Raport projektu #1

z przedmiotu Metody Głębokiego Uczenia

Piotr Podbielski, grupa nr 3

26 marca 2019

Streszczenie

Raport opisuje zastosowanie perceptronu wielowarstwowego do problemu klasyfikacji obrazów ze zbioru MNIST.

1 Wstęp

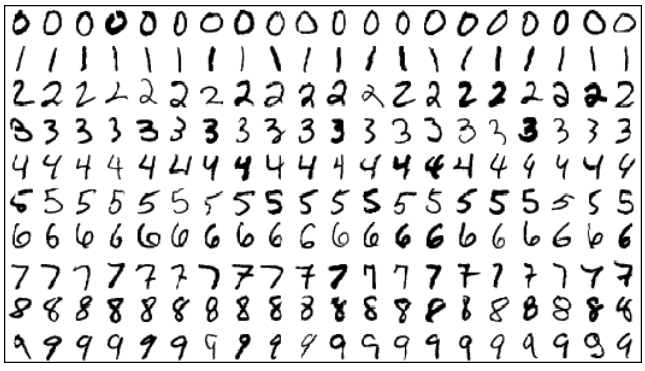
Niniejszy dokument powstał w wyniku pierwszego projektu z przedmiotu *Metody Głębokiego Uczenia* wykładanego na kierunku *Inżynierii i Analizy Danych* w roku akademickim *2018/2019* na wydziale *Matematyki i Nauk Informacyjnych Politechniki Warszawskiej*.

Celem projektu jest zaimplementowanie modelu wielowarstwowego perceptronu, który umożliwi klasyfikację obrazów ze zbioru MNIST. Implementacja musi spełniać następujące warunki:

- udostępnianie możliwości wyboru hiperparametrów,
- wizualizowanie funkcji kosztu na zbiorze treningowym i walidacyjnym podczas trenowania,
- wyświetlanie błędnie sklasyfikowanych przykładów ze zbioru walidacyjnego.

2 Zbiór danych MNIST

Zbiór danych uczących i walidacyjnych dostępny jest na stronie Yann'a Lecun'a. Zbiór treningowy składa się z 60 tyś. przykładów, zaś walidacyjny z 10 tyś. Każdy z przykładów jest obrazem o rozmiarach 20x20 pikseli wraz z klasą, do której należy. Obrazy te przedstawiają cyfry od 0 do 9. Przykładowy pozbiór przykładów widoczny jest na rysunku 1.



Rysunek 1: Przykładowy podzbiór przykładów ze zbioru MNIST.

Wartości poszczególnych pikseli są odwzorowaniem nasycenia w skali szarości. Wartości nasycenia pikseli zawierają się w skali od 0 do 255.

3 Rozwiązania użyte w projekcie

3.1 Przeskalowanie danych wejściowych

Sieci neuronowe osiągają zbieżność szybciej, gdy dane wejściowe są wystandaryzowane. Standaryzacja danych oznacza, iż ich wartość średnia jest równa 0, a wariancja jest równa 1, dzięki czemu większość wartości znajduje się w przedziale od -1 do 1.

Zamiast dokonywania standaryzacji, w implementacji dokonano przeskalowania danych wejściowych. Początkowo wartości nasycenia pikseli były z przedziału od 0 do 255, a po przeskalowaniu są one z przedziału od -1 do 1. Uzyskano to dzięki odjęciu od wartości nasycenia pikseli liczby 128 a następnie podzielenie przez liczbę 128.

3.2 Perceptron wielowarstwowy

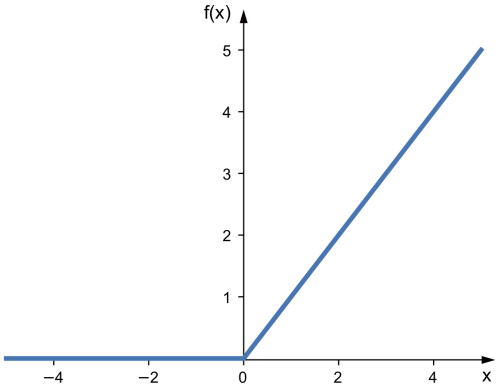
Perceptron wielowarstwowy jest typem sieci neuronowej, w której nie występują pętle zwrotne. Oznacza to, że podczas przepływu informacji *w przód*, w warstwie i wykorzystywane są tylko wartości aktywacji z warstwy $i-1$.

3.3 Funkcje aktywacji

Aby sieć neuronowa mogła aproksymować funkcje, w których zmienna objaśniana jest nieliniową zależnością zmiennych objaśniających, należy wprowadzić do sieci pewien element nieliniowości. Elementem nieliniowości w sieciach neuronowych są funkcje aktywacji. Funkcje te, aby spełniać swoją rolę, muszą być nieliniowe.

3.3.1 ReLU

Nieliniową funkcją aktywacji użytą w modelu jest funkcja *ReLU* (ang. *Rectified Linear Unit*). Wyraża się ona wzorem $f(x) = max(0, x)$. Jej charakterystykę przedstawiono na rysunku 2.



Rysunek 2: Funkcja aktywacji ReLU. Źródło.

Jest to jedna z najprostszych funkcji nieliniowych, a zarazem najczęściej używanych i sprawdzających się w sieciach neuronowych.

3.3.2 Softmax

W celu sklasyfikowania przykładów do ich klas, w warstwie wyjściowej modelu zastosowano funkcję aktywacji *Softmax*. Funkcja ta przyjmuje wektor wartości wejściowych i przekształca go na wektor, którego wartości sumują się do jedności. Oznacza to, że każda z wartości wektora wyjściowego zawiera się w przedziale od 0 do 1. Naturalnie nasuwa się w tym momencie interpretacja tych liczb jako wartości prawdopodobieństw przynależności przykładu do poszczególnych klas. Funkcja softmax wyraża się wzorem 1.

$$f(x, K) = \frac{e^x}{\sum_{k=1}^{K} e^{x_k}} \tag{1}$$

Gdzie x to wektor wejściowy o długości K, a K to liczba klas (w naszym przypadku równa 10, ponieważ przykłady ze zbioru MNIST reprezentują cyfry od 0 do 9).

Schemat działania funkcji *Softmax* przedstawiono na rysunku 3.

Rysunek 3: Funkcja aktywacji Softmax. Źródło.

3.4 Funkcja straty entropii krzyżowej

Aby optymalizować wagi perceptronów używany jest mechanizm propagacji wstecznej gradientów. Funkcją, która jest minimalizowana w celu optymalizacji wag, jest funkcja straty. W modelu zastosowano funkcję straty entropii krzyżowej. W przypadku sieci neuronowych entropia jest miarą zróżnicowania między prawdziwym, a przewidywanym prawdopodobieństwem dla każdej z klas.

Entropię krzyżową dla pojedynczego przykładu można obliczyć korzystając ze wzoru 2.

$$H(y, \hat{y}) = -\sum_{i} y_i \log \hat{y}_i \tag{2}$$

3.5 Metody regularyzacji

Aby sieci neuronowe nie ulegały przetrenowaniu, dodaje się do nich mechanizmy regularyzacji. Przetrenowanie polega na nadmiernym dopasowaniu się wag sieci neuronowej do zbioru treningowego. W momencie przetrenowania, wyniki sieci na zbiorze walidacyjnym, bądź testującym ulegają pogorszeniu.

3.5.1 Regularyzacja L2

Jednym ze sposobów regularyzacji jest spowodowanie, aby wagi neuronów występujących w sieci były relatywnie małe. Przez relatywnie małe rozumie się to, że chcemy zapobiec sytuacji, w której ciągle one rosną. Można to osiągnąć dodając do funkcji kosztu wyrażenie, które odpowiedzialne jest za ograniczenie wartości wag. Wyrażenie to wyraża się wzorem 3.

$$J_{regularized} = J + \frac{1}{m}\frac{\lambda}{2}\sum_{l}\sum_{k}\sum_{j}(W_{k,j}^{[l]})^2 \tag{3}$$

Gdzie J to funkcja kosztu bez regularyzacji L2, m to liczba przykładów w partii uczącej (ang. *training batch*), λ to hiperparametr określający jak bardzo karać model za dużą wartość wag, $W_{k,j}^{[l]}$ to wartość wagi między k-tym neuronem warstwy $l-1$ a j-tym neuronem warstwy l.

Powyższa modyfikacja funkcji kosztu ma swoje odzwierciedlenie także podczas aktualizacji wag w procesie trenowania. Gradient dla konkretnej macierzy wag trzeba zmodyfikowac według wzoru 4.

$$W_{d_{regularized}} = W_d + \frac{\lambda}{m}W \tag{4}$$

Wzór 4 intepretować należy w taki sposób, że do gradientu macierzy wag należy dodać wyrażenie odpowiedzialne za powiększenie gradientu o część dotychczasowych wartości wag. Dzięki temu, podczas kroku optymalizacji wag odjęta zostanie większa wartość, niż było by to bez regularyzacji L2. Powoduje to, że wartości wag sieci neuronowej są wyciszane.

3.5.2 Dropout

Dropout jest kolejnym z mechanizmów regularyzacji. Polega on na wyłączaniu podczas trenowania losowych neuronów z zadanym prawdopodobieństwem p. Powoduje to, że podczas trenowania sieć jest w pewien sposób ograniczona, ma mniejszą pojemność, niż wynikało by z jej bazowej architektury. Sieć o mniejszej pojemności (ang. *capacity*) pozwala na przybliżanie funkcji mniej skomplikowanych. Dzięki

temu uzyskujemy efekt ograniczenia przetrenowania sieci neuronowej. Innym efektem wyłączania losowych neuronów jest to, że na predykcję rzadziej wpływ ma wartość konkretnego neuronu. Dzięki mechanizmowi dropout różne neurony muszą nauczyć się wykrywać różne końcowe cechy. Na przykładzie zbioru MNIST oznacza to, że model nie będzie podejmował decyzji o danej klasie na przykład na podstawie wartości nasycenia jednego piksela, gdyż podczas trenowania losowe piksele zostały deaktywowane, a model nadal miał za zadanie nauczyć się poprawnie klasyfikować obrazki.

3.6 Optymalizatory

W celu optymalizacji wag sieci neuronowych używa się optymalizatorów. Poniżej przedstawiono dwa z nich zaimplementowane w projekcie.

3.6.1 Stochastic Gradient Descent

Jest to podstawowa wersja optymalizatora opartego o metodę stochastycznego spadku. Wzór na aktualizacje macierzy wag przedstawiony został we wzorze 5.

$$W = W - \alpha W_d \tag{5}$$

Gdzie α jest hiperparametrem oznaczającym szybkość uczenia, który musi zostać dobrany ręcznie do danych treningowych modelu.

3.6.2 ADAptive Momentum

Brak zmienności hiperparametru α w SGD powoduje różne problemy podczas optymalizacji. Między innymi są to: wpadanie w minima lokalne i podatność na nie osiągnięcie minimum ze względu na jego przeskakiwanie.

Celem rozwiązania tego problemu wprowadzono optymalizatory z pędem (ang. *Momentum*). Optymalizator ADAM (ang. *Adaptive Momentum*) dzięki momentom 1. oraz 2. rzędu aktualizując macierze wag sieci neuronowej korzysta z historii wartości tych macierzy. Efektem tego jest możliwość wyciszania albo wzmacniania poszczególnych gradientów. Wzór na wyliczenie macierzy gradientów przedstawiony został we wzorze 6.

$$W_d = \frac{v_t}{\sqrt{s_t + \epsilon}} * g_t \tag{6}$$

Gdzie v_t to średnia wykładnicza gradientów wyrażona wzorem 8, a s_t to średnia wykładnicza kwadratów gradientów wyrażona wzorem **??**.

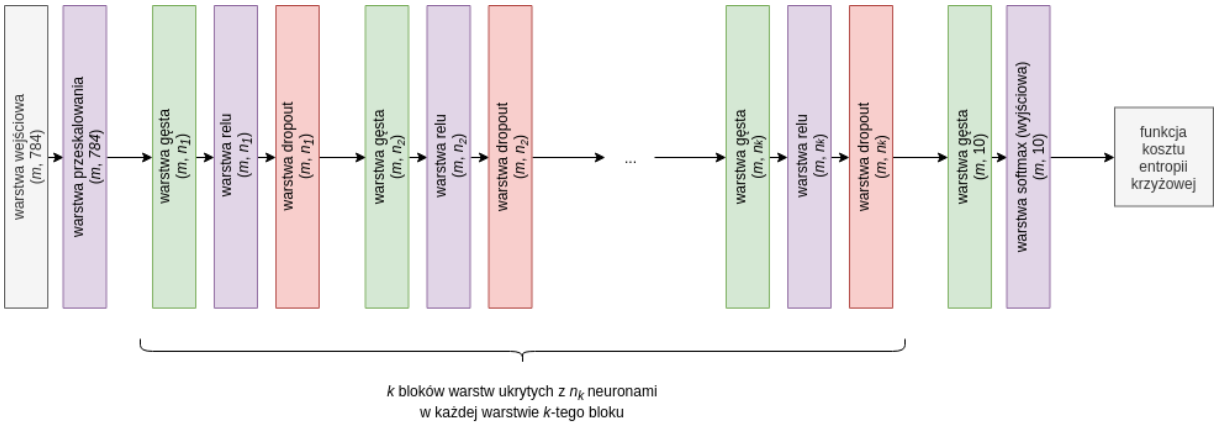
$$v_t = \beta_1 * v_{t-1} - (1 - \beta_1) * g_t \tag{7}$$

$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2 \tag{8}$$

Gdzie β_1 i β_2 to hiperparametry średniej wykładniczej a g_t to gradienty wag.

4 Architektura modelu

Architektura modelu została przedstawiona na rysunku 4.



Rysunek 4: Architektura modelu perceptronu wielowarstwowego.

Gdzie m to liczba przykładów podczas jednego przebiegu przez sieć, k to liczba bloków rozumianych jako warstwy ukryte, n to wektor z liczbą neuronów poszczególnych bloków.

5 Eksperymenty

Niektóre z rozwiązań użytych w projekcie poddano eksperymentom w celu zbadania ich wpływu na proces trenowania sieci neuronowej. W każdym z eksperymentów przedstawiono konfigurację porównywanych modeli wraz z wynikami, które one osiągają.

5.1 Wpływ użytego optymalizatora na czas treningu sieci

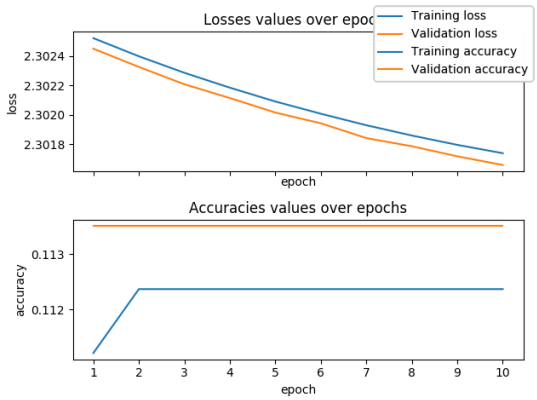
Właściwość	Wartość
Optymalizator	**SGD**
Learning rate	0.001
Weight decay lambda	0.0
Prawdopodobieństwo przepuszczenia w dropout	1.0
Liczba bloków warstw ukrytych	5
Liczba neuronów w poszczególnych blokach	512, 256, 128, 64, 32
Liczba epok	10
Liczba przykładów podczas jednego przebiegu	128

Tablica 1: Opis modelu trenowanego podczas eksperymentu nr 1 z włączonym optymalizatorem SGD.

Właściwość	Wartość
Optymalizator	**ADAptive Momentum**
Learning rate	0.001
Weight decay lambda	0.0
Prawdopodobieństwo przepuszczenia w dropout	1.0
Liczba bloków warstw ukrytych	5
Liczba neuronów w poszczególnych blokach	512, 256, 128, 64, 32
Liczba epok	10
Liczba przykładów podczas jednego przebiegu	128

Tablica 2: Opis modelu trenowanego podczas eksperymentu nr 1 z włączonym optymalizatorem ADAM.

Podczas zastosowania optymalizatora *SGD* trenowanie nie było w stanie w ciągu 10 epok znacząco poprawić wyników sieci. Trenowanie optymalizatorem *ADAM* przyniosło oczekiwane wyniki w postaci modelu wytrenowanego ze skutecznością na poziomie 97.5%. Zestawienie wykresów wygenerowanych podczas trenowania obu sieci ukazane jest poniżej.



(a) Wykres wygenerowany podczas trenowania sieci neuronowej optymalizatorem SGD.

(b) Wykres wygenerowany podczas trenowania sieci neuronowej optymalizatorem ADAM.

Rysunek 5: Zestawienie wykresów wygenerowanych podczas trenowania obu sieci.

5.2 Wpływ pojemności modelu na wyniki sieci

Właściwość	Wartość
Optymalizator	ADAptive Momentum
Learning rate	0.001
Weight decay lambda	0.0
Prawdopodobieństwo przepuszczenia w dropout	1.0
Liczba bloków warstw ukrytych	**5**
Liczba neuronów w poszczególnych blokach	**512, 256, 128, 64, 32**
Liczba epok	10
Liczba przykładów podczas jednego przebiegu	128

Tablica 3: Opis modelu trenowanego podczas eksperymentu nr 2 z 5 blokami warstw ukrytych.

Właściwość	Wartość
Optymalizator	ADAptive Momentum
Learning rate	0.001
Weight decay lambda	0.0
Prawdopodobieństwo przepuszczenia w dropout	1.0
Liczba bloków warstw ukrytych	**2**
Liczba neuronów w poszczególnych blokach	**32, 32**
Liczba epok	10
Liczba przykładów podczas jednego przebiegu	128

Tablica 4: Opis modelu trenowanego podczas eksperymentu nr 2 z 2 blokami warstw ukrytych.

Podczas trenowania sieci z 5 warstwami, w której w sumie było 992 neuronów ukrytych sieć osiągnęła wyniki na poziomie 97.5%, zaś podczas trenowania sieci z 2 warstwami, w której w sumie było 64 neuronów ukrytych sieć osiągnęła wyniki na poziomie 94.9%. Oznacza to, że w drugim przypadku sieć miała mniejszą pojemność i nie mogła dopasować się w zarówno do danych treningowych jak i uczących. Zestawienie wykresów wygenerowanych podczas trenowania obu sieci ukazane jest poniżej.

(a) Wykres wygenerowany podczas trenowania sieci neuronowej z 5 blokami warstw ukrytych.

(b) Wykres wygenerowany podczas trenowania sieci neuronowej z 2 blokami warstw ukrytych.

Rysunek 6: Zestawienie wykresów wygenerowanych podczas trenowania obu sieci.

5.3 Wpływ użycia regularyzacji L2 na wyniki sieci

Właściwość	Wartość
Optymalizator	ADAptive Momentum
Learning rate	0.001
Weight decay lambda	**0.0**
Prawdopodobieństwo przepuszczenia w dropout	1.0
Liczba bloków warstw ukrytych	3
Liczba neuronów w poszczególnych blokach	512, 512, 512
Liczba epok	10
Liczba przykładów podczas jednego przebiegu	128

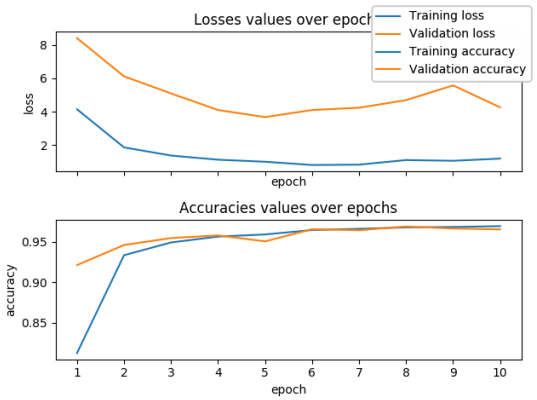
Tablica 5: Opis modelu trenowanego podczas eksperymentu nr 3 z wyłączoną regularyzacją L2.

Właściwość	Wartość
Optymalizator	ADAptive Momentum
Learning rate	0.001
Weight decay lambda	**0.2**
Prawdopodobieństwo przepuszczenia w dropout	1.0
Liczba bloków warstw ukrytych	3
Liczba neuronów w poszczególnych blokach	512, 512, 512
Liczba epok	10
Liczba przykładów podczas jednego przebiegu	128

Tablica 6: Opis modelu trenowanego podczas eksperymentu nr 3 z hiperparametrem λ regularyzacji L2 ustawionym na 0.2.

Podczas trenowania sieci z wyłączoną regularyzacją L2 sieć osiągnęła wyniki: 96.9% na zbiorze treningowym oraz 97.1% na zbiorze walidacyjnym, zaś norma druga po wszystkich wartościach macierzy wag wyniosła około 53780.43. Podczas trenowania sieci z włączoną regularyzacją L2 sieć osiągnęła wyniki: 98.7% na zbiorze treningowym oraz 96.5% na zbiorze walidacyjnym, zaś norma druga po wszystkich wartościach macierzy wag wyniosła około 1359.63. Oznacza to, że przy włączonej regularyzacji L2 model zdecydowanie starał się zachować wagi mniejsze (co do wartości) oraz zapobiegał przetrenowaniu. Zestawienie wykresów wygenerowanych podczas trenowania obu sieci ukazane jest poniżej.

(a) Wykres wygenerowany podczas trenowania sieci neuronowej z wyłączoną regularyzacją L2.



(b) Wykres wygenerowany podczas trenowania sieci neuronowej z hiperparametrem λ regularyzacji L2 ustawionym na 0.2.

Rysunek 7: Zestawienie wykresów wygenerowanych podczas trenowania obu sieci.

6 Implementacja

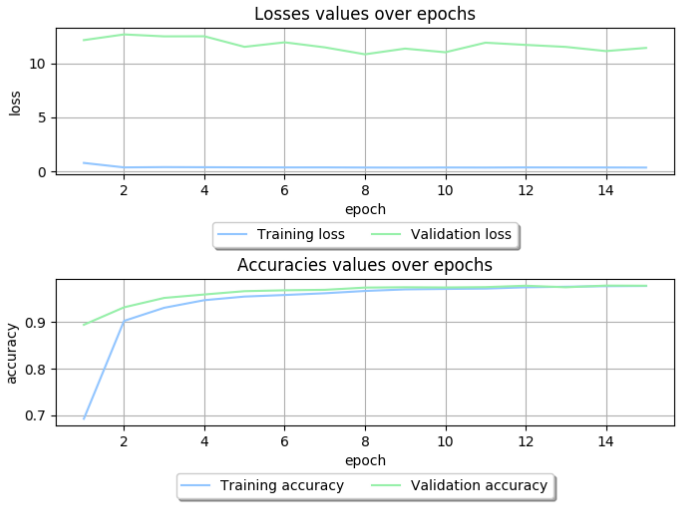
Nieodłączonym elementem raportu jest kod umieszczony w repozytorium VictorAtPL/MNIST_FCDN_NumPy na platformie *GitHub*. Model został zaimplementowany w języku *Python*. W ramach prac powstał prosty framework, który pozwala na modułową budowę modelu.

7 Najlepszy wynik modelu trenowanego lokalnie

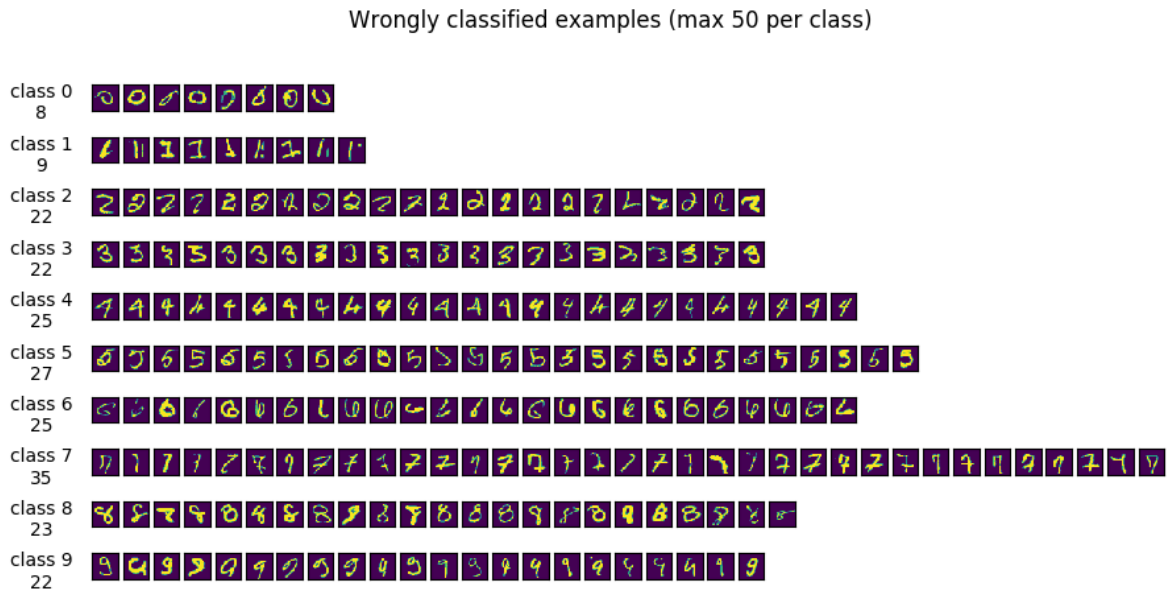
Wynik skuteczności 97.8% na zbiorze walidacyjnym uzyskano na modelu, którego konfiguracja została przedstawiona w tabeli 7, zaś wykresy wygenerowane podczas trenowania zostały przedstawione na rysunkach 8, 9 oraz 10. Model ten został wytrenowany lokalnie na karcie graficznej *NVIDIA 1030*.

Właściwość	Wartość
Optymalizator	ADAptive Momentum
Learning rate	0.001
Weight decay lambda	0.2
Prawdopodobieństwo przepuszczenia w dropout	0.6
Liczba bloków warstw ukrytych	2
Liczba neuronów w poszczególnych blokach	512, 512
Liczba epok	15
Liczba przykładów podczas jednego przebiegu	1024

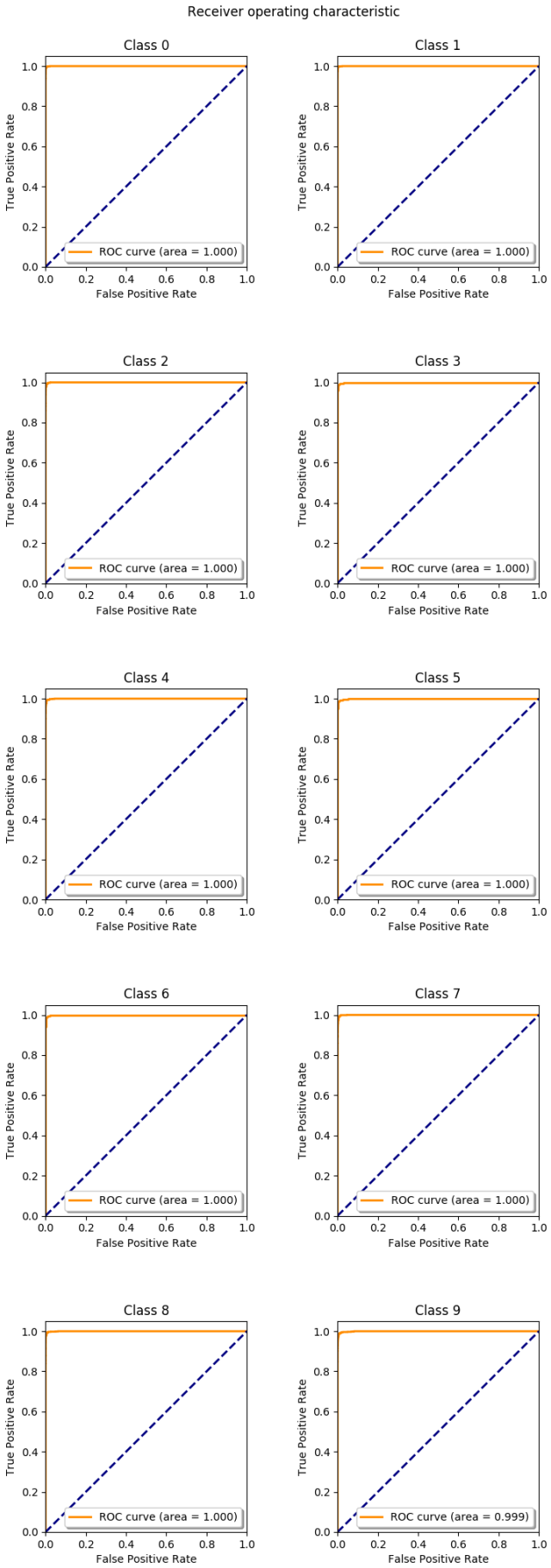
Tablica 7: Opis najlepszego lokalnie wytrenowanego modelu.



Rysunek 8: Zestawienie wykresów funkcji kosztu oraz skuteczności klasyfikatora wygenerowanych podczas lokalnego trenowania najlepszej sieci.



Rysunek 9: Wykres niepoprawnie sklasyfikowanych przykładów przez lokalnie wytrenowany model.



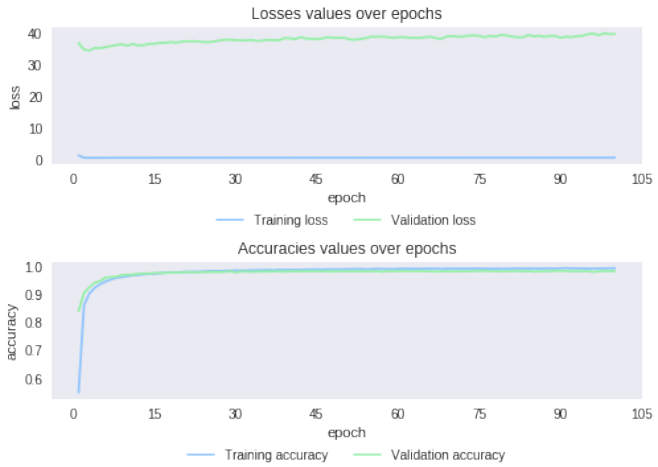
Rysunek 10: Zestawienie wykresów krzywych ROC dla każdej z klas wygenerowanych na podstawie predykcji modelu na zbiorze walidacyjnym.

8 Najlepszy wynik modelu trenowanego na Google Colab

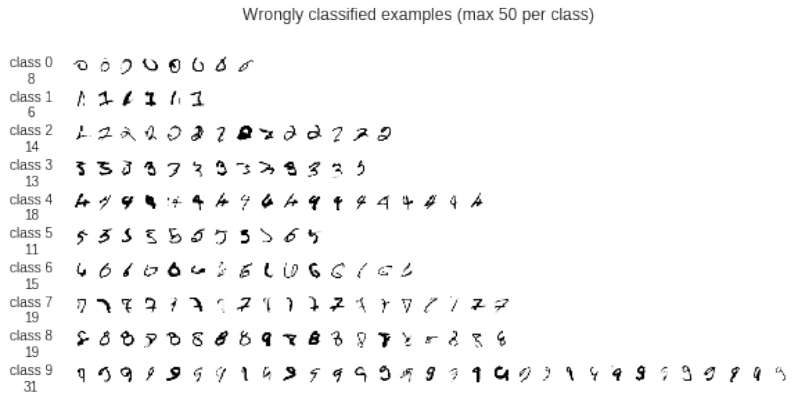
Wynik skuteczności 98.46% (zaś po jednej z epok 98.58%) na zbiorze walidacyjnym uzyskano na modelu, którego konfiguracja została przedstawiona w tabeli 8, zaś wykresy wygenerowane podczas trenowania zostały przedstawione na rysunkach 11 oraz 12. Wykres krzywych ROC nie zmienił się. Model ten został wytrenowany z użyciem platformy *Google Colab*, która udostępnia karty graficzne *Tesla K80*.

Właściwość	Wartość
Optymalizator	ADAptive Momentum
Learning rate	0.001
Weight decay lambda	0.2
Prawdopodobieństwo przepuszczenia w dropout	0.6
Liczba bloków warstw ukrytych	2
Liczba neuronów w poszczególnych blokach	512, 512
Liczba epok	1000
Liczba przykładów podczas jednego przebiegu	2048

Tablica 8: Opis najlepszego trenowanego modelu na platformie Google Colab.



Rysunek 11: Zestawienie wykresów funkcji kosztu oraz skuteczności klasyfikatora wygenerowanych podczas trenowania najlepszej sieci na platformie Google Colab.



Rysunek 12: Wykres niepoprawnie sklasyfikowanych przykładów przez model wytrenowany na platformie Google Colab.

9 Wnioski

Zaimplementowany model w języku Python osiąga wyniki na poziomie 97%–98.5%, co jest bliskie wynikom oficjalnym dla perceptronu wielowarstwowego. Jednakże podział danych jedynie na zbiór treningowy i walidacyjny powoduje, że najprawdopodobniej model został nadmiernie dopasowany do zbioru walidacyjnego poprzez ustawienia hiperparametrów i liczbę epok przez który był trenowany. Aby zwiększyć zdolność modelu do generalizacji należy zwiększyć hiperparametry związane z metodami regularyzacji (L2, dropout). Aby uzyskać wyższą skuteczność należy użyć augmentacji danych lub sieci konwolucyjnych.