Création d'un réseau de neurones – c'est facile

Steffen Nissen

Ressources en ligne :

Sur le CD vous trouverez les fichiers codes décrits dans l'article ainsi que la bibliothèque.

Depuis des années, les films de science–fiction tout droit venus d'Hollywood tels que *I, Robot* ont annoncé l'arrivée de l'Intelligence Artificielle (Artificial Intelligence) comme un signe avant-coureur de l'Armageddon. Toutefois, ces films ne pouvaient être aussi éloignés de la vérité. Alors qu'Hollywood nous régale de films d'horreur annonçant notre déclin imminent, une communauté de scientifiques n'ayant aucune volonté de détruire notre espèce, exploite l'Intelligence Artificielle afin de rendre notre vie plus simple, plus productive, plus longue, et d'une manière générale plus agréable.

Les robots dans *I, Robot* sont dotés d'un cerveau artificiel fonctionnant grâce à un réseau de neurones artificiels ; ce réseau de neurones artificiels (RNA en français pour Artificial Neural Network) est construit sur le modèle du réseau de neurones de nos propres cerveaux humains. La bibliothèque FANN (Fast Artificial Neural Network, réseau de neurones artificiels rapide) est une bibliothèque de réseau de neurones artificiels, qui peut être utilisée à partir de C, C++, PHP, Python, Delphi et Mathematica et bien qu'elle ne puisse reproduire la magie d'Hollywood, c'est tout de même un outil puissant pour les développeurs de logiciels. Les Réseaux de Neurones Artificiels peuvent être utilisés dans des domaines aussi divers que la création d'interfaces de jeux plus attrayantes pour les jeux sur PC, l'identification d'objets dans des images et l'aide aux prévisions des tendances volatiles de la bourse.

L'approximation fonctionnelle

Les Réseaux de Neurones Artificiels appliquent, entre autre, le principe de l'approximation fonctionnelle. Ainsi, ils apprennent une fonction en regardant des exemples de la dite fonction. Un des exemples les plus simples est celui d'un Réseau de Neurones artificiels apprenant la fonction XOR (OU-exclusif), mais il pourrait tout aussi bien apprendre à déterminer la langue d'un texte, ou la présence d'une tumeur dans une image passée au rayon-X.

Steffen Nissen est un scientifique danois, spécialisé dans l'informatique. Il a crée et maintient activement la bibliothèque FANN pendant que d'autres maintiennent les liens d'accès pour d'autres langages. Il est également l'auteur d'un rapport technique sur la création de la bibliothèque FANN intitulé *Implementation of a Fast Artificial Neural Network Library (fann)*.
Pour contacter l'auteur : *lukesky@diku.dk*

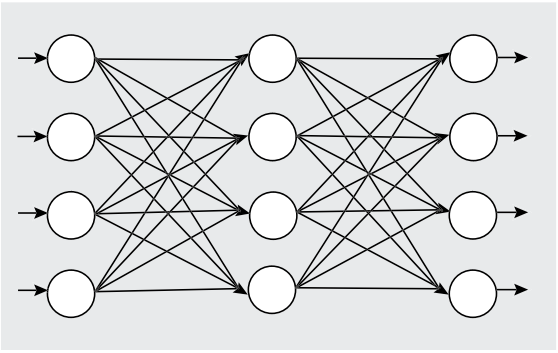

Figure 1. *Un Réseau de Neurones Artificiels constitué de quatre neurones en entrée, une couche cachée et quatre neurones en sortie*

Si un Réseau de Neurones Artificiels est capable d'apprendre un problème, ce dernier doit être défini comme une fonction contenant un ensemble de variables d'entrée et de sortie supportées par des exemples qui illustrent la façon dont cette fonction devrait effectuer l'évaluation. Un problème comme la fonction XOR est déjà défini comme une fonction contenant deux variables d'entrée binaires et une variable de sortie binaire. Les exemples sont définis par quatre modèles différents d'entrée. Des problèmes plus ardus peuvent cependant s'avérer plus difficiles à définir en tant que fonctions. Les variables d'entrée d'un problème consistant à trouver une tumeur sur une image au rayon-X pourraient être les valeurs de pixels de l'image, mais elles pourraient également prendre d'autres valeurs extraites de l'image en question. Les données de sortie pourraient être soit une valeur binaire soit une valeur à virgule flottante représentant la probabilité de la présence d'une tumeur dans l'image. Dans les Réseaux de Neurones Artificiels, cette valeur à virgule flottante serait normalement comprise entre 0 et 1 inclus.

Le cerveau humain

Un approximateur fonctionnel tel qu'un Réseau de Neurones Artificiels peut être considéré comme une boîte noire. Lorsqu'il s'agit d'un Réseau de Neurones Artificiels, c'est à peu près tout ce que vous avez besoin de savoir. Pour comprendre le fonctionnement d'un Réseau de Neurones Artificiels, une connaissance élémentaire du cerveau humain et de ses mécanismes est toutefois nécessaire.

Le cerveau humain est un système d'une extrême complexité capable de résoudre des problèmes très difficiles. Il est composé de plusieurs éléments différents, mais un des constituants les plus importants est le neurone. Le cerveau humain en contient environ 10^{11}. Ces

neurones sont reliés entre eux par environ 10^{15} connexions afin de créer un immense réseau neuronal. Les neurones s'envoient entre eux des impulsions par le biais de connexions et ces impulsions permettent au cerveau de fonctionner. Le réseau de neurones reçoit également des impulsions des cinq sens et envoie des impulsions aux muscles sous la forme de mouvements ou de paroles.

Listing 1. *Programme calculant les fréquences des lettres A-Z dans un fichier texte*

```
#include <vector>
#include <fstream>
#include <iostream>
#include <ctype.h>

void error(const char* p, const char* p2 = "")
{
  std::cerr << p << ' ' << p2 << std::endl;
  std::exit(1);
}

void generate_frequencies(const char *filename,
  float *frequencies)
{
  std::ifstream infile(filename);
  if(!infile)
    error("Cannot open input file", filename);
  std::vector<unsigned int> letter_count(26, 0);
  unsigned int num_characters = 0;
  char c;
  while(infile.get(c)){
    c = tolower(c);
    if(c >= 'a' && c <= 'z'){
      letter_count[c - 'a']++;
      num_characters++;
    }
  }

  if(!infile.eof()) error("Something strange happened");
  for(unsigned int i = 0; i != 26; i++){
    frequencies[i] =
      letter_count[i]/(double)num_characters;
  }
}

int main(int argc, char* argv[])
{
  if(argc != 2) error("Remember to specify an input file");

  float frequencies[26];
  generate_frequencies(argv[1], frequencies);

  for(unsigned int i = 0; i != 26; i++){
    std::cout << frequencies[i] << ' ';
  }
  std::cout << std::endl;
  return 0;
}
```

Le neurone pris individuellement peut être vu comme une machine d'entrées/sorties dans l'attente d'impulsions des neurones alentours qui envoie une impulsion aux autres neurones lorsqu'elle reçoit suffisamment d'impulsions.

Les Réseaux de Neurones Artificiels

Les neurones artificiels ressemblent à leurs congénères biologiques. Ils sont dotés de connexions en entrée qui s'ajoutent entre elles afin de déterminer la force de leur sortie, résultant de la somme injectée dans une fonction d'activation. Bien qu'il existe de nombreuses fonctions d'activation, la plus connue est la fonction d'activation sigmoïde dont la donnée de sortie est un nombre compris entre 0 (pour les valeurs d'entrée faibles) et 1 (pour les valeurs d'entrée élevées). La résultante de cette fonction est ensuite passée comme entrée pour d'autres neurones à travers un nombre plus élevé de connexions, chacune d'entre elles étant pondérée. Ces poids déterminent le comportement du réseau.

Dans le cerveau humain, les neurones sont connectés entre eux dans un ordre tout aussi aléatoire et envoient des impulsions de façon asynchrone. Si nous voulions modéliser un cerveau humain, il s'organiserait de la même façon qu'un Réseau de Neurones Artificiels, mais comme nous voulons à l'origine créer un approximateur fonctionnel, les Réseaux de Neurones Artificiels ne sont généralement pas organisés sur ce modèle.

Lorsque nous créons des Réseaux de Neurones Artificiels, les neurones sont normalement ordonnés en couches pourvues de connexions les reliant entre elles. La première couche contient les neurones d'entrée et la dernière couche

Listing 2. *Première partie d'un fichier d'apprentissage contenant les fréquences de caractères pour l'anglais, le français et le polonais. La première ligne est un en-tête signifiant qu'il y a 12 modèles d'apprentissage consistant en 26 entrées et 3 sorties.*

```
12 26 3

0.103 0.016 0.054 0.060 0.113 0.010 0.010 0.048 0.056
0.003 0.010 0.035 0.014 0.065 0.075 0.013 0.000 0.051
0.083 0.111 0.030 0.008 0.019 0.000 0.016 0.000

1 0 0

0.076 0.010 0.022 0.039 0.151 0.013 0.009 0.009 0.081
0.001 0.000 0.058 0.024 0.074 0.061 0.030 0.011 0.069
0.100 0.074 0.059 0.015 0.000 0.009 0.003 0.003

0 1 0

0.088 0.016 0.030 0.034 0.089 0.004 0.011 0.023 0.071
0.032 0.030 0.025 0.047 0.058 0.093 0.040 0.000 0.062
0.044 0.035 0.039 0.002 0.044 0.000 0.037 0.046

0 0 1

...
```

les neurones de sortie. Ces neurones d'entrée et de sortie représentent les variables d'entrée et de sortie de la fonction dont nous voulons nous rapprocher. Il existe entre les couches d'entrée et de sortie un nombre de couches cachées, et les connexions (ainsi que les poids) venant et sortant de ces couches cachées déterminent le degré de bon fonctionnement du Réseau de Neurones Artificiels. Lorsqu'un Réseau de Neurones Artificiels apprend à se rapprocher d'une fonction, on lui montre des exemples illustrant la façon dont la fonction effectue l'évaluation et les poids internes au Réseau de Neurones Artificiels sont lentement ajustés afin de produire la même sortie montrée dans les exemples. On espère ainsi que, lorsqu'on montre au Réseau de Neurones Artificiels un nouvel ensemble de variables d'entrées, il produira une donnée de sortie correcte. Par conséquent, si un Réseau de Neurones Artificiels est censé apprendre à repérer une tumeur sur une image au rayon-X, il faudra lui montrer de nombreuses images au rayon-X contenant des tumeurs, et d'autres présentant des tissus sains. Après une période d'apprentissage à l'aide de ces images, les poids du Réseau de Neurones Artificiels devraient normalement contenir des informations lui permettant d'identifier positivement des tumeurs sur des images au rayon-X qu'il n'a pas vues au cours de son apprentissage.

La bibliothèque FANN : tutoriel basé sur des exemples

Grâce à Internet la communication à l'échelle mondiale fait partie intégrante de la vie quotidienne de bon nombre de personnes, mais il a également soulevé le problème de la langue. Les outils d'aide à la traduction peuvent aider à franchir cette barrière linguistique, mais pour que de tels outils fonctionnent, ils ont besoin de connaître la langue dans laquelle est écrit le passage d'un texte. Une façon de déterminer la langue consiste à examiner la fréquence des lettres qui apparaissent dans un texte. Alors que cette approche peut sembler naïve en matière de détection de langues, elle s'est révélée très efficace. Pour de nombreuses langues européennes il suffit de regarder la fréquence des lettres de A à Z, même si certaines langues utilisent également d'autres lettres. La bibliothèque FANN peut être assez facilement utilisée pour élaborer un petit programme qui détermine la langue d'un fichier texte. Le Réseau de Neurones Artificiels utilisé devrait prendre en entrée un neurone pour chacune des 26 lettres, et en sortie un neurone pour chacune des langues. Toutefois, un petit programme doit en premier lieu être élaboré en vue d'effectuer une mesure de la fréquence des lettres dans un fichier texte.

Le Listing 1 générera les fréquences des lettres pour un fichier, et enverra en donnée de sortie les fréquences dans un format utilisable afin de générer un fichier d'apprentissage pour la bibliothèque FANN. Ces fichiers d'apprentissage pour la bibliothèque FANN doivent être composés d'une ligne de valeurs en entrée, suivie d'une ligne de valeurs en sortie. Si nous voulons distinguer trois langues différentes (anglais, français et polonais), nous pourrions choisir de représenter ce problème en allouant une variable de sortie 0 pour l'anglais, 0.5 pour le français et 1 pour le polonais. Les réseaux de neurones sont cependant connus pour mieux fonctionner lorsqu'une variable de sortie est allouée pour chaque langue,

Listing 3. *Programme d'apprentissage d'un Réseau de Neurones Artificiels pour distinguer trois langues*

```
#include "fann.h"

int main()
{
  struct fann *ann = fann_create(1, 0.7, 3, 26, 13, 3);
  fann_train_on_file(ann, "frequencies.data", 200,
    10, 0.0001);
  fann_save(ann, "language_classify.net");
  fann_destroy(ann);
  return 0;
}
```

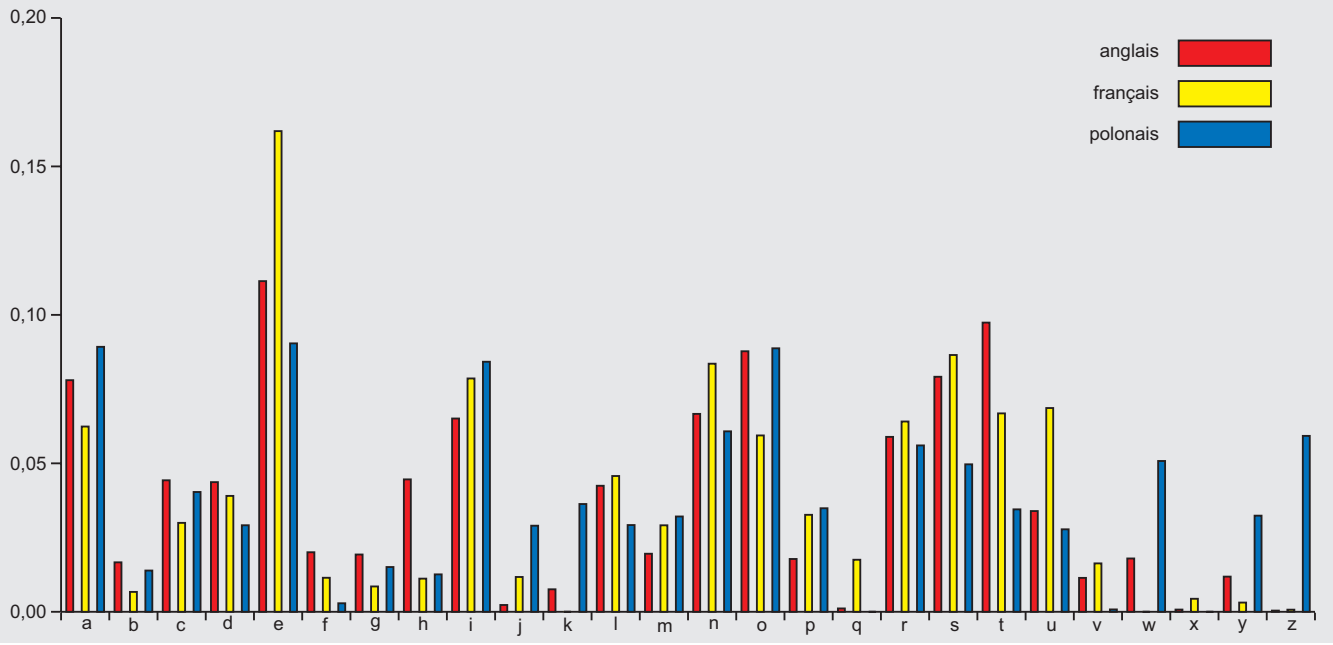

Figure 2. *Graphique à barres présentant les fréquences moyennes pour l'anglais, le français et le polonais*

Listing 4. *Données de sortie d'un Réseau Rapide de Neurones Artificiels au cours de l'apprentissage*

```
Max epochs 200. Desired error: 0.0001000000
Epochs 1. Current error: 0.7464869022
Epochs 10. Current error: 0.7226278782
Epochs 20. Current error: 0.6682052612
Epochs 30. Current error: 0.6573708057
Epochs 40. Current error: 0.5314316154
Epochs 50. Current error: 0.0589125119
Epochs 57. Current error: 0.0000702030
```

et lorsque cette variable est fixée à 1 pour la langue correcte et 0 dans les autres cas.

Grâce à ce petit programme, un fichier d'apprentissage contenant les fréquences de chaque lettre peut être généré pour des textes écrits dans différentes langues. Le Réseau de Neurones Artificiels sera bien entendu plus performant pour distinguer les langues si les fréquences d'un grand nombre de textes différents sont disponibles dans le fichier d'apprentissage, mais pour de petits exemples, 3 à 4 textes dans chaque langue devraient suffire. Un fichier d'apprentissage pré-généré utilisant 4 fichiers texte pour trois langues différentes est exposé dans le Listing 2 et une représentation graphique des fréquences présentes dans le fichier est montrée dans la Figure 2. Une analyse approfondie de ce fichier fait ressortir des tendances nettes : l'anglais présente ainsi plus de H que les deux autres langues, le français ne présente presque pas de K et le polonais a plus de W et de Z que les deux autres langues. Le fichier d'apprentissage ne fait appel qu'aux lettres comprises entre A et Z, mais, puisqu'une langue comme le polonais utilise des lettres telles que Ł, Ą et Ę qui ne sont pas présentes dans les deux autres langues, un Réseau de Neurones Artificiels plus précis pourrait être tout aussi bien composé d'un ajout de neurones d'entrée pour ces autres lettres. Lorsque le problème ne consiste qu'à comparer trois langues, il n'est toutefois pas nécessaire d'ajouter ces lettres puisque les lettres classiques contiennent assez d'informations pour classer les langues correctement. Toutefois, si un Réseau de Neurones Artificiels est supposé classer des centaines de langues différentes, plus de lettres en entrée seraient nécessaires.

Avec un fichier d'apprentissage de ce genre, il est très facile de créer un programme à l'aide de la bibliothèque FANN qui peut être utilisée pour apprendre à un Réseau de Neurones Artificiels à distinguer trois langues. Le Listing 3 illustre la simplicité avec laquelle on peut utiliser la bibliothèque FANN pour ce programme. Ce programme utilise quatre fonctions de la bibliothèque FANN `fann_create`, `fann_train_on_file`, `fann_save` et `fann_destroy`. La fonction `struct fann* fann_create(float connection_rate, float learning_rate, unsigned int num_layers, ...)` est utilisée pour créer un Réseau de Neurones Artificiels dans lequel le paramètre `connection_rate` peut être utilisé pour créer un Réseau de Neurones Artificiels qui n'est pas connecté dans son intégralité, bien que des Réseaux de Neurones Artificiels complètement connectés sont généralement préférables, et le paramètre `learning_rate` est utilisé pour spécifier le degré voulu d'agressivité de l'algorithme d'apprentissage (paramètre pertinent uniquement pour quelques algorithmes d'apprentissage). Le dernier paramètre de la fonction `fann_create` est utilisé pour définir la disposition des couches dans le Réseau de Neurones Artificiels. Dans notre cas, nous avons choisi un Réseau de Neurones Artificiels à trois couches (une en entrée, une cachée et une en sortie). La couche d'entrée est dotée de 26 neurones (un pour chaque lettre), la couche de sortie est dotée de trois neurones (un pour chaque langue) et la couche cachée en comporte 13. Le nombre de couches ainsi que le nombre de neurones dans les couches cachées ont été choisis par essai expérimental, et il n'existe quasiment pas de méthode facile pour déterminer ces valeurs. Il est cependant intéressant de noter qu'un Réseau de Neurones Artificiels apprend en ajustant ses poids. Donc, si un Réseau de Neurones Artificiels contient plus de neurones et par conséquent plus de poids, il peut apprendre des problèmes plus complexes. La présence d'un trop grand nombre de poids peut néanmoins se révéler problématique puisque l'apprentissage peut devenir plus complexe. Il existe également le risque qu'un Réseau de Neurones Artificiels n'apprenne que les traits spécifiques des variables d'entrée au lieu des modèles généraux qui peuvent être extrapolés à d'autres ensembles de données afin qu'un Réseau de Neurones Artificiels classe avec exactitude des données absentes d'un ensemble d'apprentissage. En effet, cette aptitude à généraliser est essentielle — sans elle, le Réseau de Neurones Artificiels serait incapable de distinguer les fréquences avec lesquelles il n'a pas suivi son apprentissage.

La fonction `void fann_train_on_file(struct fann *ann, char *filename, unsigned int max_epochs, unsigned int epochs_between_reports, float desired_error)` entraîne le Réseau de Neurones Artificiels. L'apprentissage consiste à ajuster de manière continue les poids de telle sorte que la donnée de sortie du Réseau de Neurones Artificiels correspondra parfaitement à la donnée de sortie du fichier d'apprentissage. On appelle une époque le cycle durant lequel les poids ont été ajustés pour correspondre à la donnée de sortie du fichier d'apprentissage. Dans cet exemple, le nombre maxi-

Listing 5. *Programme classant un texte selon l'une des trois langues (Ce programme utilise quelques fonctions définies dans le Listing 1)*

```
int main(int argc, char* argv[])
{
  if(argc != 2) error("Remember to specify an input file");
  struct fann *ann = fann_create_from_file(
    "language_classify.net");

  float frequencies[26];
  generate_frequencies(argv[1], frequencies);

  float *output = fann_run(ann, frequencies);
  std::cout << "English: " << output[0] << std::endl
  << "French : " << output[1] << std::endl
  << "Polish : " << output[2] << std::endl;

  return 0;
}
```

mum d'époques a été fixé à 200, et un rapport d'état est imprimé toutes les 10 époques. On utilise en général l'erreur quadratique moyenne pour mesurer le degré de proximité du Réseau de Neurones Artificiels par rapport à la donnée de sortie souhaitée. L'erreur quadratique moyenne est la valeur moyenne de la différence au carré entre la donnée de sortie actuelle et la donnée de sortie souhaitée dans le Réseau de Neurones Artificiels, pour des modèles d'apprentissage individuels. Une erreur quadratique moyenne faible signifie que le réseau est proche de la donnée de sortie souhaitée.

Lorsque le programme du Listing 3 est exécuté, le Réseau de Neurones Artificiels va passer par sa phase d'apprentissage et des informations relatives à son état (voir Listing 4) seront imprimées pour faciliter la surveillance de son progrès au cours de l'apprentissage. Après la phase d'apprentissage, le Réseau de Neurones Artificiels pourrait être utilisé directement pour déterminer dans quelle langue un texte est écrit, mais en règle générale, il est préférable de sauvegarder l'apprentissage et l'exécution dans deux programmes distincts de telle sorte que l'énorme quantité de temps passé à l'apprentissage ne soit nécessaire qu'une seule fois. C'est la raison pour laquelle, dans le Listing 3, nous avons sauvegardé uniquement le Réseau de Neurones Artificiels dans un fichier téléchargeable à partir d'un autre programme.

Le petit programme exposé dans le Listing 5 télécharge le Réseau de Neurones Artificiels sauvegardé et l'utilise pour classer un texte selon la langue dans laquelle il est écrit, soit l'anglais , le français ou le polonais. Lorsqu'on applique le réseau sur des textes trouvés sur Internet, il peut classer correctement de courts textes de quelques phrases. Bien que cette méthode pour distinguer des langues ne soit pas très sophistiquée, je n'ai pas trouvé un seul texte que le réseau ne classe pas correctement.

Bibliothèque FANN : détails

L'exemple de classification des langues montre à quel point il est possible d'appliquer facilement la bibliothèque FANN pour résoudre de petits problèmes informatiques communs qui pourraient être bien plus difficiles à traiter avec d'autres méthodes. Malheureusement, tous les problèmes ne peuvent être résolus aussi facilement, et lorsqu'on travaille avec des Réseaux de Neurones Artificiels, on peut souvent se retrouver dans une situation où la phase d'apprentissage pour que le Réseau de Neurones Artificiels donne le résultat correct peut se révéler très ardue. Il arrive parfois que le problème ne puisse tout simplement pas être résolu par un Réseau de Neurones Artificiels, mais l'apprentissage peut bien souvent être amélioré en modifiant légèrement les réglages de la bibliothèque FANN.

La taille du Réseau de Neurones Artificiels est un des facteurs les plus importants lors de la phase d'apprentissage. Elle ne peut être fixée que de façon expérimentale, mais la connaissance du problème permet souvent d'émettre de bonnes hypothèses. Avec un Réseau de Neurones Artificiels d'une taille raisonnable, l'apprentissage peut s'effectuer de plusieurs manières différentes. La bibliothèque FANN supporte plusieurs algorithmes d'apprentissage différents et l'algorithme par défaut (`FANN_TRAIN_RPROP`) n'est pas toujours le mieux adapté pour un problème spécifique. Si tel est le cas, la fonction `fann_set_training_algorithm` peut être utilisée pour changer l'algorithme

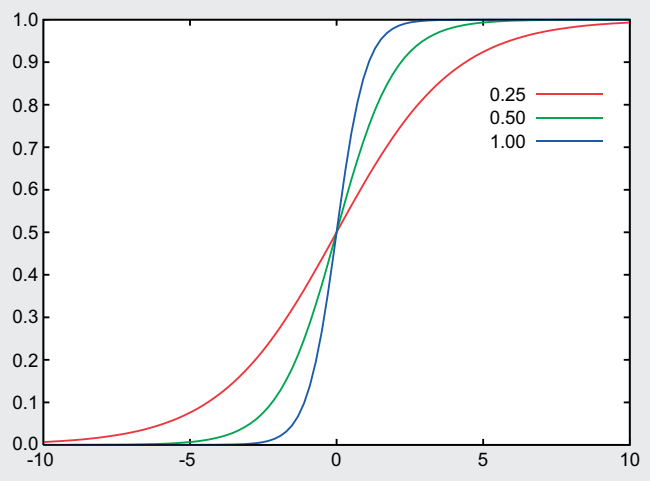

Figure 3. *Graphique illustrant la fonction d'activation sigmoïde pour une pente de 0,25, 0,50 et 1,00.*

d'apprentissage. Il existe dans la version 1.2.0 de la bibliothèque FANN quatre algorithmes d'apprentissage différents, chacun faisant appel à un genre de rétro-propagation. Les algorithmes de rétro-propagation jouent sur les poids en rétro-propageant l'erreur de la couche de sortie à la couche d'entrée tout en ajustant les poids. La valeur de l'erreur ainsi rétro-propagée, pourrait être soit l'erreur calculée pour un seul modèle d'apprentissage (dit incrémentiel), soit la somme des erreurs issues de l'ensemble du fichier d'apprentissage (dit séquentiel). `FANN_TRAIN_INCREMENTAL` implémente un algorithme d'apprentissage incrémentiel, qui modifie les poids après chaque modèle d'apprentissage. L'avantage d'un tel algorithme d'apprentissage est que les poids sont modifiés un grand nombre de fois au cours de chaque époque. Et comme chaque modèle d'apprentissage modifie les poids dans des directions légèrement différentes, l'apprentissage ne se cantonnera pas facilement aux minima locaux, ou à un état dans lequel tous les *petits* changements effectués dans les poids ne feront qu'empirer l'erreur quadratique moyenne, même si la solution optimale n'a pas encore été trouvée. `FANN_TRAIN_BATCH`, `FANN_TRAIN_RPROP` ainsi que `FANN_TRAIN_QUICKPROP` sont tous deux des exemples d'algorithmes d'apprentissage séquentiels qui modifient les poids après avoir calculé les erreurs pour un ensemble d'apprentissage dans son intégralité. Ces algorithmes présentent l'avantage de tirer profit des informations d'optimisation générale absentes de l'algorithme d'apprentissage incrémentiel. Toutefois, il est possible d'ignorer les points les plus fins des modèles d'apprentissage individuels par ce procédé. Il n'y a pas de réponse claire en la matière: quel algorithme d'apprentissage est le mieux adapté? D'une manière générale, les algorithmes sophistiqués d'apprentissage séquentiel tels que *rprop* et *quickprop* sont les mieux adaptés. Toutefois, l'apprentissage incrémentiel peut se révéler dans certains cas plus efficace surtout lorsque de nombreux modèles d'apprentissage sont disponibles. Dans l'exemple d'apprentissage des langues, l'algorithme d'apprentissage le plus efficace est l'algorithme par défaut *rprop* qui atteint la valeur de l'erreur quadratique moyenne souhaitée après seulement 57 époques. L'algorithme d'apprentissage incrémentiel a besoin de 8108 époques pour atteindre le même résultat alors que l'algorithme d'apprentissage séquentiel requiert 91985 époques. L'algorithme d'apprentissage *quickprop* a rencontré plus de problèmes et n'est pas parvenu

à atteindre la valeur de l'erreur souhaitée, mais après avoir modifié la dégradation de l'algorithme *quickprop*, il a fini par atteindre l'erreur souhaitée après 662 époques. La dégradation de l'algorithme *quickprop* est un paramètre utilisé pour contrôler le degré d'agressivité de l'algorithme d'apprentissage *quickprop* et peut être modifié à l'aide de la fonction `fann_set_quickprop_decay`. D'autres fonctions `fann_set_...` peuvent également être utilisées pour fixer de nouveaux paramètres aux algorithmes d'apprentissage individuels, bien que quelques-uns de ces paramètres puissent être un peu plus difficiles à modifier sans connaître au préalable le fonctionnement des algorithmes individuels.

Un paramètre, indépendant de l'algorithme d'apprentissage, peut toutefois être facilement modifié : il s'agit de la pente de la fonction d'activation. La fonction d'activation détermine quand la sortie doit être proche de 0 et quand elle doit être proche de 1, et la pente de cette fonction détermine le degré de transition, en douceur ou abrupte, de 0 à 1. Si la pente de la fonction prend une valeur forte, l'algorithme d'apprentissage convergera plus rapidement vers les valeurs extrêmes de 0 et 1, ce qui accéléra l'apprentissage d'un problème de classification de langues par exemple. Cependant, si la pente prend une valeur faible, il est plus facile d'entraîner un Réseau de Neurones Artificiels qui nécessite une sortie fractionnée. Comme par exemple un Réseau de Neurones Artificiels qui devrait apprendre à trouver la direction d'une ligne dans une image. Deux fonctions dans la bibliothèque FANN, `fann_set_activation_steepness_hidden` et `fann_set_activation_steepness_output`, permettent de régler la pente d'une fonction d'activation. Ces deux fonctions sont fournies parce qu'il est souvent préférable d'avoir différentes pentes pour les couches cachées et les couches de sortie.

Les possibilités de FANN

Le problème d'identification de langues appartient à la classe particulière des problèmes d'approximation fonctionnelle connus sous le nom de problèmes de classification. Les problèmes de classification ont un neurone de sortie par classification et dans chaque modèle d'apprentissage, un de ces neurones de sortie doit être précisément égal à 1. Le problème d'approximation fonctionnelle le plus répandu se rencontre lorsque les données de sortie sont des valeurs fractionnées. Il pourrait s'agir par exemple de la distance d'un objet filmé par une caméra ou même de la consommation d'énergie d'un foyer. Ces problèmes pourraient bien sûr être combinés à des problèmes de classification, afin de pouvoir résoudre un problème de classification consistant à identifier le genre d'un objet sur une image et un problème d'approximation consistant à évaluer la distance de l'objet. Bien souvent un seul Réseau de Neurones Artificiels suffit à résoudre ces problèmes mais il peut s'avérer quelquefois judicieux de traiter les deux problèmes de façon distincte et avoir un Réseau de Neurones Artificiels pour classer par exemple l'objet et un autre réseau pour chaque objet différent qui évalue la distance de l'objet.

Il existe une autre sorte de problème d'approximation : les problèmes liés aux séries chronologiques, qui évaluent une fonction se développant dans la tranche de temps suivante. Un des problèmes de séries chronologiques les plus connus consiste à prévoir le nombre de tâches solaires qu'il y aura dans une année au vue des données historiques. Les fonctions normales sont dotées d'une valeur x en entrée et d'une valeur y en sortie et le problème des tâches solaires pourrait également être défini de la même manière avec l'année en valeur x et le nombre de tâches solaires en valeurs y. Ce procédé, toutefois s'est révélé inefficace pour résoudre des problèmes de séries chronologiques. Ces problèmes peuvent être évalués à l'aide d'une période de temps comme entrée et la période de temps suivante comme sortie. Si la période est fixée à 10 années, le Réseau de Neurones Artificiels pourrait être entraîné à partir de toutes les périodes de 10 ans pour lesquelles des données historiques existent et il pourrait ensuite évaluer le nombre de tâches solaires pour 2005 en utilisant le nombre de tâches solaires des années 1995 – 2004 comme des entrées. Autrement dit, chaque ensemble de données historiques est utilisé par plusieurs modèles d'apprentissage. Par exemple, le nombre de tâches solaires de l'année 1980 est utilisé dans les modèles d'apprentissage ayant les années 1981 – 1990 comme sortie. Cette approche signifie également que le nombre de tâche solaires ne peut pas être directement évalué pour 2010, sans évaluer au préalable les années 2005 – 2009, ce qui signifie par conséquent que la moitié des valeurs d'entrée pour calculer l'année 2010 sera une évaluation et que l'évaluation pour 2010 ne sera pas aussi précise que celle de 2005. C'est la raison pour laquelle les prévisions de séries chronologiques ne sont adaptées que pour le futur proche.

La prévision de séries chronologiques peut également être utilisée afin d'introduire de la mémoire dans les contrôleurs pour les robots etc. Ce qui pourrait s'effectuer par exemple en donnant en entrée du troisième seuil temporel la direction et la rapidité des deux derniers seuils temporels, en plus d'autres entrées issues de capteurs ou de caméras. Toutefois, le principal problème de cette approche est la difficulté de produire des données d'apprentissage puisque chaque modèle d'apprentissage doit également inclure des données historiques.

Bibliothèque FANN : Astuces et pièges

De nombreuses astuces peuvent être utilisées pour entraîner et accélérer l'exécution des Réseaux Rapides de Neurones Artificiels avec une plus grande précision. Une astuce simple qui peut être utilisée pour accélérer et affiner l'apprentissage consiste à rentrer des valeurs d'entrée et de sortie comprises entre -1 et 1 par opposition de 0 à 1. Pour ce faire, il suffit

Sur le réseau

- La bibliothèque FANN
  http://fann.sourceforge.net/
- Steffen Nissen et Evan Nemerson, *Fast Artificial Neural* Network Library Reference Manual
  http://fann.sourceforge.net/fann.html
- Martin Riedmiller et Heinrich Braun, A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm
  http://citeseer.ist.psu.edu/riedmiller93direct.html
- FAQ sur les Réseaux de Neurones Artificiels
  ftp://ftp.sas.com/pub/neural/FAQ.html

de changer les valeurs dans le fichier d'apprentissage en utilisant `fann_set_activation_function_hidden` et `fann_set_activation_function_output` pour changer la fonction d'activation en `FANN_SIGMOID_SYMMETRIC` dotée de données de sortie comprises entre -1 et 1 au lieu de 0 et 1. Cette astuce fonctionne parce que les valeurs 0 dans le Réseau de Neurones Artificiels présente ce défaut qui, quelle que soit la valeur des poids, envoie toujours en sortie une valeur 0. Il existe bien sûr des contre-mesures dans les Réseaux Rapides de Neurones Artificiels qui empêchent ce phénomène de devenir un gros problème. Toutefois, il s'avère que cette astuce réduit aussi le temps d'apprentissage. La fonction `fann_set_activation_function_output` peut également être utilisée afin de changer la fonction d'activation pour une autre, `FANN_LINEAR`, qui est illimitée et peut être ainsi utilisée pour créer des Réseaux de Neurones Artificiels avec des données arbitraires.

Pendant la phase d'apprentissage d'un Réseau de Neurones Artificiels, il est souvent difficile de savoir combien d'époques seront nécessaires pour l'apprentissage. Si trop peu d'époques sont utilisées au cours de l'apprentissage, le Réseau de Neurones Artificiels ne sera pas capable de classer les données d'apprentissage. Si, cependant, un trop grand nombre d'itérations sont utilisées, le Réseau de Neurones Artificiels sera trop spécialisé dans les valeurs exactes des données de l'apprentissage et il ne classera pas correctement les données qu'il n'a pas vues au cours de son apprentissage. C'est la raison pour laquelle il est souvent judicieux d'avoir deux ensembles de données d'apprentissage, un appliqué au cours de l'apprentissage en cours, et l'autre appliqué afin de vérifier la qualité du Réseau de Neurones Artificiels en le testant sur des données qu'il n'a pas vues au cours de son apprentissage. La fonction `fann_test_data` peut être utilisée à cette fin combinée à d'autres fonctions, pouvant gérer et manipuler des données d'apprentissage.

Transformer un problème en une fonction facilement assimilable par un Réseau de Neurones Artificiels peut se révéler complexe. Toutefois, quelques conseils d'ordre général peuvent être appliques comme suit :

- Utilisez au moins un neurone d'entrée/de sortie pour chaque unité informative. Autrement dit, dans le cas des systèmes de classification de langues, un neurone d'entrée pour chaque lettre et un neurone de sortie pour chaque langue.
- Représentez l'ensemble des connaissances avec les neurones d'entrée. Si vous savez par exemple que la longueur des mots est importante pour les systèmes de classification de langues, alors vous devriez aussi ajouter un neurone d'entrée pour la longueur des mots (on pourrait également ajouter un neurone d'entrée pour la fréquence des espaces). Si vous savez également que certaines lettres ne sont utilisées que pour certaines langues, il serait donc intéressant d'ajouter un neurone d'entrée supplémentaire qui prendrait la valeur 1 si la lettre est présente dans le texte et 0 si la lettre en est absente. De cette façon, même une seule lettre polonaise présente dans un texte peut suffire à classer ce texte. Vous savez peut-être que certaines langues contiennent plus de voyelles que d'autres et que vous pouvez donc représenter la fréquence des voyelles comme un neurone d'entrée supplémentaire.
- Simplifiez le problème. Si vous voulez par exemple utiliser un Réseau de Neurones Artificiels pour détecter certaines caractéristiques dans une image, il pourrait s'avérer utile de simplifier l'image afin de simplifier le problème à résoudre, puisque l'image brute contiendra toujours beaucoup trop d'informations et le Réseau de Neurones Artificiels aura des difficultés pour filtrer les informations pertinentes. Dans les images, la simplification peut être effectuée en appliquant certains filtres pour lisser l'image, pour en détecter les contours ou les arrêtes, pour graduer les gris etc. D'autres problèmes peuvent être simplifiés en pré-traitant les données de manière différente afin d'ôter les informations inutiles. La simplification peut également être obtenue en divisant un Réseau de Neurones Artificiels en plusieurs problèmes plus faciles à résoudre. Dans les problèmes de classification de langues, un Réseau de Neurones Artificiels pourrait par exemple distinguer les langues européennes des langues asiatiques, alors que deux autres pourraient être utilisés pour classer les langues de façon individuelle dans les deux zones.

Au cours de la phase d'apprentissage, le Réseau de Neurones Artificiels a souvent besoin de beaucoup de temps, et l'exécution peut souvent être encore plus longue, surtout dans des systèmes où le Réseau de Neurones Artificiels doit être exécuté des centaines de fois chaque seconde ou si le Réseau de Neurones Artificiels est particulièrement important. C'est la raison pour laquelle on peut avoir recours à plusieurs mesures afin d'accélérer l'exécution de la bibliothèque FANN par rapport à la normale. Une de ces méthodes consiste à changer la fonction d'activation pour utiliser une fonction d'activation linéaire par paliers, plus rapide lors de l'exécution, mais un peu moins précise également. Réduire le nombre de neurones cachés dans la mesure du possible est préférable, puisque ce procédé réduira le temps passé lors de l'exécution. Une autre méthode consiste toutefois à laisser la bibliothèque FANN s'exécuter toute seule à l'aide d'entiers uniquement, méthode qui ne peut être efficace que sur des systèmes embarqués sans processeurs à virgule flottante. La bibliothèque FANN possède quelques fonctions auxiliaires lui permettant d'être exécutée à l'aide d'entiers, sur des systèmes dépourvus de processeurs à virgule flottante.

Disponibilité de la bibliothèque

Lorsque j'ai commencé par sortir la version 1.0 de la bibliothèque FANN en novembre 2003, je ne savais pas trop à quoi m'attendre, mais je pensais que n'importe qui pouvait utiliser cette nouvelle bibliothèque que je venais de créer. À ma plus grande surprise, beaucoup de personnes ont commencé en réalité à la télécharger et à l'utiliser. Et au bout de quelques mois, ils étaient de plus en plus nombreux à utiliser la bibliothèque FANN, et le statut de cette bibliothèque a évolué: d'abord ne tournant que sous Linux, elle supporte aujourd'hui la grande majorité des compilateurs et des systèmes d'exploitation (y compris MSVC++ et Borland C++). La fonctionnalité de cette bibliothèque a été également largement améliorée, et de nombreux utilisateurs y contribuent chaque jour. Très vite la bibliothèque a été dotée de liens vers PHP, Python, Delphi et Mathematica et a également été acceptée sous la distribution de Debian Linux. ■