PointNet / Shapenet

Ameni Mtibaa

Tiphaine Le Clercq de Lannoy

Introduction

Nous avons choisi comme sujet de projet le réseau PointNet, qui a été proposé par Charles R. Qi, Hao Su, Kaichun Mo et Leonidas J. Guibas dans le papier “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”, en 2017.

Pointnet fournit une architecture unifiée pour des applications allant de la classification d'objets, la segmentation de pièces à l'analyse sémantique de scènes.

Vu que le code orignal est en tensorflow, nous avons décidé de faire ce projet en utilisant keras pour éviter le copiage.

Nous avons commencé à développer en local et après nous avons switcher à collab pour bénéficier des GPU 😊 .

Le notebook collab contient une description détaillée des différentes étapes avec une petite introduction.

Nous avons essayé de reproduire le max possible l’architecture proposé dans le papier.

Données

Les données de Shapenet sont assez particulières (lien dans les références). En effet, il s’agit d’une base de données comportant des nuages de points qui correspondent à seize types d’objets différents : des avions, des chaises, des guitares, etc. Néanmoins, ces objets sont séparés en plusieurs parties, de deux à six. Par exemple, un avion est composé d’un corps, d’ailes d’une queue et d’un moteur. Nous pouvons donc considérer que chaque point d’un nuage possède deux labels : un pour définir l’objet auquel il appartient, et un autre pour la partie de l’objet. En pratique, nous avons réuni ces deux labels en un seul en même temps que nous avons extrait les données.

A noter que nous avons trouvé 44 catégories en totale (et non pas 50).

Au début, nous avons eu de certaines difficultés à comprendre la structure de la base de données. En effet, vu que c’est la première fois que nous manipulons une base de ce genre, il nous a fallu du temps pour comprendre des détails tels que chaque nuage de points était contenu dans un seul fichier et chaque fichier ne contenait qu’un seul nuage. Le traitement de données a donc été plus long que ce que nous espérions.

Les données de Shapenet sont réparties en trois parties : les données d'entraînement, les données de validation et les données de test. Toutefois, comme il s'agissait d'un challenge, les labels pour les données de test n'étaient pas fournis. Nous ne les avons donc pas utilisées. En revanche, nous avons utilisé les données de validation comme test.

Dans le jeu d'entraînement, il existe 12 137 nuages de points, toutes catégories confondues ; dans le jeu de validation, il y a 1 870 nuages de points. Pour entraîner le réseau, nous avons sélectionné au hasard 2048 points pour chaque nuage de points.

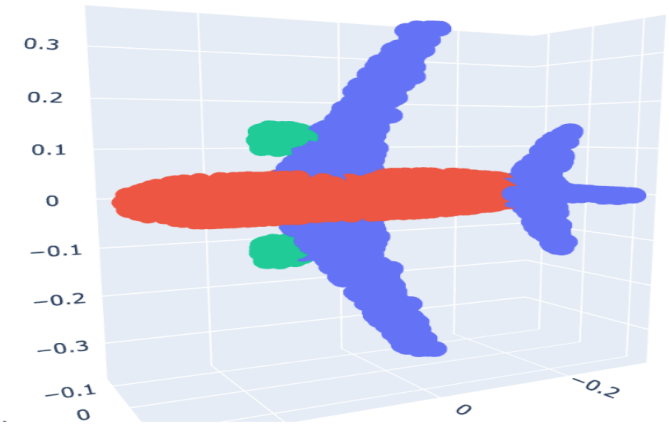
Réseau

Pour fonctionner correctement, le réseau de neurones est composé de plusieurs parties.

- Tout d'abord, une transformation est appliquée sur le nuage de points avec l'aide d'un réseau secondaire, le T-Net. Cela permet d'aligner tous les points des différents nuages dans un même espace canonique afin de pouvoir les comparer. Comme ils l'indiquent dans le papier, les points d'un nuage sont invariants par transformation.
- Ensuite, pour approximer la fonction à apprendre, un multi-layer perceptron est placé après la transformation. Il est composé de convolution1D ainsi que de batch normalization.
- Une deuxième transformation est utilisée dans le réseau, toujours à l'aide d'un T-Net. Il s'agit du feature transformation qui permet d'aligner les différentes parties des objets dans le même espace. Un multi-layer perceptron est ensuite appliqué sur le nuage de points, dont les caractéristiques sont ensuite regroupées en un seul vecteur grâce à une couche maxpooling.
- Un des points-clé du réseau réside ainsi dans le fait que, pour obtenir les prédictions les meilleures possibles, les caractéristiques des parties de l'objet et les caractéristiques du nuage de point tout entier sont concaténées. Enfin, un dernier multi-layer perceptron est rajouté après cela, suivi d'une fonction d'activation softmax.

Résultats

Concernant l'entraînement du réseau de neurones, nous avons obtenu une accuracy de 0.807 pour le jeu de test ce qui est proche du résultat du papier.



Nous pouvons voir ci-dessus une représentation 3D d'un nuage de points de la catégorie airplane faisant partie du jeu de test. Les points de différentes couleurs correspondent aux labels des points prédits par le réseau ; nous pouvons donc observer que le réseau a bien identifié les différentes parties de l'avion.

Critique du papier

Nous avons pu remarquer que le papier était très clair, compréhensible et bien détaillé. En effet, les intuitions étaient bien expliquées, ainsi que les détails plus techniques, qui concernaient l'architecture du réseau ou les explications sur l'utilité des différentes parties du réseau. Les différents paramètres utilisés étaient donc précisés dans le papier et dans l'annexe du papier.

Toutefois, nous avons pu observer une différence d'architecture pour la partie du réseau "part segmentation" entre ce qui était décrit dans le papier, l'annexe, et dans le code officiel, ce qui a créé une légère incertitude (exemple : le papier dit que les couches 'droupout' ne sont pas utilisés dans le réseau part segmentation mais l'implémentation officielle utilise les couche dropout).

Conclusion

Ainsi, nous avons réussi à implémenter le réseau PointNet et nous l'avons entraîné de telle sorte à ce qu'il renvoie une accuracy de 80.7% sur le jeu de test.

Nous avons vu de depuis il y a eu plusieurs améliorations de ce réseau (exemple : pointnet ++), ce qui prouve que ce réseau à révolutionner la manière de faire du DL sur des nuages de points.

Afin de pousser un peu plus nos recherches dans ce domaine, nous pourrions essayer d'appliquer ce réseau à d'autres jeux de données pour observer comment il y réagirait.

Références

http://stanford.edu/~rqi/pointnet/

https://github.com/charlesq34/pointnet

https://www.youtube.com/watch?v=Cge-hot0Oc0

https://shapenet.cs.stanford.edu/iccv17/

Annexes

Architecture du modèle :

```
Model: "model_1"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_1 (InputLayer)         (None, 1024, 3)           0
_________________________________________________________________________________________________
conv1d_1 (Conv1D)            (None, 1024, 64)          256         input_1[0][0]
_________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 1024, 64)       256         conv1d_1[0][0]
_________________________________________________________________________________________________
conv1d_2 (Conv1D)            (None, 1024, 128)         8320        batch_normalization_1[0][0]
_________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 1024, 128)      512         conv1d_2[0][0]
_________________________________________________________________________________________________
conv1d_3 (Conv1D)            (None, 1024, 1024)        132096      batch_normalization_2[0][0]
_________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 1024, 1024)     4096        conv1d_3[0][0]
_________________________________________________________________________________________________
global_max_pooling1d_1 (GlobalM (None, 1024)           0           batch_normalization_3[0][0]
_________________________________________________________________________________________________
dense_1 (Dense)              (None, 512)               524800      global_max_pooling1d_1[0][0]
_________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 512)            2048        dense_1[0][0]
_________________________________________________________________________________________________
dense_2 (Dense)              (None, 256)               131328      batch_normalization_4[0][0]
_________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 256)            1024        dense_2[0][0]
_________________________________________________________________________________________________
dense_3 (Dense)              (None, 9)                 2313        batch_normalization_5[0][0]
_________________________________________________________________________________________________
reshape_1 (Reshape)          (None, 3, 3)              0           dense_3[0][0]
_________________________________________________________________________________________________
dot_1 (Dot)                  (None, 1024, 3)           0           input_1[0][0]
                                                                   reshape_1[0][0]
_________________________________________________________________________________________________
conv1d_4 (Conv1D)            (None, 1024, 64)          256         dot_1[0][0]
_________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 1024, 64)       256         conv1d_4[0][0]
_________________________________________________________________________________________________
conv1d_5 (Conv1D)            (None, 1024, 128)         8320        batch_normalization_6[0][0]
_________________________________________________________________________________________________
batch_normalization_7 (BatchNor (None, 1024, 128)      512         conv1d_5[0][0]
_________________________________________________________________________________________________
conv1d_6 (Conv1D)            (None, 1024, 128)         16512       batch_normalization_7[0][0]
_________________________________________________________________________________________________
batch_normalization_8 (BatchNor (None, 1024, 128)      512         conv1d_6[0][0]
_________________________________________________________________________________________________
conv1d_7 (Conv1D)            (None, 1024, 256)         33024       batch_normalization_8[0][0]
_________________________________________________________________________________________________
batch_normalization_9 (BatchNor (None, 1024, 256)      1024        conv1d_7[0][0]
_________________________________________________________________________________________________
conv1d_8 (Conv1D)            (None, 1024, 1024)        263168      batch_normalization_9[0][0]
_________________________________________________________________________________________________
batch_normalization_10 (BatchNo (None, 1024, 1024)     4096        conv1d_8[0][0]
_________________________________________________________________________________________________
global_max_pooling1d_2 (GlobalM (None, 1024)           0           batch_normalization_10[0][0]
_________________________________________________________________________________________________
dense_4 (Dense)              (None, 512)               524800      global_max_pooling1d_2[0][0]
_________________________________________________________________________________________________
batch_normalization_11 (BatchNo (None, 512)            2048        dense_4[0][0]
_________________________________________________________________________________________________
dense_5 (Dense)              (None, 256)               131328      batch_normalization_11[0][0]
_________________________________________________________________________________________________
batch_normalization_12 (BatchNo (None, 256)            1024        dense_5[0][0]
_________________________________________________________________________________________________
dense_6 (Dense)              (None, 16384)             4210688     batch_normalization_12[0][0]
_________________________________________________________________________________________________
reshape_2 (Reshape)          (None, 128, 128)          0           dense_6[0][0]
_________________________________________________________________________________________________
dot_2 (Dot)                  (None, 1024, 128)         0           batch_normalization_8[0][0]
                                                                   reshape_2[0][0]
```

conv1d_9 (Conv1D)	(None, 1024, 512)	66048	dot_2[0][0]
batch_normalization_13 (BatchNo	(None, 1024, 512)	2048	conv1d_9[0][0]
conv1d_10 (Conv1D)	(None, 1024, 2048)	1050624	batch_normalization_13[0][0]
batch_normalization_14 (BatchNo	(None, 1024, 2048)	8192	conv1d_10[0][0]
global_max_pooling1d_3 (GlobalM	(None, 2048)	0	batch_normalization_14[0][0]
lambda_1 (Lambda)	(None, 1, 2048)	0	global_max_pooling1d_3[0][0]
lambda_2 (Lambda)	(None, 1024, 2048)	0	lambda_1[0][0]
concatenate_1 (Concatenate)	(None, 1024, 3008)	0	batch_normalization_6[0][0] batch_normalization_7[0][0] batch_normalization_8[0][0] dot_2[0][0] batch_normalization_13[0][0] lambda_2[0][0]
conv1d_11 (Conv1D)	(None, 1024, 256)	770304	concatenate_1[0][0]
batch_normalization_15 (BatchNo	(None, 1024, 256)	1024	conv1d_11[0][0]
conv1d_12 (Conv1D)	(None, 1024, 256)	65792	batch_normalization_15[0][0]
batch_normalization_16 (BatchNo	(None, 1024, 256)	1024	conv1d_12[0][0]
conv1d_13 (Conv1D)	(None, 1024, 128)	32896	batch_normalization_16[0][0]
batch_normalization_17 (BatchNo	(None, 1024, 128)	512	conv1d_13[0][0]
conv1d_14 (Conv1D)	(None, 1024, 44)	5676	batch_normalization_17[0][0]

Total params: 8,008,757
Trainable params: 7,993,653
Non-trainable params: 15,104