```
001| """ GAN """
002|
003|
004| import numpy as np
005| from matplotlib import pyplot as plt
006| from random import sample
007|
008| """ Fonction d'activation"""
009| def sigmoid(x):
010|     return 1/(1+np.exp(-x))
011|
012|
013| class Generateur():
014|
015|     def __init__(self):
016|         """ Poids et biais des neurones """
017|         self.poids=np.random.normal(size=n*m)
018|         self.biais=np.random.normal(size=n*m)
019|
020|     """ Essaie de trouver la bonne distribution """
021|     def generer_image(self,z):
022|         return sigmoid(self.poids*z + self.biais)
023|
024|     def derivation(self, z, D): #Pour que l'algo ne sature
pas
025|         """ Calculer via la rétropropagation """
026|         # on dérive ln(D(G(z))) en fonction des poids et
des biais
027|         x=self.generer_image(z)
028|         y=D.score(x)
029|         derivees_poids = D.poids * (1-y) * z * x * (1-x)
030|         derivees_biais = D.poids * (1-y) * x * (1-x)
031|         return derivees_poids, derivees_biais
032|
033|     def optimisation(self, MDP, MDB):
034|         # on veut maximiser cette fonction
035|         # donc on fait une montée de gradient
036|         self.poids += pas * MDP
037|         self.biais += pas * MDB
038|
039| ################### COMME DANS ALGO DE L'ARTICLE
040|     def derivation_sature(self, z, D):
041|         """ Calculer via la rétropropagation """
042|         # on dérive ln(1-D(G(z))) en fonction des poids et
des biais
043|         x=self.generer_image(z)
044|         y=D.score(x)
045|         derivees_poids = - D.poids * y * z * x * (1-x)
046|         derivees_biais = - D.poids * y * z * x * (1-x)
047|         return derivees_poids, derivees_biais
```

```
048|
049|     def optimisation_sature(self, MDP, MDB):
050|         # on veut minimiser cette fonction
051|         # donc on fait une descente de gradient
052|         self.poids -= pas * MDP
053|         self.biais -= pas * MDB
054| ###################
055|
056|
057| class Discriminant():
058|
059|     def __init__(self):
060|         """ Poids et biais du neurone"""
061|         self.poids=np.random.normal(size=n*m)
062|         self.biais=np.random.normal()
063|
064|     """ Retourne la proba que l'image soit vrai """
065|     def score(self, image):
066|         return sigmoid(np.dot(self.poids,image) +
self.biais)
067|
068|     def derivation_data(self, x):
069|         """ Calculer via la rétropropagation """
070|         # on dérive ln(D(x))
071|         y=self.score(x)
072|         derivees_poids = x*(1-y)
073|         derivees_biais = 1-y
074|         return derivees_poids, derivees_biais
075|
076|     def optimisation_data(self, MDP, MDB):
077|         # on veut maximiser cette fonction
078|         # donc on fait une montée de gradient
079|         self.poids += pas * MDP
080|         self.biais += pas * MDB
081|
082|     def derivation_generee(self, x):
083|         """ Calculer via la rétropropagation """
084|         # on dérive ln(1-D(G(z))) (rappel ici x=G(z))
085|         y=self.score(x)
086|         derivees_poids = -x*y
087|         derivees_biais = -y
088|         return derivees_poids, derivees_biais
089|
090|     def optimisation_generee(self, MDP, MDB):
091|         # on veut maximiser cette fonction
092|         # donc on fait une montée de gradient
093|         self.poids += pas * MDP
094|         self.biais += pas * MDB
095|
096|
```

```
097|
098| def GAN(nb_iterations, nb_etapes, nb_echantillons):
099|
100|     G = Generateur()
101|     D = Discriminant()
102|
103|     evolution=[]
104|     X = [i for i in range(nb_iterations)]
105|     YD = [0 for i in range(nb_iterations)]
106|     YG = [0 for i in range(nb_iterations)]
107|     for i in range(nb_iterations):
108|         """ On optimise nb_etapes fois le discriminant puis
une fois le générateur """
109|         for k in range(nb_etapes):
110|
111|             sample_x=sample(data, nb_echantillons)
112|
113|             # moyenneD liste les moyennes empiriques des m
dérivées des images données et générées, en fonction des poids
et biais pour optimiser le discriminant
114|             # moyenneD=["dérivée image générée en fonction
poids",
115|             #           "----------------------------------
biais",
116|             #           "-------------donnée--------------
poids",
117|             #           "----------------------------------
biais"]
118|             moyenneD=[0,0,0,0]
119|
120|             for m in range(nb_echantillons):
121|
122|                 z=np.random.rand()
123|                 image_generee=G.generer_image(z)
124|                 derive_z =
D.derivation_generee(image_generee)
125|                 moyenneD[0] += derive_z[0]
126|                 moyenneD[1] += derive_z[1]
127|
128|                 derive_x = D.derivation_data(sample_x[m])
129|                 moyenneD[2] += derive_x[0]
130|                 moyenneD[3] += derive_x[1]
131|
132|             """ Mise à jour du discriminant en montant son
gradient stochastique  """
133|             moyenneD[2] = moyenneD[2]/nb_echantillons
134|             moyenneD[3] = moyenneD[3]/nb_echantillons
135|             D.optimisation_data(moyenneD[2],moyenneD[3])
136|
137|             moyenneD[0] = moyenneD[0]/nb_echantillons
```

```
138|             moyenneD[1] = moyenneD[1]/nb_echantillons
139|             D.optimisation_generee(moyenneD[0],moyenneD[1])
140|
141|         # moyenneG liste les moyennes empiriques des m
dérivées des images générées, en fonction des poids et biais
pour optimiser le générateur
142|         # moyenneG=["dérivée image générée en fonction
poids",
143|         #           "---------------------------------
biais"]
144|         moyenneG=[0,0]
145|
146|         for m in range(nb_echantillons):
147|
148|             z=np.random.rand()
149|             derive=G.derivation(z, D) ############# ou
derive=G.derivation_sature(z, D)
150|             moyenneG[0] += derive[0]
151|             moyenneG[1] += derive[1]
152|
153|         """ Mise à jour du générateur en descendant son
gradient stochastique """
154|         moyenneG[0]=moyenneG[0]/nb_echantillons
155|         moyenneG[1]=moyenneG[1]/nb_echantillons
156|         G.optimisation(moyenneG[0],moyenneG[1])
############# ou G.optimisation_sature(moyenneG[0],moyenneG[1])
157|
158|     return G,D
159|
160| generateur,discriminant=GAN(5000,40,5)
```