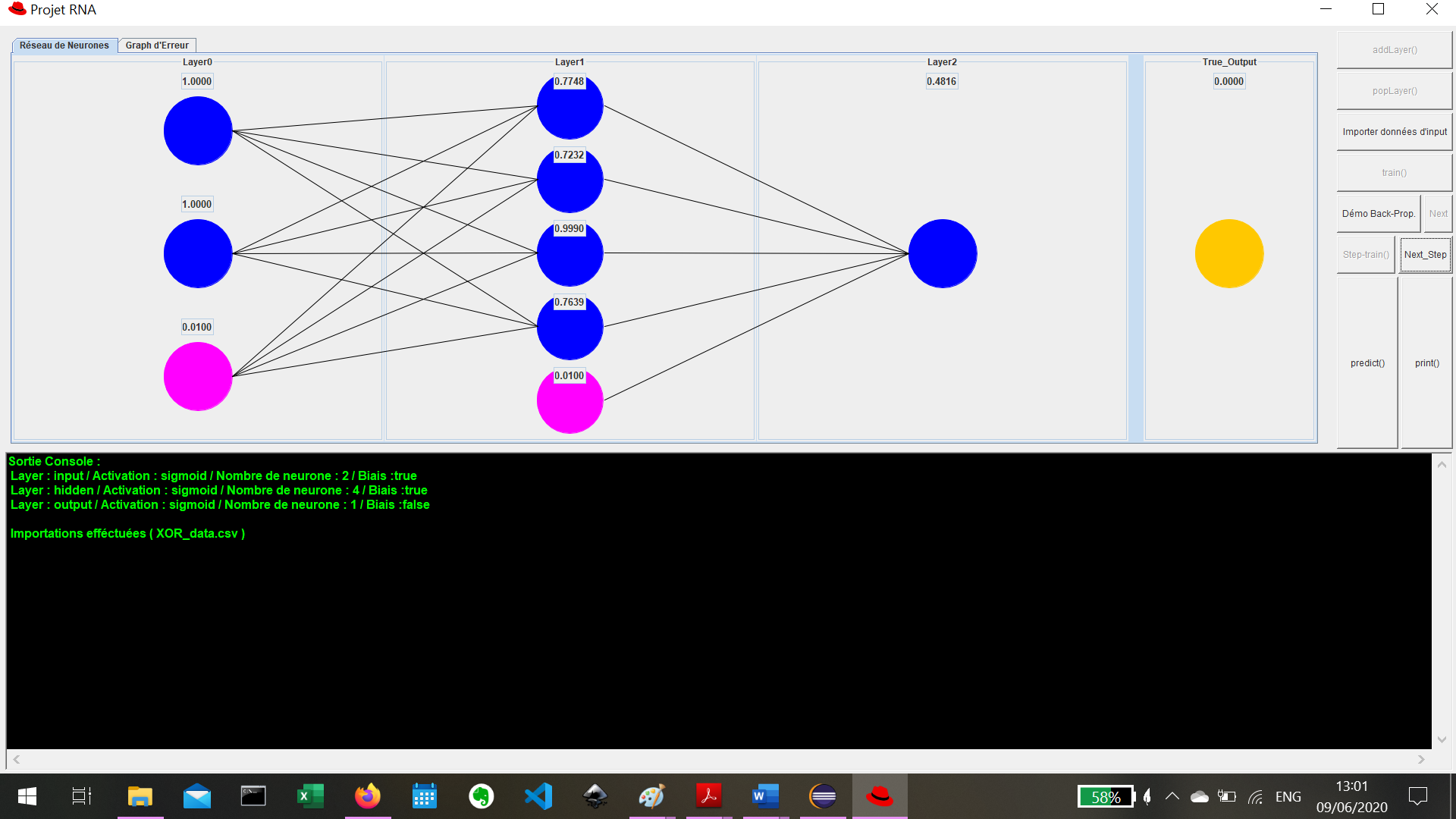


Dans l’exemple ci-dessus, on a un RNA de 3 couches avec 2 neurones **Biais** (dans les couches 0 et 1)

Les lignes noires représentent les liaisons entre neurones. Chaque ligne a un **poids**.  
  
On peut voir tous les poids d’un neurone en passant la souris sur les lignes.

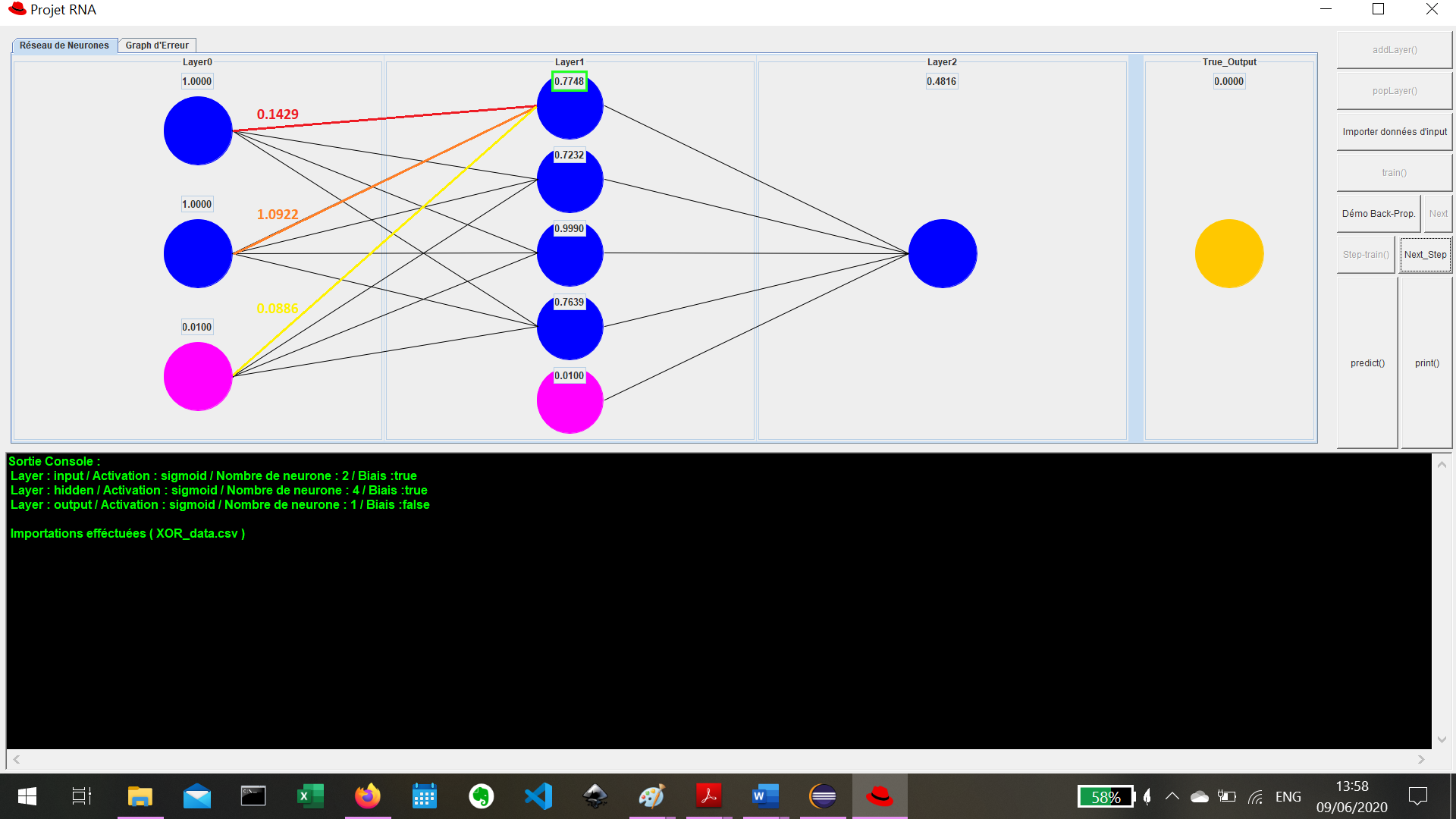


Pour chaque donnée d’entraînement, on effectue :

* La propagation en avant (*forward-propagation*)
* Le calcul de l’Erreur du Réseau pour cet exemple
* La propagation en arrière de l’Erreur (*back-propagation*)
* La mise à jour de tous les poids du Réseau

Les images suivantes expliquent brièvement chaque étape.

Les calculs sont tirés du chapitre 2 de ***Neural Networks and Deep Learning***de Michael Nielsen, qui explique beaucoup plus en détail le bref apperçu donné ici.



**Forward-propagation :**

Pour calculer l’activation de chaque neurone, on calcule en 1er le *weighted input* **z**. Pour le 1er neurone de la Layer1 :

**z** = (1.00 \* **0.1429**) + (1.00 \* **1.0922**) + (0.01 \* **0.0886**)  
 = 1.2359

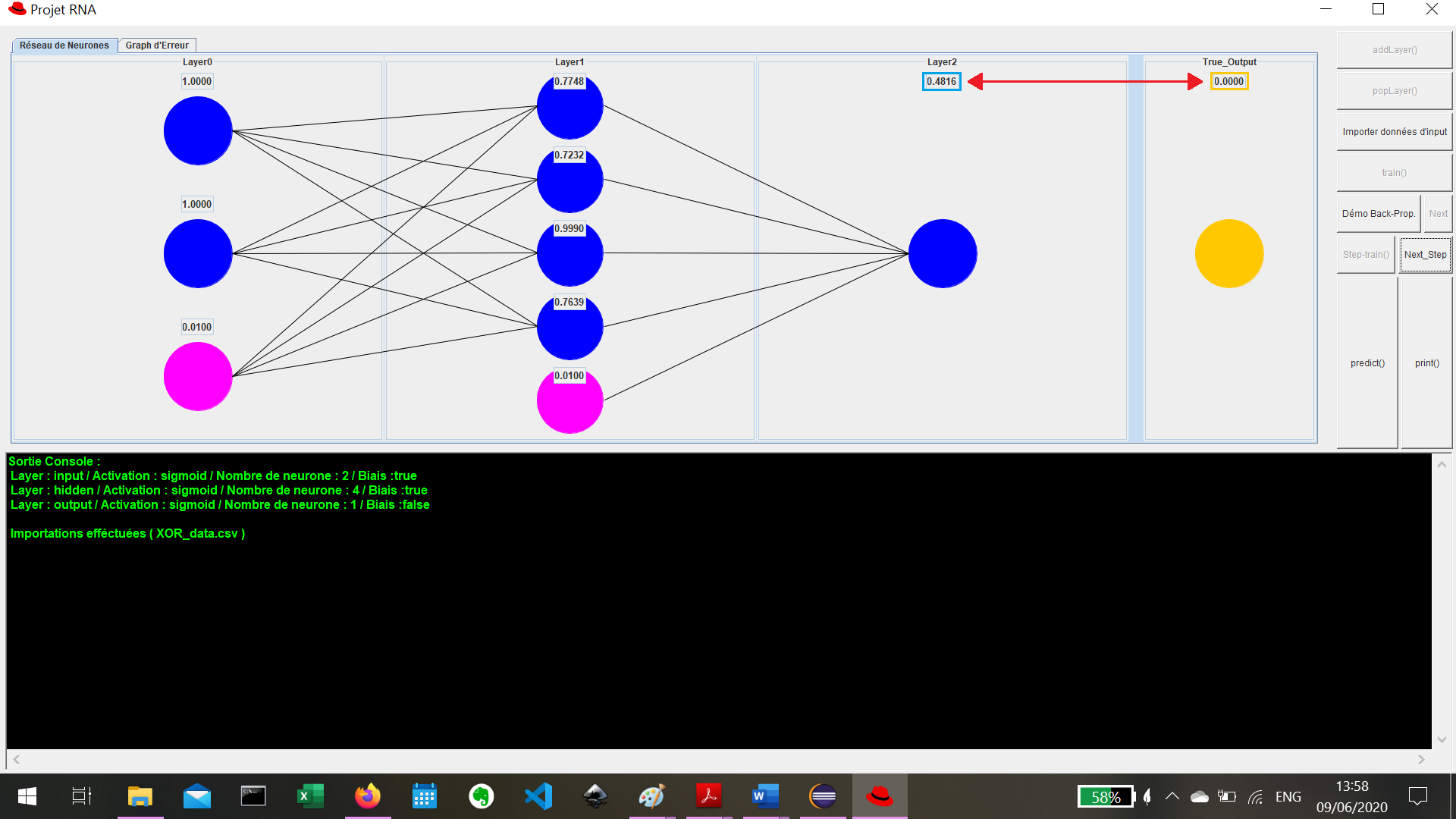
Ensuite, on applique à z la fonction d’activation choisie pour la couche ( ici sigmoid(x) ) :

activation = sigmoid(**z**)

= 1 / (1 + e**-z**)

= 0.7748

Tous les autres neurones sont calculées de la même façon.

****

**Calcul de l’Erreur :**

Il existe plusieures formules pour calculer l’erreur. Ici, on utilise le *Quadratic Cost Function* :

Cx = ‖**y(x)**-**aL**‖ **2**

y(x) est l’output réel de la donnée d’entraînement (ici 0.00), et aL est l’activation actuelle de la couche finale du Réseau (ici 0.4816). On a donc :

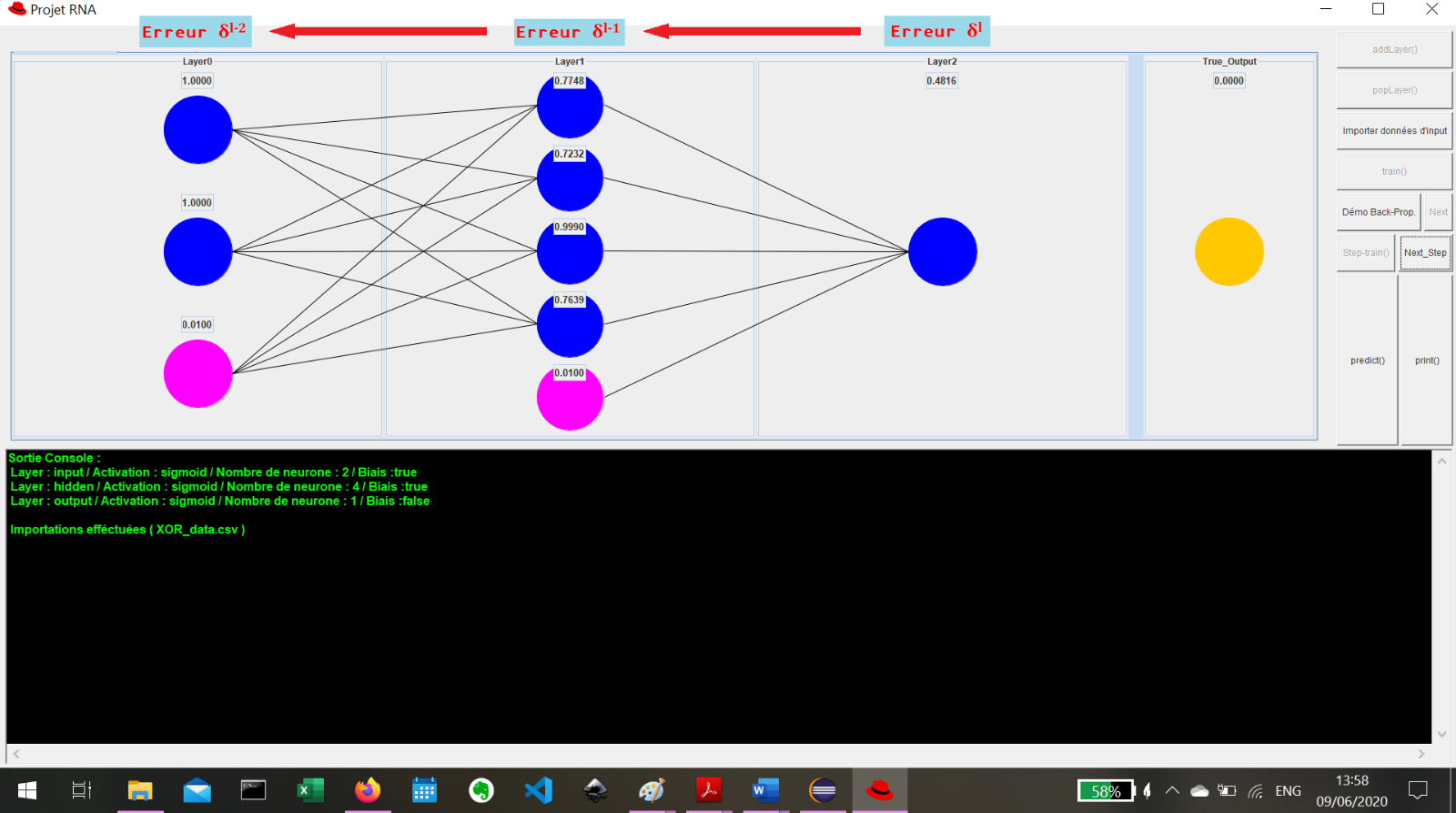
Cx = ‖**0.00**-**0.4816**‖ **2**

≈ 0.1160

Ce calcul est fait pour **toutes les données** du *Batch* : on rassemble le tout pour obtenir l’Erreur Moyenne du Réseau.  
C’est cette Erreur Moyenne, **nomée δ**, qu’on va propager en arrière (sur les couches précédentes).

En *batch Learning*, on mets à jour les poids du Réseau **après** avoir calculé l’Erreur Moyenne (de toutes les données).

D’autres techniques existent.  
Avec l’*Online Learning*, on propage en arrière l’erreur de **chaque donnée**, et on mets à jour les poids du Réseau à chaque fois.



**Back-propagation :**

On calcule l’erreur de chaque couche en partant de la dernière. La formule à nouveau varie selon la fonction d’activation.

δ**l** = ((w**l+1**)**T** δ**l+1**) ⊙ sigmoid′(z**l**)

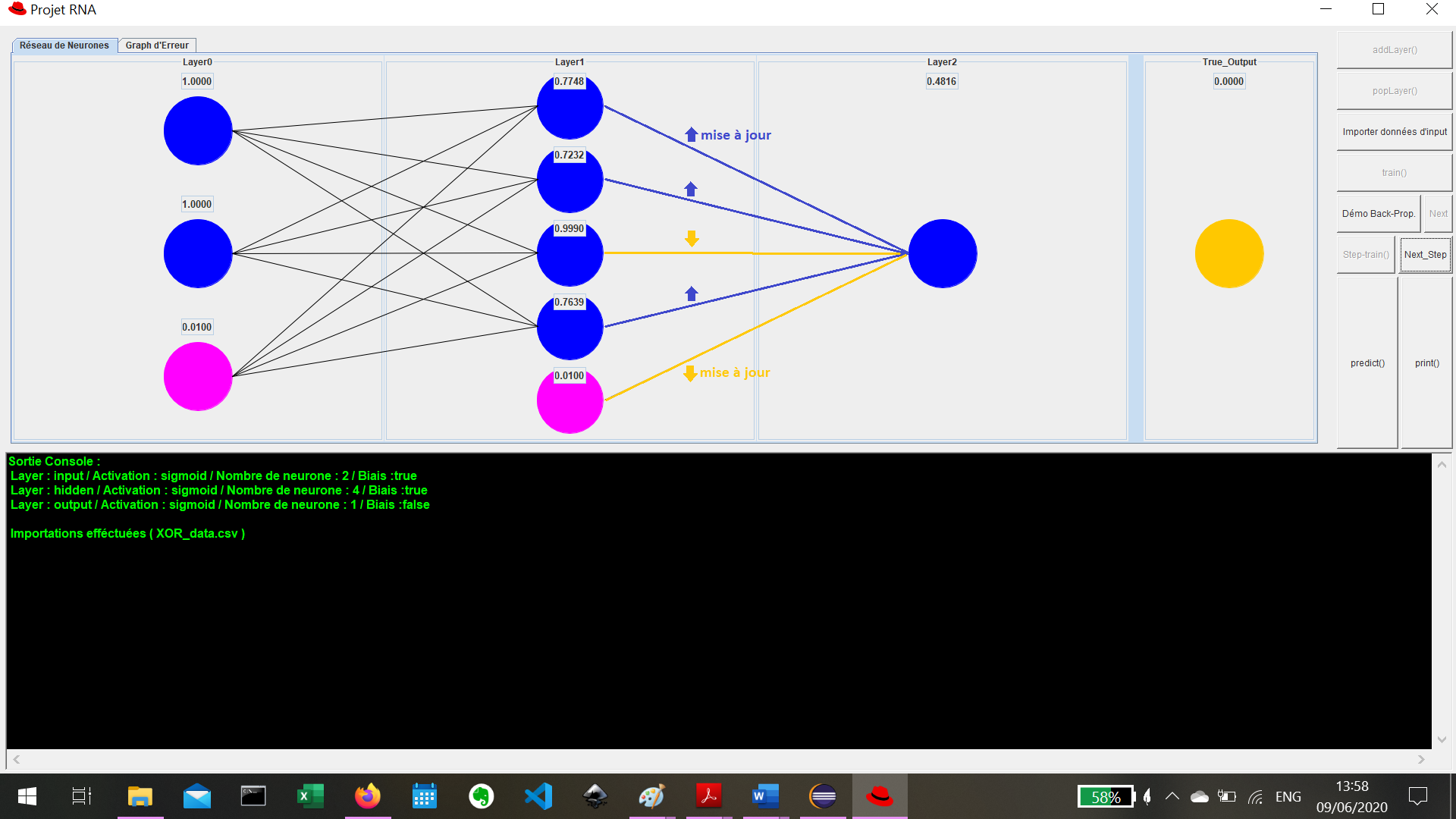
Cette formule générale utilise une représentation du réseau sous forme de matrices ( (wl+1)T = transposée de la matrice des poids de la couche l+1 ).  
Toutes les implémentations modernes de RNAs utilisent des calculs de matrices pour l’algorithme d’apprentissage.

sigmoid′ correspond à la dérivée de sigmoid.

Le ⊙ correspond au *produit de Hadamard* des 2 membres de la formule (tout simplement le produit terme-à-terme des 2 matrices).

L’important est de noter que l’erreur d’une couche δ**l** dépend de l’erreur de la couche suivanteδ**l+1**.

**Erreur δl-2**



**Mise à jour des poids :**

Une fois l’erreur de chaque couche calculée, on peut utiliser cette dernière pour mettre à jour les poids du Réseau.

Les matrices de poids w**l** sont mises à jour selon la formule :

w**l** = w**l** − \* ∑**x***δ***x,l**(a**x,l−1**)**T**

Les indices compliquent la formule, mais le principe est simple : on multiplie la matrice d’erreur de la couche par celle des activations de la couche (transposée).

Le tout est multiplié par .  
 correspond au taux d’apprentissage du Réseau. C’est une variable que l’on peut ajuster avant l’entraînement : plus est grand, plus la mise à jour des poids est importante. est simplement le nombre de données d’entraînement utilisées.

La somme obtenue est soustraite au poids actuel, ce qui nous donne le nouvel poids.

La mise à jour des poids de neurones Biais est très similaire :

b**l** = b**l** − \* ∑**x**δ**x,l**