TER_NoteBook

April 23, 2022

1 Ce travail a été effectuer sur "google colab" car le logiciel notebook sur mon pc qui me permet de faire ce type de travail ne support pas la puissance du module stellargraph.

[ ]:
```
# install StellarGraph if running on Google Colab
import sys
if 'google.colab' in sys.modules:
  %pip install -q stellargraph[demos]==1.0.0rc1
```

```
     |                  | 374 kB 4.3 MB/s
     |                  | 482 kB 52.2 MB/s
     |                  | 462 kB 50.3 MB/s
     |                  | 41 kB 506 kB/s
  Building wheel for mplleaflet (setup.py) … done
```

[ ]:
```
try:
    sg.utils.validate_notebook_version("1.0.0rc1")
except AttributeError:
    raise ValueError(
        f"This notebook requires StellarGraph version 1.0.0rc1, but a different␣
↪version {sg.__version__} is installed.  Please see <https://github.com/
↪stellargraph/stellargraph/issues/1172>."
    ) from None
```

[ ]:
```
import pandas as pd # permet de traiter nos données
import os #

import stellargraph as sg
from stellargraph.mapper import FullBatchNodeGenerator
from stellargraph.layer import GCN  # propose une implementation de␣
↪l'algorithme de GCN

from tensorflow.keras import layers, optimizers, losses, metrics, Model  #pour␣
↪le calcul numérique rapide sous la fonous permettre de decouper notre␣
↪ensemble de données en un en ensemble de train et testrme d'un graphe␣
↪d'exécution
from sklearn import preprocessing, model_selection # qui va
```

```
from IPython.display import display, HTML
import matplotlib.pyplot as plt # va nous permettre de visualiser nos donneé␣
↪(trace des graphique)
%matplotlib inline
```

1. Préparation et Chargement des données (réseau CORA):

[ ]:
```
dataset = sg.datasets.Cora()
display(HTML(dataset.description))
G, node_subjects = dataset.load()
```

```
<IPython.core.display.HTML object>
```

La méthode info nous permet de vérifier que notre graphique chargé correspond à la description

[ ]:
```
print(G.info())
```

```
StellarGraph: Undirected multigraph
 Nodes: 2708, Edges: 5429

 Node types:
  paper: [2708]
    Features: float32 vector, length 1433
    Edge types: paper-cites->paper

 Edge types:
    paper-cites->paper: [5429]
        Weights: all 1 (default)
```

Notre objectif est de former un modèle graph-ML qui prédira l'attribut "sujet" sur les nœuds. Ces sujets sont l'une des 7 catégories, certaines catégories étant plus courantes que d'autres :

[ ]:
```
node_subjects.value_counts().to_frame()
```

[ ]:

	subject
Neural_Networks	818
Probabilistic_Methods	426
Genetic_Algorithms	418
Theory	351
Case_Based	298
Reinforcement_Learning	217
Rule_Learning	180

Nous allons à présent découper notre ensemble de données en un en ensemble d'entraînement (train) et de teste (test)

[ ]:
```
train_subjects, test_subjects = model_selection.train_test_split(
    node_subjects, train_size=140, test_size=None, stratify=node_subjects
)
val_subjects, test_subjects = model_selection.train_test_split(
```

```
    test_subjects, train_size=500, test_size=None, stratify=test_subjects
)
```

Notez que l'utilisation d'un échantillonnage stratifié donne les chiffres suivants :

[ ]:
```
train_subjects.value_counts().to_frame()
```

[ ]:

	subject
Neural_Networks	42
Genetic_Algorithms	22
Probabilistic_Methods	22
Theory	18
Case_Based	16
Reinforcement_Learning	11
Rule_Learning	9

Conversion en tableaux numériques

[ ]:
```
target_encoding = preprocessing.LabelBinarizer()

train_targets = target_encoding.fit_transform(train_subjects)
val_targets = target_encoding.transform(val_subjects)
test_targets = target_encoding.transform(test_subjects)
```

2. Création des couches GCN

Spécifier l'argument method='gcn' au FullBatchNodeGenerator signifie qu'il produira des données appropriées pour l'algorithme GCN spécifiquement, en utilisant la matrice laplacienne du graphe normalisé pour capturer la structure du graphe.

[ ]:
```
generator = FullBatchNodeGenerator(G, method="gcn")
```

```
Using GCN (local pooling) filters…
```

Un générateur code simplement les informations nécessaires pour produire les entrées du modèle. L'appel de la méthode de flux (docs) avec un ensemble de nœuds et leurs véritables étiquettes produit un objet qui peut être utilisé pour former le modèle, sur les nœuds et les étiquettes qui ont été spécifiés. Nous avons créé un ensemble de formation ci-dessus, c'est donc ce que nous allons utiliser ici.

[ ]:
```
train_gen = generator.flow(train_subjects.index, train_targets)
```

Nous pouvons maintenant spécifier notre modèle d'apprentissage automatique en créant une pile de couches. Nous pouvons utiliser la classe GCN de StellarGraph (docs), qui regroupe la création de cette pile de couches de convolution et d'abandon de graphes. Nous pouvons spécifier quelques paramètres pour contrôler cela :

- layer_sizes : le nombre de couches GCN masquées et leurs tailles. Dans ce cas, deux couches GCN de 16 unités chacune.
- activations : l'activation à appliquer à la sortie de chaque couche GCN. Dans ce cas, RelU pour les deux couches.

- dropout : le taux d'abandon pour l'entrée de chaque couche GCN. Dans ce cas, 50 %.

[ ]:
```
gcn = GCN(
    layer_sizes=[16, 16], activations=["relu", "relu"], generator=generator,␣
 ↪dropout=0.5
)
```

Pour créer un modèle Keras, nous exposons maintenant les tenseurs d'entrée et de sortie du modèle GCN pour la prédiction de nœud, via la méthode GCN.in_out_tensors :

[ ]:
```
x_inp, x_out = gcn.in_out_tensors()

x_out
```

[ ]:
```
<KerasTensor: shape=(1, None, 16) dtype=float32 (created by layer
'gather_indices')>
```

[ ]:
```
predictions = layers.Dense(units=train_targets.shape[1],␣
 ↪activation="softmax")(x_out)
```

3. Entraînement et évaluation du modèle

[ ]:
```
model = Model(inputs=x_inp, outputs=predictions)
model.compile(
    optimizer=optimizers.Adam(lr=0.01),
    loss=losses.categorical_crossentropy,
    metrics=["acc"],
)
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105:
UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

[ ]:
```
val_gen = generator.flow(val_subjects.index, val_targets)
```

[ ]:
```
from tensorflow.keras.callbacks import EarlyStopping

es_callback = EarlyStopping(monitor="val_acc", patience=50,␣
 ↪restore_best_weights=True)
```

Nous pouvons directement utiliser la fonctionnalité EarlyStopping (docs) offerte par Keras pour arrêter l'entraînement si la précision de la validation ne s'améliore plus.

[ ]:
```
history = model.fit(
    train_gen,
    epochs=200,
    validation_data=val_gen,
    verbose=2,
    shuffle=False,  # this should be False, since shuffling data means␣
 ↪shuffling the whole graph
```

```
    callbacks=[es_callback],
)
```

```
Epoch 1/200
1/1 - 3s - loss: 1.9389 - acc: 0.1786 - val_loss: 1.9102 - val_acc: 0.3120 -
3s/epoch - 3s/step
Epoch 2/200
1/1 - 0s - loss: 1.8941 - acc: 0.3500 - val_loss: 1.8641 - val_acc: 0.3160 -
247ms/epoch - 247ms/step
Epoch 3/200
1/1 - 0s - loss: 1.8448 - acc: 0.3214 - val_loss: 1.8073 - val_acc: 0.3140 -
282ms/epoch - 282ms/step
Epoch 4/200
1/1 - 0s - loss: 1.7608 - acc: 0.3500 - val_loss: 1.7407 - val_acc: 0.3240 -
273ms/epoch - 273ms/step
Epoch 5/200
1/1 - 0s - loss: 1.6572 - acc: 0.3500 - val_loss: 1.6716 - val_acc: 0.3260 -
235ms/epoch - 235ms/step
Epoch 6/200
1/1 - 0s - loss: 1.5760 - acc: 0.3500 - val_loss: 1.6026 - val_acc: 0.3360 -
213ms/epoch - 213ms/step
Epoch 7/200
1/1 - 0s - loss: 1.4582 - acc: 0.3857 - val_loss: 1.5352 - val_acc: 0.3520 -
206ms/epoch - 206ms/step
Epoch 8/200
1/1 - 0s - loss: 1.3758 - acc: 0.4214 - val_loss: 1.4676 - val_acc: 0.3700 -
193ms/epoch - 193ms/step
Epoch 9/200
1/1 - 0s - loss: 1.2907 - acc: 0.4500 - val_loss: 1.4010 - val_acc: 0.4100 -
237ms/epoch - 237ms/step
Epoch 10/200
1/1 - 0s - loss: 1.1820 - acc: 0.5500 - val_loss: 1.3302 - val_acc: 0.4600 -
207ms/epoch - 207ms/step
Epoch 11/200
1/1 - 0s - loss: 1.1073 - acc: 0.5786 - val_loss: 1.2592 - val_acc: 0.5160 -
409ms/epoch - 409ms/step
Epoch 12/200
1/1 - 0s - loss: 1.0667 - acc: 0.5786 - val_loss: 1.1901 - val_acc: 0.5540 -
312ms/epoch - 312ms/step
Epoch 13/200
1/1 - 0s - loss: 0.9716 - acc: 0.6643 - val_loss: 1.1291 - val_acc: 0.5820 -
186ms/epoch - 186ms/step
Epoch 14/200
1/1 - 0s - loss: 0.8508 - acc: 0.7429 - val_loss: 1.0750 - val_acc: 0.6040 -
138ms/epoch - 138ms/step
Epoch 15/200
1/1 - 0s - loss: 0.8471 - acc: 0.6857 - val_loss: 1.0320 - val_acc: 0.6660 -
147ms/epoch - 147ms/step
```

```
Epoch 16/200
1/1 - 0s - loss: 0.7572 - acc: 0.7000 - val_loss: 0.9931 - val_acc: 0.7000 -
148ms/epoch - 148ms/step
Epoch 17/200
1/1 - 0s - loss: 0.6610 - acc: 0.7857 - val_loss: 0.9610 - val_acc: 0.7080 -
139ms/epoch - 139ms/step
Epoch 18/200
1/1 - 0s - loss: 0.6147 - acc: 0.8143 - val_loss: 0.9278 - val_acc: 0.7160 -
148ms/epoch - 148ms/step
Epoch 19/200
1/1 - 0s - loss: 0.5263 - acc: 0.8857 - val_loss: 0.8997 - val_acc: 0.7260 -
137ms/epoch - 137ms/step
Epoch 20/200
1/1 - 0s - loss: 0.5316 - acc: 0.8643 - val_loss: 0.8838 - val_acc: 0.7300 -
140ms/epoch - 140ms/step
Epoch 21/200
1/1 - 0s - loss: 0.4297 - acc: 0.9000 - val_loss: 0.8782 - val_acc: 0.7340 -
155ms/epoch - 155ms/step
Epoch 22/200
1/1 - 0s - loss: 0.3926 - acc: 0.8857 - val_loss: 0.8930 - val_acc: 0.7300 -
171ms/epoch - 171ms/step
Epoch 23/200
1/1 - 0s - loss: 0.3740 - acc: 0.9500 - val_loss: 0.9022 - val_acc: 0.7360 -
149ms/epoch - 149ms/step
Epoch 24/200
1/1 - 0s - loss: 0.3590 - acc: 0.9143 - val_loss: 0.8866 - val_acc: 0.7400 -
139ms/epoch - 139ms/step
Epoch 25/200
1/1 - 0s - loss: 0.3013 - acc: 0.9000 - val_loss: 0.8562 - val_acc: 0.7500 -
135ms/epoch - 135ms/step
Epoch 26/200
1/1 - 0s - loss: 0.3169 - acc: 0.9286 - val_loss: 0.8322 - val_acc: 0.7580 -
146ms/epoch - 146ms/step
Epoch 27/200
1/1 - 0s - loss: 0.2431 - acc: 0.9500 - val_loss: 0.8152 - val_acc: 0.7620 -
139ms/epoch - 139ms/step
Epoch 28/200
1/1 - 0s - loss: 0.2203 - acc: 0.9571 - val_loss: 0.7981 - val_acc: 0.7640 -
138ms/epoch - 138ms/step
Epoch 29/200
1/1 - 0s - loss: 0.2111 - acc: 0.9500 - val_loss: 0.7912 - val_acc: 0.7680 -
136ms/epoch - 136ms/step
Epoch 30/200
1/1 - 0s - loss: 0.1997 - acc: 0.9500 - val_loss: 0.7851 - val_acc: 0.7680 -
141ms/epoch - 141ms/step
Epoch 31/200
1/1 - 0s - loss: 0.1630 - acc: 0.9429 - val_loss: 0.7941 - val_acc: 0.7700 -
137ms/epoch - 137ms/step
```

```
Epoch 32/200
1/1 - 0s - loss: 0.1549 - acc: 0.9714 - val_loss: 0.8117 - val_acc: 0.7600 -
136ms/epoch - 136ms/step
Epoch 33/200
1/1 - 0s - loss: 0.1351 - acc: 0.9714 - val_loss: 0.8412 - val_acc: 0.7580 -
138ms/epoch - 138ms/step
Epoch 34/200
1/1 - 0s - loss: 0.1236 - acc: 0.9786 - val_loss: 0.8651 - val_acc: 0.7640 -
145ms/epoch - 145ms/step
Epoch 35/200
1/1 - 0s - loss: 0.1329 - acc: 0.9500 - val_loss: 0.8947 - val_acc: 0.7680 -
157ms/epoch - 157ms/step
Epoch 36/200
1/1 - 0s - loss: 0.1414 - acc: 0.9643 - val_loss: 0.9097 - val_acc: 0.7660 -
140ms/epoch - 140ms/step
Epoch 37/200
1/1 - 0s - loss: 0.1010 - acc: 0.9857 - val_loss: 0.9318 - val_acc: 0.7700 -
145ms/epoch - 145ms/step
Epoch 38/200
1/1 - 0s - loss: 0.1095 - acc: 0.9571 - val_loss: 0.9564 - val_acc: 0.7640 -
142ms/epoch - 142ms/step
Epoch 39/200
1/1 - 0s - loss: 0.0950 - acc: 0.9571 - val_loss: 0.9902 - val_acc: 0.7600 -
136ms/epoch - 136ms/step
Epoch 40/200
1/1 - 0s - loss: 0.1002 - acc: 0.9643 - val_loss: 1.0141 - val_acc: 0.7540 -
139ms/epoch - 139ms/step
Epoch 41/200
1/1 - 0s - loss: 0.0695 - acc: 0.9786 - val_loss: 1.0243 - val_acc: 0.7500 -
135ms/epoch - 135ms/step
Epoch 42/200
1/1 - 0s - loss: 0.0939 - acc: 0.9714 - val_loss: 1.0174 - val_acc: 0.7520 -
143ms/epoch - 143ms/step
Epoch 43/200
1/1 - 0s - loss: 0.0758 - acc: 0.9929 - val_loss: 0.9952 - val_acc: 0.7600 -
138ms/epoch - 138ms/step
Epoch 44/200
1/1 - 0s - loss: 0.1011 - acc: 0.9643 - val_loss: 0.9650 - val_acc: 0.7680 -
151ms/epoch - 151ms/step
Epoch 45/200
1/1 - 0s - loss: 0.0569 - acc: 0.9786 - val_loss: 0.9371 - val_acc: 0.7680 -
144ms/epoch - 144ms/step
Epoch 46/200
1/1 - 0s - loss: 0.0461 - acc: 0.9929 - val_loss: 0.9240 - val_acc: 0.7740 -
138ms/epoch - 138ms/step
Epoch 47/200
1/1 - 0s - loss: 0.0656 - acc: 0.9929 - val_loss: 0.9195 - val_acc: 0.7720 -
135ms/epoch - 135ms/step
```

```
Epoch 48/200
1/1 - 0s - loss: 0.0550 - acc: 0.9929 - val_loss: 0.9208 - val_acc: 0.7700 -
136ms/epoch - 136ms/step
Epoch 49/200
1/1 - 0s - loss: 0.1148 - acc: 0.9571 - val_loss: 0.9311 - val_acc: 0.7660 -
145ms/epoch - 145ms/step
Epoch 50/200
1/1 - 0s - loss: 0.0605 - acc: 0.9929 - val_loss: 0.9437 - val_acc: 0.7660 -
131ms/epoch - 131ms/step
Epoch 51/200
1/1 - 0s - loss: 0.0442 - acc: 0.9929 - val_loss: 0.9597 - val_acc: 0.7680 -
142ms/epoch - 142ms/step
Epoch 52/200
1/1 - 0s - loss: 0.0603 - acc: 0.9786 - val_loss: 0.9792 - val_acc: 0.7700 -
131ms/epoch - 131ms/step
Epoch 53/200
1/1 - 0s - loss: 0.0884 - acc: 0.9643 - val_loss: 1.0119 - val_acc: 0.7720 -
133ms/epoch - 133ms/step
Epoch 54/200
1/1 - 0s - loss: 0.0631 - acc: 0.9714 - val_loss: 1.0427 - val_acc: 0.7700 -
138ms/epoch - 138ms/step
Epoch 55/200
1/1 - 0s - loss: 0.0443 - acc: 0.9929 - val_loss: 1.0752 - val_acc: 0.7640 -
143ms/epoch - 143ms/step
Epoch 56/200
1/1 - 0s - loss: 0.0499 - acc: 0.9786 - val_loss: 1.1162 - val_acc: 0.7560 -
141ms/epoch - 141ms/step
Epoch 57/200
1/1 - 0s - loss: 0.0838 - acc: 0.9786 - val_loss: 1.1405 - val_acc: 0.7540 -
146ms/epoch - 146ms/step
Epoch 58/200
1/1 - 0s - loss: 0.0616 - acc: 0.9857 - val_loss: 1.1639 - val_acc: 0.7540 -
140ms/epoch - 140ms/step
Epoch 59/200
1/1 - 0s - loss: 0.0350 - acc: 1.0000 - val_loss: 1.1820 - val_acc: 0.7520 -
150ms/epoch - 150ms/step
Epoch 60/200
1/1 - 0s - loss: 0.0520 - acc: 0.9857 - val_loss: 1.1711 - val_acc: 0.7540 -
137ms/epoch - 137ms/step
Epoch 61/200
1/1 - 0s - loss: 0.0315 - acc: 0.9929 - val_loss: 1.1509 - val_acc: 0.7540 -
139ms/epoch - 139ms/step
Epoch 62/200
1/1 - 0s - loss: 0.0530 - acc: 0.9857 - val_loss: 1.1246 - val_acc: 0.7580 -
135ms/epoch - 135ms/step
Epoch 63/200
1/1 - 0s - loss: 0.0774 - acc: 0.9714 - val_loss: 1.0827 - val_acc: 0.7760 -
141ms/epoch - 141ms/step
```

```
Epoch 64/200
1/1 - 0s - loss: 0.0516 - acc: 0.9786 - val_loss: 1.0509 - val_acc: 0.7820 -
151ms/epoch - 151ms/step
Epoch 65/200
1/1 - 0s - loss: 0.0413 - acc: 0.9929 - val_loss: 1.0265 - val_acc: 0.7880 -
145ms/epoch - 145ms/step
Epoch 66/200
1/1 - 0s - loss: 0.0865 - acc: 0.9643 - val_loss: 1.0119 - val_acc: 0.7860 -
143ms/epoch - 143ms/step
Epoch 67/200
1/1 - 0s - loss: 0.0290 - acc: 0.9929 - val_loss: 1.0046 - val_acc: 0.7860 -
147ms/epoch - 147ms/step
Epoch 68/200
1/1 - 0s - loss: 0.0533 - acc: 0.9857 - val_loss: 1.0068 - val_acc: 0.7800 -
141ms/epoch - 141ms/step
Epoch 69/200
1/1 - 0s - loss: 0.0230 - acc: 1.0000 - val_loss: 1.0153 - val_acc: 0.7760 -
147ms/epoch - 147ms/step
Epoch 70/200
1/1 - 0s - loss: 0.0368 - acc: 0.9929 - val_loss: 1.0299 - val_acc: 0.7740 -
147ms/epoch - 147ms/step
Epoch 71/200
1/1 - 0s - loss: 0.0946 - acc: 0.9714 - val_loss: 1.0499 - val_acc: 0.7720 -
140ms/epoch - 140ms/step
Epoch 72/200
1/1 - 0s - loss: 0.0707 - acc: 0.9643 - val_loss: 1.0683 - val_acc: 0.7720 -
139ms/epoch - 139ms/step
Epoch 73/200
1/1 - 0s - loss: 0.0300 - acc: 0.9929 - val_loss: 1.0906 - val_acc: 0.7700 -
141ms/epoch - 141ms/step
Epoch 74/200
1/1 - 0s - loss: 0.0355 - acc: 1.0000 - val_loss: 1.1174 - val_acc: 0.7680 -
154ms/epoch - 154ms/step
Epoch 75/200
1/1 - 0s - loss: 0.0327 - acc: 0.9929 - val_loss: 1.1451 - val_acc: 0.7660 -
144ms/epoch - 144ms/step
Epoch 76/200
1/1 - 0s - loss: 0.0548 - acc: 0.9714 - val_loss: 1.1775 - val_acc: 0.7640 -
144ms/epoch - 144ms/step
Epoch 77/200
1/1 - 0s - loss: 0.0355 - acc: 0.9929 - val_loss: 1.2062 - val_acc: 0.7580 -
153ms/epoch - 153ms/step
Epoch 78/200
1/1 - 0s - loss: 0.0554 - acc: 0.9857 - val_loss: 1.2265 - val_acc: 0.7580 -
143ms/epoch - 143ms/step
Epoch 79/200
1/1 - 0s - loss: 0.0639 - acc: 0.9714 - val_loss: 1.2416 - val_acc: 0.7560 -
153ms/epoch - 153ms/step
```

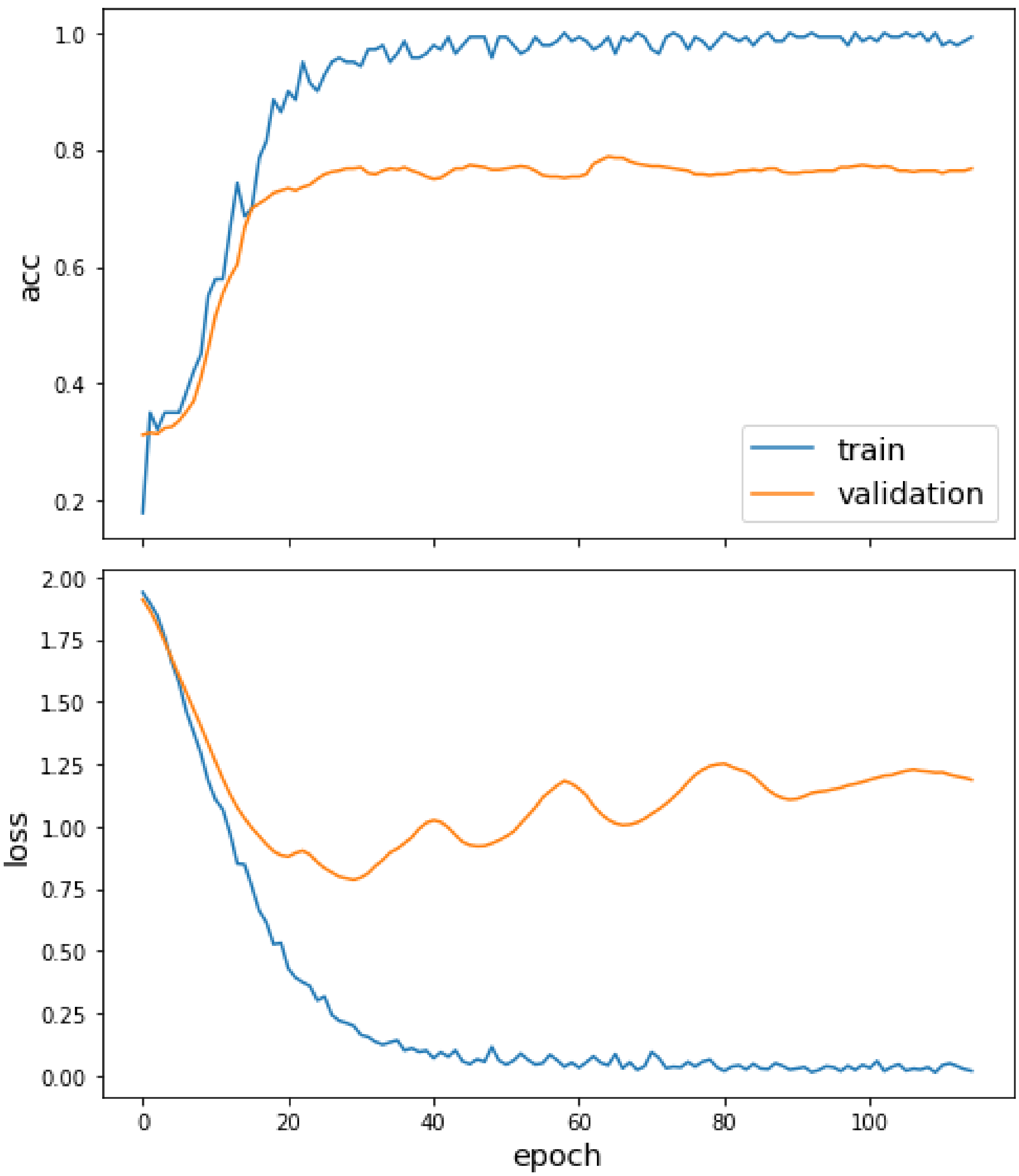
```
Epoch 80/200
1/1 - 0s - loss: 0.0329 - acc: 0.9857 - val_loss: 1.2485 - val_acc: 0.7580 -
138ms/epoch - 138ms/step
Epoch 81/200
1/1 - 0s - loss: 0.0180 - acc: 1.0000 - val_loss: 1.2520 - val_acc: 0.7580 -
153ms/epoch - 153ms/step
Epoch 82/200
1/1 - 0s - loss: 0.0355 - acc: 0.9929 - val_loss: 1.2391 - val_acc: 0.7600 -
141ms/epoch - 141ms/step
Epoch 83/200
1/1 - 0s - loss: 0.0407 - acc: 0.9857 - val_loss: 1.2274 - val_acc: 0.7640 -
141ms/epoch - 141ms/step
Epoch 84/200
1/1 - 0s - loss: 0.0246 - acc: 0.9929 - val_loss: 1.2185 - val_acc: 0.7640 -
138ms/epoch - 138ms/step
Epoch 85/200
1/1 - 0s - loss: 0.0466 - acc: 0.9786 - val_loss: 1.2000 - val_acc: 0.7660 -
158ms/epoch - 158ms/step
Epoch 86/200
1/1 - 0s - loss: 0.0282 - acc: 0.9929 - val_loss: 1.1733 - val_acc: 0.7640 -
141ms/epoch - 141ms/step
Epoch 87/200
1/1 - 0s - loss: 0.0256 - acc: 1.0000 - val_loss: 1.1459 - val_acc: 0.7680 -
140ms/epoch - 140ms/step
Epoch 88/200
1/1 - 0s - loss: 0.0496 - acc: 0.9857 - val_loss: 1.1262 - val_acc: 0.7680 -
145ms/epoch - 145ms/step
Epoch 89/200
1/1 - 0s - loss: 0.0386 - acc: 0.9857 - val_loss: 1.1138 - val_acc: 0.7620 -
176ms/epoch - 176ms/step
Epoch 90/200
1/1 - 0s - loss: 0.0236 - acc: 1.0000 - val_loss: 1.1078 - val_acc: 0.7600 -
139ms/epoch - 139ms/step
Epoch 91/200
1/1 - 0s - loss: 0.0283 - acc: 0.9929 - val_loss: 1.1107 - val_acc: 0.7600 -
144ms/epoch - 144ms/step
Epoch 92/200
1/1 - 0s - loss: 0.0342 - acc: 0.9929 - val_loss: 1.1213 - val_acc: 0.7620 -
141ms/epoch - 141ms/step
Epoch 93/200
1/1 - 0s - loss: 0.0138 - acc: 1.0000 - val_loss: 1.1334 - val_acc: 0.7620 -
150ms/epoch - 150ms/step
Epoch 94/200
1/1 - 0s - loss: 0.0220 - acc: 0.9929 - val_loss: 1.1390 - val_acc: 0.7640 -
142ms/epoch - 142ms/step
Epoch 95/200
1/1 - 0s - loss: 0.0375 - acc: 0.9929 - val_loss: 1.1424 - val_acc: 0.7640 -
173ms/epoch - 173ms/step
```

```
Epoch 96/200
1/1 - 0s - loss: 0.0334 - acc: 0.9929 - val_loss: 1.1485 - val_acc: 0.7640 -
141ms/epoch - 141ms/step
Epoch 97/200
1/1 - 0s - loss: 0.0178 - acc: 0.9929 - val_loss: 1.1547 - val_acc: 0.7700 -
137ms/epoch - 137ms/step
Epoch 98/200
1/1 - 0s - loss: 0.0391 - acc: 0.9786 - val_loss: 1.1646 - val_acc: 0.7700 -
153ms/epoch - 153ms/step
Epoch 99/200
1/1 - 0s - loss: 0.0223 - acc: 1.0000 - val_loss: 1.1703 - val_acc: 0.7720 -
140ms/epoch - 140ms/step
Epoch 100/200
1/1 - 0s - loss: 0.0424 - acc: 0.9857 - val_loss: 1.1776 - val_acc: 0.7740 -
142ms/epoch - 142ms/step
Epoch 101/200
1/1 - 0s - loss: 0.0301 - acc: 0.9929 - val_loss: 1.1869 - val_acc: 0.7720 -
140ms/epoch - 140ms/step
Epoch 102/200
1/1 - 0s - loss: 0.0583 - acc: 0.9857 - val_loss: 1.1946 - val_acc: 0.7700 -
154ms/epoch - 154ms/step
Epoch 103/200
1/1 - 0s - loss: 0.0178 - acc: 1.0000 - val_loss: 1.2032 - val_acc: 0.7720 -
145ms/epoch - 145ms/step
Epoch 104/200
1/1 - 0s - loss: 0.0342 - acc: 0.9929 - val_loss: 1.2052 - val_acc: 0.7700 -
138ms/epoch - 138ms/step
Epoch 105/200
1/1 - 0s - loss: 0.0450 - acc: 0.9929 - val_loss: 1.2150 - val_acc: 0.7640 -
140ms/epoch - 140ms/step
Epoch 106/200
1/1 - 0s - loss: 0.0195 - acc: 1.0000 - val_loss: 1.2225 - val_acc: 0.7640 -
138ms/epoch - 138ms/step
Epoch 107/200
1/1 - 0s - loss: 0.0274 - acc: 0.9929 - val_loss: 1.2274 - val_acc: 0.7620 -
143ms/epoch - 143ms/step
Epoch 108/200
1/1 - 0s - loss: 0.0237 - acc: 1.0000 - val_loss: 1.2226 - val_acc: 0.7640 -
139ms/epoch - 139ms/step
Epoch 109/200
1/1 - 0s - loss: 0.0332 - acc: 0.9857 - val_loss: 1.2198 - val_acc: 0.7640 -
149ms/epoch - 149ms/step
Epoch 110/200
1/1 - 0s - loss: 0.0124 - acc: 1.0000 - val_loss: 1.2158 - val_acc: 0.7640 -
139ms/epoch - 139ms/step
Epoch 111/200
1/1 - 0s - loss: 0.0414 - acc: 0.9786 - val_loss: 1.2166 - val_acc: 0.7600 -
136ms/epoch - 136ms/step
```

```
Epoch 112/200
1/1 - 0s - loss: 0.0488 - acc: 0.9857 - val_loss: 1.2070 - val_acc: 0.7640 -
149ms/epoch - 149ms/step
Epoch 113/200
1/1 - 0s - loss: 0.0378 - acc: 0.9786 - val_loss: 1.2003 - val_acc: 0.7640 -
132ms/epoch - 132ms/step
Epoch 114/200
1/1 - 0s - loss: 0.0256 - acc: 0.9857 - val_loss: 1.1945 - val_acc: 0.7640 -
146ms/epoch - 146ms/step
Epoch 115/200
1/1 - 0s - loss: 0.0187 - acc: 0.9929 - val_loss: 1.1865 - val_acc: 0.7680 -
136ms/epoch - 136ms/step
```

Une fois que nous avons formé le modèle, nous pouvons afficher la fonction de perte de comportement et toute autre métrique à l'aide de la fonction plot_history (docs). Dans ce cas, nous pouvons voir la perte et la précision sur les ensembles de formation et de validation.

[ ]:
```
sg.utils.plot_history(history)
```



[ ]:
```
test_gen = generator.flow(test_subjects.index, test_targets)
```

[ ]:
```
test_metrics = model.evaluate(test_gen)
print("\nTest Set Metrics:")
for name, val in zip(model.metrics_names, test_metrics):
    print("\t{}: {:0.4f}".format(name, val))
```

```
1/1 [==============================] - 0s 265ms/step - loss: 0.8468 - acc:
0.7974
```

```
Test Set Metrics:
	loss: 0.8468
	acc: 0.7974
```

Faire des prédictions avec le modèle

Obtenons maintenant les prédictions pour tous les nœuds. Vous vous y êtes probablement habitué maintenant, mais nous utilisons notre FullBatchNodeGenerator pour créer l'entrée requise, puis utilisons l'une des méthodes du modèle : prédire (docs). Cette fois, nous ne fournissons pas les étiquettes au flux, mais uniquement les nœuds, car nous essayons de prédire ces classes sans les connaître.

[ ]:
```
all_nodes = node_subjects.index
all_gen = generator.flow(all_nodes)
all_predictions = model.predict(all_gen)
```

Ces prédictions seront la sortie de la couche softmax, donc pour obtenir les catégories finales, nous utiliserons la méthode inverse_transform de notre spécification d'attribut cible pour ramener ces valeurs aux catégories d'origine.

[ ]:
```
node_predictions = target_encoding.inverse_transform(all_predictions.squeeze())
```

Examinons quelques prédictions après l'entraînement du modèle

[ ]:
```
df = pd.DataFrame({"Prediction": node_predictions, "Vrai": node_subjects})
df.head(20)
```

[ ]:

	Prediction	Vrai
31336	Neural_Networks	Neural_Networks
1061127	Rule_Learning	Rule_Learning
1106406	Reinforcement_Learning	Reinforcement_Learning
13195	Reinforcement_Learning	Reinforcement_Learning
37879	Probabilistic_Methods	Probabilistic_Methods
1126012	Probabilistic_Methods	Probabilistic_Methods
1107140	Theory	Theory
1102850	Neural_Networks	Neural_Networks
31349	Neural_Networks	Neural_Networks
1106418	Theory	Theory
1123188	Probabilistic_Methods	Neural_Networks
1128990	Genetic_Algorithms	Genetic_Algorithms
109323	Probabilistic_Methods	Probabilistic_Methods
217139	Case_Based	Case_Based
31353	Neural_Networks	Neural_Networks
32083	Neural_Networks	Neural_Networks
1126029	Reinforcement_Learning	Reinforcement_Learning
1118017	Neural_Networks	Neural_Networks
49482	Neural_Networks	Neural_Networks
753265	Neural_Networks	Neural_Networks

En plus de simplement prédire la classe de nœuds, il peut être utile d'obtenir une image plus détaillée des informations que le modèle a apprises sur les nœuds et leurs voisinages. Dans ce cas, cela signifie une intégration du nœud (également appelé « représentation ») dans un espace vectoriel latent qui capture cette information, et il se présente sous la forme soit d'un nœud de mappage de table de consultation vers un vecteur de nombres, ou un réseau neuronal qui produit ces vecteurs. Pour GCN, nous allons utiliser la deuxième option, en utilisant la dernière couche de convolution graphique du modèle GCN (appelée x_out ci-dessus), avant d'appliquer la couche de prédiction.

[ ]:
```
embedding_model = Model(inputs=x_inp, outputs=x_out)
```

[ ]:
```
emb = embedding_model.predict(all_gen)
emb.shape
```

[ ]:
```
(1, 2708, 16)
```

[ ]:
```
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

transform = TSNE
```

[ ]:
```
X = emb.squeeze(0)
X.shape
```

[ ]:
```
(2708, 16)
```

[ ]:
```
trans = transform(n_components=2)
X_reduced = trans.fit_transform(X)
X_reduced.shape
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783:
FutureWarning: The default initialization in TSNE will change from 'random' to
'pca' in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793:
FutureWarning: The default learning rate in TSNE will change from 200.0 to
'auto' in 1.2.
  FutureWarning,
```

[ ]:
```
(2708, 2)
```

[ ]:
```
fig, ax = plt.subplots(figsize=(7, 7))
ax.scatter(
    X_reduced[:, 0],
    X_reduced[:, 1],
    c=node_subjects.astype("category").cat.codes,
    cmap="jet",
    alpha=0.7,
)
```

```
ax.set(
    aspect="equal",
    xlabel="$X_1$",
    ylabel="$X_2$",
    title=f"{transform.__name__} visualisation des intégrations GCN pour␣
 ↪l'ensemble de données cora",
)
```

```
[ ]: [Text(0, 0.5, '$X_2$'),
      Text(0.5, 0, '$X_1$'),
      Text(0.5, 1.0, "TSNE visualisation des intégrations GCN pour l'ensemble de
     données cora"),
      None]
```