In [1]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RepeatedStratifiedKFold
import sklearn.metrics
import warnings
import numpy as np
# warnings.filterwarnings('ignore')
```

In [2]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import regularizers
import matplotlib.pyplot as plt
```

In [3]:
```
from sklearn.datasets import load_iris
(X, y) = load_iris(return_X_y=True, as_frame=False)
X_in1, X_in2 = X[:, :2], X[:, 2:]
n, p = X.shape
```

In [4]:
```
n_repeats = 5
n_splits = 10
metrics_list = ['recall', 'precision', 'AUC']
```

In [5]:
```
model_0 = LogisticRegression(multi_class='multinomial', max_iter=800)
print("model_0(logistic regresion in sklearn)", model_0)
```

```
model_0(logistic regresion in sklearn) LogisticRegression(max_iter=800, mul
ti_class='multinomial')
```

In [6]:
```
m1_in1 = keras.Input(shape=(2,), name='m1_in1')
m1_in1_h1 = layers.Dense(10, activation='relu')(m1_in1)
m1_in1_h2 = layers.Dense(4, activation='relu')(m1_in1_h1)
o_1 = layers.Dense(2)(m1_in1_h2)
m1_in2 = keras.Input(shape=(2,), name='m1_in2')
m1_in2_h1 = layers.Dense(10, activation='relu')(m1_in2)
m1_in2_h2 = layers.Dense(4, activation='relu')(m1_in2_h1)
o_2 = layers.Dense(2)(m1_in2_h2)
concat = layers.Concatenate()([o_1, o_2])
outputs = layers.Dense(3, activation='softmax', name='y')(concat)
model_1 = keras.Model(inputs=[m1_in1, m1_in2], outputs=outputs)
print("model_1(tf functional API)", model_1.summary())
```

```
Model: "functional_1"
__________________________________________________________________________
_____________________
Layer (type)                 Output Shape              Param #     Connect
ed to
==========================================================================
=========================
m1_in1 (InputLayer)          [(None, 2)]               0
__________________________________________________________________________
_____________________
m1_in2 (InputLayer)          [(None, 2)]               0
__________________________________________________________________________
_____________________
dense (Dense)                (None, 10)                30          m1_in1
[0][0]
__________________________________________________________________________
_____________________
dense_3 (Dense)              (None, 10)                30          m1_in2
[0][0]
__________________________________________________________________________
_____________________
dense_1 (Dense)              (None, 4)                 44          dense
[0][0]
__________________________________________________________________________
_____________________
dense_4 (Dense)              (None, 4)                 44          dense_3
[0][0]
__________________________________________________________________________
_____________________
dense_2 (Dense)              (None, 2)                 10          dense_1
[0][0]
__________________________________________________________________________
_____________________
dense_5 (Dense)              (None, 2)                 10          dense_4
[0][0]
__________________________________________________________________________
_____________________
concatenate (Concatenate)    (None, 4)                 0           dense_2
[0][0]
                                                                   dense_5

[0][0]
__________________________________________________________________________
_____________________
```

```
y (Dense)                       (None, 3)            15          concate
nate[0][0]
==================================================================================
==========================
Total params: 183
Trainable params: 183
Non-trainable params: 0
__________________________________________________________________________________
_____________________________
model_1(tf functional API) None
```

In [7]:
```
model_1.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy())
loss_model_1 = []
n_models = 2
for i in range(n_models):

    for metrics in metrics_list:
        locals()[metrics + '_{}'.format(i)] = []
```

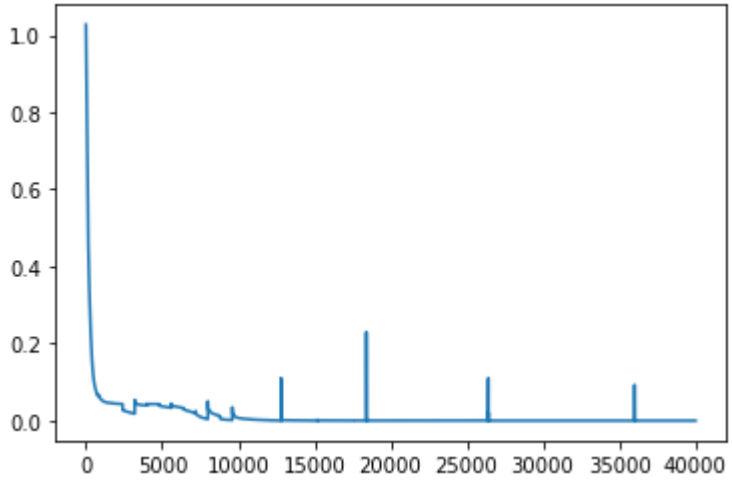
In [8]:
```
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder(handle_unknown='ignore')
def one_hot(x):
    return enc.fit_transform(x.reshape(-1, 1)).toarray()
```

In [9]:

```
rskf = RepeatedStratifiedKFold(n_repeats=n_repeats, n_splits=n_splits)

for train_index, test_index in rskf.split(X, y):

    # 切training testing data
    X_tr, X_te = X[train_index], X[test_index]
    y_tr, y_te = y[train_index], y[test_index]
    in1_tr, in2_tr = X_tr[:, :2], X_tr[:, 2:]
    in1_te, in2_te = X_te[:, :2], X_te[:, 2:]

    ###fit models###
    history_1 = model_1.fit(
        {'m1_in1': in1_tr, 'm1_in2': in2_tr},
        {'y': y_tr},
        batch_size=n, epochs=800, verbose=0)
    model_0.fit(X_tr, y_tr)

    ###把loss存起來###
    loss_model_1.extend(history_1.history['loss'])

    pred_0 = model_0.predict(X_te)
    pred_1 = np.argmax(model_1.predict\
                       ({'m1_in1': in1_te, 'm1_in2': in2_te})\
                       , axis=1)

    for i in range(n_models):
        ###multiple的predict用argmax###
        ###data很平衡 所以都用macro的就好了###

        locals()['recall_{}'.format(i)].append\
        (sklearn.metrics.recall_score(y_te, \
                                      locals()['pred_{}'.format(i)], \
                                      average='macro'))
        locals()['precision_{}'.format(i)].append\
        (sklearn.metrics.precision_score(y_te, \
                                         locals()['pred_{}'.format(i)], \
                                         average='macro'))
        locals()['AUC_{}'.format(i)].append\
        (sklearn.metrics.roc_auc_score(one_hot(y_te), \
                                       one_hot(locals()['pred_{}'.format(i)])
                                       multi_class='ovr', \
                                       average='macro'))

# 把每次cv完的結果取平均
for i in range(n_models):
    for metrics in metrics_list:
        locals()[metrics + '_{}'.format(i)] = \
        np.mean(locals()[metrics + '_{}'.format(i)])
        # mean of (n_splits * n_repeats) values
```

In [10]:
```
# 每800個epochs會換一筆data train
plt.plot(loss_model_1)
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0x136f0c98400>]
```



In [11]:
```
k = locals()
for metrics in metrics_list:
    locals()[metrics+'s'] = \
    [(k[metrics+'_{}'.format(i)]) for i in range(n_models)]
    locals()[metrics+'_max_idx'] = \
    np.argmax(locals()[metrics+'s'])
    locals()[metrics+'_max'] = \
    max(locals()[metrics+'s'])
```

In [12]:
```
print("因為資料的label很平均，metrics的weight用的都是macro")
mission_to_metrics = \
{'看recall':'recall', \
 '看precision':'precision', \
 '看綜合分數AUC':'AUC'}

for mission, metrics in mission_to_metrics.items():
    print("兩個分類器的{}: \n{}".format(metrics, locals()[metrics+'s']))
    best_idx = locals()[metrics+'_max_idx']
    print("如果{}，model_{}是最好的分類器，{}={}".format\
          (mission, best_idx, metrics, locals()[metrics+'_max']))
    print("#"*50)
```

```
因為資料的label很平均，metrics的weight用的都是macro
兩個分類器的recall:
[0.9613333333333333, 0.9880000000000001]
如果看recall，model_1是最好的分類器，recall=0.9880000000000001
##################################################
兩個分類器的precision:
[0.9686031746031746, 0.9903174603174604]
如果看precision，model_1是最好的分類器，precision=0.9903174603174604
##################################################
兩個分類器的AUC:
[0.971, 0.991]
如果看綜合分數AUC，model_1是最好的分類器，AUC=0.991
##################################################
```