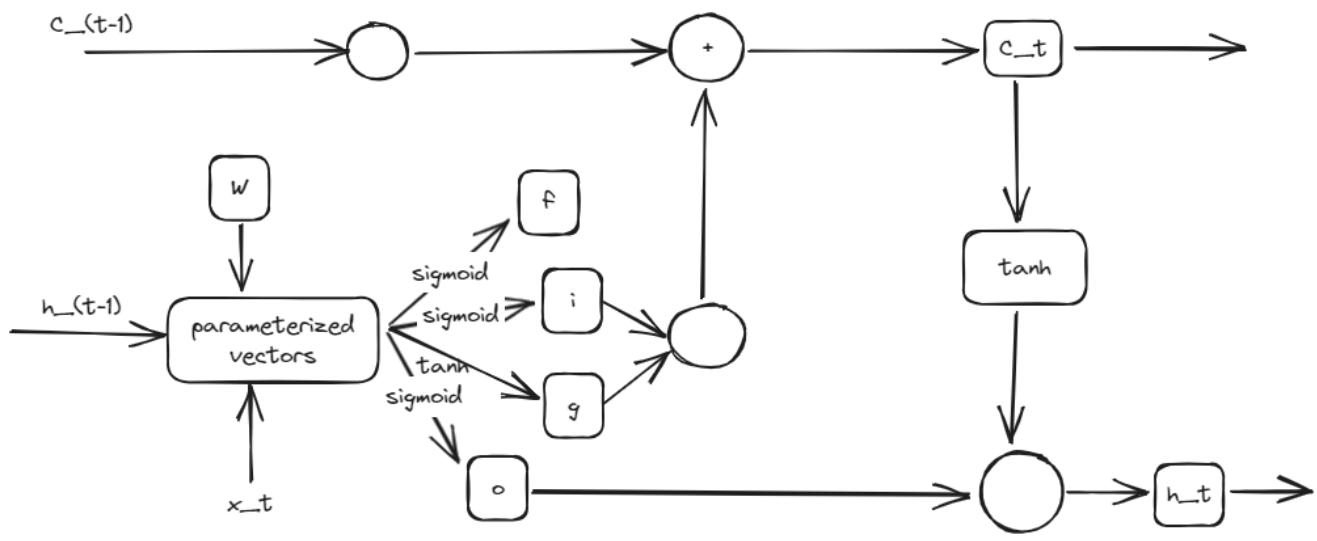
2-3-2 LSTM (الذاكرة طويلة -قصيرة الأمد)

إنها مجموعة متنوعة من الشبكات العصبية المتكررة (RNNs) ، يحتوي LSTM على اتصالات تغذية مرتبطة ، أي أنه قادر على معالجة تسلسل البيانات بالكامل ، بصرف النظر عن نقاط البيانات الفردية مثل الصور. يجد هذا التطبيق في التعرف على الكلام ، والترجمة الآلية ، وما إلى ذلك LSTM. هو نوع خاص من RNN ، والذي يظهر أداءً متميّزاً في مجموعة متنوعة من المشكلات.

يتم عقد الدور المركزي لنموذج LSTM بواسطة خلية ذاكرة تُعرف باسم "حالة الخلية" التي تحافظ على حالتها بمرور الوقت. حالة الخلية هي الخط الأفقي الذي يمر عبر الجزء العلوي من الرسم التخطيطي أدناه. يمكن تصوّرها كحزم ناقل تتدفق من خلاله المعلومات دون تغيير.

يمكن إضافة المعلومات إلى حالة الخلية أو إزالتها منها في LSTM ويتم تنظيمها بواسطة البوابات[1]. تسمح هذه البوابات اختيارياً للمعلومات بالتدفق داخل وخارج الخلية. يحتوي على pointwise sigmoid neural net layer و multiplication operation .

تعطي sigmoid layer أرقاماً بين صفر وواحد ، حيث يعني الصفر "لا يجب السماح بمرور أي شيء" ، ويعني الواحد "أنه يجب السماح لكل شيء بالمرور".



(الشكل 2-3 : شبكة LSTM)

4-2 التوابع المستخدمة

ReLU 1-4-2

يتم استخدام RELU كوظيفة تنشيط افتراضية ، وهي وظيفة التنشيط الأكثر استخداماً في الشبكات العصبية ، خاصة في شبكات CNN .

تعيد الدالة 0 إذا تلقت أي إدخال سالب ، وتعيد القيمة الموجبة نفسها التي تتلقاها[2] .

بالتالي فإنه يعطي ناتجاً يتراوح بين 0 و لانهاية .

أحد سلبيات تابع RELU أنه يقوم بتحويل جميع القيم السالبة إلى صفر ، اي انه في حال وجود قيم في التدريب مختلفة سالبة ينظر إليها التابع بشكل واحد وهو الصفر ، مما قد يتسبب بخسارة معلومات هامة لازمة لتدريب الشبكة العصبية.

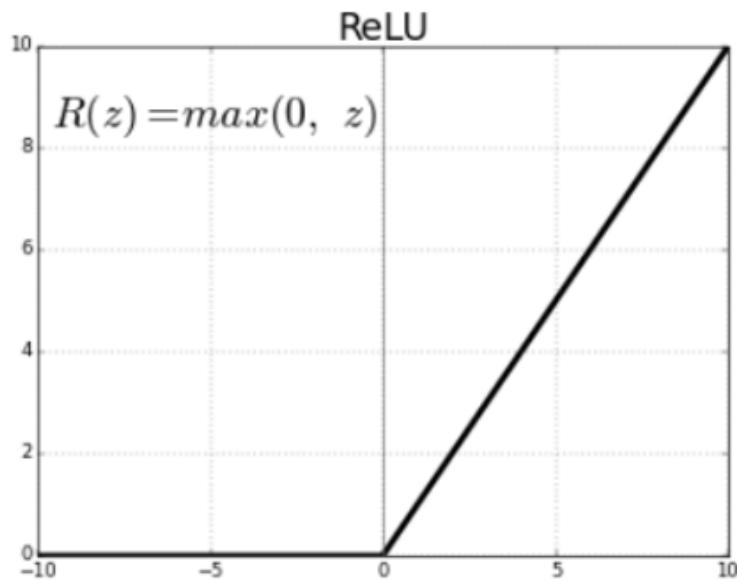
Def RLU (x):

If (x>0) :

return x

else:

return 0



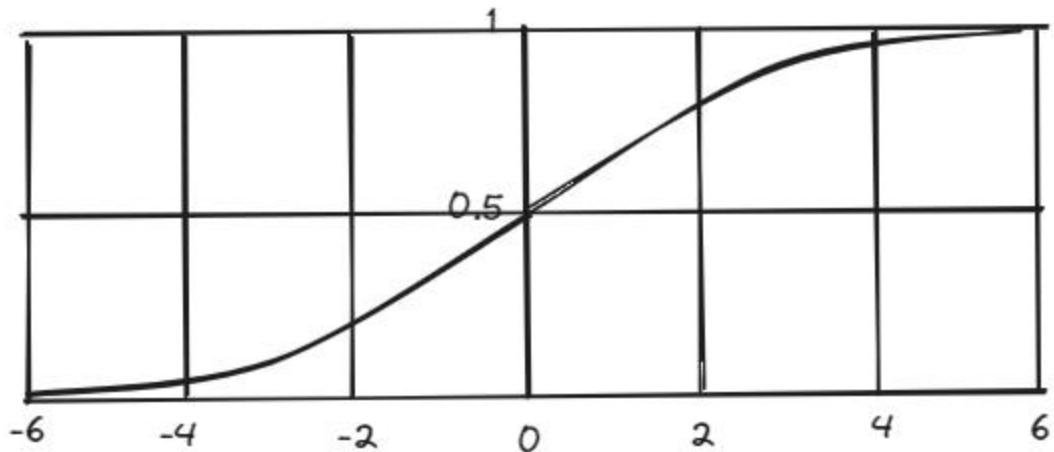
(الشكل 4-2 : تابع RELU)

sigmoid 2-4-2

تعمل الدالة sigmoid مع مدخلات من قيم حقيقية ، وتعطي قيم مستمرة بين 0 و 1 . [2]

Def sig(x):

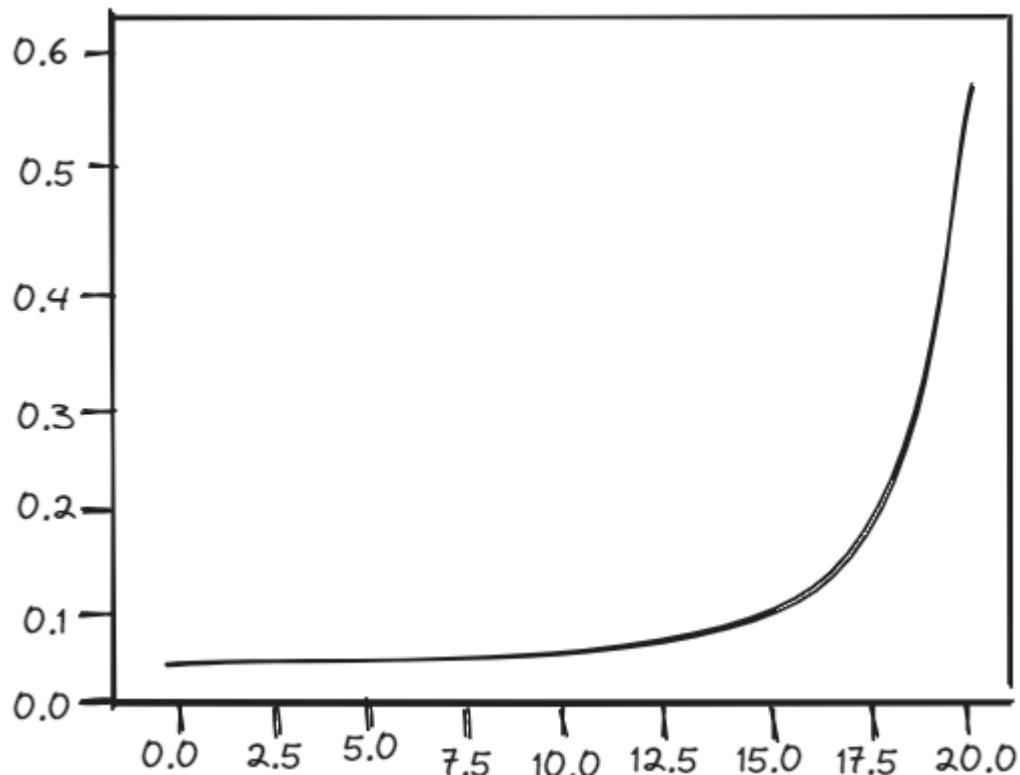
```
return 1/(1+np.exp(-x))
```



(الشكل 5-2 : تابع sigmoid)

softmax 3-4-2

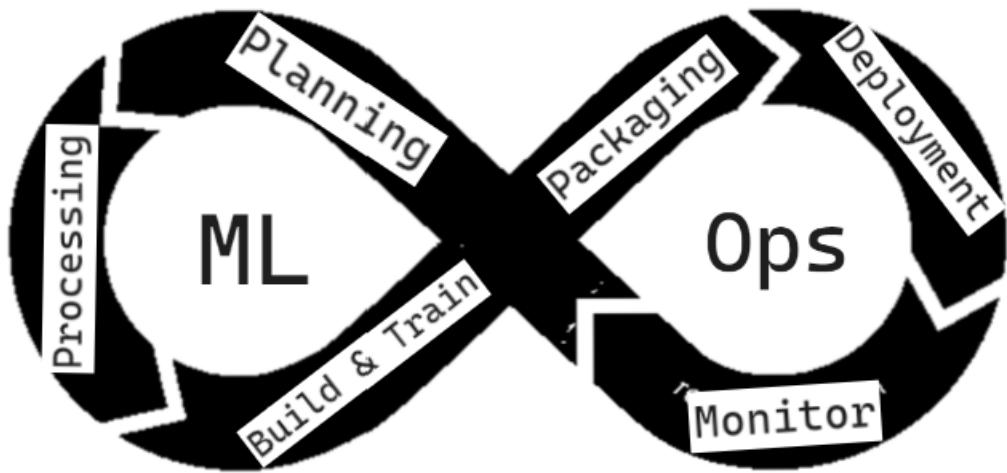
تستخدم وظيفة softmax كوظيفة تنشيط في طبقة الخرج لنماذج الشبكة العصبية التي تتبع بوزيع احتمالي متعدد الحدود . أي هي دالة رياضية تقوم بتحويل متوجه الأرقام إلى متوجه للاحتمالات ، حيث تتناسب احتمالات كل قيمة مع المقياس النسبي لكل قيمة في المتوجه [2].



(الشكل 2-6 :تابع softmax)

الفصل الثالث: الجزء العملي

تحدثنا في فقرة منهجية البحث عن عملية الترجمة و التي تبدأ بعملية المعالجة وتنتهي بمرحلة الحصول على model قادر على تحويل الجملة الانكليزية إلى فرنسية ولكن في هذا الفصل سوف نتحدث عن مفهوم أشمل وهو MLOps cycle و هي باختصار دورة حياة نموذج التعلم الآلي من مرحلة التخطيط إلى مرحلة الانتاج deployment والذي يوضحه الشكل التالي



(الشكل 1-3)

1-3 التخطيط planning

في هذه المرحلة تم العمل على :

- تقسيم أعضاء الفريق إلى عدة فرق:
1. فريق البحث (Research Team) المسؤول عن وضع معمارية النماذج و التحكم بمعاملات التدريب وتدريب النماذج ومراقبة تدريب النماذج.
 2. فريق (MLOps Team) المسؤول عن التعامل مع جميع الفرق وتأمين متطلباتهم من توفير أدوات لجعل مرحلة التدريب automation و مراقبة النماذج monitor والعمل على إيصال النماذج إلى مرحلة production .

3. فريق الاختبار (Testing Team) المسؤول عن اختبار النماذج والتأكد من كفاءتها.

- تقسيم بيئة العمل إلى بيئتين:

1. بيئة التطوير التدريب (Dev/Train)، وهنا سيتم تقسيم هذه المرحلة إلى بيئتين:

1. بيئة التدريب train وهي تتم على سيرفر التدريب الذي يحوي GPU وعلى اعتبار لا يوجد سيرفر مستقل لدينا، تم اعتبار سيرفر colab هو السيرفر الخاص بعملية التدريب.

2. بيئة التطوير dev وهي تتم على سيرفر مستقل (أجهزة الحاسب الخاص بنا) حيث يتم تحويل الأوزان الناتجة عن مرحلة التدريب إلى (Web Service).

2. بيئة الانتاج (Prod) حيث يتم فيها نشر (Web Service) على سيرفر production واختبارها وتحسينها.

- التركيز على معمارية المشروع وفقا لأفضل الممارسات في السوق الحالية، واختارنا التقسيم التالي تبعنا لاحتياجات المشروع:

(a) ملف preprocess.py: يحوي على التوابع التي تقوم بمعالجة الداتا.

(b) ملف test_preprocess.py: يحوي على اختبار لكل تابع في ملف preprocess.py.

(c) ملف models.py يحوي على عدة معماريات يمكن تحقيق أي معمارية للشبكة في هذا الملف.

(d) ملف train.yaml: يحوي على إعدادات الخاصة بعملية التدريب (عقل المشروع) مثل نوع معمارية النموذج ومكان وجود الداتا الخاصة بالجمل الانكليزية والفرنسية ومعدل التعلم وعدد العينات وbatch size و عدد epochs ونسبة validation من الداتا ونوع الجهاز الذي سيتم التدريب عليه .GPU or CPU

(e) ملف train.py: هو الملف الذي تم فيه عملية التدريب بعد تحديد الإعدادات في train.yaml من معالجة الداتا إلى توفير أوزان خاصة بعملية المعالجة وأوزان خاصة بعملية التدريب.

(f) ملف inference.py: يحتوى على اختبار بسيط للنموذج المدرب.

- (g) ملف requirements.txt: يحوي على python packages الخاصة بالمشروع مع رقم الأصدار لكل package.
- (h) ملف Dockerfile: يحوي على الإعدادات الخاصة بإنشاء بيئة معزولة لتشغيل المشروع من نظام تشغيل وكل المتطلبات التي يحتاجها المشروع بحيث يصبح كـ service قائمة بذاتها.
- (i) ملف start.sh: يقوم بتشغيل عدة Scripts لاختبار أدوات معالجة البيانات وبداية عملية التدريب واختبار بسيط بعد نهاية التدريب.
- (j) مجلد github.: يحوي على ملفات بصيغة yaml تستخدم في جعل عملية التدريب والتطوير والإنتاج بطريقة مؤتمتة automation.
- (k) مجلد data: يحوي على الداتا المستخدمة في المشروع والتي تضم الجمل الانكليزية مع مقابلاً لها من اللغة الفرنسية.
- (l) مجلد fastapi: يحتوى على الملفات الخاصة بتحويل model إلى (Web Service).
- (m) مجلد outputs: يحتوى على الأوزان الخاصة بعملية المعالجة والتدريب
- (n) مجلد papers_research: يحتوى على الأوراق البحثية التي تم اعتمادها في المشروع إلى هنا نكون قد انتهينا من مرحلة التخطيط (planning) وسوف ننتقل إلى المرحلة الثانية الخاصة بتعريف الداتا ومعالجتها (Data and Processing).
- ### 3-2 الداتا ومعالجتها
- في هذه المرحلة تم العمل على :
- التعرف على الداتا المستخدمة في عملية التدريب والتي توجد في المجلد data حيث يوجد ملفين ملف خاص بالجمل الانكليزية small_vocab_en وملف خاص بالجمل الفرنسية small_vocab_fr . عدد الجمل في كل منها هو 137861
 - تعريف التابع max_seq في ملف preprocess.py ومهمة هذا التابع هو تحديد أكبر طول جملة قبل معالجة الداتا لكي تكون الجمل ذات طول واحد.

- تعريف التابع `read_data` في ملف `preprocess.py` ومهمة هذا التابع هو قراءة كل سطر في الملف وتخزين الجمل في متحول نهائي ويأخذ المسار الكامل للملف كمحول.
- تعريف التابع `show_some_sample_and_some_statics` في ملف `preprocess.py` ومهمة هذا التابع بقراءة ملف `small_vocab_fr,small_vocab_en` ثم طباعة مجموعة من الجمل كل جملة انكليزية ومقابلتها من اللغة الفرنسية وبعدها طباعة عدد الجمل الموجودة في كل ملف ويتم اختبار التابع في ملف `test_preprocess.py` كما في الشكل (2-3).
- تعريف التابع `convert_text_to_tokenize` في ملف `preprocess.py` ومهمة هذا التابع هو بناء قاموس لمجموعة الجمل التي يتم إعطاءها للتابع وكذلك ترميز مجموعة الجمل بعد بناء هذا القاموس وتم اختبار التابع في ملف `test_preprocess.py` كما في الشكل (2-3).
- تعريف التابع `padding` في ملف `preprocess.py` ومهمة هذا التابع هو جعل سلسلة الجمل ذات طول واحد عبر إضافة أصفار أما على يمين السلسلة أو يسارها ويتم التحكم بالطول النهائي باستخدام المتحول `length_of_pad` وكذلك `type_pad` لاختيار يمين أو يسار وتم اختبار التابع في ملف `test_preprocess.py` كما في الشكل (2-3).
- تعريف التابع `full_process` في ملف `preprocess.py` ومهمة هذا التابع هو استخدام التوابع السابقة في معالجة الداتا بشكل كامل من تحديد قاموس لكاميل الداتا (Tokenizer) ثم مرحلة (padding) لجعل الجمل بنفس الطول ومن ثم تعديل أبعاد أحجام المصفوفات الناتجة لكي تلائم شكل الدخل والخرج للشبكة العصبية وتم اختبار التابع في ملف `test_preprocess.py` كما في الشكل (2-3).
- تعريف التابع `ids_to_words` في ملف `preprocess.py` ومهمة هذا التابع هي تحويل أرقام كل خرج الشبكة إلى ما يقابلها من كلمات في اللغة الفرنسية وتم اختبار التابع في ملف `test_preprocess.py` كما في الشكل (2-3).
- بالإضافة إلى اختبار التوابع السابقة في ملف `test_preprocess.py` كما في الشكل (2-3) كذلك تم طباعة بعض العبارات التوضيحية لفهم الخرج بشكل أفضل.

```

=====show_some_sample_and_some_statics=====
▶ sample from 1 to 3
sample 1: new jersey is sometimes quiet during autumn , and it is snowy in april .
⇨ sample 1: new jersey est parfois calme pendant l' automne , et il est neigeux en avril .
sample 2: the united states is usually chilly during july , and it is usually freezing in november .
sample 2: les états-unis est généralement froid en juillet , et il gèle habituellement en novembre .
sample 3: california is usually quiet during march , and it is usually hot in june .
sample 3: california est généralement calme en mars , et il est généralement chaud en juin .

=====Same Statics=====
number sentence English is : 137861
number sentence French is : 137861

=====convert_text_to_tokenize=====
index for each word
{'i': 1, 'nlp': 2, 'love': 3, 'am': 4, 'learning': 5, 'how': 6, 'are': 7, 'you': 8}

=====padding=====
sample 1: I love Nlp
token and sequence: [1 3 2]
padding sequences: [0 1 3 2]

=====
sample 2: I am learning Nlp
token and sequence: [1 4 5 2]
padding sequences: [1 4 5 2]

=====
sample 3: How are You
token and sequence: [6 7 8]
padding sequences: [0 6 7 8]

=====full_process=====
Max English sequence length before processing: 15
Max French sequence length before processing: 21
Max English sequence length after processing: 21
Max French sequence length: after processing: 21
English vocabulary size: 199
French vocabulary size: 344

=====ids_to_words=====
il en il est

```

(الشكل 2-3)

إلى هنا نكون قد انتهينا من مرحلة الاداتا ومعالجتها Data and Processing وسوف ننتقل إلى المرحلة الثالثة الخاصة ببناء عدة معماريات للنموذج وتدريبها (Build and Train Models).

3-3 بناء وتدريب النماذج Build and Train Models

في هذه المرحلة تم العمل على :

(1) في ملف train.yaml كما تحدثنا في مرحلة التخطيط يحوي على إعدادات الخاصة بعملية التدريب (عقل المشروع) والذي يستطيع الباحث أو المطور التحكم بكامل عملية التدريب من خلاله حيث تم تعريف المتغيرات التالية :

(a) en_url: هو متحول string يمثل مسار الملف الذي يحوي الجمل باللغة الانكليزية.
(b) fr_url: هو متحول string يمثل مسار الملف الذي يحوي الجمل المقابلة باللغة الفرنسية.
(c) type_model: هو متحول string يحوي على اسم معمارية الشبكة التي ستقوم عملية التدريب عليها ونستطيع استخدام أحد هذه المعماريات عبر تغيير الاسم فقط وتحوي أربع قيم :

- .a .Rnn تمثل شبكة Rnn
- .b .Rnn_Embd تمثل شبكة Rnn مع استخدام Word Embedding
- .c .Lstm تمثل شبكة Lstm
- .d .Lstm_Embd تمثل شبكة Lstm مع استخدام Word Embedding

(d) Learning_rate : هو متحول float يمثل معدل التعلم يفضل أن تكون قيمته بين 0 و 1.
(e) number_sample: هو متحول integer يمثل عدد العينات المستخدمة في عملية التدريب.

(f) batch_size : هو متحول integer يمثل عدد العينات التي يتم إرسالها في كل دفعة خلال عملية التدريب يفضل أن يكون صغير لكي لتجنب استخدام الذاكرة بشكل كبير.
(g) epochs : هو متحول integer يمثل عدد مرات تدريب النموذج على مجموعة العينات.
(h) validation_split : هو متحول float يمثل نسبة داتا الاختبار من كامل الاداتا المستخدمة.

(i) CPU على سواء تم التدريب على string تحديد لـ `type_device` تم اذا ماذا على عملية التدريب تم `string` هو متحول.

أو GPU وذلك بحسب التوفيرية لدى الباحث أو المطور ويتم استخدام نفس الاسماء لتحديد نوع الجهاز.

(j) `length_vector_word` هو متحول `integer` يمثل طول شعاع الكلمة الذي نريد الحصول عليه في طبقة (Word Embedding).

(k) `n_neuros_rnn` هو متحول `integer` يمثل عدد الحالات المخفية `hidden state` التي تعطى كل طبقة (RNN) للطبقة التالية.

(l) `n_neuros_lstm` هو متحول `integer` يمثل عدد الحالات المخفية `hidden state` التي تعطى كل طبقة (LSTM) للطبقة التالية.

(m) `n_neuros_timedistributed` هو متحول `integer` يمثل عدد العصبونات في طبقة (Time Distributed).

(2) في ملف `models.py` كما تحدثنا في مرحلة التخطيط أن هذا الملف سوف يستخدم لتحقيق أي معمارية للشبكة، لذلك تم العمل على:

1. بناء تابع `motor` يأخذ هذا التابع المتحوالت الخاصة ببناء معمارية الشبكة مثل:
 - (a) حجم مصفوفة الدخل الخاصة باللغة الانكليزية `input_shape`
 - (b) عدد كلمات اللغة الانجليزية الموجودة في الداتا المستخدمة `dict_en_size`
 - (c) عدد كلمات اللغة الفرنسية الموجودة في الداتا المستخدمة `dict_fr_size`
 - (d) باقي المتحوالت تم شرحها في ملف `train.yaml` تحت بند `Structural Models`.

2. في التابع السابق تم تحقيق 4 معماريات للشبكة كل معمارية يتم أخذها باستخدام

الموارد في ملف `train.yaml`, في حال كان يساوي:

(a) `type_model="Rnn"` اذا الشبكة المستخدمة هي RNN ولا يوجد طبقة `Word Embedding`, والطبقات على الترتيب التالي:

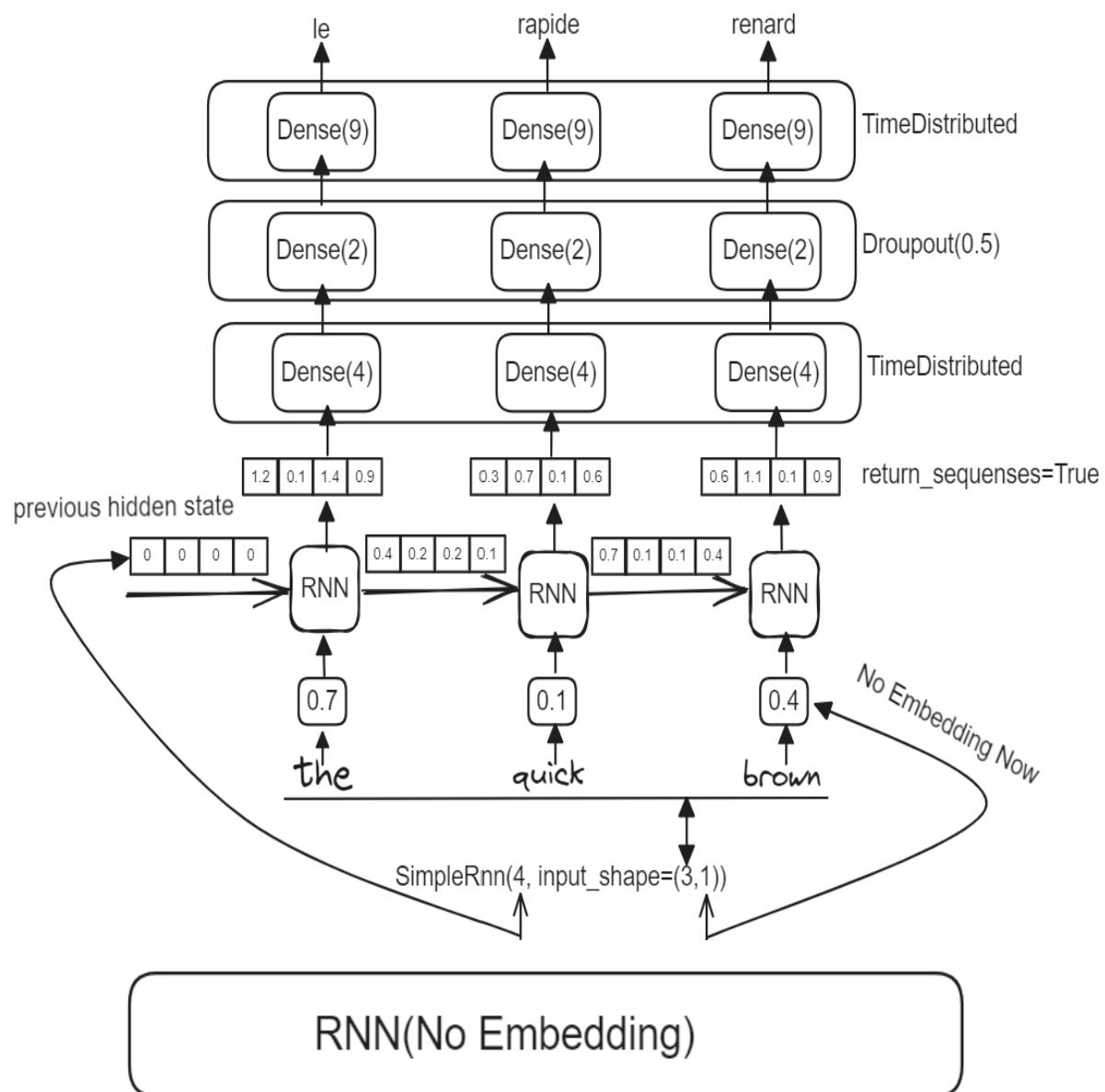
1. طبقة `SimpleRnn` حيث تأخذ `n_neuros_rnn` الذي يمثل عدد الحالات المخفية بين الطبقات و `input_shape` ثنائية الجزء الأول يضم عدد الكلمات في

الجملة ومثل أيضا time steps عدد العصبونات الموجود في طبقة rnn والتي سوف يهتم كل عصبون فيها بكلمة واحدة من الجملة والجزء الآخر من الثنائيه يمثل طول شعاع الكلمة و تم وضع return_sequences على القيمة true لكي يقوم كل عصبون بإرجاع hidden states الخاص به، الشكل (3-3) يوضح آلية الطبقة.

2. طبقة TimeDistributed تكمن فائدة هذه الطبقة في أنها لو لم تكن موجوده كان سيتم فقط معالجة hidden state لأخر عصبون وهذا سيء لأننا نريد تحديد كلمة الخرج باللغة الفرنسية بناء على الكلمة الانكليزية الحالية التي وصلنا لها وليس بناء على كامل الجملة (بكلمات أخرى لو كانت المهمة تصنيف الجمل إلى خبر كاذب و حقيقي أي 0 أو 1 كان باستطاعتنا استخدام Dense على آخر hidden state خارج من الطبقة الأخيرة لأن المشكلة هي Many-To-One ولكن في حالتنا باستخدام TimeDistributed طبقنا Dense على كل hidden state لأن المشكلة (Many-To-Many) هذه الطبقة تأخذ n_neurons_ timedistributed كعدد عصبونات لكل dense، الشكل (3-3) يوضح آلية الطبقة.

3. طبقة Dropout مع قيمة 0.5 لمنع حصول over fitting في الشبكة، الشكل (3-3) يوضح آلية الطبقة.

4. طبقة TimeDistributed ولكن نقوم باعطاءه dict_fr_size (عدد كلمات اللغة الفرنسية في الداتا) كعدد عصبونات لكل dense مع تابع التعديل softmax لإعطاء مصفوفة احتمالات الكلمات في كل dense وبعدها يتم اختيار الكلمة اعتماد على التابع ids_to_words المعرف مسبقا في مرحلة المعالجة والذي يقوم بتحويل كل مصفوفة احتمالات بعد اختيار أكبرها باستخدام argmax إلى الكلمة الفرنسية المقابلة للرقم اعتماد على القاموس ، الشكل (3-3) يوضح آلية الطبقة.



(الشكل 3-3)

اذا الشبكة المستخدمة هي RNN مع طبقة type_model="Rnn_Embd" (b

, والطبقات على الترتيب التالي: (Word Embedding)

1. طبقة Embedding حيث تأخذ dict_en_size الذي يمثل عدد الكلمات الانجليزية

المستخدمة في الداتا و length_vector_word تم شرحه سابقا في ملف train.yaml يمثل طول شعاع الكلمة الذي نريد الحصول عليه و input_length يمثل طول جملة موجود في الداتا سواء جملة فرنسية أو انكليزية وهي 21 في حالتها

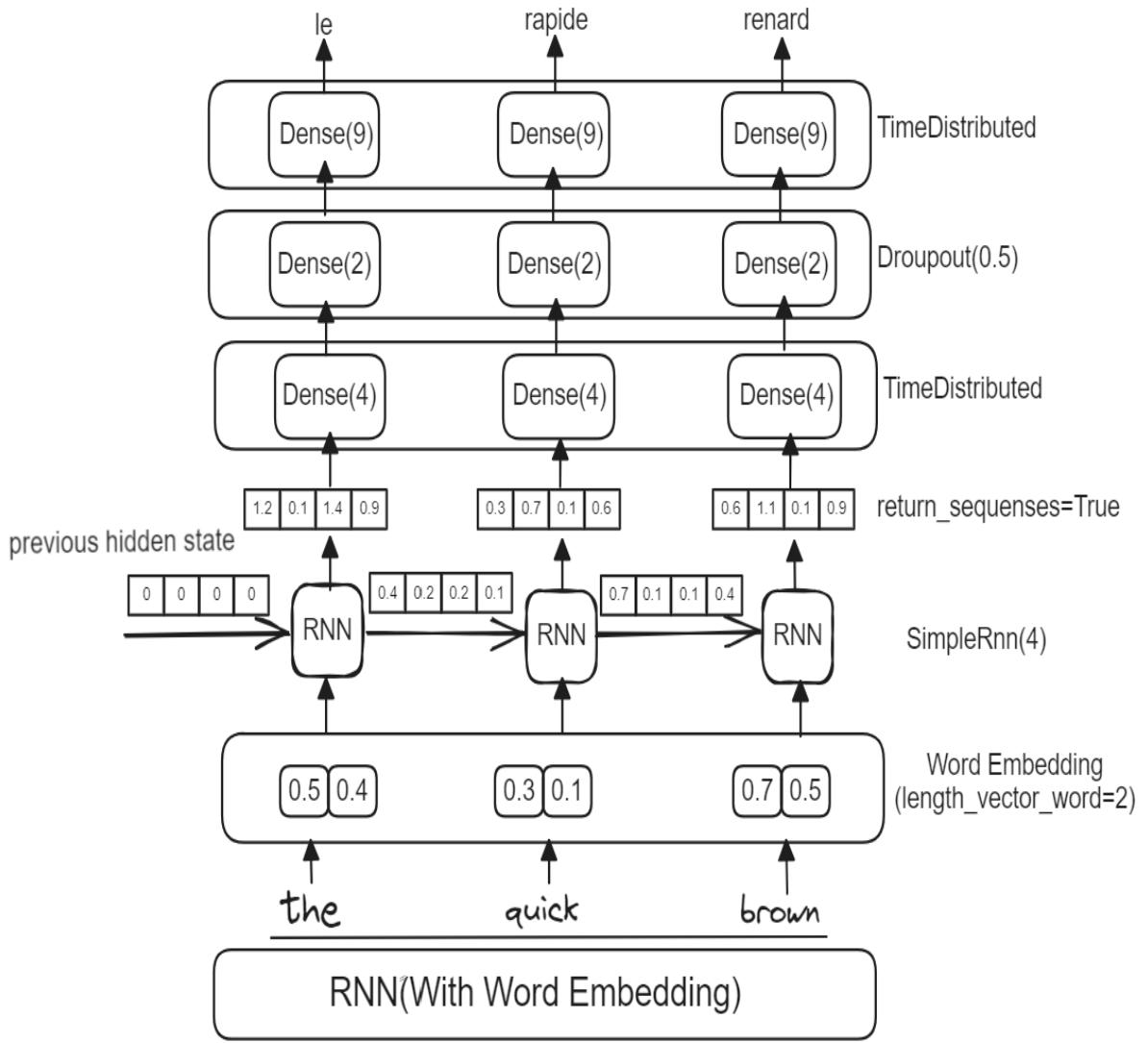
و input_shape ثانية الجزء الأول يضم عدد الكلمات في الجملة والجزء الآخر من الثانية يمثل طول شعاع الكلمة وهو 1

2. طبقة SimpleRnn حيث تأخذ n_neuro_rnn الذي يمثل عدد الحالات المخفية بين الطبقات وتم وضع return_sequences على القيمة true لكي يقوم كل عصبون بإرجاع hidden states الخاص به، الشكل (3-4) يوضح آلية الطبقة.

3. طبقة TimeDistributed نفس آلية الطبقة في المعمارية RNN، وتم شرحها بشكل مفصل، الشكل (4-3) يوضح آلية الطبقة.

4. طبقة Dropout مع قيمة 0.5 لمنع حصول over fitting في الشبكة، الشكل (3-4) يوضح آلية الطبقة.

5. طبقة TimeDistributed نفس آلية الطبقة في المعمارية RNN، وتم شرحها بشكل مفصل، الشكل (4-3) يوضح آلية الطبقة.

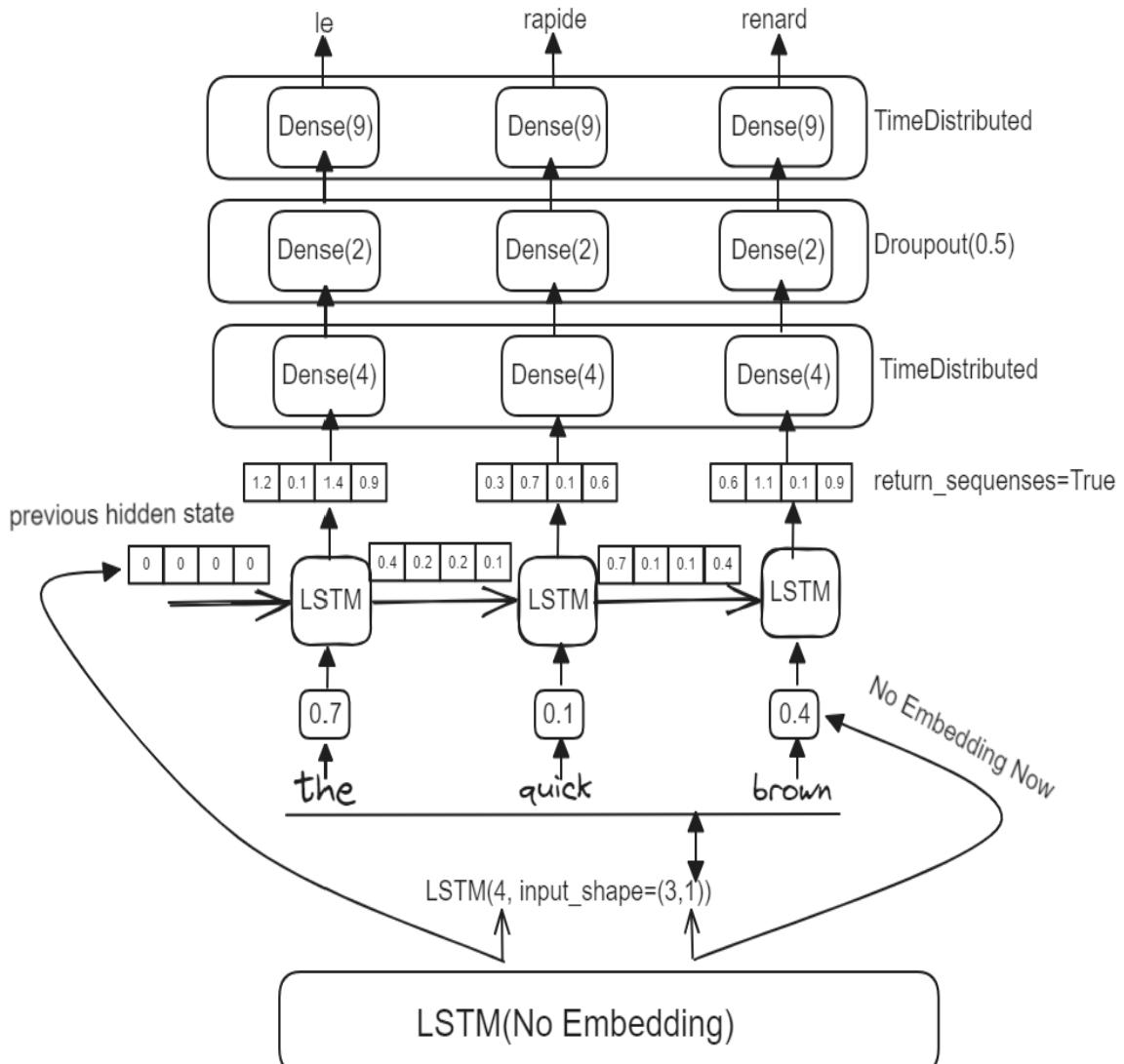


(الشكل 4-3)

اذا الشبكة المستخدمة هي LSTM ولا يوجد طبقة type_model="Lstm" (C

، والطبقات على الترتيب التالي: (Word Embedding)

1. طبقة LSTM حيث تأخذ n_neurons_lstm الذي يمثل عدد الحالات المخفية بين الطبقات و input_shape ثنائية الجزء الأول يضم عدد الكلمات في الجملة ومثل أيضا time_steps عدد العصبونات الموجود في طبقة lstm والتي سوف يهتم كل عصبون فيها بكلمة واحدة من الجملة والجزء الآخر من الثنائية يمثل طول شعاع الكلمة و تم وضع return_sequences على القيمة true لكي يقوم كل عصبون بإرجاع hidden states الخاص به، الشكل (5-3) يوضح آلية الطبقة.
2. طبقة TimeDistributed نفس آلية الطبقة في المعمارية RNN، وتم شرحها بشكل مفصل، الشكل (5-3) يوضح آلية الطبقة.
3. طبقة Dropout مع قيمة 0.5 لمنع حصول over fitting في الشبكة، الشكل (5-3) يوضح آلية الطبقة.
4. طبقة TimeDistributed نفس آلية الطبقة في المعمارية RNN، وتم شرحها بشكل مفصل، الشكل (5-3) يوضح آلية الطبقة.



(الشكل 5-3)

اذا الشبكة المستخدمة هي LSTM مع طبقة type_model="Lstm_Embd" (d

:Word Embedding)

1. طبقة Embedding حيث تأخذ dict_en_size الذي يمثل عدد الكلمات الانجليزية

المستخدمة في الداتا و length_vector_word تم شرحه سابقا في ملف

train.yaml يمثل طول شعاع الكلمة الذي نريد الحصول عليه و input_length

يمثل طول جملة موجود في الداتا سواء جملة فرنسية أو انكليزية وهي 21 في

حالتنا input_shape ثانية الجزء الأول يضم عدد الكلمات في الجملة والجزء

الأخر من الثانية يمثل طول شعاع الكلمة وهو 1

2. طبقة LSTM حيث تأخذ n_neurons_lstm الذي يمثل عدد الحالات المخفية

بين الطبقات وتم وضع return_sequences على القيمة true لكي يقوم كل

عصبون بإرجاع hidden states الخاص به، الشكل (6-3) يوضح آلية الطبقة.

3. طبقة TimeDistributed نفس آلية الطبقة في المعمارية RNN، وتم شرحها بشكل

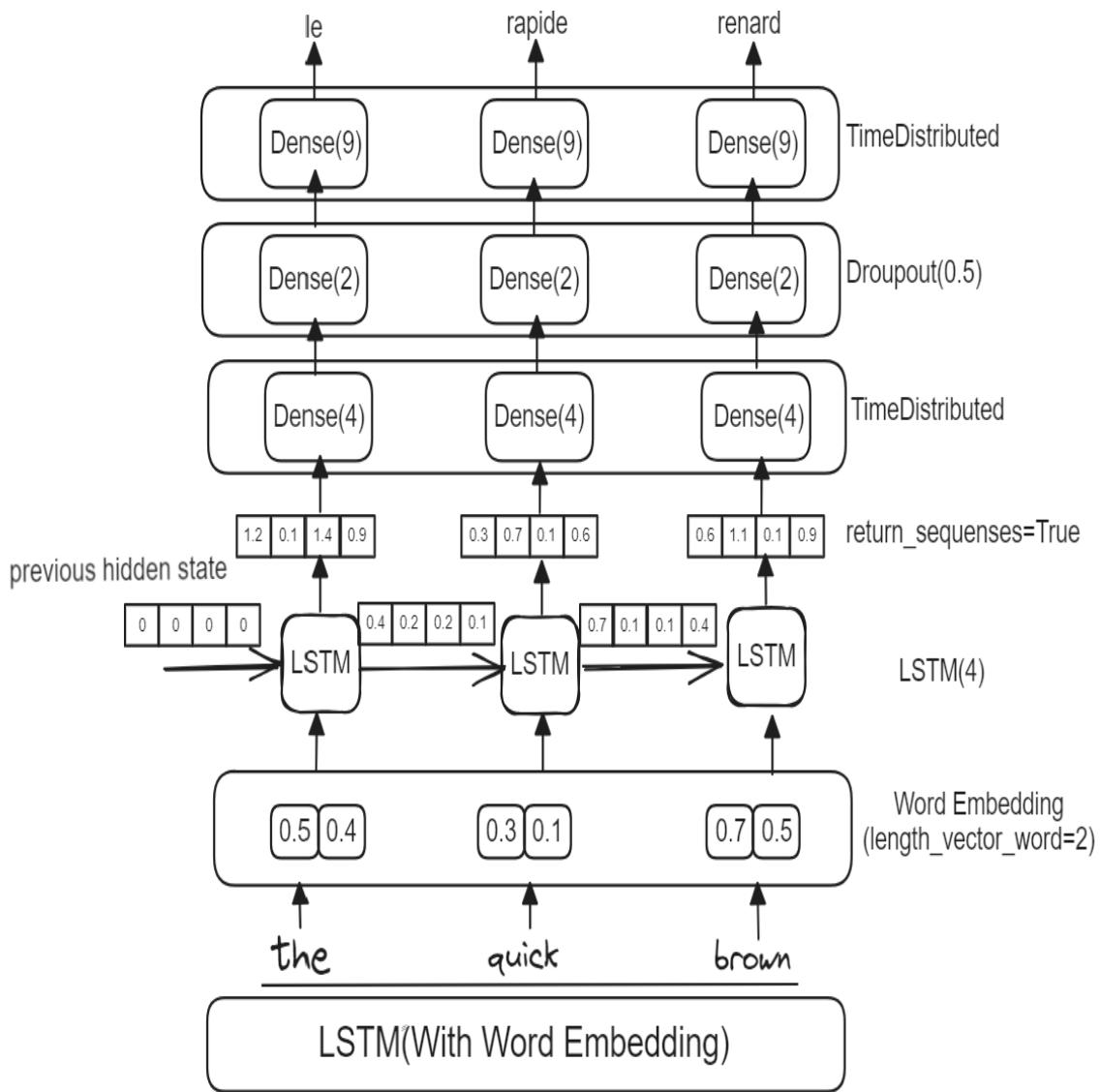
مفصل، الشكل (3-6) يوضح آلية الطبقة.

4. طبقة Dropout مع قيمة 0.5 لمنع حصول over fitting في الشبكة،

الشكل (6-3) يوضح آلية الطبقة.

5. طبقة TimeDistributed نفس آلية الطبقة في المعمارية RNN، وتم شرحها بشكل

مفصل، الشكل (3-6) يوضح آلية الطبقة.



(الشكل 6-3)

3. أيضاً في التابع السابق تم استخدام تحديد يتم طباعة المعمارية التي تم استخدامها مع حجم وأوزان كل طبقة باستخدام التابع Summary المعروف في keras وكان المعماريات مع اسم كل معمارية مرافق ويوضحها الشكل (7-3).

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 21, 2)	400
simple_rnn (SimpleRNN)	(None, 21, 256)	66304
time_distributed (TimeDistr ibuted)	(None, 21, 1024)	263168
dropout (Dropout)	(None, 21, 1024)	0
time_distributed_1 (TimeDis tributed)	(None, 21, 345)	353625

Total params: 683,497

Trainable params: 683,497

Non-trainable params: 0

RNN(Word Embedding)

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 21, 2)	400
simple_rnn (SimpleRNN)	(None, 21, 256)	66304
time_distributed (TimeDistr ibuted)	(None, 21, 1024)	263168
dropout (Dropout)	(None, 21, 1024)	0
time_distributed_1 (TimeDis tributed)	(None, 21, 345)	353625

Total params: 683,497

Trainable params: 683,497

Non-trainable params: 0

LSTM(Word Embedding)

Model: "sequential"

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 21, 256)	66048
time_distributed (TimeDistr ibuted)	(None, 21, 1024)	263168
dropout (Dropout)	(None, 21, 1024)	0
time_distributed_1 (TimeDis tributed)	(None, 21, 345)	353625

Total params: 682,841

Trainable params: 682,841

Non-trainable params: 0

RNN

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 21, 256)	264192
time_distributed (TimeDistr ibuted)	(None, 21, 1024)	263168
dropout (Dropout)	(None, 21, 1024)	0
time_distributed_1 (TimeDis tributed)	(None, 21, 345)	353625

Total params: 880,985

Trainable params: 880,985

Non-trainable params: 0

LSTM

(الشكل 7-3)

4. في التابع السابق تم استخدام compile لتعريف توابع loss,metrics,optimizer و يقوم بإنجاز motor بعد كمال التجهيز، وبذلك تكون قد انتهينا من الحديث عن ملف models.py.

(3) في ملف train.py كما تحدثنا في مرحلة التخطيط أن هذا الملف سوف يستخدم في عملية التدريب، لذلك تم بناء التابع train الذي يأخذ متحول config_path وهو مسار الملف train.yaml الذي يتضمن إعدادات عملية التدريب، بعد تعريف الإعدادات في ملف train.yaml وتعریف المعماريات في ملف models.py وتتابع المعالجة في ملف preprocess.py، يقوم التابع train بالتالي

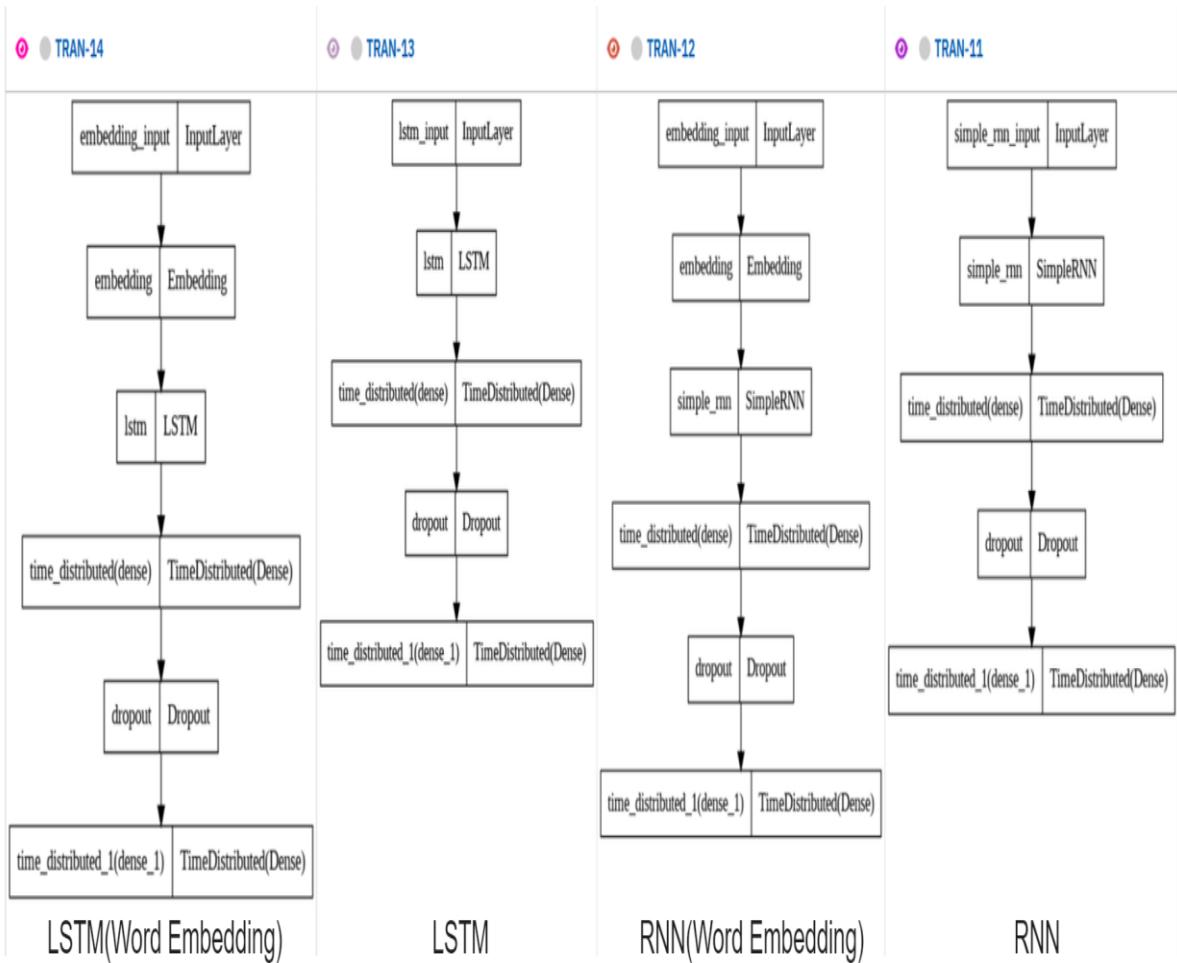
1. قراءة جميع المتغيرات في ملف train.yaml باستخدام المكتبة yaml وإسنادها إلى متغيرات مطابقة لها بالاسم منعاً من الصياغ.

2. بدء مرحلة معالجة البيانات من خلال التوابع التي عرفناها في الملف preprocess.py، وذلك بقراءة البيانات المستخدمة على مرتين الانكليزية ثم الفرنسية باستخدام read_data واستخدام التابع full_process لتطبيق مرحلة padding و tokenizer.

3. قبل البدء بمرحلة التدريب تم تخزين مرحلتي المعالجة بما يخص مرحلة tokenizer لاستفادتها من القاموس الخاص باللغتين في مرحلة التحول الكامل بعد التدريب وتم الحفظ على شكل ملفين tokenizer_fr.pkl و tokenizer_en.pkl.

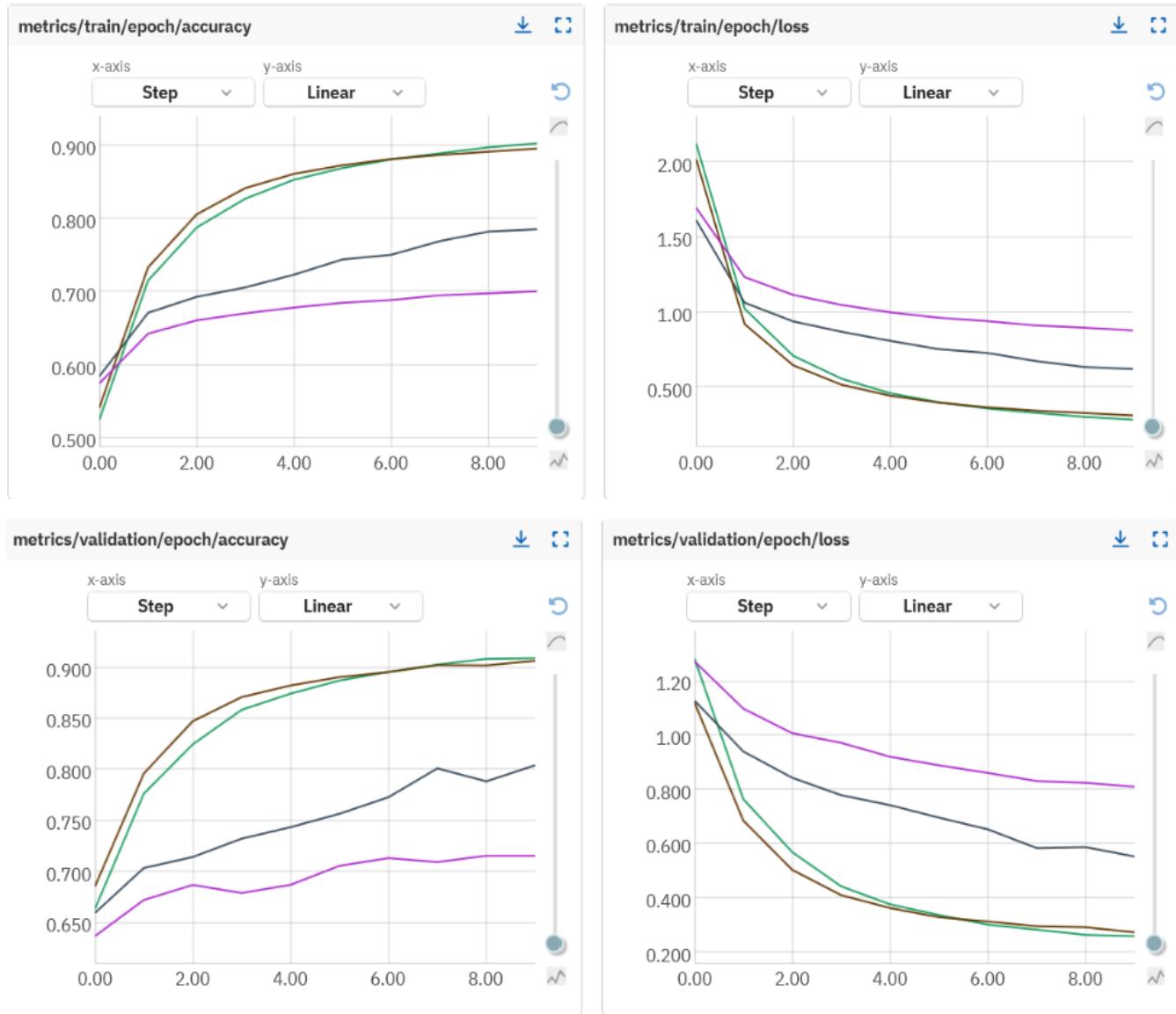
4. الحصول على معمارية الشبكة باستخدام التابع type_model عبر motor كما ذكرنا وكذلك تجهيز طبقات المعمارية المختارة باستخدام متغيرات Structural Models.
5. أما الآن سوف نتعرف على النوع الأول من monitoring في مشروعنا، حيث كان يجب عنه الحديث في مرحلة خاصة تحت مسمى Monitior، ولكن منعاً لتشتت تناسق الأفكار الخاصة بالمشروع، يستخدم هذا النوع لجعل حياة Research أسهل وتقع عاتق توفير هذه المهمة على فريق MLOps، فهو يوفر طريقة لمراقبة عدة نماذج تقوم بتدريبها على التقرير في مرحلة ومراقبة مخططات الخطأ و الكفاءة وكذلك مراقبة حالة CPU و GPU و Memory وبالتالي هذا يساعدنا على إطلاق الحكم على استخدام النموذج الأفضل بين النماذج للقيام بعملية تطويره لاحقاً، لذلك سوف نقوم أداة Neptune.ai مع keras ب استخدام التابع init_run في neptune لتجهيز المشروع باستخدام

5. بعد ذلك يوجد متتحول آخر name project="ammar.mllops/translate-en-to-fr" يأخذ قيمة type_model الحالية تكمن فائدة هذا المتتحول في تسمية كل عملية تدريب وبالتالي فRnn لها نتائج مستقلة عن RNN_Embd هذا التنظيم يساعد في التتبع الجيد لعمل كل model ضمن المشروع الواحد والمتحول الاخير api_token يمكن اعتباره بمثابة باسورد تعطي سماحية لمستخدمها من الولوج إلى موقع neptune وتخزين نتائج تدريب النماذج عليه، بعد ذلك تم ربط keras مع neptuneتعريف callbacks حيث يقوم هذا المتتحول بكتابه كل عملية التدريب من قيم الخطأ و الكفاءة ورسم مخططات في موقع neptune.
6. بعد ذلك يتم استخدام GPU أو CPU عبر المتتحول type_device, وبداية عملية التدريب عبر التابع fit المعرف في keras والذي يأخذ مت حولات تم شرحها بالتفصيل سابقا خالل المراحل السابقة مثل epochs و batch_size و process_output و process_input و callbacks الذي عرفناه مسبقا لتحصل عملية الرابط بين keras و neptune.
7. تم استخدام التابع save في keras لحفظ model وتم استخدام type_model ليختلف اسم النموذج المدرب وبالتالي يسهل التعرف عليه واستخدامه.
8. تم كتابة train.py على شكل script وهذا ما يناسب بيئه التدريب عند العمل على سيرفر مستقل أو colab بحيث يكون هذا الملف قابل للعمل في أي مكان يوضح فيه ويتم تنفيذه بالشكل التالي وتسمح لنا المكتبة argparse بإعطاءه config_path كمت حول خارجي والتنفيذ يكون على الشكل التالي:
- ```
$python train.py --config "/content/machine_translate/train.yaml"
```
9. بعد التنفيذ دخلنا إلى موقع neptune.ai وقمنا بمراقبة عمل النماذج على التوازي وكذلك مراقبة مخططات الخطأ والكفاءة وحصلنا على النتائج التالية، في البداية نستعرض مخططات المعماريات على التوازي ويوضحها الشكل (3-8)



(الشكل 8-3)

وكذلك مخططات loss و accuracy في كل epochs بالنسبة للأربع معماريات حيث نلاحظ أن بالنسبة لمخطط accuracy في الشكل (كذا) على اليسار سواء train أو validation أن الخطين البيانيين في أعلى كل مخطط يحققان أعلى زيادة قيمة epochs فيما الخطين البيانيين في أسفل كل مخطط يتحققان أقل وهذا يدل على أن النماذج تتعلم أفضل سواء LSTM أو RNN عند استخدام طبقة validation loss في الشكل (9-3) على اليسار سواء train أو validation (Word Embedding) تعطي نفس الحقيقة حيث الخطين في أسفل كل مخطط يتحققان loss أقل عند تقدم epochs وبالتالي طبقة (Word Embedding) حققت فرق ملموس وواضح بين المعماريات التي تستخدم هذه الطبقة والمعماريات التي لا تستخدمها



(الشكل 9-3)

### 3-3-1 النتائج

من خلال عملية الربط بين keras و neptune نستطيع الحصول على جدول لنتائج المقارنة بين الأربع معماريات الشكل(3-10)، لن نستعرض كامل الجدول سوف نستعرض ما يهمنا من عملية التدريب في الأربع معماريات وأي معمارية أفضل بالنسبة للنتائج الحالية ، حيث على الترتيب الجدول على مستوى الأعمدة كل عمود له اسم يمثل رقم فريد للمعمارية المستخدمة، على مستوى الاسطر في السطر الأول اسم النموذج المدرب وبعدها قيمة الكفاءة بالنسبة ل train و validation وقيمة الخطأ بالنسبة

نلاحظ نفس الحقيقة التي وجدناها في المخططات نجدها في الجدول الشكل(كذا)، المعماريات validation التي تستخدم (Word Embedding) أي العمود ذات الاسم tran12 و tran14 أفضل في عملية التعلم وبالتالي سوف تكون أفضل في عملية التحول الكامل

|                                   |      | TRAN-11  | TRAN-12  | TRAN-13   | TRAN-14  |
|-----------------------------------|------|----------|----------|-----------|----------|
| Name                              | Rnn  | Rnn_Embd | Lstm     | Lstm_Embd |          |
| metrics/train/epoch/accuracy      | last | 0.700792 | 0.895928 | 0.785602  | 0.903068 |
| metrics/validation/epoch/accuracy | last | 0.715914 | 0.906964 | 0.804588  | 0.909544 |
| metrics/train/epoch/loss          | last | 0.880449 | 0.314011 | 0.623283  | 0.28635  |
| metrics/validation/epoch/loss     | last | 0.81224  | 0.275342 | 0.554936  | 0.261188 |

(الشكل3-10)

بعد ذلك يتم اختبار النموذج بعد عملية التدريب ويتم ذلك عن طريق الملف inference.py حيث يقوم هذا الملف بتحويل جملة من اللغة الانكليزية إلى الفرنسية بعد اختيار احد النماذج الاربعة التي تم تدريبيها وإعطاءه رابط النموذج وكذلك تمرير الجملة المراد ترجمتها، فيقوم بتحميل الاوزان الموجودة في مجلد outputs، طبعاً مكتبة argparse هي التي تسمح لنا بهذا النوع من العمليات، بعد تنفيذ الملف نلاحظ الخرج الذي يمثل الجملة المراد ترجمتها والترجمة المتوقعة.

```
>python inference.py --model_path "/content/machine_translate/outputs/model_Lstm_Embd.h5"
--text "new jersey is sometimes quiet"
new jersey is sometimes quiet
=====prediction=====
new jersey est parfois calme
```

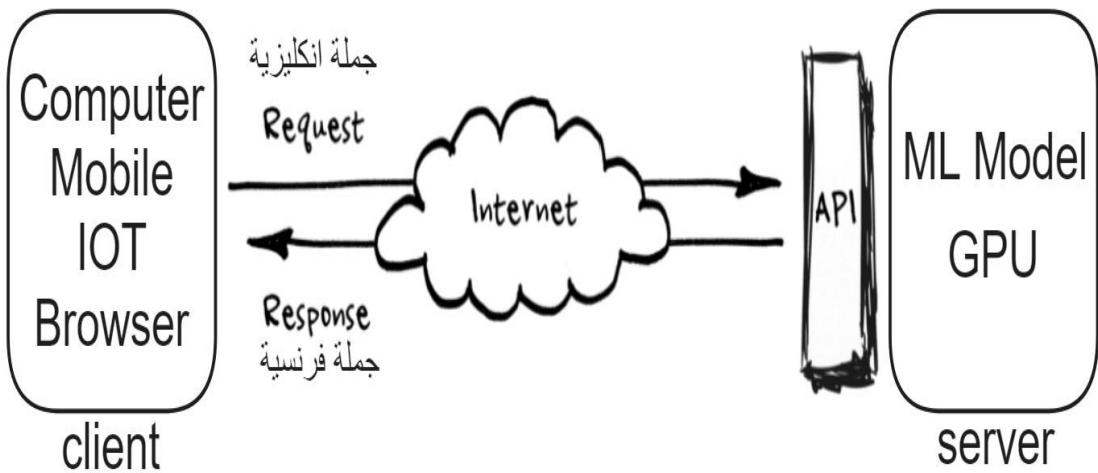
إلى هنا نكون قد انتهينا من مرحلة بناء وتدريب النماذج Build and Train Models وسوف ننتقل إلى المرحلة الرابعة الخاصة بمرحلة التعبئة (packaging)

### 4-3 التعبئة packaging

في هذه المرحلة يتم تجميع جميع المكونات والتطبيقات اللازمة لتشغيل تطبيق أو نموذج ذكاء صناعي في حزمة مستقلة. وفي حالتنا سوف نقوم باستخدام fastapi وذلك لتعريف API تقوم بتحميل نموذج وتنفيذ الترجمة واستخدام Docker لتعريف البيئة المعزولة التي تحتوي على جميع التبعيات والأدوات اللازمة لتشغيل النموذج، وبالتالي لدينا مرتبتين هما:

#### (1) مرحلة تحويل ML Model إلى API

ما هو API؟ هو مجموعة من البرمجيات والقواعد والبروتوكولات التي تسمح للتطبيقات بالتواصل والتفاعل مع بعضها البعض، يعتبر API نقطة اتصال بين تطبيقين مختلفين أو بين تطبيق وخدمة خارجية. يستخدم التطبيق المصدر (client) API للوصول إلى وظائف أو بيانات تقدمها التطبيق المستهدف (server)، الشكل(3-11) يشرح الآلية حيث يتم إرسال الجملة الانكليزية بصيغة json مثلًا عبر الأنترنت إلى API حيث يقوم بمعالجة الطلب على server ومن ثم يقوم API بإعادة الجملة الفرنسية بصيغة json أيضًا عبر الأنترنت إلى client من الممكن أن يكون أي جهاز يملك الحد الأدنى لوصف آلة تعمل بالكهرباء وتملك معالج ذاكرة ويمكنها الاتصال بالانترنت مثل computer, mobile, sensor..etc



(الشكل 11-3)

بعد شرح API ننتقل الأن إلى تقديم شرح عن دمج ml model مع fastapi وهو كما ذكرنا سابقاً إطار عمل لبناء API ويتم ذلك في مجلد fastapi والذي يحوي:

- a) المجلد الأول www يملك ملف init\_fastapi.py يقوم بأخذ object من FastAPI الذي يمثل تطبيق منفصل وذلك لتعريف المسارات والوظائف المرتبطة بهذا التطبيق حيث في تطبيقنا لدينا وظيفة واحدة وهي تحويل الجملة من الانكليزية إلى الفرنسية.
- b) المجلد الثاني routes في هذا المجلد يتم تعريف المسارات لاستقبال الطلبات واستخدام التوابع المناسبة لمعالجة الطلب على server ، وفي ملف requests قمنا ببناء التابع lang حيث تستقبل english\_text مرسلاً من قبل client قبل en\_to\_fr لمعالجة الطلب والذي يتم في المجلد الثالث ويتكون باستخدام التابع en\_to\_fr لمعالجة الطلب والذي يتم في المجلد الثالث ثم تقوم بإرجاع الجملة الفرنسية إلى client.
- c) المجلد الثالث handlers وهو المجلد الذي يقوم بمعالجة الطلب على server، حيث في ملف lang.py ضمن المجلد نقوم بتحميل الأوزان الخاصة بالمعالجة مثل tokenizer\_en.pkl وtokenizer\_fr.pkl وبعد ذلك تم اختيار النموذج الأكبر دقة وهو LSTM(Word Embedding).

التابع predict الذي يقوم باستخدام tokenizer ثم padding ثم en\_to\_fr المعرفة في keras ثم تحويل الأرقام كلمات باللغة الفرنسية ثم إعادة الناتج.

(d) ملف server.py والذي يتم فيه تعريف Port Number + IP address لإنشاء عملية الاتصال عندما يريد client التخاطب مع .FastAPI server

(e) ملف start.sh لبدء عملية تشغيل server

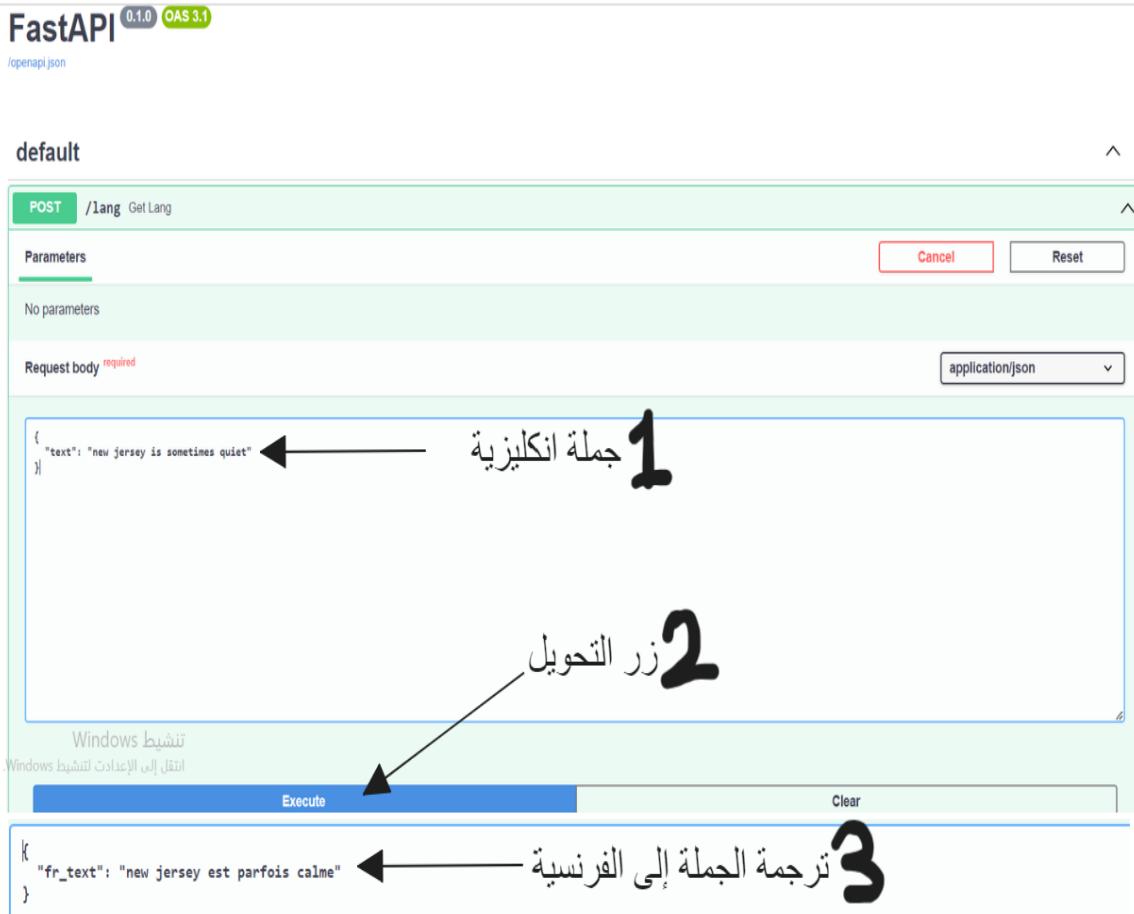
بعد تشغيل المايكرو سيرفر باستخدام الملف الأخير نلاحظ الخرج التالي وهو يعني جاهزية المايكرو سيرفر لاستقبال طلبات الترجمة على الرابط التالي للاستخدام المحلي

```
$ sh start.sh
INFO: Started server process [13620]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```

نستطيع إرسال طلب عبر استخدام command line ولكن احد اسباب اختيار FastAPI هو أنه يقوم بتوفير Swagger UI تريينا من هذه العملية وتتوفر لنا واجهة مستخدم سهلة للتعامل معها ويتم ذلك عن استخدام نفس الرابط مع إضافة كلمة docs بالشكل

<http://127.0.0.1:8000/docs>

فيظهر لنا هذه الواجهة مع كامل عملية التحويل الشكل(3-12)



(الشكل 3-12)

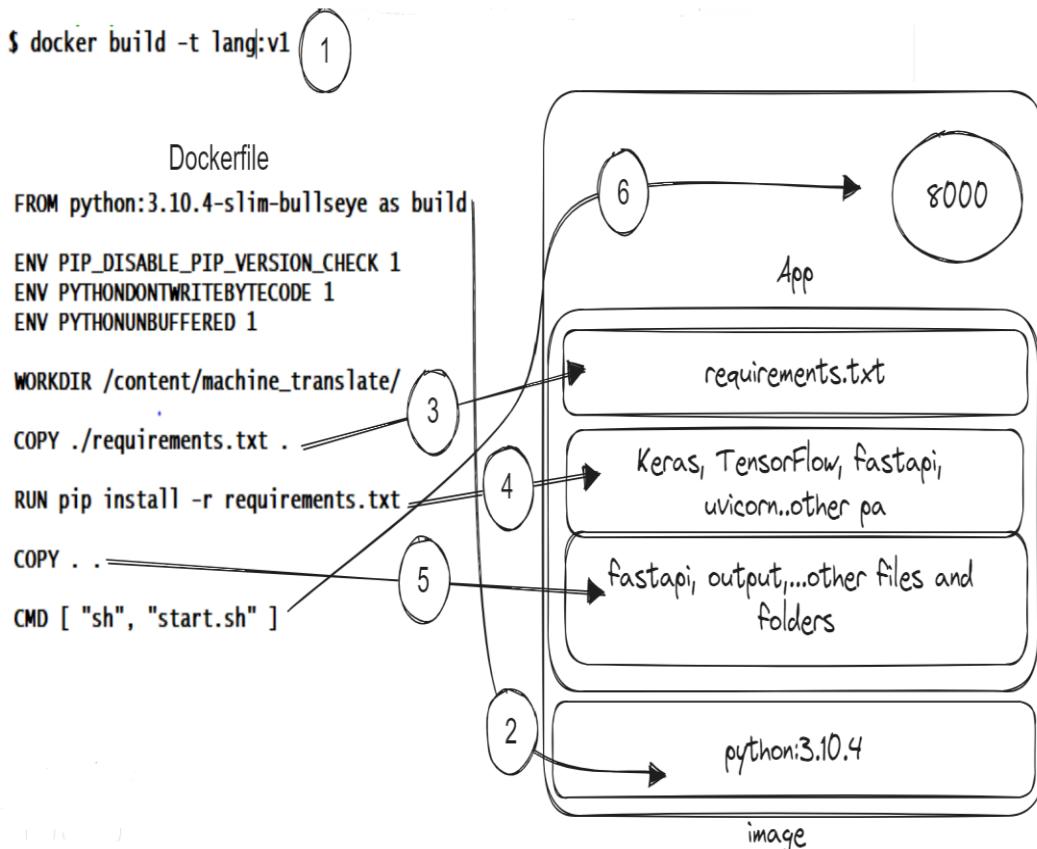
## 2) مرحلة تشغيل API في بيئة معزولة باستخدام Docker :

ما هو Docker؟ Docker هو منصة تقنية تسمح بتنفيذ وتشغيل التطبيقات والخدمات داخل بيئة معزولة عن النظام المضيف. يستخدم Docker تقنية الحاويات (containers) لتقديم بيئة مستقلة وقابلة للتكرار لتشغيل التطبيقات.

ببساطة، يمكنك اعتبار Docker كصندوق يحتوي على جميع المكونات والأدوات اللازمة لتشغيل تطبيق معين. هذا يشمل الأكواد والتبعيات والمكتبات والأدوات وأي شيء آخر يحتاجه التطبيق للعمل بشكل صحيح، عند استخدام Docker يمكنك إنشاء صورة Docker تحتوي على جميع المكونات اللازمة لتشغيل التطبيق. يمكن تشغيل هذه الصورة على أي جهاز يدعم Docker، بغض النظر عن نظام التشغيل الذي يستخدمه الجهاز، والتي توفر بيئة معزولة تماماً عن النظام المضيف.

باستخدام Docker، يمكنك تعبئة التطبيق بسهولة وتشغيله على أي جهاز بدون الحاجة لتنصيب المكونات والتطبيقات يدوياً. كما يمكنك توزيع التطبيق كحزمة مستقلة، مما يسهل على المستخدمين تشغيله واستخدامه بسهولة. وللقيام بهذه المهمة سوف نقوم بكتابة ملف Dockerfile يحوي بعض التعليمات لبناء هذا الصندوق الذي يوضحه الشكل (13-3) بشكل تخيلي وسنوضح في الأسطر

### اللاحقة أرقام المراحل



(الشكل 13-3)

عند بناء هذه الصورة (الصندوق) أحد مفاهيم docker يتم اعتبار كل أمر سطري في ملف Dockerfile أحد مفاهيم docker يتم اعتبار كل أمر سطري في ملف Dockerfile يتم تنفيذه على طبقه ونستطيع كل طبقة أن تستفيد من الطبقة التي تسبقها، دون الدخول في تفاصيل كثيرة لنبدأ في شرح المراحل الممرمة بالترتيب في الشكل (13-3):

1. بعد كتابة Dockerfile على اليسار وتنفيذ الأمر في command line والذي يمثل عملية بناء الصورة يتم بناء الصورة التخيلية على اليمين ونستطيع اعطاء اسم لهذه الصورة مع رقم يمثل

الإصدار عبر المتحول (t) وهذا يساعد في الرجوع إلى اصدارات سابقة، تم تسمية الاصدار

.lang:v1

2. استخدام الطبقة ضمن الصورة (الصندوق) التي تسمح بتشغيل أي تطبيق بايثون مع تحديد رقم اصدار، تم اختيار python:3.10.4-slim-bullseye وهو اصدار خفيف من الاصدار العادي وذلك لتوفير الذاكرة، طبعاً لو أردنا تشغيل تطبيق php كنا استخدمنا صورة خاصة بـ .php.

3. نقل الملف requirements.txt ضمن جهازنا المضيف إلى داخل الصورة (الصندوق) وذكرنا في مرحلة التخطيط أن هذا الملف يحتوي على python packages الخاصة بالمشروع فقط مع رقم الأصدار لكل packages

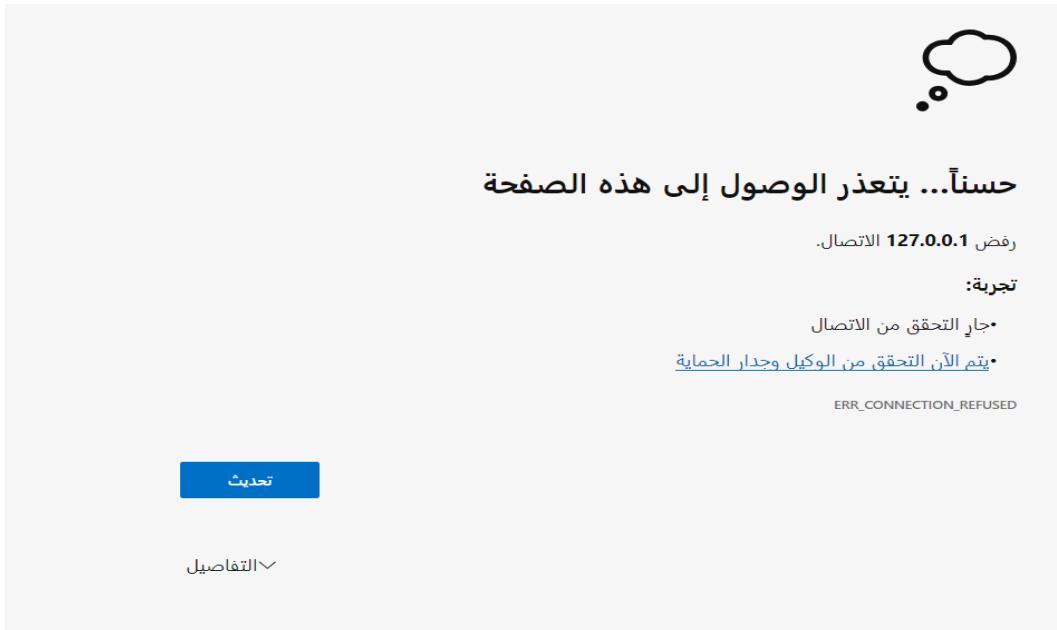
```
$ cat requirements.txt
tensorflow==2.12.0
keras==2.12.0
Keras-Preprocessing==1.1.2
numpy==1.22
six==1.16.0
PyYAML==6.0
pickleshare==0.7.5
neptune==1.3.2
neptune-tensorflow-keras==2.1.1
annotated-types==0.5.0
anyio==3.7.1
click==8.1.5
colorama==0.4.6
fastapi==0.100.0
h11==0.14.0
idna==3.4
pydantic==2.0.3
pydantic_core==2.3.0
sniffio==1.3.0
starlette==0.27.0
typing_extensions==4.7.1
uvicorn==0.23.0
waitress==2.1.2
```

4. تنفيذ أمر تثبيت الحزم بعد نقل الملف في المرحلة 3 عبر الأمر pip install
5. نقل مجلدات وملفات المشروع ضمن جهازنا المضيف إلى داخل الصورة (الصندوق)
6. وضع أمر تشغيل السيرفر الخاص بـ API في لحظة تحويل هذا الصورة (الصندوق) إلى Container عبر port 8000

ما هي Container؟ هي تعتبر المرحلة التشغيلية للمشروع والاستفادة من خدماته أي تحول هذا الصندوق الذي يحتوي جميع الأشياء السابقة إلى صندوق قابل للاستخدام، بكلمات أخرى

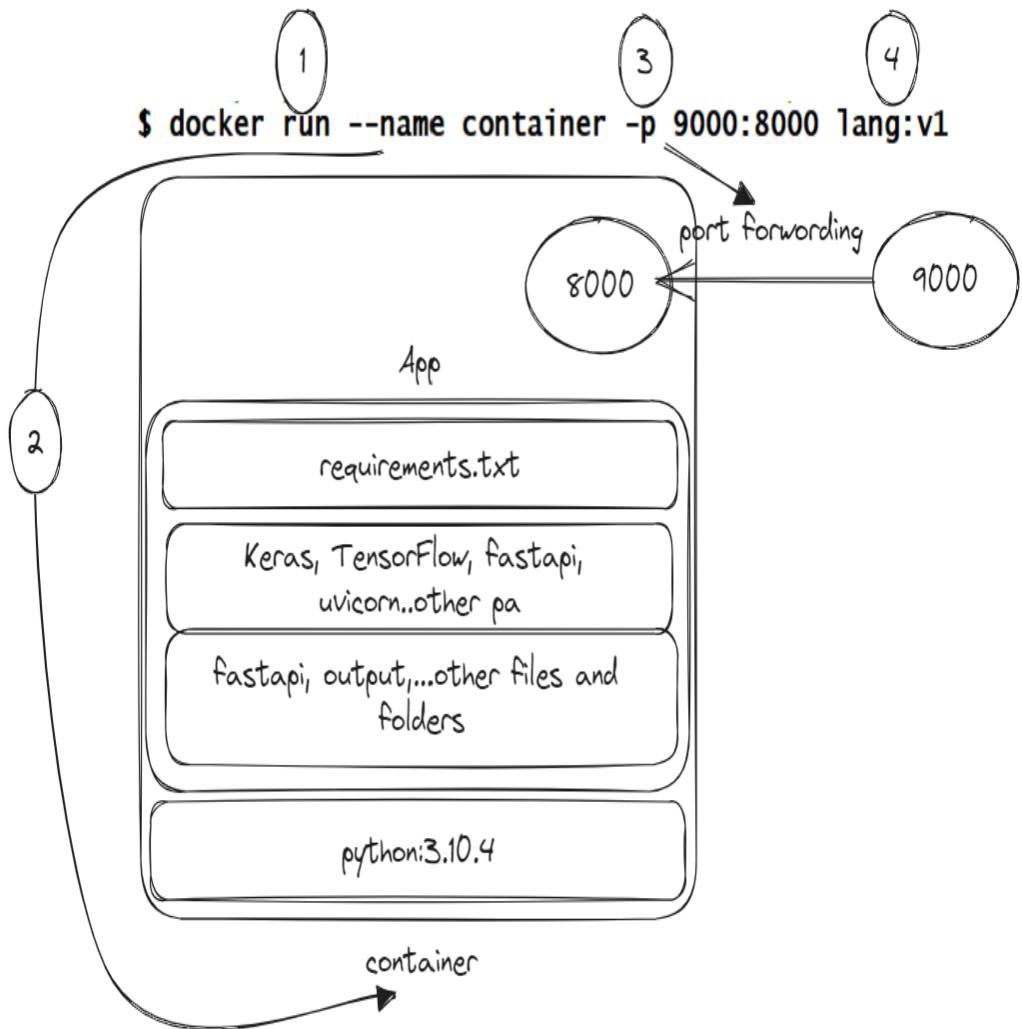
أصبح سيرفر صغير ضمن جهازنا يعمل عليه تطبيق الترجمة ويستطيع هذا Container السيرفر الاستفاده من ram وcpu وتشغيل تطبيق الترجمة بسبب الطبقات الموجودة لديه والتي عرفناها في الصورة (الصندوق) في ملف Dockerfile، الفرق بين container و image مثل الفرق بين الملف المضغوط والملف بعد فك الضغط في الملف المضغوط لا تستطيع التعديل على أي من ملفاته وكذلك الصورة بعد الكتابة عليها وبناءها أصبحت مثل الملف المضغوط لا يمكن التعديل عليها ولا ان نستفيد من خدماتها لذلك نقوم بتحويلها إلى Container وبالتالي تصبح كأنه ملف بعد فك الضغط تستطيع حذف ملفاته والتعديل عليها وكتابتها وكذلك أصبحنا Container أصبعنا باستطاعتنا أن نستفيد من الخدمات التي تقدمها وهي الترجمة في حالتنا ويكون لها ip وport الذي حددناه مسبقا عند بناء الصورة ويساوي 8000، طبعا هنا سنقابلنا مشكلة لو قمنا بتشغيل container ودخلنا إلى الرابط <http://127.0.0.1:8000>

سوف يعطنا هذا الخطأ في الشكل (14-3)



(الشكل 14-3)

لذلك سوف نستخدم مفهوم في الشبكات يدعى port forwarding وهو يقوم على ربط port من النظام الخاص بنا (وهو هنا الحاسب الخاص بنا) مع port لجهاز آخر (وهو في حالتنا container الذي يملك port 8000) وذلك عند تشغيل container الموضح في الشكل (15-3)



(الشكل 15-3)

حيث تعني المراحل الموضحة في الشكل (15-3) :

1. استخدام أمر التشغيل `run` لأخذ نسخة من الصورة وتحويلها إلى `.Container`.
2. اعطاء اسم للحاوية وهنا تم تسميتها `container`.
3. تكنيك `port forwarding` يسمح للبروت 9000 في الجهاز المضيف بالاستماع إلى البروت 8000 داخل `container` بالتالي عند فتح الرابط `http://127.0.0.1:9000` نلاحظ نجاح العملية بالنظر إلى الرسالة التالية.



{ "code": "200", "message": "I am Ready" }

4. استخدام الصورة مع اصدارها والذي قمنا ببنائها سابقا .lang:v1

إلى هنا نكون قد انتهينا من مرحلة التعبئة (packaging) وسوف ننتقل إلى المرحلة الخامسة الخاصة بمرحلة deployment

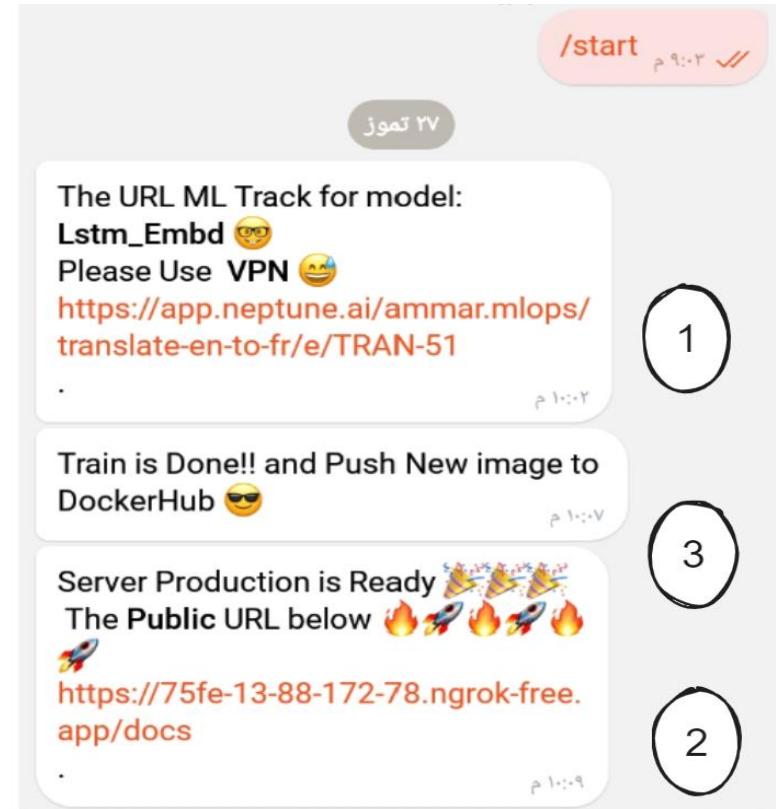
### 3-5 مرحلة المنتج النهائي Deployment

في هذه المرحلة قمنا بأكثر من عملية للوصول إلى مرحلة المنتج النهائي وأتمته جميع العمليات ليقوم فريق Research بالتركيز على تطوير المعماريات دون الدخول في تفاصيل إطلاق المنتج التي تعتبر مهمة فريق MLOps، في هذه المرحلة تم العمل على:

(1) تم استخدام pyngrok وهي مكتبة في بايثون تسمح لنا بتحويل رابط محلي إلى رابط عام يمكن الدخول إليه من أي مكان... أي أصبح الموقع الذي يقدم خدمة الترجمة متاح للجميع، بسبب عدم امتلاكنا لسيرفر خاص، تم استخدام هذا الحل وهو فقط للاستخدام التجريبي وبالتالي تم الاستفاده من هذه المكتبة في الملف server.py ضمن fastapi لتحويل الرابط المحلي http://127.0.0.1:8000 إلى الرابط من الشكل https://f8bb-34-27-208-59.ngrok-free.app طبعا وهو رابط متغير بكل مرة يتم فيها تشغيل السيرفر لذلك في مرحلة المنتج النهائي يتم تشغيل السيرفر دون وجود Downtime ويكون هذا الرابط ثابت بالنسبة لعملية التشغيل.

(2) إنشاء telegram bot باسم Slack وتم ربط هذا البوت مع كود التطوير الخاص بنا لإعلامنا بجميع العمليات التي تتم على السيرفرات الافتراضية الخاصة بنا وتحوي ثلاثة رسائل:  
1. في ملف train.py تم بناء التابع send\_telegram مهمته هذا التابع إرسال الرسائل إلى بوت تلغرام Slack وتم استخدامه ضمن التابع train لإرسال الرابط الخاص بمراقبة عملية تدريب النموذج الذي تحديثنا عنه في مرحلة التدريب train مع رسالة توضح اسم النموذج الذي يتم تدريبيها وكذلك رسالة لاستخدام VPN لكي يعمل الرابط كما توضح الرسالة رقم 1 في الشكل (3-16).

2. في ملف server.py ضمن fastapi بعد تحويل الرابط المحلي إلى رابط عام باستخدام send\_telegram يقوم التابع pyngrok بإرسال رسالة فيها الرابط العام لتجربة المترجم من أي جهاز آخر كما توضح رقم 2 في الشكل(3-16).



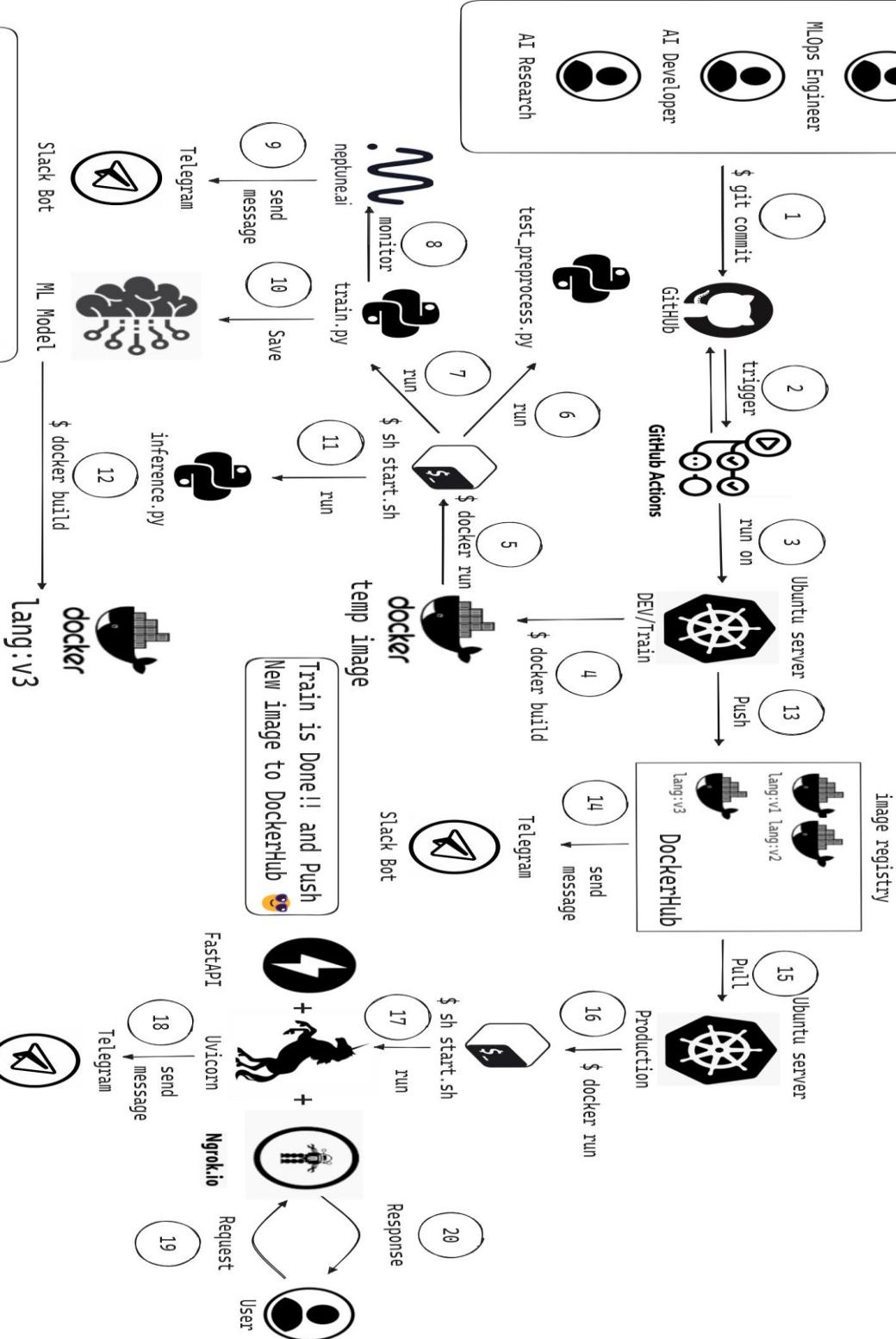
(الشكل 3-16)

3) مرحلة أتمتة العمليات باستخدام github actions من نقطة إضافة كود جديد إلى مرحلة الأنتاج ويكون ذلك في الملف ذو المسار التالي .github\workflows\CI\_CD.yaml وهو ملف يستخدم في كتابة طريقة سير العمليات في المشروع البرمجي بصيغة yaml حيث تكون البرمجة في هذا الملف على شكل declarative على شكل اسم الأمر والأمر لتنفيذ هذه العملية وتم العمل في هذا الملف على الخطوات التالية والتي توضح الملخص النهائي للمشروع في الشكل(3-17) يمكن تتبع العمل من خلال الأرقام التي تم وضعها:

- (a) رقم 1 يمثل مرحلة إضافة كود جديد باستخدام git commit إلى Github ويتم ذلك من قبل فريق المشروع سواء الذي يقوم بإضافة تعديل جديد على المعماريات أو اختبار للكود ما أو تعديل على طريقة سير المشروع.
- (b) رقم 2 يمثل خاصية trigger وهي خاصية تستخدم لتعيين الأحداث التي يجب أن يتم تشغيل سير العمل عندها، أي عند إضافة الكود إلى فرع محدد في الملف السابق مثل dev أو main فإن الأوامر المكتوبة سوف تتفذ تلقائيا دون تدخل من أي شخص.
- (c) رقم 3 يمثل بداية عملية التنفيذ سحب كامل كود المشروع إلى سيرفر مستقل .DEV/Train
- (d) رقم 4 بناء صورة مؤقتة temp image (الصندوق الذي تحدثنا عنه في مرحلة التعبئة package تحوي على جميع الاعتمادات والأدوات التي يحتاجها المشروع) على هذا السيرفر باستخدام docker build
- (e) رقم 5 إنشاء حاوية container باستخدام docker run وبداية عملية التدريب وفيها يبدأ بالعمل ويوجد فيه ثلاثة سكريبتات بايثون كما ذكرنا في مرحلة التخطيط start.sh planning تبدأ التنفيذ بشكل تسلسلي.
- (f) رقم 6 تنفيذ أول سكريبت test\_preprocess.py وهو يقوم بختبار التوابع المستخدمة في معالجة البيانات.
- (g) رقم 7 تنفيذ سكريبت التدريب train.py باستخدام الأعدادات التي تم وضعها في عقل المشروع .train.yaml
- (h) رقم 8 عندما يكون سكريبت train.py على التوازي تبدأ عملية monitoring باستخدام Neptune.ai حيث يتم تشكيل رابط ليسح فقط لفريق المشروع بالدخول لهذا الرابط
- (i) رقم 9 على التوازي أيضا ولتأمين سهولة وصول الفريق إلى جميع نتائج المشروع من مكان واحد يتم أرسال رابط عملية monitoring مع رسالة توصية باستخدام VPN وكذلك نوع النموذج الذي يتم تدريبه حاليا إلى بوت تلغرام Slack والذي يوضحه صندوق الرسائل على سوية سهم رقم 9.
- (j) رقم 10 نهاية عملية التدريب وتخزين ml model بصيغة .h5
- (k) رقم 11 تنفيذ السكريبت الثالث الذي يمثل اختبار بسيط للنموذج الناتج.

- (ا) رقم 12 عملية بناء جديدة للصورة بالاسم lang:v3 وهو الأصدار الثالث وذلك لحفظ ml الذي تم الوصول إليه في رقم 10.
- (م) رقم 13 دفع الصورة الجديدة إلى المستودع الذي تقوم فيه بالاحتفاظ بالصور الجديدة dockerhub وهو .
- (ن) رقم 14 بعد أن تتم عملية الدفع بنجاح يتم أرسال رسالة إلى بوت تلغرام Slack والذي يوضحها صندوق الرسائل على سوية سهم 14 وهذه الرسالة تعني أن "عملية التدريب قد تمت وتم دفع الصورة بنجاح إلى المستودع" وهذا يعني نهاية مرحلة التدريب والتطوير.
- (و) رقم 15 بداية مرحلة production وتبدأ بسحب الصورة الأخيرة في المستودع إلى سيرفر مستقل خاص بمرحلة الانتاج لإطلاق web server .
- (پ) رقم 16 إنشاء حاوية container باستخدام docker run start.sh ويدأعدها بالعمل والذي يقوم بتشغيل سيرفر الخاص بالاتصال مع .ml model
- (ز) رقم 17 بداية عملية التشغيل التي تتطلب من ربط fastapi الذي يمثل عملية الاتصال مع ml model وبعد ذلك تشغيل unicorn ليقوم بمهمة موازنة الحمل عند زيادة الضغط على الموقع ذلك تحويل الرابط المحلي إلى رابط عام باستخدام pyngrok .
- (ر) رقم 18 بعد تكوين الرابط العام يتم أعلام الفريق عبر أرسال رسالة برابط مرحلة الانتاج إلى بوت تلغرام Slack والذي يوضحه صندوق الرسائل على سوية سهم رقم 18 .
- (س) رقم 19 تمثل طلب المستخدم لترجمة جملة انكليزية إلى جملة فرنسية.
- (ت) رقم 20 يمثل استجابة السيرفر بإرسال ترجمة الجملة إلى المستخدم .
- رقم 21 ليس موجود في الصورة ولكن يعتبر تنويه بسيط هو أن الرابط العام في مرحلة الانتاج لا يكون متغير في عالم الشركات، حيث تقوم الشركات بشراء استضافات ويكون للموقع domain ثابت ، لذلك تعتبر المرحلة الموجودة في مشروعنا Virtual Production .
- (ع) رقم 22 تنويه ثاني برجاء قلب الصفحة عند الإطلاع على الشكل(17-3).

# AI Team



(الشكل 17-3)

إلى هنا نكون قد انتهينا من مرحلة المنتج النهائي (deployment) وسوف ننتقل إلى المرحلة السادسة الأخيرة الخاصة بمرحلة Monitior.

### 6-3 مرحلة المراقبة Monitior

في هذه المرحلة تم العمل على:

- (1) تم إضافة Neptune.ai لمراقبة سير عملية التدريب وتحذثنا عنها سابقاً باستفاضة كبيرة في مرحلة التدريب.
  - (2) يوجد مرحلة لمراقبة الأداء في ml model الناتج في مرحلة التدريب وذلك لمراقبة سرعة استجابة ml model في إعطاء ترجمة وكذلك مقدار الاستهلاك للموارد من RAM,CPU,GPU
- لن نتطرق لهذه المرحلة في المشروع وسنتركها للقارئ الذي سوف يقوم بتطوير المشروع بعدها ولكن سنعطيك كلمة مفاتيحية تقوم بحل جزء من المشكلة في سرعة الاستجابة إذا وجدت وهي onnx .format

يوجد أيضاً مرحلة لمراقبة الأداء في web service الذي قمنا بإطلاقه في مرحلة الانتاج وذلك لقياس مراقبة سير العمل من لحظة الطلب request إلى لحظة الاستجابة response وزيادة عدد المستخدمين لخدمة الترجمة على التوازي لن نتطرق لهذه المرحلة في المشروع أيضاً وسنتركها للقارئ الذي سوف يقوم بتطوير المشروع بعدها على مستوى web service ولكن سنعطيك كلمة مفاتيحية تقوم بحل جزء من المشكلة مثل prometheus .

وبذلك نكون قد وصلنا إلى المرحلة السادسة والأخيرة.

## الوصيات

---

بالنسبة لمراحل العمليات من التخطيط إلى الانتاج تم التغاضي عن مرحلة مهمة في mlops cycle وهي مرحلة Test مثل اختبار المعماريات قبل عملية التدريب و اختبارات على النماذج الناتجة وكذلك اختبارات على web service قبل الأطلاق إلى مرحلة production.

أما بالنسبة لتسريع سير العمليات وتقليل الوقت في انتاج نماذج دقة في عملية الترجمة بشكل اسرع هو امتلاك موارد افضل مثل توفر 2 او 4 GPU على الاقل وذلك يساعد في عملية اختبار 4 المعماريات في وقت واحد واختيار المعمارية الافضل لنقوم بضبط hyperparameters على عكس google colab الذي يمكننا من تشغيل GPU في وقت واحد ولا يعتبر سيرفر مستقل للدخول والخروج.

أما بالنسبة للمعماريات المستخدمة فلا تعتبر هي الافضل يوجد معماريات افضل في عملية الترجمة مثل نماذج الانتبه والمحوّلات.

## المراجع Reference

---

1. <https://arxiv.org/pdf/1301.3781.pdf> Tomas Mikolov, Kai Chen , Greg Corrado Jeffrey Dean, (September 2013) Efficient Estimation of Word Representations in Vector Space.
2. <https://arxiv.org/pdf/1409.3215.pdf> Ilya Sutskever , Oriol Vinyals ,Quoc V. Le, (December 2014) Sequence to Sequence Learning with Neural Networks.
3. <https://arxiv.org/pdf/1409.0473.pdf> Dzmitry Bahdanau, KyungHyun Cho Yoshua Bengio\*, (may 2016) Neural Machine Translation by Jointly Learning to Align and Translate.