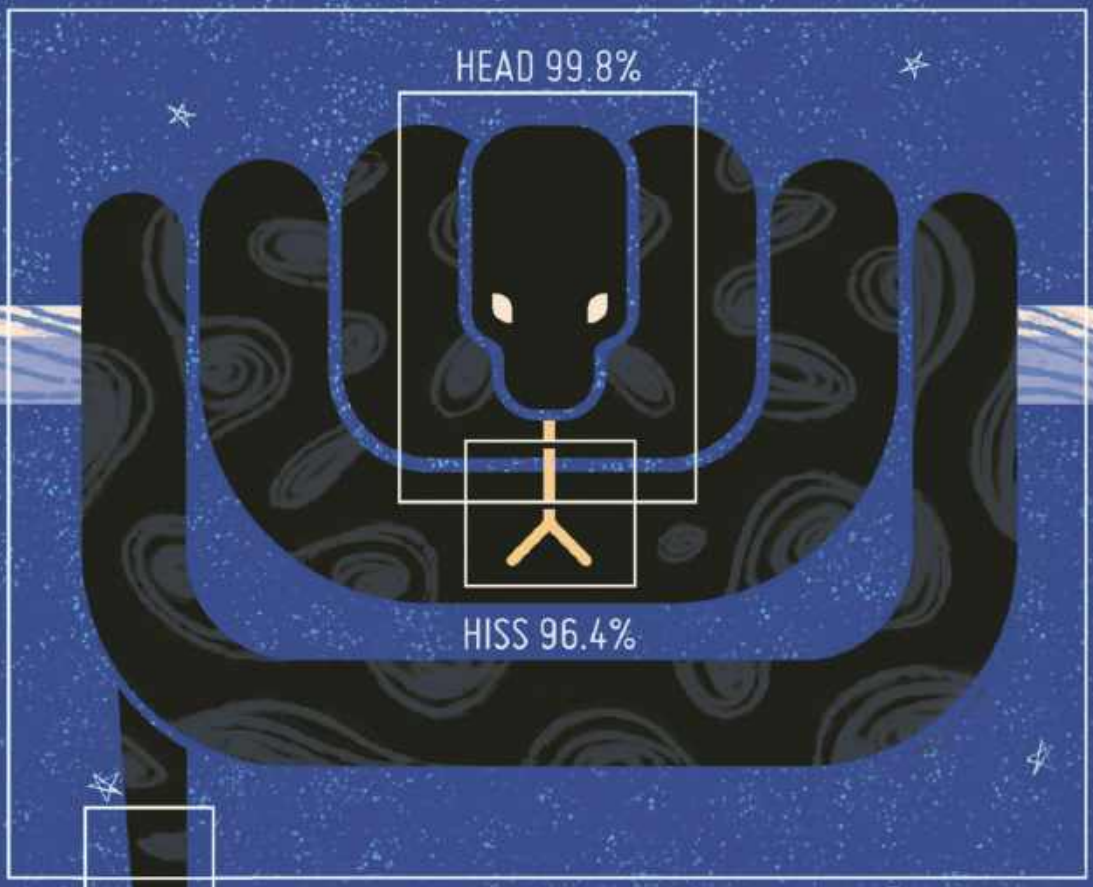
بايثون

100%



مشاريع تعلم الآلة

تأليف
ليزا تاغليفيري وآخرون

ترجمة
د. علاء طعيمة

بسمه تعالى

مشاريع تعلم الآلة: بايثون

تأليف:

ليزا تاغليفيري وآخرون

ترجمة:

د. علاء طعيمة

مقدمة المترجم

يحاول هذا الكتاب تزويد مطوري اليوم وغداً بأدوات يمكنهم استخدامها لفهم التعلم الآلي وتقييمه وتشكيله بشكل أفضل.

سيقوم بإعداد بيئة برمجة بايثون إذا لم يكن لديك واحدة بالفعل، ثم يزودك بفهم تصوري للتعلم الآليفي الفصل "مقدمة إلى التعلم الآلي". فيما يلي ثلاثة مشاريع لتعلم آلة بايثون. سوف يساعدونكفي إنشاء مصنف للتعلم الآلي، وبناء شبكة عصبية للتعرف على الأرقام المكتوبة بخط اليد، وإعطائك خلفيةفي التعلم المعزز العميق من خلال بناء بوت لـ Atari.

إذا كنت تعرف بعض لغة بايثون وتريد استخدام التعلم الآلي والتعلم العميق، فاختر هذا الكتاب. سواء كنت تريد البدء من نقطة الصفر أو توسيع معرفتك بالتعلم الآلي، فهذا مورد أساسي.

كُتب هذا الكتاب للمطورين وعلماء البيانات الذين يرغبونفي إنشاء تعلم آلي عملي وكود التعلم العميق، وهو مثالي لأي شخص يرغبفي تعليم أجهزة الكمبيوتر كيفية التعلم من البيانات.

د. علاء طعيمة

كلية علوم الحاسوب وتكنولوجيا المعلومات

جامعة القادسية

العراق

المحتويات

1

مقدمة

مقدمة

نظرًا لزيادة الاستفادة من التعلم الآليفي العثور على الأنماط وإجراء التحليل واتخاذ القرارات دون مدخلات نهائية من البشر، فمن الأهمية بمكان توفير الموارد ليس فقط لتطوير الخوارزميات والمنهجيات، ولكن أيضًا الاستثمارفي جذب المزيد من أصحاب المصلحة إلى هذا المجال. يحاول كتاب مشاريع بايثونفي التعلم الآلي أن يفعل ذلك تمامًا: لتزويد مطوري اليوم والغد بأدوات يمكنهم استخدامها لفهم التعلم الآلي وتقييمه وتشكيله بشكل أفضل للمساعدةفي ضمان أنه يخدمنا جميعًا.

سيُعد لك هذا الكتاب بيئة برمجة بايثون إذا لم يكن لديك واحدة بالفعل، ثم يزودك بفهم تصوري للتعلم الآليفي الفصل "مقدمة إلى التعلم الآلي". فيما يلي ثلاثة مشاريع لتعلم آلة بايثون. سوف يساعدونكفي إنشاء فصل دراسي للتعلم الآلي، وبناء شبكة عصبية للتعرف على الأرقام المكتوبة بخط اليد، وإعطائك خلفيةفي التعلم المعزز العميق من خلال بناء بوت لـ Atari.

ظهرت هذه الفصولفي الأصل كمقالات عن مجتمع DigitalOcean، كتبها أعضاء من مجتمع مطوري البرامج الدولي. إذا كنت مهتمًا بالمساهمةفي قاعدة المعرفة هذه، ففكرفي اقتراح برنامج تعليمي لبرنامج Write for DOnations على do.co/w4do. تقدم DigitalOcean الدفع للمؤلفين وتوفر تبرعًا مطابقًا لغير العاملينفي المجال التقني.

كتب أخرى في هذه السلسلة

إذا كنت تتعلم لغة بايثون أو تبحث عن مواد مرجعية، فيمكنك تنزيل كتاب بايثون الإلكتروني المجاني، How To Code in Python 3 والذي يتوفر عبر do.co/python-book.

بالنسبة إلى لغات البرمجة الأخرى ومقالات هندسة DevOps، تتوفر قاعدة معارفنا التي تضم أكثر من 2100 برنامج تعليمي كمورد مرخص من Creative-Commons عبر do.co/tutorials.

2

إعداد بيئة برمجة بايثون

إعداد بيئة برمجة بايثون

بايثون هي لغة برمجة مرنة ومتعددة الاستخدامات ومناسبة للعديد من حالات الاستخدام، مع نقاط قوةفي البرمجة النصية والأتمتة وتحليل البيانات والتعلم الآلي والتطوير الخلفي. نُشر لأول مرةفي عام 1991، استلهم فريق تطوير بايثون من مجموعة الكوميديا البريطانية Monty Python لإنشاء لغة برمجة ممتعةفي الاستخدام. بايثون 3 هي أحدث نسخة من اللغة وتعتبر مستقبل بايثون.

سيساعدك هذا البرنامج التعليميفي إعداد الخادم البعيد أو الكمبيوتر المحلي مع بيئة برمجة بايثون 3. إذا كان لديك بايثون 3 مثبتًا بالفعل، جنبًا إلى جنب مع pip وvenv، فلا تترددفي الانتقال إلى الفصل التالي!

المتطلبات الأساسية

سيعتمد هذا البرنامج التعليمي على العمل مع نظام Linux أو نظام يشبه Unix (* nix) واستخدام سطر أوامر أو بيئة طرفية. يجب أن يكون كل من macOS وبرنامج PowerShell الخاص بنظام Windows قادرين على تحقيق نتائج مماثلة.

الخطوة1 - تثبيت بايثون 3

تأتي العديد من أنظمة التشغيل مثبت عليها بايثون 3 بالفعل. يمكنك التحقق لمعرفة ما إذا كان لديك بايثون 3 مثبتًا عن طريق فتح نافذة طرفية وكتابة ما يلي:

```
python3 -V
```

ستتلقى مخرجاتفي نافذة التيرمينال ستعلمك برقم الإصدار. بينما قد يختلف هذا الرقم، سيكون الإخراج مشابهًا لما يلي:

```
Output
Python 3.7.2
```

إذا تلقيت مخرجات بديلة، فيمكنك التنقلفي متصفح الويب إلى python.org لتنزيل بايثون 3 وتثبيته على جهازك باتباع الإرشادات.

بمجرد أن تتمكن من كتابة الأمر `python3 -V` أعلاه وتلقي الإخراج الذي يوضح رقم إصدار بايثون لجهاز الكمبيوتر الخاص بك، فأنت جاهز للمتابعة.

الخطوة 2 – تثبيت pip

لإدارة حزم البرامج الخاصة ببايثون، فلنقم بتثبيت pip، وهي أداة ستقوم بتثبيت وإدارة حزم البرمجة التي قد نرغبفي استخدامهافي مشاريع التطوير الخاصة بنا.

إذا قمت بتنزيل بايثون من python.org، فيجب أن يكون لديك pip مثبتًا بالفعل. إذا كنت تستخدم خادم Ubuntu أو Debian أو كمبيوتر، فيمكنك تنزيل pip عن طريق كتابة ما يلي:

```
pip3 install package_name
```

هنا، يمكن أن يشير اسم package_name إلى أي حزمة أو مكتبة بايثون، مثل Django لتطوير الويب أو NumPy للحوسبة العلمية. لذلك إذا كنت ترغب في تثبيت NumPy، فيمكنك القيام بذلك باستخدام الأمر pip3

install numpy.

هناك عدد قليل من الحزم وأدوات التطوير التي يجب تثبيتها للتأكد من أن لدينا إعدادًا قويًا لبيئة البرمجة لدينا:

```
sudo apt install build-essential libssl-dev libffi-
dev python3-dev
```

بمجرد إعداد بايثون وpip والأدوات الأخرى، يمكننا إعداد بيئة افتراضية لمشاريعنا التطويرية.

الخطوة 3 - إنشاء بيئة افتراضية

تمكّنك البيئات الافتراضية من الحصول على مساحة معزولة على خادمك لمشاريع بايثون، مما يضمن أن كل مشروع من مشاريعك يمكن أن يكون له مجموعة التبعيات الخاصة به والتي لن تعطل أيًا من مشاريعك الأخرى.

يوفر لنا إعداد بيئة البرمجة تحكمًا أكبر في مشاريع بايثون الخاصة بنا وكيفية التعامل مع الإصدارات المختلفة من الحزم. هذا مهم بشكل خاص عند العمل مع حزم الطرف الثالث.

يمكنك إعداد أي عدد تريده من بيئات برمجة بايثون. كل بيئة هي في الأساس دليل أو مجلد على الخادم الخاص بك يحتوي على بعض السكريبتات فيه لجعله يعمل كبيئة.

بينما توجد عدة طرق لتحقيق بيئة برمجة في بايثون، سنستخدم وحدة venv هنا، والتي تعد جزءًا من مكتبة بايثون 3 القياسية.

إذا قمت بتثبيت بايثون من خلال برنامج التثبيت المتاح من python.org، فيجب أن يكون لديك venv جاهزًا للعمل.

لتثبيت venv في خادم أو جهاز Ubuntu أو Debian، يمكنك تثبيته باستخدام ما يلي:

```
sudo apt install -y python3-venv
```

مع تثبيت venv، يمكننا الآن إنشاء بيئات. دعنا إما نختار الدليل الذي نرغبفي وضع بيئات برمجة بايثون فيه، أو ننشئ دليلًا جديدًا باستخدام mkdir، كمافي:

```
mkdir environments
cd environments
```

بمجرد أن تكونفي الدليل حيث تريد أن تعيش البيئات، يمكنك إنشاء بيئة. يجب عليك استخدام إصدار بايثون المثبت على جهازك باعتباره الجزء الأول من الأمر (الإخراج الذي تلقيته عند كتابة python -V) إذا كان هذا الإصدار هو Python 3.6.3 ، فيمكنك كتابة ما يلي:

```
python3.6 -m venv my_env
```

بدلاً من ذلك، إذا تم تثبيت Python 3.7.3 على جهاز الكمبيوتر، فاستخدم الأمر التالي:

```
python3.7 -m venv my_env
```

قد تسمح لك أجهزة Windows بإزالة رقم الإصدار بالكامل:

```
python -m venv my_env
```

بمجرد تشغيل الأمر المناسب، يمكنك التحقق من استمرار إعداد البيئة.

بشكل أساسي، يُنشئ pyvenv دليلًا جديدًا يحتوي على بعض العناصر التي يمكننا عرضها باستخدام الأمر ls:

```
ls my_env
```

Output

```
bin include lib lib64 pyvenv.cfg share
```

تعمل هذه الميزات معًا للتأكد من عزل مشاريعك عن السياق الأوسع لآلتك المحلية، بحيث لا تختلط بين أنظمة النظام والمشروع. هذه ممارسة جيدة للتحكمفي الإصدار وللتأكد من أن كل مشروع من مشاريعك لديه حق الوصول إلى الحزم المعينة التي يحتاجها. سيكون Python Wheels، وهو تنسيق حزمة مدمج لبايثون يمكنه تسريع إنتاج البرامج الخاصة بك عن طريق تقليل عدد المرات التي تحتاج فيها إلى الترجمة،في دليل share .Ubuntu 18.04

لاستخدام هذه البيئة، تحتاج إلى تنشيطها، وهو ما يمكنك تحقيقه عن طريق كتابة الأمر التالي الذي يستدعي سكربت التنشيط:

```
source my_env/bin/activate
```

سيتم الآن تحديد موجه الأوامر الخاص بك مسبقًا باسم بيئتك، وفي هذه الحالة يطلق عليه my_env. اعتمادًا على الإصدار Debian Linux الذي تقوم بتشغيله، قد يظهر مسبقك بشكل مختلف إلى حد ما، ولكن يجب أن يكون اسم بيئتك بين قوسين هو الشيء الأول الذي تراه على خطك:

```
((my_env) sammy@sammy:~/environments$
```

يتيح لنا هذا الإصدار المسبق معرفة أن البيئة my_env نشطة حاليًا، مما يعني أنه عند إنشاء برامج هنا، فإنها ستستخدم فقط إعدادات وحزم هذه البيئة المعينة.

ملاحظة:في البيئة الافتراضية، يمكنك استخدام الأمر `python` بدلاً من `python3`، و `pip` بدلاً من `pip3` إذا كنت تفضل ذلك. إذا كنت تستخدم بايثون 3 على جهازك خارج بيئة ما، فستحتاج إلى استخدام الأمرين `python3` و `pip3`حصريًا.

بعد اتباع هذه الخطوات، تصبح بيئتك الافتراضية جاهزة للاستخدام.

الخطوة 4 - بناء برنامج "Hello، World"

الآن بعد أن تم إعداد بيئتنا الافتراضية، فلنقم بإنشاء برنامج "Hello, World!". سيسمح لنا هذا باختبار بيئتنا ويوفر لنا الفرصة للتعرف أكثر على بايثون إذا لم نكن كذلك بالفعل.

للقيام بذلك، سنفتح محرر نص سطر أوامر مثل nano وننشئ ملفًا جديدًا:

```
(my_env) sammy@sammy:~/environments$ nano hello.py
```

بمجرد أن يفتح النصفي نافذة التيرمينال، سنقوم بكتابة برنامجنا:

```
print("Hello, World!")
```

الخروج من Nano عن طريق كتابة مفاتيح CTRL وX، وللحفظ اضغط Y.

بمجرد الخروج من nano والعودة إلى shell الخاص بك، فلنقم بتشغيل البرنامج:

```
(my_env) sammy@sammy:~/environments$ python hello.py
```

يجب أن يتسبب برنامج `hello.py` الذي أنشأته للتو في قيام التيرمينال الخاص بك بإنتاج المخرجات التالية:

```
Output
Hello, World!
```

لمغادرة البيئة، ما عليك سوى كتابة الأمر `deactivate` وستعود إلى دليلك الأصلي.

الاستنتاج

في هذه المرحلة، لديك بيئة برمجة بايثون 3 تم إعدادها على جهازك ويمكنك الآن بدء مشروع البرمجة!

إذا كنت ترغب في معرفة المزيد عن بايثون، يمكنك تنزيل كتاب How To Code في Python 3 الإلكتروني المجاني عبر do.co/python-book.

3

مقدمة لتعلم الآلة

مقدمة لتعلم الآلة

التعلم الآلي هو مجال فرعي من الذكاء الاصطناعي (AI). الهدف من التعلم الآلي عمومًا هو فهم بُنية البيانات وملائمة هذه البياناتفي النماذج التي يمكن فهمها واستخدامها من قبل الأشخاص.

على الرغم من أن التعلم الآلي هو أحد مجالات علوم الكمبيوتر، إلا أنه يختلف عن الأساليب الحسابية التقليدية.في الحوسبة التقليدية، الخوارزميات عبارة عن مجموعات من التعليمات المبرمجة بشكل صريح تستخدمها أجهزة الكمبيوتر للحساب أو حل المشكلات. وبدلاً من ذلك، تسمح خوارزميات التعلم الآلي لأجهزة الكمبيوتر بالتدرب على مدخلات البيانات واستخدام التحليل الإحصائي من أجل إخراج قيم تقع ضمن نطاق محدد. لهذا السبب، يسهل التعلم الآلي أجهزة الكمبيوترفي بناء النماذج من بيانات العينة من أجل أتمتة عمليات صنع القرار بناءً على مدخلات البيانات.

استفاد أي مستخدم للتكنولوجيا اليوم من التعلم الآلي. تسمح تقنية التعرف على الوجوه لمنصات التواصل الاجتماعي بمساعدة المستخدمين على مشاركة صور الأصدقاء. تعمل تقنية التعرف البصري على الأحرف (OCR) على تحويل صور النص إلى نوع متحرك. تقترح محركات التوصية، المدعومة بالتعلم الآلي، الأفلام أو البرامج التلفزيونية لمشاهدتها بعد ذلك بناءً على تفضيلات المستخدم. قد تكون السيارات ذاتية القيادة التي تعتمد على التعلم الآلي للتنقل متاحة قريبًا للمستهلكين.

التعلم الآلي هو مجال يتطور باستمرار. لهذا السبب، هناك بعض الاعتبارات التي يجب وضعها في الاعتبار أثناء العمل باستخدام منهجيات التعلم الآلي، أو تحليل تأثير عمليات التعلم الآلي.

في هذا البرنامج التعليمي، سننظرفي أساليب التعلم الآلي الشائعة للتعلم الخاضع للإشراف والتعلم غير الخاضع للإشراف، والأساليب الخوارزمية الشائعةفي التعلم الآلي، بمافي ذلك خوارزمية k-أقرب الجيران، وتعلم شجرة القرار، والتعلم العميق. سنستكشف لغات البرمجة الأكثر استخدامًا في التعلم الآلي، ونزودك ببعض السمات الإيجابية والسلبية لكل منها. بالإضافة إلى ذلك، سنناقش التحيزات التي تكرسها خوارزميات التعلم الآلي، ونأخذفي الاعتبار ما يمكن وضعهفي الاعتبار لمنع هذه التحيزات عند بناء الخوارزميات.

طرق التعلم الآلي

في التعلم الآلي، تُصنف المهام عمومًا إلى فئات واسعة. تستند هذه الفئات إلى كيفية تلقي التعلم أو كيفية تقديم الملاحظات على التعلم إلى النظام الذي تم تطويره.

هناك طريقتان من أكثر طرق التعلم الآلي التي يتم تبنيها على نطاق واسع وهما التعلم الخاضع للإشراف والذي يقوم بتدريب الخوارزميات بناءً على بيانات المدخلات والمخرجات التي يصنفها البشر، والتعلم غير الخاضع للإشراف الذي يوفر الخوارزمية بدون بيانات معنونة من أجل السماح لها بإيجاد بُنية ضمن مدخلاتها بيانات. دعونا نستكشف هذه الأساليب بمزيد من التفصيل.

التعلم الخاضع للإشراف

في التعلم الخاضع للإشراف، يتم تزويد الكمبيوتر بأمثلة على المدخلات التي تم تصنيفها بالمخرجات المرغوبة. الغرض من هذه الطريقة هو أن تكون الخوارزمية قادرة على "التعلم" من خلال مقارنة ناتجها الفعلي مع المخرجات "المُدرَّسة" بالأخطاء الثانية، وتعديل النموذج وفقًا لذلك. لذلك يستخدم التعلم الخاضع للإشراف أنماطًا للتنبؤ بقيم التسمية على البيانات الإضافية غير المسماة.

على سبيل المثال، من خلال التعلم الخاضع للإشراف، قد يتم تغذية خوارزمية بالبيانات بصور لأسماك القرش المصنفة على أنها أسماك وصور للمحيطات التي تم تصنيفها على أنها مياه. من خلال التدريب على هذه البيانات، يجب أن تكون خوارزمية التعلم الخاضعة للإشراف قادرة لاحقًا على تحديد صور أسماك القرش غير المسماة على أنها أسماك وصور المحيطات غير المسماة على أنها مياه.

من حالات الاستخدام الشائع للتعلم الخاضع للإشراف استخدام البيانات التاريخية للتنبؤ بالأحداث المستقبلية المحتملة إحصائيًا. قد تستخدم معلومات سوق الأوراق المالية التاريخية لتوقع التغيرات القادمة، أو يتم توظيفها لترشيح رسائل البريد الإلكتروني العشوائية.في التعلم الخاضع للإشراف، يمكن استخدام صور الكلاب المعلمة كبيانات إدخال لتصنيف الصور غير المعلمة للكلاب.

التعلم غير الخاضع للاشراف

في التعلم غير الخاضع للإشراف، لا يتم تصنيف البيانات، لذلك تُترك خوارزمية التعلم لتجد القواسم المشتركة بين بيانات الإدخال الخاصة بها. نظرًا لأن البيانات غير المصنفة أكثر وفرة من البيانات المصنفة، فإن طرق التعلم الآلي التي تسهل التعلم غير الخاضع للإشراف تعتبر ذات قيمة خاصة.

قد يكون هدف التعلم غير الخاضع للإشراف مباشرًا مثل اكتشاف الأنماط المخفية داخل مجموعة البيانات، ولكن قد يكون أيضًا هدفًا لتعلم الميزات، والذي يسمح للآلة الحسابية باكتشاف التمثيلات المطلوبة تلقائيًا لتصنيف البيانات الأولية.

يشيع استخدام التعلم غير الخاضع للإشراف لبيانات المعاملات. قد يكون لديك مجموعة بيانات كبيرة من العملاء ومشترياتهم، ولكن كإنسان، من المحتمل ألا تكون قادرًا على فهم السمات المماثلة التي يمكن استخلاصها من عروض العملاء وأنواع مشترياتهم. من خلال إدخال هذه البياناتفي خوارزمية تعلم غير خاضعة للإشراف، يمكن تحديد أن النساءفي فئة عمرية معينة اللائي يشترون الصابون غير المعطر من المرجح أن يكونوا حاملاً، وبالتالي يمكن استهداف حملة تسويقية متعلقة بالحمل ومنتجات الأطفال لهذا الجمهور بالترتيب لزيادة عدد مشترياتهم.

دون أن يتم إخبارنا بإجابة "صحيحة"، يمكن لأساليب التعلم غير الخاضعة للاشراف أن تنظرفي البيانات المعقدة الأكثر توسعية والتي تبدو غير ذات صلة من أجل تنظيمها بطرق يحتمل أن تكون ذات مغزى. غالبًا ما يتم استخدام التعلم غير الخاضع للإشراف لاكتشاف الحالات الشاذة بمافي ذلك عمليات الشراء الاحتيالية باستخدام بطاقات الائتمان وأنظمة التوصية التي توصي بالمنتجات التي يجب شراؤها بعد ذلك.في التعلم غير الخاضع للإشراف، يمكن استخدام الصور غير المعلمة للكلاب كبيانات إدخال للخوارزمية للعثور على تشابه وتصنيف صور الكلاب معًا.

الاساليب

كحقل، يرتبط التعلم الآلي ارتباطًا وثيقًا بالإحصاءات الحسابية، لذا فإن امتلاك معرفة أساسيةفي الإحصاء مفيد لفهم خوارزميات التعلم الآلي والاستفادة منها.

بالنسبة لأولئك الذين لم يدرسوا الإحصاء، قد يكون من المفيد تحديد الارتباط correlation والانحدار regression لأول مرة، حيث إنهما تقنيات شائعة الاستخدام لفحص العلاقة بين المتغيرات الكمية. الارتباط هو مقياس للارتباط بين متغيرين لم يتم تحديدهما على أنهما تابعان أو مستقلان. يُستخدم الانحدار عند المستوى الأساسي لفحص العلاقة بين متغير تابع واحد ومتغير مستقل واحد. نظرًا لأنه يمكن استخدام إحصائيات الانحدار لتوقع المتغير التابع عندما يكون المتغير المستقل معروفًا، فإن الانحدار يتيح إمكانات التنبؤ.

يتم تطوير مناهج التعلم الآلي باستمرار. لأغراضنا، سنستعرض بعض الأساليب الشائعة التي يتم استخدامهافي التعلم الآلي وقت كتابة هذا الكتاب.

k - أقرب جار

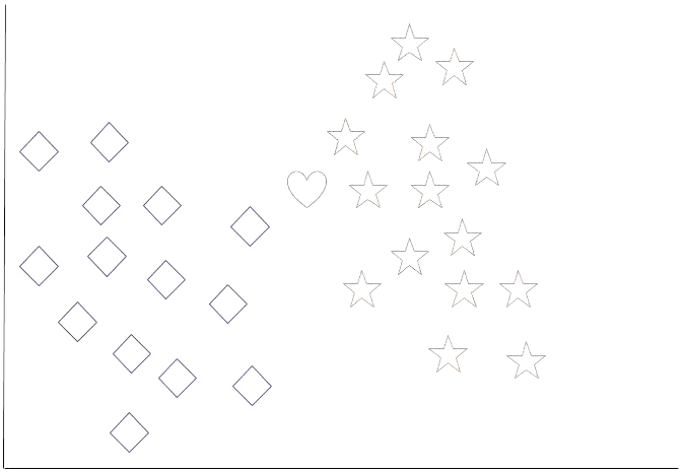
تعد خوارزمية k_أقرب جار k-nearest neighbor نموذجًا للتعرف على الأنماط يمكن استخدامهفي التصنيف وكذلك الانحدار. غالبًا ما يتم اختصارها كـ k- NN ، فإن kفي k أقرب جار هو عدد صحيح موجب، والذي يكون عادةً صغيرًا.في أي من التصنيف أو الانحدار، ستتألف المدخلات من أمثلة التدريب الأقرب لـ k داخل مساحة.

سوف نركز على تصنيف k- NN.في هذه الطريقة، يكون الإخراج هو عضوية فئة. سيؤدي هذا إلى تعيين كائن جديد للفئة الأكثر شيوعًا بين جيرانها الأقرب.في حالة k = 1، يتم تخصيص الكائن لفئة أقرب جار منفرد.

دعونا نلقي نظرة على مثال k ـ الجار الأقرب.في الرسم البياني أدناه، توجد كائنات ماسية زرقاء وأجسام ذات نجمة برتقالية. تنتمي هذه إلى فئتين منفصلتين: فئة الماس وفئة النجوم.

k ـ أقرب جار مجموعة البيانات الأولية

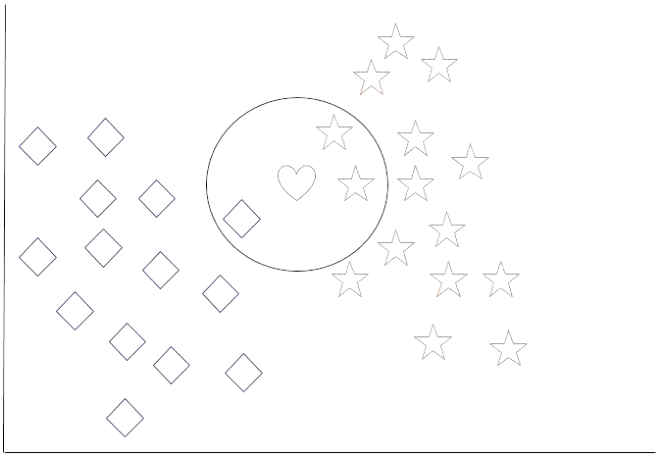
عند إضافة كائن جديد إلى الفضاء ـ في هذه الحالة قلب أخضر ـ سنريد من خوارزمية التعلم الآلي أن تصنف القلب إلى فئة معينة.



مجموعة بيانات k ـ الجار الأقرب مع كائن جديد لتصنيفه

عندما نختار k = 3، سوف تجد الخوارزمية أقرب ثلاثة جيران للقلب الأخضر من أجل تصنيفها إما إلى فئة الماس أو فئة النجوم.

في الرسم البياني لدينا، أقرب ثلاثة جيران للقلب الأخضر هم ماسة واحدة ونجمتان. لذلك، ستصنف الخوارزمية القلب بفئة النجوم.



مجموعة بيانات k_ الجار الأقرب مع اكتمال التصنيف

من بين أبسط خوارزميات التعلم الآلي، يعتبر k_أقرب جار نوعًا من "التعلم الكسول" حيث لا يحدث التعميم خارج بيانات التدريب حتى يتم إجراء استعلام على النظام.

تعلم شجرة القرار

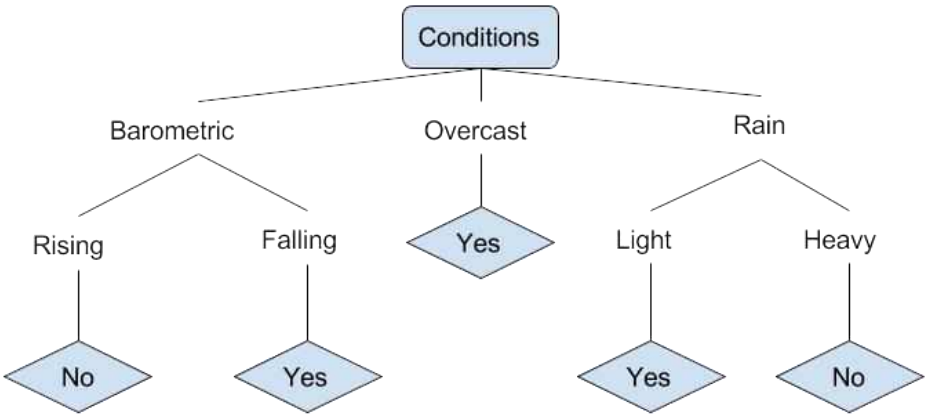
للاستخدام العام، يتم استخدام أشجار القرار Decision Tree لتمثيل القرارات بصريًا وإظهار أو إبلاغ عملية صنع القرار. عند العمل مع التعلم الآلي واستخراج البيانات، يتم استخدام أشجار القرار كنموذج تنبؤي. ترسم هذه النماذج الملاحظات حول البيانات إلى استنتاجات حول القيمة المستهدفة للبيانات.

الهدف من تعلم شجرة القرار هو إنشاء نموذج يتنبأ بقيمة الهدف بناءً على متغيرات الإدخال.

في النموذج التنبئي، يتم تمثيل سمات البيانات التي يتم تحديدها من خلال الملاحظة بواسطة الفروع، بينما يتم تمثيل الاستنتاجات حول القيمة المستهدفة للبياناتفي الأوراق.

عند "تعلم" شجرة، يتم تقسيم بيانات المصدر إلى مجموعات فرعية بناءً على اختبار قيمة السمة، والذي يتكرر على كل مجموعة فرعية مشتقة بشكل متكرر. بمجرد أن يكون للمجموعة الفرعيةفي العقدة القيمة المكافئة للقيمة المستهدفة، ستكتمل العملية العودية recursion.

دعونا نلقي نظرة على مثال لمختلف الحالات التي يمكن أن تحدد ما إذا كان يجب على شخص ما أن يذهب أو لا. وهذا يشمل الظروف الجوية وكذلك ظروف الضغط الجوي.



مثال شجرة القرار

في شجرة القرار المبسطة أعلاه، تم تصنيف المثال بفرزه عبر الشجرة إلى العقدة الورقية المناسبة. ثم يعيد هذا التصنيف المرتبط بورقة معينة، وهوفي هذه الحالة إما نعم أو لا. تصنف الشجرة ظروف اليوم بناءً على ما إذا كانت مناسبة للتنزه أم لا.

قد تحتوي مجموعة بيانات شجرة التصنيف الحقيقية على ميزات أكثر بكثير مما تم توضيحه أعلاه، ولكن يجب أن تكون العلاقات مباشرة لتحديدها. عند العمل مع تعلم شجرة القرار، يجب إجراء العديد من التحديدات، بمافي ذلك الميزات التي يجب اختيارها، والشروط التي يجب استخدامها للتقسيم، وفهم متى تصل شجرة القرار إلى نهاية واضحة.

التعلم العميق

يحاول التعلم العميق Deep Learning تقليد كيف يمكن للدماغ البشري معالجة محفزات الضوء والصوتفي الرؤية والسمع. بُنية التعلم العميق مستوحاة من الشبكات العصبية البيولوجية وتتكون من طبقات متعددةفي شبكة عصبية صناعية مكونة من أجهزة ووحدات معالجة رسومات.

يستخدم التعلم العميق سلسلة من طبقات وحدة المعالجة غير الخطية لاستخراج أو تحويل ميزات (أو تمثيلات) البيانات. يعمل إخراج طبقة واحدة كمدخل للطبقة المتتالية.في التعلم العميق، يمكن أن تخضع الخوارزميات للإشراف وتعمل على تصنيف البيانات، أو بدون إشراف وإجراء تحليل النمط.

من بين خوارزميات التعلم الآلي التي يتم استخدامها وتطويرها حاليًا، يمتص التعلم العميق معظم البيانات وتمكن من التغلب على البشرفي بعض المهام المعرفية. بسبب هذه السمات، أصبح التعلم العميق هو النهج الذي ينطوي على إمكانات كبيرةفي فضاء الذكاء الاصطناعي

لقد حقق كل من الرؤية الحاسوبية والتعرف على الكلام تطورات مهمة من مناهج التعلم العميق. يعد IBM Watson مثالًا معروفًا على النظام الذي يستفيد من التعلم العميق.

التحيزات البشرية

على الرغم من أن البيانات والتحليلات الحسابية قد تجعلنا نعتقد أننا نتلقى معلومات موضوعية، إلا أن هذا ليس هو الحال؛ الاستناد إلى البيانات لا يعني أن مخرجات التعلم الآلي محايدة. يلعب التحيز البشري Human Biase دورًافي كيفية جمع البيانات وتنظيمها وفي النهايةفي الخوارزميات التي تحدد كيفية تفاعل التعلم الآلي مع تلك البيانات.

على سبيل المثال، إذا كان الأشخاص يقدمون صورًا لـ "سمكة" كبيانات لتدريب خوارزمية، واختيار هؤلاء الأشخاص بأغلبية ساحقة من السمك الذهبي، فإن الكمبيوتر قد لا يصنف سمكة قرش على أنها سمكة. هذا من شأنه أن يخلق تحيزًا ضد أسماك القرش كسمكة، ولن يتم احتساب أسماك القرش على أنها سمكة.

عند استخدام الصور التاريخية للعلماء كبيانات تدريبية، قد لا يصنف الكمبيوتر العلماء الملونين أو النساء بشكل صحيح.في الواقع، أشارت الأبحاث الحديثة التي تمت مراجعتها من قبل الأقران إلى أن برامج الذكاء الاصطناعي والتعلم الآلي تظهر تحيزات شبيهة بالإنسان تشمل التحيزات العرقية والجنسية. راجع، على سبيل المثال، "الدلالات المشتقة تلقائيًا من مجموعات اللغة تحتوي على تحيزات شبيهة بالإنسان" و "الرجال يحبون التسوق أيضًا: تقليل تضخيم التحيز الجنسي باستخدام قيود على مستوى المجموعة" [PDF].

نظرًا لتزايد الاستفادة من التعلم الآليفي الأعمال التجارية، يمكن للتحيزات غير المعلنة أن تديم المشكلات المنهجية التي قد تمنع الأشخاص من التأهل للحصول على قروض، أو من عرض إعلانات لفرص عمل عالية الأجر، أو من تلقي خيارات التسليمفي نفس اليوم.

نظرًا لأن التحيز البشري يمكن أن يؤثر سلبًا على الآخرين، فمن المهم للغاية أن تكون على دراية به، وأن تعمل أيضًا على القضاء عليه قدر الإمكان. تتمثل إحدى طرق العمل نحو تحقيق ذلكفي ضمان وجود أشخاص متنوعين يعملونفي مشروع ما وأن الأشخاص المتنوعين يقومون باختباره ومراجعته. دعا آخرون الأطراف الثالثة التنظيمية إلى مراقبة الخوارزميات ومراجعتها، وبناء أنظمة بديلة يمكنها اكتشاف التحيزات، ومراجعات الأخلاقيات كجزء من تخطيط مشروع علم البيانات. يمكن أن تعمل زيادة الوعي حول التحيزات، ومراعاة التحيزات اللاواعية لدينا، وهيكلة العدالةفي مشاريع التعلم الآلي وخطوط الأنابيب لدينا على مكافحة التحيزفي هذا المجال.

الاستنتاج

استعرض هذا البرنامج التعليمي بعض حالات استخدام التعلم الآلي، والطرق الشائعة والأساليب الشائعة المستخدمةفي هذا المجال، ولغات برمجة التعلم الآلي المناسبة، كما تناول بعض الأشياء التي يجب وضعهافي الاعتبار من حيث التحيزات اللاواعية التي يتم تكرارهافي الخوارزميات.

نظرًا لأن التعلم الآلي مجال يتم ابتكاره باستمرار، فمن المهم أن تضعفي اعتبارك أن الخوارزميات والأساليب والنهج ستستمرفي التغيير.

تعد بايثون حاليًا واحدة من أكثر لغات البرمجة شيوعًا للاستخدام مع تطبيقات التعلم الآليفي المجالات المهنية.

4

كيفية بناء مصنف للتعلم الآلي في بايثون باستخدام Scikit-Learn

كيفية بناء مصنف للتعلم الآلي في بايثون باستخدام Scikit-Learn

في هذا البرنامج التعليمي، ستنفذ خوارزمية بسيطة للتعلم الآليفي بايثون باستخدام Scikit-Learn، وهي أداة تعلم آلي لبايثون. باستخدام قاعدة بيانات لمعلومات أورام سرطان الثدي، ستستخدم تصنيف نايف بايز Naive Bayes (NB) الذي يتنبأ بما إذا كان الورم خبيثًا أم حميدًا أم لا.

بنهاية هذا البرنامج التعليمي، ستعرف كيفية بناء نموذج التعلم الآلي الخاص بكفي بايثون.

المتطلبات الأساسية

لإكمال هذا البرنامج التعليمي، سنستخدم Jupyter Notebooks، وهي طريقة مفيدة وتفاعلية لإجراء تجارب التعلم الآلي. باستخدام Jupyter Notebooks، يمكنك تشغيل مجموعات قصيرة من التعليمات البرمجية والاطلاع على النتائج بسرعة، مما يسهل اختبار التعليمات البرمجية وتصحيحها.

للاستعداد والتشغيل بسرعة، يمكنك فتح مستعرض ويب والانتقال إلى موقع Try Jupyter على الويب: jupyter.org/try. من هناك، انقر فوق Try Jupyter with Python، وسيتم نقلك إلى Jupyter Notebooks تفاعلي حيث يمكنك البدءفي كتابة كود بايثون.

إذا كنت ترغبفي معرفة المزيد حول Jupyter Notebooks وكيفية إعداد بيئة برمجة بايثون الخاصة بك لاستخدامها مع Jupyter، فيمكنك قراءة البرنامج التعليمي الخاص بنا حول كيفية إعداد Jupyter Notebook لـ بايثون 3.

الخطوة 1 - استيراد Scikit-Learn

لنبدأ بتثبيت وحدة بايثون النمطية Scikit-Learn، وهي واحدة من أفضل مكتبات التعلم الآلي وأكثرها توثيقًافي بايثون.

لبدء مشروع الترميز الخاص بنا، دعنا ننشط بيئة برمجة بايثون 3 الخاصة بنا. تأكد من أنكفي الدليل حيث توجد بيئتك، وقم بتشغيل الأمر التالي:

```
. my_env/bin/activate
```

مع تنشيط بيئة البرمجة لدينا، تحقق لمعرفة ما إذا كانت وحدة Scikit-Learn مثبتة بالفعل:

```
(my_env) $ python -c "import sklearn"
```

إذا تم تثبيت `sklearn`، فسيكتمل هذا الأمر بدون أخطاء. إذا لم يتم تثبيته، فسترى رسالة الخطأ التالية:

Output

```
Traceback (most recent call last): File "<string>",
line 1, in <module>

ImportError: No module named 'sklearn'
```

تشير رسالة الخطأ إلى أن `sklearn` غير مثبت، لذا قم بتنزيل المكتبة باستخدام `pip`:

```
(my_env) $ pip install scikit-learn[alldeps]
```

بمجرد اكتمال التثبيت، قم بتشغيل Jupyter Notebook:

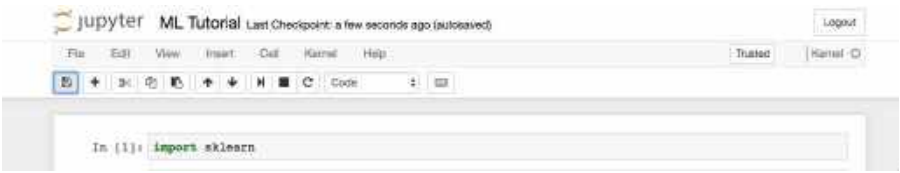
```
(my_env) $ jupyter notebook
```

في Jupyter، قم بإنشاء Python Notebook جديد يسمى ML Tutorial.في الخلية الأولى من Notebook، قم باستيراد وحدة `sklearn` النمطية:

ML Tutorial

```
import sklearn
```

يجب أن يبدو notebookالخاص بك بالشكل التالي:



Jupyter Notebook مع خلية بايثون واحدة ، والتي تستورد sklearn

الآن بعد أن تم استيراد `sklearn` في notebook الخاص بنا ، يمكننا البدءفي العمل مع مجموعة البيانات الخاصة بنموذج التعلم الآلي الخاص بنا.

الخطوة 2 - استيراد مجموعة بيانات Scikit-Learn

مجموعة البيانات التي سنعمل معهافي هذا البرنامج التعليمي هي قاعدة بيانات تشخيص سرطان الثديفي ويسكونسن Breast Cancer Wisconsin Diagnostic Database. تتضمن مجموعة البيانات معلومات متنوعة حول أورام سرطان الثدي، بالإضافة إلى تسميات تصنيف الأورام الخبيثة أو الحميدة. تحتوي مجموعة البيانات على 569 حالات أو بيانات عن 569 ورمًا وتتضمن معلومات عن 30 خاصية أو سمة، مثل نصف قطر الورم والملمس والنعومة والمنطقة.

باستخدام مجموعة البيانات هذه، سنبني نموذجًا للتعلم الآلي لاستخدام معلومات الورم للتنبؤ بما إذا كان الورم خبيثًا أم حميداً.

يأتي Scikit-Learn مثبتًا بمجموعات بيانات مختلفة يمكننا تحميلهافي بايثون، ويتم تضمين مجموعة البيانات التي نريدها. استيراد مجموعة البيانات وتحميلها:

ML Tutorial

```
...
from sklearn.datasets import load_breast_cancer
# Load dataset
data = load_breast_cancer()
```

يمثل المتغير `data` كائن بايثون الذي يعمل مثل القاموس dictionary. مفاتيح القاموس المهمة التي يجب مراعاتها هي أسماء تسميات التصنيف (`target_names`)، والتسميات الفعلية (`target`)، وأسماء الخصائص / السمات (`feature_names`)، والسمات (`data`).

السمات جزء مهم من أي تصنيف. تلتقط السمات خصائص مهمة حول طبيعة البيانات. بالنظر إلى التسمية الذي نحاول التنبؤ به (الورم الخبيث مقابل الورم الحميد)، تشمل السمات المفيدة المحتملة حجم الورم ونصف قطره وملمسه.

قم بإنشاء متغيرات جديدة لكل مجموعة مهمة من المعلومات وقم بتعيين البيانات:

ML Tutorial

```
...
# Organize our data

label_names = data['target_names']
labels =  data['target']
feature_names = data['feature_names']
features = data['data']
```

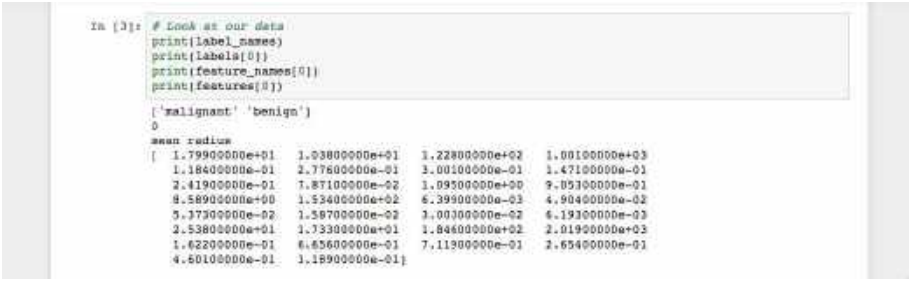
لدينا الآن قوائم lists لكل مجموعة من المعلومات. للحصول على فهم أفضل لمجموعة البيانات الخاصة بنا، دعنا نلقي نظرة على بياناتنا عن طريق طباعة تسميات الفصول الخاصة بنا، وتسمية مثيل البيانات الأول، وأسماء الميزات الخاصة بنا، وقيم الميزة لمثيل البيانات الأول:

ML Tutorial

```
...
```

```
# Look at our data
print(label_names)
print(labels[0])
print(feature_names[0])
print(features[0])
```

سترى النتائج التالية إذا قمت بتشغيل الكود:



Jupyter Notebook مع ثلاث خلايا بايثون ، والذي يطبع المثيل الأولفي مجموعة البيانات الخاصة بنا

كما تظهر الصورة، فإن أسماء الفئات لدينا خبيثة وحميدة، والتي يتم تعيينها بعد ذلك إلى قيم ثنائية من 0 و1، حيث يمثل 0 أورامًا خبيثة ويمثل 1 أورامًا حميدة. لذلك، فإن أول مثيل بياناتنا هو ورم خبيث يبلغ متوسط نصف قطره `1.79900000e+01`.

الآن بعد أن تم تحميل بياناتنا، يمكننا العمل مع بياناتنا لبناء مصنف للتعلم الآلي.

الخطوة 3 - تنظيم البيانات في مجموعات

لتقييم مدى جودة أداء المصنف، يجب دائمًا اختبار النموذج على بيانات غير مرئية. لذلك، قبل إنشاء نموذج، قسّم بياناتك إلى جزأين: مجموعة تدريب ومجموعة اختبار.

أنت تستخدم مجموعة التدريب لتدريب النموذج وتقييمه أثناء مرحلة التطوير. يمكنك بعد ذلك استخدام النموذج المدرب لعمل تنبؤات حول مجموعة الاختبار غير المرئية. يمنحك هذا الأسلوب إحساسًا بأداء النموذج وقوته.

لحسن الحظ، لدى `sklearn` دالة تسمى `train_test_split()`، والتي تقسم بياناتك إلى هذه المجموعات. استورد الدالة ثم استخدمها لتقسيم البيانات:

ML Tutorial

...

```
from sklearn.model_selection import train_test_split
# Split our data
train, test, train_labels, test_labels =
train_test_split(features, labels, test_size=0.33,
random_state=42)
```

تقوم الدالة بتقسيم البيانات بشكل عشوائي باستخدام معلمة`test_size`. في هذا المثال، لدينا الآن مجموعة اختبار (`test`) تمثل 33٪ من مجموعة البيانات الأصلية. ثم تشكل البيانات المتبقية (`train`) بيانات التدريب. لدينا أيضًا التسميات الخاصة بكل من متغيرات التدريب / الاختبار، أي `train_labels` و`test_labels`.

يمكننا الآن الانتقال إلى تدريب نموذجنا الأول.

الخطوة 4 - بناء النموذج وتقييمه

هناك العديد من النماذج للتعلم الآلي، ولكل نموذج نقاط قوته وضعفه.في هذا البرنامج التعليمي، سنركز على خوارزمية بسيطة تؤدي عادةً أداءً جيدًافي مهام التصنيف الثنائي، وهي Naive Bayes (NB).

أولاً، قم باستيراد وحدة `GaussianNB`. ثم قم بتهيئة النموذج بدالة `GaussianNB()`، ثم قم بتدريب النموذج عن طريق تحويله إلى البيانات باستخدام `gnb.fit()`:

ML Tutorial

```
...
from sklearn.naive_bayes import GaussianNB
# Initialize our classifier
gnb = GaussianNB()
# Train our classifier
model = gnb.fit(train, train_labels)
```

بعد أن نقوم بتدريب النموذج، يمكننا بعد ذلك استخدام النموذج المدرب لعمل تنبؤات على مجموعة الاختبار الخاصة بنا، والتي نقوم بها باستخدام دالة `predict()`. ترجع الدالة `predict()` مجموعة من التنبؤات لكل مثيل بياناتفي مجموعة الاختبار. يمكننا بعد ذلك طباعة تنبؤاتنا لفهم ما حدده النموذج.

استخدم دالة `predict()` مع مجموعة الاختبار واطبع النتائج:

ML Tutorial

```
...

# Make predictions

preds = gnb.predict(test) print(preds)
```

قم بتشغيل الكود وسترى النتائج التالية:

Jupyter Notebook مع خلية بايثون التي تطبع القيم المتوقعة لفئة Naive Bayes على بيانات الاختبار الخاصة بنا

كما ترىفي إخراج Jupyter Notebook، أعادت دالة `predict()` مصفوفة من الاصفار و والواحدات والتي تمثل قيمنا المتوقعة لفئة الورم (الخبيثة مقابل الحميدة).

الآن بعد أن أصبحت لدينا توقعاتنا، فلنقم بتقييم مدى جودة أداء فريقنا.

الخطوة 5 - تقييم دقة النموذج

باستخدام مصفوفة تسميات الفئات الحقيقية، يمكننا تقييم دقة القيم المتوقعة لنموذجنا من خلال مقارنة المصفوفتين (`test_labels` مقابل `preds`). سنستخدم الدالة `accuracy_score()` من `sklearn` لتحديد دقة تصنيف التعلم الآلي لدينا.

ML Tutorial

```
...

from sklearn.metrics import accuracy_score
# Evaluate accuracy
print(accuracy_score(test_labels,  preds))
```

سترى النتائج التالية:

Jupyter Notebook مع خلية بايثون التي تطبع دقة تصنيف NB الخاص بنا

كما ترىفي المخرجات، فإن تصنيف NB دقيق بنسبة 94.15٪. وهذا يعني أن 94.15في المائة من الوقت يكون المصنف قادرًا على التنبؤ الصحيح بما إذاكان الورم خبيثًا أم حميدًا أم لا.

تشير هذه النتائج إلى أن مجموعتنا المميزة المكونة من 30 سمة هي مؤشرات جيدة لفئة الورم.

لقد نجحتفي بناء أول مصنف للتعلم الآلي. دعنا نعيد تنظيم الكود بوضع جميع عبارات `import` في الجزء العلوي من Notebook أو النص البرمجي. يجب أن تبدو النسخة النهائية من الكود كما يلي:

ML Tutorial

```
from sklearn.datasets import load_breast_cancer

from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score


# Load dataset

data = load_breast_cancer()


# Organize our data


label_names = data['target_names']
labels = data['target']
feature_names = data['feature_names']
features = data['data']
```

```
# Look at our data

print(label_names)

print('Class label = ', labels[0])
print(feature_names)

print(features[0])

# Split our data


train, test, train_labels, test_labels =
train_test_split(features,labels, test_size=0.33,
random_state=42)

# Initialize our classifier

gnb = GaussianNB()


# Train our classifier

model = gnb.fit(train, train_labels)


# Make predictions

preds = gnb.predict(test)

print(preds)


# Evaluate accuracy
print(accuracy_score(test_labels, preds))
```

يمكنك الآن مواصلة العمل مع الكود الخاص بك لمعرفة ما إذا كان بإمكانك تحسين أداء المصنف الخاص بك بشكل أفضل. يمكنك تجربة مجموعات فرعية مختلفة من الميزات أو حتى تجربة خوارزميات مختلفة تمامًا. تحقق من موقع Scikit-Learn على الويب على scikit-learn.org/stable للحصول على مزيد من أفكار التعلم الآلي.

الاستنتاج

في هذا البرنامج التعليمي، تعلمت كيفية إنشاء مصنف للتعلم الآليفي بايثون. يمكنك الآن تحميل البيانات وتنظيم البيانات وتدريب مصنفات التعلم الآلي والتنبؤ بهم وتقييمهمفي بايثون باستخدام Scikit-Learn. يجب أن تساعدك الخطواتفي هذا البرنامج التعليمي على تسهيل عملية العمل مع بياناتك الخاصةفي بايثون.

5

كيفية بناء شبكة عصبية للتعرف على الأرقام المكتوبة بخط اليد باستخدام TensorFlow

كيفية بناء شبكة عصبية للتعرف على الأرقام المكتوبة بخط اليد باستخدام TensorFlow

تُستخدم الشبكات العصبية كوسيلة للتعلم العميق، وهي واحدة من العديد من الحقول الفرعية للذكاء الاصطناعي. تم اقتراحها لأول مرة منذ حوالي 70 عامًا كمحاولة لمحاكاة الطريقة التي يعمل بها الدماغ البشري، وإن كان ذلكفي شكل أكثر بساطة. ترتبط "الخلايا العصبية" الفرديةفي طبقات، مع تحديد أوزان لتحديد كيفية استجابة الخلايا العصبية عندما تنتشر الإشارات عبر الشبكة.في السابق، كانت الشبكات العصبية محدودةفي عدد الخلايا العصبية التي كانت قادرة على محاكاتها، وبالتالي فإن تعقيد التعلم الذي يمكنهم تحقيقه. ولكنفي السنوات الأخيرة، وبسبب التقدمفي تطوير الأجهزة، تمكنا من بناء شبكات عميقة للغاية، وتدريبها على مجموعات بيانات هائلة لتحقيق اختراقاتفي الذكاء الآلي.

سمحت هذه الاختراقات للآلات بمطابقة قدرات البشر وتجاوزهافي أداء مهام معينة. إحدى هذه المهام هي التعرف على الأشياء. على الرغم من أن الآلات لم تكن تاريخياً قادرة على مطابقة الرؤية البشرية، إلا أن التطورات الحديثةفي التعلم العميق جعلت من الممكن بناء شبكات عصبية يمكنها التعرف على الأشياء والوجوه والنصوص وحتى المشاعر.

في هذا البرنامج التعليمي، ستقوم بتنفيذ قسم فرعي صغير من التعرف على الكائن (التعرف على الأرقام). باستخدام TensorFlow (https: //www.tensorflow.org/)، مكتبة بايثون مفتوحة المصدر.

تم تطويرها بواسطة Google Brain labs لأبحاث التعلم العميق، وسوف تلتقط صورًا مرسومة يدويًا للأرقام من 0 إلى 9 وتقوم ببناء وتدريب شبكة عصبية للتعرف والتنبؤ بالتسمية الصحيحة للرقم المعروض. على الرغم من أنك لن تحتاج إلى خبرة سابقةفي التعلم العميق العملي أو TensorFlow لمتابعة هذا البرنامج التعليمي، فإننا نفترض بعض الإلمام بمصطلحات ومفاهيم التعلم الآلي مثل التدريب والاختبار والميزات والتسميات والتحسين والتقييم.

المتطلبات الأساسية

لإكمال هذا البرنامج التعليمي، ستحتاج إلى بيئة تطوير بايثون 3 محلية أو بعيدة تتضمن نقطة لتثبيت حزم بايثون وvenv لإنشاء بيئات افتراضية.

الخطوة 1 – اعداد المشروع

قبل أن تتمكن من تطوير برنامج التعرف، ستحتاج إلى تثبيت بعض التبعيات وإنشاء مساحة عمل لتضمين ملفاتك.

سنستخدم بيئة بايثون 3 الافتراضية لإدارة تبعيات مشروعنا. قم بإنشاء دليل جديد لمشروعك وانتقل إلى الدليل الجديد:

```
mkdir tensorflow-democd
tensorflow-demo
```

قم بتنفيذ الأوامر التالية لإعداد البيئة الافتراضية لهذا البرنامج التعليمي:

```
python3 -m venv tensorflow-demo source
tensorflow-demo/bin/activate
```

بعد ذلك، قم بتثبيت المكتبات التي ستستخدمهافي هذا البرنامج التعليمي. سنستخدم إصدارات محددة من هذه المكتبات من خلال إنشاء ملف `requirements.txt` في دليل المشروع الذي يحدد المتطلبات والإصدار الذي نحتاجه. قم بإنشاء `requirements.txt`:

```
(tensorflow-demo) $ touch requirements.txt
```

افتح الملففي محرر النصوص وأضف الأسطر التالية لتحديد مكتبات Image وNumPy وTensorFlow وإصداراتها:

requirements.txt

```
image==1.5.20
numpy==1.14.3
tensorflow==1.4.0
```

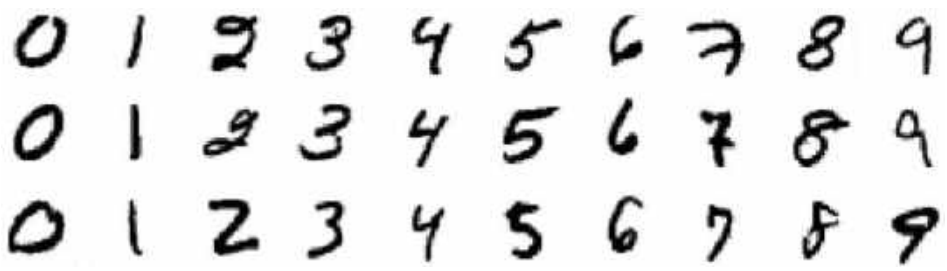
احفظ الملف واخرج من المحرر. ثم قم بتثبيت هذه المكتبات باستخدام الأمر التالي:

```
(tensorflow-demo) $ pip install -r requirements.txt
```

مع تثبيت التبعيات، يمكننا البدءفي العمل على مشروعنا.

الخطوة 2 - استيراد مجموعة بيانات MNIST

تسمى مجموعة البيانات التي سنستخدمهافي هذا البرنامج التعليمي مجموعة بيانات MNIST، وهي مجموعة بيانات كلاسيكيةفي مجتمع التعلم الآلي. تتكون مجموعة البيانات هذه من صور بأرقام مكتوبة بخط اليد، بحجم 28 × 28 بكسل. فيما يلي بعض الأمثلة على الأرقام المضمنةفي مجموعة البيانات:



أمثلة على صور MNIST

دعنا ننشئ برنامج بايثون للعمل مع مجموعة البيانات هذه. سنستخدم ملفًا واحدًا لكل عملنافي هذا البرنامج التعليمي. قم بإنشاء ملف جديد يسمى `main.py`:

```
(tensorflow-demo) $ touch main.py
```

افتح الآن هذا الملففي محرر النصوص الذي تختاره وأضف هذا السطر من التعليمات البرمجية إلى الملف لاستيراد مكتبة TensorFlow:

main.py

```
import tensorflow as tf
```

أضف سطور التعليمات البرمجية التالية إلى ملفك لاستيراد مجموعة بيانات MNIST وتخزين بيانات الصورةفي المتغير `mnist`:

main.py

```
...

from tensorflow.examples.tutorials.mnist import
input_data

mnist = input_data.read_data_sets("MNIST_data/",

one_hot=True) # ylabels are oh-encoded
```

عند قراءة البيانات، نستخدم ترميزًا واحدًا ساخنًا one-hot-encoding لتمثيل التسميات (الرقم الفعلي المرسوم، على سبيل المثال "3") للصور. يستخدم الترميز الواحد الساخن متجهًا للقيم الثنائية لتمثيل القيم الرقمية أو الفئوية. نظرًا لأن تسمياتنا للأرقام من 0 إلى 9، فإن المتجه يحتوي على عشر قيم، واحدة لكل رقم محتمل. يتم تعيين إحدى هذه القيم على 1، لتمثيل الرقم

في هذا الفهرس للمتجه، ويتم تعيين الباقي على 0. على سبيل المثال، يتم تمثيل الرقم 3 باستخدام المتجه:

```
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
```

نظرًا لأنه يتم تخزين القيمةفي الفهرس 3 على أنها 1، فإن المتجه يمثل الرقم 3.

لتمثيل الصور الفعلية نفسها، يتم تسطيح 28 × 28 بكسلفي متجه 1D بحجم 784 بكسل. يتم تخزين كل بكسل من الـ 784 بكسل المكونة للصورة كقيمة بين 0 و255. وهذا يحدد التدرج الرمادي للبكسل، حيث يتم تقديم صورنا بالأبيض والأسود فقط. لذلك يتم تمثيل البكسل الأسود بـ 255، والبكسل الأبيض بـ 0، مع ظلال مختلفة من الرماديفي مكان ما بينهما.

يمكننا استخدام المتغير `mnist` لمعرفة حجم مجموعة البيانات التي قمنا باستيرادها للتو. بالنظر إلى عدد_الأمثلة لكل مجموعة من المجموعات الفرعية الثلاث، يمكننا تحديد أن مجموعة البيانات قد تم تقسيمها إلى 55000 صورة للتدريب و5000 صورة للتحقق و10000 صورة للاختبار. أضف الأسطر التالية إلى ملفك:

main.py

```
...
n_train = mnist.train.num_examples # 55,000
n_validation = mnist.validation.num_examples # 5000
n_test = mnist.test.num_examples # 10,000
```

الآن بعد أن تم استيراد بياناتنا، حان الوقت للتفكيرفي الشبكة العصبية.

الخطوة 3 - تحديد بُنية الشبكة العصبية

تشير بُنية الشبكة العصبية إلى عناصر مثل عدد الطبقاتفي الشبكة، وعدد الوحداتفي كل طبقة، وكيفية توصيل الوحدات بين الطبقات. نظرًا لأن الشبكات العصبية مستوحاة بشكل فضفاض من عمل الدماغ البشري، يتم استخدام مصطلح الوحدة هنا لتمثيل ما قد نفكر فيه من الناحية البيولوجية على أنه خلية عصبية. مثل الخلايا العصبية التي تمرر إشارات حول الدماغ، تأخذ الوحدات بعض القيم من الوحدات السابقة كمدخلات، وتجري عملية حسابية، ثم تنقل القيمة الجديدة كمخرجات إلى وحدات أخرى. يتم وضع هذه الوحداتفي طبقات لتشكيل الشبكة، بدءًا من طبقة واحدة على الأقل لإدخال القيم وطبقة واحدة لقيم الإخراج. يستخدم مصطلح الطبقة المخفية لجميع الطبقات الموجودة بين طبقات الإدخال والإخراج، أي تلك "المخفية" من العالم الحقيقي.

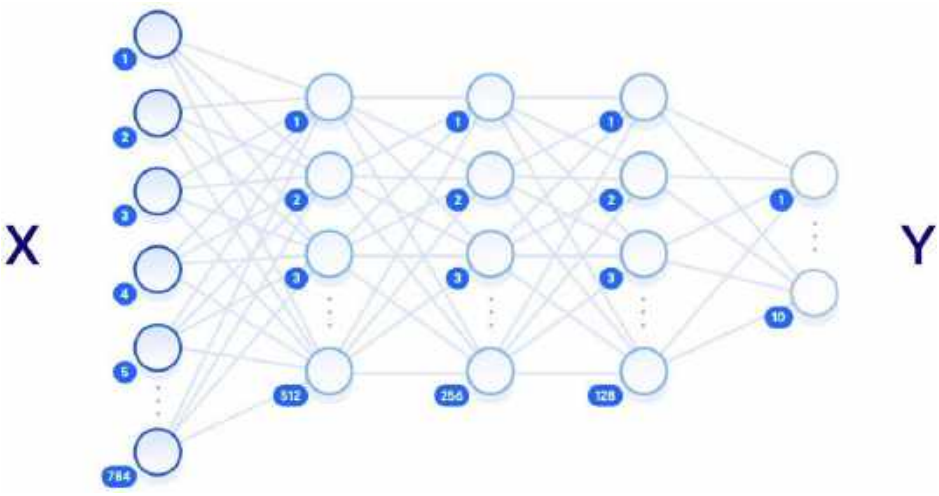
يمكن أن تسفر البُنى المختلفة عن نتائج مختلفة بشكل كبير، حيث يمكن التفكيرفي الأداء كدالة للهندسة المعمارية من بين أشياء أخرى، مثل المعلمات والبيانات ومدة التدريب.

أضف سطور التعليمات البرمجية التالية إلى ملفك لتخزين عدد الوحدات لكل طبقةفي المتغيرات العامة. يتيح لنا ذلك تغيير بُنية الشبكةفي مكان واحد، وفي نهاية البرنامج التعليمي يمكنك اختبار بنفسك كيف ستؤثر الأرقام المختلفة للطبقات والوحدات على نتائج نموذجنا:

main.py

```
...
n_input = 784 # input layer (28x28 pixels)
n_hidden1 = 512 # 1st hidden layer
n_hidden2 = 256 # 2nd hidden layer
n_hidden3 = 128 # 3rd hidden layer
n_output = 10  # output layer (0-9 digits)
```

يُظهر الرسم البياني التالي تصورًا للبنية التي صممناها، مع كل طبقة متصلة بالكامل بالطبقات المحيطة:



رسم تخطيطي للشبكة العصبية

يرتبط مصطلح "الشبكة العصبية العميقة" بعدد الطبقات المخفية، حيث تعني كلمة "الضحلة shallow" عادةً طبقة مخفية واحدة فقط، بينما تشير كلمة "عميق deep" إلى طبقات مخفية متعددة. بالنظر إلى بيانات تدريب كافية، يجب أن تكون الشبكة العصبية الضحلة التي تحتوي على عدد كافٍ من الوحدات قادرة نظريًا على تمثيل أي دالة يمكن لشبكة عصبية عميقة. ولكن

غالبًا ما يكون من المفيد أكثر من الناحية الحسابية استخدام شبكة عصبية عميقة أصغر لتحقيق نفس المهمة التي تتطلب شبكة ضحلة بها وحدات مخفية أكثر بشكل كبير.

غالبًا ما تصادف الشبكات العصبونية الضحلة إفراطًافي التعلم overfitting، حيث تحفظ الشبكة أساسًا بيانات التدريب التي شاهدتها، ولا تكون قادرة على تعميم المعرفة على البيانات الجديدة. هذا هو السببفي استخدام الشبكات العصبية العميقة بشكل أكثر شيوعًا: تسمح الطبقات المتعددة بين بيانات الإدخال الأولية وتسمية الإخراج للشبكة بتعلم الميزات على مستويات مختلفة من التجريد، مما يجعل الشبكة نفسها أكثر قدرة على التعميم.

العناصر الأخرى للشبكة العصبية التي يجب تحديدها هنا هي المعلمات الفائقة hyperparameters. على عكس المعلمات التي سيتم تحديثها أثناء التدريب، يتم تعيين هذه القيمفي البداية وتبقى ثابتة طوال العملية.في ملفك، اضبط المتغيرات والقيم التالية:

main.py

```
...

learning_rate = 1e-4
n_iterations = 1000
batch_size = 128
dropout = 0.5
```

يمثل معدل التعلم learning rate مقدار تعديل المعلماتفي كل خطوة من خطوات عملية التعلم. تعد هذه التعديلات مكونًا رئيسيًا للتدريب: بعد كل مرور عبر الشبكة، نقوم بضبط الأوزان قليلاً لمحاولة تقليل الخسارة. يمكن أن تتقارب معدلات التعلم الأكبر بشكل أسرع، ولكن لديها أيضًا القدرة على تجاوز القيم المثلى عند تحديثها. يشير عدد التكرارات number of iterations إلى عدد المرات التي نمر فيها بخطوة التدريب، ويشير حجم الدُفعة batch size إلى عدد أمثلة التدريب التي نستخدمهافي كل خطوة. يمثل متغير التسرب `dropout` الحد الذي عنده نحذف بعض الوحدات بشكل عشوائي. سنستخدم `dropout`في الطبقة المخفية النهائية لمنح كل وحدة فرصة بنسبة 50٪ للتخلص منهافي كل خطوة تدريب. هذا يساعد على منع الإفراط في التعلم.

لقد حددنا الآن بنية شبكتنا العصبية، والمعلمات الفائقة التي تؤثر على عملية التعلم. الخطوة التالية هي بناء الشبكة كرسم بياني TensorFlow.

الخطوة 4 - بناء الرسم البياني لـ TensorFlow

لبناء شبكتنا، سنقوم بإعداد الشبكة كرسم بياني حسابي لتنفيذه TensorFlow. المفهوم الأساسي لـ TensorFlow هو الموتر tensor، وهي بُنية بيانات مشابهة لمصفوفة أو قائمة. تهيئتها ومعالجتها أثناء تمريرها عبر الرسم البياني وتحديثها من خلال عملية التعلم.

سنبدأ بتحديد ثلاث موترات كعناصر نائبة placeholders، وهي عبارة عن موترات سنقوم بتغذية القيم بها لاحقًا. أضف ما يلي إلى ملفك:

main.py

```
...

X = tf.placeholder("float", [None, n_input])
Y = tf.placeholder("float", [None, n_output])
keep_prob = tf.placeholder(tf.float32)
```

المعلمة الوحيدة التي يجب تحديدهافي إعلانها هي حجم البيانات التي سنغديها. بالنسبة إلى X، نستخدم شكل `[None, 784]`، حيث `None` يمثل أي كمية، حيث سنقوم بتغذية غير محدد عدد الصور 784 بكسل. شكل Y هو `[None, 10]` لأننا سنستخدمه لعدد غير محدد من مخرجات التسمية، مع 10 فئات محتملة. يتم استخدام موتر `keep_prob` للتحكمفي معدل التسرب dropout rate، ونقوم بتهيئته كعنصر نائب بدلاً من متغير غير قابل للتعديل immutable variable لأننا نريد استخدام نفس الموتر للتدريب (عند تعيين `dropout` على `0.5`) والاختبار (عند تعيين `dropout` على `1.0`).

المعلمات التي ستقوم الشبكة بتحديثهافي عملية التدريب هي قيم الوزن `weight` والتحيز `bias`، لذلك نحتاج إلى تعيين قيمة أولية بدلاً من عنصر نائب فارغ. هذه القيم هي أساسًا حيث تقوم الشبكة بالتعلم، حيث يتم استخدامهافي دوال التنشيط للخلايا العصبية، والتي تمثل قوة الاتصالات بين الوحدات.

نظرًا لأنه تم تحسين القيم أثناء التدريب، يمكننا ضبطها على صفرفي الوقت الحالي. لكن القيمة الأولية لهافي الواقع تأثير كبير على الدقة النهائية للنموذج. سنستخدم قيمًا عشوائية من التوزيع الطبيعي المقتطع للأوزان. نريدهم أن يكونوا قريبين من الصفر، حتى يتمكنوا من التكيف إمافي اتجاه إيجابي أو سلبي، ومختلفة بعض الشيء، بحيث يولدون أخطاء مختلفة. سيضمن هذا أن يتعلم النموذج شيئًا مفيدًا. أضف هذه الأسطر:

main.py

```
...

weights = {
    'w1':  tf.Variable(tf.truncated_normal([n_input,
n_hidden1],stddev=0.1)),

    'w2':
tf.Variable(tf.truncated_normal([n_hidden1,
n_hidden2],stddev=0.1)),

    'w3':
tf.Variable(tf.truncated_normal([n_hidden2,
n_hidden3],stddev=0.1)),

    'out':
tf.Variable(tf.truncated_normal([n_hidden3,
n_output],

stddev=0.1)),

}
```

بالنسبة للانحياز، نستخدم قيمة ثابتة صغيرة لضمان تنشيط الموترات في المراحل الأولية وبالتالي المساهمة في الانتشار. يتم تخزين الأوزان وموترات التحيز في كائنات القاموس لسهولة الوصول إليها. أضف هذه الشفرة البرمجية إلى ملفك لتحديد التحيزات:

main.py

```
...

biases = {
```

```
  b1':tf.Variable(tf.constant(0.1,shape=[n_hidden1])
  ),
  b2':tf.Variable(tf.constant(0.1,shape=[n_hidden2])
  ), '
  b3':tf.Variable(tf.constant(0.1,shape=[n_hidden3])
  ), '
  out':tf.Variable(tf.constant(0.1,
  shape=[n_output]))
}
```

بعد ذلك، قم بإعداد طبقات الشبكة عن طريق تحديد العمليات التي ستعالج الموترات. أضف هذه السطور إلى ملفك:

main.py

```
...

layer_1=tf.add(tf.matmul(X,weights['w1']),biases['
b1'])
layer_2        =        tf.add(tf.matmul(layer_1,
weights['w2']), biases['b2'])
layer_3        =        tf.add(tf.matmul(layer_2,
weights['w3']), biases['b3'])
layer_drop = tf.nn.dropout(layer_3, keep_prob)
output_layer = tf.matmul(layer_3, weights['out'])
+ biases['out']
```

ستنفذ كل طبقة مخفية عملية ضرب المصفوفة على مخرجات الطبقة السابقة وأوزان الطبقة الحالية، وتضيف الانحياز إلى هذه القيم.في آخر طبقة مخفية، سنطبق عملية التسرب باستخدام قيمة `keep_prob` الخاصة بنا 0.5.

الخطوة النهائيةفي بناء الرسم البياني هي تحديد دالة الخسارة التي نريد تحسينها. الاختيار الشائع لدالة الخسارةفي برامج TensorFlow هو الانتروبيا المتقاطعة cross-entropy، والمعروفة أيضًا باسم log-loss، والتي تحدد الفرق بين توزيعين احتماليين (التنبؤات والتسميات). سيؤدي التصنيف المثالي إلى إنتروبيا متقاطعة مقدارها 0، مع تقليل الخسارة تمامًا.

نحتاج أيضًا إلى اختيار خوارزمية التحسين التي سيتم استخدامها لتقليل دالة الخسارة. عملية تسمى تحسين التدرج الاشتقاقي gradient descent هي طريقة شائعة لإيجاد الحد الأدنى

(المحلي) لدالة ما عن طريق اتخاذ خطوات تكرارية على طول التدرجفي اتجاه سلبي (تنازلي). هناك العديد من الخيارات لخوارزميات تحسين التدرج الاشتقاقي التي تم تنفيذها بالفعلفي TensorFlow، وفي هذا البرنامج التعليمي سنستخدم مُحسِّن آدم Adam optimizer. يمتد هذا إلى تحسين التدرج الاشتقاقي باستخدام الزخم لتسريع العملية من خلال حساب متوسط مرجح أُسيًا للتدرجات واستخدام ذلكفي التعديلات. أضف الكود التالي إلى ملفك:

main.py

```
...
cross_entropy = tf.reduce_mean(
tf.nn.softmax_cross_entropy_with_logits(
labels=Y, logits=output_layer
))
 train_step = tf.train.AdamOptimizer(1e-
 4).minimize(cross_entropy)
```

لقد حددنا الآن الشبكة وقمنا ببنائها باستخدام .TensorFlow الخطوة التالية هي تغذية البيانات من خلال الرسم البياني لتدريبها، ثم اختبار ما إذا كانت قد تعلمت شيئًا بالفعل.

الخطوة 5 - التدريب والاختبار

تتضمن عملية التدريب تغذية مجموعة بيانات التدريب من خلال الرسم البياني وتحسين دالة الخسارة.في كل مرة تقوم الشبكة بالتكرار من خلال مجموعة من المزيد من صور التدريب، تقوم بتحديث المعلمات لتقليل الخسارة من أجل التنبؤ بشكل أكثر دقة بالأرقام المعروضة. تتضمن عملية الاختبار تشغيل مجموعة بيانات الاختبار الخاصة بنا من خلال الرسم البياني المدرب، وتتبع عدد الصور التي تم توقعها بشكل صحيح، حتى نتمكن من حساب الدقة.

قبل بدء عملية التدريب، سنحدد طريقتنافي تقييم الدقة حتى نتمكن من طباعتها على دفعات صغيرة mini-batches من البيانات أثناء التدريب. ستسمح لنا هذه البيانات المطبوعة بالتحقق من أنه بدءًا من التكرار الأول وحتى الأخير، تقل الخسارة وتزداد الدقة؛ سيسمحون لنا أيضًا بتتبع ما إذا كنا قد أجرينا عددًا كافيًا من التكرارات للوصول إلى نتيجة متسقة ومثلى:

main.py

```
...
correct_pred = tf.equal(tf.argmax(output_layer,
1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred,
tf.float32))
```

في correct_pred ، نستخدم الدالة arg_max لمقارنة الصور التي يتم توقعها بشكل صحيح من خلال النظر إلى output_layer (التنبؤات) و Y (التسميات) ، ونستخدم دالة equal لإرجاعها كقائمة من القيم المنطقية Booleans. يمكننا بعد ذلك بث هذه القائمة الى floats وحساب الوسط للحصول على درجة دقة كاملة.

نحن الآن جاهزون لتهيئة جلسة لتشغيل الرسم البياني.في هذه الجلسة، سنقوم بتغذية الشبكة بأمثلة التدريب الخاصة بنا، وبمجرد التدريب، نقوم بتغذية نفس الرسم البياني بأمثلة اختبار جديدة لتحديد دقة النموذج. أضف سطور التعليمات البرمجية التالية إلى ملفك:

main.py

```
...
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
```

جوهر عملية التدريبفي التعلم العميق هو تحسين دالة الخسارة. نحن هنا نهدف إلى تقليل الاختلاف بين التسميات المتوقعة للصور والتسميات الحقيقية للصور. تتضمن العملية أربع خطوات تتكرر لعدد محدد من التكرارات:

- انشر القيم إلى الأمام عبر الشبكة.
- احسب الخسارة.
- انشر القيم بشكل عكسي عبر الشبكة.
- قم بتحديث المعلمات.

في كل خطوة تدريب، يتم تعديل المعلمات قليلاً لمحاولة تقليل الخسارة للخطوة التالية. مع تقدم التعلم، يجب أن نرى انخفاضًافي الخسارة، وفي النهاية يمكننا التوقف عن التدريب واستخدام الشبكة كنموذج لاختبار بياناتنا الجديدة.

أضف هذا الكود إلى الملف:

```
# train on mini batches

for i in range(n_iterations):
    batch_x,batch_y=mnist.train.next_batch(batch_
    size)
```

```
    sess.run(train_step, feed_dict={X: batch_x, Y:
    batch_y, keep_prob: dropout})


    # print loss and accuracy (per minibatch)
    if i % 100 == 0:
        minibatch_loss,minibatch_accuracy=
    sess.run( [cross_entropy, accuracy],
        feed_dict={X:    batch_x,    Y:    batch_y,
    keep_prob: 1.0}
)
    print(

        "Iteration",
        str(i),
        "\t| Loss =",
        str(minibatch_loss),
        "\t| Accuracy =",
        str(minibatch_accuracy)
        )
```

بعد 100 تكرار لكل خطوة تدريب نقوم فيها بتغذية دفعة صغيرة من الصور عبر الشبكة، نقوم بطباعة فقد تلك الدفعة ودقتها. لاحظ أنه لا ينبغي أن نتوقع خسارة متناقصة وزيادة الدقة هنا، حيث إن القيم لكل دفعة، وليس للنموذج بأكمله. نحن نستخدم دفعات صغيرة من الصور بدلاً من إطعامها بشكل فردي لتسريع عملية التدريب والسماح للشبكة بمشاهدة عدد من الأمثلة المختلفة قبل تحديث المعلمات.

بمجرد اكتمال التدريب، يمكننا تشغيل الجلسة على صور الاختبار. نستخدم هذه المرة معدل تسرب `keep_prob` مساوي لـ`1.0` للتأكد من أن جميع الوحدات نشطةفي عملية الاختبار.

أضف هذا الكود إلى الملف:

main.py

```
...
```

```
test_accuracy = sess.run(accuracy, feed_dict={X:
mnist.test.images,      Y:      mnist.test.labels,
keep_prob: 1.0})
print("\nAccuracy on test set:", test_accuracy)
```

حان الوقت الآن لتشغيل برنامجنا ومعرفة مدى دقة شبكتنا العصبيةفي التعرف على هذه الأرقام المكتوبة بخط اليد. احفظ `main.py` وقم بتنفيذ الأمر التاليفي التيرمينال لتشغيل السكريبت:

```
(tensorflow-demo) $ python main.py
```

سترى ناتجًا مشابهًا لما يلي، على الرغم من أن نتائج الخسارة الفردية والدقة قد تختلف قليلاً:

```
Iteration 0       |  Loss = 3.67079      |    Accuracy = 0.140625
Iteration 100     |  Loss = 0.492122     |    Accuracy = 0.84375
Iteration 200     |  Loss =    0.421595  |    Accuracy = 0.882812
Iteration 300     |  Loss = 0.307726     |    Accuracy = 0.921875
Iteration 400     |  Loss = 0.392948     |    Accuracy = 0.882812
Iteration 500     |  Loss = 0.371461     |    Accuracy = 0.90625
Iteration 600     |  Loss = 0.378425     |    Accuracy = 0.882812
Iteration 700     |  Loss = 0.338605     |    Accuracy = 0.914062
Iteration 800     |  Loss = 0.379697     |    Accuracy = 0.875
Iteration 900     |  Loss = 0.444303     |    Accuracy = 0.90625

  Accuracy on test set: 0.9206
```

لمحاولة تحسين دقة نموذجنا، أو لمعرفة المزيد حول تأثير ضبط المعلمات الفائقة، يمكننا اختبار تأثير تغيير معدل التعلم، وعتبة التسرب، وحجم الدفعة، وعدد التكرارات. يمكننا أيضًا تغيير عدد الوحداتفي طبقاتنا المخفية، وتغيير كمية الطبقات المخفية نفسها، لنرى كيف تزيد البنى المختلفة من دقة النموذج أو تقللها.

لإثبات أن الشبكة تتعرف بالفعل على الصور المرسومة يدويًا، دعنا نختبرها على صورة واحدة خاصة بنا.

إذا كنت تستخدم جهازًا محليًا وترغبفي استخدام الرقم المرسوم يدويًا، فيمكنك استخدام محرر رسومات لإنشاء صورة رقم 28 × 28 بكسل الخاصة بك. بخلاف ذلك، يمكنك استخدام `curl` لتنزيل عينة الاختبار التالية إلى الخادم أو الكمبيوتر:

```
(tensorflow-demo) $ curl -O images/test_img.png
```

افتح `main.py` في المحرر الخاص بك وأضف سطور التعليمات البرمجية التالية إلى أعلى الملف لاستيراد مكتبتين ضروريتين لمعالجة الصور.

main.py

```
import numpy as np
from PIL import Image
...
```

ثمفي نهاية الملف، أضف السطر التالي من التعليمات البرمجية لتحميل صورة الاختبار للرقم المكتوب بخط اليد:

main.py

```
...
img=np.invert(Image.open("test_img.png").convert('
L')).ravel()
```

تقوم دالة `open` لمكتبة `Image` بتحميل صورة الاختبار كمصفوفة 4D تحتوي على قنوات ألوان RGB الثلاث وشفافية ألفا. هذا ليس نفس التمثيل الذي استخدمناه سابقًا عند القراءةفي مجموعة البيانات باستخدامTensorFlow ، لذلك سنحتاج إلى القيام ببعض الأعمال الإضافية لمطابقة التنسيق.

أولاً، نستخدم دالة `convert` مع المعلمة `L` لتقليل تمثيل 4D RGBA إلى قناة ألوان رمادية واحدة. نخزن هذا كمصفوفة `numpy` ونعكسها باستخدام `np.invert` ، لأن المصفوفة الحالية تمثل الأسود 0 والأبيض كـ 255 ، بينما نحتاج إلى العكس. أخيرًا، نسمي`ravel` لتسطيح المصفوفة.

الآن بعد أن تم تنظيم بيانات الصورة بشكل صحيح، يمكننا تشغيل جلسة بنفس الطريقة كمافي السابق، ولكن هذه المرة يتم التغذية فقطفي صورة واحدة للاختبار.

أضف الكود التالي إلى ملفك لاختبار الصورة وطباعة التسمية الناتج.

main.py

```
...

prediction = sess.run(tf.argmax(output_layer, 1),
feed_dict={X: [img]})
```

```
print ("Prediction for test image:", np.squeeze(prediction))
```

يتم استدعاء دالة `np.squeeze` عند التنبؤ لإرجاع عدد صحيح واحد من المصفوفة (أي الانتقال من [2] إلى 2). يوضح الاخراج الناتج أن الشبكة قد تعرفت على هذه الصورة على أنها الرقم 2.

Output

```
Prediction for test image: 2
```

يمكنك محاولة اختبار الشبكة باستخدام صور أكثر تعقيداً ـ أرقام تشبه الأرقام الأخرى، على سبيل المثال، أو الأرقام التي تم رسمها بشكل سيء أو غير صحيح — لنرى كم هو جيد.

الاستنتاج

في هذا البرنامج التعليمي، نجحتفي تدريب شبكة عصبية لتصنيف مجموعة بيانات MNIST بدقة تصل إلى 92٪ واختبرتها على صورة خاصة بك. تحقق أحدث الأبحاث الحالية حوالي 99٪ حول هذه المشكلة نفسها، باستخدام بُنى شبكات أكثر تعقيداً تتضمن طبقات تلافيفية. تستخدم هذه البُنية ثنائية الأبعاد للصورة لتمثيل المحتويات بشكل أفضل، على عكس طريقتنا التي ـ خففت كل وحدات البكسلفي متجه واحد من 784 وحدة. يمكنك قراءة المزيد حول هذا الموضوع على موقع TensorFlow الإلكتروني، والاطلاع على المقالات البحثية التي توضح بالتفصيل النتائج الأكثر دقة على موقع MNIST.

الآن بعد أن عرفت كيفية إنشاء شبكة عصبية وتدريبها، يمكنك تجربة هذا التطبيق واستخدامه على بياناتك الخاصة، أو اختباره على مجموعات البيانات الشائعة الأخرى مثل Google StreetView House Numbers ، أو مجموعة بيانات CIFAR-10 لمزيد من التعرف على الصور العامة.

6

التحيز-التباين للتعلم المعزز العميق: كيفية بناء بوت لـ Atari باستخدام OpenAI Gym

التحيز-التباين للتعلم المعزز العميق: كيفية بناء بوت لـ Atari باستخدام OpenAI Gym

التعلم المعزز Reinforcement learning هو مجال فرعي ضمن نظرية التحكم control theory، والذي يهتم بأنظمة التحكم التي تتغير بمرور الوقت وتشمل على نطاق واسع تطبيقات مثل السيارات ذاتية القيادة والروبوتات والروبوتات للألعاب.في هذا الدليل، ستستخدم التعلم المعزز لبناء بوت لألعاب الفيديو أتاري. لا يتم منح هذا البوت الوصول إلى المعلومات الداخلية حول اللعبة. بدلاً من ذلك، يتم منحه فقط حق الوصول إلى عرض اللعبة والمكافأة على هذا العرض، مما يعني أنه يمكنه فقط رؤية ما يراه اللاعب البشري.

في التعلم الآلي، يُعرف البوت رسميًا باسم الوكيل agent.في حالة هذا البرنامج التعليمي، الوكيل هو "لاعب"في النظام يعمل وفقًا لوظيفة اتخاذ القرار، تسمى السياسة policy. الهدف الأساسي هو تطوير وكلاء أقوياء من خلال تسليحهم بسياسات قوية. بعبارة أخرى، هدفنا هو تطوير بوتات ذكية من خلال تسليحها بقدرات قوية على اتخاذ القرار.

ستبدأ هذا البرنامج التعليمي من خلال تدريب وكيل التعلم المعزز الأساسي الذي يتخذ إجراءات عشوائية عند لعب Space Invaders، لعبة أركيد Atari الكلاسيكية، والتي ستكون بمثابة خط الأساس للمقارنة. بعد ذلك، سوف تستكشف العديد من التقنيات الأخرى ـ بمافي ذلك Q ـ التعلم Q- learning، والتعلم العميق deep Q-learning، والمربعات الصغرى least squares ـ أثناء بناء الوكلاء الذين يلعبون لعبة Space Invaders و Frozen Lake، وهي بيئة ألعاب بسيطة مدرجةفي Gym (https: //gym.openai .com /)، مجموعة أدوات التعلم المعزز الصادرة عن OpenAI (https://openai.com/). باتباع هذا البرنامج التعليمي، ستكتسب فهمًا للمفاهيم الأساسية التي تحكم اختيار الفرد لتعقيد النموذجفي التعلم الآلي.

المتطلبات الأساسية

لإكمال هذا البرنامج التعليمي، سوف تحتاج إلى:

- خادم يعمل بنظام التشغيل Ubuntu 18.04، مع ذاكرة وصول عشوائي (RAM) لا تقل عن 1 جيجابايت. يجب أن يحتوي هذا الخادم على مستخدم غير جذر مع امتيازات sudo، بالإضافة إلى إعادة تم إعدادها باستخدام UFW. يمكنك إعداد هذا باتباع دليل الإعداد الأولي للخادم لـ Ubuntu 18.04.
- بيئة افتراضية بايثون 3 يمكنك تحقيقها من خلال قراءة دليلنا "كيفية تثبيت بايثون 3 وإعداد بيئة برمجة على خادم Ubuntu 18.04".

بدلاً من ذلك، إذا كنت تستخدم جهازًا محليًا، فيمكنك تثبيت بايثون 3 وإعداد بيئة برمجة محلية من خلال قراءة البرنامج التعليمي المناسب لنظام التشغيل الخاص بك عبر سلسلة تثبيت وإعداد بايثون.

الخطوة 1 - إنشاء المشروع وتثبيت التبعيات

من أجل إعداد بيئة التطوير لبوتاتك، يجب عليك تنزيل اللعبة نفسها والمكتبات اللازمة للحساب.

ابدأ بإنشاء مساحة عمل لهذا المشروع باسم `AtariBot`:

```
mkdir ~/AtariBot
```

انتقل إلى دليل `AtariBot` الجديد:

```
cd ~/AtariBot
```

ثم قم بإنشاء بيئة افتراضية جديدة للمشروع. يمكنك تسمية هذه البيئة الافتراضية بأي شيء تريده؛ هنا، سوف نسميها `ataribot`:

```
python3 -m venv ataribot
```

قم بتنشيط بيئتك:

```
source ataribot/bin/activate
```

في Ubuntu، اعتبارًا من الإصدار 16.04، يتطلب OpenCV تثبيت بعض الحزم الإضافية حتى تعمل. يتضمن ذلك CMake ـ وهو تطبيق يدير عمليات بناء البرامج ـ بالإضافة إلى مدير الجلسة والإضافات المتنوعة وتكوين الصور الرقمية. قم بتشغيل الأمر التالي لتثبيت هذه الحزم:

```
sudo apt-get install -y cmake libsm6 libxext6
libxrender-dev libz-dev
```

ملاحظة: إذا كنت تتبع هذا الدليل على جهاز محلي يعمل بنظام MacOS، فإن البرنامج الإضافي الوحيد الذي تحتاج إلى تثبيته هو CMake. قم بتثبيته باستخدام Homebrew (الذي ستثبته إذا اتبعت البرنامج التعليمي المتطلب MacOS) عن طريق كتابة:

```
brew install cmake
```

بعد ذلك، استخدم `pip` لتثبيت حزمة `wheel`، والتطبيق المرجعي لمعيار تحزيم wheel. مكتبة بايثون، هذه الحزمة بمثابة امتداد لبناء wheels وتتضمن أداة سطر الأوامر للعمل مع `.whl`:

```
python -m pip install wheel
```

بالإضافة إلى `wheel`، ستحتاج إلى تثبيت الحزم التالية:

- Gym، مكتبة بايثون التي تتيح العديد من الألعاب للبحث، بالإضافة إلى جميع التبعيات لألعاب Atari. يقدم Gym، الذي طورته OpenAI، معايير عامة لكل لعبة من الألعاب بحيث يمكن تقييم أداء الوكلاء والخوارزميات المختلفة بشكل موحد.
- Tensorflow ، مكتبة تعليمية عميقة. تمنحنا هذه المكتبة القدرة على إجراء العمليات الحسابية بكفاءة أكبر. على وجه التحديد، يقوم بذلك عن طريق بناء دوال رياضية باستخدام تجريدات Tensorflow التي تعمل بشكل حصري على وحدة معالجة الرسومات الخاصة بك.
- OpenCV ، مكتبة الرؤية الحاسوبية المذكورة سابقًا.
- SciPy ، مكتبة حوسبة علمية تقدم خوارزميات تحسين فعالة.
- NumPy ، مكتبة الجبر الخطي.

قم بتثبيت كل من هذه الحزم باستخدام الأمر التالي. لاحظ أن هذا الأمر يحدد إصدار كل حزمة لتثبيتها:

```
python -m pip install gym==0.9.5 tensorflow==1.5.0
tensorpack==0.8.0 numpy==1.14.0 scipy==1.1.0
opencv-python==3.4.1.15
```

بعد ذلك، استخدم `pip` مرة أخرى لتثبيت بيئات Atariفي Gym، والتي تتضمن مجموعة متنوعة من ألعاب الفيديو Atari، بمافي ذلك Space Invaders:

```
python -m pip install gym[atari]
```

إذا نجح تثبيت حزمة `gym[atari]`، سينتهي إخراجك بما يلي:

Output

```
Installing collected packages: atari-py, Pillow,
PyOpenGL Successfully installed Pillow-5.4.1
PyOpenGL-3.1.0 atari-py-0.1.7
```

مع تثبيت هذه التبعيات، ستكون جاهزًا للمضي قدمًا وبناء وكيل يلعب بشكل عشوائي ليكون بمثابة الأساس للمقارنة.

الخطوة 2 - إنشاء وكيل عشوائي أساسي باستخدام Gym

الآن بعد أن أصبح البرنامج المطلوب على الخادم الخاص بك، ستقوم بإعداد وكيل يقوم بتشغيل نسخة مبسطة من لعبة Atari الكلاسيكية، Space Invaders. لأي تجربة، من الضروري الحصول على خط أساس لمساعدتك على فهم مدى جودة أداء النموذج الخاص بك. نظرًا لأن هذا الوكيل يتخذ إجراءات عشوائيةفي كل إطار، فسنشير إليه على أنه وكيلنا الأساسي العشوائي.في هذه الحالة، ستقارن بالعامل الأساسي هذا لفهم مدى جودة أداء وكلائكفي خطوات لاحقة.

مع Gym، يمكنك الحفاظ على حلقة اللعب الخاصة بك. هذا يعني أنك تتعامل مع كل خطوةفي تنفيذ اللعبة:في كل خطوة، تمنح gym حركة جديدة وتطلب من gym تحديد حالة اللعبة.في هذا البرنامج التعليمي، حالة اللعبة هي مظهر اللعبةفي خطوة زمنية معينة، وهي بالضبط ما ستراه إذا كنت تلعب اللعبة.

باستخدام محرر النصوص المفضل لديك، قم بإنشاء ملف بايثون باسم bot_2_random.py. هنا، سنستخدم nano:

```
nano bot_2_random.py
```

ملاحظة: خلال هذا الدليل، تتم محاذاة أسماء برامج البوت مع رقم الخطوة الذي تظهر به، بدلاً من الترتيب الذي تظهر به. ومن ثم، فإن هذا البوت يسمى bot_2_random.py بدلاً من bot_1_random.py.

ابدأ هذا السكريبت بإضافة الأسطر المميزة التالية. تتضمن هذه الأسطر كتلة تعليق تشرح ما سيفعله هذا السكريبت وجملتي import لاستيراد الحزم التي سيحتاجها هذا البرنامج النصي في النهاية لكي يعمل:

```
AtariBot/bot_2_random.py

"""
```

```
Bot 2 -- Make a random, baseline agent for the
SpaceInvaders game. """
import gym
import random
```

أضف دالة `main`.في هذه الدالة، قم بإنشاء بيئة اللعبة `SpaceInvaders-v0` ثم قم بتهيئة اللعبة باستخدام `env.reset` :

/AtariBot/bot_2_random.py

```
. . .
import gym
import random
def main():
env = gym.make('SpaceInvaders-v0')
env.reset()
```

بعد ذلك، أضف دالة `env.step`. يمكن لهذه الدالة إرجاع أنواع القيم التالية:

- `state`: الحالة الجديدة للعبة بعد تطبيق الإجراء المقدم.
- `reward`: الزيادةفي النتيجة التي تتكبدها الحالة. على سبيل المثال، يمكن أن يحدث هذا عندما تدمر رصاصة كائنًا فضائيًا، وتزيد النتيجة بمقدار 50 نقطة. بعد ذلك، `reward=50`.في لعب أي لعبة قائمة على النتائج، يكون هدف اللاعب هو تعظيم النتيجة. هذا مرادف لتعظيم المكافأة الإجمالية.
- `done`: ما إذا كانت الحلقة قد انتهت أم لا ، والتي تحدث عادةً عندما يفقد اللاعب كل الأرواح.
- `info`: المعلومات الدخيلة التي ستضعها جانبًافي الوقت الحالي.

سوف تستخدم `reward` لحساب إجمالي مكافأتك. ستستخدم `done` أيضًا لتحديد موعد وفاة اللاعب، والذي سيكون عندما `done` ترجع `True`.

أضف حلقة اللعبة التالية، والتي ترشد اللعبة إلى التكرار حتى يموت اللاعب:

/AtariBot/bot_2_random.py

```
. . .
def main():
env = gym.make('SpaceInvaders-v0')
env.reset()
      episode_reward = 0
      while True:
            action = env.action_space.sample()
            _, reward, done, _ = env.step(action)
            episode_reward += reward
            if done:
                  print('Reward: %s' %episode_reward)
                  break
```

أخيرًا، قم بتشغيل دالة `main`. قم بتضمين `__name__` للتأكد من أن `main` تعمل فقط عند استدعائها مباشرة باستخدام `python bot_2_random.py`. إذا لم تقم بإضافة `if` للتحقق، فسيتم تشغيل `main` دائمًا عند تنفيذ ملف بايثون، حتى عند استيراد الملف. وبالتالي، فمن الممارسات الجيدة وضع الكودفي دالة `main` يتم تنفيذها فقط عندما يكون `'__main__' == __name__`.

/AtariBot/bot_2_random.py

```
. . .
def main():
. . .
if done:
print('Reward %s' % episode_reward)

break
if __name__ == '__main__':
main()
```

احفظ الملف واخرج من المحرر. إذا كنت تستخدم `nano`، فافعل ذلك بالضغط على CTRL+X, Y، ثم ENTER. بعد ذلك، قم بتشغيل السكريبت الخاص بك عن طريق كتابة:

```
python bot_2_random.py
```

سيقوم برنامجك بإخراج رقم مشابه لما يلي. لاحظ أنهفي كل مرة تقوم فيها بتشغيل الملف ستحصل على نتيجة مختلفة:

Output

```
Making new env: SpaceInvaders-v0
Reward: 210.0
```

هذه النتائج العشوائية تمثل مشكلة. من أجل إنتاج عمل يمكن للباحثين والممارسين الآخرين الاستفادة منه، يجب أن تكون نتائجك وتجاربك قابلة للتكرار. لتصحيح ذلك، أعد فتح السكريبت:

```
nano bot_2_random.py
```

بعد `import random`، أضف `random.seed(0)` بعد `env = gym.make('SpaceInvaders-v0')`، أضف `env.seed(0)` تعمل هذه الخطوط معًا على "seed " البيئة من خلال نقطة انطلاق متسقة، مما يضمن إمكانية تكرار النتائج دائمًا. سيتطابق ملفك النهائي مع ما يلي تمامًا:

/AtariBot/bot_2_random.py

```
"""
Bot 2 -- Make a random, baseline agent for the
SpaceInvaders game.
"""

import gym
import random

random.seed(0)

def main():
env = gym.make('SpaceInvaders-v0')
env.seed(0)

      env.reset()
      episode_reward = 0
      while True:
            action = env.action_space.sample()
            _, reward, done, _ = env.step(action)
            episode_reward += reward
            if done:
                  print('Reward: %s' %episode_reward)
                  break
if **name** == '**main**':
main()
```

احفظ الملف وأغلق المحرر، ثم قم بتشغيل السكريبت عن طريق كتابة ما يليفي التيرمينال:

```
python bot_2_random.py
```

سيخرج هذا المكافأة التالية بالضبط:

Output
```
Making new env: SpaceInvaders-v0
Reward: 555.0
```

هذا هو البوت الأول الخاص بك، على الرغم من أنه غير ذكي إلى حد ما لأنه لا يأخذفي الحسبان البيئة المحيطة عند اتخاذ القرارات. للحصول على تقدير أكثر موثوقية لأداء البوت الخاص بك، يمكنك تشغيل الوكيل لعدة حلقاتفي وقت واحد، والإبلاغ عن المكافآت التي تم حساب متوسطها عبر حلقات متعددة. للتأكد من ذلك أعد فتح الملف أولاً:

```
nano bot_2_random.py
```

بعد`random.seed(0)` ، أضف السطر المميز التالي الذي يخبر الوكيل بلعب اللعبة لمدة 10 حلقات:

```
. . .
random.seed(0)
num_episodes = 10
. . .
```

مباشرة بعد`env.seed(0)` ، ابدأ قائمة جديدة من المكافآت:

/AtariBot/bot_2_random.py
```
. . .
env.seed(0)
rewards = []
. . .
```

راجع كل التعليمات البرمجية من `env.reset()` إلى نهاية `main()` في حلقة `for` ، مع تكرار `num_episodes`. تأكد من وضع مسافة بادئة لكل سطر من `env.reset()` لـ `break` بأربع مسافات:

/AtariBot/bot_2_random.py
```
. . .
def main():
env = gym.make('SpaceInvaders-v0')
env.seed(0)
rewards = []
for _ in range(num_episodes):
      env.reset()
      episode_reward = 0
      while True:
            ...
```

قبل break مباشرةً، وهو السطر الأخير حاليًا من حلقة اللعبة الرئيسية، أضف مكافأة الحلقة الحالية إلى قائمة جميع المكافآت:

/AtariBot/bot_2_random.py

```
. . .
if done:
print('Reward: %s' % episode_reward)
rewards.append(episode_reward)
break
. . .
```

في نهاية دالة main، قم بالإبلاغ عن متوسط المكافأة:

/AtariBot/bot_2_random.py

```
. . .
def main():
...
print('Reward: %s' % episode_reward)
break
print('Average reward: %.2f' % (sum(rewards) /
len(rewards)))
. . .
```

ستتم محاذاة ملفك الآن مع ما يلي. يرجى ملاحظة أن مقطع التعليمات البرمجية التالي يتضمن بعض التعليقات لتوضيح الأجزاء الرئيسية من السكريبت:

/AtariBot/bot_2_random.py

```
"""
Bot 2 -- Make a random, baseline agent for the
SpaceInvaders game.
"""

import gym
import random

random.seed(0) # make results reproducible

num_episodes = 10

def main():
      env = gym.make('SpaceInvaders-v0') # create
the game
      env.seed(0) # make results reproducible
      rewards = []
```

```
    for _ in range(num_episodes):
        env.reset()
        episode_reward = 0
        while True:
            action = env.action_space.sample()
            _, reward, done, _ =
            env.step(action) # random action
            episode_reward += reward
            if done:
                print('Reward: %d' %
                episode_reward)
                rewards.append(episode_reward
                )
                break
print('Average reward: %.2f' % (sum(rewards) /
len(rewards)))
if __name__ == '__main__':
main()
```

احفظ الملف، واخرج من المحرر، وقم بتشغيل السكريبت:

```
python bot_2_random.py
```

سيؤدي هذا إلى طباعة متوسط المكافأة التالية، بالضبط:

```
Output
Making new env: SpaceInvaders-v0
. . .
Average reward: 163.50
```

لدينا الآن تقدير أكثر موثوقية لدرجة خط الأساس للتغلب عليها. لإنشاء وكيل متفوق، على الرغم من ذلك، ستحتاج إلى فهم إطار عمل التعلم المعزز. كيف يمكن للمرء أن يجعل الفكرة المجردة "لصنع القرار" أكثر واقعية؟

فهم التعلم المعزز

في أي لعبة، هدف اللاعب هو زيادة نقاطه إلى الحد الأقصى.في هذا الدليل، يُشار إلى درجة اللاعب على أنها مكافأته. لتعظيم مكافآتهم، يجب أن يكون اللاعب قادرًا على إعادة بناء قدراته على اتخاذ القرار. بشكل رسمي، القرار هو عملية النظر إلى اللعبة، أو مراقبة حالة اللعبة، واختيار الإجراء. تسمى دالة صنع القرار لدينا بالسياسة؛ تقبل السياسة الحالة كمدخلات و "تقرر" إجراءً:

```
policy: state -> action
```

لبناء مثل هذه الدالة، سنبدأ بمجموعة محددة من الخوارزمياتفي التعلم المعزز تسمى خوارزميات.Q-Learning لتوضيح ذلك، ضعفي اعتبارك الحالة الأولية للعبة، والتي سنسميها `state0`:سفينة الفضاء الخاصة بك والفضائيين جميعهمفي مواقعهم الأولية. بعد ذلك، افترض أن لدينا إمكانية الوصول إلى "Q-table" سحري يخبرنا مقدار المكافأة التي سيكسبها كل إجراء:

STATE	ACTION	REWARD
state0	shoot	10
state0	right	3
state0	left	3

سيؤدي إجراء `shoot` إلى زيادة مكافأتك إلى الحد الأقصى، حيث ينتج عنه المكافأة بأعلى قيمة: 10. كما ترى، يوفر جدول Q طريقة مباشرة لاتخاذ القرارات، بناءً على الحالة المرصودة:

```
policy: state -> look at Q-table, pick action with
greatest reward
```

ومع ذلك، تحتوي معظم الألعاب على عدد كبير جدًا من الحالات لإدراجهافي جدول.في مثل هذه الحالات، يتعلم وكيل Q-Learning دالة Q بدلاً من Q-table .نحن نستخدم دالة Q هذه بشكل مشابه لكيفية استخدامنا لجدول Q سابقًا. تعطينا إعادة كتابة مدخلات الجدول كدوال ما يلي:

```
Q(state0, shoot) = 10
Q(state0, right) = 3
Q(state0, left) = 3
```

بالنظر إلى حالة معينة، من السهل علينا اتخاذ قرار: فنحن ببساطة ننظر إلى كل إجراء محتمل ومكافأته، ثم نتخذ الإجراء الذي يتوافق مع أعلى مكافأة متوقعة. إعادة صياغة السياسة السابقة بشكل أكثر رسمية، لدينا:

```
policy: state -> argmax_{action} Q(state, action)
```

هذا يفي بمتطلبات دالة اتخاذ القرار: نظرًا لوجود حالةفي اللعبة، فإنه يقرر إجراء ما. ومع ذلك، فإن هذا الحل يعتمد على معرفة`Q(state, action)` لكل حالة واجراء. لتقدير `Q(state, action)` ، ضعفي اعتبارك ما يلي:

1. بالنظر إلى العديد من الملاحظات حول حالات الوكيل وأفعاله ومكافآته، يمكن للمرء الحصول على تقدير للمكافأة لكل حالة وإجراء من خلال أخذ متوسط التشغيل.
2. Space Invaders هي لعبة ذات مكافآت متأخرة: يكافأ اللاعب عندما يتم تفجير الفضائي وليس عندما يطلق اللاعب النار. ومع ذلك، فإن قيام اللاعب بعمل ما عن طريق التسديد هو الدافع الحقيقي للمكافأة. بطريقة ما، يجب أن تخصص دالة Q مكافأة إيجابية `(state0, shoot)`.

تم ترميز هاتين البصيرتين في المعادلات التالية:

```
Q(state, action) = (1 - learning_rate) * Q(state,
action) +
learning_rate * Q_target
Q_target = reward + discount_factor * max_{action'}
Q(state', action')
```

تستخدم هذه المعادلات التعريفات التالية:

- `state`: الحالة في الوقت الحالي.
- `action`: الإجراء الذي تم اتخاذه في الوقت الحالي الخطوة.
- `reward`: مكافأة الخطوة الزمنية الحالية.
- `state`: الحالة الجديدة للخطوة التالية، بالنظر إلى أننا اتخذنا الاجراء `a`.
- `action`: جميع الإجراءات الممكنة.
- `learning_rate`: معدل التعلم.
- `discount_factor`: عامل الخصم ، مقدار المكافأة "المتدهورة" عند نشرها.

للحصول على شرح كامل لهاتين المعادلتين، راجع هذه المقالة حول فهم Q-Learning .

مع وضع هذا الفهم للتعلم المعزز في الاعتبار، كل ما تبقى هو تشغيل اللعبة فعليًا والحصول على تقديرات قيمة Q لسياسة جديدة.

الخطوة 3 - إنشاء عامل Q-Learning بسيط لـ Frozen Lake

الآن بعد أن أصبح لديك وكيل أساسي، يمكنك البدء في إنشاء وكلاء جدد ومقارنتهم بالأصل. في هذه الخطوة، ستنشئ وكيلًا يستخدم Q-Learning، وهي تقنية تعلم معزز تُستخدم لتعليم الوكيل الإجراء الذي يجب اتخاذه في حالة معينة. سيلعب هذا الوكيل لعبة جديدة، FrozenLake. تم وصف إعداد هذه اللعبة على النحو التالي على موقع Gym:

"الشتاء هنا. كنت أنت وأصدقاؤك تتجولون حول طبق فريسبي في الحديقة عندما قمت برمية بربة تركت الطبق الطائر في وسط البحيرة. يتم تجميد الماء في الغالب، ولكن هناك عدد قليل من الثقوب حيث ذاب الجليد. إذا دخلت في إحدى تلك الثقوب، فسوف تسقط في الماء المتجمد. في هذا الوقت، يوجد نقص في لعبة الطبق الطائر، لذلك من الضروري للغاية أن تتنقل عبر البحيرة وتسترجع القرص. ومع ذلك، فإن الجليد زلق، لذلك لن تتحرك دائمًا في الاتجاه الذي تريده" .

يوصف السطح باستخدام شبكة مثل ما يلي:

```
SFFF (S: starting point, safe)
FHFH (F: frozen surface, safe)
FFFH (H: hole, fall to your doom)
HFFG (G: goal, where the frisbee is located)
```

يبدأ اللاعب من أعلى اليسار، ويُشار إليه بالرمزS ، ويشق طريقه إلى الهدف في أسفل اليمين، ويُشار إليه بالرمز G. الإجراءات المتاحة هي اليمين واليسار والأعلى والأسفل، والوصول إلى نتيجة الهدف بنتيجة 1. يوجد عدد من الثقوب، يُشار إليها بالرمز H ، ويؤدي الوقوع في واحدة على الفور إلى الحصول على درجة 0.

في هذا القسم، ستقوم بتطبيق عامل Q-Learning بسيط. باستخدام ما تعلمته سابقًا، ستنشئ وكيلًا يتبادل بين الاستكشاف exploration والاستغلال exploitation.في هذا السياق، يعني الاستكشاف أن الوكيل يتصرف بشكل عشوائي، والاستغلال يعني أنه يستخدم قيم Q لاختيار ما يعتقد أنه الإجراء الأمثل. ستقوم أيضًا بإنشاء جدول للاحتفاظ بقيمQ ، وتحديثه بشكل تدريجي حيث يعمل الوكيل ويتعلم.

قم بعمل نسخة من السكريبت الخاص بك من الخطوة 2:

```
cp bot_2_random.py bot_3_q_table.py
```

ثم افتح هذا الملف الجديد للتعديل:

```
nano bot_3_q_table.py
```

ابدأ بتحديث التعليق أعلى الملف الذي يصف الغرض من السكريبت. نظرًا لأن هذا مجرد تعليق، فإن هذا التغيير ليس ضروريًا لكي يعمل السكريبت بشكل صحيح، ولكنه قد يكون مفيدًا لتتبع ما يفعله السكريبت:

```
/AtariBot/bot_3_q_table.py
"""
Bot 3 -- Build simple q-learning agent for
FrozenLake
"""
```

```
. . .
```

قبل إجراء تعديلات وظيفية على السكريبت، ستحتاج إلى استيراد `numpy` لادوات الجبر الخطي. أسفل `import gym` مباشرةً، أضف السطر المميز:

/AtariBot/bot_3_q_table.py

```
"""
Bot 3 -- Build simple q-learning agent for
FrozenLake
"""

import gym
import numpy as np
import random

random.seed(0) # make results reproducible
. . .
```

أسفل `random.seed(0)`، أضف بذرة لـ `numpy`:

/AtariBot/bot_3_q_table.py

```
. . .
import random

random.seed(0) # make results reproducible
np.random.seed(0)
. . .
```

بعد ذلك، اجعل حالات اللعبة متاحة. قم بتحديث سطر `env.reset()` لقول ما يلي، والذي يخزن الحالة الأولية للعبةفي `state` المتغيرة:

/AtariBot/bot_3_q_table.py

```
. . .
for \_ in range(num_episodes):
state = env.reset()
. . .
```

قم بتحديث سطر `env.step(...)` ليقول التالي، الذي يخزن الحالة التالية، `state2`. ستحتاج إلى كل من `state` الحالية والحالة التالية - `state2` - لتحديث دالةQ .

/AtariBot/bot_3_q_table.py

```
. . .
while True:
      action = env.action_space.sample()
```

```
      state2, reward, done, _ = env.step(action)
                                                . . .
```

بعد `episode_reward += reward` ، أضف سطرًا لتحديث المتغير `state`. هذا يحافظ على المتغير `state` محدثة للتكرار التالي، حيث ستتوقع أن تعكس `state` الحالة الحالية:

/AtariBot/bot_3_q_table.py

```
. . .
while True:
. . .
episode_reward += reward
state = state2
if done:
. . .
```

في خانة `if done`، احذف بيان `print` الذي يطبع مكافأة كل حلقة. بدلاً من ذلك، ستخرج متوسط المكافأة على عدة حلقات. ستبدو الكتلة `if done` كما يلي:

/AtariBot/bot_3_q_table.py

```
      . . .
      if done:
            rewards.append(episode_reward)
            break
            . . .
```

بعد هذه التعديلات، ستتطابق حلقة اللعبة مع ما يلي:

/AtariBot/bot_3_q_table.py

```
. . .
for _ in range(num_episodes):
      state = env.reset()
      episode_reward = 0
      while True:
            action = env.action_space.sample()
            state2, reward, done, _ =
            env.step(action)
            episode_reward += reward
            state = state2
            if done:
                  rewards.append(episode_reward))
                  break
                  . . .
```

بعد ذلك، أضف قدرة الوكيل على الموازنة بين الاستكشاف والاستغلال. مباشرة قبل حلقة اللعبة الرئيسية (التي تبدأ بـ `for . . .`)، أنشئ جدول قيم Q:

/AtariBot/bot_3_q_table.py

```
. . .
      Q = np.zeros((env.observation_space.n,
      env.action_space.n))
      for _ in range(num_episodes):
      . . .
```

ثم أعد كتابة الحلقة `for` لكشف رقم الحلقة:

/AtariBot/bot_3_q_table.py

```
. . .
      Q = np.zeros((env.observation_space.n,
      env.action_space.n))
      for episode in range(1, num_episodes + 1):
            . . .
```

داخل `while True`: حلقة اللعبة الداخلية، تخلق `noise`. يتم أحيانًا إدخال الضوضاء، أو البيانات العشوائية التي لا معنى لها، عند تدريب الشبكات العصبية العميقة لأنها يمكن أن تحسن أداء ودقة النموذج. لاحظ أنه كلما زادت الضوضاء، قلت القيمفي `Q[state, :]`. نتيجة لذلك، كلما زادت الضوضاء، زاد احتمال تصرف الوكيل بشكل مستقل عن معرفته باللعبة. بمعنى آخر، تشجع الضوضاء العالية الوكيل على استكشاف الإجراءات العشوائية:

/AtariBot/bot_3_q_table.py

```
      . . .
      while True:
            noise = np.random.random((1,
            env.action_space.n)) /
            episode**2.)
            action = env.action_space.sample()
      . . .
```

لاحظ أنه مع زيادة `episodes`، يقل مقدار الضوضاء بشكل تربيعي: مع مرور الوقت، يستكشف الوكيل أقل وأقل لأنه يمكنه الوثوقفي تقييمه الخاص لمكافأة اللعبة والبدءفي استغلال معرفتها.

قم بتحديث سطر `action` لجعل وكيلك يختار الإجراءات وفقًا لجدولQ-value ، مع بعض الاستكشافات المضمنة:

/AtariBot/bot_3_q_table.py

```
      . . .
      noise = np.random.random((1,
      env.action_space.n)) /
      (episode**2.)
      action = np.argmax(Q[state, :] + noise)
      state2, reward, done, _ = env.step(action)
      . . .
```

ستتطابق حلقة اللعبة الرئيسية بعد ذلك مع ما يلي:

/AtariBot/bot_3_q_table.py

```
. . .
Q = np.zeros((env.observation_space.n,
env.action_space.n))
for episode in range(1, num_episodes + 1):
      state = env.reset()
      episode_reward = 0
      while True:
      noise = np.random.random((1,
      env.action_space.n)) /
      (episode**2.)
      action = np.argmax(Q[state, :] + noise)
      state2, reward, done, _ = env.step(action)
      episode_reward += reward
      state = state2
      if done:
            rewards.append(episode_reward)
            break
            . . .
```

بعد ذلك، ستقوم بتحديث جدول Q-value باستخدام معادلة تحديث Bellman ، وهي معادلة مستخدمة على نطاق واسعفي التعلم الآلي للعثور على السياسة المثلىفي بيئة معينة.

تتضمن معادلة Bellman فكرتين وثيقتي الصلة بهذا المشروع. أولاً، سيؤدي اتخاذ إجراء معين من حالة معينة عدة مرات إلى تقدير جيد لقيمة Q المرتبطة بهذه الحالة والإجراء. تحقيقًا لهذه الغاية، ستزيد عدد الحلقات التي يجب على هذا البوت تشغيلها من أجل إرجاع تقدير قيمة Q أقوى. ثانيًا، يجب أن تنتشر المكافآت عبر الوقت، بحيث يتم تعيين مكافأة غير صفرية للإجراء الأصلي. هذه الفكرة واضحةفي الألعاب ذات المكافآت المتأخرة؛ على سبيل المثال،في Space Invaders، يكافأ اللاعب عندما يتم تفجير الكائن الفضائي وليس عندما يطلق اللاعب النار. ومع ذلك، فإن اللاعب الذي يطلق النار هو الدافع الحقيقي للمكافأة. وبالمثل، يجب أن تقوم دالة Q بتعيين (state0, shoot)مكافأة إيجابية.

أولاً، قم بتحديث num_episodes لتساوي 4000:

/AtariBot/bot_3_q_table.py

```
. . .
np.random.seed(0)

num_episodes = 4000
. . .
```

بعد ذلك، أضف المعلمات الفائقة الضرورية إلى الجزء العلوي من الملفـفي شكل متغيرين آخرين:

/AtariBot/bot_3_q_table.py

```
. . .
num_episodes = 4000
discount_factor = 0.8
learning_rate = 0.9
. . .
```

احسب قيمة Q الجديدة المستهدفة، مباشرة بعد السطر الذي يحتوي على env.step(...)

/AtariBot/bot_3_q_table.py

```
    . . .
    state2, reward, done, _ = env.step(action)
    Qtarget = reward + discount_factor *
    np.max(Q[state2, :])
    episode_reward += reward
    . . .
```

في السطر بعد Qtarget مباشرةً، حدِّث جدول Q-value باستخدام المتوسط المرجح لقيم Q القديمة والجديدة:

/AtariBot/bot_3_q_table.py

```
    . . .
    Qtarget = reward + discount_factor *
    np.max(Q[state2, :])
    Q[state, action] = (1-learning_rate) *
    Q[state,action] + learning_rate * Qtarget
    episode_reward += reward
    . .
```

تحقق من أن حلقة اللعبة الرئيسية تتطابق الآن مع ما يلي:

/AtariBot/bot_3_q_table.py

```
. . .
      Q = np.zeros((env.observation_space.n,
      env.action_space.n))
      for episode in range(1, num_episodes + 1):
            state = env.reset()
            episode_reward = 0
            while True:
                  noise = np.random.random((1,
                  env.action_space.n)) /
                  (episode**2.)
                  action = np.argmax(Q[state, :] +
                  noise)
                  state2, reward, done, _ =
                  env.step(action)
                  Qtarget = reward + discount_factor
                  * np.max(Q[state2, :])
                  Q[state, action] = (
                  1-learning_rate) * Q[state, action]
                  + learning_rate * Qtarget
                  episode_reward += reward
                  state = state2
                  if done:
                        rewards.append(episode_rewar)
                        break
                        . . .
```

لقد اكتمل الآن منطقنافي تدريب الوكيل. كل ما تبقى هو إضافة آليات إعداد التقارير.

على الرغم من أن بايثون لا تفرض فحصًا صارمًا للنوع، أضف أنواعًا إلى إعلانات الدوال الخاصة بك من أجل النظافة.في الجزء العلوي من الملف، قبل السطر الأول لـ `import gym`، قم باستيراد نوع `List`:

/AtariBot/bot_3_q_table.py

```
. . .
from typing import List
import gym
. . .
```

مباشرة بعد `learning_rate = 0.9`، خارج دالة `main`، قم بتعريف الفاصل الزمني والشكل للتقارير:

/AtariBot/bot_3_q_table.py

```
. . .
```

```
learning_rate = 0.9
report_interval = 500
report = '100-ep Average: %.2f . Best 100-ep
Average: %.2f . Average:
%.2f ' \
      '(Episode %d)'
def main():
. . .
```

قبل دالة `main`، أضف دالة جديدة تملأ سلسلة `report` هذه، باستخدام قائمة جميع المكافآت:

/AtariBot/bot_3_q_table.py

```
. . .
report = '100-ep Average: %.2f . Best 100-ep
Average: %.2f . Average:
%.2f ' \
'(Episode %d)'
def print_report(rewards: List, episode: int):
"""Print rewards report for current episode
- Average for last 100 episodes
- Best 100-episode average across all time
- Average for all episodes across time
"""
print(report % (
np.mean(rewards[-100:]),
max([np.mean(rewards[i:i+100]) for i in
range(len(rewards) - 100)]),
np.mean(rewards),
episode))
def main():
. . .
```

قم بتغيير اللعبة إلى Frozen Lake بدلاً من Space Invaders:

/AtariBot/bot_3_q_table.py

```
. . .
def main():
env = gym.make('FrozenLake-v0') # create the game
. . .
```

بعد `rewards.append(...)`، اطبع متوسط المكافأة على آخر 100 حلقة واطبع متوسط المكافأة عبر جميع الحلقات:

/AtariBot/bot_3_q_table.py

```
. . .
if done:
      rewards.append(episode_reward)
      if episode % report_interval == 0:
            print_report(rewards, episode)
      . . .
```

في نهاية الدالة `main()`، قم بالإبلاغ عن كلا المتوسطين مرة أخرى. افعل ذلك عن طريق استبدال السطر الذي يقرأ:

```
print('Average reward:%.2f' % (sum(rewards) /
len(rewards)))
```

مع السطر المميز التالي:

/AtariBot/bot_3_q_table.py

```
. . .
def main():
...
break
print_report(rewards, -1)
. . .
```

أخيرًا، لقد أكملت وكيل Q-Learning الخاص بك. تأكد من أن السكريبت الخاص بك يتوافق مع ما يلي:

/AtariBot/bot_3_q_table.py

```
"""
Bot 3 -- Build simple q-learning agent for
FrozenLake
"""
from typing import List
import gym
import numpy as np
import random

random.seed(0) # make results reproducible
np.random.seed(0) # make results reproducible

num_episodes = 4000
discount_factor = 0.8
learning_rate = 0.9
report_interval = 500
```

```
report = '100-ep Average: %.2f . Best 100-ep
Average: %.2f . Average:
%.2f ' \
'(Episode %d)'

def print_report(rewards: List, episode: int):
      """Print rewards report for current episode
      - Average for last 100 episodes
      - Best 100-episode average across all time
      - Average for all episodes across time
      """
      print(report % (
            np.mean(rewards[-100:]),
            max([np.mean(rewards[i:i+100]) for i in
            range(len(rewards) -
            100)]),
            np.mean(rewards),
            episode))
def main():
      env = gym.make('FrozenLake-v0') # create the
      game
      env.seed(0) # make results reproducible
      rewards = []

      Q = np.zeros((env.observation_space.n,
      env.action_space.n))
      for episode in range(1, num_episodes + 1):
            state = env.reset()
            episode_reward = 0
            while True:
                  noise = np.random.random((1,
                  env.action_space.n)) /
                  (episode**2.)
                  action = np.argmax(Q[state, :] +
                  noise)
                  state2, reward, done, _ =
                  env.step(action)
                  Qtarget = reward + discount_factor
                  * np.max(Q[state2, :])
                  Q[state, action] = (
                  1-learning_rate
                  ) * Q[state, action] +
                  learning_rate * Qtarget
                  episode_reward += reward
```

```
                state = state2
                if done:
                      rewards.append(episode_reward
                      )
                      if episode % report_interval
                      == 0:
                            print_report(rewards,
                            episode)
                            break
print_report(rewards, -1)
if __name__ == '__main__':
      main()
```

احفظ الملف، واخرج من المحرر، وقم بتشغيل السكريبت:

```
python bot_3_q_table.py
```

سيتطابق إخراجك مع ما يلي:

Output

```
100-ep Average: 0.11 . Best 100-ep Average: 0.12 .
Average: 0.03
(Episode 500)
100-ep Average: 0.25 . Best 100-ep Average: 0.24 .
Average: 0.09
(Episode 1000)
100-ep Average: 0.39 . Best 100-ep Average: 0.48 .
Average: 0.19
(Episode 1500)
100-ep Average: 0.43 . Best 100-ep Average: 0.55 .
Average: 0.25
(Episode 2000)
100-ep Average: 0.44 . Best 100-ep Average: 0.55 .
Average: 0.29
(Episode 2500)
100-ep Average: 0.64 . Best 100-ep Average: 0.68 .
Average: 0.32
(Episode 3000)
100-ep Average: 0.63 . Best 100-ep Average: 0.71 .
Average: 0.36
(Episode 3500)
100-ep Average: 0.56 . Best 100-ep Average: 0.78 .
Average: 0.40
(Episode 4000)
100-ep Average: 0.56 . Best 100-ep Average: 0.78 .
Average: 0.40
```

```
(Episode -1)
```

لديك الآن أول بوت غير عادي للألعاب، لكن دعونا نضع هذا متوسط المكافأة البالغ 0.78في المنظور الصحيح. وفقًا لصفحة Gym FrozenLake، فإن "حل" اللعبة يعني الوصول إلى معدل 100 حلقة يبلغ 0.78. بشكل غير رسمي، "الحل" يعني "لعب اللعبة بشكل جيد للغاية". بينما ليسفي وقت قياسي، فإن وكيل جدول Q قادر على حل FrozenLakeفي 4000 حلقة.

ومع ذلك، قد تكون اللعبة أكثر تعقيداً. هنا، استخدمت جدولًا لتخزين جميع الحالات الـ 144 المحتملة، لكن ضعفي اعتبارك أن tic tac toe يوجد فيها 19683 حالة محتملة. وبالمثل، ضع في اعتبارك Space Invaders حيث يوجد عدد كبير جداً من الحالات التي يمكن حسابها. جدول Q غير مستدام لأن الألعاب تزداد تعقيداً. لهذا السبب، تحتاج إلى طريقة ما لتقريب جدول Q. مع استمرار التجريبفي الخطوة التالية، ستصمم دالة يمكنها قبول الحالات والإجراءات كمدخلات وإخراج قيمة Q.

الخطوة 4 - بناء عامل Q-Learning العميق لـ Frozen Lake

في التعلم المعزز، تتنبأ الشبكة العصبية بشكل فعال بقيمة Q بناءً على مدخلات الحالة والإجراء، باستخدام جدول لتخزين جميع القيم الممكنة، لكن هذا يصبح غير مستقرفي الألعاب المعقدة. بدلاً من ذلك، يستخدم التعلم المعزز العميق شبكة عصبية لتقريب دالة Q. لمزيد من التفاصيل، راجع فهم Deep Q-Learning.

للتعود على Tensorflow، مكتبة التعلم العميق قمت بتثبيتهافي الخطوة 1، ستعيد تطبيق كل المنطق المستخدم حتى الآن مع تجريدات Tensorflow وستستخدم شبكة عصبية من أجل تقريب دالة Q الخاصة بك. ومع ذلك، ستكون شبكتك العصبية بسيطة للغاية: إخراجك `Q(s)` هو مصفوفة `W` مضروبةفي المدخلات الخاصة بك `s`. يُعرف هذا بالشبكة العصبية بطبقة واحدة متصلة بالكامل:

```
Q(s) = Ws
```

للتكرار، فإن الهدف هو إعادة تنفيذ جميع المنطق من البوتات التي قمنا ببنيها بالفعل باستخدام تجريدات Tensorflow. سيؤدي ذلك إلى جعل عملياتك أكثر أهمية، حيث يمكن لـ Tensorflow بعد ذلك إجراء جميع الحسابات على وحدة معالجة الرسومات.

ابدأ بتكرار نص جدول Q من الخطوة 3:

```
cp bot_3_q_table.py bot_4_q_network.py
```

ثم افتح الملف الجديد باستخدام `nano` أو محرر النصوص المفضل لديك:

```
nano bot_4_q_network.py
```

أولاً، قم بتحديث التعليق أعلى الملف:

/AtariBot/bot_4_q_network.py
```
"""
Bot 4 -- Use Q-learning network to train bot
"""
. . .
```

بعد ذلك، استيراد حزمة Tensorflow عن طريق إضافة توجيه `import` مباشرة اسفل `import random`. بالإضافة إلى ذلك، أضف `tf.set_radon_seed(0)` مباشرة أسفل `np.random.seed(0)`. سيضمن هذا إمكانية تكرار نتائج هذا السكريبت في جميع الجلسات:

/AtariBot/bot_4_q_network.py
```
. . .
import random
import tensorflow as tf

random.seed(0)
np.random.seed(0)
tf.set_random_seed(0)
. . .
```

أعد إنشاء المعلمات الفائقة الخاصة بكفي الجزء العلوي من الملف لمطابقة ما يلي وإضافة دالة تسمى `exploration_probability` ، والتي ستعيد احتمالية الاستكشاففي كل خطوة. تذكر أنهفي هذا السياق، يعني "الاستكشاف" اتخاذ إجراء عشوائي، بدلاً من اتخاذ الإجراء الموصى بهفي تقديرات قيمة Q:

/AtariBot/bot_4_q_network.py
```
. . .
num_episodes = 4000
discount_factor = 0.99
learning_rate = 0.15
report_interval = 500
exploration_probability = lambda episode: 50. /
(episode + 10)
report = '100-ep Average: %.2f . Best 100-ep
Average: %.2f . Average:
%.2f ' \
'(Episode %d)'
. . .
```

بعد ذلك، ستضيف دالة ترميز واحد ساخن. باختصار، ترميز واحد ساخن هو عملية يتم من خلالها تحويل المتغيرات إلى شكل يساعد خوارزميات التعلم الآلي على عمل تنبؤات أفضل. إذا كنت ترغبفي معرفة المزيد حول الترميز الواحد الساخن، فيمكنك التحقق من أمثلة الخصومةفي الرؤية الحاسوبية: كيفية إنشاء عامل فلتر الكلب القائم على العاطفة ثم خداعه.

مباشرة أسفل `report = ...`، أضف دالة `one_hot`:

/AtariBot/bot_4_q_network.py

```
. . .
report = '100-ep Average: %.2f . Best 100-ep
Average: %.2f . Average:
%.2f ' \
'(Episode %d)'

def one_hot(i: int, n: int) -> np.array:
"""Implements one-hot encoding by selecting the ith
standard basis
vector"""
return np.identity(n)[i].reshape((1, -1))

def print_report(rewards: List, episode: int):
. . .
```

بعد ذلك، ستقوم بإعادة كتابة منطق الخوارزمية باستخدام تجريدات Tensorflow. قبل القيام بذلك، ستحتاج أولاً إلى إنشاء عناصر نائبة لبياناتك.

في دالتك `main`، أسفل `rewards=[]` مباشرةً، أدخل المحتوى المميز التالي. هنا، حددت عناصر نائبة لملاحظتكفي الوقت t (مثل `obs_t_ph`) والوقت t + 1 (مثل `obs_tp1_ph`)، بالإضافة إلى العناصر النائبة للإجراء والمكافأة والهدف Q:

/AtariBot/bot_4_q_network.py

```
. . .
def main():
env = gym.make('FrozenLake-v0') # create the game
env.seed(0) # make results reproducible
rewards = []

    # 1. Setup placeholders
    n_obs, n_actions = env.observation_space.n,
    env.action_space.n
```

```
obs_t_ph = tf.placeholder(shape=[1, n_obs],
dtype=tf.float32)
obs_tp1_ph = tf.placeholder(shape=[1, n_obs],
dtype=tf.float32)
act_ph = tf.placeholder(tf.int32, shape=())
rew_ph = tf.placeholder(shape=(),
dtype=tf.float32)
q_target_ph = tf.placeholder(shape=[1,
n_actions], dtype=tf.float32)
Q = np.zeros((env.observation_space.n,
env.action_space.n))
for episode in range(1, num_episodes + 1):
. . .
```

مباشرة أسفل السطر الذي يبدأ بـ `q_target_ph =`، أدخل الأسطر المميزة التالية. يبدأ هذا الرمز الحساب الخاص بك عن طريق حساب Q(s, a)، لجميع a لجعل `q_current` و Q(s', a')، للجميع لجعل `q_target`:

/AtariBot/bot_4_q_network.py

```
. . .
rew_ph = tf.placeholder(shape=(),
dtype=tf.float32)
q_target_ph = tf.placeholder(shape=[1,
n_actions], dtype=tf.float32)

# 2. Setup computation graph
W = tf.Variable(tf.random_uniform([n_obs,
n_actions], 0, 0.01))
q_current = tf.matmul(obs_t_ph, W)
q_target = tf.matmul(obs_tp1_ph, W)

Q = np.zeros((env.observation_space.n,
env.action_space.n))
for episode in range(1, num_episodes + 1):
. . .
```

مرة أخرى أسفل السطر الأخير الذي أضفته مباشرةً، أدخل الكود المميز التالي. أول سطرين مكافئين للسطر المضافي الخطوة 3 التي تحسب `Qtarget`، حيث:

```
Qtarget = reward + discount_factor *
np.max(Q[state2, :])
```

يحدد السطران التاليان خسارتك، بينما يحسب السطر الأخير الإجراء الذي يزيد قيمة Q الخاصة بك:

/AtariBot/bot_4_q_network.py

```
. . .
      q_current = tf.matmul(obs_t_ph, W)
      q_target = tf.matmul(obs_tp1_ph, W)

      q_target_max = tf.reduce_max(q_target_ph,
      axis=1)
      q_target_sa = rew_ph + discount_factor *
      q_target_max
      q_current_sa = q_current[0, act_ph]
      error = tf.reduce_sum(tf.square(q_target_sa -
      q_current_sa))
      pred_act_ph = tf.argmax(q_current, 1)

      Q = np.zeros((env.observation_space.n,
      env.action_space.n))
      for episode in range(1, num_episodes + 1):
      . . .
```

بعد إعداد الخوارزمية ودالة الخسارة، حدد المُحسِّن الخاص بك:

/AtariBot/bot_4_q_network.py

```
      . . .
      error = tf.reduce_sum(tf.square(q_target_sa -
      q_current_sa))
      pred_act_ph = tf.argmax(q_current, 1)

      # 3. Setup optimization
      trainer =
      tf.train.GradientDescentOptimizer(learning_rat
      e=learning_rate)
      update_model = trainer.minimize(error)

      Q = np.zeros((env.observation_space.n,
      env.action_space.n))
      for episode in range(1, num_episodes + 1):
      . . .
```

بعد ذلك، قم بإعداد جسم حلقة اللعبة. للقيام بذلك، تمرير البيانات إلى العناصر النائبة وتجريدات Tensorflow، ستتعامل مع الحساب على وحدة معالجة الرسومات، مع إرجاع نتيجة الخوارزمية.

ابدأ بحذف جدول Q القديم والمنطق. على وجه التحديد، احذف الأسطر التي تحدد Q (قبل حلقة for مباشرة)، وnoise (في حلقة while)، وaction ، و Qtarget، و Q[state, action]. أعد تسمية state إلى obs_t وstate2 إلى obs_tp1 لمحاذاة العناصر النائبة لـ Tensor التي قمت بتعيينها مسبقًا. عند الانتهاء، ستتطابق حلقة for مع ما يلي:

/AtariBot/bot_4_q_network.py

```
. . .
# 3. Setup optimization
trainer =
tf.train.GradientDescentOptimizer(learning_rat
e=learning_rate)
update_model = trainer.minimize(error)

for episode in range(1, num_episodes + 1):
      obs_t = env.reset()
      episode_reward = 0
      while True:
            obs_tp1, reward, done, _ =
            env.step(action)

            episode_reward += reward
            obs_t = obs_tp1
            if done:
                  ...
```

مباشرة فوق حلقة for، أضف السطرين المميزين التاليين. هذه الخطوط تقوم بتهيئة جلسة Tensorflow التي تدير بدورها الموارد اللازمة لتشغيل العمليات على وحدة معالجة الرسومات. يقوم السطر الثاني بتهيئة جميع المتغيراتفي الرسم البياني الحسابي الخاص بك؛ على سبيل المثال، تهيئة الأوزان إلى 0 قبل تحديثها. بالإضافة إلى ذلك، ستقوم بتداخل الحلقة for داخل تعليمة with، لذلك ضع مسافة بادئة لحلقة for بأكملها بأربع مسافات:

/AtariBot/bot_4_q_network.py

```
. . .
trainer =
tf.train.GradientDescentOptimizer(learning_rat
e=learning_rate)
update_model = trainer.minimize(error)

with tf.Session() as session:
      session.run(tf.global_variables_initiali
zer())
```

```
for episode in range(1,num_episodes+ 1):
    obs_t = env.reset()
    ...
```

قبل السطر الذي يقرأ `obs_tp1, reward, done, _ =` `env.step(action)`، أدخل الأسطر التالية لحساب `action`. يقيم هذا الرمز العنصر النائب المقابل ويستبدل الإجراء بإجراء عشوائي ببعض الاحتمالات:

/AtariBot/bot_4_q_network.py

```
. . .
while True:
    # 4. Take step using best action or
    random action
    obs_t_oh = one_hot(obs_t, n_obs)
    action = session.run(pred_act_ph,
    feed_dict={obs_t_ph:
    obs_t_oh})[0]
    if np.random.rand(1) <
    exploration_probability(episode):
    action = env.action_space.sample()
    . . .
```

بعد السطر الذي يحتوي على `env.step(action)` ، أدخل ما يلي لتدريب الشبكة العصبية على تقدير دالة Q-value الخاصة بك:

/AtariBot/bot_4_q_network.py

```
. . .
obs_tp1, reward, done, _ = env.step(action)
# 5. Train model
obs_tp1_oh = one_hot(obs_tp1, n_obs)
q_target_val = session.run(q_target,
feed_dict={
obs_tp1_ph: obs_tp1_oh
})
session.run(update_model, feed_dict={
    obs_t_ph: obs_t_oh,
    rew_ph: reward,
    q_target_ph: q_target_val,
    act_ph: action
    })
episode_reward += reward
. . .
```

سيطابق ملفك النهائي كود المصدر هذا:

/AtariBot/bot_4_q_network.py

```
"""
Bot 4 -- Use Q-learning network to train bot
"""

from typing import List
import gym
import numpy as np
import random
import tensorflow as tf

random.seed(0)
np.random.seed(0)
tf.set_random_seed(0)

num_episodes = 4000
discount_factor = 0.99
learning_rate = 0.15
report_interval = 500
exploration_probability = lambda episode: 50. /
(episode + 10)
report = '100-ep Average: %.2f . Best 100-ep
Average: %.2f . Average:
%.2f ' \
'(Episode %d)'

def one_hot(i: int, n: int) -> np.array:
      """Implements one-hot encoding by selecting
      the ith standard basis
      vector"""
      return np.identity(n)[i].reshape((1, -1))
def print_report(rewards: List, episode: int):
      """Print rewards report for current episode
      - Average for last 100 episodes
      - Best 100-episode average across all time
      - Average for all episodes across time
      """
      print(report % (
      np.mean(rewards[-100:]),
      max([np.mean(rewards[i:i+100]) for i in
      range(len(rewards) -
      100)]),
      np.mean(rewards),
```

```
        episode))
def main():
        env = gym.make('FrozenLake-v0') # create the
        game
        env.seed(0) # make results reproducible
        rewards = []

        # 1. Setup placeholders
        n_obs, n_actions = env.observation_space.n,
        env.action_space.n
        obs_t_ph = tf.placeholder(shape=[1, n_obs],
        dtype=tf.float32)
        obs_tp1_ph = tf.placeholder(shape=[1, n_obs],
        dtype=tf.float32)
        act_ph = tf.placeholder(tf.int32, shape=())
        rew_ph = tf.placeholder(shape=(),
        dtype=tf.float32)
        q_target_ph = tf.placeholder(shape=[1,
        n_actions], dtype=tf.float32)

        # 2. Setup computation graph
        W = tf.Variable(tf.random_uniform([n_obs,
        n_actions], 0, 0.01))
        q_current = tf.matmul(obs_t_ph, W)
        q_target = tf.matmul(obs_tp1_ph, W)
        q_target_max = tf.reduce_max(q_target_ph,
        axis=1)
        q_target_sa = rew_ph + discount_factor *
        q_target_max
        q_current_sa = q_current[0, act_ph]
        error = tf.reduce_sum(tf.square(q_target_sa -
        q_current_sa))
        pred_act_ph = tf.argmax(q_current, 1)

        # 3. Setup optimization
        trainer =
        tf.train.GradientDescentOptimizer(learning_rat
        e=learning_rate)
        update_model = trainer.minimize(error)
        with tf.Session() as session:
        session.run(tf.global_variables_initializer())
        for episode in range(1, num_episodes + 1):
        obs_t = env.reset()
        episode_reward = 0
```

```
        while True:

                # 4. Take step using best action or
                random action
                obs_t_oh = one_hot(obs_t, n_obs)
                action = session.run(pred_act_ph,
                feed_dict={obs_t_ph:
                obs_t_oh})[0]
                if np.random.rand(1) <
                exploration_probability(episode):
                        action = env.action_space.sample()
                obs_tp1, reward, done, _ =
                env.step(action)

                # 5. Train model
                obs_tp1_oh = one_hot(obs_tp1, n_obs)
                q_target_val = session.run(q_target,
                feed_dict={
                obs_tp1_ph: obs_tp1_oh
                })
                session.run(update_model, feed_dict={
                obs_t_ph: obs_t_oh,
                rew_ph: reward,
                q_target_ph: q_target_val,
                act_ph: action
                })
                episode_reward += reward
                obs_t = obs_tp1
                if done:
                        rewards.append(episode_reward)
                        if episode % report_interval == 0:
                                print_report(rewards,
                                episode)
                                break
print_report(rewards, -1)
if __name__ == '__main__':
main()
```

احفظ الملف، واخرج من المحرر، وقم بتشغيل السكريبت:

```
python bot_4_q_network.py
```

سينتهي إخراجك بما يلي بالضبط:

Output

```
100-ep Average: 0.11 . Best 100-ep Average: 0.11 .
Average: 0.05
(Episode 500)
```

```
100-ep Average: 0.41 . Best 100-ep Average: 0.54 .
Average: 0.19
(Episode 1000)
100-ep Average: 0.56 . Best 100-ep Average: 0.73 .
Average: 0.31
(Episode 1500)
100-ep Average: 0.57 . Best 100-ep Average: 0.73 .
Average: 0.36
(Episode 2000)
100-ep Average: 0.65 . Best 100-ep Average: 0.73 .
Average: 0.41
(Episode 2500)
100-ep Average: 0.65 . Best 100-ep Average: 0.73 .
Average: 0.43
(Episode 3000)
100-ep Average: 0.69 . Best 100-ep Average: 0.73 .
Average: 0.46
(Episode 3500)
100-ep Average: 0.77 . Best 100-ep Average: 0.79 .
Average: 0.48
(Episode 4000)
100-ep Average: 0.77 . Best 100-ep Average: 0.79 .
Average: 0.48
(Episode -1)
```

لقد دربت الآن وكيل Q-Learning الأول الخاص بك. للحصول على لعبة بسيطة مثل FrozenLake، تطلب وكيل Q-Learning العميق 4000 حلقة للتدريب. تخيل لو كانت اللعبة أكثر تعقيدًا بكثير. كم عدد عينات التدريب التي تتطلب التدريب؟ كما اتضح، يمكن أن يطلب الوكيل ملايين العينات. يُشار إلى عدد العينات المطلوبة باسم تعقيد العينة sample complexity، وهو مفهوم يتم استكشافه بمزيد من التفصيلفي القسم التالي.

فهم موازنات التحيز -التباين

بشكل عام، يتعارض تعقيد العينة مع تعقيد النموذجفي التعلم الآلي:

1. **تعقيد النموذج**: يريد المرء نموذجًا معقدًا بدرجة كافية لحل مشكلته. على سبيل المثال، نموذج بسيط مثل الخط ليس معقدًا بدرجة كافية للتنبؤ بمسار السيارة.
2. **تعقيد العينة:** قد يرغب المرءفي نموذج لا يتطلب العديد من العينات. قد يكون هذا بسبب محدودية وصولهم إلى البيانات المصنفة، وكمية غير كافية من قوة الحوسبة، وذاكرة محدودة، وما إلى ذلك.

لنفترض أن لدينا نموذجين، أحدهما بسيط والآخر معقد للغاية. لكي يحصل كلا النموذجين على نفس الأداء، يخبرنا التحيز–التباين أن النموذج المعقد للغاية سيحتاج إلى مزيد من العينات بشكل كبير للتدريب. مثال على ذلك: يتطلب وكيل Q-Learning القائم على الشبكة العصبية 4000 حلقة لحل FrozenLake. تؤدي إضافة طبقة ثانية إلى وكيل الشبكة العصبية إلى مضاعفة عدد حلقات التدريب اللازمة أربع مرات. مع تزايد تعقيد الشبكات العصبية، ينمو هذا الانقسام فقط. للحفاظ على نفس معدل الخطأ، تؤدي زيادة تعقيد النموذج إلى زيادة تعقيد العينة بشكل كبير. وبالمثل، فإن تقليل تعقيد العينة يقلل من تعقيد النموذج. وبالتالي، لا يمكننا تعظيم تعقيد النموذج وتقليل تعقيد العينة وفقًا لرغبة قلوبنا.

ومع ذلك، يمكننا الاستفادة من معرفتنا بهذه الموازنة. للحصول على تفسير مرئي للرياضيات وراء تحلل التحيز–التباين، راجع فهم موازنة التحيز–التباين. على مستوى عالٍ، فإن تحلل التحيز–التباين هو تفصيل لـ "الخطأ الحقيقي" إلى عنصرين: التحيز والتباين. نشير إلى "الخطأ الحقيقي" على أنه الخطأ التربيعي المتوسط (MSE)، وهو الفرق المتوقع بين تسمياتنا المتوقعة والتسميات الحقيقية. فيما يلي مخطط يوضح تغيير "الخطأ الحقيقي" مع زيادة تعقيد النموذج:

منحنى الخطأ التربيعي المتوسط

الخطوة 5 - بناء وكيل المربعات الصغرى لـ Frozen Lake

طريقة المربعات الصغرى least squares method، والمعروفة أيضًا باسم الانحدار الخطي linear regression، هي وسيلة لتحليل الانحدار تستخدم على نطاق واسعفي مجالات الرياضيات وعلم البيانات.في التعلم الآلي، غالبًا ما تُستخدم للعثور على النموذج الخطي الأمثل لمعلمتين أو مجموعتي بيانات.

في الخطوة 4، قمت ببناء شبكة عصبية لحساب قيم Q. بدلاً من الشبكة العصبية، ستستخدمفي هذه الخطوة انحدار ريدج ridge regression، وهو متغير من المربعات الصغرى، لحساب هذا

المتجه لقيم Q. الأمل هو أنه مع وجود نموذج غير معقد مثل المربعات الصغرى، فإن حل اللعبة سيتطلب حلقات تدريب أقل.

ابدأ بتكرار البرنامج النصي من الخطوة 3:

```
cp bot_3_q_table.py bot_5_ls.py
```

افتح الملف الجديد:

```
nano bot_5_ls.py
```

مرة أخرى، قم بتحديث التعليق أعلى الملف الذي يصف ما سيفعله هذا السكريبت:

/AtariBot/bot_4_q_network.py

```
"""
Bot 5 -- Build least squares q-learning agent for
FrozenLake
"""
. . .
```

قبل حظر الاستيراد بالقرب من الجزء العلوي من الملف، أضف عمليتي استيراد إضافيتين للتحقق من النوع:

/AtariBot/bot_5_ls.py

```
. . .
from typing import Tuple
from typing import Callable
from typing import List
import gym
. . .
```

في قائمة المعامِلات الفائقة، أضف معلمة فائقة أخرى، `w_lr`، للتحكمفي معدل تعلم دالة Q الثانية. بالإضافة إلى ذلك، قم بتحديث عدد الحلقات إلى 5000 وعامل الخصم إلى `0.85`. من خلال تغيير كل من المعلمات الفائقة `num_episodes` و `discount_factor` إلى قيم أكبر، سيتمكن الوكيل من إصدار أداء أقوى:

/AtariBot/bot_5_ls.py

```
. . .
num_episodes = 5000
discount_factor = 0.85
learning_rate = 0.9
w_lr = 0.5
report_interval = 500
```

```
. . .
```

قبل دالة `print_report` الخاصة بك، أضف دالة الترتيب الأعلى التالية. تقوم بإرجاع lambda ـ دالة مجهولة — التي يلخص النموذج:

/AtariBot/bot_5_ls.py

```
. . .
report_interval = 500
report = '100-ep Average: %.2f . Best 100-ep
Average: %.2f . Average:
%.2f ' \
'(Episode %d)'

def makeQ(model: np.array) -> Callable[[np.array],
np.array]:
"""Returns a Q-function, which takes state ->
distribution over
actions"""
return lambda X: X.dot(model)

def print_report(rewards: List, episode: int):
. . .
```

بعد `makeQ`، أضف دالة أخرى، قم بالتهيئة `initialize`، والتي تهيئ النموذج باستخدام القيم الموزعة بشكل طبيعي:

/AtariBot/bot_5_ls.py

```
. . .
def makeQ(model: np.array) -> Callable[[np.array],
np.array]:
"""Returns a Q-function, which takes state ->
distribution over
actions"""
return lambda X: X.dot(model)

def initialize(shape: Tuple):
"""Initialize model"""
W = np.random.normal(0.0, 0.1, shape)
Q = makeQ(W)
return W, Q

def print_report(rewards: List, episode: int):
. . .
```

بعد كتلة التهيئة `initialize`، أضف طريقة `train` التي تحسب حل الشكل المغلق لانحدار ريدج، ثم تزن النموذج القديم بالنموذج الجديد. تقوم بإرجاع كل من النموذج ودالة Q المستخرجة:

/AtariBot/bot_5_ls.py

```
. . .
def initialize(shape: Tuple):
...
return W, Q

def train(X: np.array, y: np.array, W: np.array) ->
Tuple[np.array,
Callable]:
"""Train the model, using solution to ridge
regression"""
I = np.eye(X.shape[1])
newW = np.linalg.inv(X.T.dot(X) + 10e-4 _
I).dot(X.T.dot(y))
W = w_lr _ newW + (1 - w_lr) \* W
Q = makeQ(W)
return W, Q

def print_report(rewards: List, episode: int):
. . .
```

بعد التدريب، أضف دالة أخيرة، `one_hot`، لإجراء ترميز واحد ساخن لحالاتك وإجراءاتك:

/AtariBot/bot_5_ls.py

```
. . .
def train(X: np.array, y: np.array, W: np.array) ->
Tuple[np.array,
Callable]:
...
return W, Q

def one_hot(i: int, n: int) -> np.array:
"""Implements one-hot encoding by selecting the ith
standard basis
vector"""
return np.identity(n)[i]

def print_report(rewards: List, episode: int):
. . .
```

بعد ذلك، سوف تحتاج إلى تعديل منطق التدريب.في النص السابق الذي كتبته، تم تحديث جدول Q كل تكرار. ومع ذلك، فإن هذا السكريبت سيجمع العينات والتسميات في كل مرة خطوة ويدرب نموذجًا جديدًا كل 10 خطوات. بالإضافة إلى ذلك، بدلاً من الاحتفاظ بجدول Q أو شبكة عصبية، سيستخدم نموذج المربعات الصغرى للتنبؤ بقيم Q.

انتقل إلى دالة `main` واستبدل تعريف جدول Q (`Q = np.zeros(...)`) بما يلي:

/AtariBot/bot_5_ls.py

```
. . .
def main():
...
rewards = []

      n_obs, n_actions = env.observation_space.n,
      env.action_space.n
      W, Q = initialize((n_obs, n_actions))
      states, labels = [], []
      for episode in range(1, num_episodes + 1):
      . . .
```

قم بالتمرير لأسفل قبل حلقة `for`. أسفل هذا مباشرة، أضف الأسطر التالية التي تعيد تعيين قوائم `states` و`labels` إذا كان هناك الكثير من المعلومات المخزنة:

/AtariBot/bot_5_ls.py

```
. . .
def main():
...
for episode in range(1, num_episodes + 1):
if len(states) >= 10000:
states, labels = [], []
. . .
```

قم بتعديل الخط مباشرة بعد هذا الخط، والذي يحدد `state =env.reset()` ، بحيث يصبح كما يلي. سيؤدي هذا إلى ترميز الحالة على الفور، حيث ستتطلب جميع استخداماتها متجهًا واحدًا ساخنًا:

/AtariBot/bot_5_ls.py

```
. . .
for episode in range(1, num_episodes + 1):
if len(states) >= 10000:
states, labels = [], []
state = one_hot(env.reset(), n_obs)
. . .
```

قبل السطر الأولفي حلقة اللعبة الرئيسية الخاصة بك، قم بتعديل قائمة states:

/AtariBot/bot_5_ls.py

```
. . .
for episode in range(1, num_episodes + 1):
...
episode_reward = 0
while True:
states.append(state)
noise = np.random.random((1, env.action_space.n)) /
(episode\*\*2.)
. . .
```

قم بتحديث حساب action، وتقليل احتمالية الضوضاء، وتعديل تقييم دالة Q:

/AtariBot/bot_5_ls.py

```
. . .
while True:
states.append(state)
noise = np.random.random((1, n*actions)) / episode
action = np.argmax(Q(state) + noise)
state2, reward, done, * = env.step(action)
. . .
```

أضف إصدارًا واحدًا ساخنًا من state2 وقم بتعديل استدعاء دالة Qفي تعريفك لـ Qtarget على النحو التالي:

/AtariBot/bot_5_ls.py

```
. . .
while True:
...
state2, reward, done, \_ = env.step(action)
      state2 = one_hot(state2, n_obs)
      Qtarget = reward + discount_factor *
      np.max(Q(state2))
      . . .
```

احذف السطر الذي يحدّث Q[state,action] = ... واستبدله بالأسطر التالية. يأخذ هذا الرمز إخراج النموذج الحالي ويقوم فقط بتحديث القيمةفي هذا الإخراج الذي يتوافق مع الإجراء الحالي المتخذ. نتيجة لذلك، لا تتسبب قيم Q للإجراءات الأخرىفي الخسارة:

/AtariBot/bot_5_ls.py

```
. . .
```

```
state2 = one_hot(state2, n_obs)
Qtarget = reward + discount_factor _
np.max(Q(state2))
label = Q(state)
label[action] = (1 - learning_rate) _ label[action]
+ learning_rate \*
Qtarget
labels.append(label)
      episode_reward += reward
            . . .
```

مباشرةً بعد `state = state2`، أضف تحديثًا دوريًا للنموذج.

يؤدي هذا إلى تدريب نموذجك كل 10 خطوات زمنية:

/AtariBot/bot_5_ls.py

```
. . .
state = state2
if len(states) % 10 == 0:
W, Q = train(np.array(states), np.array(labels), W)
if done:
. . .
```

تأكد من أن الكود الخاص بك يطابق ما يلي:

/AtariBot_5_ls.py

```
"""
Bot 5 -- Build least squares q-learning agent for
FrozenLake
"""

from typing import Tuple
from typing import Callable
from typing import List
import gym
import numpy as np
import random
random.seed(0) # make results reproducible
np.random.seed(0) # make results reproducible

num_episodes = 5000
discount_factor = 0.85
learning_rate = 0.9
w_lr = 0.5
report_interval = 500
```

```
report = '100-ep Average: %.2f . Best 100-ep
Average: %.2f . Average:
%.2f ' \
'(Episode %d)'

def makeQ(model: np.array) -> Callable[[np.array],
np.array]:
      """Returns a Q-function, which takes state ->
      distribution over
      actions"""
      return lambda X: X.dot(model)

def initialize(shape: Tuple):
      """Initialize model"""
      W = np.random.normal(0.0, 0.1, shape)
      Q = makeQ(W)
      return W, Q

def train(X: np.array, y: np.array, W: np.array) ->
Tuple[np.array,
      Callable]:
      """Train the model, using solution to ridge
      regression"""
      I = np.eye(X.shape[1])
      newW = np.linalg.inv(X.T.dot(X) + 10e-4 *
      I).dot(X.T.dot(y))
      W = w_lr * newW + (1 - w_lr) * W
      Q = makeQ(W)
      return W, Q

def one_hot(i: int, n: int) -> np.array:
      """Implements one-hot encoding by selecting
      the ith standard basis
      vector"""
      return np.identity(n)[i]

def print_report(rewards: List, episode: int):
      """Print rewards report for current episode
      - Average for last 100 episodes
      - Best 100-episode average across all time
      - Average for all episodes across time
      """
      print(report % (
      np.mean(rewards[-100:]),
```

```
      max([np.mean(rewards[i:i+100]) for i in
      range(len(rewards) -
      100)]),
      np.mean(rewards),
      episode))

def main():
      env = gym.make('FrozenLake-v0') # create the
      game
      env.seed(0) # make results reproducible
      rewards = []

      n_obs, n_actions = env.observation_space.n,
      env.action_space.n
      W, Q = initialize((n_obs, n_actions))
      states, labels = [], []

      for episode in range(1, num_episodes + 1):
            if len(states) >= 10000:
                  states, labels = [], []
            state = one_hot(env.reset(), n_obs)
            episode_reward = 0

            while True:
                  states.append(state)
                  noise = np.random.random((1,
                  n_actions)) / episode
                  action = np.argmax(Q(state) +
                  noise)

                  state2, reward, done, _ =
                  env.step(action)
                  state2 = one_hot(state2, n_obs)
                  Qtarget = reward + discount_factor
                  * np.max(Q(state2))
                  label = Q(state)
                  label[action] = (1 - learning_rate)
                  * label[action] + \
                  learning_rate * Qtarget
                  labels.append(label)

                  episode_reward += reward
                  state = state2

                  if len(states) % 10 == 0:
```

```
                    W, Q =
                    train(np.array(states),
                    np.array(labels), W)

              if done:
                    rewards.append(episode_reward
                    )
                    if episode % report_interval
                    == 0:
                          print_report(rewards,
                    episode)
              break
    print_report(rewards, -1)
    if __name__ == '__main__':
          main()
```

بعد ذلك، احفظ الملف، واخرج من المحرر، وقم بتشغيل السكريبت:

```
python bot_5_ls.py
```

سينتج هذا ما يلي:

Output

```
100-ep Average: 0.17 . Best 100-ep Average: 0.17 .
Average: 0.09
(Episode 500)
100-ep Average: 0.11 . Best 100-ep Average: 0.24 .
Average: 0.10
(Episode 1000)
100-ep Average: 0.08 . Best 100-ep Average: 0.24 .
Average: 0.10
(Episode 1500)
100-ep Average: 0.24 . Best 100-ep Average: 0.25 .
Average: 0.11
(Episode 2000)
100-ep Average: 0.32 . Best 100-ep Average: 0.31 .
Average: 0.14
(Episode 2500)
100-ep Average: 0.35 . Best 100-ep Average: 0.38 .
Average: 0.16
(Episode 3000)
100-ep Average: 0.59 . Best 100-ep Average: 0.62 .
Average: 0.22
(Episode 3500)
```

```
100-ep Average: 0.66 . Best 100-ep Average: 0.66 .
Average: 0.26
(Episode 4000)
100-ep Average: 0.60 . Best 100-ep Average: 0.72 .
Average: 0.30
(Episode 4500)
100-ep Average: 0.75 . Best 100-ep Average: 0.82 .
Average: 0.34
(Episode 5000)
100-ep Average: 0.75 . Best 100-ep Average: 0.82 .
Average: 0.34
(Episode -1)
```

تذكر أنه وفقًا لصفحة Gym FrozenLake ، فإن "حل" اللعبة يعني الوصول إلى معدل 100 حلقة يبلغ 0.78. هنا يحقق الوكيل 0.82في المتوسط، مما يعني أنه كان قادرًا على حل اللعبةفي 5000 حلقة. على الرغم من أن هذا لا يحل اللعبةفي حلقات أقل، إلا أن طريقة المربعات الصغرى الأساسية هذه لا تزال قادرة على حل لعبة بسيطة بنفس عدد حلقات التدريب تقريبًا. على الرغم من أن شبكاتك العصبية قد تزداد تعقيدًا، فقد أظهرت أن النماذج البسيطة مناسبة لـ FrozenLake.

باستخدام ذلك، تكون قد استكشفت ثلاثة عوامل Q-Learning: أحدها يستخدم جدول Q، والآخر يستخدم شبكة عصبية، والثالث يستخدم المربعات الصغرى. بعد ذلك، ستقوم ببناء عامل تعلم معزز عميق للعبة أكثر تعقيدًا: Space Invaders.

الخطوة 6 - إنشاء عامل Q-Learning عميق لغزاة الفضاء

لنفترض أنك ضبطت تعقيد نموذج خوارزمية Q-Learning السابقة وعينة التعقيد بشكل مثالي، بغض النظر عما إذاكنت قد اخترت شبكة عصبية أو طريقة المربعات الصغرى.كما اتضح، لا يزال أداء عامل Q-Learning غير الذكي هذا ضعيفًافي الألعاب الأكثر تعقيدًا، حتى مع وجود عدد كبير بشكل خاص من حلقات التدريب. سيغطي هذا القسم طريقتين من شأنها تحسين الأداء، ثم ستختبر وكيلًا تم تدريبه باستخدام هذه الأساليب.

تم تطوير أول وكيل للأغراض العامة قادر على تكييف سلوكه باستمرار دون أي تدخل بشري من قبل الباحثينفي DeepMind، الذين قاموا أيضًا بتدريب وكيلهم على لعب مجموعة متنوعة من ألعاب Atari. أقرت مقالة التعلم العميق Q-Learning (DQN) الأصلية لـ DeepMind مسألتين مهمتين:

1. **الحالات المترابطة:** خذ حالة لعبتانفي الوقت 0، والذي سنسميه s0. لنفترض أننا قمنا بتحديث Q (s0)، وفقًا للقواعد التي استخلصناها سابقًا. الآن، خذ الحالةفي الوقت

1، والتي نسميها s1، وقم بتحديث Q (s1) وفقًا لنفس القواعد. لاحظ أن حالة اللعبة في الوقت 0 تشبه إلى حد كبير حالتهافي الوقت 1.في Space Invaders، على سبيل المثال، ربما تحركت الكائنات الفضائية بمقدار بكسل واحد لكل منهم. يقال بإيجاز أكبر، s0 وs1 متشابهان للغاية. وبالمثل، نتوقع أيضًا أن تكون Q (s0) و Q (s1) متشابهة جدًا، لذا فإن تحديث أحدهما يؤثر على الآخر. هذا يؤدي إلى تقلب قيم Q، حيث قد يكون تحديثًا لـ Q (S0)في الواقع مواجهة التحديث إلى Q (S1). بشكل أكثر رسمية، ترتبط s0 وs1. نظرًا لأن دالة Q حتمية، فإن Q (s1) مرتبطة بـ Q (s0).

2. **عدم استقرار دالة Q**: تذكر أن دالة Q هي النموذج الذي نقوم بتدريبه ومصدر تسمياتنا. لنفترض أن تسمياتنا عبارة عن قيم تم اختيارها عشوائيًا تمثل حقًا التوزيع، L.في كل مرة نقوم فيها بتحديث Q، نقوم بتغيير L، مما يعني أن نموذجنا يحاول معرفة هدف متحرك. هذه مشكلة، لأن النماذج التي نستخدمها تفترض توزيعًا ثابتًا.

لمواجهة الحالات المرتبطة ودالة Q غير المستقرة:

1. يمكن للمرء الاحتفاظ بقائمة من الحالات تسمى المخزن المؤقت لإعادة التشغيل replay buffer.في كل خطوة زمنية، تقوم بإضافة حالة اللعبة التي تلاحظها إلى المخزن المؤقت لإعادة التشغيل هذا. يمكنك أيضًا أخذ عينة عشوائية من الحالات من هذه القائمة، والتدرب على تلك الحالات.
2. قام الفريقفي DeepMind بتكرار Q(s, a) .واحد يسمى(Q_current(s, a ، وهي دالة Q التي تقوم بتحديثها. أنت بحاجة إلى دالة Q أخرى للحالات اللاحقة، Q_target(s', a')، والتي لن تقوم بتحديثها. يتم استخدام استدعاء Q_target(s', a') لإنشاء تسمياتك. من خلال فصل Q_current عن Q_target واصلاح الأخير، يمكنك إصلاح التوزيع الذي يتم أخذ عينات من تسمياتك منه. بعد ذلك، يمكن أن يقضي نموذج التعلم العميق الخاص بك فترة قصيرةفي تعلم هذا التوزيع. بعد فترة من الوقت، تقوم بإعادة تكرار Q_current من أجل Q_target جديد.

لن تقوم بتنفيذها بنفسك، لكنك ستقوم بتحميل نماذج مُدربة مسبقًا تم تدريبها باستخدام هذه الحلول. للقيام بذلك، أنشئ دليلًا جديدًا حيث ستخزن معلمات هذه النماذج:

```
mkdir models
```

ثم استخدم أداة `wget` لتنزيل معلمات نموذج Space Invaders الذي تم اختباره مسبقًا:

```
Wget http://models.tensorpack.com/OpenAIGym/SpaceInvaders-v0.tfmodel -
P models
```

بعد ذلك، قم بتنزيل سكريبت بايثون الذي يحدد النموذج المرتبط بالمعلمات التي قمت بتنزيلها للتو. لاحظ أن هذا النموذج الذي تم اختباره مسبقًا يحتوي على قيدين على المدخلات التي يجب وضعهافي الاعتبار:

- يجب تصغير حجم الحالات أو تقليل حجمها إلى 84 × 84.
- يتكون الإدخال من أربع حالات مكدسة.

سوف نتناول هذه القيود بمزيد من التفصيل فيما بعد.في الوقت الحالي، قم بتنزيل السكريبت عن طريق كتابة:

```
wget https://github.com/alvinwan/bots-for-atarigames/
raw/master/src/bot_6_a3c.py
```

ستقوم الآن بتشغيل وكيل Space Invaders الذي تم اختباره مسبقًا لمعرفة كيفية أدائه. على عكس برامج التتبع القليلة الماضية التي استخدمناها، ستكتب هذا النص من البداية.

قم بإنشاء سكريبت جديد:

```
nano bot_6_dqn.py
```

ابدأ هذا السكريبت بإضافة تعليق رئيسي، واستيراد الأدوات المساعدة اللازمة، وبدء حلقة اللعبة الرئيسية:

/AtariBot/bot_6_dqn.py

```
"""
Bot 6 - Fully featured deep q-learning network.
"""
import cv2
import gym
import numpy as np
import random
import tensorflow as tf
from bot_6_a3c import a3c_model
def main():
if **name** == '**main**':
main()
```

مباشرة بعد الاستيرادات الخاصة بك، قم بتعيين بذور عشوائية لجعل نتائجك قابلة للتكرار. أيضًا، حدد معلمة فائقة num_episodes والتي ستخبر السكريبت بعدد الحلقات التي سيتم تشغيل الوكيل من أجلها:

/AtariBot/bot_6_dqn.py

```
. . .
import tensorflow as tf
```

```
from bot_6_a3c import a3c_model
random.seed(0) # make results reproducible
tf.set_random_seed(0)

num_episodes = 10

def main():
. . .
```

سطرين بعد الإعلان عن `num_episodes`، حدد دالة `downsample` التي تختزل كل الصور بحجم 84 × 84. ستختزل كل الصور قبل تمريرها إلى الشبكة العصبية المدربة مسبقاً، حيث تم تدريب النموذج قبل التدريب على 84 × 84 صورة:

/AtariBot/bot_6_dqn.py

```
. . .
num_episodes = 10

def downsample(state):
return cv2.resize(state, (84, 84),
interpolation=cv2.INTER_LINEAR)[None]

def main():
. . .
```

قم بإنشاء بيئة اللعبةفي بداية دالتك `main` وزرع البذورفي البيئة بحيث تكون النتائج قابلة للتكرار:

/AtariBot/bot_6_dqn.py

```
. . .
def main():

env = gym.make('SpaceInvaders-v0') # create the
game
env.seed(0) # make results reproducible
. . .
```

مباشرة بعد بذرة البيئة، قم بتهيئة قائمة فارغة للاحتفاظ بالمكافآت:

/AtariBot/bot_6_dqn.py

```
. . .
def main():
env = gym.make('SpaceInvaders-v0')# create the game
env.seed(0) # make results reproducible
rewards = []
. . .
```

قم بتهيئة النموذج الذي تم اختباره مسبقًا باستخدام معلمات النموذج التي تم اختبارها مسبقًا التي قمت بتنزيلهافي بداية هذه الخطوة:

/AtariBot/bot_6_dqn.py

```
. . .
def main():
env = gym.make('SpaceInvaders-v0') # create the
game
env.seed(0) # make results reproducible
rewards = []
model = a3c_model(load='models/SpaceInvaders-
v0.tfmodel')
. . .
```

بعد ذلك، أضف بعض الأسطر التي تخبر السكريبت بالتكرار لعدد مرات الحلقات لحساب متوسط الأداء وتهيئة مكافأة كل حلقة إلى 0. بالإضافة إلى ذلك، أضف سطرًا لإعادة تعيين البيئة (`env.reset()`) ، وجمع الحالة الأولية الجديدةفي العملية، اختزل هذه الحالة الأولية باستخدام `downsample()`، وابدأ حلقة اللعبة باستخدام حلقة `while`:

/AtariBot/bot_6_dqn.py

```
. . .
def main():
env = gym.make('SpaceInvaders-v0') # create the
game
env.seed(0) # make results reproducible
rewards = []
model = a3c*model(load='models/SpaceInvaders-
v0.tfmodel')
for * in range(num_episodes):
episode_reward = 0
states = [downsample(env.reset())]
while True:
. . .
```

بدلاً من قبول حالة واحدةفي كل مرة، تقبل الشبكة العصبية الجديدة أربع حالاتفي المرة الواحدة. نتيجة لذلك، يجب عليك الانتظار حتى تحتوي قائمة الحالات على أربع حالات على الأقل قبل تطبيق النموذج الذي تم اختباره مسبقًا. أضف الأسطر التالية أسفل السطر قراءة بينما `while True:`. يخبر هؤلاء الوكيل أن يتخذ إجراءً عشوائيًا إذا كان هناك أقل من أربع حالات أو أن يسلسل الحالات ويمررها إلى النموذج قبل التدريب إذا كان هناك أربع حالات على الأقل:

/AtariBot/bot_6_dqn.py

```
. . .
```

```
while True:
if len(states) < 4:
action = env.action_space.sample()
else:
frames = np.concatenate(states[-4:], axis=3)
action = np.argmax(model([frames]))
. . .
```

ثم اتخذ إجراءً وقم بتحديث البيانات ذات الصلة. أضف نسخة مختزلة من الحالة الملحوظة، وقم بتحديث المكافأة لهذه الحلقة:

/AtariBot/bot_6_dqn.py

```
. . .
while True:
...
action = np.argmax(model([frames]))
state, reward, done, _ = env.step(action)
states.append(downsample(state))
episode_reward += reward
. . .
```

بعد ذلك، أضف الأسطر التالية التي تتحقق مما إذا كانت الحلقة قد تمت أم لا، وإذا كانت كذلك، اطبع إجمالي المكافأة للحلقة وقم بتعديل قائمة جميع النتائج وكسر حلقة الوقت مبكرًا:

/AtariBot/bot_6_dqn.py

```
. . .
while True:
      ...
      episode_reward += reward

      if done:
            print('Reward: %d' % episode_reward)
            rewards.append(episode_reward)
            break
            . . .
```

خارج حلقات `while` و`for`، اطبع متوسط المكافأة. ضع هذا في نهاية دالتك `main`:

/AtariBot/bot_6_dqn.py

```
def main():
...
break
print('Average reward: %.2f' % (sum(rewards) /
len(rewards)))
```

تأكد من أن ملفك يطابق ما يلي:

/AtariBot/bot_6_dqn.py

```
"""
Bot 6 - Fully featured deep q-learning network.
"""

import cv2
import gym
import numpy as np
import random
import tensorflow as tf
from bot_6_a3c import a3c_model

random.seed(0) # make results reproducible
tf.set_random_seed(0)
num_episodes = 10

def downsample(state):
      return cv2.resize(state, (84, 84),
      interpolation=cv2.INTER_LINEAR)
      [None]


def main():
      env = gym.make('SpaceInvaders-v0') # create
      the game
      env.seed(0) # make results reproducible
      rewards = []

      model = a3c_model(load='models/SpaceInvaders-
      v0.tfmodel')

      for _ in range(num_episodes):
            episode_reward = 0
            states = [downsample(env.reset())]
            while True:
                  if len(states) < 4:
                        action =
                  env.action_space.sample()
                  else:
                        frames =
                        np.concatenate(states[-4:],
                        axis=3)
```

```
                        action =
                        np.argmax(model([frames]))
                    state, reward, done, _ =
                    env.step(action)
                    states.append(downsample(state))
                    episode_reward += reward
                    if done:
                        print('Reward: %d' %
                        episode_reward)
                        rewards.append(episode_reward
                        )
                        break
print('Average reward: %.2f' % (sum(rewards) /
len(rewards)))
if __name__ == '__main__':
main()
```

احفظ الملف واخرج من المحرر. بعد ذلك، قم بتشغيل السكريبت:

```
python bot_6_dqn.py
```

سينتهي إخراجك بما يلي:

Output

```
. . .
Reward: 1230
Reward: 4510
Reward: 1860
Reward: 2555
Reward: 515
Reward: 1830
Reward: 4100
Reward: 4350
Reward: 1705
Reward: 4905
Average reward: 2756.00
```

قارن هذا بالنتيجة من السيناريو الأول، حيث قمت بتشغيل وكيل عشوائي لـ Space Invaders. كان متوسط المكافأةفي هذه الحالة حوالي 150 فقط، مما يعني أن هذه النتيجة أفضل بأكثر من عشرين مرة. ومع ذلك، فقد قمت بتشغيل الكود الخاص بك لثلاث حلقات فقط، لأنها بطيئة إلى حد ما، ومتوسط ثلاث حلقات ليس مقياسًا موثوقًا به. تشغيل هذا على مدى 10 حلقات، بمتوسط 2756؛ أكثر من 100 حلقة، المتوسط حوالي 2500. فقط بهذه المعدلات يمكنك أن

تستنتج بشكل مريح أن وكيلك يؤدي بالفعل ترتيبًا أفضل من حيث الحجم، وأن لديك الآن وكيل يلعب دور Space Invaders بشكل معقول.

ومع ذلك، تذكر المشكلة التي أثيرتفي القسم السابق بخصوص تعقيد العينة. كما اتضح، يأخذ عميل Space Invaders ملايين العينات للتدريب.في الواقع، طلب هذا الوكيل 24 ساعة على أربع وحدات معالجة رسومات Titan X للتدريب حتى هذا المستوى الحالي؛ وبعبارة أخرى، فقد تطلب الأمر قدرًا كبيرًا من الحوسبة لتدريبها بشكل مناسب. هل يمكنك تدريب عامل مشابه عالي الأداء بعينات أقل بكثير؟ يجب أن تزودك الخطوات السابقة بالمعرفة الكافية لبدء استكشاف هذا السؤال. باستخدام نماذج أبسط بكثير وموازنات التحيز_التباين ، قد يكون ذلك ممكنًا.

الاستنتاج

في هذا البرنامج التعليمي، قمت ببناء العديد من برامج البوت للألعاب واستكشفت مفهومًا أساسيًافي التعلم الآلي يسمى التحيز_التباين. السؤال التالي الطبيعي هو: هل يمكنك إنشاء بوت لألعاب أكثر تعقيدًا، مثل StarCraft 2؟ كما اتضح، هذا سؤال بحث معلق، مكمل بأدوات مفتوحة المصدر من متعاونين عبر Google وDeepMind وBlizzard. إذا كانت هذه مشكلات تهمك، فراجع الدعوات المفتوحة للبحثفي OpenAI، للتعرف على المشكلات الحالية.

الوجبات الرئيسية من هذا البرنامج التعليمي هي موازنة التحيز_التباين. الأمر متروك لممارس التعلم الآلي للنظرفي تأثيرات تعقيد النموذج.في حين أنه من الممكن الاستفادة من النماذج المعقدة للغاية والطبقة على كميات زائدة من الحوسبة والعينات والوقت، فإن تقليل تعقيد النموذج يمكن أن يقلل بشكل كبير من الموارد المطلوبة.