Compte rendu du projet:
Créer un réseau de neurones

APARISI Mélanie
JOBERT Pauline
RIAHI Mathilde

Module utils (codé par Pauline Jobert) :

dépendances : utils.h

fonctions codées :

log show position : cadeau du professeur, indique la position du curseur dans un fichier

lire_entier : permet de lire un entier dans un fichier texte déjà ouvert par un algorithme antérieur.
Paramètres : pointeur sur le fichier dont on veut lire l'entier FILE * fichier
Fonctions particulières utilisées : -fscanf (nom_du_fichier, « %d »,entier)
Fonctionnement : Premièrement, on définit un entier destiné à contenir l'entier lu. Deuxièmement, on lit l'entier du fichier et on le stocke dans l'entier défini durant la première étape. La fonction retourne l'entier lu.

lire_float : permet de lire un flottant dans un fichier texte déjà ouvert par un algorithme antérieur.
Paramètres : pointeur sur le fichier dont on veut lire le flottant FILE * fichier
Fonctions particulières utilisées : -fscanf (nom_du_fichier, « %f»,flottant)
Fonctionnement : Premièrement ,on définit un flottant destiné à contenir l'entier lu. Deuxièmement on lit le flottant du fichier et on le stocke dans le flottant défini durant la première étape. La fonction retourne le flottant lu

lire_entier_intervalle : reprend le concept de lire_entier.
Paramètres : - pointeur sur le fichier dont on veut lire le flottant: FILE * fichier
- intervalle : min et max qui sont des entiers
Fonctions particulières utilisées : lire_entier
Fonctionnement : La fonction lit, dans un fichier ouvert précédemment, un entier et vérifie s'il appartient à l'intervalle donné en argument. S'il n'appartient pas à l'intervalle, on réaplique la fonction (qui lira l'entier suivant) jusqu'à ce que l'entier lu appartienne bien à l'intervalle. Avant tout, on vérifie que l'intervalle soit bien défini. La fonction retourne l'entier lu.

lire_string : permet de lire une chaîne de caractères dans un fichier texte déjà ouvert par un algorithme antérieur.
Paramètres : pointeur sur le fichier dont on veut lire la chaîne de caractères : FILE * fichier
Fonctions particulières utilisées : malloc et fscanf (nom_du_fichier, « %s»,string)
Fonctionnement : Il faut premièrement définir un tableau (un pointeur) destiné à lire la chaîne de caractère. Il faut deuxièmement lui allouer une place suffisante pour contenir (un char pour une lettre, donc on réserve une vingtaine d'octets pour être sûr). Il faut, troisièmement, lire la chaîne de

caractère dans le fichier et la stocker dans le tableau. La fonction retourne le tableau contenant la chaîne de caractère lue.

lire_mot_clef :lit un mot et indique s'il s'agit du mot clef mis en argument ou non

Module matrices (codé par Mathilde Riahi) :

dépendances : utils.h , matrices.h, matrices_struct.h

fonctions codées :

detruit_matrice : libère toute la mémoire allouée pour stocker une matrice.
Paramètres : pointeur sur une structure de matrice (struct matrice_s * m)
Fonctions particulière utilisées : free()
Fonctionnement : En premier, la mémoire de chaque colonnes, à chaque ligne est libérée. Ensuite on libère la mémoire de chaque ligne, puis celle de la matrice toute entière. Le nombre de lignes et de colonnes de la matrice vaut donc 0.

creation_matrice : Alloue de la mémoire pour y stocker une structure de matrice.
Paramètres : deux entiers, un pour le nombre de lignes et un pour le nombre de colonnes
Fonctions particulières utilisées : malloc ()
Fonctionnement : On commence par allouer de la mémoire à la structure de la matrice. On donne au nombre de lignes et de colonnes de cette structure la valeur des paramètres de cette fonction. Ensuite on alloue de la mémoire pour qu'à chaque ligne il y ai assez de mémoire pour stocker un pointeur sur un float. Enfin, on alloue de la mémoire pour qu'il y en ai assez pour stocker un float. La fonction retourne la matrice (vide) créée.

matrice_ligne : cadeau du prof

creation_matrice_aleatoire : demande le nombre de lignes et de colonnes de la matrice puis des coefficients (float) entre -1 et 1
Paramètres : deux entiers, un pour le nombre de lignes, un autre pour celui de colonnes
Fonctions particulières utilisées : rand() [initialisation : srand(time(NULL))]
Fonctionnement : on appelle la fonction creation_matrice afin d'allouer de la mémoire pour la matrice que l'on va créer. Chaque coefficient de la matrice est choisi aléatoirement par la fonction rand().
La fonction rand() prend normalement des valeurs entre 0 et RAND_MAX. Afin d'avoir des valeurs entre -1 et 1, on multiplie par deux rand(), on lui soustrait 1 et enfin on divise le tout par RAND_MAX. La fonction retourne la matrice (avec ses coefficients aléatoire) créée.

creation_matrice_utilisateur : même principe que la fonction précédente mais cette fois ci c'est l'utilisateur qui choisit les coefficients (toujours entre -1 et 1)
Paramètres : aucun
Fonctions particulières utilisées : scanf ()
Fonctionnement : On commence par demander à l'utilisateur (avec la fonction scanf()) le nombre de lignes et de colonnes souhaités pour créer la matrice. Puis la mémoire est allouée pour la matrice souhaitée en appelant la fonction creation_matrice. Après, on demande (avec la fonction scanf()) à l'utilisateur la valeur du coefficient (entre -1 et 1) qu'il veut. La fonction retourne la matrice (avec les coefficients choisis par l'utilisateur) créée.

lire_matrice_fichier : lit dans un fichier texte [déjà ouvert par un algorithme antérieur] le

nombre de lignes, de colonnes et les coefficients d'une matrice.
Paramètres : pointeur sur le fichier dont on veut lire la matrice : FILE * f et un pointeur sur un pointeur de structure matrice (struct matrice_s * * m)
Fonctions particulières utilisées : fscanf (pour lire les valeurs dans le fichier)
Fonctionnement : On commence d'abord par lire le nombre de lignes et de colonnes dans le fichier. Ensuite on appelle la fonction creation_matrice afin d'allouer de la place pour la stocker. Puis, on vient lire les coefficients de la matrice. Enfin, on affiche la matrice qui vient d'être lue.

sauve_matrice_fichier : écrit dans un fichier [déjà ouvert par un algorithme antérieur] le nombre de lignes, de colonnes et les coefficients d'une matrice.
Paramètres : pointeur sur le fichier dont on veut lire la matrice : FILE * f et un pointeur sur une structure matrice (struct matrice_s * m)
Fonctions particulières utilisées : fprintf (pour écrire dans un fichier)
Fonctionnement : On commence par écrire le nombre de lignes puis de colonnes de la matrice, puis on écrit les coefficients.

affiche_matrice : affiche une matrice.
Paramètres : pointeur sur une structure matrice (struct matrice_s * m)
Fonctions particulières utilisées : aucune
Fonctionnement : Affiche (printf) le contenu de la matrice, une tabulation (\t) entre chaque coefficients est insérée.

copie_matrice : Copie le contenu d'une matrice m1 dans une matrice m2
Paramètres : 2 pointeurs sur une structure matrice (struct matrice_s * m1, struct matrice_s *m2)
Fonctions particulières utilisées : aucune
Fonctionnement : Prend le coefficient à la ligne i, colonne j de la matrice m1 et le met à la ligne i, colonne j de la matrice m2.

Module matrices_accesseurs (codé par Mélanie Aparisi)

Dépendances : matrices_struct.h matrices_accesseurs.h

Fonctions codées :

nb_lignes : Renvoie le nombre de lignes de la matrice.
Paramètres : Pointeur sur une structure de matrice (struct matrice_s * m)
Fonctions particulières utilisées : Aucune
Fonctionnement : Une variable (de type int) est égale à la valeur du nombre de lignes de la matrice associée à la structure
On obtient cette valeur en utilisant un pointeur sur la structure
Exemple : On imagine une matrice de deux lignes et trois colonnes, cette fonction renvoie le nombre de lignes, c'est-à-dire 2

nb_cols : Renvoie le nombre de colonnes de la matrice
Paramètres : Pointeur sur une structure de matrice (struct matrice_s * m)
Fonctions particulières utilisées : Aucune
Fonctionnement : Une variable (de type int) est égale à la valeur du nombre de colonnes de la matrice associée à la structure

On obtient cette valeur en utilisant un pointeur sur la structure
Exemple : On imagine une matrice de deux lignes et trois colonnes, cette fonction renvoie le nombre de colonnes, c'est-à-dire 3

matrice_set : Permet de mettre une valeur à une certaine position de la matrice
Paramètres :Pointeur sur une structure de matrice (struct matrice_s * m), le numéro de la ligne souhaitée (int num_ligne), le numéro de la colonne souhaitée (int num_colonne) et la valeur que l'on souhaite entrer dans la matrice, de type float (float value)
Fonctions particulières utilisées : Aucune
Fonctionnement : En renseignant une ligne et une colonne, on entre une certaine valeur à la position indiquée
Exemple : Si on souhaite entrer la valeur 3.00 à la ligne 1, colonne 1, il faudrait écrire :

```
int ligne = 1, col =1; float f = 3.00;
matrice_set (struct matrice_s * m, ligne, col, f);
```

matrice_get : Permet de récupérer une valeur à une certaine position de la matrice
Paramètres : Pointeur sur une structure de matrice (struct matrice_s * m), le numéro de la ligne souhaitée (int num_ligne), le numéro de la colonne souhaitée (int num_colonne) et un pointeur de type float (float *)
Fonctions particulières utilisées : Aucune
Fonctionnement : En renseignant une ligne et une colonne, on accède à l'adresse de la valeur souhaitée
Exemple : Si on reprend l'exemple précédent, on souhaite récupérer la valeur à l'emplacement ligne 1, colonne 1 (qui est 3.00)
Il faut créer une variable de type float * (exemple : float * pointeur;), appeler la fonction puis lorsque l'on printf :
Si on printf pointeur : on obtient l'adresse de la valeur
Si on printf *pointeur : on retrouve la valeur souhaité (c'est-à-dire 3.00)

matrice_raw : Permet de récupérer l'adresse de m
Paramètres : Pointeur sur une structure de matrice (struct matrice_s * m)
Fonctions particulières utilisées : Aucune
Fonctionnement : En entrant cette structure de matrice, on récupère l'adresse de cette dernière
Exemple : Il faut bien penser à créer une variable de type float *** (pointeur triple car m est techniquement déjà un pointeur double, l'adresse de ce dernier est donc un pointeur triple)
En faisant cela :

```
float *** exemple;
exemple = matrice_raw(struct matrice_s * m);
```

La variable "exemple" est désormais égale à l'adresse de m

Module matrices_operations (Pauline Jobert)

Dépendances : matrices_struct.h

Fonctions codées :

transpose_matrice : transpose une matrice m1 et met le résultat dans une matrice m2.
Paramètres : pointeurs structures de matrice (struct matrice_s * m1 [IDEM avec m2])
Fonctions particulières utilisées : aucune

Fonctionnement : La valeur à la ligne i et colonne j de la m1 est égale à la valeur de la ligne j et colonne i de la m2
Exemple : dans la matrice 3x3 m2, la valeur à la ligne 2 et colonne 1 est égale à la valeur de ligne 1 et colonne 2 de la matrice 3x3 m1.

addition_matrice_scalaire : additionne les matrices m1 et (mu) m2 et met le résultat dans la matrice m3
Paramètres : trois pointeurs sur structures de matrice (struct matrice_s * m1 [IDEM avec m2 et m3]) et un float (mu)
Fonctions particulières utilisées : aucune
Fonctionnement : On additionne les coefficients a(ij)+mu b(ij) des matrices respectives m1 et m2 et on place le résultat dans la m3 à l'emplacement (ij) (i désigne l'emplacement de la case sur la ligne et j le l'emplacement de la case sur la colonne) . On répète l'opération pour tous les emplacements des matrices

multiplication_matrice : multiplie les matrices m1 et m2 et met le résultat dans la matrice m3
Paramètres : trois pointeurs sur structures de matrice (struct matrice_s * m1 [IDEM avec m2 et m3])
Fonctions particulières utilisées : aucune
Fonctionnement : Le coefficient c(ij) de la matrice m3 est égale à $\sum$ a(ik)b(kj) avec k allant de 1 à (nombre de ligne de la matrice m2). On répète l'opération pour tous les coefficients de la matrice m3

matrice_apply_one_arg :
Paramètres : 2 pointeurs sur les matrices m1 et m2(struct matrice_s*) et un pointeur sur fonction (*f) , celle ci prend en argument un float et renvoie un float
Fonctions particulières utilisées : appel de la fonction f par l'intermédiaire d'un pointeur
Fonctionnement : on applique la fonction f au premier coefficient de la matrice m1 (a(00)) et on met le résultat dans le premier emplacement de m2 (b(00)) : b(00) = f (a(00)). On répète l'opération pour tous les emplacements : b(ij) = f (a(ij))

matrice_apply_two_args :
Paramètres : 3 pointeurs sur les matrices m1, m2 et m3(struct matrice_s*) et un pointeur sur fonction (*f) , celle ci prend en argument un 2 float et renvoie un float
Fonctions particulières utilisées :appel de la fonction f par l'intermédiaire d'un pointeur
Fonctionnement :on applique la fonction f aux premiers coefficients des matrices m1 (a(00)) et m2 (b(00)) et on met le résultat dans le premier emplacement de m3 (c(00)) : c(00) = f (a(00),b(00)). On répète l'opération pour tous les emplacements : c(ij) = f (a(ij), b(ij))

matrice_apply_three_args :
Paramètres : 4 pointeurs sur les matrices m1, m2 , m3 et m4(struct matrice_s*) et un pointeur sur fonction (*f) , celle ci prend en argument un 3 float et renvoie un float
Fonctions particulières utilisées :appel de la fonction f par l'intermédiaire d'un pointeur
Fonctionnement :on applique la fonction f aux premiers coefficients des matrices m1 (a(00)) , m2 (b(00)) et m3 c(00) et on met le résultat dans le premier emplacement de m4 (d(00)) :
d(00) = f (a(00),b(00),c(00)). On répète l'opération pour tous les emplacements : d(ij) = f (a(ij), b(ij),c(ij))

multiplication_matrice_retro_propagation :
Paramètres : trois pointeurs sur structures de matrice (struct matrice_s * m1 [IDEM avec m2 et m3])

Fonctions particulières utilisées : Aucune
Fonctionnement :
Calcul du gradient, puis normalisation des données du réseau (pour éviter des valeurs trop grandes)

matrice_mise_a_jour_coefficients :
Paramètres : trois structures de matrice (pointeur simple, struct matrice_s * erreurs_couche_suivante [IDEM avec activations_couche_precedente et coefficients]) et un float (lambda)
Fonctions particulières utilisées : Aucune
Fonctionnement :
On met à jour le poids dans toutes les couches, avec le float (lambda) étant le taux d'apprentissage

Module activation (codé par Mathilde Riahi)

Dépendances : activation_struct.h activation.h utils.h

Fonctions codées :

choix_fonction_d_activation : on choisit la fonction d'activation que l'on souhaite utiliser.
Paramètres : un id (qui correspond à la fonction d'activation)
Fonctions particulières utilisées : malloc ()
Fonctionnement : Si l'id choisi par l'utilisateur vaut 0, alors, au réseau (res) pointant sur applique (resp. derivee) on lui donne l'adresse de la fonction_d_activation_Identity (resp. derivee_fonction_d_activation_Identity)
Si l'id choisi par l'utilisateur est différent de 0 (=1), alors, au réseau (res) pointant sur applique (resp. derivee) on lui donne l'adresse de la fonction_d_activation_Racine_carree (resp. derivee_fonction_d_activation_Racine_carree).
La fonction retourne res.

sauve_fonction_d_activation : écrit l'id de la fonction d'activation choisie dans un fichier
Paramètres : pointeur sur le fichier dont où on veut écrire l'id : FILE * f et pointeur structure de fonction d'activation (struct fonction_d_activation_s *)
Fonctions particulières utilisées : fprintf ()
Fonctionnement : Ecrit dans le fichier préalablement ouvert, l'id de la fonction d'activation choisie.

lit_fonction_d_activation : lit dans un fichier l'id d'une fonction d'activation
Paramètres : pointeur sur le fichier dont on veut lire l'id : FILE * f et pointeur sur un pointeur de structure de fonction d'activation (struct fonction_d_activation_s ** fun)
Fonctions particulières utilisées : lire_entier (f) (codée dans le module utils)
Fonctionnement : On lit dans le fichier, préalablement ouvert, l'id de la fonction d'activation écrite dans le fichier.

demande_fonction_d_activation :
Paramètres : aucun
Fonctions particulières utilisées : printf() et scanf()
Fonctionnement : Affiche les id et leur numéros correspondant grâce à printf(). Ensuite on demande à l'utilisateur le numéro de l'id de la fonction d'activation qu'il souhaite grâce à scanf(). La fonction return l'id choisi par l'utilisateur.

Module réseau (Pauline Jobert)

dependances: utils.h, matrice.h, matrices_accesseur.h, activation.h, couche_struct.h, reseau_struct.h, reseau.h, couche_entree.h, couche_sortie, couche_activation, couche_matrice

fonctions codées:

detruit_reseau : libère la mémoire utilisée pour stocker un réseau dans la mémoire
Paramètres : pointeurs structures de réseau (struct reseau_s *)
Fonctions particulières utilisées : appel d'un pointeur sur fonction (detruit) contenue dans une structure (couche) pointée par une autre structure (réseau)
Fonctionnement : La structure réseau pointe sur la structure couche qui contient un pointeur sur fonction qui détruit une couche d'un réseau. On appelle cette fonction et on l'applique sur toutes les couches du réseau (on libère donc les struct couche_s). ensuite on libère la structure du réseau.

matrice_s * reseau_get_entree : Renvoie activation.s de la couche d'entrée
Paramètres : pointeurs structures de réseau (struct reseau_s *)
Fonctions particulières utilisées : aucune
Fonctionnement : Il faut d'abord parcourir le réseau et trouver la couche d'entrée, pour ça,on teste le type de toutes les couches jusqu'à ce qu'on la trouve. Une fois trouvée , on renvoie l'activation.s de cette couche contenue (voir couche_struct.h pour le visualiser).

matrice_s * reseau_get_sortie :
Paramètres : pointeurs structures de réseau (struct reseau_s *)
Fonctions particulières utilisées :
Fonctionnement : Il faut d'abord parcourir le réseau et trouver la couche de sortie, pour ça,on teste le type de toutes les couches jusqu'à ce qu'on la trouve. Une fois trouvée , on renvoie l'activation.p de cette couche contenue (voir couche_struct.h pour le visualiser).

void propagation: Appelle la fonction propagation de chaque couche en partant de la couche 0
Paramètres : pointeurs structures de réseau (struct reseau_s *) et pointeur sur une structure matrice(struct matrice_s * entree) ,
Fonctions particulières utilisées : appel d'une fonction (ici propagation) contenue dans une structure (ici couche)
Fonctionnement : Il s'agit de parcourir toutes les couches du réseau (de la première à la dernière couche) et pour chacune d'elles, d'appeler la fonction propagation qui prendra en argument le pointeur de la couche sur laquelle on est et un pointeur sur une structure de matrice.

void rétropropagation: Appelle la fonction rétro-propagation de chaque couche en partant de la dernière couche
Paramètres : pointeurs structures de réseau (struct reseau_s *)
Fonctions particulières utilisées : appel d'une fonction (ici rétro-propagation) contenue dans une structure (ici couche)

Fonctionnement :Il s'agit de parcourir toutes les couches du réseau (de la dernière à la première) et pour chacune d'elles, d'appeler la fonction propagation qui prendra en argument le pointeur de la couche sur laquelle on est et un pointeur sur une structure de matrice.

int écrit_reseau_fichier :Écrit le nombre de couches, et appelle la fonction "sauve" de chaque couche en partant de la dernière couche.

Paramètres : pointeurs structures de réseau (struct reseau_s *) et un FILE * f
Fonctions particulières utilisées : fprintf , appel d'un pointeur sur fonction
Fonctionnement :On écrit le nombre de couche en appelant fprintf qui écrit dans le fichier, puis on fait appel à une boucle (qui part de la dernière jusqu'à la première couche du réseau) pour appeler le pointeur sur la fonction "sauve" (qui est dans la structure de la couche) pour chaque couche du réseau.

int lire_type_couche : lit un type de couche, renvoie 0 si un type est trouvé, et 1 sinon. (cadeau du professeur)

Paramètres : pointeur sur un fichier (FILE * f) ,pointeur sur une enum (type_couche * t)
Fonctions particulières utilisées :lire_mot_clef (module utils).
Fonctionnement : Il s'agit de lire un mot dans un fichier avec lire_mot_clef , puis de renvoyer 1 si le mot correspond à un type de couche existant (entrée, sortie, matrice, ou activation). Sinon on renvoie 0. On appelle donc la fonction (on fait donc 5 disjonctions de cas, 4 qui renvoient 0 et 1 qui renvoie 1)

lit_reseau_fichier: Lit un réseau dans un fichier qui a été écrit par ecrit_reseau_fichier.

Paramètres : pointeurs structures de réseau (struct reseau_s *)
Fonctions particulières utilisées : lire_type_couche, lire_mot_clef (module utils), appel de pointeur sur fonction, fscanf,
Fonctionnement : Il s'agit d'une série d'étape :
-on alloue la mémoire nécessaire pour stocker un réseau
-on lit en premier le nombre de couche (et on le stocke)
-à partir de là on crée une boucle qui parcourt toutes les couches en partant de la fin
pour chaque boucle:
-on vérifie que le type existe avec lire_type
-on lit le type de la couche avec lire_mot_clef (on le stocke)
-si couche sortie : on appelle couche_sortie
-si couche entrée: on appelle couche_entrée
-si couche matrice : on lit la matrice avec lire_matrice_fichier (module matrice)(et on la stocke)
-si couche activation : on lit la fonction d'activation avec lire_fonction d'activation (on la stocke)

Module spécification (codé par M.Chevalier, commentaires et explications par Mélanie Aparisi)

Dépendances : utils.h activation.h activation.h matrices.h couches_struct.h couches_entree.h couches_sortie.h couches_matrice.h couches_activation.h specification.h specification_reseau_struct.h

Fonctions codées :

detruit_specification :
Paramètres : Pointeur sur la structure “struct specification_reseau_s”
Fonctions particulières utilisées : free()
Fonctionnement : Libère la mémoire allouée à couche, puis libère la mémoire allouée à spec

print_specification : Permet d’afficher, pour chaque couche, soit le nombre de neurones, soit la fonction d’activation
Paramètres : Pointeur sur la structure “struct specification_reseau_s”
Fonctions particulières utilisées : fflush(stdout)
Fonctionnement : Premièrement, la fonction affiche le nombre de couches
On réalise une boucle pour chaque couche, afin de différencier chaque situation de l’énumération dans type_couche
Si on essaie d’entrer une valeur qui n’est pas présente dans l’énumération, alors on bascule sur la partie default du switch, une erreur est renvoyée
Enfin, on vide la mémoire tampon

demande_type_couche_interne :
Paramètres : Aucun
Fonctions particulières utilisées : lire_entier_intervalle du module utils
Fonctionnement : On demande le type de la couche suivante, il y a deux choix possibles (matrice et fonction d’activation)
Avec la fonction lire_entier_intervalle, le choix est réalisé entre les deux possibilités

demande_specification_utilisateur :
Paramètres : Aucun
Fonctions particulières utilisées : malloc(...), lire_entier_intervalle, demande_fonction_d_activation, print_specification, demande_type_couche_interne
Fonctionnement : But final : indiquer toutes les spécifications de res, un pointeur sur la structure specification_reseau_s et on renvoie ce “res”

Explication détaillée :
Tout d’abord on initialise type_couche t, on récupère le nombre de couches internes et on alloue la mémoire nécessaire à couche (un pointeur sur la structure specification_couche_s)
On ajoute deux couches, pour l’entrée et la sortie
On cherche à savoir le nombre de neurones en entrée, pour ensuite réaliser une boucle qui scindera en plusieurs cas suivant le type de la couche
Premier cas : C’est une matrice
Alors, la couche prends la valeur 2 (c’est-à-dire la valeur de type_couche_matrice), type_couche_precedent de l’union paramètre prend la valeur de taille_entree et on affiche le nombre de neurones sur la couche suivante
Deuxième cas : C’est une fonction d’activation
La couche prends la valeur 3 (c’est-à-dire la valeur de type_couche_activation de l’énumération type_couche) ensuite dans parametre, pour avoir la valeur d'activation_id on utilise la fonction demande_fonction_d_activation du module activation

Pour terminer, on donne la valeur de 1 à la couche de sortie (c’est-à-dire la valeur de type_couche_sortie), on demande le nombre de neurones dans cette couche et on utilise la fonction print_specification pour afficher res, et enfin on renvoie res

creation_reseau :

Paramètres : Pointeur sur la structure “struct specification_reseau_s”
Fonctions particulières utilisées : malloc(...), couche_sortie, couche_matrice, couche_activation, couche_entree, choix_fonction_d_activation
Fonctionnement : But final : création de res, un pointeur sur la structure reseau_s, avec les allocations de mémoire nécessaire et spec, le pointeur sur la structure specification_reseau_s

Explication détaillée :
En premier on réalise les allocations de mémoire, tout d’abord pour res et ensuite pour la couche
On réalise une boucle pour parcourir les couches, et on diffère les cas suivant si c’est une matrice ou bien une fonction d’activation
Si c’est une matrice, couche prend la valeur renvoyée par la fonction couche_matrice (à partir de spec et de la couche suivante)
Si c’est une fonction d’activation, couche prend la valeur renvoyée par la fonction couche_activation (à partir de spec et de la couche suivante)

Après la boucle, type_couche t prend la valeur renvoyée par la fonction couche_entree

Au final, on renvoie le réseau créé