Tensorflow.JS

Tensorflow.js et le WebAssembly

Qu'est-ce que l'IA et le WebAssembly ?

L'arrivée du WebAssembly dans l'IA :

On peut dire que le WebAssembly a profondément pénétré le monde de l'IA le jour où TensorFlow.js a officiellement annoncé son support pour le WebAssembly en tant que backend alternatif à WebGL le 11 mars 2020.

Comment le WebAssembly peut-il aider l'IA ?

Le WebAssembly, et plus particulièrement l'IA dans les navigateurs, répond à différentes problématiques :

Rapidité :

Exécution rapide sur CPU.

Très utile pour les machines qui ne disposent pas d'un GPU performant ou même d'un GPU et donc pas d'une exécution via WebGL.

À noter que le WebAssembly est disponible sur la très grande majorité des navigateurs web.

Partage :

La possibilité d'exécuter l'apprentissage automatique dans les navigateurs, sans aucune installation supplémentaire.

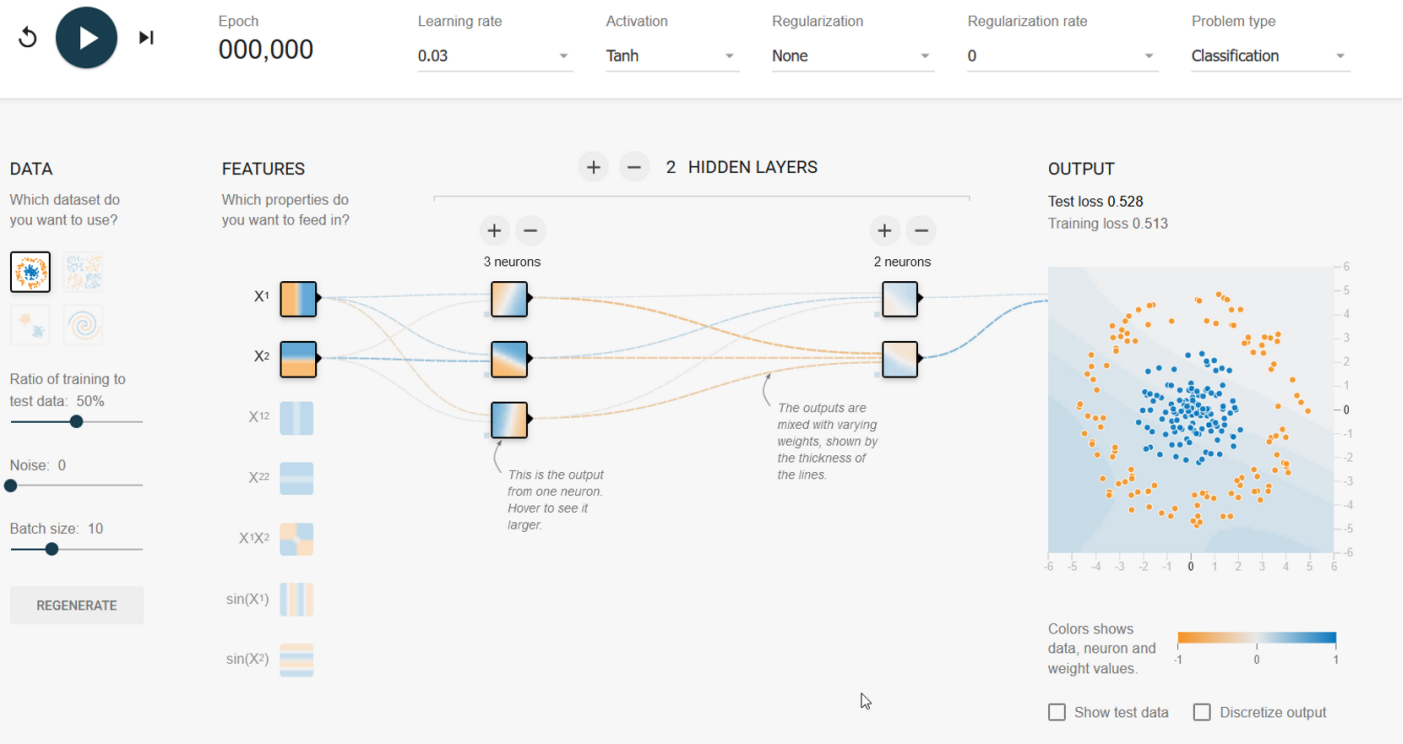
Les modèles et les applications écrites sont faciles à partager sur le Web, ce qui réduit la barrière à l'entrée de l'apprentissage automatique.

Ceci est particulièrement important pour les cas d'utilisation éducative et pour augmenter la diversité des contributeurs dans le domaine.

Interactivité :

Du point de vue de l'apprentissage automatique, la nature interactive des navigateurs Web et les capacités polyvalentes des API Web ouvrent la voie à un large éventail de nouvelles applications d'apprentissage automatique centrées sur l'utilisateur, qui peuvent servir à la fois à l'enseignement et à la recherche.

Les visualisations de réseaux neuronaux peuvent ainsi être représentés ainsi :



> “ https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=3,2&seed=0.47115&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false”

Calcul local et sécurité :

L'accès normalisé à divers composants matériels du dispositif, tels que la caméra web, le microphone et l'accéléromètre, dans le navigateur, permet une intégration simple entre les modèles de ML et les données des capteurs.

Un résultat important de cette intégration est que les données de l'utilisateur peuvent rester sur le dispositif et préserver sa vie privée, ce qui permet des applications dans les domaines de la médecine, de l'accessibilité et du ML personnalisé.

Par exemple, les personnes souffrant de troubles de la parole peuvent utiliser leur téléphone pour recueillir des échantillons audio afin d'entraîner un modèle personnalisé dans le navigateur.

Une autre technologie pourrait permettre aux dispositifs d'entraîner de manière collaborative un modèle centralisé tout en conservant les données sensibles sur le dispositif.

Les navigateurs sont une plateforme naturelle pour ce type d'application.

Présentation de TensorFlow.js

Historique

TensorFlow a été développé en 2011 chez Google comme leur bibliothèque propriétaire pour les applications d'apprentissage automatique/apprentissage profond. Elle a été mise en open source en 2015 sous la licence Apache.

TensorFlow est construit en C++, ce qui permet au code de s'exécuter à un niveau très bas. Il est lié à différents langages tels que Python, R et Java, ce qui permet son utilisation dans ces langages.

Traditionnellement, en JavaScript, le ML/DL (Machine Learning / Deep Learning) était réalisé en utilisant une API. Une API était créée à l'aide d'un cadre quelconque, et le modèle était déployé sur le serveur. Le client envoie une requête en JavaScript pour obtenir les résultats du serveur.

En 2017, un projet appelé Deeplearn.js est apparu, qui visait à permettre le ML/DL en JavaScript, sans le besoin d'une API. Cependant, la question de la vitesse se posait. Le code JavaScript ne pouvait pas s'exécuter sur GPU. Pour résoudre ce problème, WebGL a été introduit. Il s'agit d'une interface de navigateur pour OpenGL, qui a permis l'exécution de code JavaScript sur GPU.

En mars 2018, l'équipe DeepLearn.js a fusionné avec l'équipe TensorFlow de Google et a été renommée TensorFlow.js.

Qu'est-ce que c'est ?

TensorFlow.js est une bibliothèque permettant de construire et d'exécuter des algorithmes d'apprentissage automatique en JavaScript. Les modèles TensorFlow.js s'exécutent dans un navigateur Web et dans l'environnement Node.js. La bibliothèque fait partie de l'écosystème TensorFlow, fournissant un ensemble d'API compatibles avec celles de Python, ce qui permet aux modèles d'être portés entre les écosystèmes Python et JavaScript.

TensorFlow.js a permis à un nouveau groupe de développeurs issus de la vaste communauté JavaScript de créer et de déployer des modèles d'apprentissage automatique et de mettre en place de nouvelles classes de calcul sur appareil.

TensorFlow.js permet :

- D'exécuter des modèles existants :
  - Utilisez des modèles JavaScript prêts à l'emploi ou convertissez des modèles TensorFlow codés en Python pour les exécuter dans un navigateur ou sous Node.js.
- D'entraîner à nouveau des modèles existants à l'aide de nos propres données.
- De créer et entraîner des modèles directement en JavaScript à l'aide d'API flexibles et intuitives.

TensorFlow.js fournit deux API :

- L'interface CoreAPI, qui traite le code de bas niveau.
- LayerAPI, qui est construit sur l'interface CoreAPI et facilite le travail en augmentant le niveau d'abstraction.

JavaScript VS WebGL VS WebAssembly

Comparaison avec JavaScript

WASM est généralement beaucoup plus rapide que JavaScript pour les charges de travail numériques communes dans les tâches d'apprentissage machine. En outre, WASM peut être décodé nativement jusqu'à 20 fois plus rapidement que JavaScript.

JavaScript est typé dynamiquement et possède un **garbage collector**, ce qui peut causer des ralentissements non déterministes importants au moment de l'exécution. De plus, les bibliothèques JavaScript modernes (telles que TensorFlow.js) utilisent des outils de compilation tels que TypeScript et des transpileurs ES6 qui génèrent du code ES5 (pour une large prise en charge par les navigateurs) dont l'exécution est plus lente que celle du JavaScript ES6 classique. Le WASM utilise la bibliothèque `XNNPack` pour accélérer les opérations.

Comparaison avec WebGL

Il faut savoir que le WebGL peut être utilisé via le WebAssembly, donc ce ne sont pas forcément deux choses décorrélées. Néanmoins, on peut se demander si le WebGL utilisé avec du code JavaScript est plus rapide que le WebAssembly.

La réponse est comme pour beaucoup de choses en Informatique, ça dépend. Ça dépend de la taille du réseau de neurone à utiliser, mais cela dépend aussi du système d'exécution du processeur pour le WebAssembly.

Nous allons le voir plus tard, mais le SIMD et le multithreading ont véritablement apporté un coup de pouce au WebAssembly qui, sans cela, n'était pas capable d'égaler les performances du WebGL ou du WebGPU.

À noter que, si le WebAssembly est pris en charge par tous les principaux navigateurs, il ne dispose pas tous d'instructions SIMD.

Arrivée du SIMD

Qu'est-ce que le SIMD ?

Single Instruction on Multiple Data, ou **SIMD**, est l'un des quatre modes de fonctionnement définis par la taxinomie de Flynn et désigne un mode de fonctionnement des ordinateurs dotés de plusieurs unités de calcul en parallèle.

Dans ce mode, la même instruction est appliquée simultanément à plusieurs données pour produire plusieurs résultats.

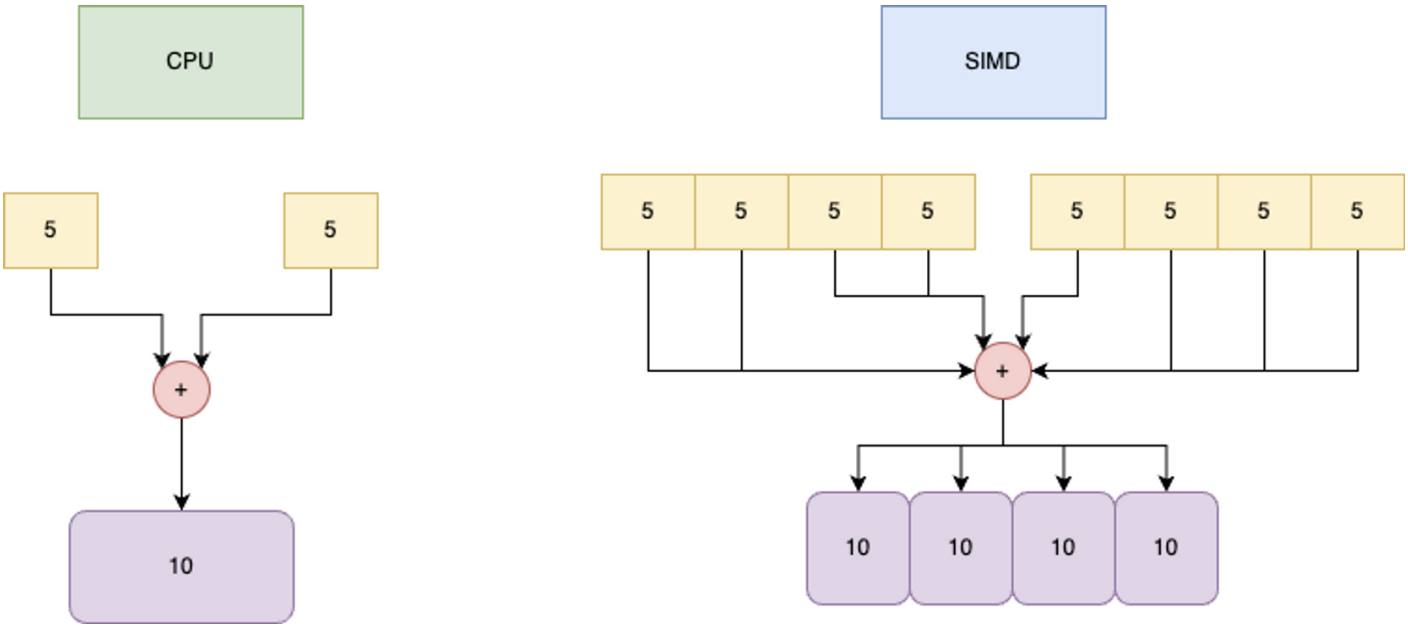
On utilise cette abréviation par opposition à SISD (*Single Instruction on Single Data*), le fonctionnement traditionnel, et MIMD (*Multiple Instructions on Multiple Data*), le fonctionnement avec plusieurs processeurs indépendants. Le modèle SIMD convient particulièrement bien aux traitements dont la structure est très régulière, comme c'est le cas pour le calcul matriciel.

Ce calcul matriciel est à la base de nos modèles de *Deep Learning* et notamment des modèles de vision par ordinateur.

Les instructions SIMD ont été ajoutées aux processeurs modernes pour améliorer la vitesse de traitement des calculs impliquant des nombres en virgule flottante.

Le SIMD est souvent implémenté pour nous dans des bibliothèques hautes performances (comme OpenCV) et généré par notre compilateur lorsque nous compilons avec la vectorisation activée.

Cela se passe au niveau du processeur :



Analogie entre instruction unique / SIMD / Multithreading

Si nous faisons une analogie avec la préparation d'un repas, le SIMD revient à aligner tous nos haricots verts et à les couper en tranches en une seule fois.

L'instruction unique est la "tranche", les données multiples et répétées sont les haricots. En fait, l'alignement des données ("alignement de la mémoire") est un aspect important du SIMD.

Le multithreading, c'est comme si plusieurs chefs prenaient des ingrédients dans un garde-manger commun, les préparaient et les mettaient dans une grande marmite commune. Le travail est effectué plus rapidement car plusieurs chefs sont impliqués.

Arrivé du multithreading

Le multithreading désigne le fait d'avoir plusieurs fils d'exécution, normalement sur différents cœurs de processeur, en même temps. Il s'agit d'un niveau plus élevé que le SIMD et les threads existent généralement beaucoup plus longtemps.

Un thread peut acquérir des images, un autre peut détecter des objets, un autre peut suivre les objets et un dernier peut afficher les résultats.

Une caractéristique du multithreading est que les threads partagent tous le même espace d'adressage, de sorte que les données d'un thread peuvent être vues et manipulées par les autres. Cela rend les threads plus légers que les processus multiples, mais peut rendre le débogage plus difficile, par exemple lorsqu'un thread interfère avec un autre.

Les threads sont dits "légers" car leur création et leur démarrage prennent généralement moins de temps que ceux des processus complets.

Rien n'empêche une application d'utiliser à la fois le SIMD et le multithreading.

Pour revenir à l'analogie de la cuisine, vous pouvez avoir plusieurs chefs (multithreading) qui découpent tous leurs haricots verts de manière efficace (SIMD).

Cas pratique

Introduction

MobileNet

BlazeFace

Revision #1
Created 5 May 2023 13:50:42 by Noé Larrieu-Lacoste
Updated 5 May 2023 13:54:36 by Noé Larrieu-Lacoste