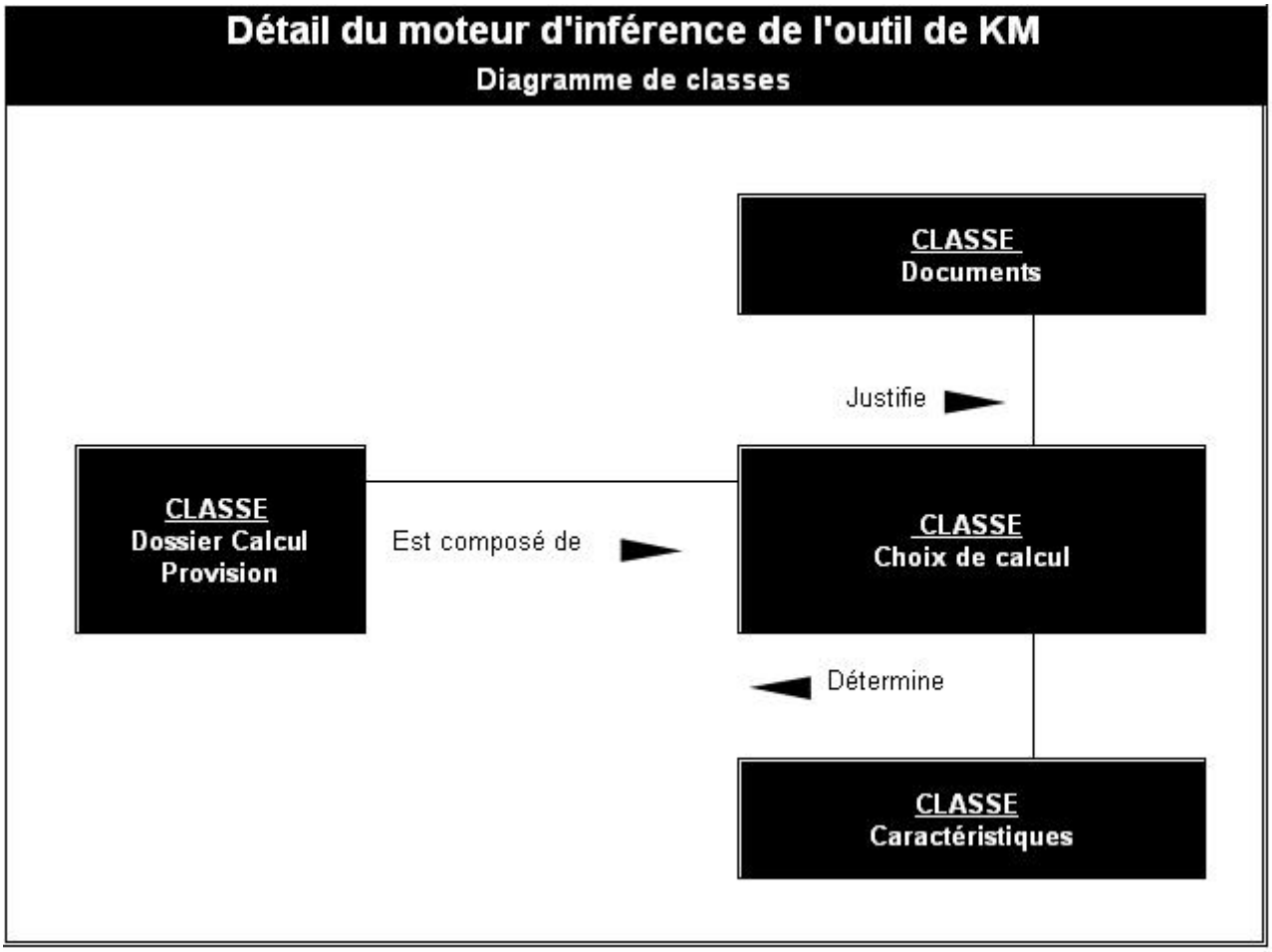
Description du principe d'inférence de l'outil de KM du projet :

Le principe d'inférence de l'outil de KM s'appuie sur les relations entre classes présentées dans la diagramme de classes UML. Le principe consiste à inférer les attributs de certaines classes, à partir des attributs de classes qui sont déjà renseignés par l'utilisateur.

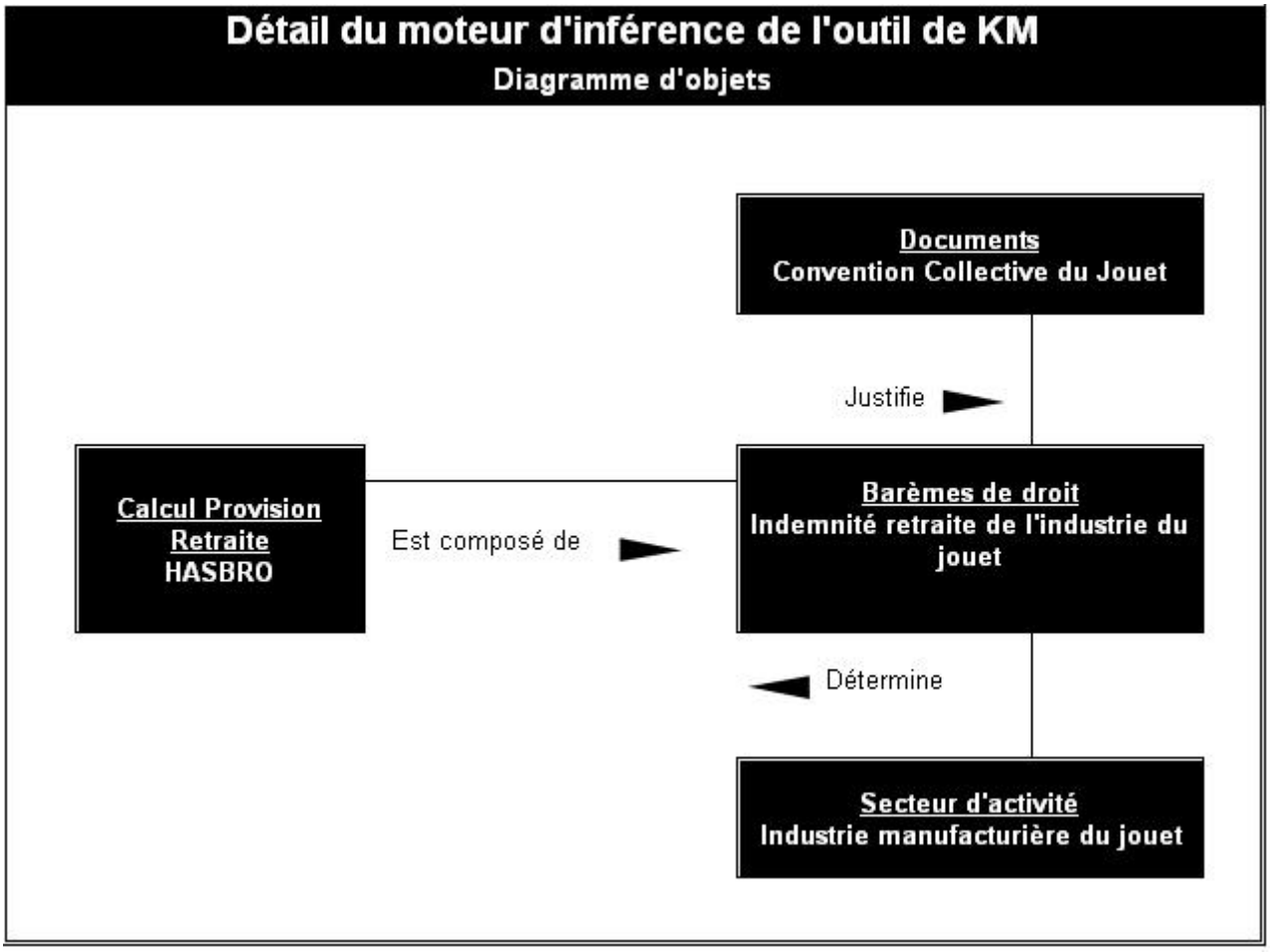
Pour mieux, se rendre compte de la logique à l'œuvre dans la manipulation des différents concepts, nous allons nous pencher sur le principe d'inférence entre un choix de calcul et des caractéristiques de l'entreprise renseignées par l'utilisateur.

Les liens entre les classes Choix de calcul et Caractéristiques au sein du diagramme UML sont les suivants :



Un Calcul de Provision est composé de choix de choix de calcul déterminé par les Caractéristiques de l'entreprise et justifié par les documents attestant des liens logiques entre les attributs des caractéristiques de l'entreprise et des choix de calcul de la provision qui sont effectués.

Pour décrire avec plus de détail et de clarté les principes d'inférence à l'œuvre dans l'outil de KM, nous allons nous pencher sur une inférence particulière.



On considère la société Hasbro qui est une entreprise fabriquant des produits manufacturés de type jouet. Pour calculer sa provision comptable d'indemnité de fin de carrière versée lors du départ à la retraite de ses salariés, il est nécessaire de connaître le barème des droits (généralement une table de correspondance entre ancienneté du salarié et montant de la prime).

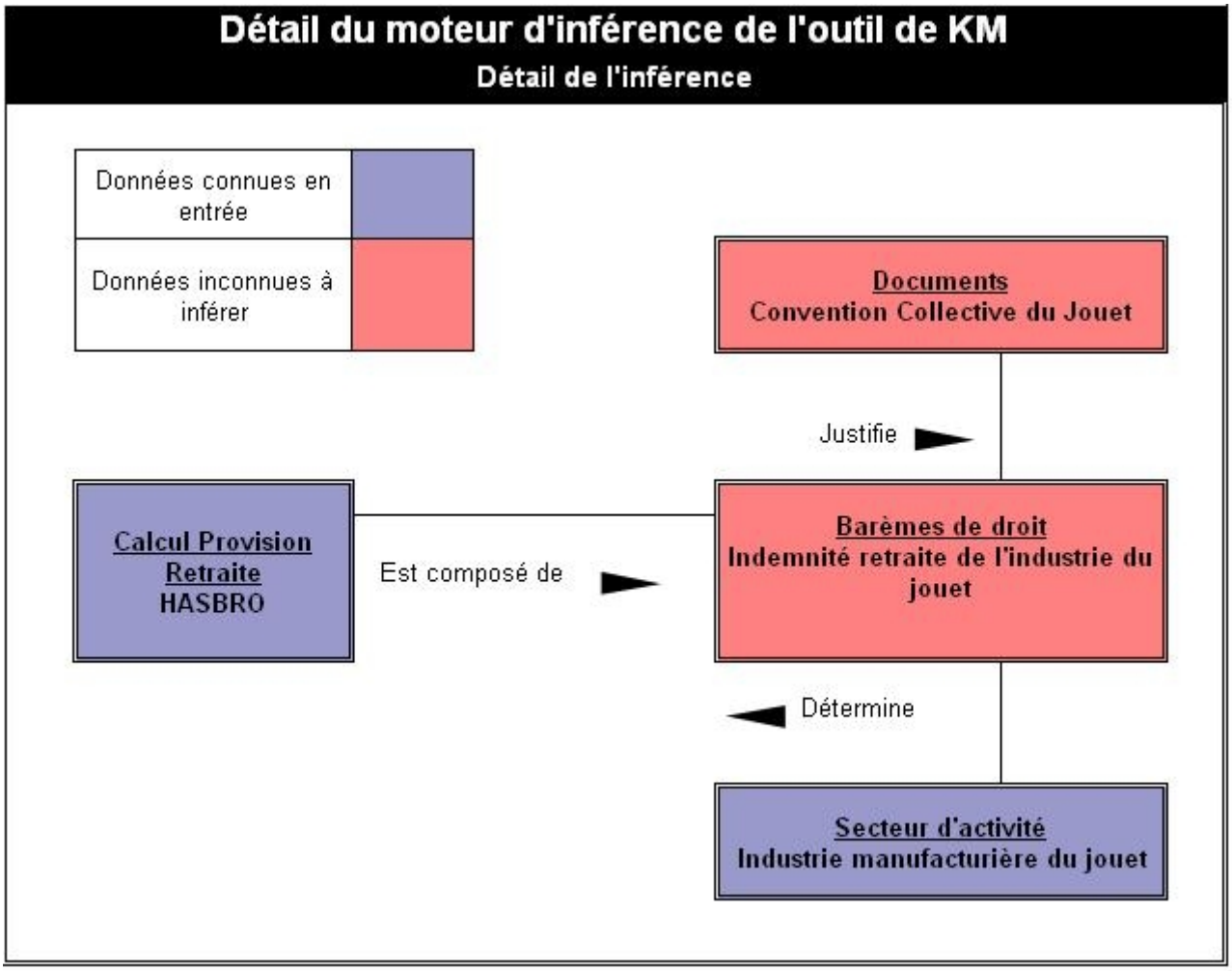
Pour déterminer ce barème, il est nécessaire de connaître la convention collective qui s'applique à l'entreprise. Cette convention collective se déduit à partir des caractéristiques de l'entreprise (secteur d'activité, taille de la population des salariés). Pour justifier le barème des droits, il sera nécessaire de lui adjoindre le document juridique de la convention collective choisie.

Les données en entrée du moteur d'inférence seront donc les suivantes :

- Type de Provision : Calcul de Provision de retraite,
- Secteur d'activité : Industrie manufacturière du Jouet,

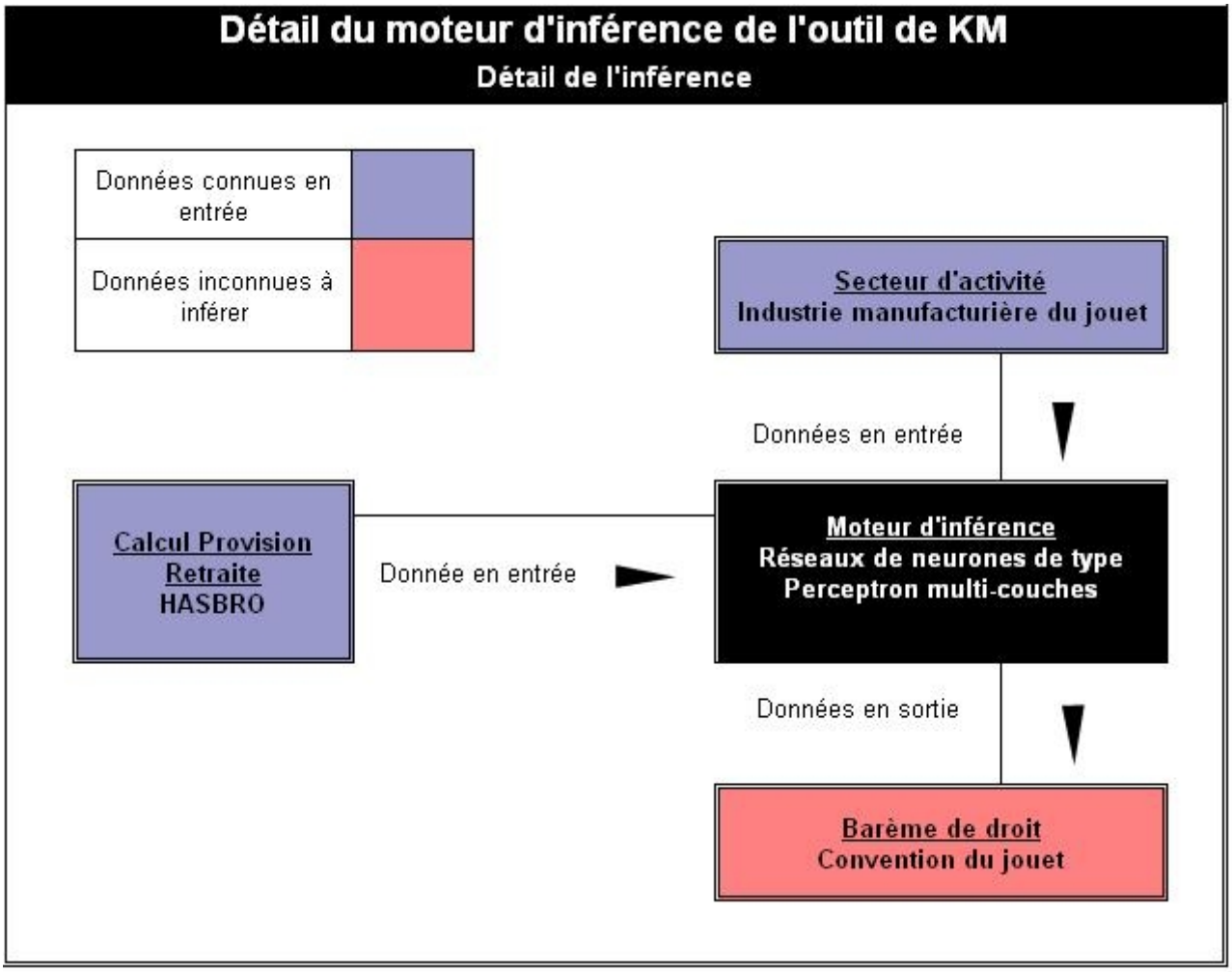
Les données en sortie seront les suivants :

- Document juridique : Convention collective du Jouet,
- Barème de droit : Indemnité de retraite de l'industrie du jouet,



Toute la question est de savoir, comment réaliser l'inférence qui nous permette d'arriver aux éléments de sortie que sont Barème IFC du Jouet et Document Convention collective du Jouet. Pour ce faire, nous allons disséquer les arcanes du moteur d'inférence.

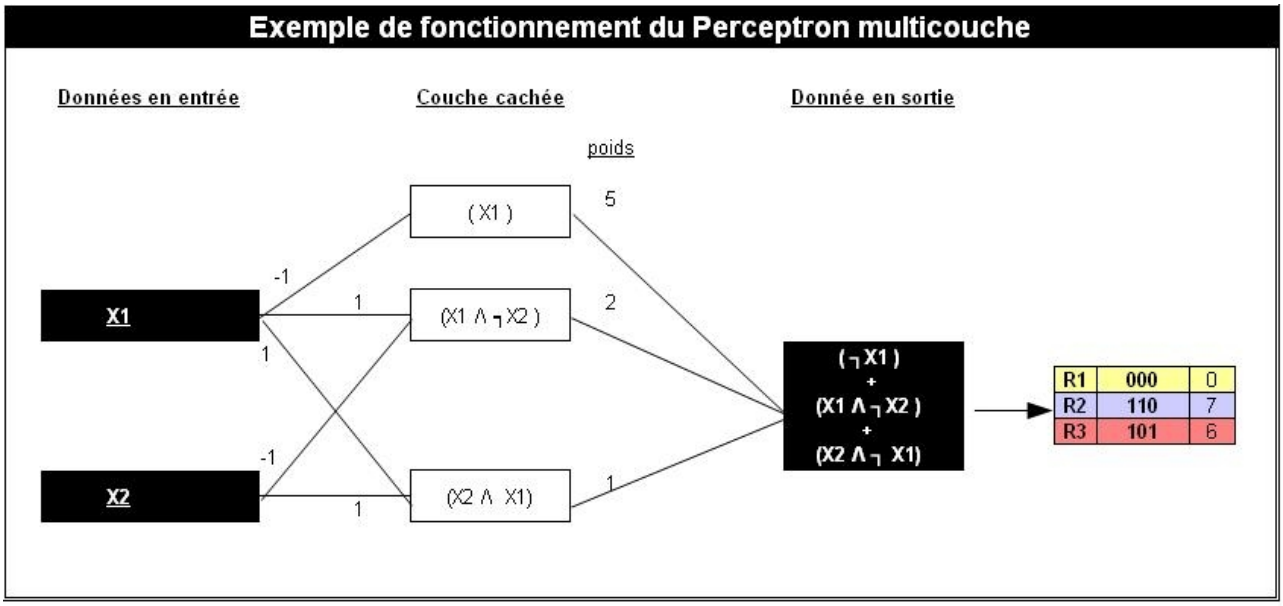
Le moteur d'inférence de l'outil de KM (proposition de documents, de paramètre et de tables) a été construit sur le modèle des réseaux de neurones. Plus précisément sur le modèle du réseau de neurone de type Perceptron multi-couche. Seul le module d'auto-contrôle a été construit sur le modèle logique du Raisonnement par cas, cependant il n'a jamais été complètement implémenté, donc nous ne détaillerons pas son mode de fonctionnement.



Pour mieux comprendre, les mécanismes à l'œuvre dans ce moteur d'inférence, il est nécessaire de rappeler le principe du réseau de neurones de type Perceptron multi-couches.

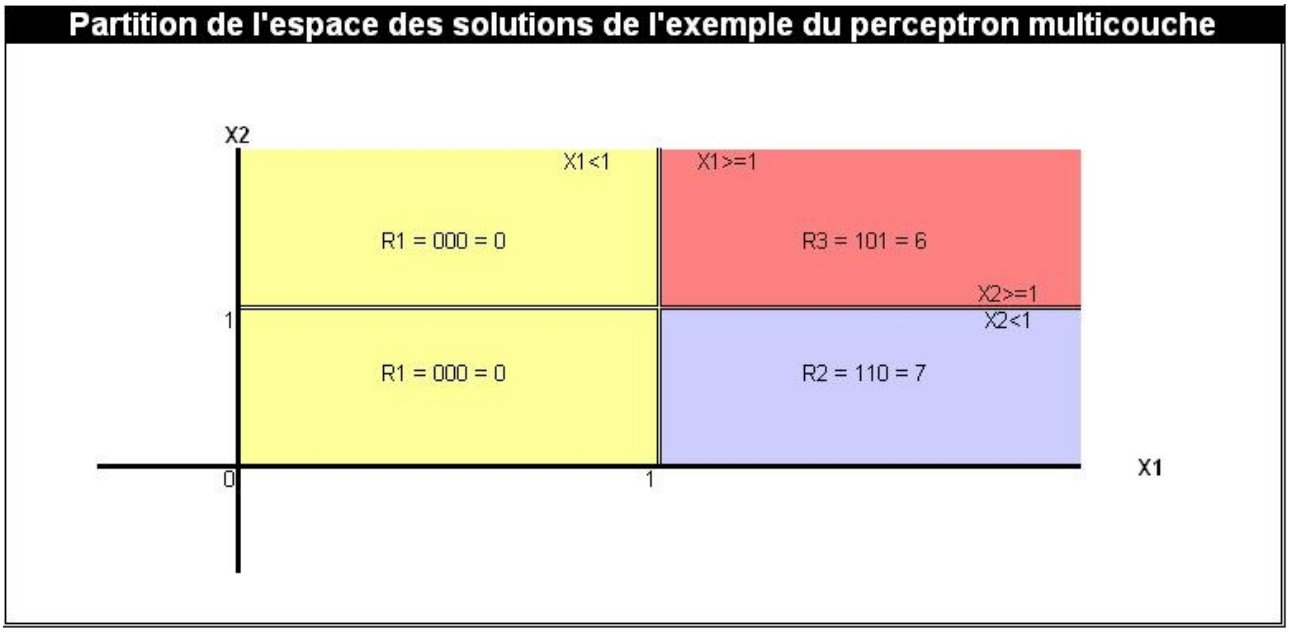
Le Perceptron est un type de réseau de neurones développé à l'origine par F. Rosenblatt. Il s'agit d'un modèle opérationnel apprenant qui s'appuie sur le fonctionnement psychologique du cerveau. Le perceptron est l'origine un système théorique très utilisé en neuroanatomie, et dont l'étude éclaire la biophysique des systèmes cognitifs. Un perceptron est composé de neurones d'entrée relié à un neurone de sortie, et couplés à des poids synaptiques, qui sont des poids de connexion, permettant de pondérer les signaux d'entrée.

Le Perceptron est susceptible d'apprentissage, c'est à dire qu'on peut le faire passer par une phase au cours de laquelle les poids synaptiques sont progressivement modifiés, pour que la sortie du réseau corresponde à la sortie désirée.



On parle de perceptron multi-couche, car on ajoute une couche de neurones entre la couche d'entrée et la couche de sortie. Cette couche est dite cachée car elle n'a pas de lien direct avec l'extérieur. L'utilisation de neurones cachées permet en fait de réaliser une classification quelconque. Pour chacune de ces classifications, il est possible de construire un neurone caché qui ne s'activera que pour une configuration spécifique des données d'entrée.

L'espace des solutions est partionné en plusieurs régions. Ces régions peuvent se coder avec plusieurs bits. Les neurones de la couche cachée réalisent ce codage et le neurone de sortie calcule la disjonction des bits à partir des poids synaptiques associés à chaque neurone caché. La couche cachée construit donc une représentation interne du problème posé et les unités de la dernière couche de réseau calculent la sortie finale, en décodant la classification réalisée par les unités cachées.



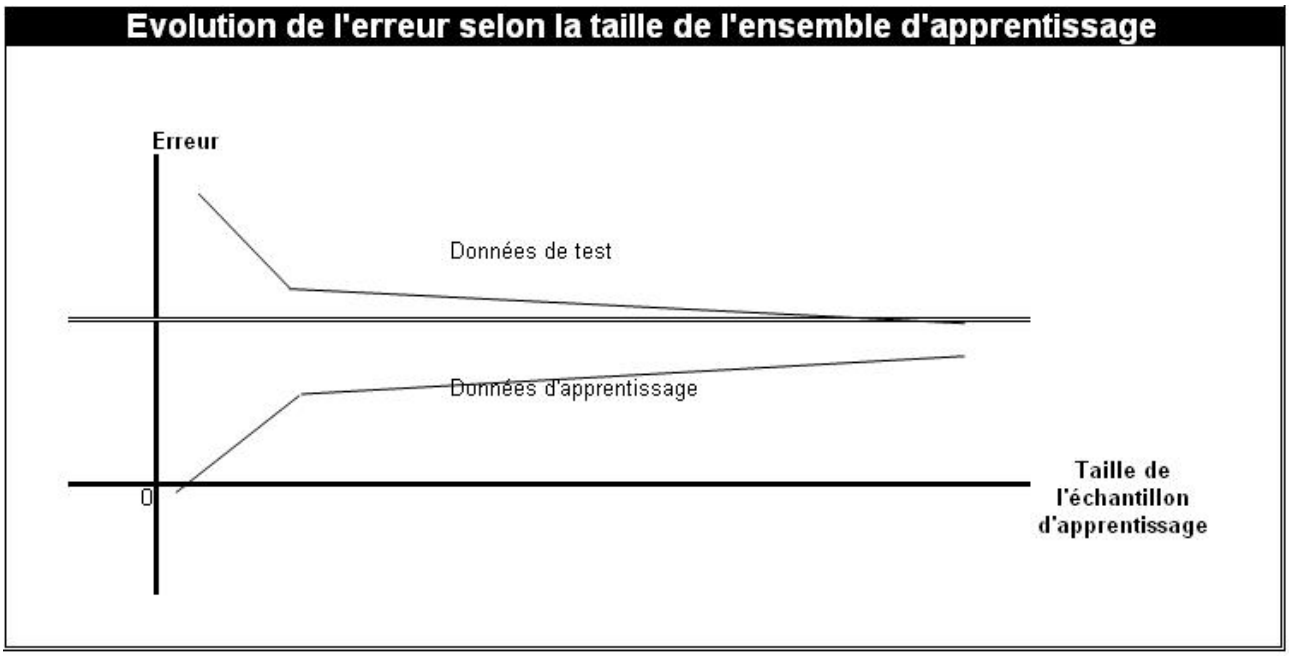
L'utilisation d'une couche cachée permet ainsi de définir des surfaces de décisions. La disjonction des séparateurs linéaires définis par trois neurones cachés permet ainsi de définir une surface de décision assurant une classification en 3 catégories possibles.

L'apprentissage supervisé du perceptron multi-couches permet de corriger les connexions en fonction de l'écart entre les sorties obtenues et souhaitées. Cette procédure d'apprentissage utilise l'algorithme du rétro-propagation du gradient, qui est une extension de la règle delta qui utilise une fonction sigmoïde de type $1/ (1 + e^{-cx})$ pour le calcul du signal de sortie.

Cet algorithme va consister à minimiser l'erreur quadratique sur la sortie et à chercher le minimum de l'équation suivante :

$E = ½ (d_k - s_k)^2$

avec d_k le signal désiré pour le neurone de sortie k
avec s_k le signal réel de sortie du neurone de sortie k



Se pose également le problème de la taille de l'ensemble des données d'apprentissage. Il est essentiel, si l'on veut atteindre une généralisation correcte que cet ensemble soit d'une taille adéquate. Si l'on utilise trop peu d'exemples, l'algorithme pourra sembler converger fortement (l'erreur totale sera faible), mais la généralisation sera mauvaise (l'erreur sur les données non apprises sera forte). De manière générale, si l'on augmente la taille de l'échantillon d'apprentissage, on augmente l'erreur sur cet échantillon; en revanche, on tend à réduire l'erreur sur les données non apprises. Ceci doit toutefois être nuancé par le fait qu'un trop grand nombre d'exemples augmente la sensibilité du réseau au bruit et nuit donc à ses capacités de généralisation.

Un méthode simple consiste à suivre la règle empirique de Widrow :

$N = W / \varepsilon$

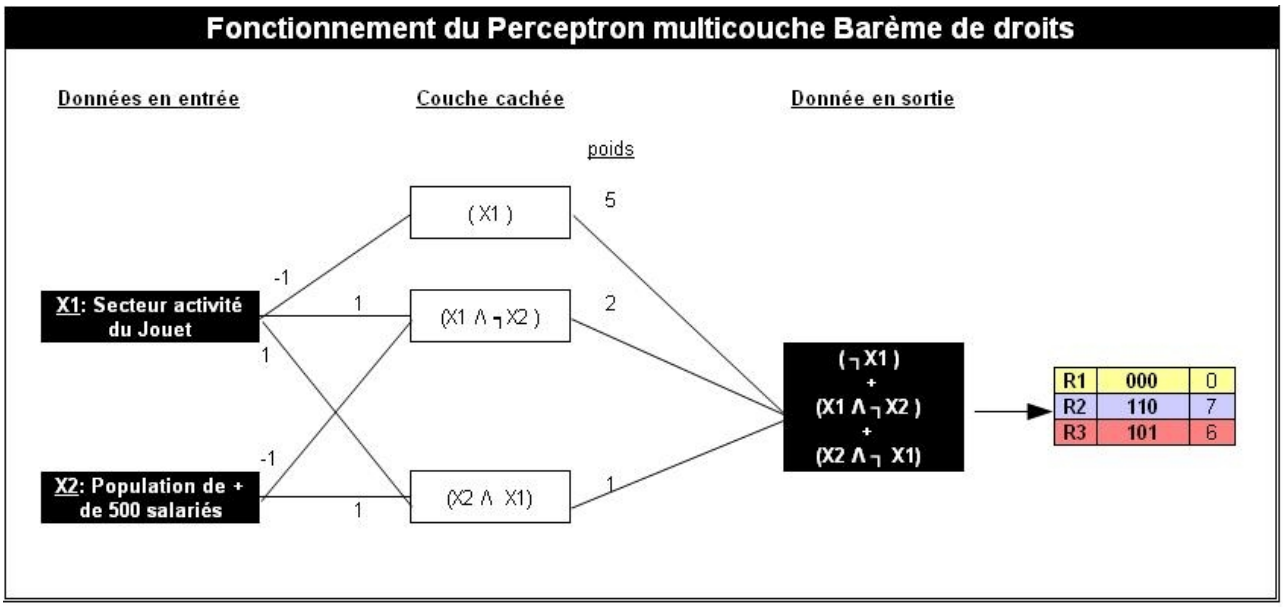
avec N nombre d'exemples
avec W nombre de paramètres et de connexions
avec ε l'erreur relative acceptée (exemple 10% d'erreur tolérée)

Ainsi, pour un taux d'erreur toléré de 10 % et 4 paramètres, on devra présenter 40 exemples au moteur d'inférence.

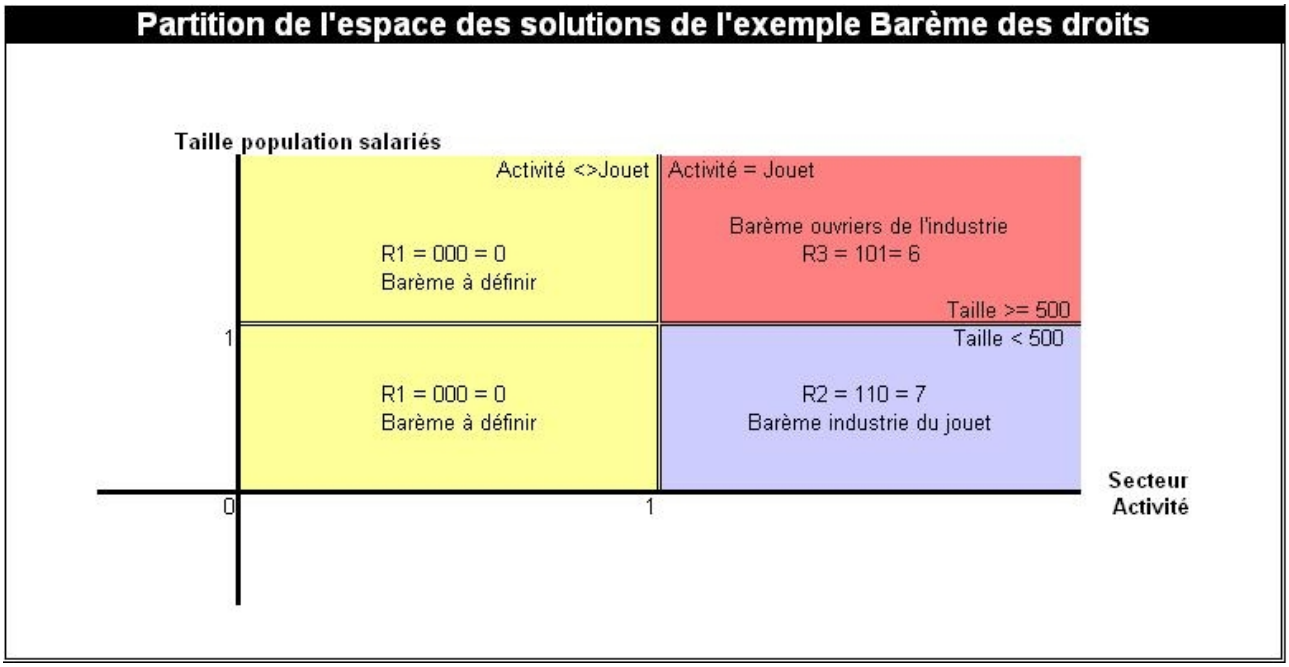
Pour mieux illustrer le fonctionnement de ce réseau de neurones de type perceptron multicouches, nous allons étudier le cas de l'inférence sur le barème des droits.

Les données en entrée sont :

- X1: Secteur d'activité du jouet
- X2 : Un nombre de salariés supérieur à 500



Avec en donnée de sortie, un neurone désignant la nature du barème des droits à choisir pour le calcul de la provision comptable de l'indemnité de fin de carrière :

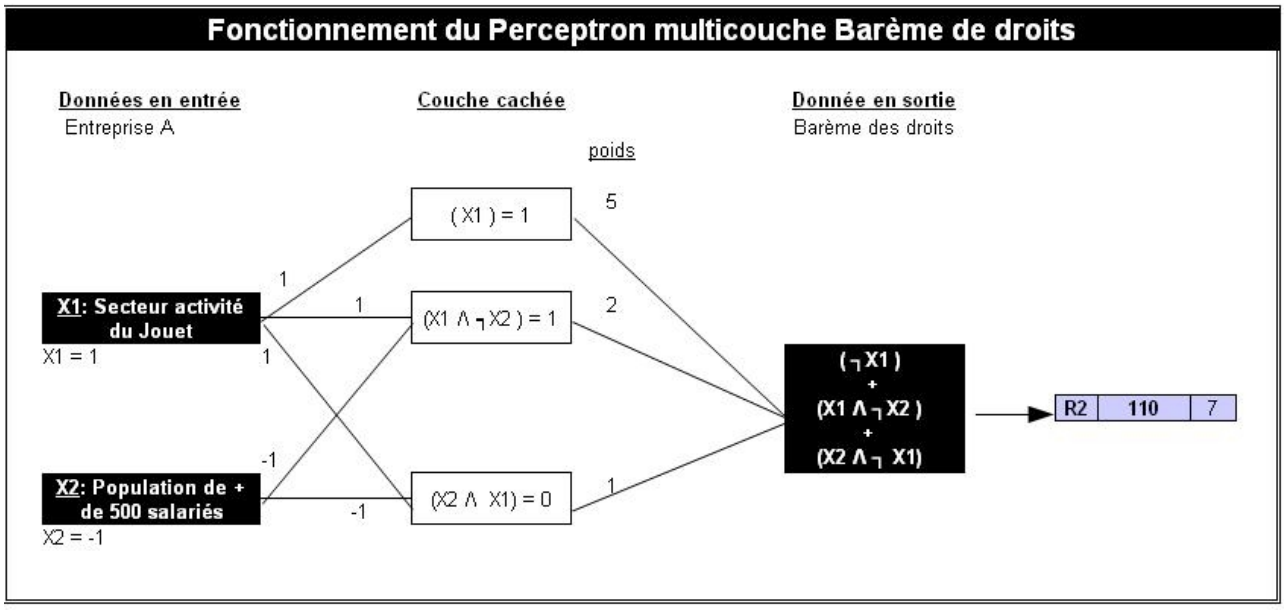


On considère pour 3 cas concrets différents :

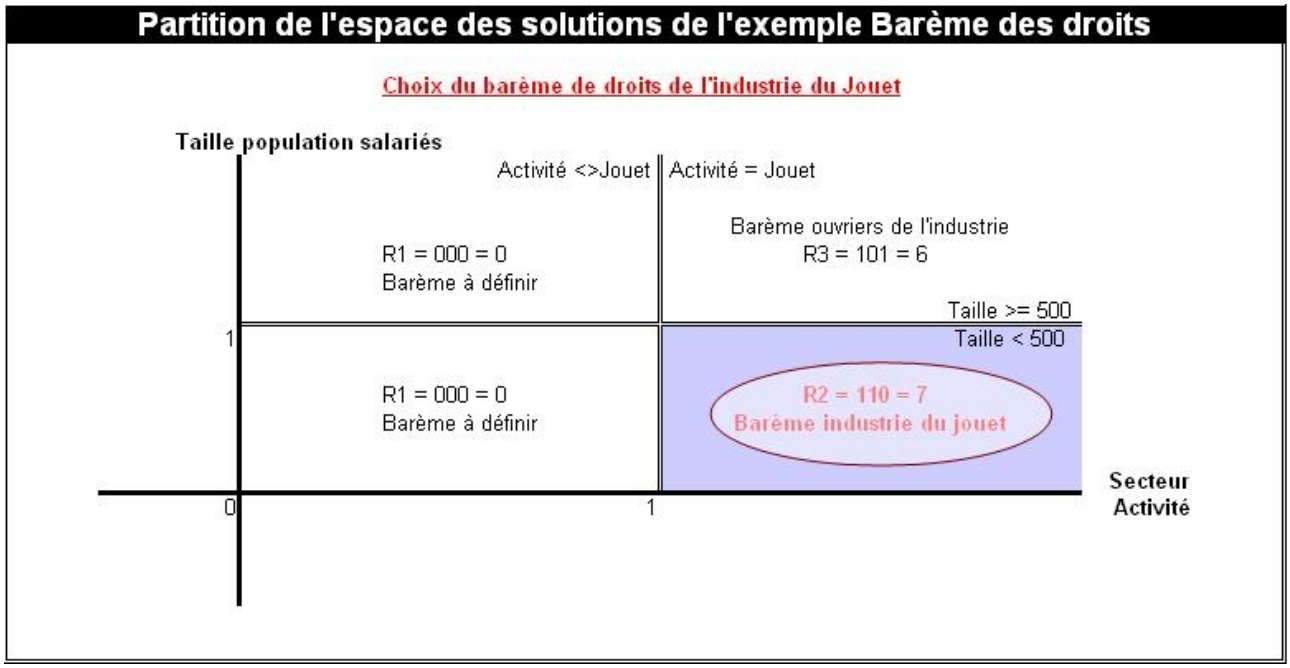
- 1er cas : une entreprise de 350 salariés du secteur d'activité du Jouet
- 2ème cas : une entreprise de 750 salariés du secteur d'activité du Jouet
- 3ème cas: une entreprise de 350 salariés du secteur d'activité agro-alimentaire

On rappelle que pour calculer une provision pour indemnité de fin de carrière, les règles comptables préconisent l'utilisation d'un barème de droit de la convention collective du Jouet lorsque le nombre de salariés est inférieur à 350 salariés et un barème de droit de la convention collective dues ouvriers de l'industrie, lorsque le nombre de salariés de l'entreprise est supérieur de 350 salariés.

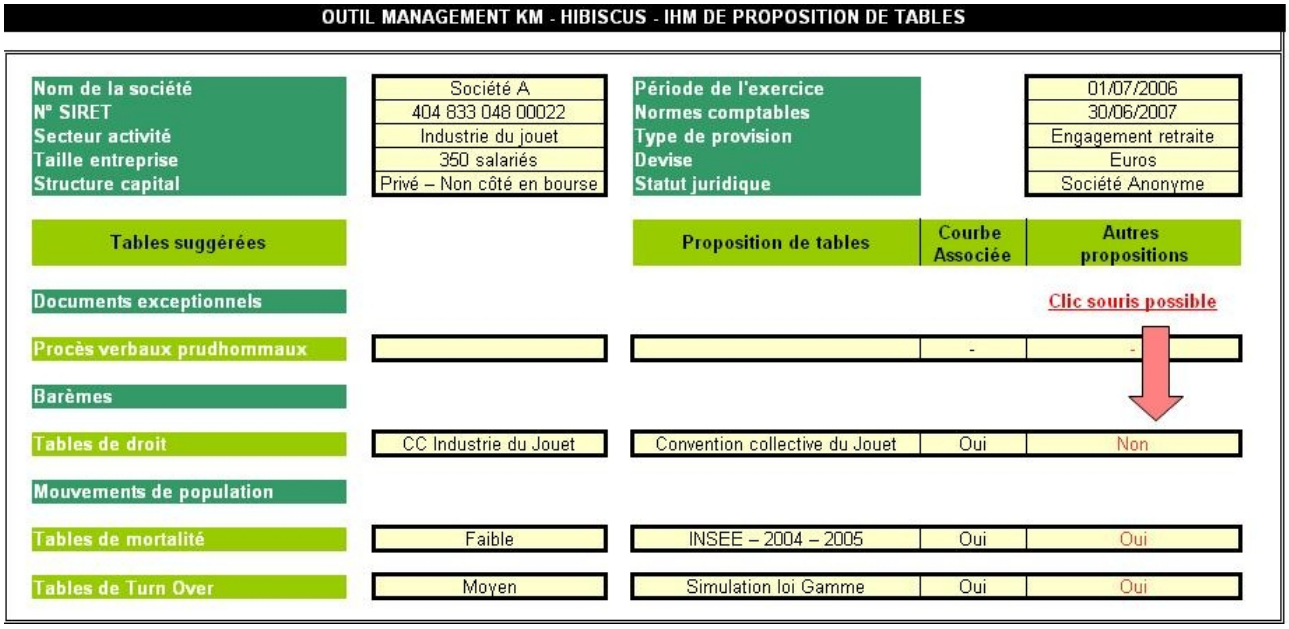
■ CAS N°1 : Entreprise du secteur d'activité du jouet et ayant une population salariés de 350 personnes :



Le moteur d'inférence donne en sortie un barème des droits correspondant à celui de l'industrie des jouets.



L'outil de KM propose alors la table de droits de la convention collective de l'industrie du jouet, par le biais de l'interface graphique suivant:

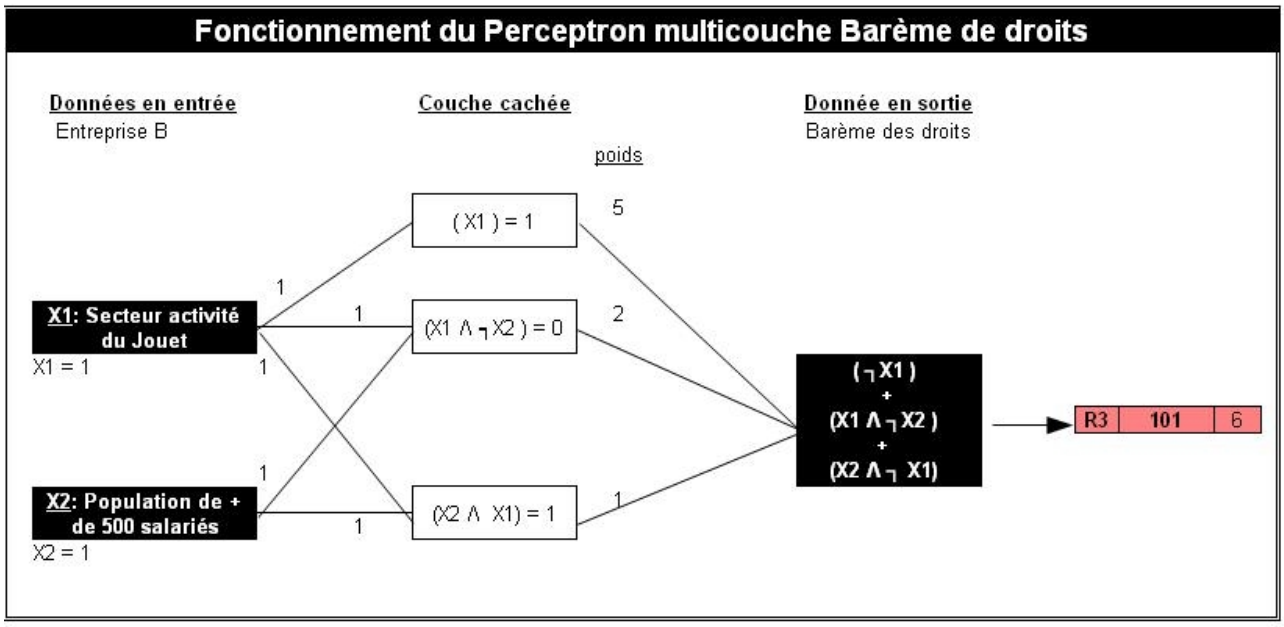


Cependant, l'utilisateur a la possibilité de choisir une autre table de droits, en cliquant sur bouton Autres propositions. En effet, il s'agit d'une proposition et il peut s'avérer qu'un autre barème de droits soit plus judicieux (notamment lorsque peu de dossiers similaires ont été enregistrés dans la base de l'outil de KM).

L'utilisateur devra alors choisir un barème indépendamment de toute proposition de l'outil, d'où la nécessité absolue que cette opération soit effectuée par un expert actuaire. En effet, si un barème de droits erroné est choisi, c'est tout la justesse du moteur d'inférence qui est menacé.

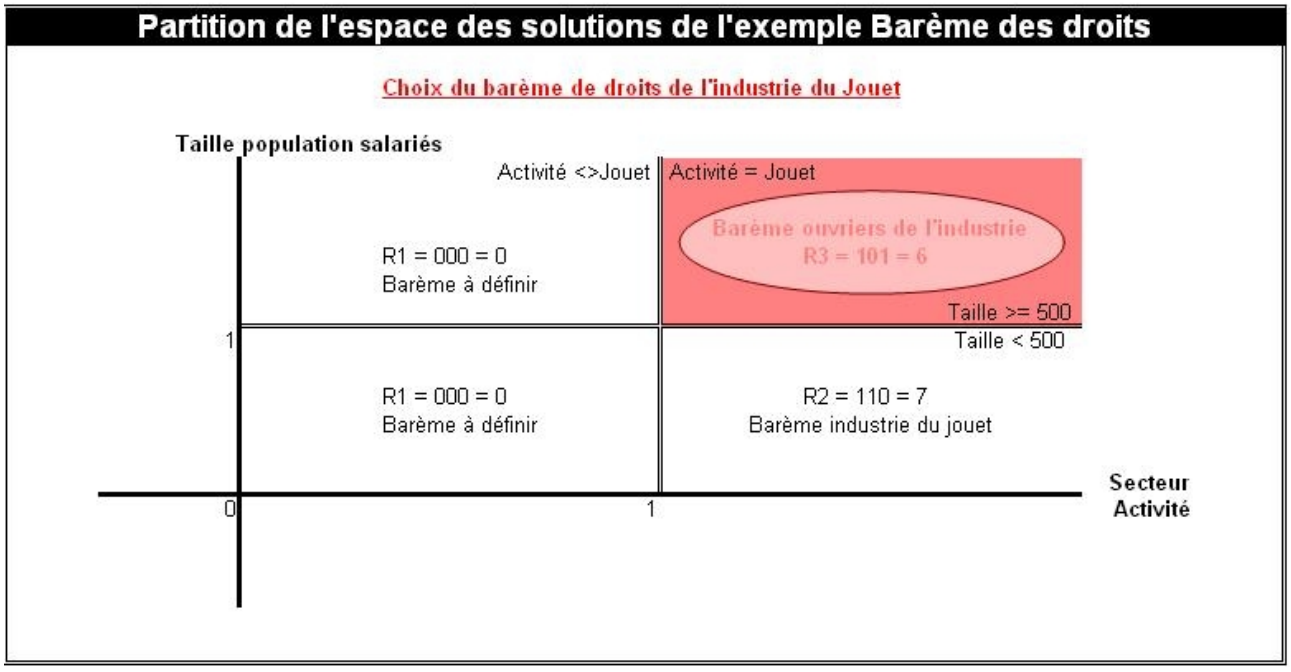
– 2ème cas : une entreprise de 750 salariés du secteur d'activité du Jouet

■ CAS N°2 : Entreprise du secteur d'activité du jouet et ayant une population salariés de 750 personnes :

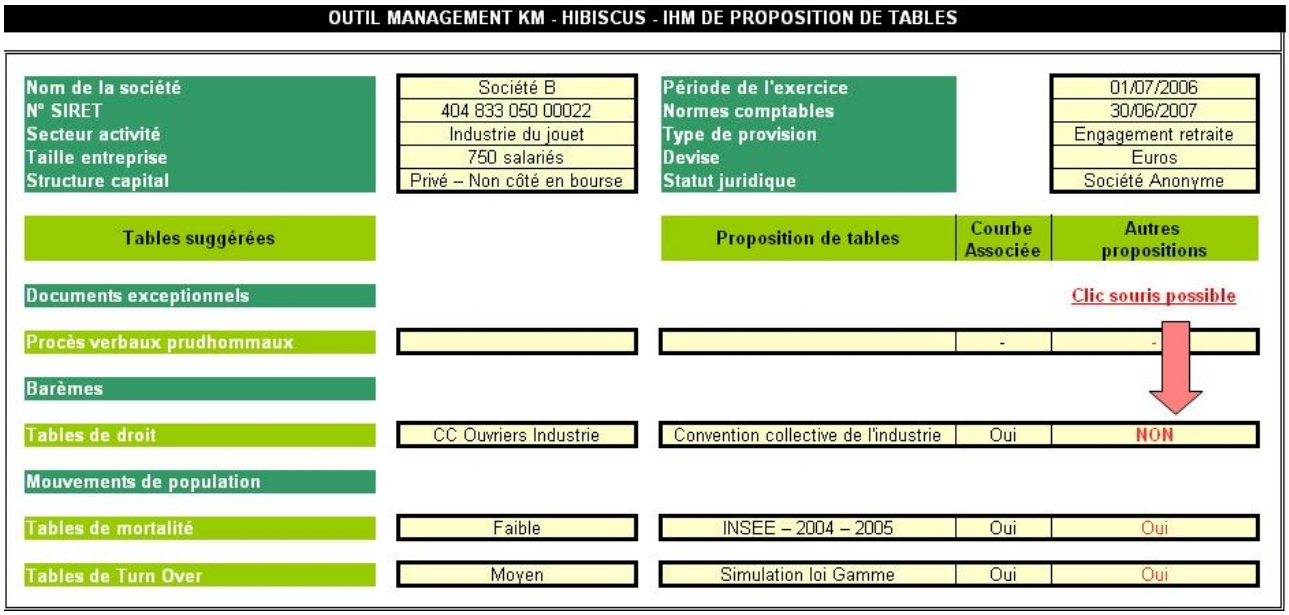


Le moteur d'inférence donne en sortie un barème des droits correspondant à celui de des ouvriers de l'industrie.

■ CAS N°2 : Entreprise du secteur d'activité du jouet et ayant une population salariés de 750 personnes :



L'outil de KM propose alors la table de droits de la convention collective des ouvriers de l'industrie, par le biais de l'interface graphique suivant:

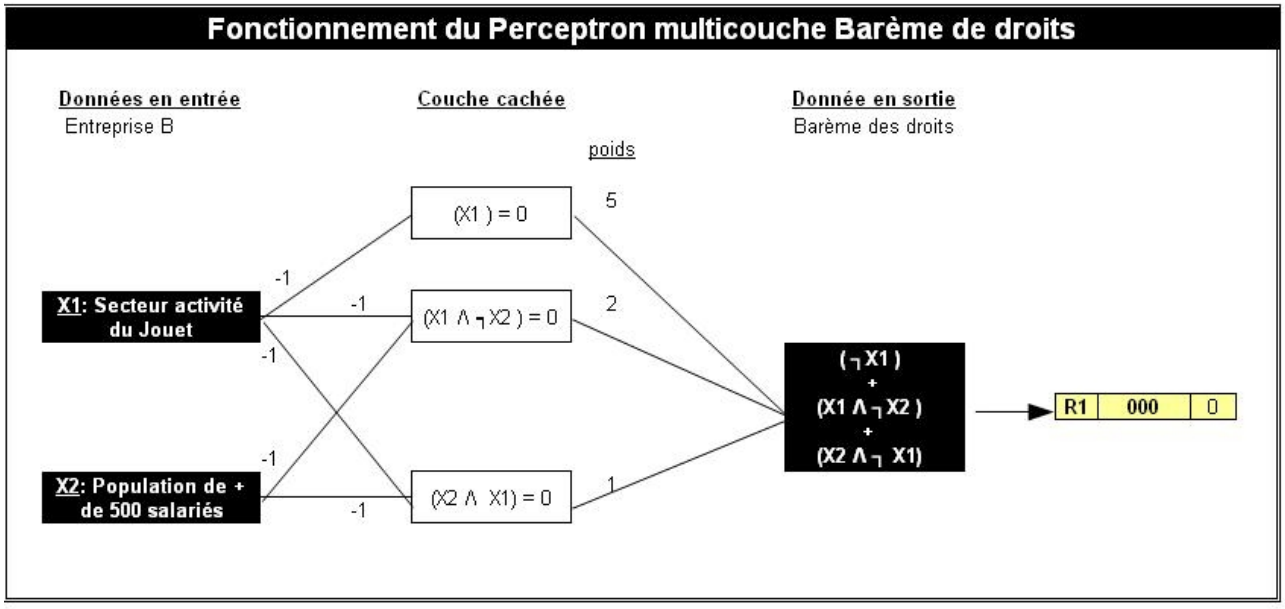


Cependant, l'utilisateur a la possibilité de choisir une autre table de droits, en cliquant sur bouton Autres propositions. En effet, il s'agit d'une proposition et il peut s'avérer qu'un autre barème de droits soit plus judicieux (notamment lorsque peu de dossiers similaires ont été enregistrés dans la base de l'outil de KM).

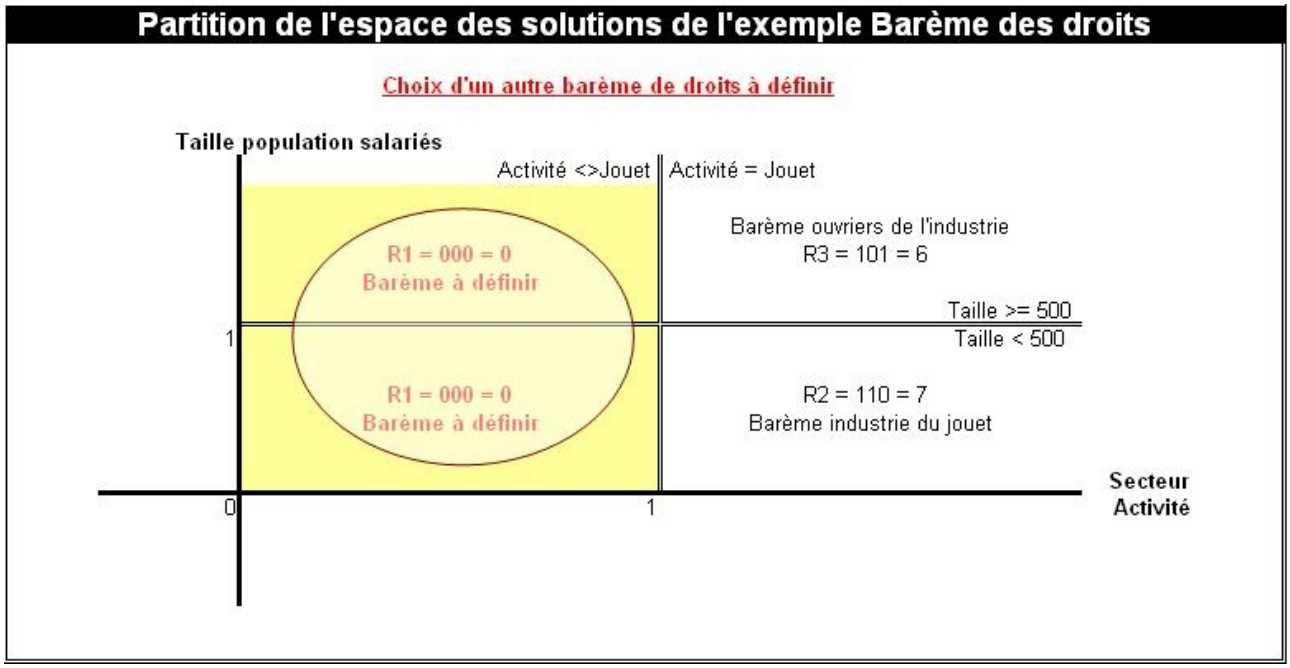
L'utilisateur devra alors choisir un barème indépendamment de toute proposition de l'outil, d'où la nécessité absolue que cette opération soit effectuée par un expert actuaire. En effet, si un barème de droits erroné est choisi, c'est tout la justesse du moteur d'inférence qui est menacé.

– 3ème cas: une entreprise de 350 salariés du secteur d'activité agro-alimentaire

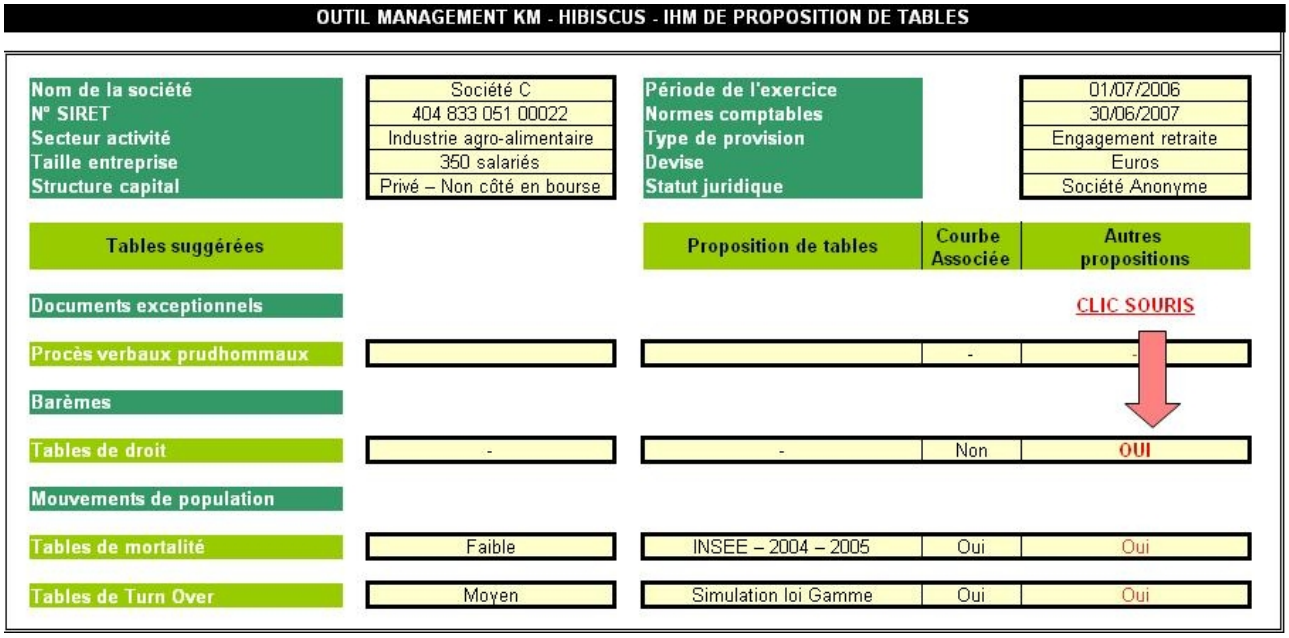
■ CAS N°3 : Entreprise du secteur d'activité de l'agro-alimentaire et ayant une population salariés de 350 personnes :



Le moteur d'inférence donne en sortie un barème des droits indéterminé à définir :



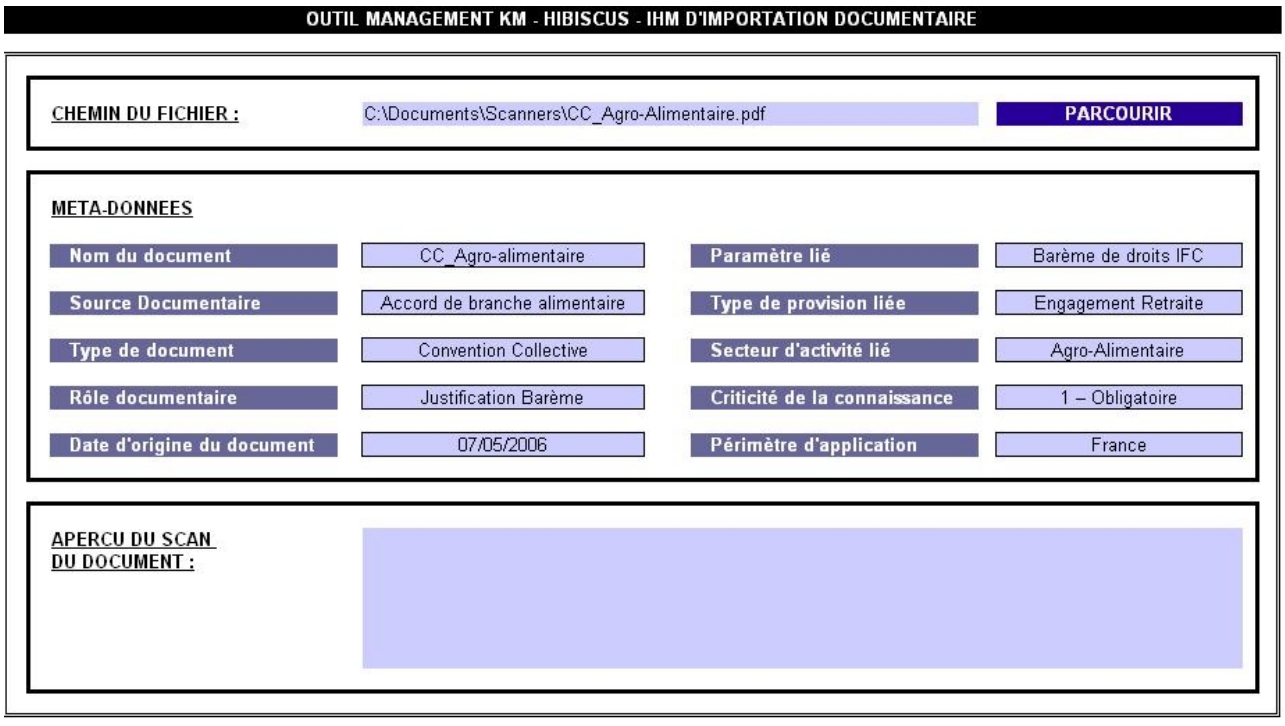
L'outil de KM ne propose aucune table de droits et affiche l'interface graphique suivante:



Cependant, l'utilisateur a alors la possibilité de choisir une autre table de droits, en cliquant sur bouton Autres propositions.

L'utilisateur devra alors choisir un barème indépendamment de toute proposition de l'outil, d'où la nécessité absolue que cette opération soit effectuée par un expert actuaire et non par un technicien. En effet, si un barème de droits erroné est choisi, c'est tout la justesse du moteur d'inférence qui est menacé.

L'outil de Knowledge Management propose alors une interface graphique qui permet soit de télécharger des documents déjà scannés, soit de sélectionner des documents déjà enregistrés dans la base de données de l'outil de KM.

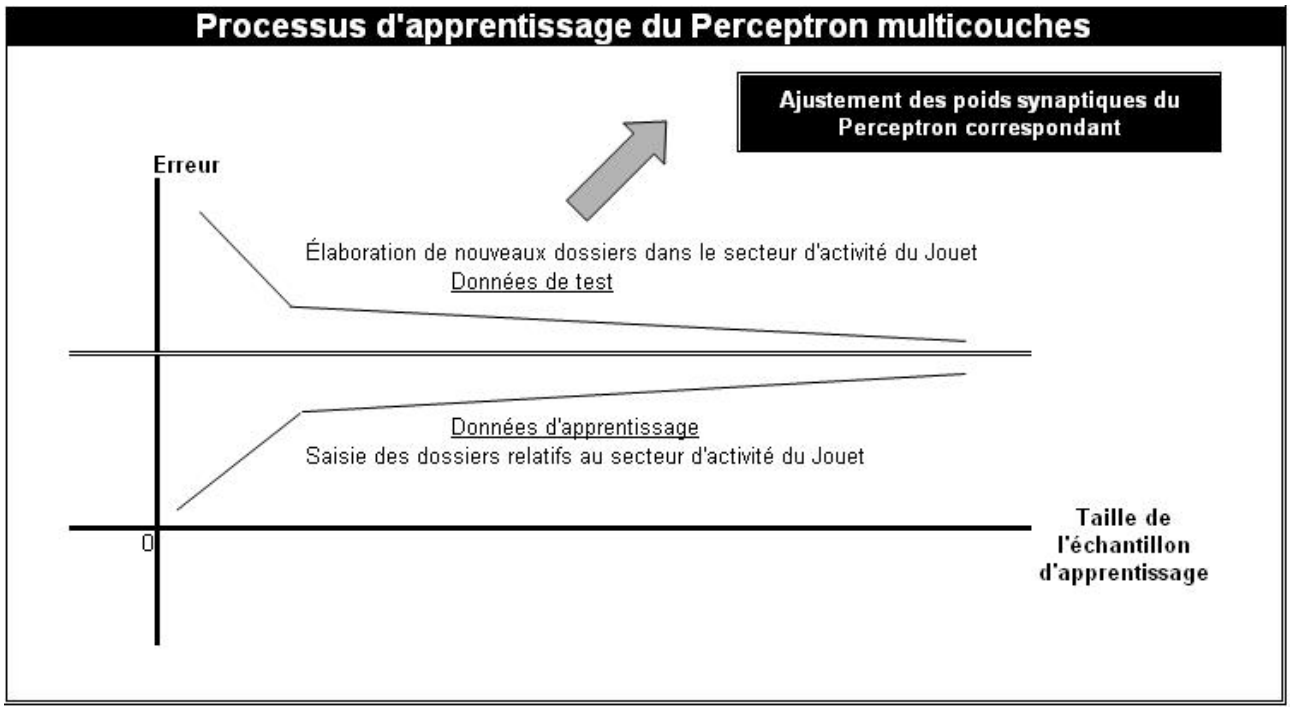


Il est à noter que le 3ème cas est exposé est purement conceptuel et est esquissé uniquement pour exposer les rouages du perceptron multicouche du moteur d'inférence. En effet, dans l'utilisation réelle de l'outil, l'option « Autre Barème à définir » n'existe pas car il existe autant de ligne de perceptron multicouche, qu'il y a de barème de droits possibles. De ce fait, c'est un barème de droits particulier inférés à partir des différentes caractéristiques saisies pour l'entreprise, qui sera proposé à l'utilisateur, et ce par le biais de l'IHM de l'outil de KM.

Il est à noter que le moteur d'inférence est évolutif, puisque le processus d'apprentissage du perceptron multicouches permet d'affiner les poids des neurones cachées, dans un mouvement de double herméneutique :

- Lorsque pour un dossier donné, l'utilisateur valide le barème des droits proposé par le moteur d'inférence, les poids synaptiques des neurones de la couche cachée sont recalculés de façon à ce que la valeur des signaux sortant de la couche cachée, déterminés par les différentes caractéristiques du dossier, soient égaux à ceux des signaux du neurone de sortie.

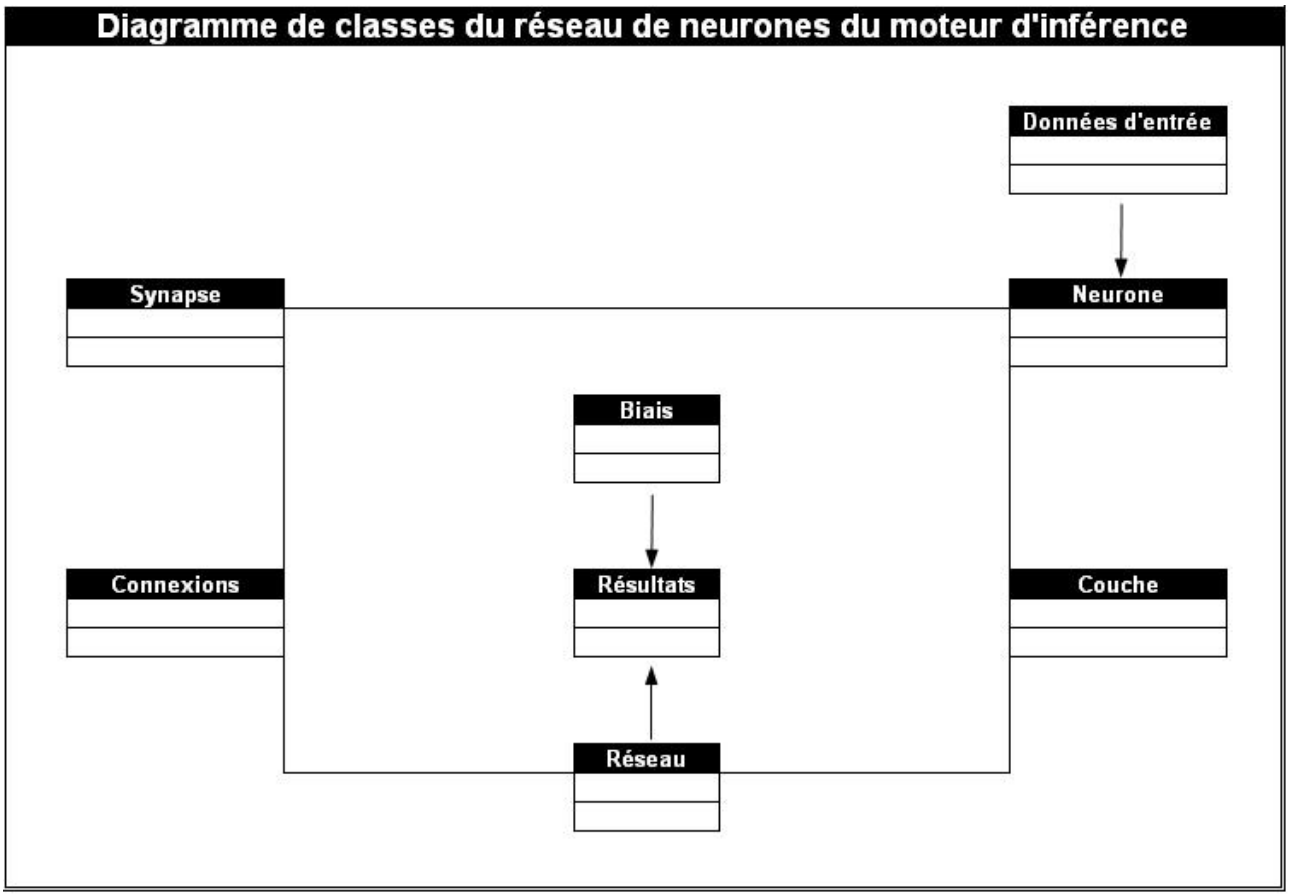
- Lorsque l'utilisateur rentre les caractéristiques du dossier, les proposition de barèmes inférées sont alors encore plus pertinentes, car les poids des neurones intermédiaires ont été affinés sur un grain de calcul beaucoup plus précis, dont la probabilité de pertinence sera plus élevée. C'est donc un cycle de calcul vertueux...



Enfin, il convient de rappeler que les exemples de calcul présenté sont purement pédagogiques, car ils ont été simplifiés à l'extrême. Le seul but était d'exposer grossièrement et trivialement les principes du moteur d'inférence utilisé par l'outil de KM. En effet, en réalité, l'architecture des réseaux de neurones utilisés pour le calcul des inférences était beaucoup plus complexe et les algorithmes du moteur d'inférence de cet outil de KM s'avéraient terriblement plus compliqués.

ANNEXES

Voici le diagramme de classes du réseau de neurone au coeur du moteur d'inférence de l'outil de Knowledge Management d'Hibiscus:



Exemple d'implémentation en Java de la classe Neurone :

```
package neuronespack;

/**
 * Package Neurones
 * Fonctionalités de base des réseaux neuronaux.
 **/

import java.util.*;

/**
 * <p>Title: Neurone</p>
 * <p>Description: Gestion des fonctions de base d'un neurone.</p>
 */
public abstract class Neurone implements Unite {

        public static final int VALEUR_SIGNAL = 0;
        public static final int VALEUR_POTENTIEL = 1;

        /** Dimension du tableau des paramètres */
        private int nbParametresNeurone;
        /** Paramètres du neurone */
        private double parametreNeurone[];
        /** Nombre de synapses entrantes */
        private int nbSynapsesIn;
        /** Nombre de synapses sortantes */
        private int nbSynapsesOut;
        /** Vecteur des synapses entrantes */
        private Vector synapsesIn;
        /** Vecteur des synapses sortantes */
        private Vector synapsesOut;
        /** Identifiant */
        private int idNeurone;
        /** Couche contenant le neurone */
        private Couche maCouche;

        /**
         * Constructeur du neurone
         * @param c Couche : Couche contenant le neurone
         * @param Id int : Identifiant du neurone
         * @param nbp int : Nombre de paramètres du neurone
         */
        public Neurone(Couche c, int Id, int nbp) {
          maCouche=c;
          idNeurone = Id;
          nbParametresNeurone = nbp;
          parametreNeurone = new double[nbParametresNeurone];
          nbSynapsesIn = 0;
          nbSynapsesOut = 0;
          synapsesIn = new Vector(nbSynapsesIn);
          synapsesOut = new Vector(nbSynapsesOut);
```

```
}

/** Ajoute une synapse entrant dans le neurone */
public void addSynapsesIn(Synapse s) {
  nbSynapsesIn++;
  synapsesIn.addElement(s);
}

/** Ajoute une synapse sortant du neurone */
public void addSynapsesOut(Synapse s) {
  nbSynapsesOut++;
  synapsesOut.addElement(s);
}

/** Getter d'une synapse entrante */
public Synapse getSynapseIn(int n) {
  return (Synapse) synapsesIn.elementAt(n);
}

/** Getter d'une synapse sortante */
public Synapse getSynapseOut(int n) {
  return (Synapse) synapsesOut.elementAt(n);
}

/** Getter du nombre de synapses entrantes */
public int getNbSynapsesIn() {
  return nbSynapsesIn;
}

/** Getter du nombre de synapses sortantes */
public int getNbSynapsesOut() {
  return nbSynapsesOut;
}

/** Setter du potentiel */
public void setPotentiel(double p) {
  parametreNeurone[VALEUR_POTENTIEL] = p;
}

/** Renvoie le potentiel */
public double getPotentiel() {
  return parametreNeurone[VALEUR_POTENTIEL];
}

/** Calcul du potentiel */
public double calcPotentiel() {
  double p=0.0D;
  for(int i=0;i<nbSynapsesIn;i++) {
    p += ((Synapse) synapsesIn.elementAt(i)).getValeurSynapse();
  }
  return p;
}
```

```
    /** Setter du signal */
    public void setSignal(double s) {
      parametreNeurone[VALEUR_SIGNAL] = s;
    }

    /** Renvoie le signal du neurone */
    public double getSignalBrut() {
      return parametreNeurone[VALEUR_SIGNAL];
    }

    /** Calcul du signal */
    public double calcSignal() {
      return (FonctionTransfert.calcTransfert(maCouche.getTypeFT(),
                    parametreNeurone[VALEUR_POTENTIEL],   maCouche.getParamFT()));//
  .paramFT));
    }

    /** Getter de l'identifiant */
    public int getID() {
      return idNeurone;
    }

    /** Getter d'un paramètre */
    public double getParametreNeurone(int n) {
      return parametreNeurone[n];
    }

    /** Setter d'un paramètre */
    public void setParametreNeurone(int n, double v) {
      parametreNeurone[n] = v;
    }

    /** Getter de la couche du neurone */
    public Couche getMaCouche() {
      return maCouche;
    }

    public abstract void run();

}
```