



**AGH**

**AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W  
KRAKOWIE**

**WYDZIAŁ FIZYKI I INFORMATYKI**

WYDZIAŁ INFORMATYKI STOSOWANEJ

Praca dyplomowa inżynierska

*Wykorzystanie algorytmu sztucznej inteligencji do  
rozwiązania problemu optymalnego wyboru ścieżki lub  
wyjścia ewakuacyjnego*

Autor:

*Amadeusz Hercog*

Kierunek studiów:

*Informatyka stosowana*

Opiekun pracy:

*mgr. inż. Robert Lubaś*

Kraków, 21 listopada 2018

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

*Serdecznie dziękuję ...tu ciąg  
dalszych podziękowań np. dla  
promotora, żony, sąsiada itp.*



# 1. Wstęp

*W tym rozdziale zostanie opisane podłoże historyczne, cel i jego założenia projektu. Zaprezentowane zostanie proponowane rozwiązanie postawionego problemu oraz technologie do tego użyte.*

## 1.1. Cel projektu

W wielu miastach, gdzie istnieją duże skupiska ludzi prędzej czy później pojawia się problem alokacji przestrzeni na zabudowę. W tym celu nowoczesne budynki użytku publicznego są projektowane pod kątem wykorzystania jak największej przestrzeni dla celów użytkowych, ale i z drugiej strony - dla zapewnienia jak największego bezpieczeństwa ludziom w nich przebywającym. Jednym z czynników określających wchodzącym w skład szeroko pojętego bezpieczeństwa, jest zapewnienie ludziom ścieżek ewakuacyjnych w razie pożaru lub innych wypadków. Z tego powodu celu postawiłem następujący problem - rozwiązanie problemu optymalizacji wyboru ścieżki lub wyjścia ewakuacyjnego ludzi w budynku. Co więcej, w związku z moim wcześniejszym doświadczeniem z tematem uczenia maszynowego, postanowiłem rozwiązać ten problem za pomocą technik machine learning.

## 1.2. Użyte technologie oraz programy

Wszystkie obliczenia oraz rysowanie wykresów zostało zrealizowane w języku programowania **Python** w wersji 3.6 przy użyciu bibliotek:

- **NumPy** - służąca do obliczeń numerycznych,
- **Pandas** - służąca łatwiejszemu zarządzaniu dużą ilością danych,
- **Matplotlib** - służąca do rysowania wykresów,
- **Tensorflow** - używana do głębokiego uczenia maszynowego.

Aby mieć możliwość symulacji ruchu pieszych w budynku, użyłem symulatora **PSP** (*Pederastian Simulation Project*) autorstwa Roberta Lubaś oraz Wojciecha Myśliwiec - opisany szerzej w rozdziale [2](#).

## 1.3. Proponowane rozwiązanie

Rozwiązanie proponowane przeze mnie zostało podzielone na kilka kroków:

1. Wygenerowanie  $n$  zestawów parametrów wejściowych do symulatora *PSP* w celu pozyskania jego danych wyjściowych.
2. Stworzenie modelu uczenia maszynowego na podstawie wygenerowanych zestawów danych wejściowych oraz wyjściowych symulatora i nauczanie go na danych z punktu [1](#). Daje to możliwość generowania nowych danych wyjściowych bez używania symulatora.
3. Optymalizacja zestawu danych wyjściowych modelu na podstawie jego danych wejściowych używając metod optymalizacji.

## 2. PSP - Pederastian Simulator Project

*W tym rozdziale opisany zostanie symulator ruchu pieszych użyty w projekcie i jego sposób użycia wraz z niezbędnymi dodatkowymi plikami. Opisane zostaną użyte w projekcie parametry wejściowe do symulatora oraz metryki stworzone na podstawie jego danych wyjściowych.*

### 2.1. Opis

Symulator *Pederastian Simulator Project* jest programem napisanym w języku C++ autorstwa Roberta Lubaś oraz Wojciecha Myśliwiec. Służy on do modelowania ruchu agentów (ludzi) podczas ewakuacji w danej przestrzeni. Symulowana przestrzeń jest dwuwymiarowa, z możliwością dodania wielu poziomów odpowiadających kolejnym piętrům budynku. Główne okno programu jest widoczne na rysunku [2.1](#).

### 2.2. Sposób użycia

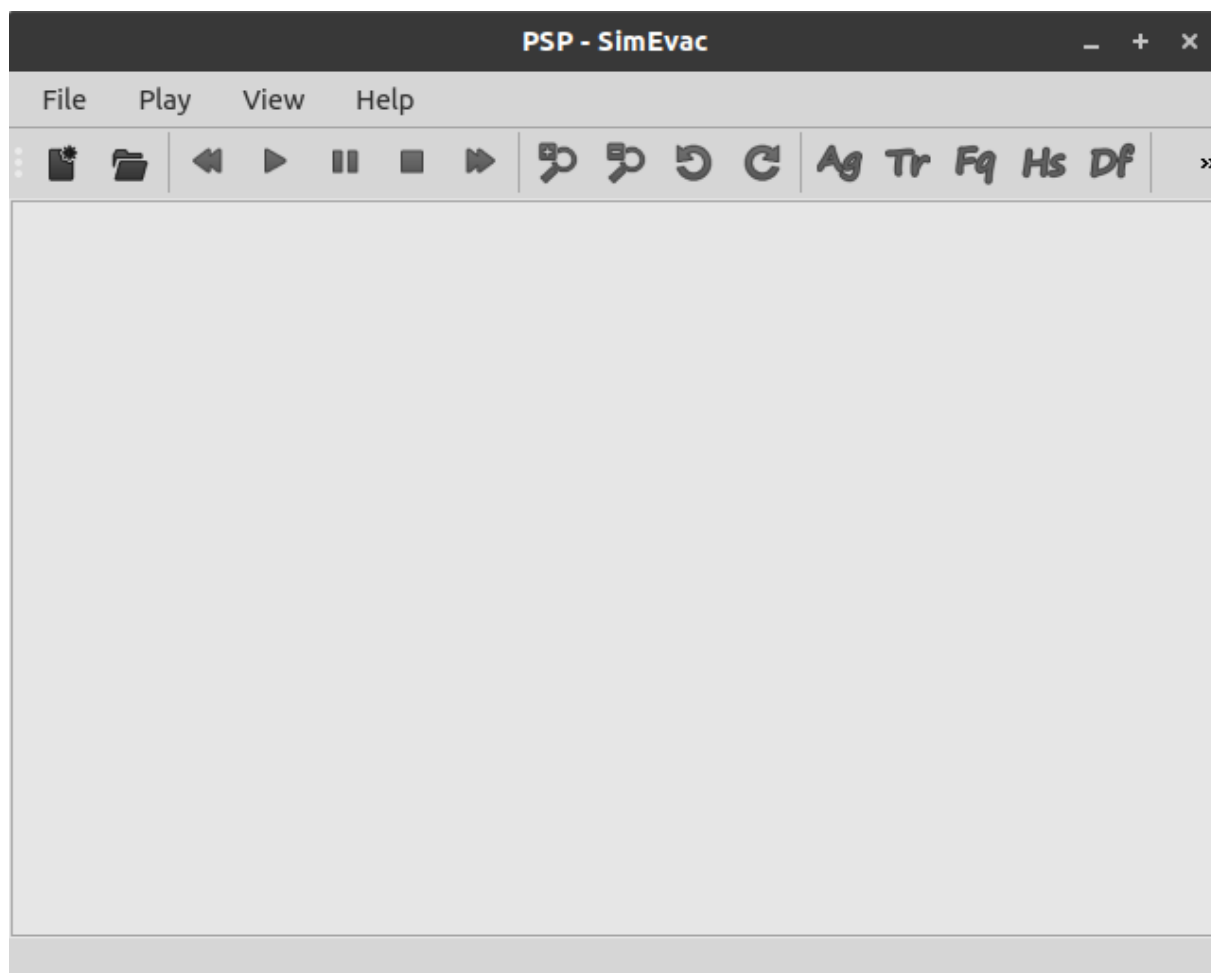
Do przeprowadzenia symulacji potrzebne są następujące rzeczy:

- plik *.xml* zawierający parametry wejściowe symulacji,
- pliki *.jpg* i *.bmp* zawierające przestrzeń użytą do symulacji.

W oknie głównym programu należy wybrać opcję otwarcia pliku *.xml* po czym od razu zaczyna się symulacja w czasie rzeczywistym. Po symulacji, we wskazanym w pliku z parametrami folderze zostaną wygenerowane pliki *.csv* z danymi wyjściowymi symulacji.

### 2.3. Dane wejściowe i wyjściowe

Dane, na których miałem możliwość przeprowadzenia badań znajdowały się w:



**Rys. 2.1.** Główne okno symulatora.

- dane wejściowe - plik *.xml*,
- dane wyjściowe - pliki *.csv*.

Danymi wejściowymi, które wybrałem - na podstawie łatwości modyfikacji - do dalszej analizy były:

- **Panic spread factor** - współczynnik, który decyduje jak duży wpływ ma **tryb paniki** (tryb, w którym agenci zwracają mniejszą uwagę na otoczenie oraz charakteryzują się bardziej chaotycznym ruchem) na agentów.
- **Panic cancel zone** - współczynnik określający odległość od wyjść, w obrębie jakiej agenci mają szansę na deaktywację trybu paniki.
- **Cancel panic chance**- podczas testów na anulowanie trybu paniki określa procent szans na powodzenie.



- **Choosing evacuation path mode** - tryb wyboru drogi ewakuacyjnej przez agentów. Dostępne są 4 tryby:
  - odległości,
  - gęstości przy wyjściu,
  - odległości oraz gęstości przy wyjściu,
  - odległości, gęstości przy wyjściu oraz popularności wyboru wyjścia.
- **Number of pederastians** - liczba agentów biorąca udział w symulacji.
- **Chaos level** - szansa na aktywację trybu paniki u agentów.
- **Density factor** - określa wpływ współczynnika gęstości wokół agenta na funkcję kary.
- **Frequency factor** - określa wpływ częstości wyboru danego pola na funkcję kary.
- **Panic factor** - potęguje część współczynników biorących udział w obliczaniu funkcji kary.
- **Distance factor** - wpływa na wybór agentów ruchu po skosie lub na wprost.
- **Randomness factor** - czynnik losowości dla wartości kary.
- **Pre-movement time mean value** oraz **Pre-movement time standard deviation** - średnia i odchylenie czasu *Pre-movement*, który ma wpływ na częstotliwość wykonywania testu na zmianę obranego wyjścia przez agentów.
- **Speed distribution mean value** oraz **Speed distribution standard deviation** - średnia i odchylenie prędkości agentów.

Dane wyjściowe wybrane do analizy przeze mnie były następujące:

- czas ewakuacji ostatniego agenta (odpowiadający czasowi ewakuacji wszystkich agentów),
- błąd średniokwadratowy średnich prędkości wszystkich agentów w stosunku do średniej prędkości najszybszego agenta.

Dodatkowe parametry wejściowe, które były mi potrzebne (lecz nie wpływały na sam proces badawczy) to:

- **Repeat number** - ilość powtórzeń symulacji wykonanej na pojedynczym pliku *.xml*.
- **Sim directory** - określa ścieżkę do folderu zawierającego pliki symulowanej przestrzeni.
- **Sim stat directory** - określa ścieżkę do folderu, do którego zostaną zapisane dane wyjściowe symulacji (pliki *.csv*).

## 3. Budowa projektu

*W tym rozdziale opisany będzie proces zbierania danych za pomocą symulatora PSP, tworzenie modelu uczenia maszynowego oraz optymalizacji jego wartości wyjściowych.*

### 3.1. Zebranie danych

W celu zebrania danych, które miałyby trafić później do modelu uczenia maszynowego, stworzyłem skrypt pythonowy, który wygenerował 200 różnych zestawów parametrów wejściowych do symulatora. Rodzaje danych, które wygenerowałem znajdują się w rozdziale 2.3. Dla każdego zestawu danych ustawiłem parametrowi *Repeat number* wartość 10, dzięki czemu dla każdego zestawu danych wejściowych generowane było 10 zestawów danych wyjściowych. Było to niezwykle cenne z 2 powodów:

- Do symulatora trzeba było ręcznie ładować plik *.xml*, a dzięki temu z jednego pliku otrzymywałem od razu 10 zestawów plików wyjściowych.
- Ruchy agentów, nawet z takim samym zestawem parametrów, różnią się między symulacjami, a dzięki powtórzeniom łatwiej było wyeliminować te różnice.

Dla części symulacji, symulator niespodziewanie wyłączył się podczas pracy, przez co niektórym zestawom danych wejściowych odpowiada mniej niż 10 zestawów danych wyjściowych. Ostatecznie z 200 zestawów parametrów wejściowych uzyskałem 1888 zestawów parametrów wyjściowych. Oznacza to, że 5,6% z symulacji, które miały zostać przeprowadzone nie udało się.

W pierwszym podejściu, aby móc przetestować model w prostych warunkach, wybrałem z każdego zestawu danych: wszystkie dane wejściowe, jedną daną wyjściową. Daną wyjściową, którą wybrałem był czas ewakuacji ostatniego agenta. Każdy zestaw danych zawierał więc **15** parametrów wejściowych oraz **1** parametr

wyjściowy. Z tak przygotowanymi danymi miałem możliwość wybrania pierwszego modelu.

## 3.2. Opis modelu Perceptron

Na początek, wybrałem prosty model uczenia maszynowego - regresję liniową. Stworzyłem go wykorzystując **Perceptron**. Jest to prosta sieć neuronowa, która posiada zestaw neuronów wejściowych, każdy odpowiadający pojedynczej danej wejściowej. Składa się on także z jednego lub wielu niezależnych neuronów wyjściowych. Każdy z nich jest połączony ze wszystkimi neuronami wejściowymi za pomocą krawędzi posiadających wagi - definiują one jak duży wpływ na wyjście ma każda dana wejściowa. Jeśli wyrazimy:

- wejście jako wektor  $X$ ,
- wyjście pojedynczego Perceptronu jako  $a$ ,
- neuron jako wektor wag  $N$ ,
- funkcję aktywacji jako  $\sigma$  - jest to funkcja wprowadzająca nieliniowość danych, najpopularniejsze z nich to *Sigmoid*, *ReLU (Rectified linear unit)* czy też *Softmax*

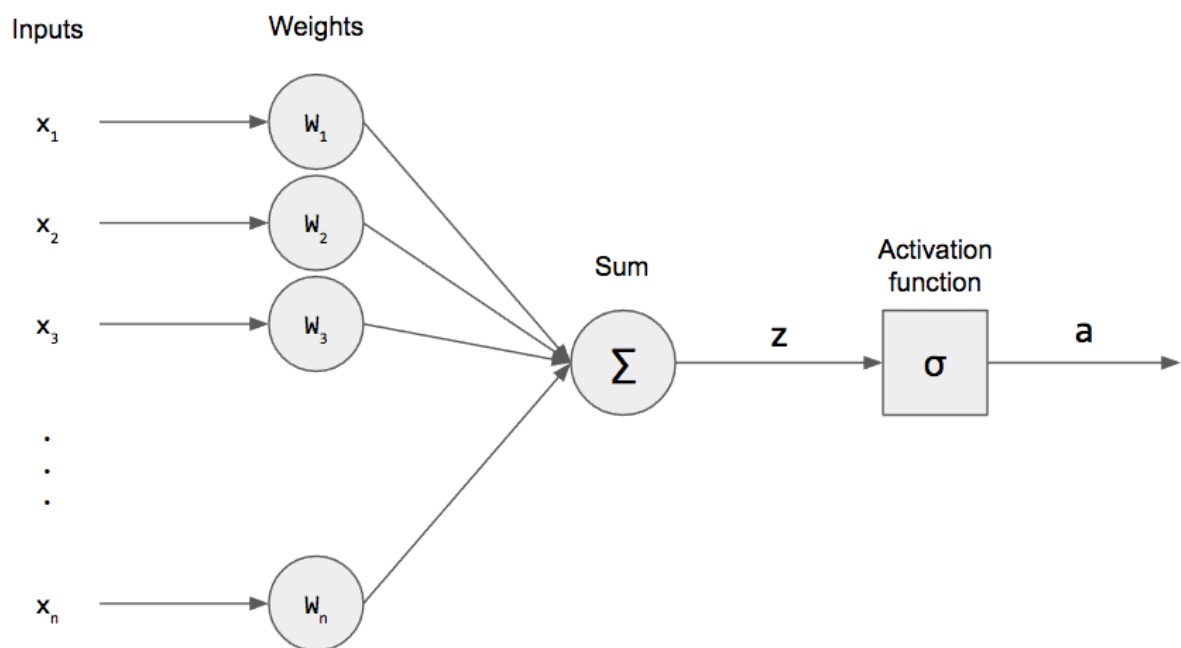
to zachodzi równanie:

$$Y = \sigma(NX) \quad (3.1)$$

Budowa Perceptronu z jednym neuronem wyjściowym jest widoczna na rysunku 3.1<sup>1</sup>.

Podczas procesu uczenia *Perceptronu* ważne jest aby określić **loss function** - metrykę którą chcemy optymalizować. Dla problemów regresji najczęściej używaną funkcją kosztu jest **MSE** (Mean Squared Error). Kolejnym parametrem sieci *Perceptron* jest parametr **learning rate**, który określa jak bardzo będzie się zmieniać macierz/wektor wag na podstawie gradientu podczas każdej iteracji uczącej. Zbyt duży współczynnik *learning rate* może spowodować, że zamiast optymalizować funkcję kosztu będzie wręcz przeciwnie - będzie ona osiągała coraz większe wartości. Gdy jest on zbyt mały, jest on mniej niebezpieczny gdyż nie doprowadzi on do wzrostu wartości funkcji kosztu; problemem jest jednak wolniejszy proces uczący sieci co wymaga więcej epok uczących. Nie ma jednak uniwersalnej metody regulującej sposób dobierania *learning rate* - trzeba to robić eksperymentalnie.

<sup>1</sup> <https://pythonmachinelearning.pro/perceptrons-the-first-neural-networks/>



Rys. 3.1. Budowa Perceptronu z jednym neuronem wyjściowym.

### 3.3. Tworzenie pierwszego modelu

Dla powyższych zastosowań stworzyłem więc *Perceptron* posiadający pojedyncze wyjście oraz nie posiadający funkcji aktywacji - jako że model miał służyć do regresji a nie klasyfikacji. Zestawy danych podzieliłem w stosunku 80% : 20% na **dane treningowe** i **dane testowe**, pozwala to na trenowanie sieci na danych treningowych oraz sprawdzanie jej efektywności na danych testowych co daje możliwość walidacji tego, jak dobrze sieć radzi sobie z danymi, z którymi nie miała styczności.

Podsumowując, parametry pierwszego modelu były więc następujące:

- pojedyncza warstwa składająca się z jednego neuronu,
- **funkcja aktywacji** - brak,
- **learning rate** - 0.1,
- **metryka do optymalizacji** - MSE,
- **liczba epok uczących** - 300,
- podział na **dane treningowe** i **dane testowe** - 80% : 20%.



## Bibliografia

- [1] L. Lamport. *LaTeX system przygotowywania dokumentów*. Kraków: Wydawnictwo Ariel, 1992.
- [2] *Ada Reference Manual ISO/IEC 8652:200y(E) Ed. 3*. Ada Europe. 2006.
- [3] A. Burns i B. Dobbing. „The Ravenscar Profile for Real–Time and High Integrity Systems”. W: *CrossTalk* 16.11 (2003), s. 9–12.