Rafał Skrzypiec - Analiza
Kilka niezależnych paragrafów

1 Walidacja krzyżowa

Algorytm wykorzystywany podczas nauki modelu ma za zadnianie znalezienie takich parametrów, które sprawiają, że model odwzorowuje dane wykorzystane do nauki w sposób jak najlepszy z możliwych. Jeśli do walidacji modelu wykorzystamy inną, niezależną próbkę danych pochodzącą z tego samego zbioru co podzbiór uczący, zazwyczaj okaże się, że model nie działa aż tak dobrze jak przy użyciu zbioru uczącego. Rozmiar tej różnicy zwiększa się, szczególnie wtedy gdy wielkość zbioru treningowego jest niewielka, lub gdy liczba parametrów modelu jest bardzo duża. Walidacja krzyżowa to metoda statystyczna, która ma za zadanie zminimalizować tę różnicę przez co pomaga ocenić i zwiększyć trafność przewidywań modelu predykcyjnego.

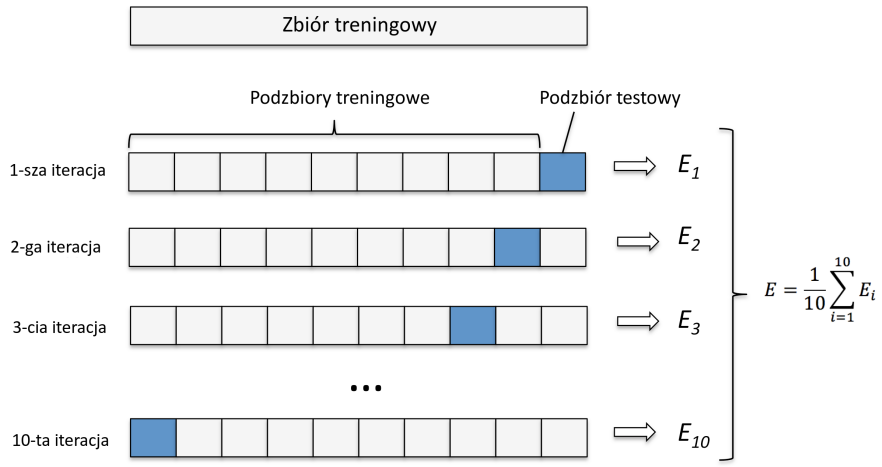
W najprostszym przykładzie walidacji krzyżowej zbiór danych dzieli się na dwa podzbiory: uczący i walidacyjny. Podczas gdy zbiór uczący służy do nauki modelu, zbiór walidacyjny wykorzystuje się aby mierzyć błąd modelu na nieznanym zbiorze danych.

W algorytmie k-krotnej walidacji krzyżowej zbiór danych jest losowo dzielony na k równych wielkością podzbiorów. Jeden z k podzbiorów jest przeznaczany na zbiór walidacyjny, pozostałe $k-1$ podzbiorów służą jako dane teningowe. Powyżej opisana procedura jest powtarzana k razy, a każdy k podzbiorów dokładnie raz zostaje wykorzystany jako zbiór testowy. Następnie k wyników modelu jest uśrednianych dając w rezultacie jeden wynik. Rysunek 1 przedstawia sposób działania 10-krotnej walidacji krzyżowej.

Cytując [Page 184, An Introduction to Statistical Learning, 2013.], "(...) istnieje pewien kompromis między obciążeniem a wariancją, związany z wyborem parametru k w k-krotnej walidacji krzyżowej. Zazwyczaj stosuje się wartości z przedziału od 5 do 10, ponieważ pokazano empirycznie, że w takim wypadku otrzymujemy przewidywania, które nie cierpią nadmiernie ani z powodu dużego obciążenia ani dużej wariancji." Podczas treningu modelu wybrano $k = 6$, wykorzystując fakt, że liczba próbek w zbiorze danych jest całkowicie podzielna przez tę liczbę co zapewnia równy rozmiar wszystkich zbiorów treningowych i walidacyjnych.

2 Wczesne zatrzymanie

Algorytmy uczenia maszynowego dopasowują parametry modelu na podstawie danych treningowych o skończonym rozmiarze. Podczas procesu szkolenia model jest oceniany na podstawie tego, jak dobrze przewiduje obserwacje zawarte w tym zbiorze. Jednak celem uczenia maszynowego jest stworzenie modelu, który ma zdolność do przewidywania uprzednio niewidzianych obserwacji. Nadmierne dopasowanie to zjawisko pojawiające się wtedy gdy model za bardzo dopasowuje



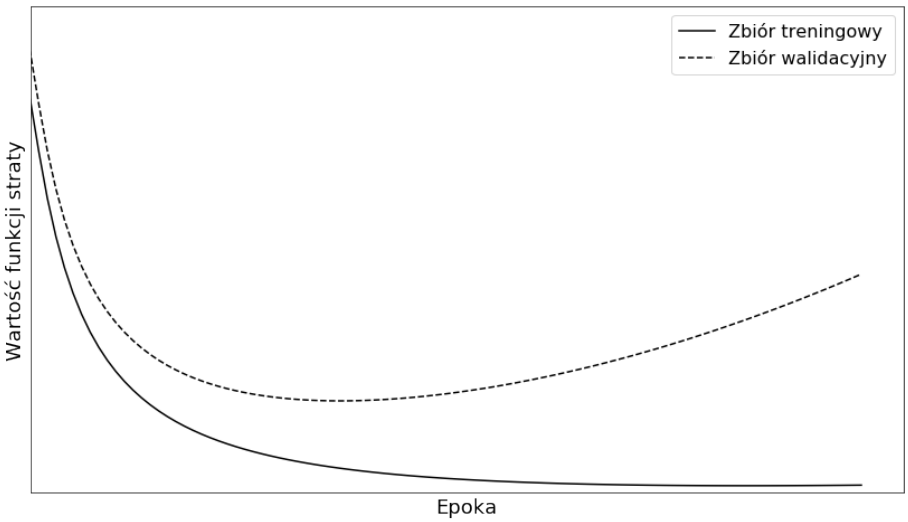
Rysunek 1: Na podstawie [Python Machine Learning Book by Sebastian Raschka]

się do danych w zbiorze uczącym co powoduje zmniejszenie wartości błędu na tym zbiorze lecz równocześnie jest przyczyną wzrostu błędu na zbiorze testowym. Nadmierne dopasowanie modelu to problem, który może się pojawiać gdy model zawiera więcej parametrów niż wymagałaby tego natura modelowanego zjawiska. Sieć neuronowa to struktura skłonna do przeuczania. Podczas gdy obserwowany błąd obliczany w oparciu o dane treningowe spada, w pewnym momencie wartość błędu dla zbioru walidacyjnego zaczyna wzrastać. Rysunek 2 przedstawia często zamieszczane w literaturze, wyidealizowane krzywe zmiany wartości funkcji straty w czasie, dla zbiorów treningowego i walidacyjnego. Najlepszy model predykcyjny miałby parametry, które odpowiadają momentowi globalnego minimum dla zbioru walidacyjnego.

W dziedzinie uczenia maszynowego, metoda wczesnego zatrzymania to forma regularyzacji, która pozwala uniknąć problemu przeuczenia, zatrzymując naukę modelu gdy wartość funkcji straty na zbiorze walidacyjnym zaczyna wzrastać. Rzeczywisty przebieg wartości funkcji straty ma wiele lokalnych minimów, dlatego na podstawie obserwacji krzywych uczenia dokonano wyboru kryteriów zatrzymania nauki modelu. Niech $\Theta_{wa}(t)$ to wartość funkcji straty na zbiorze walidacyjnym po t epokach, $\Theta_{min}(t)$ to dotychczasowe minimum funkcji straty na zbiorze walidacyjnym po t epokach, definiowane jako:

$$\Theta_{min}(t) \equiv \min_{t'<t} \Theta_{wa}(t')$$

Niech $\Theta_{sr}(t)$ będzię średnią wartością funkcji straty dla zbioru walidacyjnego z ostatnich 10 epok.



Rysunek 2: Wyidealizowane przykłady krzywych przedstawiających zmianę wartości funkcji straty na zbiorach treningowym i walidacyjnym, podczas nauki modelu

$$\Theta_{sr}(t) \equiv \frac{1}{10}\sum_{i=0}^{10}\Theta_{wa}(t-i)$$

Oraz zdefiniujmy pomocniczy parametr $GL(t)$

$$GL(t) \equiv \frac{\Theta_{sr}(t)}{\Theta_{min}} - 1$$

Podczas nauki przedstawioneg modelu, do wczesnego zatrzymania wystarczyło spełnienie jednego z dwóch obowiązujących warunków:

- $\Theta_{min}(t) = \Theta_{min}(t+200)$ dla wszystkich $t \in [t, t+200]$, brak zmniejszenia minimalnej wartości funkcji straty dla zbioru walidacyjnego przez 200 epok
- $GL(t) > 2$, względny wzrost średniej wartości funkcji straty przez ostatnie 10 epok względem osiągniętego minimum jest większy niż 200%

Po skończeniu nauki, wybierany jest model, który ma najmniejszą wartość funkcji straty na zbiorze testowym.

3 Ilość neuronów

Architektura sieci neuronowej, tzn. ilość warstw ukrytych oraz ilość neuronów w warstwach ukrytych jest zdeterminowana przez wymiar danych wejściowych,

Tabela 1: Liczba parametrów sieci neuonowej z dwoma warstwami ukrytymi w zależności od liczby neuronów w warstwach

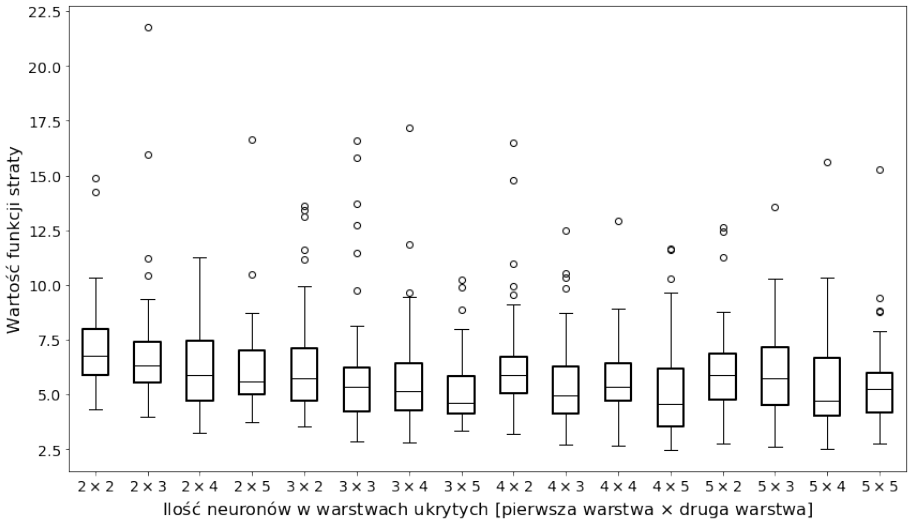
I warstwa \ II warstwa	2	3	4	5
2	14	18	22	26
3	19	24	29	34
4	24	30	36	42
5	29	36	43	50

rodzaj rozwiązywanego problemu (klasyfikacja czy regresja) oraz relację między zmiennymi objaśniającymi i zmienną objaśnianą.

Uogólniony model liniowy przydatny w szerokim zakresie zastosowań, nie potrzebuję żadnej warstwy ukrytej. Bywa szczególnie przydatny gdy zbiór zawiera mało danych lub są one obarczone dużą niedokładnością. Nawet w przypadku gdy relacja między zmiennymi jest lekko nieliniowa, użycie prostego modelu liniowego może skutkować lepszym uogólnieniem problemu niż skomplikowany model będący wrażliwy na każdy szum znajdujący się w danych. Zgodnie z uniwersalnym twierdzeniem aproksymacyjnym jedna warstwa ukryta z wystarczająco dużą liczbą neuronów wystarcza aby z dowolną dokładnością dowolną ciągłą funkcję [cybenko]. Jeśli zmienna objaśniająca jest jednowymiarowa, wydaje się, że nie odniesiemy żadnej korzyści z skonstruowania sieci neuronowej o więcej niż jednej warstwie ukrytej. Sprawy komplikują się jednak gdy zmienna wejściowa jest dwu lub więcej wymiarowa. Dwuwarstowa sieć neuronowa zachowuje właściwości jednowarstwowej sieci neuronowej oraz osiąga zdolność nauki każdego problemu klasyfikacyjnego [1995 Bishop 123], ponadto wielowarstwowa sieć neuronowa z dwoma warstwami może skutkować dokładniejszymi wynikami wykorzystując mniejszą ilość parametrów niż jednowarstwowa sieć [Chester (1990)]. Na tej podstawie, do rozwiązania problemu regresji gdzie wejściem jest para liczb (ε, Q^2) postanowiłem wybrać sieć neuronowa z dwoma warstwami ukrytymi.

Aby znaleźć odpowiednią liczbę neuronów w dwóch warstwach ukrytych, stworzyłem siatkę [2, 3, 4, 5] $\times$ [2, 3, 4, 5] neuronów i sprawdziłem, która konfiguracja daje najmniejszy błąd zbioru walidacyjnego. Dane zostały podzielone na zbiór treningowy i testowy w stosunku 2:1. Dla każdej konfiguracji wytrenowano 50 sieci i sprawdzono jak wygląda statystyka błędu. Tabela 1 zawiera porównanie liczby parametrów sieci neuronowej w zależności od liczby neuronów w warstwach ukrytych. Do eksperymentów wybrano konfiguracje charakteryzujące się rozsądną w porównaniu do rozmiaru danych wejsciowych liczbą parametrów. Rysunek 3 przedstawia rozkłady minimalnej wartości funkcji straty uzyskanej na danych walidacyjnych uzyskanej z 50 treningów sieci dla każdej konfiguracji ilości neuronów. Wykres pudełkowy to forma graficznej prezentacji rozkładu, która pozwala w łatwy sposób ukazać położenie, rozproszenie oraz kształt empirycznego rozkładu badanej cechy statystycznej. Konfiguracja 3 $\times$ 5 charakteryzuje się najniższą medianą wartości funkcji straty oraz małą liczbą

wartości odstających. Ta obserwacja pozwoliła zdecydować, że liczby neuronów będą wynosiły 3 i 5 w odpowiednio pierwszej i drugiej warstwie ukrytej, co za tym idzie sieć będzie miała 36 parametrów.

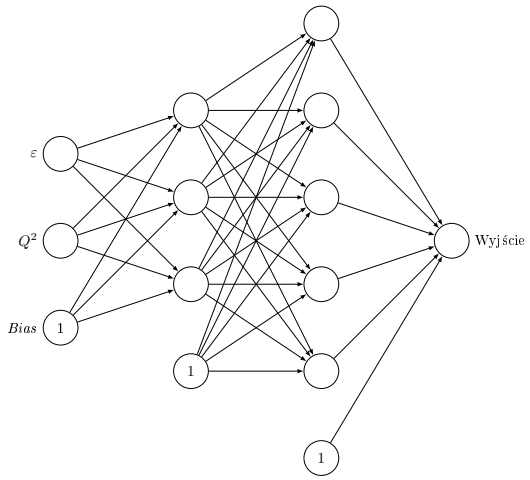


Rysunek 3: Wykresy pudełkowe przedstawiające rozkład wartości funkcji straty w zależności od ilości neuronów w pierwszej i drugiej warstwie ukrytej

4 Algorytm uczący

Bardzo istotnym elementem tworzonego modelu jest wybór algorytmu poszukującego minimum funkcji straty oznaczonej na potrzeby tego paragrafu jako $J(\theta)$. Na podstawie jego wyników aktualizowane będą parametry tworzonej sieci neuronowej. Bardzo pomocną koncepcją pozwalającą zrozumieć istotę trudności problemu jest powierzchnia błędu.

"*Każda z N wag i wartości progowych sieci (tzn. wszystkie wolne parametry modelu) traktowana jest jako jeden z wymiarów przestrzeni. W ten sposób każdy stan sieci, wyznaczony przez aktualne wartości jej N parametrów może być traktowany jako punkt na N-wymiarowej hiperpłaszczyźnie. N+1 wymiarem (zaznaczanym jako wysokość ponad wspomnianą wyżej hiperpowierzchnią) jest błąd, jaki popełnia sieć. Dla każdego możliwego zestawu wag i progów może więc zostać narysowany punkt w przestrzeni N+1 wymiarowej, w taki sposób, że stan sieci wynikający z aktualnego zestawu jej parametrów lokuje ten punkt na wspomnianej wyżej N-wymiarowej hiperpłaszczyźnie zaś wartość błędu, jaki popełnia sieć dla tych właśnie wartości parametrów stanowi wysokość umieszczenia punktu ponad tą płaszczyzną. Gdybyśmy opisaną procedurę powtórzyli dla wszystkich możliwych wartości kombinacji wag i progów sieci, wówczas otrzymalibyśmy "chmurę" punktów rozciągających się ponad wszystkimi punktami*



Rysunek 4: Schemat zastosowanej sieci neuronowej, która składa się z: i) warstwy wejściowej z dwoma neuronami, ii) dwóch warstw ukrytych z odpowiednio trzema i pięcioma neuronami, iii) warstwy wyjściowej z jednym neuronem. Linie zakończone strzałką oznaczają wagę odpowiadającą każdej z par neuronów

N-wymiarowej hiperpłaszczyzny parametrów sieci, tworzącą właśnie rozważaną powierzchnię błędu. Celem uczenia sieci jest znalezienie na tej wielowymiarowej powierzchni punktu o najmniejszej wysokości, czyli ustalenie takiego zestawu wag i progów, który odpowiada najmniejszej wartości błędu. Przy stosowaniu modeli liniowych z funkcją błędu opartą na sumie kwadratów powierzchnia błędu ma kształt paraboloidy (funkcji kwadratowej), ma więc kształt kielicha o gładkich powierzchniach bocznych i o jednym wyraźnym minimum. Z tego powodu wyznaczenie w tym przypadku wartości minimalnej nie stwarza większych problemów."[$https://www.statsoft.pl/textbook/stathome_stat.html?$]

Po za losowym poszukiwaniem parametrów, najłatwiejszym z nich i bardzo intuicyjnym jest metoda gradientu prostego (*gradient descent*)

$$\theta^{k+1} = \theta^k - \alpha \nabla J(\theta^k) \tag{1}$$

gdzie α to wybrany odpowiednio parametr szybkości uczenia (*learning rate*) odpowiedzialny za stopień zmiany parametrów w kolejnych iteracjach. Jeśli θ^0 znajduje się odpowiednio blisko minimum funkcji, i parametr α jest wystarczająco niewielki, algorytm osiąga liniową zbieżność [Dennis, J., Schnabel, R.B.: Numerical Methods For Unconstrained Optimization and Nonlinear Equations. Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1983)]. W ogólności metoda gradientu prostego gwarantuje zbieżność do globalnego minimum w przypadku funkcji błędu o wypukłej powierzchni i do lokalnego minimum dla funkcji błędu o powierzchni nie wypukłej. W ogólności jednak jest bardzo wolny, co jest jego największą słabością.

Wyobraźmy sobie, że poszukiwanie minimum powierzchni błędu to przemierzanie przestrzeni pełnej dolin, pagórków, wąwozów. W kolejnych iteracjach przeskakujemy między tymi obszarami, w pewnym momencie może się zdarzyć, że gradient zaniknie lub będzie bardzo słaby a nasze poszukiwania zatrzymają się nie osiągając wystarczającego minimum. Idea pędu inspirowana zjawiskami fizycznymi to nadanie gradientowi krótkotrwałej pamięci. Posługując się kolej-

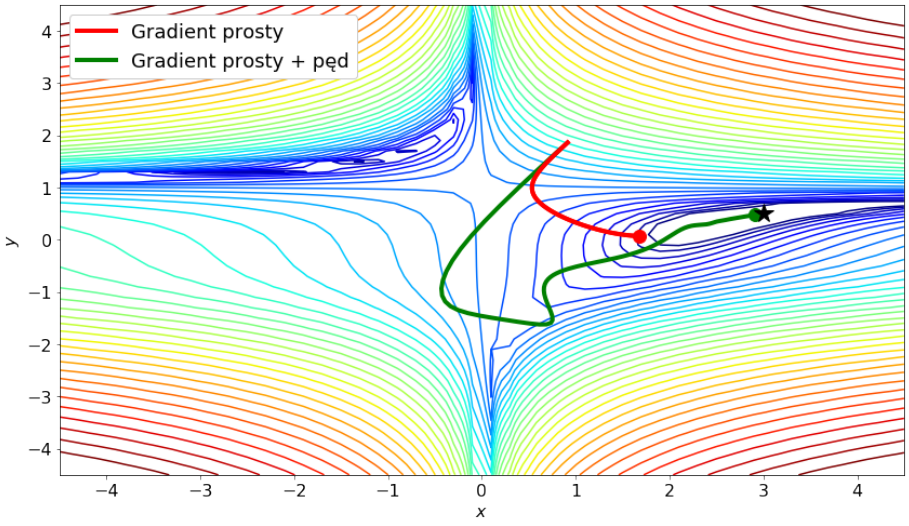
ną analogią, popchnięta w dół piłka nabierając prędkości zwiększa swój pęd. To samo dzieje się z parametrami sieci, wartość pędu wzrasta dla wymiarów, których gradienty wskazują te same kierunki i zmniejsza modyfikacje wartości dla wymiarów, w których gradienty zmieniają kierunki. W rezultacje otrzymujemy szybszą zbieżność i mniejsze oscylacje.

$$v^{k+1} = \beta v^k + \nabla J(\theta^k) \tag{2}$$

$$\theta^{k+1} = \theta^k - \alpha v^{k+1} \tag{3}$$

Zmiana jest niewielka, gdy $\beta = 0$, otrzymujemy zwykłą metodę gradientu prostego, zazwyczaj jednak ustala się wartość parametru β, zwanego pędem na około 0.9. [1986 Nature, Learning representations by back-propagating errors David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams]

Porównanie efektywności przedstawionych wyżej algorytmów znajduje się na Rysunek 5, w zaprezentowanym przykładzie metoda gradientu prostego potrzebuje około 10 razy więcej iteracji od modyfikacji z pędem aby dotrzeć do minimum zaprezentowanej funkcji. Jest to przykład świadczący o tym jak duży wpływ na szybkość działania algorytmu wywiera ta niewieka modyfikacja.



Rysunek 5: Funkcja $f(x, y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$, osiąga minimum równe 0, w punkcie $(3, 0.5)$ oznaczonym czarną gwiazdą. Grafika przedstawia porównanie działania metody gradientu prostego oraz jego modyfikacji poprzez dodanie pędu. Przyjmując, że punkt początkowy to $(2, 1)$, $\alpha = 0.001$ i $\beta = 0.9$, możemy prześledzić trajektorie algorytmów przez pierwsze 500 iteracji działania.

Wykorzystany podczas treningu modelu algorytm korzysta jednak z jeszcze z jednej modyfikacji. Nie chcielibyśmy aby piłka spuszczona w dół ślepo podążała

za zboczem widząc, że za niedługo mocno się ono podniesie. Przyspieszenie Nesterova (*NAG*) jest sposobem na uwzględnienie podczas obliczania gradientu przybliżonej przyszłej pozycji parametrów sieci.

$$v^{k+1} = \beta v^k + \nabla J(\theta^k - \beta v^k) \tag{4}$$

$$\theta^{k+1} = \theta^k - \alpha v^{k+1} \tag{5}$$

[https://arxiv.org/abs/1609.04747]

Niezywkle istotnym parametrem algorytmu jest α, jego niezmienność wraz z postępem iteracji powoduje bardzo niską efektywność algorytmu. Ze względu na metodę zmiany tego parametru, który może być indywidualnie ustalany dla każdej wagi powstało wiele szeroko wykorzystywanych algorytmów. Do najpopularniejszych należą między innymi *Adam, Nadam, Adagrad, Adadelta, AMSGrad, RMSprop*.

W swoim algorytmie postanowiłem dokonywać zmiany parametru α wraz ze wzrostem iteracji. Ponadto szybkość uczenia zależna jest od wybranego parametru λ decydującego o tym z jaką szybkością maleje.

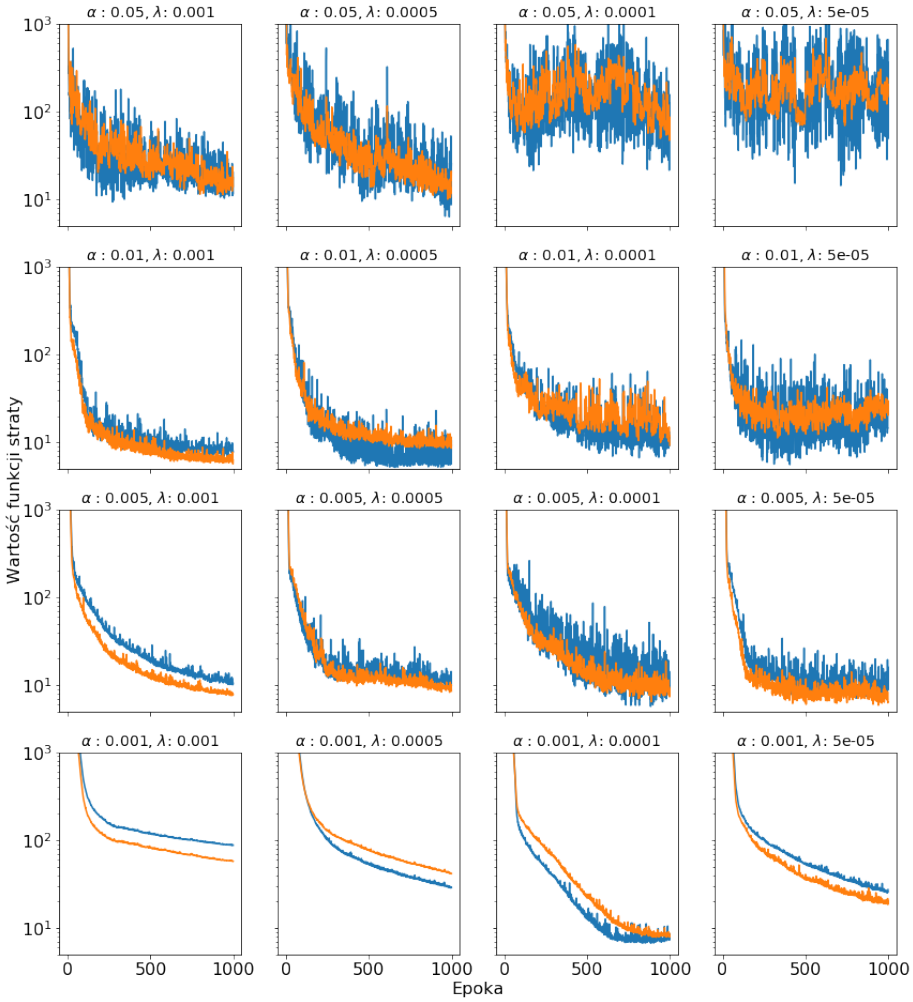
$$\alpha(i) = \alpha_0 \times \frac{1}{1 + \lambda \times i} \tag{6}$$

Rysunek 6 przedstawia porównanie przykładowych krzywych zmian wartości funkcji straty w czasie dla różnych wartości α i λ. Na ich podstawie widać jak duży wpływ wnosi parametr α w proces nauki modelu. Zbyt duża szybkość uczenia powoduje bardzo duże oscylacje krzywej funkcji straty, za mała wartość α bardzo mocno spowalnia proces nauki. Pewien kompromis przynosi wybranie odpowiednio dużej początkowej wartości szybkości uczenia, co przynosi szybkie przejście algorytmu w obszar minimum i następnie zmniejszenie go do wartości potrafiącej efektywnie dalej poszukiwać optimum. Zadowalający przebieg mają krzywe o parametrach $\alpha = 0.005$, $\lambda = 0.001$, które przedstawiają porządany, eksponencjalny kształt o niewielkiej oscylacji. Na podstawie powyższej analizy to właśnie te hiperparametry zostały wykorzystane w modelu, dodatkowo parametr pędu β został ustalony na wartość 0.9

5 Dane wejściowe

6 Funkcja straty

7 Replikacja danych



Rysunek 6: Porównanie przykładowych krzywych zmiany wartości funkcji straty w czasie dla zbiorów treningowego (kolor pomarańczowy) i walidacyjnego (kolor niebieski) ze względu na parametry α (*learning rate*) oraz λ (*decay*)