Développement Logiciel, 2014

TP 2, Notions avancées et Exceptions

Dans le cadre de ce TP, on se propose de concevoir et programmer en Java un réseau de neurone artificiel. Un neurone artificiel est une unité de traitement très simple qui calcule une valeur de sortie à partir d'une valeur d'entrée selon une méthode de calcul précise. Un neurone artificiel est ainsi simplement définit comme suit :

- une valeur d'entrée (flottant),
- une valeur de sortie (flottant),
- une méthode permettant de calculer la valeur de sortie en fonction de l'entrée.

Il existe plusieurs méthodes de calcul pour obtenir la sortie, la plus classique étant d'utiliser une fonction de type sigmoïde (par exemple : tangente hyperbolique). Pour commencer, on ne vous donne que le code (incomplet) d'une classe MyNetwork :

```
public final class MyNetwork {

	public static void main(String[] args) {

		System.out.println("\nArtificial neurons.");

		Neuron node1 = new SigmoidNeuron();
		Neuron node2 = new SigmoidNeuron();
		Neuron node3 = new LinearNeuron();

		node1.setInputValue(4);
		node1.compute();
		node2.setInputValue(-3);
		node2.compute();
		node3.setInputValue(node1.getOutputValue()+node2.getOutputValue());
		node3.compute();

		node1.displayInfo();
		node2.displayInfo();
		node3.displayInfo();

		System.out.println("\nTerminate.");
	}
```

Question 1 : Arborescence de classes.

On vous demande d'ajouter les classes Neuron, SigmoidNeuron, LinearNeuron. Vous devrez choisir une arborescence de classe qui vous paraît cohérente et implémenter les méthodes nécessaires pour que MyNetwork.java puisse être compilé et exécuté.

En particulier, la classe Neuron devra définir des champs outputValue (accès restreint aux objets de la classe) et inputValue (accès restreint aux objets de la classe ou des classes héritées). On vous demande aussi d'être prudent lors de la conception (ie. interdire l'héritage et spécifier des classes abstraites quand c'est pertinent, etc.)

On souhaite que la méthode displayInfo() soit définie au niveau de la classe Neuron (renvoie la valeur de la sortie suivi d'un retour à la ligne - ex. : "0.5") et de ses classes dérivées (renvoie la valeur d'entrée précédée d'un texte identifiant le type de fonction appliquée par le neurone et suivi du symbole "=" - ex. : "sigmoide(0.5) = "). La sortie attendue, à partir du code donnée au début de ce sujet, et la suivante :

```
Artificial neurons.
sig(4.0) => (0.999329299739067)
sig(-3.0) => (-0.9950547536867305)
lin(0.00427454605233657) => (0.00427454605233657)

Terminate.
```

Question 2 : Découpage en package

On vous demande maintenant d'ajouter une classe qui n'apparaît pas ici du nom de Node, dont héritera la classe Neuron. La classe Node n'est pas utile pour l'instant et on vous demande juste de spécifier le fait que la méthode compute() doit être définie dans les classes héritant de Node.

On vous demande maintenant d'organiser votre projet en paquets, afin de marquer la différence entre le paquet node (qui contiendra éventuellement les champs et méthodes d'intérêt général), et le paquet node.neuron (qui contiendra tout ce qui concerne la gestion des neurones).

Question 3 : LogInterface

Vous devez maintenant ajouter un package interfaces dans lequel vous allez créer une interface LogInterface contenant la signature de la méthode displayInfo(). En effet, cette méthode est déjà définie pour tous les objets utiles de votre projet, et devrait continuer à l'être à l'avenir. Mettez à jour le reste de votre projet pour respecter la cohérence de l'existence de cette interface.

Question 4 : La classe Network

Une nouvelle classe Network doit maintenant être créée, qui appartient au package network. Cette classe va servir pour l'instant à contenir une liste de tous les noeuds crées. Cette

classe implémente l'interface LogInterface (displayInfo() appellera la méthode éponyme de chaque noeud enregistré), ainsi que les methodes addNode(Node node). Implémentez cette classe ainsi que les attributs nécessaires.

Mettez à jour MyNetwork.java pour ajouter les noeuds existant à un objet de type Network, et ajoutez un appel à displayInfo() de l'objet Network à la fin de la fonction main pour afficher l'état de tous les noeuds du réseaux.

Question 5 : Exceptions

On envisage maintenant d'ouvrir notre projet pour intégrer d'autres types de noeuds. En prévision de cela, la première étape consiste à préparer le terrain en sécurisant notre application :
concrètement, on souhaite mettre en place un mécanisme qui garantie que les noeuds manipulés sont compatibles entre eux. On fixe par défaut que les types hérités de Node sont compatibles s'ils appartiennent au même package (e.g. les différents types de neurones peuvent être stockés dans le même Network).

Pour cela, on souhaite :

- définir une méthode `public abstract Object getClassOfReference();` dans la classe `Node`.
- implémenter une vérification dans la classe Network lors de l'enregistrement d'un nouveau noeud. Il faut que le noeud nouvellement ajouté est la même parenté que ceux déjà enregistré dans l'objet Network (ie. uniquement des neurones). Si ce n'est pas le cas, addNode devra lancer une exception (qui sera traitée dans la fonction main). Cela sera utile pour la question suivante. On vous demande par ailleurs de tester si le type est connu (ie. pour prévoir le cas ou un nouveau package a été ajouté mais n'était pas prévu initialement par Network).

Question 6 : Le package Boolean Operators

On ajoute le code suivant dans la fonction main de MyNetwork, juste après la partie réseaux de neurones et avant l'affichage du texte "Terminate." :

```
(...)

// --- boolean operations ---

System.out.println("\nBoolean Operations.");

Network myCircuit = new Network();

TwoInputsNode node4 = new AndNode();
```

```
TwoInputsNode node5 = new OrNode();
TwoInputsNode node6 = new OrNode();
OneInputNode node7 = new NotNode();

try{
       myCircuit.addNode(node4);
       myCircuit.addNode(node5);
       myCircuit.addNode(node6);
       myCircuit.addNode(node7);
}
catch ( Exception e )
{
       System.out.println("[error] adding nodes to circuit.");
}

boolean in1 = true;
boolean in2 = false;

node4.setInputValues(in1, in2);
node4.compute();
node5.setInputValues(in1, in2);
node5.compute();
node6.setInputValues(node4.getOutputValue(),node5.getOutputValue());
node6.compute();
node7.setInputValue(node6.getOutputValue());
node7.compute();

myCircuit.displayInfo();

(...)
```

Il s'agit d'ajouter un nouveau package node.booleanOperator, qui contiendra tous les objets nécessaires pour faire des opérations booléennes. En particulier, on souhaite créer une classe BooleanOperatorNode ainsi que toutes la hiérarchie nécessaire pour créer les classes utilisées dans l'exemple.

Question 7 : Etendre la classe Network [**question libre**]

On souhaite pouvoir encapsuler dans l'objet Network toutes les informations pour connecter des opérateurs les uns aux autres (ie. la "topologie" d'un réseau). Ceci devrait permettre de réutiliser facilement un circuit donné. A titre d'exemple, on souhaite que le pseudo-code suivant soit utilisable dans la fonction main de MyNetwork :

```
...
Network net;
...
net.addNodeInput(node1);
net.addNode(node2); net.addNodeOutput(node3);
...
net.addLink(node1,node2);
...
net.setInputValues(...);
net.compute();
ArrayList outputValues = net.getOutputValues();
...
```

Vous pouvez observer que maintenant, un réseau (Network) permet d'enregistrer des noeuds comme noeud d'entrée ("input") et de sortie ("output"), ainsi que des liens entre noeuds. Ainsi, si le réseau est correctement définit, l'appel à la méthode "compute" doit simplement calculer l'état de l'ensemble des noeuds du réseau en respectant l'ordre de connexion.
a) Modifiez votre programme pour permettre de définir des liens entre les noeuds.
b) Modifiez votre programme pour permettre de définir des liens pondérés entre les neurones.
c) Construire un réseau capable de calculer (A xor B)
... avec les opérateurs booléen
... avec les opérateurs neuronaux
il faut trouver les poids - on considèrera que $x < 0 => false, x > 0 => true$
on modifiera la méthode addLink de Network : addLink($< node1 >, < node2 >, < poids >$) ;

Astuce #1 : on se limitera aux réseaux non récurrents (ie. pas de boucle)
Astuce #2 : il est plus facile de calculer la sortie du réseau en partant des sorties et en faisant des appels récursifs jusqu'à atteindre les entrées.