Walid GHETTAS EISE-4

Compte-rendu TP1 bruit à l'analyseur de spectre

1. L'instruction qui permet de trouver z_12 est:

   z_12 = x_1 * weights[2] + x_2 * weights[3] + biases[1].

   Nous procédons à un simple affichage ensuite:

   print('The weighted sum of the inputs at the second node in the hidden layer is {}'.format(z_12))

   Ainsi, lors du test, nous obtenons:

```
[ 0.45  0.88  0.56  0.86  0.72  0.95]
[ 0.51  0.55  0.49]
x1 is 0.5 and x2 is 0.85
The weighted sum of the inputs at the first node in the hidden layer is 1.483
The weighted sum of the inputs at the second node in the hidden layer is 1.5610000000000002
```

2. En suivant le modèle énoncé à la question 1 pour a_11, nous calculons également l'activation du deuxième nœud,1,2, dans la couche masqué:

   a_12 = 1.0 / (1.0 + np.exp(-z_12))
   print('The activation of the second node in the hidden layer is {}'.format(np.around(a_12, decimals=4)))

   Ainsi, lors du test, nous obtenons:

```
[ 0.84  0.5   0.74  0.09  0.15  0.39]
[ 0.67  0.02  0.2 ]
x1 is 0.5 and x2 is 0.85
The weighted sum of the inputs at the first node in the hidden layer is 1.5150000000000001
The weighted sum of the inputs at the second node in the hidden layer is 0.4665
The activation of the first node in the hidden layer is 0.8198
The activation of the second node in the hidden layer is 0.6146
```

3. Maintenant, ces activations serviront d'entrées à la couche de sortie. Nous calculons donc la somme pondérée de ces entrées dans le nœud de la couche en sortie. Nous ajoutons au code:
   z_2 = a_12 * weights[5] + a_11 * weights[4] + biases[2]

print('The weighted sum of the inputs at the exit lawyer is {}'.format(z_2))

Ainsi, lors du test, nous obtenons:

```
[ 0.96  0.03  0.32  0.66  0.1   0.82]
[ 0.23  0.14  0.89]
x1 is 0.5 and x2 is 0.85
The weighted sum of the inputs at the first node in the hidden layer is 0.7354999999999999
The weighted sum of the inputs at the second node in the hidden layer is 0.8610000000000001
The activation of the first node in the hidden layer is 0.676
The activation of the second node in the hidden layer is 0.7029
The weighted sum of the inputs at the exit lawyer is 1.5339541280776035
```

4. Pour finir, nous calculons la sortie du réseau après l'activation du noeud dans la couche de sortie. Nous concluons le code par:

   a_2 = 1.0 / (1.0 + np.exp(-z_2))
   print('The activation of the node in the output layer is {}'.format(np.around(a_2, decimals=4)))

   Ainsi, lors du test, nous obtenons:

```
[ 0.61  0.79  0.11  0.58  0.68  0.15]
[ 0.78  0.52  0.14]
x1 is 0.5 and x2 is 0.85
The weighted sum of the inputs at the first node in the hidden layer is 1.7565
The weighted sum of the inputs at the second node in the hidden layer is 1.068
The activation of the first node in the hidden layer is 0.8528
The activation of the second node in the hidden layer is 0.7442
The weighted sum of the inputs at the exit lawyer is 0.8315165798676412
The activation of the node in the output layer is 0.6967
```

5. En utilisant la fonction initialize_network pour créer un réseau qui prend 5 entrées, qui a trois couches cachées (avec 3 nœuds dans la 1ère couche, 2 nœuds dans la 2ème couche et 3 nœuds dans la 3ème couche) et a 1 unique nœud dans la couche de sortie, nous obtenons l'affichage suivant définissant le réseau appelé small_network dans le code.

```
{'layer_1': {'node_1': {'weights': array([ 0.84,  0.91,  0.61,  0.68,  0.77]), 'bias': array([ 0.67])}, 'node_2': {'weights': array([ 0.7 ,
  0.11,  0.72,  0.48,  0.95]), 'bias': array([ 0.48])}, 'node_3': {'weights': array([ 0.38,  0.75,  0.87,  0.97,  0.23]), 'bias': array([ 0
.37])}}, 'layer_2': {'node_1': {'weights': array([ 0.03,  0.32,  0.55]), 'bias': array([ 0.52])}, 'node_2': {'weights': array([ 0.12,  0.66
,  0.4 ]), 'bias': array([ 0.38])}}, 'layer_3': {'node_1': {'weights': array([ 0.18,  0.15]), 'bias': array([ 0.7])}, 'node_2': {'weights':
 array([ 0.64,  0.66]), 'bias': array([ 0.41])}, 'node_3': {'weights': array([ 0.47,  0.92]), 'bias': array([ 0.01])}}, 'output': {'node_1'
: {'weights': array([ 0.74,  0.28,  0.35]), 'bias': array([ 0.96])}}}
```

   (*code à ligne 42 de tp1initiationDuNeurone.py*)

6. En utilisant la fonction compute_weighted_sum pour calculer la somme pondérée au premier nœud de la première couche masquée, nous obtenons l'affichage suivant:

```
The inputs to the network are [ 0.08  0.78  0.44  0.72  0.98]
[ 2.639]
```

   (*code à ligne 45 de tp1initiationDuNeurone.py*)

A partir de cette question, nous garderons les même entrées et nous les afficherons sur le terminal.

7. En utilisant la fonction node_activation pour calculer la sortie du premier nœud de la première couche masquée, nous avons le résultat suivant:

```
The inputs to the network are [ 0.08  0.78  0.44  0.72  0.98]
0.95212031015
```

(*code à ligne 51 de tp1initiationDuNeurone.py*)

8. On utilise la fonction forward_propagate pour calculer la prédiction de notre petit réseau (small_network), ainsi, nous obtenons:

```
The inputs to the network are [0.08 0.78 0.44 0.72 0.98]
The outputs of the nodes in hidden layer number 1 is [0.8863, 0.8548, 0.948]
The outputs of the nodes in hidden layer number 2 is [0.882, 0.8805]
The outputs of the nodes in hidden layer number 3 is [0.6305, 0.6791, 0.7237]
[0.718]
```

(*code à ligne 69 de tp1initiationDuNeurone.py*)

9. Enfin, nous concluons par la création de différents réseaux de neurones avec des structures différentes. On effectue des prédictions en utilisant la fonction forward_propagate:

```
The inputs to the network are [0.08 0.78 0.44 0.72 0.98]
The outputs of the nodes in hidden layer number 1 is [0.8572, 0.8237, 0.8082]
The outputs of the nodes in hidden layer number 2 is [0.6865, 0.7646]
The outputs of the nodes in hidden layer number 3 is [0.7647, 0.7408, 0.8596]
The outputs of the nodes in hidden layer number 1 is [0.8489, 0.8427]
The outputs of the nodes in hidden layer number 2 is [0.766, 0.8235, 0.7964]
The outputs of the nodes in hidden layer number 3 is [0.8828, 0.7911]
The predicted values by the network for the given input are [0.8534, 0.732, 0.7915]
```

(*code à ligne 72 de tp1initiationDuNeurone.py*)