Politechnika Wrocławska
Wydział Elektroniki
Informatyka

Podstawy obliczeń neuronowych

Sieć neuronowa odgadująca flagi państwowe.

Autorzy:
Łukasz Matysiak
Kamil Machnicki

Prowadzący:
Dr inż. Edward Puchała

Termin zajęć:
Wtorek TP, godz. 13:15

Wrocław
3 stycznia 2017

Spis treści

Rozdział 1

Wstęp

1.1 Cele i założenia projektowe

Celem projektu było zapoznanie się z tematem obliczeń neuronowych i zastosowanie ich do wybranego problemu. Realizacja projektu miała umożliwić zrozumienie zasady działania sieci neuronowych, wpływ zbioru uczącego oraz parametrów sieci na efektywność uczenia oraz sposoby testowania nauczonego algorytmu i generowanie zbioru testowego.

1.2 Opis projektu

Jako przedmiot zainteresowania wybrane zostało zagadnienie stworzenia gry typu człowiek-komputer, w której to gracz wybierałby flagę istniejącego obecnie państwa, zaś algorytm zadając odpowiednie pytania starałby się zgadnąć flaga którego państwa została wybrana.

Głównym założeniem projektu stało się sprawdzenie, czy sieć neuronowa sprawdzi się przy zagadnieniu stworzenia w miarę interaktywnej rozgrywki z graczem oraz czy nauczony algorytm wykazałby się wysoką dokładnością przy zgadywaniu. Problemem tutaj byłoby odpowiednie dobranie parametrów sieci tak, aby jej efektywność w zgadywaniu była największa.

Kolejnym wyzwaniem było odpowiednie dobranie zbioru uczącego. Zbiór taki musiał zapewnić maksymalną efektywność w odnajdywaniu odpowiedzi oraz posiadać jak najmniejszą liczbę nieskorelowanych danych, nie związanych z rozpatrywanym problemem.

Jako dodatkowy cel postawiono również na stworzenie graficznego interfejsu dla aplikacji, aby ułatwić interakcję gracza z komputerem.

Rozdział 2

Realizacja projektu

2.1 Budowa aplikacji

Aplikacja została stworzona przy wykorzystaniu języka `Python` w wersji `3.5`. Do sieci neuronowej wykorzystano moduł `scikit-learn` w wersji `0.18.dev0`, zaś do zbudowania graficznego interfejsu użyto pakietu `Tkinter`.

Na aplikację składa się kilka modułów:

- `main.py` - główny moduł uruchamiający aplikację.
- `prototype.py` - moduł testowy oraz wybierający parametry dla sieci neuronowej.
- `dataset.py` - importuje dane z pliku CSV i przechowuje je w wygodnej postaci.
- `consts.py` - zawiera parametry sieci neuronowej i inne dane.

Aby przygotować aplikację do działania, należy dodać katalog projektu do zmiennej środowiskowej `PYTHONPATH`. Można to zrobić za pomocą polecenia:

```
$ export PYTHONPATH="${PYTHONPATH}:${PWD}"
```

Następnie uruchomienie aplikacji odbywa się przy użyciu komendy:

```
python3 main.py
```

2.2 Zbiór uczący

W projekcie bardzo ważny był dobór odpowiedniego zbioru uczącego jako wejście do sieci neuronowej. Wybrany został darmowy zbiór *Flags Data Set* z internetowego zasobu *UCI Machine Learning Repository*[1]. Wybrany zestaw danych obejmuje 194 kraje z całego świata na stan z 1990 roku wraz z 30 atrybutami opisującymi cechy danego kraju jak i wygląd flagi.

Bazowy zbiór należało następnie dostosować do problemu. Przede wszystkim, trzeba było zaktualizować zbiór odejmując z niego nie istniejące już kraje oraz dodać te które doszły, na przykład kraje powstałe po upadku ZSRR bądź Jugosławii. Część krajów wprowadziła zmiany do swoich flag i je również należało poprawić. Ostatecznie zbiór wyniósł 201 krajów.

Następnie usunięto wszystkie cechy nieskorelowane z wyglądem flagi, a dokładniej: kontynent, strefa geograficzna, powierzchnia, liczba ludności, język i religia. Pozostały **23** cechy:

- `bars` - Liczba pionowych pasów
- `stripes` - Liczba poziomych pasów
- `colours` - Liczba różnych kolorów
- `red, green, blue, gold, white, black, orange` - Czy występuje kolor czerwony, zielony, niebieski, złoty, biały, czarny, pomarańczowy
- `mainhue` - Dominujący kolor na fladze
- `circles` - Liczba okręgów
- `crosses` - Liczba pionowych krzyży
- `saltires` - Liczba krzyży św. Andrzeja
- `quarters` - Liczba ćwiartek
- `sunstars` - Liczba symboli Słońca lub gwiazdy
- `crescent` - Czy występuje półksiężyc
- `triangle` - Czy występuje trójkąt
- `icon` - Czy występuje element nieożywiony (symbol/emblemat)
- `animate` - Czy występuje element ożywiony (zwierzę/roślina/człowiek)
- `text` - Czy występuje tekst
- `topleft` - Dominujący kolor w lewym górnym rogu
- `botright` - Dominujący kolor w prawym dolnym

[1] http://archive.ics.uci.edu/ml/datasets/Flags

2.3 Badania

Badania sieci neuronowej przeprowadzono w celu sprawdzenia jakie wartości parametrów sprawdzają się najlepiej dla rozpatrywanego zagadnienia, oraz aby zbadać wpływ zbioru wejściowego na efektywność algorytmu. Badania rozpoczęto od zbudowania sieci neuronowej. Wybrano prosty perceptron wielowarstwowy bazujący na uczeniu nadzorowanym i jako jego realizację wybrano klasyfikator `MLPClassifier`.

Następnie wybrano odpowiednie współczynniki posługując się wiedzą nabytą na poprzednich kursach oraz testując efektywność algorytmu dla różnych wartości danego parametru z zadanego przedziału przy stałych innych współczynnikach. Ostateczne wartości parametrów przedstawia poniższa tabela 2.1.

Tabela 2.1: Parametry algorytmów.

Parametr	**Wartość**	**Opis**
`algorithm`	`l-bfgs`	Algorytm działania (*quasi-Newton method*).
`max_iter`	`500`	Maksymalna liczba iteracji.
`alpha`	`0.88`	Parametr regularyzacji.
`learning_rate`	`constant`	Tempo uczenia.
`activation`	`logistic`	Funkcja aktywacji algorytmu.

W kolejnym kroku przeprowadzono testy mające na celu dobranie odpowiedniej liczby warstw ukrytych oraz liczby neuronów w każdej. Badania wykonano zarówno dla zbioru bazowego, bez przeprowadzonych modyfikacji opisanych w rozdziale 2.2, jak i dla zbioru przerobionego. Stwierdzono, że usunięcie zbędnych danych pozwoliło na znaczący wzrost efektywności nauczonej sieci.

Efektywność algorytmu sprawdzano testując go na tym samym zbiorze, na którym został nauczony poprzez liczenie liczby poprawnych odpowiedzi, którą to następnie dzielono przez całkowitą liczbę instancji, otrzymując tym samym procentowy wskaźnik.

Dla zbioru bazowego zastosowano jedną oraz dwie warstwy ukryte, gdzie dla każdej z nich przetestowano liczbę neuronów z przedziału $\langle 1, 25 \rangle$. Wyniki testu pokazały, że największą efektywność wynoszącą **63.4%** uzyskano dla 15 oraz 18 neuronów dla odpowiednio pierwszej i drugiej warstwy ukrytej.

Listing 2.1: Wynik badań dla zbioru bazowego

```
Highest score:
1st hidden layer size: 15
2nd hidden layer size: 18
Number of correct answers: 123
Total number of questions: 194
Effectiveness: 63.4%
```

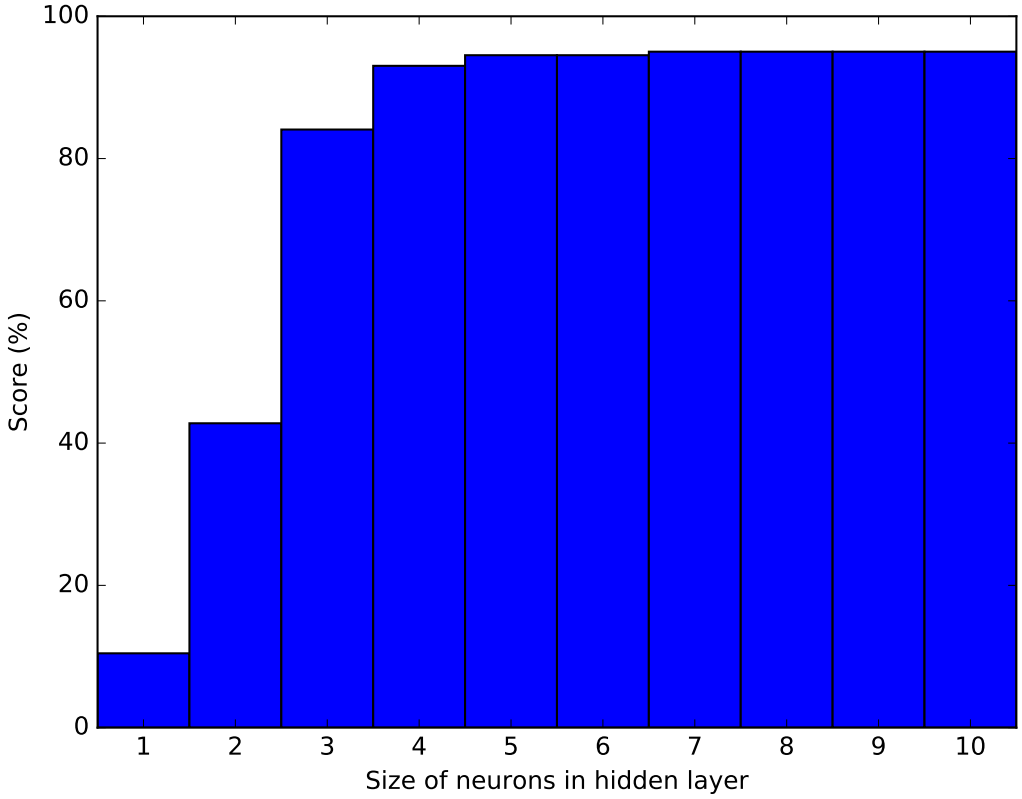
Wynik efektywności algorytmu dla przeprowadzonego testu nie był zadowalający, dlatego też przeprowadzono badanie dla zbioru zmodyfikowanego z usuniętymi nieskorelowanymi danymi. Wyniki pokazały, że aby osiągnąć dużą dokładność wystarczy tylko jedna warstwa ukryta i już przy 3 neuronach udało się osiągnąć większą efektywność niż dla zbioru bazowego. Od 7 neuronów udało się otrzymać dokładność rzędu **95.0%** i była ona od tego momentu stała przy zwiększaniu liczby neuronów. 7 neuronów zastosowano zatem do klasyfikatora do finalnej wersji aplikacji.

Listing 2.2: Wynik badań dla zbioru zmodyfikowanego

```
Highest score:
1st hidden layer size: 7
Number of correct answers: 191
Total number of questions: 201
Effectiveness: 95.0%
```

Tabela 2.2: Procentowa dokładność dla różnych wartości neuronów w jednej warstwie ukrytej

Liczba neuronów	1	2	3	4	5	6	7	8	9	10
Efektywność (%)	10.4	42.8	84.1	93.0	94.5	94.5	95.0	95.0	95.0	95.0



Rysunek 2.1: Wpływ liczby neuronów w jednej warstwie ukrytej na efektywność algorytmu

Nauczony algorytm radził sobie bardzo dobrze z odgadywaniem państw, gdyż większość z nich odgadywana była z ponad 90% prawdopodobieństwem (do testu w dalszym ciągu używano zbioru wejściowego), co zbadano za pomocą funkcji `predict_proba` udostępnioną przez klasyfikator, która to ukazuje procentowy rozkład pewności klasyfikacji wszystkich klas. Dla sprawdzenia wybierane były 3 klasy z najwyższą pewnością.

Listing 2.3: Przykładowy rozkład pewności klas dla poprawnego wskazania państwa Nepal

```
Country: Nepal
1st probability: 94.0% of country: Nepal
2nd probability: 2.0% of country: Italy
3rd probability: 1.8% of country: Israel
```

Uprzednie testy wykazały 95.0% efektywność przy odgadywaniu, dlaczego więc nie udało się osiągnąć 100%? Jest to związane z tym, że część krajów ma bardzo podobne do siebie flagi i cechy jakimi były one określane okazały się identyczne. Sprawiło to, że algorytm oznaczał z takim samym prawdopodobieństwem wystąpienie i jednego i drugiego kraju, co spowodowało obniżenie ogólnej efektywności. Przykładami takich krajów są chociażby Luksemburg i Holandia, Indonezja i Maroko czy Dania oraz Szwajcaria.

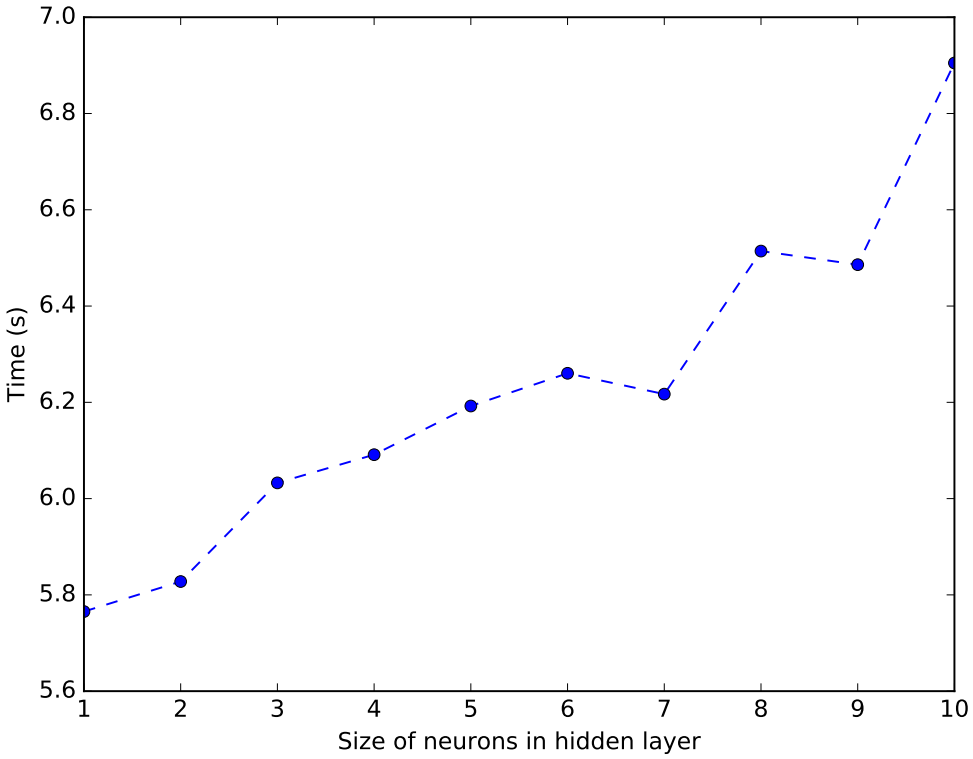
Listing 2.4: Przykładowy rozkład pewności klas dla niepoprawnego wskazania Szwajcarii

```
Country: Switzerland
1st probability: 35.9% of country: Denmark
2nd probability: 35.9% of country: Switzerland
3rd probability: 5.4% of country: Norway
```

Po badaniach efektywności nauczonego algorytmu, sprawdzono również czasy uczenia dla różnej liczby neuronów w warstwie ukrytej. Stwierdzono, że czas uczenia rośnie ze złożonością zbliżoną do stałej, oraz potwierdziły słuszność wybrania 7 neuronów, gdyż stosunek jakości algorytmu do czasu jego nauczania był optymalny. Nie było sensu zatem wybierania wyższej liczby, gdyż efektywność dla nich była taka sama jak dla 7, zaś czasy uczenia rosły.

Tabela 2.3: Czasy uczenia dla różnych wartości neuronów w jednej warstwie ukrytej

Liczba neuronów	1	2	3	4	5	6	7	8	9	10
Czas uczenia (s)	5.77	5.83	6.03	6.09	6.19	6.26	6.21	6.51	6.49	6.9



Rysunek 2.2: Wykres czasów uczenia dla różnych wartości neuronów w jednej warstwie ukrytej

Do tej pory wszystkie badania przeprowadzano testując algorytm na tym samym zestawie danych, na którym był on uczony. Postanowiono sprawdzić jakie uzyska się wyniki gdy zestaw zostanie odpowiednio zdywersyfikowany. Dla każdej z 201 próbek losowo wybrano 2 z 23 cech, które zmodyfikowano tak, że gdy cecha miała wartość > 0, zmieniano ją na 0, zaś dla 0 ustawiano wartość 1. Operację wykonano po 10 razy dla każdej klasy, za każdym razem zapisując jako nową próbkę, co poskutkowało wytworzeniem zbioru o wielkości 2010 próbek. Następnie przeprowadzono testy odpowiednio zmieniając zbiory podczas uczenia oraz testowania.

Tabela 2.4: Porównanie efektywności uczenia dla różnych zbiorów uczących i testowych

Zbiór uczący	Bazowy	Bazowy	Wygenerowany	Wygenerowany	Wygenerowany
Zbiór testowy	Bazowy	Wygenerowany	Wygenerowany	Bazowy	Inny wygenerowany
Efektywność (%)	**95.0**	**64.7**	**82.4**	**91.0**	**65.1**

Jak widać po wynikach przeprowadzonych testów, żaden test nie zbliżył się do efektywności, jaką cechuje testowanie na bazowym zbiorze uczącym, czyli 95%. Jednakże czynione jest tutaj założenie, że gracz odpowie poprawnie na wszystkie pytania. Sprawdzenie na zbiorze wygenerowanym, czyli takim, w którym założone są pomyłki, cechuje się niestety bardzo niską efektywnością wynoszącą 64.7%. Testując na zbiorze wygenerowanym o wiele lepiej radzi sobie algorytm nauczony bezpośrednio na nim samym, jednakże nie jest to również zadowalająca efektywność, zaś testowanie go innym zbiorem wygenerowanym powoduje znowuż znaczny spadek efektywności. Do finalnej wersji aplikacji przeznaczono zatem algorytm nauczony na bazowej, niezdywersyfikowanej wersji zbioru.

Na koniec, postanowiono sprawdzić, które atrybuty najbardziej przysłużyły się do poprawnego odgadywania flag. Pozwoliło to odpowiednio ułożyć pytania w kolejności od najbardziej przydatnych do najmniej, zgodnie z przewidywanym zaangażowaniem gracza w poprawne odgadywanie na zadawane pytania, gdzie na początku zdolność skupienia gracza jest najwyższa, zaś po otrzymaniu serii pytań, jego odpowiedzi są przeważnie mniej dokładne.

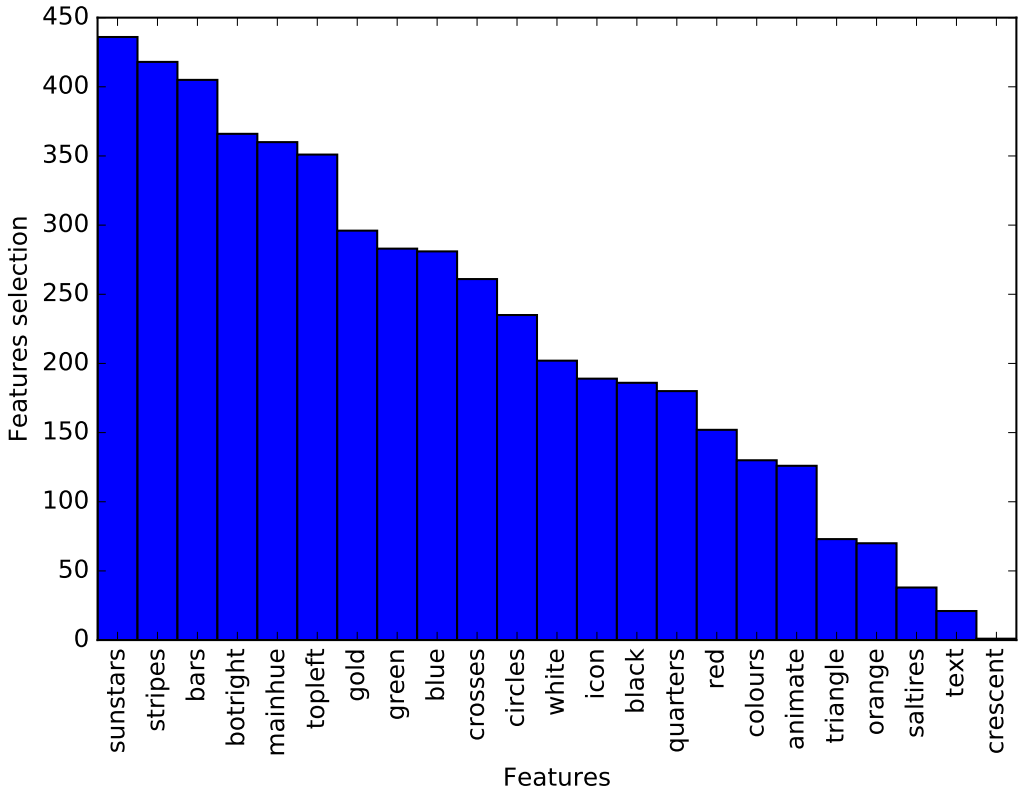
Do wyboru przydatności atrybutów posłużono się selekcją cech, która jest udostępniona metodą `SelectKBest` wybierającą k-najlepszych cech. Wygenerowano zdywersyfikowany zbiór, następnie wybrano $k = \langle 1, 23 \rangle$ najlepszych cech i dodano do siebie częstość występowania danej cechy. Testy przeprowadzono 20 razy, za każdym razem generując nowy zbiór, zaś wyniki uśredniono.

Tabela 2.5: Selekcja cech 1/2

Atrybut	sunstars	stripes	bars	botright	mainhue	topleft	gold	green	blue	crosses	circles	white
Selekcja	436	418	405	366	360	351	296	283	281	261	235	202

Tabela 2.6: Selekcja cech 2/2

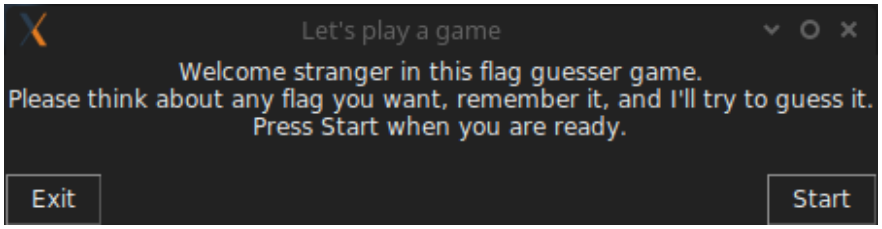
Atrybut	icon	black	quarters	red	colours	animate	triangle	orange	saltires	text	crescent
Selekcja	189	186	180	152	130	126	73	70	38	21	1



Rysunek 2.3: Posortowany ranking cech

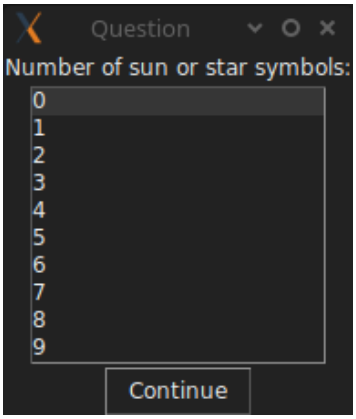
2.4 Opis interfejsu i przebiegu aplikacji

Zgodnie z założeniami dla aplikacji zbudowano graficzny interfejs ułatwiający interakcję gracza z komputerem. Po uruchomieniu programu, pojawia się główne okno, w którym wybrać możemy rozpoczęcie rozgrywki bądź zakończenie działania.

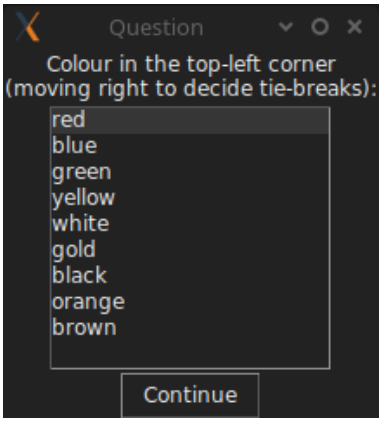


Rysunek 2.4: Główne okno

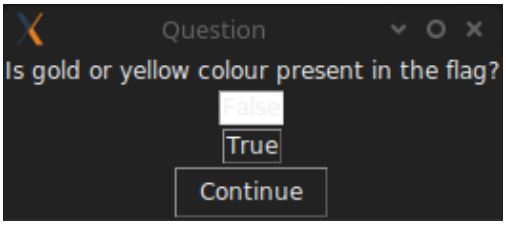
Po rozpoczęciu wyświetlać się będą po kolei pytania dotyczące każdej z cech nauczonego zbioru. Pytania dzielą się na 3 typy: okno wyboru liczby, koloru, bądź prawda/fałsz. Dla ułatwienia można odpowiadać przesuwając odpowiedzi strzałkami góra/dół i zatwierdzając odpowiedź po wciśnięciu klawisza ENTER.



Rysunek 2.5: Okno wyboru liczby

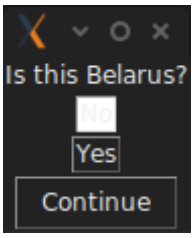


Rysunek 2.6: Okno wyboru koloru



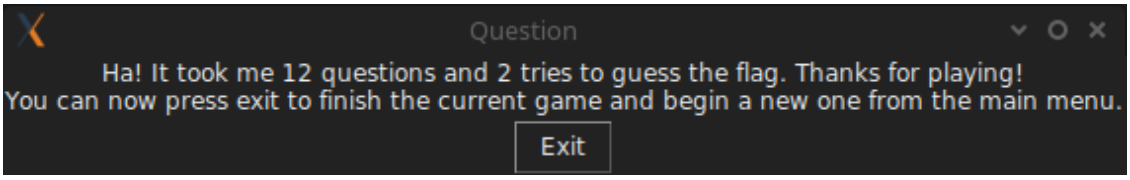
Rysunek 2.7: Okno wyboru prawda/fałsz

Wyświetlanie 23 pytań podzielone jest na 4 sekcje, gdzie co 6 pytań wyświetlane jest zapytanie czy gracz nie ma na myśli państwa, które znajduje się w tym momencie na pierwszym miejscu według rozkładu pewności. Dzięki ułożeniu pytań względem rankingu cech istnieje możliwość odgadnięcia państwa bez przechodzenia przez wszystkie pytania. Cechy, które jeszcze nie pojawiły się w pytaniach ustawiane są na 0.



Rysunek 2.8: Okno pytania o flagę

W momencie, gdy algorytm niepoprawnie odgadł państwo, aplikacja przechodzi do zadawania kolejnych pytań, bądź po przejściu przez wszystkie, przedstawia propozycję kolejnego państwa znajdującego się na kolejnej pozycji w rozkładzie pewności. Po odgadnięciu państwa wyświetlane jest okno końcowe, przedstawiające informację o tym ile pytań zostało do tej pory zadanych oraz ile prób zajęło zgadnięcie flagi.



Rysunek 2.9: Okno końcowe

Po zamknięciu okna końcowego aplikacja wraca do menu głównego, z którego można rozpocząć kolejną rozgrywkę.

Rozdział 3

Podsumowanie

Realizacja tego projektu nauczyła wiele na temat działania sieci neuronowych i zasad operowania na zbiorze uczącym. Okazało się, że algorytmy uczenia maszynowego mogą być zastosowane do stworzenia interaktywnej gry. Bardzo dużo zależy jednakże od dwóch głównych czynników: doboru odpowiednich parametrów sieci oraz posiadania odpowiedniego zbioru uczącego.

Parametry sieci można ustawić bazując na posiadanych doświadczeniach oraz przeprowadzając szereg testów. Możliwych parametrów jest bardzo wiele, lecz najważniejsze są dwa: liczba warstw ukrytych oraz liczba neuronów w każdej warstwie. Projekt pokazał, że nie ma potrzeby inwestowania w wiele warstw o ile zbiór jest odpowiednio dobrany, więc jedna warstwa jest wystarczająca dla tego problemu. Liczbę neuronów najlepiej jest wybierać sprawdzając efektywność algorytmu dla wielu różnych wartości z zadanego przedziału, najlepiej zaczynając od 1 neuronu. Efektywność może być miarą różnej postaci, w tym wypadku został zastosowany iloraz poprawnie odgadniętych państw przez całkowitą liczbę państw. Ważne jest tutaj także sprawdzenie ile czasu zajmuje uczenie algorytmu przy różnych liczbach neuronów, aby wybrać optymalną wartość.

Kolejną bardzo ważną kwestią jest dobranie odpowiedniego zbioru uczącego. W przypadku tego problemu operowano na zbiorze, w którym liczba próbek była taka sama jak liczba klas, co oznaczało że każda klasa charakteryzowała się tylko jednym opisem cech, algorytm zatem bardzo łatwo popadał w lokalne ekstrema. Aby poradzić sobie z tym problemem należy przede wszystkim usunąć wszystkie cechy nieskorelowane bezpośrednio z rozpatrywanym problemem - w tym wypadku dane nie określające flagi, lecz cechy kraju. Można także dodać do zbioru dodatkową liczbę wygenerowanych ręcznie próbek, które to bazują na istniejącym zbiorze.

Warto także mieć na uwadze, że wiele pakietów do uczenia maszynowego udostępnia mnogość różnych narzędzi do operowania czy testowania algorytmów. Bardzo przydatnymi może się okazać chociażby narzędzie do selekcji cech, które można użyć do sprawdzenia, które cechy liczą się najbardziej przy rozpatrywaniu przynależności do klasy. Pozwoli to na przykład usunąć najmniej potrzebne cechy ze zbioru. Przydatnym także jest rozkład pewności, za pomocą którego można sprawdzić z jaką pewnością algorytm wybrał przynależność do danej klasy.

Bibliografia

[1] Feature selection for machine learning in python. http://machinelearningmastery.com/feature-selection-machine-learning-python/. Aktualne dnia: 2017-01-02.

[2] Flags dataset. http://archive.ics.uci.edu/ml/datasets/Flags. Aktualne dnia: 2017-01-02.

[3] Scikit-learn documentation. http://scikit-learn.org/dev/documentation.html. Aktualne dnia: 2017-01-02.

[4] Tactics to combat imbalanced classes in your machine learning dataset. http://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/. Aktualne dnia: 2017-01-02.

[5] Tkinker documentation. https://wiki.python.org/moin/TkInter. Aktualne dnia: 2017-01-02.