Tworzenie sieci neuronowych — to proste

Steffen Nissen

Na CD:

Na płycie CD zamieściliśmy omawianą bibliotekę oraz listingi użyte w tym artykule.

Przez lata filmy science-fiction, takie jak *Ja, Robot* pochodzące z hollywoodzkich wytwórni przedstawiały sztuczną inteligencję (SI) jako wieszcza zagłady. Nic bardziej mylnego. Podczas gdy Hollywood karmi nas opowieściami o naszym nieuchronnym odejściu w niepamięć, ludzie zupełnie niezainteresowani wyniszczeniem naszego gatunku pracowali nad SI po to, by uczynić nasze życie łatwiejszym, bardziej produktywnym, dłuższym. Po prostu lepszym.

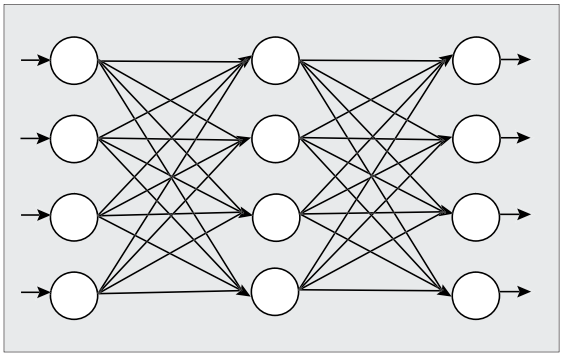
Roboty w filmie *Ja, Robot* dysponują centralnym układem decyzyjnym zbudowanym z sieci sztucznych neuronów; ta sieć neuronowa (SSN) została stworzona w celu wymodelowania prawdziwej sieci neuronowej znajdującej się w ludzkim mózgu. Fast Artificial Neural Network (FANN) jest biblioteką implementującą SSN, którą można wykorzystać w C, C++, PHP, Pythonie, Delphi a nawet w środowisku Mathematica. Mimo iż nie zagwarantuje ona hollywoodzkiej magii, to jednak stanowi potężne narzędzie dla twórców oprogramowania. SSN mogą być użyte w zaskakująco wielu dziedzinach. Tworzenie bardziej realistycznych gier komputerowych, rozpoznawanie obiektów na obrazach czy pomaganie maklerowi w przewidywaniu trendów na nieustannie ulegającej zmianom giełdzie papierów wartościowych. pobieżne, bardzo selektywne wyliczenie nie odda potencjału, jaki drzemie w sieciach neuronowych.

Aproksymacja funkcji

Zasada działania SSN opiera się na aproksymowaniu pewnej funkcji. Innymi słowy uczą się zadanej funkcji poprzez obserwowanie przykładów jej działania. Najprostszym przykładem może być SSN ucząca się funkcji XOR; jednak w ten sam sposób mogłaby ona uczyć się rozpoznawania języka na podstawie tekstu bądź sprawdzania, czy na zdjęciu rentgenowskim znajduje się obraz nowotworu.

Jeśli SSN ma być zdolna do rozwiązywania jakiegoś problemu, zagadnienie musi być zdefiniowane jako pewna funkcja ze zbiorem zmiennych wejściowych i wyjściowych, poparta przykładowymi zestawami danych wejściowych i wyjściowych. Problem taki jak

Steffen Nissen jest informatykiem z Danii. Autor biblioteki FANN, którą nadal rozbudowuje, podczas gdy inni zajmują się utrzymywaniem jej powiązań z innymi językami. Opracował raport techniczny obejmujący proces powstawania biblioteki FANN zatytułowany *Implementation of a Fast Artificial Neural Network Library (fann)*. Kontakt z autorem: *lukesky@diku.dk*



Rysunek 1. *SSN z czterema neuronami wejściowymi, warstwą ukrytą i czterema neuronami wyjściowymi.*

funkcja XOR jest już z założenia funkcją z dwoma zmiennymi na wejściu (o możliwych wartościach 0 i 1) i jedną na wyjściu, a przykłady zdefiniowane są jako cztery różne kombinacje wejściowe. Oczywiście bardziej skomplikowane problemy mogą być trudniejsze do zdefiniowania jako funkcje. Zmiennymi wejściowymi, w przypadku problemu wyszukiwania nowotworu na zdjęciu rentgenowskim, mogą być wartości pikseli na danym obrazie, ale mogą też to być inne informacje uzyskane ze zdjęcia. Na wyjściu może się wtedy pojawić wartość binarna lub zmiennoprzecinkowa, określająca prawdopodobieństwo wystąpienia nowotworu na zdjęciu. W przypadku znormalizowanego rozkładu prawdopodobieństwa wartość zmiennoprzecinkowa przyjmowałaby wartości od 0 do 1 włącznie.

Ludzki mózg

Aproksymator funkcyjny taki jak SSN może być postrzegany jako czarna skrzynka. Właściwie mniej więcej tyle użytkownik biblioteki FANN musi wiedzieć o działaniu aproksymatorów. Jednak, aby zrozumieć jak działają SSN, potrzebna jest podstawowa wiedza o działaniu ludzkiego mózgu.

Mózg ludzki jest bardzo skomplikowanym systemem zdolnym do rozwiązywania wielu nierzadko złożonych problemów. Składa się z wielu różnych elementów, lecz jedną z najistotniejszych jego części jest neuron. Mózg zawiera ich około 10^{11}, a liczba połączeń między poszczególnymi neuronami sięga około 10^{15}, tworząc tym samym ogromną sieć neuronową. Neurony wysyłają do siebie impulsy poprzez połączenia, co stanowi podstawową funkcję tego narządu. Sieć neuronowa otrzymuje też impulsy od pięciu zmysłów i narządów wewnętrznych; wysyła także impulsy do mięśni odpowiedzialnych za ruch czy mowę.

Pojedynczy neuron może być postrzegany jako maszyna dysponująca wejściem i wyjściem, czeka-

jąca na impulsy od połączonych z nim neuronów. Kiedy otrzyma wystarczającą liczbę impulsów, sama wysyła impuls do innych sąsiadów.

Listing 1. *Program mierzący częstotliwość występowania liter A-Z w pliku tekstowym*

```
#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 an 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;
}
```

Sieci Neuronowe

Sztuczne neurony wykazują wiele podobieństw w budowie, jeśli porówna je się do biologicznych protoplastów. Mają połączenia wejściowe, które są sumowane w celu wyznaczenia mocy wyjścia. Wartości mocy wyliczne za pomocą funkcji aktywacji. Choć istnieje wiele funkcji aktywacji, najbardziej popularna jest funkcja sigmoidalna, której wynikiem jest liczba od 0 (dla niskich wartości na wejściu) do 1 (dla wysokich wartości na wejściu). Wynik tego typu funkcji jest następnie przekazywany jako wartość wejściowa do następnych neuronów przez kolejne złącza, z których każde ma swoją wagę statystyczną. Wagi są bardzo istotnym elementem SSN, gdyż określają sposób zachowywania się całej sieci.

Listing 2. *Początkowy fragment pliku trenującego zawierającego częstotliwości występowania liter w języku angielskim, francuskim i polskim; pierwsza linia jest nagłówkiem informującym o tym, że plik zawiera 12 wzorców treningowych składających się z 26 wejść i 3 wyjść*

```
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

0.078 0.013 0.043 0.043 0.113 0.024 0.023 0.041 0.068
0.000 0.005 0.045 0.024 0.069 0.095 0.020 0.001 0.061
0.080 0.090 0.029 0.015 0.014 0.000 0.008 0.000


1 0 0

0.061 0.005 0.028 0.040 0.161 0.019 0.010 0.010 0.066
0.016 0.000 0.035 0.028 0.092 0.061 0.031 0.019 0.059
0.101 0.064 0.076 0.016 0.000 0.002 0.002 0.000

0 1 0

0.092 0.016 0.038 0.025 0.083 0.000 0.015 0.009 0.087
0.030 0.040 0.032 0.033 0.063 0.085 0.033 0.000 0.049
0.053 0.033 0.025 0.000 0.053 0.000 0.038 0.067

0 0 1
...
```

W ludzkim mózgu neurony połączone są ze sobą w sposób robiący wrażenie przypadkowego. Co więcej, wysyłają impulsy asynchronicznie. Jeśli chcielibyśmy dokładnie odwzorować mózg, to właśnie te cechy musiałby być uwzględnionie. Jednak SSN są z reguły zorganizowane w inny sposób, z tego względu że chodzi nam przede wszystkim o zbudowanie aproksymatora funkcyjnego, a nie dokładne odwzorowanie budowy mózgu.

W SSN neurony są z reguły uporządkowane warstwami z połączeniami pomiędzy kolejnymi. Pierwsza warstwa zawiera neurony wejściowe a ostatnia – wyjściowe. Neurony wejściowe i wyjściowe reprezentują odpowiednio wejście i wyjście funkcji, którą chcemy aproksymować. Między warstwami wejściową a wyjściową istnieje pewna liczba warstw ukrytych, natomiast połączenia (i odpowiadające im wagi) do i z warstw ukrytych określają, jak dobrze sieć radzi sobie z danym zagadnieniem. Kiedy SSN uczy się aproksymować pewną funkcję, musi otrzymać przykłady działania tej funkcji. Na tej podstawie SSN powoli zmienia swoje wagi tak, aby wyprodukować wyniki identyczne z wynikami podanymi w przykładach. Jest wtedy nadzieja, że kiedy SSN otrzyma inny zestaw wartości wejściowych również wyprodukuje poprawne wyniki. Zatem, jeśli SSN ma za zadanie nauczyć się wykrywania nowotworów na zdjęciach rentgenowskich, otrzyma na wstępie wiele obrazów zawierających nowotwory i wiele obrazów zwierających jedynie zdrowe tkanki. Po pewnym okresie uczenia z tymi obrazami, wagi w SSN powinny zawierać informacje pozwalające na prawidłową identyfikację nowotworów na zdjęciach rentgenowskich, które sieć analizuje po raz pierwszy.

Biblioteka FANN: kurs dla początkujących

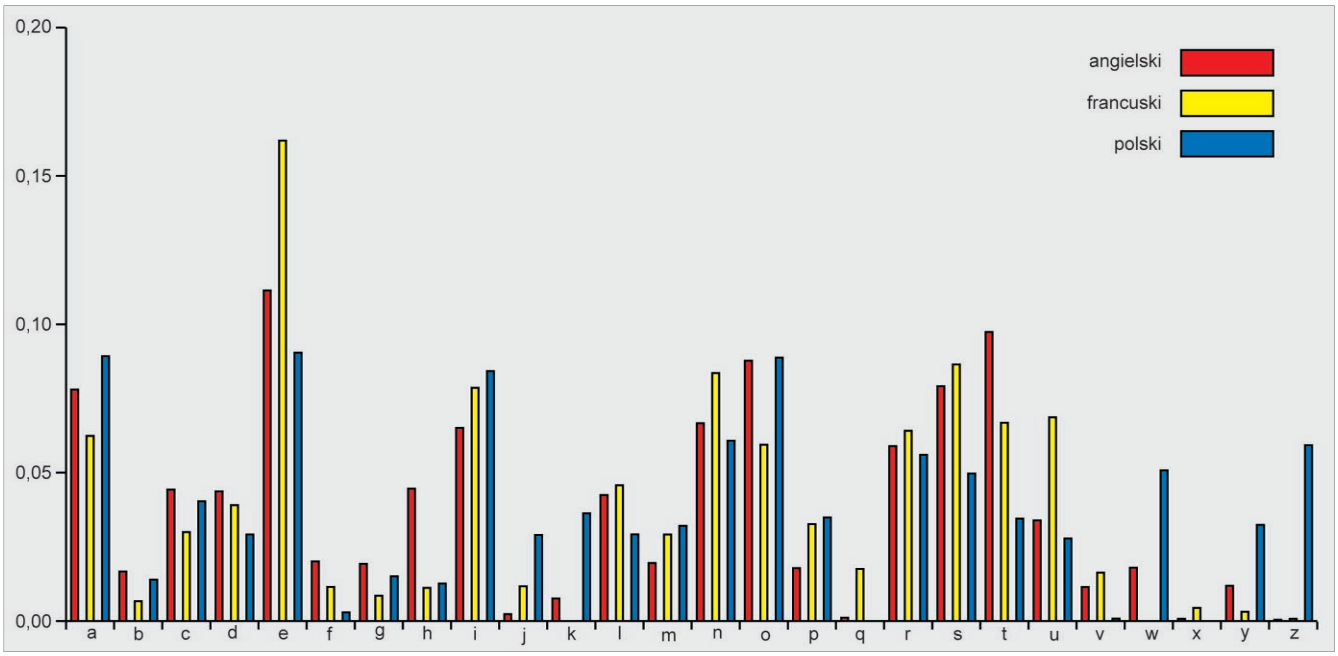
Internet spowodował, że globalna komunikacja stała się częścią życia wielu ludzi, ale również dobitnie wyeksponował problem polegający na tym, że nie każdy posługuje się tym samym językiem. Narzędzia tłumaczące pozwalają w pewnym stopniu na rozwiązanie tego problemu, ale aby mogły one działać poprawnie, muszą wiedzieć, w jakim języku dany tekst został napisany. Jedną z metod na rozpoznanie języka jest analiza częstotliwości występowania w tekście liter alfabetu. Choć może się to wydawać bardzo naiwnym podejściem do rozpoznawania języka, technika ta wielokrotnie się sprawdziła. Dla wielu języków europejskich wystarczy wziąć pod uwagę częstotliwość występowania tylko liter A-Z, pomimo nawet tego, że wiele języków wykorzystuje również inne litery. Z pomocą biblioteki FANN napisanie programu, którego zadaniem będzie określenie, w jakim języku został napisany dany plik tekstowy, jest niezwykle proste. Zastosowana w tym celu SSN powinna mieć neuron wejściowy dla każdej z 26 liter i jeden neuron wyjściowy dla każdego języka. Zanim jednak przystąpimy do pisania właściwego programu, całą zabawę rozpoczniemy od stworzenia małego programu mierzącego częstotliwość występowania liter w pliku tekstowym.

Listing 1. wygeneruje częstotliwości występwania liter w pliku tekstowym i zapisze je w formacie, który następnie zostanie użyty do wygenerowania pliku trenującego dla biblioteki FANN. Pliki treningowe dla biblioteki FANN muszą składać się z linii zawierającej wartości wejściowe, po której

Listing 3. *Program uczący SSN rozpoznawania różnych języków*

```
#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;
}
```



Rysunek *2. Wykres słupkowy przedstawiający uśrednionych częstotliwości występowania liter w językach francuskim, angielskim i polskim*

Listing 4. *Wyjście FANN podczas trenowania*

```
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
```

następuje linia zawierająca wartości wyjściowe. Jeśli chcemy rozpoznawać trzy różne języki (angielski, francuski i polski), moglibyśmy reprezentować je poprzez przyporządkowanie jednej zmiennej wyjściowej przyjmującej wartość 0 dla angielskiego, 0.5 dla francuskiego i 1 dla polskiego. Jednak sieci neuronowe znacznie lepiej sie sprawują, jeśli dla każdego języka zarezerwujemy osobną zmienną wyjściową, która będzie przyjmowała wartość 1 dla swojego języka lub 0 w przeciwnym wypadku.

Mając już gotowy ten prosty program, można wygenerować plik trenujący, zawierający częstotliwości występowania liter w różnych językach. Oczywiście SSN będzie lepiej radziło z rozróżnianiem języków, jeśli w pliku trenującym zostaną umieszczone częstotliwości opracowane na podstawie wielu przykładowych plików tekstowych. Jednak w tym przykładzie wystarczające będzie, jeśli będziemy mieli do dyspozycji po 3-4 teksty przypadające na język. Listing 2. pokazuje zawartość pliku trenującego, wygenerowanego przy użyciu czterech plików tekstowych dla każdego z trzech języków. Natomiast Rysunek 2. prezentuje wykres uśrednionych częstotliwości występowania liter w plikach trenujących. Dokładniejsza analiza pliku wskazuje wyraźne trendy, jako że w angielskim litera H występuje częściej niż pozostałych językach, we francuskim prawie nie występuje litera K, zaś polskim częściej występują litery W i Z niż w pozostałych. Użyty plik treningowy zawiera jedynie litery z zakresu A-Z. Oczywiście można by ulepszyć działanie programu poprzez dodanie neuronów wejściowych odpowiedzialnych za te litery charakterystyczne dla danego języka (w przypadku polskiego: Ą, Ł czy Ś). Jednak z uwagi na to, że porównujemy jedynie trzy języki, nie ma potrzeby dodawania dodatkowych liter, ponieważ rozkłady występowania liter, które są dostępne w alfabecie każdego z tych trzech języków, zawierają wystarczająco dużo informacji by prawidłowo zaklasyfikować język. Aczkolwiek jeśli SSN miałaby klasyfikować setki języków, z pewnością potrzebowalibyśmy więcej liter.

Dysponując takim plikiem trenującym bardzo łatwo jest stworzyć program mający na celu przystosowanie SSN do rozpoznawania trzech języków. Listing 3. pokazuje, jak łatwo jest tego dokonać przy użyciu FANN. Program używa czterech funkcji FANN: `fann_create`, `fann_train_on_file`, `fann_save` i `fann_destroy`. Funkcja `struct fann* fann_create(float connection_rate, float learning_rate, unisigned int num_layers, ...)` jest stosowana do tworzenia SSN, gdzie parametr `connection_rate` może być użyty do utworzenia SSN nie w pełni połączonej (z reguły preferuje się w pełni połączone sieci), zaś `learning_rate` stosowany jest do określenia intensywności, z jaką ma być przeprowadzony proces uczenia (ma to znaczenie tylko dla niektórych algorytmów uczenia). Ostatni parametr funkcji `fann_create` służy do zdefiniowania układu warstw w SNN. W tym wypadku wybrano SSN z trzema warstwami (wejściową, ukrytą i wyjściową). Warstwa wejściowa ma 26 neuronów (jeden na każdą literę), wyjściowa ma ich 3 (jeden na każdy język), zaś ukryta posiada 13 neuronów. Liczba warstw oraz neuronów w warstwie ukrytej zostały dobrane w wyniku przeprowadzenia eksperymentów. Niestety w zasadzie nie ma łatwej metody na dobranie optymalnych wartości. Pomocna może być jednak informacja, że SSN uczy się poprzez modyfikowanie swoich wag. Jeśli zatem SSN zawiera więcej warstw i neuronów, to może nauczyć się rozwiązywania bardziej skomplikowanych problemów. Jednak posiadanie zbyt wielu wag również może okazać się niebezpieczne, ponieważ proces uczenia jest utrudniony i może okazać się, że SSN zostanie wyuczona rozpoznawania tylko specyficznych cech właściwych dla testowych danych zamiast uogólnionych wzorców. Z tego względu, aby SSN poprawnie klasyfikowała dane niepochodzące ze zbioru treningowego, bardzo ważna jest opisana przed chwilą umiejętność generalizowania, gdyż bez niej SSN nie będzie w stanie rozpoznać danych, z którymi wcześniej nie miała do czynienia.

Funkcja `void fann_train_on_file(struct fann *ann, char *filename, unsigned int max_epochs, unsigned int epochs_between_reports, float desired_error)` uczy SSN. Trening wykonywany jest poprzez ciągłe zmienianie wag tak, aby wartości na wyjściu SSN odpowiadały wartościom wyjściowym w pliku trenującym. Jeden cykl, w którym wagi są modyfikowane w celu uzyskania wyniku zgodnego z plikiem trenującym, nazywany jest epoką. W tym przykładzie ustawiono maksymalną liczbę epok na dwieście, a raport o aktualnym statusie generowany jest co dziesięć epok. W celu zmierzenia, w jakim stopniu wynik działania SSN odpowiada oczekiwaniom, używa się uśrednionego błędu kwadratowego. Uśredniony błąd kwadratowy jest wartością średnią z kwadratów różnic pomiędzy faktyczną a pożądaną wartością wyj-

Listing 5. *Program klasyfikujący tekst jako napisany w jednym z trzech języków (program używa niektórych funkcji zdefiniowanych w Listingu 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;
}
```

ściową SSN dla danego wzorca treningowego. Niska wartość uśrednionego błędu kwadratowego oznacza dobrą zbieżność wyniku działania sieci z oczekiwaniami.

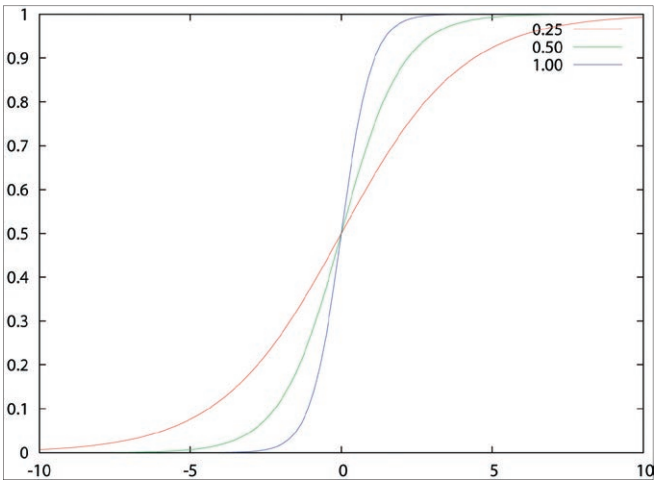
Kiedy program z Listingu 3. zostanie uruchomiony, SSN zostanie wyuczona i pewne informacje dotyczące jej funkcjonowania (patrz Listing 4.) zostaną wyświetlone, aby umożliwić nam monitorowanie postępów w treningu. Po zakończeniu tego procesu SSN mogłaby od razu zostać wykorzystana do zadania, dla którego została stworzona — stwierdzenia, w jakim języku napisano dany tekst. Jednak z reguły lepiej jest rozdzielić część kodu odpowiedzialną za trening oraz za wywoływanie właściwej analizy na dwa programy, gdyż wówczas czasochłonny proces uczenia wykonywany jest tylko raz. Z tego względu program z Listingu 3. po prostu zapisuje parametry określające SSN do pliku, który potem mogą zostać użyte przez inny program.

Mały program z Listingu 5. wczytuje zachowaną SSN i używa jej do klasyfikowania tekstu jako angielski, francuski lub polski. Podczas testowania na tekstach znalezionych w internecie, sieć potrafiła poprawnie sklasyfikować język nawet dla tekstów zawierających jedynie kilka linii. Mimo że ta metoda nie jest niezawodna, nie udało mi się znaleźć nawet jednego tekstu, który zostałby błędnie zaklasyfikowany.

Biblioteka FANN: szczegóły

Przykład z klasyfikacją języków pokazuje, jak łatwo można użyć biblioteki FANN do rozwiązywania prostych, codziennych problemów informatyki, które dużo trudniej rozwiązywałoby się przy użyciu innych technik. Niestety nie wszystkie problemy są tak łatwe do rozwiązania, a podczas pracy z SSN często znajdujemy się w sytuacji, w której ciężko jest wytrenować sieć tak, aby dawała poprawne wyniki. Czasami jest to spowodowane tym, że problemu nie da się rozwiązać przy użyciu SSN. Jednak często można wspomóc proces uczenia poprzez odpowiednią modyfikację parametrów biblioteki FANN.

Najważniejszym czynnikiem mającym wpływ na uczenie SSN jest jej rozmiar. Może on być prawidłowo ustawiony jedynie w drodze eksperymentu. Znajomość problemu często pomaga poprawnie go oszacować. W przypadku SNN o rozsądnych rozmiarach trening może być przeprowadzony na kilka sposobów. Biblioteka FANN oddaje do dyspozycji wiele różnych algorytmów uczenia. Warto pamiętać o tym, że algorytm domyślny (`FANN_TRAIN_RPROP`) nie zawsze jest najlepszy do rozwiązania danego problemu. Jeśli tak jest, można użyć funkcji `fann_set_training_algorithm` do zmiany algorytmu uczenia. W wersji 1.2.0 biblioteki FANN dostępne są cztery różne algorytmy uczenia, przy czym każdy z nich w jakimś stopniu stosuje propagację wsteczną. Algorytmy propagacji wstecznej zmieniają działanie sieci poprzez propagowanie błędu od warstwy wyjściowej do wejściowej i odpowiednie ustawianie napotykanych wag. Propagowana wartość błędu może być wyliczona albo na podstawie błędu pochodzącego od jednego wzorca (*incremental*), albo też na podstawie sumy błędów wynikłych z przejścia przez wszystkie wzorce (*batch*). `FANN_TRAIN_INCREMENTAL` implementuje rosnący algorytm uczący, który zmienia wagi po każdym wzorcu treningowym. Zaletą takiego algorytmu jest to, że wagi są zmieniane wielokrotnie podczas każdej epoki. A ponieważ każdy wzorzec zmienia wagi w trochę innym kierunku, mało prawdopodobne jest, by cały proces uczenia utknął w jakimś minimum lokalnym lub w stanie, w którym niewielka zmiana wag pogorszy średni błąd kwadratowy, mimo że optymalne rozwiązanie nie zostało jeszcze znalezione. `FANN_TRAIN_BATCH`, `FANN_TRAIN_RPROP` i `FANN_TRAIN_QUICKPROP` są przykładami algorytmów uczących typu *batch*. Algorytmy te zmieniają wagi po przeliczeniu błędów dla całego zbioru wzorców treningowych. Zaletą tych algorytmów jest to, że mogą one skorzystać z informacji o optymalizacji globalnej, która nie jest dostępna dla algorytmów typu *incremental*. Może to jednak oznaczać, że niektóre co bardziej subtelne cechy charakterystyczne dla pojedynczych wzorców treningowych mogą zostać pominięte. Nie ma prostej odpowiedzi na pytanie, który algorytm uczący jest najlepszy. Jednak z reguły jeden z zaawansowanych algorytmów uczących tupu *batch,* taki jak *rprop* lub *quickprop,* jest najlepszym rozwiązaniem. Tym niemniej czasami trening typu *incremental* jest bardziej skuteczny, zwłaszcza jeśli mamy do dyspozycji wiele wzorców treningowych. W przykładzie z językami najefektywniejszym algorytmem był domyślny algorytm *rprop*, który osiągnął wyznaczoną wartość średniego błędu kwadratowego juz po 57 epokach. Algorytm `FANN_TRAIN_INCREMENTAL` potrzebował 8108 epok do osiągnięcia tego samego celu, zaś algorytm `FANN_TRAIN_BATCH` aż 91985 epok. Algorytm *quickprop* miał naj-



Rysunek 3. *Wykres przedstawiający sigmoidalną funkcję aktywacji mającą stromiznę o wartości odpowiednio 0.25, 0.50 i 1.00*

Sztuczna Inteligencja

Kiedy kogoś można określić mianem inteligentnego? Czy pies jest inteligentny? A co z nowonarodzonym dzieckiem? Z reguły definiujemy inteligencję jako umiejętność zdobywania i używania wiedzy, wciągania wniosków i wykazywania kreatywności. Jeśli chcielibyśmy użyć tych samych standardów wobec sztucznej inteligencji (SI), doszlibyśmy do wniosku, że na dzień dzisiejszy coś takiego nie istnieje. Zwykle jednak SI definiowana jest jako umiejętność wykonywania zadań, które z reguły utożsamiane są z ludzką inteligencją. Zgodnie z tą definicją terminem SI można oznaczyć wszystkie komputerowe działania związane z przyswajaniem informacji lub wykorzystaniem ludzkiej wiedzy. Definicja ta pozwala na nazwanie mianem SI najprostszego nawet komputera szachowego bądź postaci w grze komputerowej.

więcej problemów i z początku w ogóle nie osiągnął żądanej wartości błędu, lecz po poprawieniu *czasu działania* algorytmu osiągnął wymaganą wartość po 662 epokach. Czas działania (ang. *decay*) algorytmu *quickprop* jest parametrem używanym do kontrolowania agresywności algorytmu i może być zmieniany poprzez funkcję `fann_set_quickprop_decay`. Inne funkcje `fann_set_...` również mogą być używane do modyfikowania parametrów poszczególnych algorytmów uczących, jednakże poprawne ustawienie wartości niektórych z tych parametrów może być dosyć kłopotliwe bez uprzedniej, gruntownej znajomości działania algorytmu.

Jeden parametr, niezależny od wyboru algorytmu uczącego, może być dopasowany w dość prosty sposób. Parametrem tym jest stromizna funkcji aktywacji. Funkcja aktywacji jest funkcją, która określa, kiedy wyjście powinno być zbliżone do 0, a kiedy do 1. Stromizna może być rozumiana (choć nie dosłownie) jako pochodna funkcji aktywacji — precyzuje, jak szybkie ma być przejście z 0 do 1. Jeśli stromizna ma wysoką wartość, algorytm uczący będzie szybciej dążył do wartości ekstremalnych 0 lub 1, co przyspieszy proces uczenia, np. dla problemu rozpoznawania języków. Tym niemniej, jeśli stromizna ustawiona jest na wartość niską, łatwiej jest wytrenować sieć dla przypadku, gdy wymagamy otrzymania na wyjściu wartości ułamkowych. FANN oferuje dwie funkcje umożliwiające ustawienie stromizny funkcji aktywacji. Są nimi: `fann_set_activation_steepness_hidden` oraz `fann_set_activation_steepness_output`. Funkcje są dwie, ponieważ często chcielibyśmy mieć możliwość ustawienia innej stromizny funkcji aktywacji dla warstw ukrytych a innej dla warstwy wyjściowej.

Możliwości FANN

Zagadnienie identyfikacji języków należy do specjalnego rodzaju problemów nazywanych problemami klasyfikacji. Problemy klasyfikacji posiadają jeden neuron wyjściowy na każdą kategorię i przy każdym wzorcu treningowym dokładnie jedna wartość wyjściowa musi być równa 1. Bardziej ogólnym problemem aproksymacji funkcji jest przypadek, w którym wyjścia przyjmują wartości ułamkowe. Może to być np. aproksymowanie odległości do obiektu zarejestrowanego przez kamerę lub nawet zużycia energii w domu. Zadania te mogą oczywiście być łączone z zadaniami klasyfikacji. Możemy zatem mieć zadanie, by zidentyfikować obiekt na obrazie z kamery, a następnie oszacować, w jakiej znajduje się odległości. Niejednokrotnie możemy takie zadanie rozwiązać poprzez jedną SSN, jednak czasami dobrym pomysłem może być rozdzielenie zadań np. stworzenie jednej sieci neuronowej do klasyfikacji obiektu i po jednej sieci na określenie odległości każdego obiektu danej kategorii.

Kolejnym rodzajem zadań aproksymacji to problemy szeregów czasowych, które aproksymują funkcję ewoluującą w czasie. Dobrze znanym zadaniem tego typu jest zadanie polegające na oszacowaniu, ile w danym roku pojawi się plam na Słońcu na podstawie danych historycznych. Zwykłe funkcje biorą wartość x na wejściu i generują wartość y na wyjściu. Zadanie z plamami słonecznymi również mogłoby zostać przedstawione w taki sposób, jeśli za rok przyjmiemy wartość x, a ilość plam jako wartość y. Jednakże okazuje się, że nie jest to najlepsze podejście do rozwiązywania zagadnień związanych z ewolucją w czasie. Znacznie lepszą strategią jest podanie wartości z pewnego przedziału czasu na wejściu i zdefiniowanie wartości dla następnego kroku czasowego jako wyjścia. Jeśli odstęp ustawimy na 10 lat, nasza SSN mogłaby zostać wyuczona przez wszystkie dziesięcioletnie odstępy czasu, dla których istnieją dane historyczne, po czym byłaby w stanie aproksymować liczbę plam na słońcu w roku 2005 na podstawie liczby plam w latach 1995-2004 podanych na wejścia. Z zastosowania takiego podejścia wynika, że każdy zbiór danych historycznych używany jest wielokrotnie we wzorcach trenujących, np. liczba plam z roku 1980 jest używana we wzorcach mających na wyjściu liczbę plam w latach 1981-1990. Podejście takie oznacza również, że ilość plam słonecznych nie może być bezpośrednio wyznaczona dla roku 2010 bez uprzedniego wykonania aproksymacji dla lat 2005-2009. W rezultacie połowa wejść potrzebnych dla obliczeń dla tego roku będzie aproksymacją, a — co za tym idzie — wartość uzyskana dla roku 2010 będzie mniej dokładna niż dla roku 2005. Z tego względu rozważanie szeregów czasowych jest wskazane jedynie dla zdarzeń mających wydarzyć się w niedalekiej przyszłości.

Szeregi czasowe mogą również zostać użyte w celu nadania pamięci robotom. Mogłoby to zostać wykonane np. poprzez podanie kierunku i prędkości z ostatnich dwóch kroków czasowych na wejściu w celu obliczenia kroku następnego, ale oczywiście tylko jako dodatek do innych danych wejściowych takich jak sygnały z czujników lub kamer. Jednakże przy takim podejściu trudności może sprawiać utworzenie wzorców trenujących, jako że każdy z nich musi zawierać dane historyczne.

Sztuczki i porady

Istnieje mnóstwo trików, które mogą zostać użyte w celu przyspieszenia procesów uczenia i właściwej analizy sieci neuronowej, jak również w celu poprawienia dokładności jej działania. Prosty zabieg, który może zostać użyty do przyspieszenia i zwiększenia dokładności procesu uczenia, polega na zmianie wartości wejściowych i wyjściowych z 0 i 1 na -1 i 1. Można tego dokonać poprzez zmianę wartości w pliku trenującym i wywołanie funkcji `fann_set_activation_function_hidden` oraz `fann_set_activation_function_output` w celu zmiany funkcji aktywacji na `FANN_SIGMOID_SYMMETRIC` mającej wartości wyjściowe z przedziału -1 do 1 zamiast od 0 do 1. Trik ten działa, ponieważ war-

W Sieci

- The FANN library
  http://fann.sourceforge.net/
- Steffen Nissen, *Implementation of a Fast Artificial Neural Network Library (fann)*
  http://prdownloads.sourceforge.net/fann/fann_doc_complete_1.0.pdf?download
- Steffen Nissen and Evan Nemerson, *Fast Artificial Neural Network Library Reference Manual*
  http://fann.sourceforge.net/fann.html
- Martin Riedmiller and Heinrich Braun, *A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm*
  http://citeseer.ist.psu.edu/riedmiller93direct.html
- ANN FAQ
  ftp://ftp.sas.com/pub/neural/FAQ.html

tość 0 ma w SSN tę nieszczęsną własność, że bez względu na wagi wyjście zawsze jest równe 0. FANN oczywiście posiada mechanizmy zapobiegające sytuacji, w której stałoby się to dużym problemem. Jednak okazuje się, że zastosowanie tej sztuczki skraca czas uczenia. Funkcja `fann_set_activation_function_output` może być również użyta do zmiany funkcji aktywacji na `FANN_LINEAR`, która jest nieograniczona i może zostać użyta do tworzenia SSN mogących przyjmować dowolne wartości na wyjściu.

Podczas uczenia SSN zwykle trudno oszacować, ile epok jest koniecznych dla prawidłowego funkcjonowania sieci neuronowej. Jeśli użyjemy zbyt mało epok, SSN nie będzie w stanie zaklasyfikować danych treningowych. Z drugiej strony, jeśli użyjemy zbyt wielu epok, SSN stanie się zbyt wyspecjalizowana w rozpoznawaniu danych wzorcowych i nie będzie prawidłowo klasyfikować tych, których wcześniej nie widziała. Z tego powodu zwykle dobrze jest mieć dwa zestawy danych treningowych: jeden stosowany podczas faktycznego procesu uczenia i jeden stosowany w celu zweryfikowania jakości SSN poprzez sprawdzenia jej na danych, które nie zostały użyte podczas treningu. W tym celu może zostać użyta funkcja `fann_test_data` razem z innymi funkcjami służącymi do obsługi i manipulacji danych treningowych.

Wyrażenie konkretnego problemu poprzez funkcję, której SSN mogłaby nauczyć się z łatwością, czasami może okazać się kłopotliwe. Warto zatem kierować się ogólnymi wskazówkami:

- Używaj co najmniej jednego neuronu wejścia/wyjścia na każdą jednostkę informacyjną. W wypadku systemu klasyfikacji języków oznacza to, że mamy jeden neuron wejściowy na każdą literę i jeden wyjściowy na każdy język.
- Podczas określania potrzebnej liczby neuronów wejściowych wykorzystaj całą wiedzę na temat danego zadania, którą posiadasz jako programista. Jeśli na przykład wiesz, że długość słowa jest istotna w systemie klasyfikacji języków, to powinieneś utworzyć dodatkowy neuron wejściowy przyjmujący długość wyrazu (ten sam cel mógłby zostać osiągnięty poprzez dodanie neuronu przyjmującego częstotliwość występowania spacji). Tudzież, jeśli wiesz, że niektóre litery są używane tylko w niektórych językach, to niezłym pomysłem może okazać się dodanie dodatkowego neurony wejściowego, na który podamy 1 jeśli taka litera jest obecna w tekście lub 0 w przeciwnym przypadku. W ten sposób nawet jedna polska litera w całym tekście może pomóc w poprawnym zaklasyfikowaniu tekstu. Niektóre języki zawierają więcej samogłosek niż inne, dlatego też określenie częstotliwości występowania samogłosek jako jeszcze jednego neuronu wejściowego może okazać się pomocne.
- Uprość zadanie. Jeśli na przykład chcesz użyć SSN do wykrycia pewnych cech jakiegoś obrazu, to może dobrym pomysłem byłoby uproszczenie tegoż obrazu tak, aby zadanie stało się łatwiejsze do rozwiązania. Obraz w swojej pierwotnej formie często zawiera zdecydowanie za dużo informacji i SSN nie będzie łatwo uzyskać z niego istotnych informacji. Uproszczenie obrazu może zostać osiągnięte przez zastosowanie filtrów do wygładzania, wykrywania konturów, wykrywania krawędzi, odcieni szarości itp. Inne problemy mogą zostać uproszczone na drodze wstępnej obróbki danych innymi metodami, mającymi na celu usunięcie zbędnych informacji. Uproszczeń można dokonywać również poprzez rozbijanie złożonych problemów na wiele, łatwiejszych do rozwiązania zadań. W systemie klasyfikacji języków jedna SSN mogłaby zostać użyta do rozdzielania języków na europejskie i azjatyckie, a dwie następne do rozpoznawania konkretnych języków z tych obszarów.

O ile proces uczenia SSN zwykle zabiera wiele czasu — i z tym trzeba się pogodzić — o tyle czas potrzebny do przeprowadzenia właściwej analizy powinien być jak najkrótszy, zwłaszcza w systemach korzystających z bardzo złożonych sieci neuronowych bądź wykonujących analizy setki razy na sekundę. Istnieje wiele działań zmierzających do usprawnienia FANN tak, by działała jeszcze szybciej niż do tej pory. Jedną z takich metod jest wybranie liniowej funkcji skokowej jako funkcji aktywacji, która szybciej się wykonuje, ale jest również mniej dokładna. Dobrym pomysłem jest również zredukowanie liczby ukrytych neuronów na tyle, na ile to możliwe, bo to również zmniejszy czas wykonania. Kolejną metodą, choć efektywna jedynie na systemach wbudowanych nieposiadających jednostki liczb zmiennoprzecinkowych, jest uruchomienie FANN tak, aby używała jedynie liczb całkowitych. Biblioteka FANN ma kilka funkcji pomocniczych, które pozwalają na korzystanie z jej z użyciem jedynie liczb całkowitych. W systemach nieposiadających układów odpowiedzialnych z operacje zmiennoprzecinkowe można w ten sposób poprawić wydajność nawet o więcej niż 5000%!

Opowieść ze świata Open Source

Kiedy po raz pierwszy w listopadzie 2003 roku udostępniłem bibliotekę FANN w wersji 1.0, nie bardzo wiedziałem, czego się spodziewać. Jednak byłem zdania, że każdy powinien mieć możliwość używania stworzonej przeze mnie biblioteki. Ku memu zdziwieniu ludzie zaczęli ją ściągać i używać. Miesiące mijały, a FANN była używana przez coraz większą liczbę użytkowników.

Z początku dostępna była jedynie pod Linuksem. Obecnie dostępna jest dla większości kompilatorów i systemów operacyjnych (niewyłączając MSVC++ i Borland C++).

Zestaw dodatkowych funkcji biblioteki również został znacząco rozszerzony, a wielu użytkowników zaczęło pracować nad jej dalszym rozwojem. W niedługim czasie biblioteka miała powiązania z PHP, Pythonem, Delphi i Mathematicą oraz została włączona do linuksowej dystrybucji Debian.

Moja praca z FANN i jej użytkownikami zabiera trochę mojego wolnego czasu, jednak jest to praca, której chętnie się poświęcam. FANN daje mi możliwość oddania czegoś z powrotem środowisku Open Source. Dzięki temu mogę jednocześnie pomagać ludziom i robić coś, co lubię.

Nie mogę powiedzieć, że tworzenie oprogramowania Open Source jest czymś, czym powinien zająć się każdy twórca oprogramowania. Powiem jednak, że mi dało to ogromną satysfakcję. Jeśli myślisz, że to może być coś dla ciebie, to znajdź jakiś projekt, przy którym miałbyś ochotę popracować, i zrób to. Albo jeszcze lepiej – stwórz swój własny projekt Open Source. ■