JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 21 (2021) 330–335
Received: 12 September 2021
Accepted: 10 October 2021

Performance analysis of the TensorFlow library with different optimisation algorithms

Analiza wydajności biblioteki TensorFlow z wykorzystaniem różnych algorytmów optymalizacji

Maciej Wadas*, Jakub Smołka

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
This paper presents the results of performance analysis of the Tensorflow library used in machine learning and deep neural networks. The analysis focuses on comparing the parameters obtained when training the neural network model for optimization algorithms: Adam, Nadam, AdaMax, AdaDelta, AdaGrad. Special attention has been paid to the differences between the training efficiency on tasks using microprocessor and graphics card. For the study, neural network models were created in order to recognise Polish handwritten characters. The results obtained showed that the most efficient algorithm is AdaMax, while the computer component used during the research only affects the training time of the neural network model used.

Keywords: machine learning; neural networks

Streszczenie
W artykule zaprezentowano wyniki analizy wydajności biblioteki TensorFlow wykorzystywanej w uczeniu maszynowym i głębokich sieciach neuronowych. Analiza skupia się na porównaniu parametrów otrzymanych podczas treningu modelu sieci neuronowej dla algorytmów optymalizacji: Adam, Nadam, AdaMax, AdaDelta, AdaGrad. Zwrócono szczególną uwagę na różnice pomiędzy efektywnością treningu na zadaniach wykorzystujących mikroprocesor i kartę graficzną. Do przeprowadzenia badań utworzono modele sieci neuronowej, której zadaniem było rozpoznawanie znaków języka polskiego pisanych odręcznie. Otrzymane wyniki wykazały, że najwydajniejszym algorytmem jest AdaMax, zaś podzespół komputera wykorzystywany podczas badań wpływa jedynie na czas treningu wykorzystanego modelu sieci neuronowej.

Słowa kluczowe: uczenie maszynowe; sieci neuronowe

*Corresponding author
Email address: maciej.wadas@pollub.edu.pl (M. Wadas)

1. Wstęp

Rozwój sztucznej inteligencji na przestrzeni XXI wieku dowiódł, że istnieje wiele zadań, których wykonanie jest niemożliwe bez ingerencji człowieka. Ludzkość traktuje sztuczną inteligencję jako narzędzie wspomagające ich życie [1]. Obecnie systemy oparte na uczeniu maszynowym otwierają przed ludźmi wiele nowych możliwości, a co za tym idzie wiele nowych problemów i rozwiązań. Na przestrzeni ostatnich lat znaleziono wiele zastosowań sztucznej inteligencji takich jak chatboty, których zadaniem jest replikacja zachowań ludzkich poprzez automatyzację odpowiedzi na często pojawiające się pytania [2]. Implementacja sztucznej inteligencji została zastosowana również w tłumaczeniu języka naturalnego np. w aplikacji Tłumacz Google [3]. Uczenie maszynowe znajduje również zastosowanie w rozpoznawaniu głosu (mowy ludzkiej) czy obrazów [4, 5].

Ze względu na rosnącą liczbę systemów opartych o sztuczną inteligencję na przestrzeni ostatnich lat powstało wiele bibliotek wykorzystywanych w uczeniu maszynowym i głębokich sieciach neuronowych, takich jak SciKit-Learn, Keras, Theano czy TensorFlow [6-9]. Ze względu na rosnącą popularność uczenia maszynowego wielu naukowców z całego świata rozpoczęło badania dotyczące wydajności tego typu technologii. Z czasem uzyskane wyniki rozpoczęły wpływać na rynek informatyczny. Czego skutkiem był wzrost lub spadek popularności wielu narzędzi. Niniejszy artykuł poświęcony jest bibliotece TensorFlow, która jest jedną z najpopularniejszych bibliotek do uczenia maszynowego na rynku [10].

2. Cel badań

Celem badania jest analiza wydajności biblioteki TensorFlow na podstawie parametrów otrzymanych podczas treningu modelu sieci neuronowej. Trening odbędzie się przy pomocy pięciu algorytmów optymalizacji AdaGrad, AdaDelta, Adam, AdaMax, Nadam, dla pojedynczego mikroprocesora (CPU) i karty graficznej (GPU).

Przeprowadzone badania posłużą do zweryfikowania następujących hipotez badawczych:
H1: Najwydajniejszym algorytmem optymalizacji, wśród wybranych, jest Adam.
H2: Urządzenie peryferyjne nie ma wpływu na parametry otrzymane przy treningu modelu.

3. Kategorie uczenia maszynowego

Istnieje wiele różnic między bibliotekami do uczenia maszynowego, jednak w działaniu każdej z nich można wyróżnić powtarzające się schematy postępowania. Widoczne jest to w procesie analizy danych, mającym na celu nauczenie maszyny rozpoznawać pewien wzór występujący w wybranym zestawie danych. Ze względu na to można wyróżnić trzy rodzaje uczenia maszynowego: uczenie nadzorowane, uczenie nienadzorowane, uczenie przez wzmacnianie [11].

W procesie uczenia nadzorowanego (ang. *supervised learning*) uczenie modelu przebiega za pomocą oznakowanych danych uczących (ang. *training data*). Tak wytrenowany model pozwala przewidzieć niewidoczne lub wygenerowane w przyszłości informacje. Człon nadzorowanie odnosi się do zestawu danych wykorzystywanych w treningu, gdzie każdy badany element ma przypisaną etykietę. Przykładowo chcąc wytrenować model danych tak, aby rozpoznawał czy na danym obrazie jest kot czy pies, należy w zestawie oznaczyć, który obraz przedstawia jakie zwierze.

W przypadku pracy z uczeniem przez wzmacnianie (ang. *reinforcement learning*) nie jest wymagane przygotowanie zestawu danych uczących. Celem jest utworzenie systemu (agenta), który poprawia własną skuteczność na podstawie danych otrzymanych w wyniku interakcji ze środowiskiem. Przykładem może być silnik aplikacji gry szachowej. Agent wybiera kolejne ruchy figur na podstawie aktualnego stanu szachownicy (środowiska).

Ostatnim rodzajem uczenia maszynowego jest uczenie nienadzorowane (ang. *unsupervised learning*), gdzie dane treningowe są nieoznakowane lub posiadają nieznaną strukturę. Modele utworzone za pomocą tej techniki uczenia maszynowego pozwalają na poznanie struktury przetwarzanych danych oraz uzyskanie użytecznych informacji bez stosowania znanej zmiennej wyjściowej [11].

4. Architektura biblioteki TensorFlow

TensorFlow jest biblioteką typu open source opracowaną przez firmę Google, która w ciągu ostatnich kilku lat zyskała na popularności. Jest wykorzystywana przez znane firmy takie jak Lenovo, PayPal, inSpace, Intel, Spotify i wiele innych [12]. Platforma TensorFlow cechuje się wieloplatformową strukturą, udostępnia interfejsy API wysokiego poziomu: Keras i Estymator do tworzenia modeli uczenia głębokiego. Na najniższym poziomie operacje TensorFlow są implementowane przy pomocy języka C++ [13].

5. Algorytmy optymalizacji

Biblioteka uczenia maszynowego TensorFlow udostępnia szereg algorytmów optymalizacji. Wybór odpowiedniego ma istotny wpływ na proces uczenia oraz ostateczny uzyskany wynik detekcji na zbiorze walidacyjnym. W niniejszej pracy zostanie porównane pięć algorytmów optymalizacji: Adam, Nadam, AdaMax, AdaDelta, AdaGrad.

Optymalizator Adam jest stochastyczną metodą zejścia gradientowego, która opiera się na adaptacyjnej estymacji momentów pierwszego i drugiego rzędu. Charakteryzuje się tym, że nie przeprowadza optymalizacji funkcji dla wszystkich danych treningowych tylko dla kolejnych partii (ang. *batch*) danych. Ma małe wymagania pamięciowe i jest wydajna obliczeniowo dla problemów dużych pod względem danych [14].

AdaMax jest odmianą algorytmu Adam opartą na normie nieskończoności. Posiada możliwość dostosowania prędkości uczenia do charakterystyki danych, dlatego też często wykorzystywany jest podczas uczenia, gdzie proces jest zmienny w czasie [14].

Algorytm Nadam działa bardzo podobnie do algorytmu Adam. Różnica polega na tym, że Adam postrzegany jest jako połączenie RMSProb i pędu, zaś Nadam łączy algorytm Adam z przyspieszonym gradientem Nesterowa [15].

AdaGrad jest pierwszym algorytmem z rodziny metod gradientowych, dynamicznie wykorzystującym informacje o geometrii danych zaobserwowanych we wcześniejszych iteracjach. Szybkość uczenia algorytmu AdaGrad jest zmienna. AdaGrad utrzymuje wysoki współczynnik uczenia dla rzadko występujących cech, zaś niski w przypadku cech pojawiających się często w zestawie danych [16].

Optymalizator AdaDelta powstał w celu poprawy dwóch głównych wad algorytmu AdaGrad: ciągłego spadku szybkości uczenia się w trakcie treningu oraz konieczności ręcznego wyboru globalnego współczynnika uczenia (ang. *learning rate*) [17].

6. Realizacja badań

Do realizacji badań wymagane jest przygotowanie następujących elementów: środowisko badawcze, zbiór danych treningowych, sieć neuronową, na której podstawie będzie trenowany model. W celu odpowiedzi na hipotezy badawcze zawarte w rozdziale 2 wymagane również jest określenie metody badawczej.

6.1. Środowisko badawcze

W badaniach wykorzystano laptop z 15 calowym wyświetlaczem LCD o rozdzielczości 1920x1080. Standardową myszką i wbudowaną klawiaturą. Procesorem firmy Intel Core i5 oraz kartą graficzną firmy Nvidia. Szczegółową specyfikację urządzenia przedstawiono w Tabeli 1.

Tabela 1: Parametry systemu użytego w badaniach

Pamięć RAM	8 GiB
Procesor	Intel Core i5-6300HQ @ 2,3GHz x 4
Grafika	Nvidia GeForce GTX 960M
Pojemność dysku	1,0 TB
System operacyjny	Ubuntu 20.04.2 LTS

Do pracy z biblioteką zostanie wykorzystane środowisko Docker. Rozwiązanie to zapewnia łatwe urucho-

mienie TensorFlow dla CPU i GPU. W badaniach zostaną wykorzystane obrazy biblioteki Tensorflow w wersji 2.5.0.

6.2. Zbiór danych treningowych

W eksperymencie wykorzystano zestaw danych, w którym zebrano dane o odręcznie pisanych znakach języka polskiego. Dane zawierają:

- cyfry: 0-9,
- małe i wielkie litery alfabetu łacińskiego: A-z
- małe litery alfabetu polskiego: ą, ć, ę, ł, ń, ó, ś, ź, ż,
- wielkie litery alfabetu polskiego: Ą, Ć, Ę, Ł, Ń, Ó, Ź, Ż,
- znaki specjalne: +, -, :, ;, $, !, @.

Zgodnie z dokumentacją dla każdego znaku zebrano co najmniej 6000 obrazów o wielkości 32px (szerokość) na 32px (wysokość) [18]. W pracy wykorzystano małe litery alfabetu polskiego i łacińskiego. Zebrane dane podzielono na zestaw uczący i testowy, gdzie zestaw testowy stanowi 10% całości.

6.3. Model sieci neuronowej

W badaniach wykorzystano model sieci neuronowej, który oparty został o architekturę sieci CNN (ang. *convolutional neural network*) wykorzystanej w artykule [19]. Danymi wejściowymi jest macierz binarna o wymiarach 32x32. Dane wejściowe są propagowane przez dwie warstwy konwolucyjne, które mają kolejno 32 i 64 filtry o rozmiarze 3x3 i rozstępie (ang. *stride*) 1. Otrzymane dane zostają poddane spłaszczeniu i są przepuszczane przez w pełni połączoną warstwę sieci o 256 neuronach. Warstwa wyjściowa jest warstwą sieci w pełni połączonej z funkcją aktywacji Softmax. Dane otrzymane na wyjściu pozwolą na rozpoznawanie znaków ze zbioru danych treningowych. W sieci użyto funkcji straty *categorical cross-entropy*. Schemat powstałej sieci przedstawiono na Rysunku 1.

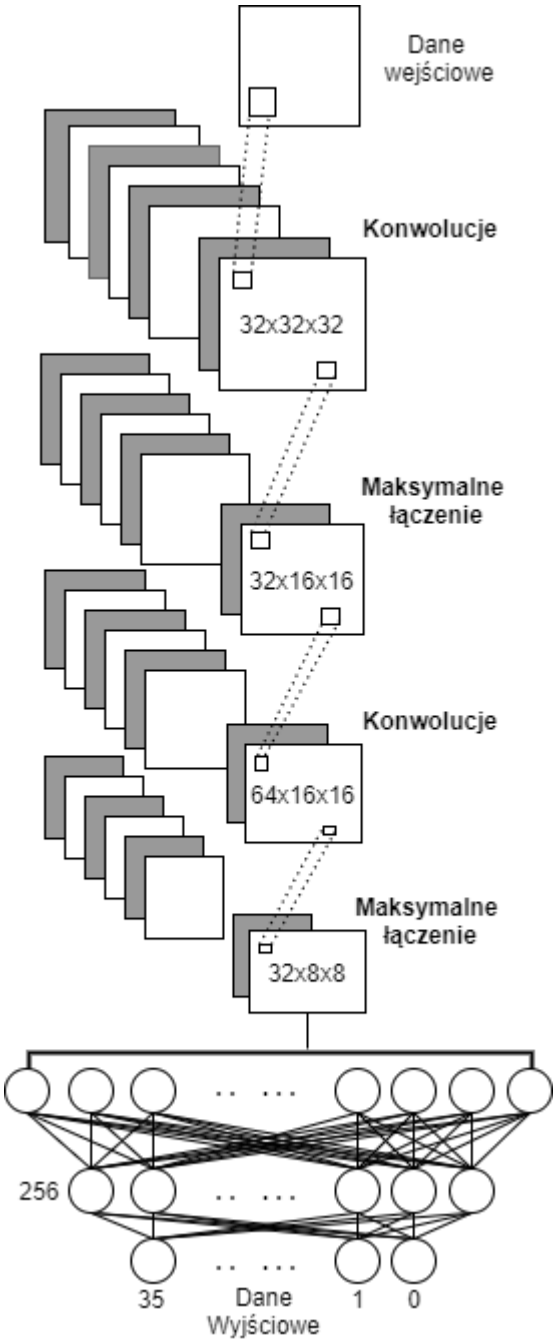
6.4. Metoda badawcza

Dla każdego algorytmu optymalizacji przeprowadzono badanie na CPU i GPU. W celu określenia, która konfiguracja najefektywniej wykonuje zadanie, przeprowadzono analizę wielokryterialną metodą sumy ważonej [20]. Polega ona na łączeniu kolejnych kryteriów w funkcję celu przy pomocą wzoru:

$$F(x) = \sum_{i=1}^{m} w_i f_i(x) \qquad (1)$$

gdzie $w_i \in [0, 1]$ jest wagą kryterium, x wektorem rozwiązań. Metoda ta wymaga wcześniejszej normalizacji danych. Normalizacja została przeprowadzona za pomocą metody unitaryzacji zerowej [21]. Aby zastosować tę metodę należy wyznaczyć kryteria oceny oraz określić, które z nich wpływają pozytywnie na ocenę końcową, a które negatywnie. Wyznaczono następujące kryteria oceny:

1) precyzja (ang. *accuracy*) - oblicza jak często predykcje pasują do etykiet danych szkoleniowych,
2) strata (ang. *loss*) - określa jak bardzo przewidywane wartości odbiegają od rzeczywistych wartości w danych szkoleniowych,
3) precyzja walidacji - precyzja dla danych testowych,
4) strata walidacji - strata dla danych testowych,
5) czas - łączny czas uczenia.



Rysunek 1: Schemat sieci neuronowej na podstawie [19].

Wśród kryteriów oceny należało wskazać stymulatory i destymulatory. Stymulatory wraz ze wzrostem wartości wpływają pozytywnie na ocenę końcową. Wśród badanych kryteriów te cechy posiada: precyzja i precyzja walidacji. Stymulatory wyznaczano za pomocą wzoru:

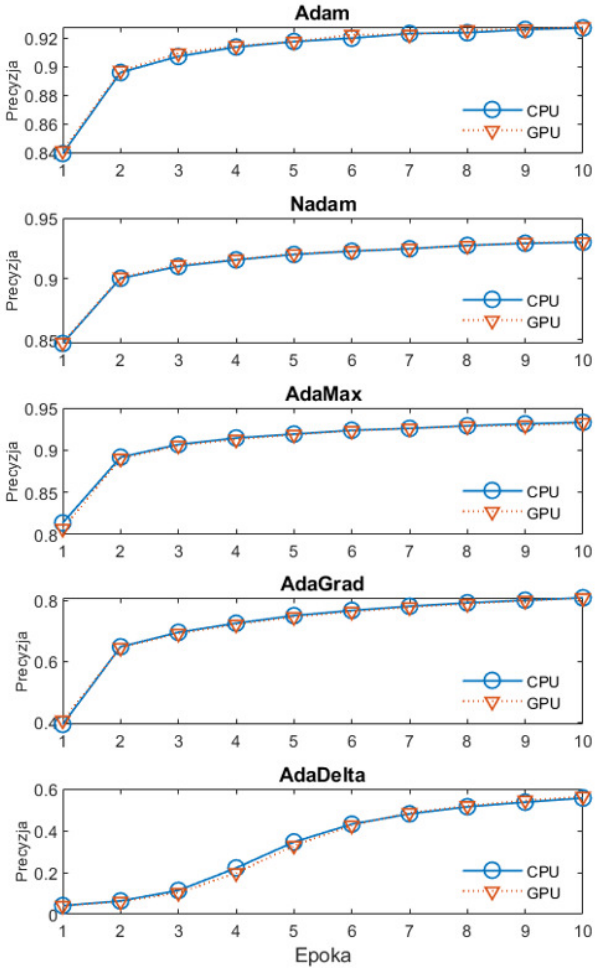
$$Z_{ij} = \frac{x_{ij} - min(x_{ij})}{max(x_{ij}) - min(x_{ij})} \quad (2)$$

gdzie i, j ∈ N, X_{ij} jest wartością badanego kryterium, zaś
$max(X_{ij})$ oraz $min(X_{ij})$ to kolejno wartości maksymalne i minimalne ze zbioru badanego kryterium oceny.

Destymulatory wraz ze wzrostem wartości mają negatywny wpływ na ocenę końcową. Wśród badanych kryteriów te cechy posiada: strata, strata walidacji, czas. Destymulatory wyznaczano za pomocą wzoru:

$$Z_{ij} = \frac{max(x_{ij}) - x_{ij}}{max(x_{ij}) - min(x_{ij})} \quad (3)$$

gdzie *i, j,* X_{ij}, $max(X_{ij})$, $min(X_{ij})$ odpowiadają zmiennym ze wzoru 2. W ostatnim kroku wyznaczono wagi dla poszczególnych kryteriów oceny. Najważniejszymi kryteriami oceny algorytmu optymalizacji jest precyzja oraz strata, które kolejno mają wagi: 0,4; 0,3. Kryteriom precyzja oraz strata walidacji nadano wagę 0,2, gdyż dane dla tych parametrów dobierane są losowo. Na ostatnie kryterium badawcze wpływa w dużym stopniu środowisko badawcze, dlatego też czas ma wagę 0,3.



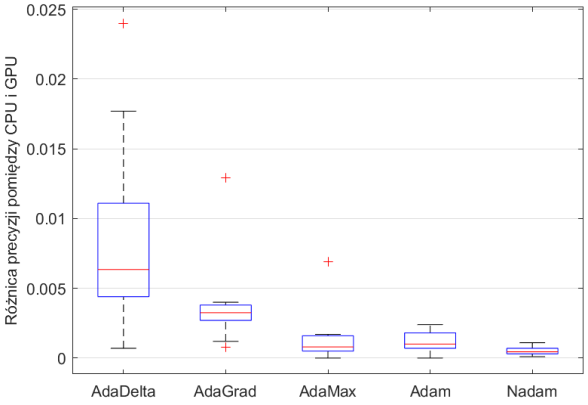
Rysunek 2: Porównanie precyzji treningu algorytmów dla badanych podzespołów.

7. Analiza rezultatów

Badania przeprowadzono według scenariusza badawczego, który składał się z:

- załadowania danych do programu,
- określenia algorytmu optymalizacji,
- przeprowadzenia szkolenia,
- zapisu otrzymanych rezultatów.

Na Rysunku 2 przedstawiono wykres obrazujący precyzję badanych algorytmów dla kolejnych epok (ang. *epochs*) programu. Dane przedstawione na Rysunku 2 pokazują, że dla każdego badanego algorytmu wartość precyzji na kolejnych epokach jest bardzo podobna dla mikroprocesora i karty graficznej. Różnice jest trudno dostrzec ludzkim okiem, dlatego też utworzono wykres pudełkowy pokazujący, który z analizowanych algorytmów optymalizacji wykazuje największe różnice w wartościach precyzji dla CPU i GPU (Rysunek 3).



Rysunek 3: Różnice precyzji algorytmów dla CPU i GPU.

Z Rysunku 3 można odczytać, że największe różnice wykazał algorytm AdaDelta, różnice otrzymane dla tego algorytmu wahały się pomiędzy 0,0007 oraz 0,0240. W porównaniu do algorytmów Adam, Nadam oraz AdaMax różnica ta jest stosunkowo duża. Wśród badanych optymalizatorów najmniejsze różnice wykazał algorytm Nadam.

Tabela 1: Rezultaty eksperymentu dla GPU

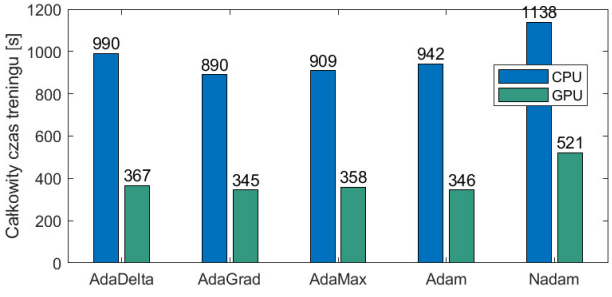
	precyzja	strata	precyzja walidacji	strata walidacji
AdaDelta	0,5635	1,4788	0,6547	1,2370
AdaGrad	0,8089	0,6353	0,8556	0,4923
Adam	0,9278	0,2285	0,9299	0,2308
Nadam	0,9302	0,2180	0,9331	0,2210
AdaMax	0,9324	0,2187	0,9374	0,2055

Tabela 2: Rezultaty eksperymentu dla CPU

	precyzja	strata	precyzja walidacji	strata walidacji
AdaDelta	0,5561	1,4929	0,6477	1,2632
AdaGrad	0,8097	0,6336	0,8542	0,4902
Adam	0,9271	0,2310	0,9322	0,2216
Nadam	0,9301	0,2213	0,9347	0,2204
AdaMax	0,9329	0,2166	0,9364	0,2059

W Tabelach 1 i 2 przedstawiono rezultaty eksperymentu dla poszczególnych podzespołów komputera w ostatniej iteracji programu. Z danych zamieszczonych w tabelach 1 i 2 wynika, że algorytm AdaMax osiągnął najlepsze rezultaty pod względem każdego z badanych kryteriów. Precyzja tego algorytmu osiągnęła wartość 0,9329 dla GPU, przy stosunkowo niskich stratach 0,2166. Najgorsze wyniki uzyskał algorytm AdaDelta. Precyzja osiągnęła wartość 0,5635 przy czym straty algorytmu osiągnęły wysokie wartości w porównaniu z pozostałymi algorytmami. Dla każdego algorytmu wartości precyzji walidacji pokrywają się z wartościami precyzji szkolenia.

W celu porównania czasów treningu poszczególnych algorytmów sporządzono wykres zależności algorytmu od jego całkowitego czasu treningu (Rysunek 4).



Rysunek 4: Czas treningu algorytmów optymalizacji.

Z Rysunku 4 wynika, że trening przeprowadzony za pomocą GPU jest blisko 3 razy szybszy dla każdego z algorytmów. Obliczenia wykazały ponad 60% spadek czasu treningu dla algorytmów AdaDelta, AdaGrad, Adam i AdaMax. Algorytm Nadam osiągnął 54% spadek czasu treningu, przy czym okazał się najwolniejszym ze wszystkich algorytmów.

W Tabeli 3 i 4 przedstawiono dane po przeprowadzeniu normalizacji danych przedstawionych w Tabelach 1 i 2.

Tabela 3: Znormalizowane rezultaty badań dla CPU

Algorytm	precyzja	strata	precyzja walidacji	strata walidacji	czas
AdaDelta	0,0000	0,0000	0,0000	0,0000	0,5968
AdaGrad	0,6730	0,6733	0,7153	0,7311	1,0000
Adam	0,9846	0,9887	0,9855	0,9852	0,7903
Nadam	0,9926	0,9963	0,9941	0,9863	0,0000
AdaMax	1,0000	1,0000	1,0000	1,0000	0,9234

Tabela 4: Znormalizowane rezultaty badań dla GPU

Algorytm	precyzja	strata	precyzja walidacji	strata walidacji	czas
AdaDelta	0,0000	0,0000	0,0000	0,0000	0,8750
AdaGrad	0,6652	0,6690	0,7106	0,7220	1,0000
Adam	0,9875	0,9917	0,9735	0,9755	0,9943
Nadam	0,9940	1,0000	0,9848	0,9850	0,0000
AdaMax	1,0000	0,9994	1,0000	1,0000	0,9261

Wyniki otrzymane w Tabelach 3 i 4 zostały wykorzystane do przeprowadzenia analizy wielokryterialnej metodą sumy ważonej. Wyniki przeprowadzonych obliczeń przedstawiono w Tabeli 5.

Tabela 5: Suma ważona badanych algorytmów optymalizacji

	CPU	GPU	
Algorytm	Suma ważona		Średnia
AdaDelta	0,1279	0,1875	0,1577
AdaGrad	0,7575	0,7524	0,7550
Adam	0,9441	0,9861	0,9651
Nadam	0,7800	0,7797	0,7799
AdaMax	0,9836	0,9841	0,9839

Według danych przedstawionych w Tabeli 5 najwydajniejszym algorytmem optymalizacji dla GPU jest algorytm Adam, zaś dla CPU AdaMax, jednak różnice pomiędzy tymi wartościami są bardzo małe. Wyliczone średnie sum ważonych, badanych podzespołów komputera, wskazują na to, że w przypadku korzystania z obu podzespołów najlepiej sprawdzi się algorytm AdaMax.

8. Podsumowanie i wnioski

W artykule przeprowadzono badania wydajności biblioteki Tensorflow przy pomocy metody sumy ważonej, która pozwoliła na wybór najlepszego algorytmu optymalizacji dla mikroprocesora i karty graficznej. Po obliczeniu średniej z sumy ważonej dla poszczególnych podzespołów (Tabela 5) można uznać hipotezę H1 za nieprawdziwą. Najbardziej wydajnym algorytmem optymalizacji jest AdaMax.

Na podstawie analizy danych przedstawionych na Rysunku 2 potwierdzono prawdziwość hipotezy H2. Czas treningu na GPU dla większości algorytmów był o blisko 60% krótszy niż w przypadku CPU. Wykorzystywany podzespół komputera nie ma dużego wpływu na precyzję w procesie uczenia. Różnice w wartościach parametrów otrzymanych podczas treningu były bardzo małe.

Wnioski wynikające z przeprowadzonego badania mogą być wykorzystane w procesie wyboru algorytmu optymalizacji w procesach tworzenia systemów uczenia maszynowego do rozpoznawania znaków pisma odręcznego.

Literatura

[1] J. McCarthy, From here to human-level AI, Artificial Intelligence 171 (2007) 1174–1182, https://doi.org/10.1016/j.artint.2007.10.009.

[2] T. Okuda, S. Shoda, AI-based chatbot service for financial industry, Fujitsu Scientific and Technical Journal 54 (2018) 4–8.

[3] S. Green, J. Heer, C. D. Manning, Natural language translation at the intersection of AI and HCI, Communications of the ACM 58 (2015) 46–53, https://doi.org/10.1145/2767151.

[4] K. Chakraborty, A. Talele, S. Upadhya, Voice recognition using MFCC algorithm, International Journal

of Innovative Research in Advanced Engineering (IJIRAE) 1 (2014) 158–161.

[5] H. Fujiyoshi, T. Hirakawa, T. Yamashita, Deep learning-based image recognition for autonomous driving, IATSS research 43 (2019) 244–252, https://doi.org/10.1016/j.iatssr.2019.11.008.

[6] A. Abraham, F. Pedregosa, M. Eickenberg, P. Gervais, A. Mueller, J. Kossaifi, A. Gramfort, B. Thirion, G. Varoquaux, Machine learning for neuroimaging with scikit-learn, Frontiers in neuroinformatics 8 (2014) 1–14, https://doi.org/10.3389/fninf.2014.00014.

[7] J. Moolayil, J. Moolayil, S. John, Learn Keras for Deep Neural Networks, Birmingham: Apress, 2019.

[8] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer et al., Theano: A Python framework for fast computation of mathematical expressions, Computing Research Repository (2016) 1–19.

[9] G. Zaccone, M. R. Karim, Deep learning with TensorFlow: Explore neural networks and build intelligent systems with python, Packt Publishing Ltd, 2018.

[10] S. Bahrampour, N. Ramakrishnan, L. Schott, M. Shah, Comparative study of deep learning software frameworks, Computing Research Repository (2015).

[11] S. Raschka, Python. Uczenie maszynowe, Packt Publishing Ltd, 2017.

[12] Firmy wykorzystujące bibliotekę Tensorflow, https://www.tensorflow.org/about/case-studies, [26.06.2021].

[13] Opis architektury biblioteki Tensorflow, https://developers.googleblog.com/2017/09/introducing-tensorflow-datasets.html, [26.06.2021].

[14] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, International Conference on Learning Representations (2015) 1–15.

[15] T. Dozat, Incorporating Nesterov Momentum into Adam, International Conference on Learning Representations (2016) 1–4.

[16] A. Lydia, S. Francis, Adagrad–an optimizer for stochastic gradient descent, International Journal of Information and Computing Science 6 (2019) 566–568.

[17] M. D. Zeiler, Adadelta: an adaptive learning rate method, Computing Research Repository (2012).

[18] M. Tokovarov, M. Kaczorowska, M. Miłosz, Development of Extensive Polish Handwritten Characters Database for Text Recognition Research, Advances in Science and Technology Research Journal 14 (2020) 30–38, https://doi.org/10.12913/22998624/122567.

[19] E. Łukasik, M. Charytanowicz, M. Miłosz, M. Tokovarov, M. Kaczorowska, D. Czerwinski, T. Zientarski, Recognition of handwritten Latin characters with diacritics using CNN, Bulletin of the Polish Academy of Sciences: Technical Sciences 69 (2021) 1–12, http://dx.doi.org/10.24425/bpasts.2020.136210.

[20] E. Zitzler, Evolutionary algorithms for multiobjective optimization: Methods and applications, Ithaca: Shaker. 1999.

[21] K. Kukuła, Metoda unitaryzacji zerowanej na tle wybranych metod normowania cech diagnostycznych, Acta Scientifica Academiae Ostroviensis 4 (1999) 5–31.