Algorytmy uczenia sieci neuronowych odporne na błędy w danych

autor: Andrzej Rusiecki

Lipiec 2007

Spis treści

Spis rysunków

Spis tabel

Wstęp

Wprowadzenie

Sieci neuronowe. Już od ponad dwudziestu lat dziedzina wiedzy zajmująca się opisem, projektowaniem i konstruowaniem sztucznych sieci neuronowych (SSN) przeżywa okres dynamicznego rozwoju [44, 59, 25]. Mimo, iż przeznaczone były początkowo, do modelowania działania układu nerwowego, sieci neuronowe zyskały popularność w wielu praktycznych zastosowaniach. Stało się to głównie dzięki takim ich zaletom, jak możliwość równoległego przetwarzania informacji, zdolność uogólniania obserwowanych danych, czy możliwość tworzenia modelu na podstawie zbioru danych uczących [19], co pozwala im w wielu przypadkach z powodzeniem zastępować, często skomplikowane i mniej uniwersalne, klasyczne algorytmy dedykowane do rozwiązania konkretnych problemów.

Nie bez znaczenia wydaje się również fakt, że do stosowania sieci neuronowych nie jest konieczna, z praktycznego punktu widzenia, znajomość zasady ich działania. Traktować je można bowiem jako tzw. czarną skrzynkę, która, przy odpowiednio podanych danych wejściowych, potrafi zbudować szukaną zależność.

Jak już wspomniano, sieci neuronowe stosowane są coraz częściej jako proste narzędzie, które nie wymaga od potencjalnego użytkownika znajomości zasad jego działania. Zwykle otrzymuje on gotowy pakiet (program), który wymaga od niego jedynie podania danych, na których sieć będzie uczona, w odpowiednim formacie. Po przeprowadzeniu symulacji dostarczany jest wynik, o którym często nie można wiele powiedzieć, tzn. nie wiadomo jakie warunki muszą być spełnione, aby można było uznać go za prawidłowy. Oczywiście, odpowiednie przetworzenie danych, znalezienie właściwej struktury sieci, a potem dobranie odpowiednich parametrów algorytmu

uczenia znacznie zwiększają szansę dobrego działania sieci, niemniej w sytuacji, kiedy użytkownik nie jest w stanie samodzielnie ustalić najbardziej korzystnych warunków, wymaga się w praktyce, aby sieci neuronowe umożliwiały uzyskiwanie dobrych wyników (a więc prawidłowe modelowanie zadanych systemów) bez konieczności jego ingerencji.

Błędy w danych uczących. Jednym z problemów, z jakimi należy się zmierzyć, starając się usprawnić działania SSN, jest, częstokroć pomijany, problem błędów mogących pojawić się w danych uczących. O ile różne metody klasyfikacji, czy aproksymacji, w tym oparte na SSN, są stosunkowo odporne na niewielkie zakłócenia danych uczących, istnienie nawet małej ilości danych znacznie różniących się od ogółu, spowodować może ich całkowicie nieprawidłowe działanie.

Jak się okazuje, w naukach przyrodniczych, w przypadku bezpośrednio mierzalnych danych pomiarowych, zawartość błędów grubych waha się zwykle w granicach kilku procent, jeśli jednak weźmie się pod uwagę bardziej opisowe dziedziny wiedzy, np. medycynę, okazuje się, że ilość danych odstających przekracza nierzadko 10% [21], sięgając czasami 20% wszystkich obserwacji.

By uświadomić sobie, jak katastrofalny wpływ na budowanie modelu opartego o serię pomiarów, może mieć pojawienie się w nich błędu grubego, wystarczy banalny przykład. Wyobrazić sobie wystarczy, że mierzymy wzrost kilkudziesięciu sadzonek roślin w pewnym botanicznym eksperymencie. Przypadkowa zamiana jednostki przy odczycie jednego pomiaru, np. z milimetrów na metry, w konsekwencji powoduje powstanie wartości kilka rzędów większej od ogółu pomiarów. Policzona przy jej uwzględnieniu średnia będzie więc również znacznie większa od rzeczywistej wartości, a największy wpływ na nią będzie mieć właśnie owa błędna dana.

Oczywiście, powiedzieć można, iż tego typu błąd jest łatwy do wykrycia, ale ten przerysowany przykład ma jedynie pokazać potencjalny wpływ pojedynczego grubego błędu. Gdy bowiem skomplikujemy eksperyment myślowy, dodając, że wzrost sadzonki jest zaledwie jednym z mierzonych parametrów, ponadto liczona jest nie średnia, lecz za pomocą sieci neuronowej szuka się zależności pomiędzy badanymi parametrami, a np. zwartością pewnej substancji w tkankach roślin, możemy

jedynie stwierdzić, że tak naprawdę nie wiadomo, jaki wpływ będzie miał ów odstający pomiar na proces budowania modelu zjawiska. Sytuacja jest tu o tyle bardziej niebezpieczna, że otrzymany wynik, wcale nie musi skłaniać badacza do powzięcia jakichkolwiek podejrzeń, że może on być nieprawidłowy.

Motywacja podjęcia badań

O ile na gruncie statystyki rozwija się już od dłuższego czasu nurt zwany statystyką odporną [21, 28, 42, 46] i zajmujący się właśnie opisywaniem i docelowo niwelowaniem wpływu danych odstających i błędów grubych na rezultaty działania metod statystycznych, o tyle analogiczne podejście nie pojawiało się jeszcze praktycznie w dziedzinie sztucznych sieci neuronowych, a problem występowania dużych zakłóceń w danych uczących poruszony został zaledwie w kilku artykułach [7, 9, 34, 46]. Ponadto wszystkie, z kilku istniejących do tej pory metod, działają na bardzo podobnej zasadzie, która powoduje, że ich odporność i efektywność jest mocno ograniczona. Opierają się one bowiem jedynie na pewnych modyfikacjach funkcji błędu, mających zmniejszać wpływ dużych zakłóceń na proces uczenia sieci. Z tego powodu uzasadnione wydaje się podjęcie próby uwzględnienia tematyki błędów grubych i danych odstających od ogółu w metodach uczenia sieci neuronowych.

Charakterystyka problemu. Podsumowując powyższe rozważania, stwierdzić należy, że:

- błędy grube mogą pojawić się w danych uczących, a ich ilość sięga nierzadko kilkunastu procent, dodatkowo nie zawsze są łatwe do wykrycia;
- algorytmy uczenia sieci neuronowych przeznaczone są do uczenia na danych czystych i w przypadku pojawienia się w nich dużych zakłóceń przestają działać prawidłowo;
- działanie sieci neuronowej uczonej na zanieczyszczonych danych jest nieprzewidywalne i najczęściej zupełnie błędne;

- użytkownicy sieci neuronowych, stosując je jako narzędzie, mogą nie zidentyfikować błędów w danych i nie zauważyć, że wyniki otrzymywane na podstawie symulacji sieci są nieprawidłowe.

Sformułowanie zadania. Problem postawiony więc może zostać w następujący sposób: opracować należy algorytmy uczenia sieci neuronowych, które będą odporne na różne typy dużych zakłóceń mogących pojawić się w danych uczących.

Przyjęte założenia. Realizując zadanie poszukiwania metod uczenia odpornych na błędy w danych, przyjęto kilka założeń, które powinny zostać spełnione przez tego typu algorytm.

- Po pierwsze, odporny algorytm uczenia (bo tak w skrócie nazywane będą algorytmy uczenia odporne na błędy w danych uczących) powinien działać prawidłowo również dla ciągu uczącego pozbawionego jakichkolwiek zakłóceń. Ewentualnie można sobie bowiem wyobrazić sytuację, w której algorytm działa prawidłowo jedynie dla pewnego określonego typu zakłóceń, przy zadanej ich ilości, ale jego wartość praktyczna jest wtedy raczej znikoma, biorąc pod uwagę, że, jak już wspominano, najczęściej nie znamy ani charakteru, ani liczby błędów, jakie wystąpiły podczas przygotowywania danych.

- Następny postulat dotyczy właśnie odporności. Odporne algorytmy uczenia muszą działać stosunkowo dobrze również w obecności dużych zakłóceń pojawiających się w ciągu uczącym. Co to znaczy stosunkowo dobrze? Otóż, niewątpliwie idealną byłaby sytuacja, w której błędne dane nie miałyby żadnego wpływu na proces uczenia. W praktyce jednak można uznać, że odporną nazwiemy metodę, która działa lepiej niż algorytm klasyczny, nieprzeznaczony do uczenia na danych zawierających zakłócenia.

- Ostatnie z założeń, jakie przyjęto na temat odpornych algorytmów uczenia sieci neuronowych, dotyczy zakresu możliwych do wykorzystania podejść, a więc granic zakreślanych przez pojęcie "algorytmu uczenia". Najprościej mówiąc, w niniejszej pracy nie zajmowano się metodami mogącymi wyłapać błędy grube podczas wstępnej analizy i przetwarzania danych poza siecią neuronową.

Do takich zastosowań można bowiem z powodzeniem używać różnych dedykowanych metod statystycznych, które w ogóle nie biorą pod uwagę, do czego owe przetworzone dane będą potem użyte. Takie wyłapywanie i usuwanie dużych zakłóceń działać będzie zapewne równie dobrze, jeśli następnie do dalszej obróbki danych zastosujemy algorytmy dedykowane, czy np. spróbujemy je zamodelować za pomocą sieci neuronowych.

Mówiąc więc o odpornych algorytmach uczenia, mamy na myśli sytuację, w której mechanizm uzyskiwania odporności, a więc eliminacji, bądź też zmniejszania wpływu zakłóceń na proces uczenia, wbudowany jest w samą metodą trenowania sieci. Ewentualna wstępna obróbka ciągu uczącego również powinna uwzględniać specyfikę używanego dalej narzędzia, jakim jest sieć neuronowa.

Nowe elementy w pracy. Badania wykonane w ramach przygotowania niniejszej pracy, można traktować jako zupełnie nowe z dwóch powodów:

1. Istniejące wcześniej próby konstruowania odpornych algorytmów uczenia biorą pod uwagę tylko zakłócenia w wektorze wyjściowym systemu, modelowane za pomocą sumy rozkładów normalnych, podczas, gdy w niniejszej pracy dodatkowo skupiono się na zakłóceniach wektora wejściowego, oraz błędach wynikających z zastąpienia części danych szumem.

2. W niniejszej pracy wykorzystano różne (nie używane wcześniej) techniki uzyskiwania odporności, takie jak zmienny współczynnik uczenia, zmienna funkcja aktywacji neuronów, wstępny podział danych uczących na zakłócenia i dane czyste, czy wreszcie zastosowanie nowych funkcji kryterialnych wykorzystujących odrzucanie części błędów.

Można więc powiedzieć, że w pracy tej rozwiązywano zupełnie nowymi metodami problem, którego niektóre aspekty nie były jeszcze brane pod uwagę, a inne nie zostały rozpatrzone w sposób gruntowny.

Zadania zrealizowane w pracy

Za najważniejsze osiągnięcia niniejszej pracy uznać należy opracowanie odpornych algorytmów uczenia sieci jednokierunkowych, które działają lepiej niż metody istniejące i uodparniają proces uczenia na różnego typu zakłócenia mogące pojawić się w danych.

Najogólniej rzecz ujmując, w ramach badań służących rozwiązaniu omówionego problemu, zrealizowano następujące zadania:

1. Sformułowano problem błędów grubych mogących pojawić się w danych służących do uczenia sieci neuronowych.

2. Opracowano odporne algorytmy uczenia opierające się na różnych mechanizmach niwelowania wpływu zakłóceń na przebieg procesu uczenia, działające lepiej niż istniejące rozwiązania.

3. Opracowano metodykę badania odporności algorytmów uczenia sieci.

4. Przeprowadzono badania symulacyjne mające na celu zweryfikowanie skuteczności i porównanie działania zaprojektowanych metod.

Podczas badań mających na celu rozwiązanie postawionego problemu, opracowano kilka wersji algorytmów uczenia, które pozwalają na zmniejszenie wpływu danych odstających na przebieg procesu uczenia sieci. Wykorzystują one różne mechanizmy, aby zmniejszyć lub wyeliminować wpływ błędów grubych. Opracowane metody idą w poszukiwaniu owych mechanizmów znacznie dalej, niż kilka istniejących wcześniej modyfikacji algorytmu uczenia i w wielu przypadkach dają w praktyce zdecydowanie lepsze rezultaty. Ponadto, są one w większości, w przeciwieństwie do algorytmów wcześniejszych, przeznaczone do uczenia skumulowanego, a nie "on-line" i mogą być stosowane również z metodami drugiego rzędu.

Opracowane algorytmy można pogrupować na: metody ze zmodyfikowaną funkcją kryterialną, metody ze zmiennym współczynnikiem uczenia, metodę ze zmienną funkcją aktywacji, algorytm dedykowany do zmodyfikowanej funkcji kryterialnej, oraz algorytm ze wstępną identyfikacją zakłóceń.

Opracowane odporne algorytmy uczenia

Odporne algorytmy uczenia sieci, opracowane przez autora niniejszej pracy, zostały krótko przedstawione i omówione poniżej:

- Pierwszy z algorytmów bazuje na odpornym estymatorze najmniejszych przycinanych kwadratów LTS (*least trimmed squares*) [50]. Zaproponowano tu zastąpienie, minimalizowanej zwykle w procesie uczenia sieci neuronowych, funkcji kryterialnej nową funkcją, w której zamiast sumy kwadratów błędów stosuje się sumę przycinaną. Nie wymaga to wprowadzania dużych modyfikacji w algorytm uczenia sieci, pozwalając jednocześnie na ograniczenie wpływu na proces uczenia zakłóceń różnego typu. Metoda ta zapewnia częściową odporność nie tylko na błędy w zadanym wektorze wyjściowym sieci, ale również na zakłócenia na wejściu (tzw. *leverage points*) [21, 52]. W pracy przedstawiono dwa różniące się nieznaczne warianty algorytmu, z których jeden wymaga apriorycznej wiedzy na temat zawartości błędów w danych uczących, drugi zaś umożliwia jej szacowanie na podstawie aktualnego błędu popełnianego przez sieć.

- Na podstawie algorytmu LTS, opracowano kolejny algorytm z nową minimalizowaną funkcją celu LTLS (*least trimmed log squares*). Funkcję taką, która nie bazuje bezpośrednio na żadnym z odpornych estymatorów, zaproponowano jako połączenie wspomnianej idei przycinania residuów z zastosowaniem ciągłej funkcji ograniczającej wpływ dużych zakłóceń przedstawionej w [34].

- Kolejna z opracowanych odpornych metod, nazwana VLR (*variable learning rate*), wykorzystuje zmienny i zależny od aktualnego błędu sieci, współczynnik kroku uczenia i przeznaczona jest do stosowania w strategii, w której modyfikacja wag sieci następuje po prezentacji każdego wzorca uczącego. Odpowiednie tłumienie postępu algorytmu optymalizacyjnego, w kierunku ekstremów powodowanych przez dane podejrzane o bycie zakłóceniami, powoduje zmniejszenie wpływu tych ostatnich na działanie sieci.

- Opracowany został również algorytm ALR (*adaptive learning rate*), w którym współczynnik uczenia dobierany jest adaptacyjnie poprzez minimalizację kierunkową pewnej funkcji zależnej od aktualnych błędów tak, aby zmniejszyć wpływ największych residuów na postęp procesu uczenia. Zaprojektowany on został do uczenia skumulowanego algorytmami gradientowymi drugiego rzędu.

- Kolejny z nowych algorytmów, nazwany ATF (*adaptive transfer function*), wykorzystuje ideę zmiennych w czasie funkcji aktywacji neuronów, które zmieniają swój kształt tak, aby umożliwić zwiększenie odporności na pojawianie się dużych zakłóceń. Choć podejście to jest, od strony efektów, podobne do zmiennego współczynnika uczenia, różni się od niego sposobem wprowadzania zmian w trajektorii uczenia.

- Bazując na algorytmie Levenberga-Marquardt'a [37, 20], opracowano metodę uczenia dedykowaną dla kryterium LMLS (*Least Mean Log Sqaures*) [34]. Pozwala ona na potencjalnie szybsze uczenie sieci przy wykorzystaniu jednej ze znanych z literatury funkcji kryterialnych, zapewniających częściową odporność na błędy grube. Metoda ta opiera się na aproksymacji hesjanu za pomocą odpowiedniego przekształcenia macierzy residuów, oraz jakobianu.

- Ostatni z opracowanych algorytmów wykorzystuje wstępną analizę danych uczących za pomocą odpornego estymatora MCD (*minimum covariance determinant*) [39] i na tej podstawie przypisuje poszczególnym obserwacjom odpowiednie współczynniki wagowe, które wykorzystywane są potem w procesie uczenia. Są one dobrane tak, aby zmniejszyć wpływ na budowany przez sieć model, danych podejrzanych o bycie błędami grubymi. Wykrycie obrazów uczących, podejrzanych o bycie błędami grubymi, następuje więc przed rozpoczęciem trenowania sieci, natomiast informacja o nich przenoszona jest do zmodyfikowanej odpowiednio funkcji kryterialnej.

Układ pracy

Niniejsza praca składa się ze wstępu i siedmiu rozdziałów, w których pogrupowane zostały: sformułowanie problemu badawczego, podstawy teoretyczne, wyniki pracy, rezultaty przeprowadzonych eksperymentów, oraz ich omówienie.

Rozdział 1. W rozdziale pierwszym przedstawiono problem błędów grubych, oraz obserwacji odstających, które mogą się potencjalnie pojawić w danych uczących przeznaczonych do trenowania sieci neuronowych. Druga sekcja rozdziału przeznaczona została na opisanie, używanych podczas przeprowadzonych badań, modeli zakłóceń. W trzeciej sekcji zawarto krótkie podsumowanie problemu badawczego.

Rozdział 2. Rozdział drugi poświęcony został podstawom teoretycznym. W pierwszej sekcji zawarto krótkie wprowadzenie w tematykę sieci neuronowych, a w szczególności sieci jednokierunkowych sigmoidalnych. Sekcja druga przedstawia opis wybranych gradientowych algorytmów uczenia sieci jednokierunkowych, wykorzystywanych dalej w niniejszej pracy.

Rozdział 3. Rozdział trzeci dotyczy odpornych algorytmów uczenia. Wprowadza on w tematykę odporności metod uczenia sieci neuronowych i prezentuje istniejące na tym polu osiągnięcia. Można w nim znaleźć dokładny opis czterech istniejących odpornych algorytmów. Dodatkowo, w ostatniej sekcji, zaprezentowano w nim krótkie eksperymentalne porównane omówionych metod.

Rozdział 4. Osobny rozdział, najistotniejszy z punktu widzenia niniejszej pracy, poświęcony został odpornym algorytmom uczenia opracowanym przez autora. W kolejnych sekcjach podaje on ich pełen przegląd, wraz z wyprowadzeniem, dokładnym sposobem działania i opisem każdego z nich.

Rozdział 5. Rozdział piąty przedstawia opis metodyki badania opracowanych odpornych metod uczenia sieci. Zawarto w nim dokładne opisy sposobu przeprowadzania symulacji, stosowanych modeli błędów w danych, oraz zadań testowych.

Rozdział 6. W rozdziale szóstym przedstawiono wyniki eksperymentów numerycznych. W sekcji pierwszej, zebrano w formie tabel i wykresów, rezultaty przeprowadzonych badań, wraz z komentarzem. W sekcji drugiej znajduje się krótkie podsumowanie, oraz porównanie działania poszczególnych algorytmów. Sekcja trzecia omawia, wykryte na podstawie symulacji, dodatkowe czynniki mogące wpływać na odporność sieci neuronowych.

Rozdział 7. Ostatni rozdział pracy zawiera podsumowanie przedstawionych wyników, wraz z podkreślonym jeszcze raz krótkim spisem osiągniętych rezultatów, oraz opis dalszych potencjalnych kierunków badań, związanych z uodparnianiem metod uczenia sieci neuronowych.

Podziękowania

Pragnę szczególnie podziękować Pani dr hab. inż. Ewie Skubalskiej-Rafajłowicz za opiekę naukową, wszechstronną pomoc, oraz celne uwagi dotyczące zarówno zawartości, jak i formy niniejszej rozprawy.

Chciałbym również wyrazić wdzięczność Profesorowi Ewarystowi Rafajłowiczowi, oraz wszystkim kolegom z zakładu, których komentarze pozwoliły mi poprawić jakość moich badań.

Serdecznie dziękuję także Rodzicom, na których wsparcie zawsze mogłem liczyć.

Najgorętsze podziękowania należą się mojej Żonie i Synowi, bez których wyrozumiałości i cierpliwości ta praca nigdy by nie powstała.

Finansowanie. Badania związane z powstaniem tej pracy były częściowo finansowane z grantu Ministerstwa Nauki i Szkolnictwa Wyższego na lata 2006-2009.

Rozdział 1

Sformułowanie problemu

1.1 Dane odstające od ogółu i błędy grube

Angielskie słowo *outlier* nie ma jednoznacznego odpowiednika w języku polskim. W większości przypadków tłumaczy się je jako daną odstającą (w domyśle "od reszty danych"). Brak tu również niestety synonimów, w związku z czym w niniejszej pracy określenie to pada stosunkowo często. Wzorując się na [22], daną odstającą nazwiemy każdą obserwację, która odróżnia się od większości danych. Podobna definicja to: obserwacja, nie przystająca do obrazu tworzonego przez większość danych [21]. Jak widać, określenie to jest wielce nieprecyzyjne, gdyby bowiem dało się je uściślić problem danych odstających praktycznie przestał by istnieć, gdyż mielibyśmy prosty sposób ich identyfikacji.

Dane odstające stosunkowo często myli się, w potocznym rozumieniu, z danymi błędnymi, podczas gdy nierzadko okazują się one dostarczać najbardziej istotnych informacji. Przysłowiowymi stały się już w literaturze statystycznej przykłady poszukiwania osoby do poślubienia, poszukiwania najkorzystniejszej inwestycji, najlepszego studenta, doktoranta, naukowca, czy innych osób lub zjawisk "odstających" w dziedzinie swoich umiejętności lub właściwości. Dobrych przykładów dostarcza także klimatologia, w której ważną częścią składającą się na opis klimatu są obserwowane na przestrzeni wielu lat ekstrema temperatury, opadów, etc. Odkrycie dziury ozonowej nad Antarktydą wiąże się właśnie z prawidłowym i poważnym przeanalizowaniem danych odstających [42]. Niemniej jednak, w wielu przypadkach dane odstające mogą być rzeczywiście tożsame z błędami grubymi, będąc następstwem

błędów pomiarowych, czy pomyłek związanych z archiwizacją i przetwarzaniem danych.

Gdy już się pojawią, dane odstające powinny zostać przeanalizowane pod względem tego, czy można się w nich doszukać jakiegoś wzorca, czy są skutkiem błędów powstałych w procesie ich akwizycji, czy też może dają się wyjaśnić przy zastosowaniu trochę innego modelu opisywanego zjawiska lub procesu. Najogólniej rzecz ujmując, założyć można dwa sposoby postępowania z nimi [21]. Po pierwsze, możliwe jest wstępne wyodrębnienie zbioru obserwacji uznawanych za dane odstające i ewentualne dalsze ich przetwarzanie oddzielnie w stosunku do reszty. Druga możliwość to z kolei zastosowanie pewnych procedur, które w jakiś sposób wyeliminują wpływ błędów grubych, a zarazem nie zatracą istotnych informacji, dopiero na etapie budowania modelu. Ten drugi przypadek wydaje się trudniejszy w realizacji, ale równocześnie wskazany w przypadku uodporniania algorytmów uczenia sieci neuronowych.

Jeśli szukać przykładów prób rozwiązania problemu błędów grubych, zauważyć można, iż większość metod służących wykrywaniu i analizowaniu danych zawierających punkty odstające, związana jest bezpośrednio z dziedziną odpornej statystyki [28, 42], będącej poddyscypliną statystyki matematycznej. Podczas, gdy klasyczne procedury statystyczne powinny dawać prawidłowe wyniki przy określonych zbiorach założeń, metody odporne mają radzić sobie również w sytuacji, gdy jedno, konkretne założenie nie jest spełnione. Założenie takie, w praktyce, dotyczy najczęściej rozkładu danych lub residuów, więc jeżeli nie jest ono zachowane, tradycyjne metody stają się w większości niewiarygodne. Łatwo zauważyć, że sytuacja taka odpowiada właśnie istnieniu w zbiorze obserwacji danych odstających.

W literaturze statystycznej wyróżnić możemy dwa główne paradygmaty dotyczące odpornych procedur [28, 21]. Pierwszy z nich mówi, że mamy do czynienia z danymi, spośród których mniej niż połowa to dane odstające. Odporna metoda powinna idealnie klasyfikować całość danych na 2 zbiory - danych "czystych" i odstających właśnie. Drugi paradygmat zwany paradygmatem asymptotyczności (*asymptotic paradigm*) używa asymptotycznego rozkładu do aproksymacji rozkładu estymatora przy dużych ilościach obserwacji. W tym przypadku niezbędne jest określenie tego, co estymować ma estymator, jaki jest jego asymptotyczny rozkład

(w granicy w nieskończoności), zbieżność i jak duża musi być próbka danych, aby zadziałał prawidłowo. Ponadto, ze względów praktycznych istotne jest, czy da się go w prosty sposób (a czasem w ogóle) obliczyć. Jest to o tyle ważne, że w bogatej literaturze statystycznej znaleźć można wiele odpornych estymatorów, których użycie wymaga często rozwiązania skomplikowanych problemów optymalizacyjnych [43].

Dodać również należy, iż nie istnieje oczywiście żadne ogólne i uniwersalne rozwiązanie problemu błędów grubych również w dziedzinie poszukiwania odpornych metod statystycznych. Każda z nich ma trochę inne własności i wykazuje inną efektywność w zależności od typu danych, do których jest stosowana.

Jak już wspomniano, ważną klasę odpornych statystyk stanowią metody powstałe w celu radzenia sobie z sytuacją, gdy założenie na temat rozkładu błędów w modelowanych danych jest niewłaściwe. Zjawisko takie ma, na przykład, miejsce właśnie wtedy, gdy w zbiorze obserwacji pojawiają się dane odstające. Inna możliwość to rozkłady błędów z tzw. "ciężkimi ogonami" (*heavy-tailed*), zbliżone do rozkładu Cauchy'ego, czy rozkładu podwójnie ekspotencjalnego.

Bezpośrednie przeniesienie dokonań odpornej statystyki na grunt informatyki, a w tym wypadku sieci neuronowych, jest oczywiście niemożliwe, ale warto mieć świadomość, że niektóre z jej wytworów mogą również znaleźć zastosowanie w algorytmach uczenia sieci. W szczególności funkcje służące do konstruowania odpornych estymatorów wartości oczekiwanej, bądź wariancji mogą stanowić wskazówkę właśnie przy konstruowaniu funkcji minimalizowanych w procesie uczenia sieci. W ten właśnie sposób działają istniejące odporne algorytmy uczenia sieci, oraz część metod opracowanych w ramach niniejszych badań.

1.2 Modele błędów w danych

Przy próbie konstruowania algorytmów uczących odpornych na błędy w danych, należy sobie odpowiedzieć na pytanie, w jaki sposób owe błędy można zamodelować. Oczywiście, jak już wspominano, idealną byłaby sytuacja, w której metoda uzyskiwałaby dobre wyniki dla każdego rodzaju zakłóceń, niemniej stworzenie odpowiedniego modelu przydatne może być choćby z punktu widzenia oceny jakości działania i porównania poszczególnych algorytmów.

W literaturze, dotyczącej odpornych metod uczenia sieci [7, 9, 34], spotyka się najczęściej błędy modelowane za pomocą sumy rozkładów normalnych o zerowej wartości oczekiwanej, a więc tzw. *gross error model*, opisany w sekcji 1.2.1. Dodatkowo [46] pojawiają się błędy modelowane w podobny sposób, lecz przy użyciu rozkładu niesymetrycznego (sekcja 1.2.2). W niniejszej pracy założono ponadto model błędów, w którym mogą się one pojawiać również w wektorze wejściowym sieci, czy też, ogólniej mówiąc, modelowanej relacji.

1.2.1 Model błędów grubych

Najczęściej spotykany w literaturze, w tym praktycznie we wszystkich publikacjach dotyczących odpornych metod uczenia sieci neuronowych, jest tzw. *gross error model* [7, 9]. Do danych czystych, (dokładniej rzecz biorąc do wektora y) dodawane są tu zakłócenia generowane za pomocą dwóch składowych następująco:

$$F = (1 - \delta)G + \delta H, \tag{1.1}$$

gdzie F jest rozkładem błędów w danych, natomiast G i H mają rozkłady normalne o zerowej wartości oczekiwanej $N(0, \sigma_G)$ i $N(0, \sigma_H)$, przy czym $\sigma_H >> \sigma_G$. Pierwszy ze składników to rozkład o stosunkowo niewielkiej wariancji, mający reprezentować typowy biały szum, natomiast drugi z nich ma wariancję odpowiednio dużą, by symulować występowanie w danych błędów grubych. Pojawiają się one wśród zakłóceń z założonymi prawdopodobieństwami, odpowiednio $(1 - \delta)$ oraz δ. Ten typ błędów oznaczano dalej jako **Typ I**.

1.2.2 Błędy o niesymetrycznym rozkładzie

Bardziej nietypowym sposobem modelowania danych jest wprowadzenie rozkładu innego niż gaussowski, najlepiej niesymetrycznego, do sztucznie generowanych zakłóceń (błędy **Typu II**). W pracy [46] zaproponowano, aby miał on postać:

$$F = (1 - \delta)G + \delta(H_1 + H_2 + H_3 + H_4). \tag{1.2}$$

Podobnie, jak wcześniej, rozkład G jest po prostu białym szumem, natomiast, drugi z nich staje się bardziej skomplikowany, będąc złożeniem kilku dodatkowych rozkładów. Dla uproszczenia przyjęto, iż $H_i \sim N(m_i, \sigma_i)$, przy czym oba parametry

rozkładów są tu z założenia niezerowe i różne dla poszczególnych H_i. Prowadzi to do powstania rozkładu błędów niesymetrycznego (z wyjątkiem szczególnych przypadków, których się nie rozważa), z błędami o różnej amplitudzie.

1.2.3 Błędy w wektorze wejściowym

Wymienione wcześniej rodzaje błędów dotyczyły jedynie wyjścia systemu, a więc były to zakłócenia pojawiające się w wartościach, a nie argumentach modelowanej zależności. Błędy w wektorze wejściowym, czyli tzw. *leverage points* [52], wprowadzane były zgodnie z modelem opisanym jako *gross error model*, z tą różnicą, że zakłócenia dodawane były do wartości podawanych w procesie uczenia na wejście sieci (**Typ III**).

1.2.4 Szum tła

Kolejny sposób modelowania błędów (**Typ IV**), to zastąpienie części danych uczących szumem tła o rozkładzie jednostajnym. Symulować może to utratę części danych podczas, np. transmisji ich na odległość, choć przyznać trzeba, że sieci neuronowe w żaden sposób nie uodpornione radzą sobie w tym przypadku nienajgorzej, przez co należy rozumieć, że w większości przypadków zakłócenia takie mają mniejszy wpływ na proces uczenia, niż inne typy błędów.

1.3 Podsumowanie problemu

We wstępie do niniejszej pracy przedstawiono już pokrótce motywację podjęcia badań, oraz sformułowano ogólnie zadania do zrealizowania. Aby uściślić założenia, wykorzystując opisane w poprzedniej sekcji modele błędów, można jednak przedstawić je jeszcze raz.

Problem, który należało rozwiązać dotyczył błędów pojawiających się w danych uczących sieci neuronowych. Nie zajmowano się jednak ich pochodzeniem i analizą, lecz starano się opracować metody umożliwiające relatywnie dobre trenowanie sieci w ich obecności.

Zawartość dużych zakłóceń, w danych pochodzących z rzeczywistych obserwacji, sięgać może zwykle, jak już wspomniano, od kilku do kilkunastu procent. Sieci trenowane na danych zawierających błędy tworzą model niejednokrotnie daleki od zamierzonego dopasowując się do nieprawidłowych punktów z ciągu uczącego. W związku z tym, nietrudno zauważyć, że działanie nauczonej w takich warunkach sieci jest nieprzewidywalne, nawet jeśli potem stosuje się ją do działania na danych czystych.

Otóż, celem tej pracy było rozwiązanie problemu błędów grubych poprzez skonstruowanie algorytmów uczenia sieci neuronowych, które byłyby odporne na konkretne modele błędów (przedstawione powyżej typy I-IV). Odporność owa objawiać powinna się przez lepsze, niż w przypadku algorytmów klasycznych, działanie sieci uczonych odpornymi metodami, na danych zawierających zakłócenia generowane według jednego z owych modeli.

Opracowane algorytmy nie powinny wymagać od użytkownika informacji dotyczących typu możliwych zakłóceń i winny działać prawidłowo również dla danych nie zawierających żadnych błędów. Dopuszczono naturalnie możliwość, że poszczególne algorytmy będą predestynowane do uczenia sieci w konkretnych warunkach i żaden z nich nie będzie uniwersalny.

Rozdział 2

Podstawowe algorytmy uczenia sieci jednokierunkowych

2.1 Sztuczne sieci neuronowe

Sztuczne sieci neuronowe (SSN) będące początkowo próbą modelowania mechanizmów pozwalających na funkcjonowanie komórek nerwowych [44, 59], a w dalszej perspektywie mające umożliwić skonstruowanie sztucznej inteligencji podobnej w swej strukturze do ludzkiego mózgu, dość szybko przekształciły się w osobną dziedzinę wiedzy, której wiele gałęzi niewiele ma już wspólnego z owymi pierwotnymi założeniami.

Powstałe jako próba modelowania zjawisk zachodzących w ludzkim mózgu, sieci neuronowe ewoluowały w kierunku czysto algorytmicznym, ustanawiając nowy interdyscyplinarny obszar badań z pogranicza matematyki, informatyki, biologii i medycyny. Ów swoisty renesans tej gałęzi, wiedzy mający miejsce szczególnie w ostatnich latach, tłumaczyć można, po pierwsze, szybkim wzrostem mocy obliczeniowej komputerów osobistych, co umożliwiło symulowanie sieci na powszechnie dostępnych urządzeniach, oraz, po drugie, opracowaniem na przełomie lat siedemdziesiątych i osiemdziesiątych ubiegłego stulecia, algorytmu propagacji wstecznej, który stał się modelowym sposobem uczenia różnych typów sztucznych sieci neuronowych.

Nietrudno więc zrozumieć, że SSN znajdują coraz więcej rzeczywistych aplikacji w nauce i technice. Wraz z nowymi algorytmami uczenia, tworzone są ciągle nowsze i bardziej skomplikowane ich architektury, które, co prawda, coraz mniej wspólnego

mają ze swym biologicznym pierwowzorem, lecz z drugiej strony umożliwiają szybsze i dokładniejsze rozwiązywanie odpowiednich zadań. Oczywistym jest, że nadal jednak mózg ludzki pozostaje niedościgłym wzorem dla projektantów i teoretyków zajmujących się SSN.

O ile bowiem, prawie zawsze możliwy jest opis zachowania się pojedynczego neuronu, najmniejszego i podstawowego elementu budującego każdą sieć, gdyż jego konstrukcja jest zwykle zdumiewająco prosta i łatwa do analizy, o tyle zajmowanie się tworami składającymi się z dziesiątek, a często nawet setek owych prymitywnych procesorów, stwarzać może już poważne problemy, zwłaszcza, gdy zwróci się uwagę, że połączenia pomiędzy nimi są funkcją czasu i sygnałów wejściowych. Z tego powodu również dokładny schemat procesów zachodzących podczas czynności myślenia, czy zapamiętywania w ludzkim mózgu pozostaje nie do końca zbadany.

Ich rozwój podzielić można na kilka okresów [25]. Początkowe powodzenie dziedziny datowane jest od stworzenia pierwszego teoretycznego modelu neuronu McCullocha-Pittsa (1943) [38] i trochę późniejszej pracy Hebba [24]. Kolejna fala fascynacji, mająca miejsce w latach sześćdziesiątych ubiegłego wieku, związana jest głównie z pracami Rosenblatta [49], oraz Widrowa i Hoffa [70, 71]. Późniejszy spadek zainteresowania dziedziną spowodowany był w dużej mierze ograniczeniami perceptronu jednowarstwowego wykazanymi w książce Minsky'ego i Paperta [40]. Dopiero praca Hopfielda [26] z 1982 roku i późniejsze różnych badaczy, traktujące o algorytmie propagacji wstecznej, rozpoczęły ponowny wzrost zainteresowania SSN, który trwa do dnia dzisiejszego.

Obecnie jednak sieci neuronowe traktowane są raczej jako pewna grupa algorytmów, bądź też, ogólniej, podejść, sposobów postępowania[1] znajdujących zastosowania na najrozmaitszych polach, poczynając od aproksymacji, przez rozpoznawanie, przetwarzanie obrazu, identyfikację, sterowanie, po pamięci asocjacyjne, etc. Umiejscowić więc można je obok dziedzin takich jak uczenie maszynowe (*machine learning*), uczenie statystyczne (*statistical learning*), systemy ekspertowe. Oczywiście nadal trwają badania mające na celu zbliżenie się do biologicznych podstaw działania mózgu, lecz należy je uznać za sytuujące się dziś na peryferiach dziedziny.

[1] Naturalnie, stosunkowo często podejmowane są próby fizycznych implementacji SSN, nie zmienia to jednak faktu, że w większości przypadków mamy do czynienia z symulacjami czysto algorytmicznymi.

Można pokusić się o stwierdzenie, że różne typy sztucznych sieci neuronowych łączy głównie właśnie ich sieciowa struktura, w której najczęściej wyróżnić można dwa rodzaje elementów: neurony (jednostki obliczeniowe, najczęściej wykonujące jakieś przekształcenie funkcyjne swoich wejść), oraz połączenie pomiędzy nimi (wagi, posiadające liczbowe wartości). Szczegółowa topologia, algorytmy uczące, realizowane zadania mogą się już bardzo różnić, w zależności od tego z jakim rodzajem SSN mamy do czynienia.

Aby przytoczyć tu jedynie najważniejsze typy SSN, warto wymienić sieci jednokierunkowe wielowarstwowe osobno o sigmoidalnych i radialnych funkcjach bazowych, sieci rekurencyjne, w tym sieci Hopfielda [33], Hamminga [35], Elmana [13], oraz sieci samoorganizujące się, takie jak sieci Hebba [24], czy Kohonena [31, 62]. Istnieje oczywiście wiele ich modyfikacji, ponadto w zależności od użytego algorytmu uczenia uzyskać można struktury mające nieco odmienne własności i nadające się do różnych zastosowań, widać więc wyraźnie istniejące już jak i potencjalne bogactwo SSN.

2.1.1 Sieci jednokierunkowe sigmoidalne

Wielowarstwowe sieci jednokierunkowe o sigmoidalnych funkcjach aktywacji neuronów, znane czasem pod nazwą perceptronów wielowarstwowych, stanowią najlepiej opisaną i zarazem chyba najczęściej używaną w praktyce grupę struktur SSN [2]. W sieciach tych sygnał propagowany jest, jak sama nazwa wskazuje, tylko w jednym kierunku, od wejść do wyjść sieci, przechodząc przez jedną lub kilka warstw neuronów. Z perceptronami wielowarstwowymi łączy się bezpośrednio filozofia uczenia zwana wsteczną propagacją błędu (*backpropagation*), umożliwiająca stosunkowo proste obliczenie gradientu funkcji celu względem wszystkich wag sieci. Pozwala to na zastosowanie wielu różnych gradientowych algorytmów optymalizacji, co sprowadza proces uczenia sieci jednokierunkowej do minimalizacji funkcji wielu zmiennych.

Jak już wspomniano, elementarnym składnikiem każdej sztucznej sieci neuronowej jest pojedyncza, najczęściej nieskomplikowana jednostka obliczeniowa, zwana neuronem [59, 64]. Jej zadaniem jest wygenerowanie pewnego sygnału wyjściowego zależnego od sygnałów dochodzących do jej wejść, oraz pewnej funkcji, do której wykonywania owa jednostka ma służyć.

Podobnie podstawowy element budujący sieć jednokierunkową, pojedynczy neuron, składa się z wejść przemnożonych wstępnie przez wartości wag i po zsumowaniu podanych do bloku funkcji bazowej. Dodatkowo często stosuje się nadmiarowe wejście o stałej wartości (np. +1 lub -1) zwane przesunięciem lub progiem (*bias*). Funkcja bazowa ma zwykle kształt sigmoidalny, będący ciągłym przybliżeniem stosowanej w pierwszych modelach neuronu funkcji skokowej, aczkolwiek nierzadko przyjmuje się w neuronach warstwy wyjściowej również funkcję liniową. Sygnał wyjściowy neuronu można więc zapisać jako:

$$y = f(\sum_{i=1}^{n} w_i x_i + w_0), \tag{2.1}$$

gdzie w_i oznacza wagę łączącą wejście x_i z sumatorem, w_0 jest progiem określającym stałe przesunięcie funkcji, natomiast $f(.)$ funkcją bazową. Jak już wspomniano, funkcja bazowa powinna zachowywać swój skokowy charakter przy równoczesnym zapewnieniu ciągłości i różniczkowalności. Warunki te spełniają, stosowane najczęściej funkcje logistyczna i oparta na tangensie hiperbolicznym, dane następującymi zależnościami:

$$f_1(x) = \frac{1}{1 + \exp(-\beta x)}, \tag{2.2}$$

oraz

$$f_2(x) = \mathrm{tgh}(\beta x). \tag{2.3}$$

Przyjmują one wartości z przedziałów kolejno [0,1] lub [-1,1], przy czym możliwe jest również skonstruowanie, na podstawie pierwszej z nich, używanej czasem funkcji o przeciwdziedzinie [-1,1] postaci:

$$f_3 = 2f_1(x) - 1. \tag{2.4}$$

W przypadku sieci jednokierunkowych proces uczenia polega na adaptacyjnym dopasowywaniu wartości parametrów (wag) sieci w celu zminimalizowania pewnego zadanego kryterium. Sprowadza się to do problemu optymalizacji funkcji w przestrzeni o liczbie wymiarów równej ilości wag sieci. Bardziej szczegółowy opis metod uczenia opartych na wstecznej propagacji błędu pojawi się w dalszej części pracy.

Sieć jednokierunkowa wielowarstwowa składać się może potencjalnie z dowolnej liczby warstw neuronów, przy czym istotne jest, aby pojawiły się wśród nich warstwa wejściowa, wyjściowa, oraz co najmniej jedna warstwa zwana ukrytą. Neurony

wejściowe nie otrzymują sygnału od innych neuronów sieci, a ich jedynym zadaniem jest przekazanie sygnałów zewnętrznych, bez dodatkowego ich przetwarzania, do warstwy ukrytej. Neurony wyjściowe z kolei, określane często po prostu wyjściem sieci, choć najczęściej wykonują na dostarczanych im sygnałach pewne przekształcenia, odpowiadają za dostarczenie wyniku działania struktury do jej otoczenia. Warstwami ukrytymi nazywane są natomiast warstwy neuronów, do których nie ma bezpośredniego dostępu od strony wejścia, ani wyjścia sieci, a więc których sygnały nie są jawnie dane. W praktyce używane są zwykle sieci o jednej, dwóch warstwach ukrytych. Jest to prostą konsekwencją faktu, że taka ilość warstw ukrytych wystarcza do rozwiązania praktycznie każdego zadania stawianego tego rodzaju sieciom [27].

2.1.2 Sieci o bazach radialnych

Teoria sieci radialnych [8] odwołuje się do twierdzenia Covera dotyczącego separowalności wzorców przy nieliniowym rzutowaniu problemu w przestrzeń o większej liczbie wymiarów. O ile w przypadku sieci jednokierunkowych o sigmoidalnych, bądź skokowych funkcjach aktywacji, działanie pojedynczych neuronów utożsamiać można z hiperpłaszczyznami dzielącymi wielowymiarową przestrzeń, każda na dwie części, o tyle w przypadku sieci o bazach radialnych, mówić można raczej o rozciągnięciu nad zbiorem danych uczących hiperpłaszczyzny dopasowanej do poszczególnych punktów danych. W praktyce, najczęściej stosuje się sieci oparte na funkcjach radialnych zmieniających się wokół pewnego centrum.

Oczywiście aproksymacja uzyskana poprzez superpozycję funkcji radialnych odpowiadających wszystkim punktom danych uczących byłaby zadaniem skomplikowanym i złożonym obliczeniowo, w związku z tym zwykle rozwiązanie problemu jest jedynie przybliżone z dostatecznie dobrą dokładnością. Tworzy się więc sieć neuronową, której wyjście określić można [8, 44] (w przypadku jednego jednowymiarowego wyjścia) jako:

$$y = \sum_{i=1}^{K} w_i \varphi_i(r), \tag{2.5}$$

gdzie $r = ||x - t||$. Ma ona jedną warstwę ukrytą z neuronami radialnymi i warstwę wyjściową z neuronem liniowym. Dodatkowo pojawić się może również odpowiadająca za przesunięcie waga zerowa. Norma obliczana w powyższej zależności może

być zarówno normą euklidesową, jak i normą zawierającą macierz wag umożliwiających zniwelowanie efektu różnej zmienności funkcji w poszczególnych kierunkach. Architektura sieci jednokierunkowych o bazach radialnych jest, w przeciwieństwie do przypadku sieci sigmoidalnych, ustalona i zawsze składa się na nią zaledwie jedna warstwa ukryta realizująca funkcje radialne, oraz liniowy neuron wyjściowy sumujący sygnały pochodzące z neuronów warstwy ukrytej. Jako baza radialna, najczęściej stosowana jest gaussowska funkcja bazowa dana zależnością:

$$\varphi(r) = \exp(-\frac{r^2}{2\sigma^2}), \tag{2.6}$$

gdzie $\sigma > 0$ jest dobieranym parametrem. Z innych funkcji radialnych wymienić można następujące: $\varphi(r) = \sqrt{r^2 + \sigma^2}$ i $\varphi(r) = \frac{1}{\sqrt{r^2+\sigma^2}}$.

Sieci o bazach radialnych uczone mogą być różnymi metodami, w tym opartymi na propagacji wstecznej, podobnie jak sieci sigmoidalne. Jako, że algorytmy uczenia takich sieci [44, 23, 2] nie były wykorzystywane w niniejszej pracy, dlatego nie zostaną dalej zaprezentowane [2].

2.2 Algorytmy uczenia sieci jednokierunkowych

Metody uczenia sieci neuronowych, nie tylko jednokierunkowych, podzielić można na metody uczenia nadzorowanego (*supervised*) i bez nadzoru (*unsupervised*). W tej pierwszej grupie, zwanej również grupą metod uczenia z nauczycielem, dobór parametrów sieci dokonywany jest na podstawie porównania wartości na wyjściu sieci z zadanymi wartościami dla poszczególnych obrazów uczących. Tworzy się więc pewną funkcję kryterialną zawierającą błędy popełniane przez sieć, a następnie próbuje się znaleźć jej minimum. Z kolei w przypadku uczenia nienadzorowanego parametry sieci dobierane są tak, aby wykryć zależności w danych uczących, które mogą zostać w oparciu o nie poklasyfikowane.

2.2.1 Metoda wstecznej propagacji błędu

Większość z istniejących metod uczenia sieci jednokierunkowych opiera się na algorytmie propagacji wstecznej (*backpropagation*) [54], stanowiącym klasykę omawianej

[2]Trzeba jednak zwrócić uwagę, że część z opracowanych metod można wykorzystać również do uczenia sieci o bazach radialnych uczonych sposobem wstecznej propagacji błędu

dziedziny. Umożliwił on nieskomplikowane rozwiązanie problemu uczenia sieci wielowarstwowej, który przez wiele lat traktowany był jako główna przeszkoda w rozwoju teorii i zastosowań sztucznych sieci neuronowych.

W metodzie wstecznej propagacji błędu przyjęto następujące założenia [44] dotyczące zadania uczenia: sieć jest ściśle wielowarstwowa (istnieją jedynie połączenia pomiędzy neuronami kolejnych warstw), funkcje aktywacji sieci, oraz funkcja celu są ciągłe i różniczkowalne, dane uczące składają się z wartości podawanych na wejście sieci i odpowiadających im pożądanych wartości wyjść. W takiej sytuacji postępowanie w metodzie propagacji wstecznej przedstawia się następująco [23, 25]:

1. Sieć analizowana jest w kierunku od wejścia do wyjścia. Przy ustalonych sygnałach wejściowych obliczane są wartości wyjść neuronów poszczególnych warstw, oraz pochodne cząstkowe wszystkich funkcji aktywacji.

2. Sieć analizowana jest w kierunku odwrotnym do zwykłego przepływu sygnałów. Funkcje aktywacji zastępowane są przez swoje pochodne, do wyjścia sieci podawane jest wymuszenie w postaci różnicy między obliczonym wcześniej wyjściem, a wartością zadaną. W tak zdefiniowanym stanie, od warstwy ostatniej do pierwszej propagowany jest błąd (różnice wsteczne) obliczany dla wszystkich neuronów kolejnych warstw.

3. Następuje aktualizacja wszystkich wag sieci na podstawie zadanej reguły uczenia i wyników uzyskanych w poprzednich krokach algorytmu. Jeżeli nie został spełniony warunek stopu, powraca się do punktu pierwszego.

Jak już wspomniano, postępowanie mające prowadzić do nauczenia sieci neuronowej metodą gradientową ma na celu minimalizację pewnej, zależnej od parametrów sieci, funkcji celu. W każdym kroku (epoce) algorytmu uczenia uaktualnienie wektora wag odbywa się poprzez dodanie do niego pewnego wyznaczonego odpowiednio wektora $\Delta\vec{w}$ w sposób następujący:

$$\vec{w}(t+1) = \vec{w}(t) + \Delta\vec{w}, \tag{2.7}$$

gdzie t oznacza numer epoki procesu uczenia. Przyrost wag obliczany jest jako:

$$\Delta\vec{w} = \eta p_t, \tag{2.8}$$

przy czym η jest współczynnikiem uczenia, natomiast p wyznaczonym kierunkiem.

2.2.2 Algorytmy gradientowe

Algorytm optymalizacji bazuje zwykle na rozwinięciu funkcji celu w szereg Taylora w pewnym otoczeniu punktu początkowego, w obliczonym wcześniej kierunku p. Aproksymowana wartość przedstawia się wtedy następująco:

$$E(\vec{w}+p) = E(\vec{w}) + [\nabla E(\vec{w})]^T p + 1/2 p^T H(\vec{w}) p + \dots . \tag{2.9}$$

W zależności tej występują kolejne pochodne funkcji celu, a więc jej gradient:

$$\nabla E(\vec{w}) = \left[\begin{array}{cccc} \frac{\partial E}{\partial w_1} & \frac{\partial E}{\partial w_2} & \dots & \frac{\partial E}{\partial w_n} \end{array} \right], \tag{2.10}$$

gdzie n oznacza liczbę wag sieci, Macierz drugich pochodnych, czyli hesjan:

$$H(\vec{w}) = \left[\begin{array}{cccc} \frac{\partial E}{\partial w_1 \partial w_1} & \frac{\partial E}{\partial w_2 \partial w_1} & \dots & \frac{\partial E}{\partial w_n \partial w_1} \\ \frac{\partial E}{\partial w_1 \partial w_2} & \frac{\partial E}{\partial w_2 \partial w_2} & \dots & \frac{\partial E}{\partial w_n \partial w_2} \\ \dots & \dots & \dots & \dots \\ \frac{\partial E}{\partial w_1 \partial w_n} & \frac{\partial E}{\partial w_2 \partial w_n} & \dots & \frac{\partial E}{\partial w_n \partial w_n} \end{array} \right], \tag{2.11}$$

oraz ewentualnie człony zależne od wyższych pochodnych funkcji celu, które, z oczywistych względów nie są zwykle wykorzystywane w praktyce. Znając analityczny warunek mówiący, że punkt $\vec{w^*}$ jest punktem optymalnym, jeżeli gradient funkcji zeruje się, zaś hesjan jest w nim dodatnio określony można projektować procedury optymalizacyjne służące uczeniu sieci. Algorytmy uczące podzielić więc można również ze względu na rząd wykorzystywanej pochodnej funkcji celu [64], co w rzeczywistości implikuje podział na algorytmy pierwszego i drugiego rzędu. W szczególnym przypadku, jeśli rozważymy tylko trzy pierwsze składniki szeregu, otrzymujemy:

$$E(\vec{w}+p) = E(\vec{w}) + [\nabla E(\vec{w})]^T p + 1/2 p^T H(\vec{w}) p + R, \tag{2.12}$$

gdzie R jest resztą wynikającą z zaniedbania dalszych składników rozwinięcia. Z warunku optymalności można wtedy obliczyć krok jako:

$$p = -H^{-1}(\vec{w}) \nabla E(\vec{w}). \tag{2.13}$$

W przypadku najprostszym, gdy wykorzystywana jest jedynie informacja o gradiencie funkcji, a rozwinięcie w szereg ogranicza się do dwóch elementów, wybór kierunku

optymalizacji dokonywany jest metodą największego spadku w kierunku malejącego gradientu jako:

$$p = -\nabla E(\vec{w}). \tag{2.14}$$

Pomimo swej prostoty cierpi ona na wiele niedoskonałości, takich jak wolna zbieżność i tendencja do utykania w minimach lokalnych [17]. W dalszej części rozdziału omówione zostaną bardziej szczegółowo algorytm Levenberga-Marquardta [20] i algorytm gradientów sprzężonych [6], jako, że one same, jak i ich modyfikacje były wykorzystywane w niniejszej pracy, natomiast należy w tym miejscu wspomnieć, że w literaturze dotyczącej sieci neuronowych napotkać można wiele innych metod i wariantów algorytmów uczenia. Wymienić tu można szczególnie algorytmy drugiego rzędu, takie jak metoda gradientów sprzężonych z regularyzacją [41], *one-step secant* [3], pseudo-newtonowska metoda BPQ [60], algorytm quickprop [14], RBROP [48], czy nawet bezpośrednio stosowane podejście newtonowskie [45] i wiele innych.

2.2.3 Algorytm Levenberga-Marquardta

Algorytm Levenberga-Marquardta (L-M) [20] jest obecnie jednym z najczęściej stosowanych do uczenia sieci neuronowych jednokierunkowych, algorytmów optymalizacyjnych drugiego rzędu. Wynika to głównie z szybkiej zbieżności, niezbyt wielkiej złożoności obliczeniowej, oraz prostej implementacji. Opiera się on na algorytmie rozwiązywania nieliniowego problemu najmniejszych kwadratów przedstawionym przez Marquardt'a w pracy [37]. Metoda regularyzacji L-M polega, w skrócie rzecz ujmując, na zastąpieniu macierzy hesjanu, w optymalizacji newtonowskiej, jej przybliżeniem opartym na obliczeniach gradientu, wraz z odpowiednio dobranym czynnikiem regularyzacyjnym. Jako, że jest to konieczne ze względu na wykorzystanie jej elementów w dalszej częście tej pracy, zostanie ona tu pokrótce przedstawiona.

W algorytmie L-M zakłada się kwadratową postać funkcji błędu daną dla N obrazów uczących zależnością:

$$E(\vec{w}) = \sum_{k=1}^{N} \sum_{i=1}^{m} [r_{ki}(\vec{w})]^2, \tag{2.15}$$

przy czym $r_{ki} = (y_{ki}(\vec{w}) - t_{ki})$ oznacza błąd popełniany przez i-te wyjście sieci dla k-tego obrazu uczącego, a m jest liczbą wyjść. Zapisując jakobian jako:

$$J(\vec{w}) = \begin{bmatrix} \frac{\partial r_{11}}{\partial w_1} & \frac{\partial r_{11}}{\partial w_2} & \dots & \frac{\partial r_{11}}{\partial w_n} \\ \frac{\partial r_{21}}{\partial w_1} & \frac{\partial r_{21}}{\partial w_2} & \dots & \frac{\partial r_{21}}{\partial w_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial r_{N1}}{\partial w_1} & \frac{\partial r_{N1}}{\partial w_2} & \dots & \frac{\partial r_{N1}}{\partial w_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial r_{1m}}{\partial w_1} & \frac{\partial r_{1m}}{\partial w_2} & \dots & \frac{\partial r_{1m}}{\partial w_n} \\ \frac{\partial r_{2m}}{\partial w_1} & \frac{\partial r_{2m}}{\partial w_2} & \dots & \frac{\partial r_{2m}}{\partial w_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial r_{Nm}}{\partial w_1} & \frac{\partial r_{Nm}}{\partial w_2} & \dots & \frac{\partial r_{Nm}}{\partial w_n} \end{bmatrix}, \tag{2.16}$$

gdzie

$$r(\vec{w}) = \begin{bmatrix} r_{11} & \dots & r_{N1} & r_{12} & \dots & r_{N2} & \dots & r_{1m} & \dots & r_{Nm} \end{bmatrix}^T, \tag{2.17}$$

a n jest ilością wag w sieci, możemy na jego podstawie napisać zależność opisującą gradient, oraz aproksymującą hesjan. Jak można zauważyć, dla przyjętej funkcji błędu gradient wyraża się za pomocą macierzy jakobianu jako:

$$\nabla E(\vec{w}) = J(\vec{w})^T r(\vec{w}). \tag{2.18}$$

Przyjęte w algorytmie L-M przybliżenie macierzy hesjanu to z kolei:

$$H(\vec{w}) = J(\vec{w})^T J(\vec{w}) + u1, \tag{2.19}$$

gdzie u jest czynnikiem regularyzacyjnym.

Przyrost wag odbywa się tu analogicznie jak w innych metodach opierających się na kwadratowym rozwinięciu funkcji celu w pobliżu rozwiązania, z tą różnicą, że zamiast dokładnej wartości hesjanu używane jest powyższe oszacowanie. Stąd wektor wag w epoce $t+1$ to:

$$\vec{w}_{t+1} = \vec{w}_t - \left[J(\vec{w}_t)^T J(\vec{w}_t) + u_t 1 \right]^{-1} \nabla E(\vec{w}_t) \tag{2.20}$$

Czynnik regularyzacyjny u_t jest tu funkcją postępu procesu uczenia i dobierany jest w różnych implementacjach algorytmu w rozmaity sposób. Z założenia, powinien on na początku przyjmować wartość bardzo dużą w porównaniu z wartościami

własnymi macierzy $J(\vec{w})^T J(\vec{w})$, by w miarę zmniejszania się błędów dążyć powoli do zera. Powoduje to, że początkowo algorytm przeprowadza minimalizację metodą największego spadku, natomiast, w pobliżu rozwiązania hesjan aproksymowany jest jedynie za pomocą rozwinięcia pierwszego rzędu, co prowadzi do przejścia do algorytmu Gaussa-Newtona o kwadratowej zbieżności, przy niezmienionym nakładzie obliczeniowym.

Najprostszy sposób modyfikacji czynnika u_t przedstawia się następująco: w konkretnej epoce jest on zwiększany według schematu ηu_t aż do momentu uzyskania poprawy wartości funkcji celu, następnie, po wykonaniu kroku uczenia, jest on zmniejszany jak βu_t w głównym algorytmie. Konieczne jest tu dobranie właściwych wartości stałych η i β odpowiadających za zmianę u_t w taki sposób, aby nie zakłócić działania całej metody, to znaczy, aby czynnik ulegał zmniejszeniu w miarę postępu procesu uczenia.

Istnieją również ulepszenia tej metody, jak zmodyfikowana wersja algorytmu L-M [72], w której uzyskuje się dzięki innemu indeksowaniu macierzy, szybszą zbieżność i mniejsze wymagania dotyczące pamięci.

2.2.4 Algorytm gradientów sprzężonych

W algorytmie gradientów sprzężonych [6, 19] rezygnuje się z bezpośredniego wykorzystywania macierzy hesjanu do konstrukcji nowego kierunku poszukiwań minimum. Opiera się on na założeniu, że aby zapewnić właściwie przeszukiwanie dziedziny funkcji, kierunek poszukiwania powinien tworzony być za każdym razem nie zgodnie z malejącym gradientem, lecz tak, by był sprzężony z poprzednią wartością gradientu i o ile to możliwe, z poprzednimi kierunkami [16]. Jest to mało praktyczne, gdyż wektor taki musiałby być tworzony na podstawie wszystkich poprzednich kierunków poszukiwań i miałby postać:

$$p_t = -\nabla E(\vec{w_t}) + \sum_{j=0}^{t-1} \beta_t p_j, \tag{2.21}$$

dla t oznaczającego aktualną epokę uczenia.

W metodzie gradientów sprzężonych, zależność powyższą upraszcza się, uwzględniając warunek ortogonalności, do postaci:

$$p_t = -\nabla E(\vec{w_t}) + \beta_t p_{t-1}. \tag{2.22}$$

Widać więc, że nowy kierunek zależy od gradientu funkcji, oraz poprzedniego kierunku pomnożonego przez współczynnik sprzężenia β. Istnieje wiele sposobów doboru owego współczynnika [41], wykorzystujących gradient, bądź kierunek poszukiwań z poprzedniej iteracji, z których najpopularniejsze to metoda Flethera-Reevesa [15], oraz wykorzystywany w pracy niniejszej sposób określania β wg Polaka-Ribiere'a [65, 47]:

$$\beta = \frac{[\nabla E(\vec{w_k})]^T(\nabla E(\vec{w_k}) - \nabla E(\vec{w_{k-1}}))}{[\nabla E(\vec{w_{k-1}})]^T \nabla E(\vec{w_{k-1}})}. \tag{2.23}$$

Metoda ta wymaga ponownego startu po przekroczeniu pewnej liczby epok, ze względu na mogące pojawić się błędy zaokrągleń, a w rezultacie zatracanie właściwości ortogonalności pomiędzy wektorami kierunków[44]. Pomimo tej niedogodności, algorytm gradientów sprzężonych jest bardzo często stosowany z uwagi na stosunkowo szybką zbieżność (zbliżoną do liniowej), jak i niewielkie wymagania dotyczące pamięci. Dla zadań optymalizacyjnych w przestrzeni o bardzo dużej liczbie wymiarów (a do takich przecież należy problem uczenia sieci neuronowych), algorytm ten jest jednym z najefektywniejszych.

Rozdział 3

Odporne algorytmy uczenia sieci neuronowych

3.1 Wprowadzenie do odpornych algorytmów uczenia

Jak już wspomniano, algorytmy uczenia sieci neuronowych, podobnie, jak wszystkie inne metody opierające się przy budowaniu modelu na pewnym zestawie danych uczących, są wrażliwe na błędy mogące się w nim pojawić. Teza ta wydaje się dość oczywista, a wiele ilustrujących ją przykładów znaleźć można w rozdziale poświęconym wynikom eksperymentów numerycznych (rozdział 6).

Jak zostanie dalej pokazane, algorytm uczenia sieci zakładający minimalizację błędu średniokwadratowego, z założenia działać powinien prawidłowo jedynie dla danych czystych, albo zakłóconych niewielkim białym szumem. W przypadku, kiedy w obrazach uczących pojawiają się duże zakłócenia, działania nauczonej na nich sieci staje się w praktyce nieprzewidywalne, nawet, gdy testować ją będziemy na zupełnie czystym zbiorze danych.

Tematyka związana z uodparnianiem procesu uczenia sieci neuronowych na błędy w zbiorze uczącym nie jest zagadnieniem, które doczekało się wielu propozycji rozwiązania. W literaturze znaleźć można zaledwie kilka publikacji, które proponują modyfikacje uczenia jednokierunkowych sieci sigmoidalnych (z pewnym jednak wyjątkiem) mające na celu zwiększenie ich odporności na błędy w danych. Wszystkie opierają się na wprowadzeniu zmodyfikowanej (różnej od kwadratowej) funkcji błędu, przy zachowaniu wszelkich innych cech algorytmu uczenia. Użycie odpowiednio

skonstruowanego kryterium powoduje praktyczne wykluczenie z wpływu na proces uczenia danych powodujących największe błędy, a więc podejrzanych o bycie błędami grubymi. W skrócie rzecz ujmując, proponowane w literaturze odporne algorytmy uczenia sieci sigmoidalnych są prostym rozszerzeniem algorytmu największego spadku, w którym główny nacisk położono na dobór właściwej funkcji kryterialnej. Należy zaznaczyć, że wszystkie istniejące, opisane poniżej, algorytmy przeznaczone zostały do uczenia typu "on-line". Przy ich konstruowaniu, autorzy zakładali, że błędy modelować można za pomocą sumy rozkładów normalnych [7], w sposób dokładnie przedstawiony w przedostatnim rozdziale niniejszej pracy (sekcja 1.2).

Powstaje pytanie, dlaczego opisane dalej algorytmy dotyczą właśnie sieci jednokierunkowych. Po pierwsze, są one chyba najlepiej opisaną klasą sieci: istnieje wiele różnorakich algorytmów ich uczenia, oraz udoskonaleń pomagających w szczególnych warunkach. Ponadto, są z historycznego punktu widzenia, najstarsze i prawdopodobnie nadal najbardziej popularne, a więc najczęściej stosowane. Wyróżnia je dodatkowo uniwersalność zastosowań i prostota użycia. Ponadto można znaleźć pewne związki pomiędzy odpornymi estymatorami a sieciami jednokierunkowymi.

Dlaczego jednak sieci sigmoidalne? Tu odpowiedź wydaje się już prostsza. Są one bowiem z natury swej bardziej odporne, niż sieci o bazach radialnych, jak wykazano w [36]. Łatwo to sobie uświadomić pamiętając, że podczas, gdy w sieciach sigmoidalnych dokonuje się podziału przestrzeni pewnymi hiperpłaszczyznami, w sieciach RBF mamy do czynienia z rozciąganiem hiperpłaszczyzn wokół punktów z ciągu uczącego, tak więc, niejako automatycznie wpływ błędów grubych jest większy.

Wspomniany wyżej wyjątek w odpornych algorytmach to krótka praca (w formie *letter*) [11], w której podjęto próbę uodpornienia prostej sieci RBF w sposób identyczny, jak we wcześniejszym, chronologicznie rzecz ujmując, odpornym algorytmie RBP (*robust backpropagation*) [7] przeznaczonym do uczenia sieci o sigmoidalnych funkcjach aktywacji neuronów. Nie jest ona dalej dokładnie opisana, dlatego, że sposób postępowania autora jest identyczny, jak w przypadku metody RBP. Ponadto w pracy [10] powtórzono rozumowanie z [9] proponując algorytm analogiczny do opisanego dalej ARBP (*annealing robust backpropagation*), lecz przeznaczony do

uczenia sieci z radialnymi funkcjami bazowymi. W obu cytowanych pozycjach autorzy nie zajmowali się porównaniem swoich metod z analogicznymi algorytmami uczenia sieci sigmoidalnych, choć powinny one działać gorzej niż ich pierwowzory.

Warto zwrócić uwagę, że omówione dalej odporne algorytmy uczenia, jak i metody opracowane w ramach tej pracy, skupiły się na reprezentowaniu przez sieć zależności z ciągłymi, albo mogącymi przybierać wiele wartości, wyjściami systemu. W związku z tym, uodpornienie dotyczy tu przede wszystkim zadań takich, jak aproksymacja funkcji, predykcja sygnałów, itp. Dla problemów klasyfikacji, szczególnie przy niewielkiej liczbie klas, podejście do uodparniania byłoby zapewne inne [18].

3.2 Odporne algorytmy uczenia

3.2.1 Odporność na zakłócenia a dobór funkcji błędu

Uczenie nadzorowane sieci neuronowej polega na minimalizacji pewnej funkcji błędu zależnej od różnic pomiędzy wartościami zadanymi, a rzeczywistymi wyjść sieci dla danych uczących. W poniższych rozważaniach przyjęto, dla jasności zapisu, że sieć posiada tylko jedno wyjście. W najbardziej ogólnym przypadku funkcja błędów może być również zależna od parametrów sieci [23], jednak najczęściej (również w badanych algorytmach) ma ona następującą postać:

$$E = \frac{1}{N} \sum_{i=1}^{N} \rho\left(r_i\right), \tag{3.1}$$

gdzie $\rho(r_i)$ oznacza funkcję kary (*loss function*) [21], zwykle ciągłą i symetryczną, r_i jest błędem dla i-tego obrazu uczącego, natomiast N liczbą elementów ciągu uczącego. W najbardziej typowym przypadku jako funkcję strat przyjmuje się funkcję kwadratową:

$$\rho(r_i) = {r_i}^2. \tag{3.2}$$

Wtedy minimalizowane kryterium ma postać:

$$E = \frac{1}{N} \sum_{i=1}^{N} {r_i}^2, \tag{3.3}$$

czyli jest równoznaczne z błędem średniokwadratowym.

Przyjęcie funkcji kwadratowej, jako minimalizowanego kryterium błędów ma swoje uzasadnienie, które pokazać można korzystając z metody największej wiarogodności [9, 28]. Dla tego celu załóżmy, że błędy pomiarowe r_i są niezależne i generowane według pewnego rozkładu $f(r_i)$. Możemy więc, dla ciągłego rozkładu, obliczyć prawdopodobieństwo modelu, jako produkt poszczególnych prawdopodobieństw:

$$L = \prod_{i=1}^{N} f(r_i)\Delta, \tag{3.4}$$

przy czym $f(r_i)\Delta$ oznacza prawdopodobieństwo r_i w sąsiedztwie Δ dla ciągłej dystrybuanty. Maksymalizacja prawdopodobieństwa może zostać zapisana jako maksymalizacja jego logarytmu, lub alternatywnie, minimalizacja:

$$\min_{\vec{w}} \sum_{i=1}^{N} (-\log f(r_i)) - N \log \Delta. \tag{3.5}$$

Minimalizacja odbywa się tu po wektorze parametrów $\vec{w}$, który w tym konkretnym przypadku może być traktowany jako wektor parametrów (wag) sieci. Ponieważ N oraz Δ to stałe, otrzymujemy:

$$\min_{\vec{w}} \sum_{i=1}^{N} (-\log f(r_i)). \tag{3.6}$$

Przy typowym założeniu, mówiącym, że błędy generowane są według rozkładu normalnego danego jako:

$$f(r) = \frac{1}{\sqrt{2\pi}} \exp(-1/2r^2), \tag{3.7}$$

po podstawieniu $f(r)$ do równania (3.6) otrzymujemy:

$$\min_{\vec{w}} \left(\sum_{i=1}^{N} \frac{1}{2} {r_i}^2 + \frac{N}{2} \log(2\pi) \right) \tag{3.8}$$

Zaniedbując stały człon, dostajemy w rezultacie właśnie minimalizację błędu kwadratowego:

$$\min_{\vec{w}} \sum_{i=1}^{N} r_i^2. \tag{3.9}$$

Miarą oddziaływania błędów na proces uczenia jest tzw. funkcja wpływu (*influence function*) [21], będąca pochodną stosowanej funkcji kary/strat:

$$\psi(r_i) = \frac{\partial \rho(r_i)}{\partial r_i}. \tag{3.10}$$

Można ją traktować jako szczególny przypadek (a tak naprawdę rozszerzenie na teorię uczenia sieci) tak samo nazwanej funkcji używanej często w literaturze statystycznej [21, 28], w szczególności w rozważaniach dotyczących odpornych estymatorów.

Gdy policzyć funkcję wpływu dla kryterium kwadratowego, ma ona postać liniową:

$$\psi(r_i) = r_i. \tag{3.11}$$

Na tej podstawie można zauważyć, że błędy o największych wartościach mają największy wpływ na uczenie sieci z tak dobraną funkcją kryterialną. Oznacza to, że każdy pojawiający się w danych uczących błąd gruby będzie skutkował stosunkowo dużą zmianą wag sieci w procesie uczenia.

Jak już wspomniano, wszystkie wymienione, znane z literatury odporne algorytmy uczenia sieci neuronowych, wykorzystują do osiągnięcia odporności zmodyfikowane postacie funkcji celu. W skrócie więc, można zapisać procedurę postępowania w każdym z nich w podobny sposób. Przedstawia się ona następująco:

1. Wylosuj początkowe wartości parametrów sieci.
2. Oblicz wartość nowej funkcji kryterialnej.
3. Wykonaj krok uczenia poprzez odpowiednią modyfikację wag.
4. Jeśli spełniony jest jeden z warunków stopu, zatrzymaj uczenie, jeśli nie, wróć do punktu drugiego.

W tak ogólnym zapisie chodzi jedynie o to, aby podkreślić, że zamiast liczenia typowych wartości błędów, czyli błędu średniokwadratowego, należy przeprowadzać proces minimalizacji dla innej, danej dla konkretnego algorytmu, funkcji.

3.2.2 Odporny algorytm z kryterium LMLS

Jedną z możliwości niwelowania wpływu błędnych danych na proces uczenia jest zastosowanie odpowiednio ograniczonej funkcji wpływu. W pracy Liano [34] do skonstruowania odpornego kryterium błędu w algorytmie tym zaproponowano funkcję kary nazwaną LMLS (*Least Mean Log Squares*) następującej postaci:

$$\rho(r_i) = \log(1 + \frac{1}{2}{r_i}^2) \tag{3.12}$$

Wynika to stąd, że w przypadku przyjęcia ρ jak w powyższej zależności, funkcja wpływu jest ograniczona i można ją zapisać jako:

$$\psi(r_i) = \frac{r_i}{1 + \frac{1}{2}{r_i}^2}. \tag{3.13}$$

Dla niewielkich wartości residuów jest ona praktycznie rzecz biorąc liniowa, podczas, gdy wraz z ich wzrostem zaczyna asymptotycznie dążyć do zera. W rezultacie:

$$\lim_{r->\infty} \frac{r_i}{1 + \frac{1}{2}{r_i}^2} = 0, \tag{3.14}$$

jak więc widać największe błędy nie mają praktycznego wpływu na wartość funkcji celu, a więc i na proces uczenia.

Podobnie, jak kryterium najmniejszych kwadratów, również LMLS można wyprowadzić za pomocą metody największej wiarogodności, przy pewnym istotnym założeniu dotyczącym rozkładu błędów. Mianowicie, aby uwzględnić obecność w danych punktów odstających, założyć można, iż rozkład błędów jest rozkładem Cauchy'ego, czyli:

$$f(r) = \frac{1}{\pi(1 + r^2)}. \tag{3.15}$$

Dla tak zdefiniowanego rozkładu otrzymamy po podstawieniu:

$$\min_{\vec{w}} \sum_{i=1}^{N} \log(1 + 1/2r^2), \tag{3.16}$$

czyli zaproponowane wcześniej kryterium LMLS. Odporna funkcja kryterialna LMLS dedykowana była oryginalnie do algorytmu największego spadku typu on-line, niemniej, jak pokazuje praktyka, stosować ją można z powodzeniem również w strategii uczenia, w której przed aktualizacją wag brany jest pod uwagę cały ciąg obrazów uczących.

Podsumowanie. Algorytm z kryterium LMLS, mimo, że chronologicznie drugi z odpornych algorytmów, jest równocześnie najprostszym i najczęściej cytowanym. Ponadto bywał implementowany i używany jako wzorcowy algorytm do porównań w pracach [7] i [46], dlatego też również w niniejszej pracy użyto go do porównania z opracowanymi w niej metodami. Było to możliwe, jak już wspomniano, dzięki temu, że daje się go również stosować z metodami uczenia drugiego rzędu, choć nie zostało to zaznaczone przez jego autora.

Prostota implementacji wynika tu oczywiście z faktu, że w algorytmie tym jedyna różnica w stosunku do metody klasycznej polega na niewielkiej modyfikacji funkcji celu.

3.2.3 Odporny algorytm propagacji wstecznej RBP

W algorytmie tym [7] Chen *et al.* zaproponowali przyjęcie zmiennej w czasie funkcji kary bazującej na M-estymatorze opracowanym przez Hampela [21], a wykorzystującym tangens hiperboliczny tgh. Bardzo podobne podejście znaleźć można również w późniejszej o kilka lat pracy Andreou [1], a praktycznie niezmienione, choć mniej szczegółowe powtórzenie rozumowania w artykule Walczaka [69]. Odporne M-estymatory stanowią klasę uogólnioną z estymatorów największej wiarogodności, dedykowaną do sytuacji, w której rozkład błędu jest nieznany. Powinny one zachowywać się stabilnie dla danych czystych lub zakłócanych niewielkim białym szumem, a równocześnie zapewniać odporność procedury na występowanie błędów grubych. Zamiast maksymalizacji logarytmu funkcji prawdopodobieństwa, rozwiązuje się:

$$\min_{\theta} \sum_{i=1}^{N} \rho(r_i), \tag{3.17}$$

gdzie θ jest pewną ciągłą i różniczkowalną funkcją zmiennej rzeczywistej, która ponadto jest symetryczna i posiada swe jedyne minimum w zerze. Jeżeli dodatkowo założy się jej ograniczoność, można mówić już o zjawisku zmniejszania wpływu dużych zakłóceń na wartość estymatora. Jeśli przyjmiemy, że $\psi(r_i)$ oznacza pochodną $\theta(r_i)$ możemy zauważyć, że minimalizacja (3.17) równoważna jest z rozwiązaniem równania:

$$\sum_{i=1}^{N} \psi(r_i) = 0. \tag{3.18}$$

W algorytmie RBP funkcja wpływu oparta na estymatorze Hampela [21] przedstawia się więc następująco:

$$\psi_t(r_i) = \begin{cases} r \quad \text{dla } |r| \leqslant a(t) \\ c_1 \operatorname{tgh}(c_2(b(t) - |r|))\operatorname{sign}(r) \quad \text{dla } a(t) < |r| \leqslant b(t) \\ 0 \quad \text{dla } |r| > b(t) \end{cases} \tag{3.19}$$

Wyrażenia $a(t)$ oraz $b(t)$ oznaczają zmienne w czasie (a więc zależne od bieżącej epoki t) punkty odcięcia funkcji, zaś c_1 i c_2 pewne stałe.

Strategia zmian punktów odcięcia jest następująca: na początku zakłada się maksymalną ilość błędów q, które mogą pojawić się w danych. Następnie wybiera się najmniejsze $(1-q)N$ bezwzględne residua $|r(t)|_{1:N} \leqslant |r(t)|_{2:N} \leqslant \ldots \leqslant |r(t)|_{(1-q)N:N}$, po czym za pomocą metody bootstrap szacuje się przedział ufności spełniający zależność:

$$Prob\left\{l_{(1-q)}(t) < |r(t)|_{(1-q)n:n} < u_{(1-q)}(t)\right\} = 0.95 \tag{3.20}$$

Pod a i b podstawia się kolejno $l_{(1-q)}(t)$ i $u_{(1-q)}(t)$. Wartości te aktualizowane się co pewną, ustaloną liczbę epok procesu uczenia. Warto zaznaczyć, że metoda ta przewidziana została do algorytmów, w których aktualizacja wag odbywa się po prezentacji każdego z obrazów uczących. Dla uproszczenia założyć można mniej skomplikowaną postać funkcji kary, posiadającą tylko jeden punkt odcięcia. I tak dla funkcji Cauchy'ego otrzymujemy:

$$\rho(r_i, \beta) = \frac{\beta}{2} \log\left(1 + \frac{r_i^2}{\beta}\right), \tag{3.21}$$

przy czym parametr β przyjmowany jest jako:

$$\beta(t) = |r(t)|_{(1-q)N:N}. \tag{3.22}$$

Problemem pozostaje tu początkowa faza uczenia, którą autorzy proponują przeprowadzić za pomocą metody tradycyjnej, a więc ucząc sieć przez kilka epok minimalizując błąd średniokwadratowy.

Algorytmu RBP - procedura. Całą metodę można przedstawić algorytmicznie w następujący sposób:

1. Wylosuj wartości początkowe parametrów sieci.
2. Oblicz błąd kwadratowy dla wszystkich wyjść sieci, oraz błędy warstw ukrytych (propagacją wsteczną).
3. Wykonaj krok uczenia aktualizując wagi sieci.
4. Jeżeli różnica pomiędzy błędem aktualnym, a poprzednim jest większa od założonego progu, idź do punktu następnego, jeśli nie, wróć do punktu 2.
5. Zeruj licznik epok.

6. Oblicz początkowe parametry opisujące punkty odcięcia.

7. Oblicz błędy warstwy wyjściowej i warstw ukrytych używając odpornej funkcji błędu. Jeśli kryteria stopu spełnione, zakończ uczenie.

8. Aktualizuj parametry opisujące punkty odcięcia (może się to odbywać co określoną ilość epok).

9. Wykonaj kolejny krok uczenia i wróć do punktu 6.

Podsumowanie. Jak więc widać, algorytm ten składa się z dwóch części. W pierwszej z nich ma miejsce uczenia sieci metodą klasyczną, dopiero potem rozpoczyna się właściwe uczenie odporne na błędy. Co pewną ilość epok uczenia odpornego algorytmu, oblicza się szacunkowy przedział błędów, w którym zawarte są błędy grube, następnie ustalone w ten sposób punkty odcięcia używane są do konstruowania, opartej na odpornym M-estymatorze, funkcji kryterialnej.

Algorytm ten dzięki dodatkowemu wykorzystywaniu informacji dotyczącej aktualnych błędów umożliwia lepsze uczenie sieci w przypadku większej zawartości dużych zakłóceń, w porównaniu z metodą LMLS. Doczekał się on publikacji dotyczącej jego praktycznych zastosowań [68].

3.2.4 Odporny algorytm propagacji wstecznej z wyżarzaniem ARBP

Funkcja strat została tu przyjęta jak w zależności (3.21). Autorzy pracy [9] zaproponowali jednak, aby zmiany parametru β odbywały się zgodnie z zadanym schematem wyżarzania. Umożliwia to działanie algorytmu, w którym zbędne staje się początkowe założenie dotyczące ilości błędów grubych zawartych w danych uczących. W pierwszej fazie uczenia minimalizowane kryterium powinno być bliskie błędowi średniokwadratowemu, by po pewnej ilości iteracji zacząć eliminować dane powodujące największe błędy. Aby osiągnąć taki rezultat, punkt odcięcia musi zmieniać się w czasie i zaczynając od stosunkowo dużej wartości stopniowo dążyć do zera. Podejście to znane jest również w dziedzinie odpornej statystyki - odpowiada to sytuacji przechodzenia estymatora od normy L_2 asymptotycznie do normy L_1. Autorzy algorytmu ARBP przebadali następujące sposoby zmian punktu odcięcia:

$\beta(t) = (1.5)^{\gamma/t} - 1,$
$\beta(t) = k\xi^t,$
$\beta(t) = \xi - t/k,$
$\beta(t) = k/t,$
gdzie k, γ, ξ to pewne stałe. Wykazano eksperymentalnie, że najbardziej efektywny jest schemat wyżarzania postaci $\beta(t) = k/t$. Problemem pozostaje jednak nadal dobór parametru, który ostatecznie przyjęto jako $k = 10$, gdyż taka wartość sprawdzała się najlepiej dla testowych przypadków. Warto zwrócić uwagę na fakt, że powyższa procedura dotyczy uczenia typu on-line, w którym douczenie sieci następuje po prezentacji każdego obrazu uczącego.

Algorytm ARBP - procedura. Procedurę postępowania zapisać można następująco:

1. Wylosuj wartości początkowe parametrów sieci i oblicz początkową wartość parametru β.
2. Oblicz błędy warstwy wyjściowej i warstw ukrytych używając odpornej funkcji błędu. Jeśli kryteria stopu spełnione, zakończ uczenie.
3. Aktualizuj parametr β według ustalonego schematu.
4. Wykonaj kolejny krok uczenia i wróć do punktu 2.

Podsumowanie. Odporny algorytm ARBP jest udoskonalaniem opisanego wcześniej algorytmu RBP (sekcja 3.2.3). Nie wykorzystuje on jednak informacji na temat aktualnych wartości błędów, lecz zmienia punkt odcięcia, a więc wartość błędu powyżej której zmniejsza się wpływ powodującego go obrazu, na proces uczenia. Metoda ta wykorzystuje taką samą funkcję kryterialną, co algorytm RBP, jednak zmiany jej parametrów odbywają się w miarę postępów uczenia, zgodnie z wybranym schematem wyżarzania. Zaproponowany przez autorów to odwrotnie proporcjonalna zależność parametru odpowiadającego za punkt odcięcia od aktualnej liczby epok.

3.2.5 TAO - odporny algorytm propagacji wstecznej

Autorzy algorytmu TAO, Pernia-Espinoza *et al.* [46] również skupili się na zwiększeniu odporności procesu uczenia sieci, na błędy pojawiające się w danych uczących, poprzez wprowadzenie nowej funkcji kryterialnej. Funkcja ta obliczana jest w dość skomplikowany sposób, a do jej skonstruowania posłużono się ideą τ-estymatorów zaproponowanych przez Yohai i Zamara w 1988r [73]. Aby zdefiniować τ-estymator zacząć należy od omówienia tzw. S-estymatorów. S-estymator uzyskuje się minimalizując M-estymator wariancji zastosowany do residuów:

$$S = \arg\min_{r_i} s(r_i), \tag{3.23}$$

przy czym

$$\sum_{i=1}^{N} h_1(\frac{r_i}{s(r_i)}) = NE_\phi(h_1(r_i)), \tag{3.24}$$

gdzie $\vec{r} = [r_1, \ldots, r_N]$ jest wektorem residuów, natomiast $h_1(v)$ jest pewną funkcją zmiennej rzeczywistej v, mającą określone własności, np. postaci:

$$h(v) = \begin{cases} \frac{v^2}{2}(1 - \frac{v^2}{c^2} + \frac{v^4}{3c^4}) & \text{dla } |v| \leqslant c \\ 0 & \text{dla } |v| > c \end{cases} \tag{3.25}$$

Na tej podstawie możemy określić już τ-estymator skali, jako:

$$\tau(\vec{r}) = S(\vec{r})[\frac{1}{N}\sum_{i=1}^{N} h_2(\frac{r_i}{S(\vec{r})})]^{1/2}. \tag{3.26}$$

Funkcja h_2, ma postać podobną jak h_1, różni się jednak doborem stałych. Dla wektora parametrów θ definiujemy więc τ-estymator jako:

$$\hat{\tau} = \arg\min_{\theta} \tau(\vec{r}(\theta)). \tag{3.27}$$

Warto zwrócić uwagę, że przy założeniu:

$$2h_2(v) - \psi_2(v)v \geqslant 0, \tag{3.28}$$

gdzie ψ_2 jest pochodną h_2, możemy traktować τ-estymator jako M-estymator, z adaptacyjnie zmienną funkcją wpływu będącą ważoną sumą:

$$\psi(v) = w(\theta)\psi_1(v) + \psi_2(v). \tag{3.29}$$

Wartość $w(\theta)$ uzyskać można po odpowiednim różniczkowaniu.

Powyższe rozważania służyły wprowadzeniu w odpornym algorytmie uczenia TAO funkcji kryterialnej minimalizowanej w procesie uczenia i danej zależnością:

$$E_{TAO} = \sum_{i=1}^{N} \tau^2(r_i, S_t(\vec{r})) \tag{3.30}$$

Odpowiadający za oszacowanie skali estymator $S_t(\vec{r})$ aktualizowany jest w każdej epoce uczenia w sposób następujący:

$$S_{t+1}(\vec{r}) = \{\sum_{i=1}^{N} \frac{S_t^2 h_1(\frac{r_i}{S_t})}{N E_\phi(h_1(\frac{r_i}{S_t}))})\} \tag{3.31}$$

Przy odpowiednio dobranych parametrach c osobno dla funkcji h_1 i h_2 otrzymuje się teoretyczny punkt załamania (*breakdown point* - BDP) [21, 50] dla estymatora równy 0.5 [1], przy jednoczesnym zachowaniu efektywności dla błędów generowanych zgodnie z rozkładem normalnym. W tym celu autorzy przyjęli następujące wartości $c_1 = 1.56$, $c_2 = 6.08$. Metoda ta wprowadzona została dla algorytmu największego spadku, niemniej stosowana być może, jak sugerują jej autorzy, również w połączeniu z innymi gradientowymi algorytmami uczenia.

Algorytm TAO - procedura. W rezultacie procedura postępowania przedstawia się, w skrócie rzecz ujmując, następująco:

1. Wylosuj wartości początkowe parametrów sieci i oblicz początkową wartość parametru opisującego punkt odcięcia S_0.

2. Oblicz błędy warstwy wyjściowej, obliczając wartości h, E_{TAO} i ich pochodnych, oraz bazujące na nich błędy warstw ukrytych. Jeśli kryteria stopu spełnione, zakończ uczenie.

3. Aktualizuj parametr S_t.

4. Wykonaj kolejny krok uczenia i wróć do punktu 2.

[1] W skrócie rzecz ujmując, punkt załamania estymatora, to największa dopuszczalna ilość błędnych obserwacji, dla której zmienia się on w ograniczonym zakresie, przy założeniu, że zakłócenia mogą być dowolnego typu i rozmiaru. Innymi słowy, to największa, tolerowana przez estymator dawka zakłóceń.

Podsumowanie. W odporny algorytmie TAO autorzy zaproponowali postać funkcji kryterialnej opartą na odpornym τ-estymatorze. W metodzie tej, podobnie, jak w przedstawionych poprzednio, uzyskiwanie odporności odbywa się poprzez minimalizowanie wpływu błędów grubych za pomocą odpowiednio skonstruowanego kryterium błędu. Parametry odpowiedzialne za punkty odcięcia błędów aktualizowane są w każdej epoce.

Metoda ta wykorzystuje dość skomplikowany sposób obliczania funkcji kryterialnej, choć sama jej idea, a więc wykorzystanie jako kryterium odpornego estymatora, jest podobna, jak dla algorytmów LMLS, RBP i ARBP.

3.3 Porównanie istniejących odpornych algorytmów uczenia

Spośród opisanych w literaturze czterech odpornych algorytmów uczenia sieci wszystkie oparte są na metodzie uczenia typu "on-line", przy czym jeden z nich, mimo, że predestynowany do takiej formy uczenia, można, jak pokazano dalej, stosować również przy aktualizacji wag po prezentacji wszystkich obrazów uczących. Najprostszy i najstarszy z nich, choć jak pokazują symulacje, całkiem skuteczny, odporny algorytm LMLS polega na przyjęciu zmodyfikowanej postaci funkcji celu, która nie zmienia się w czasie. Algorytmy RBP, ARBP i TAO budują funkcję kary na podstawie aktualnych błędów i/lub postępu procesu uczenia, przy czym skomplikowanie tej procedury może być, jak w przypadku odpornego algorytmu TAO, dość znaczne.

W pracy [9] porównano działanie sieci uczonych algorytmami BP (klasyczny algorytm największego spadku z kwadratową funkcją błędu), LMLS, RBP i ARBP wykazując wyższość tego ostatniego w przypadku wprowadzenia do danych uczących sztucznych zakłóceń typu *gross error* opisanych w sekcji 1.2. Z kolei w artykule prezentującym metodę TAO [46] pokazano na przykładzie symulującym pewne określone okoliczności, jej wyższość nad algorytmem z kryterium LMLS. Trudno uznać owe wyniki za miarodajne, jako, że uzyskane zostały przez kilkukrotne zaledwie trenowanie sieci na zadanych przykładach.

Aby zweryfikować te rezultaty, na potrzeby niniejszej pracy przeprowadzono porównanie istniejących odpornych algorytmów uczenia w warunkach opisanych przez

ich autorów. Jak się okazało, dokładne powtórzenie owych eksperymentów przyniosło wyniki nieco inne od spodziewanych. Szczegółowy opis i uwagi dotyczące tych symulacji zebrane zostały w raporcie [57]. Aby pokrótce pokazać różnice w jakości działania algorytmów w tabelach 3.1, oraz 3.2 zawarty został błąd średniokwadratowy popełniany przez sieci uczone na danych odpowiednio zawierających biały szum lub niewielką ilość błędów grubych, oraz na danych czystych. Szczegółowy sposób przeprowadzenia badań numerycznych był analogiczny, jak w przypadku badania algorytmów własnych i został dokładnie opisany w sekcji 5.4.

Symulacje wykonano dla zadania aproksymacji funkcji jednej zmiennej opisanego w rozdziale poświęconym zadaniom testowym (sekcja 5.4) dla sieci o 5 i 10 neuronach w warstwie ukrytej.

Wnioski Warto zauważyć, że do prawidłowego nauczenia sieci na danych czystych wystarczy tu struktura zawierająca zaledwie 5 neuronów ukrytych, co wyraźnie widać, jeśli porówna się wyniki uzyskane po nauczeniu sieci algorytmem z kryterium kwadratowym. W przypadku zaś danych z dużymi zakłóceniami, błąd dla większej sieci staje się o ponad 50% większy, co świadczy o tym, że jej nadmiarowość przyczynia się do uwzględniania w procesie uczenia danych nie należących do aproksymowanej funkcji, a więc dodanych sztucznie błędów grubych. Podobne prawidłowości zaobserwować można dla sieci uczonych przy użyciu algorytmu z kryterium LMLS. Wyniki dla danych czystych, oraz posiadających niewielkie zakłócenia, są zbliżone do uzyskanych przy uczeniu algorytmem klasycznym, niezależnie od rozmiaru sieci. W przypadku danych, w których zasymulowano występowanie błędów grubych, otrzymane wartości błędu są jednak trzy-, czterokrotnie lepsze od tych otrzymanych metodą tradycyjną.

Zupełnie inaczej przedstawia się analiza błędu dla algorytmu RBP. Jakość jego działania przy uczeniu sieci o mniejszym rozmiarze była zdecydowanie gorsza od rezultatów uzyskanych za pomocą omówionych wcześniej metod. Wystarczy wspomnieć, że dla danych bez zakłóceń błąd aproksymacji był prawie czterokrotnie większy i dochodził do poziomu osiąganego dla metody LMLS przy danych z błędami grubymi, dla tego zaś typu danych był jedynie dwukrotnie lepszy od algorytmu BP. Z kolei dla sieci o nadmiarowej liczbie neuronów ukrytych ujawniły się opisywane

Tabela 3.1: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami typu "on-line" na danych zaszumionych.((5, (10) - oznacza 5 lub 10 neuronów ukrytych)

Algorytm	Dane z $\delta = 0.05$ błędów grubych				Dane z białym szumem			
	Średnia	σ	Max	Min	Średnia	σ	Max	Min
BP(5)	0,009	0,0035	0,0163	0,0024	0,0022	0,0007	0,0037	0,0011
RBP(5)	0,0041	0,001	0,0066	0,0024	0,0048	0,0018	0,0095	0,0031
LMLS(5)	0,0028	0,0013	0,0074	0,001	0,0022	0,0007	0,004	0,0013
ARBP(5)	0,0076	0,0031	0,0128	0,0035	0,0088	0,0037	0,0156	0,0035
BP(10)	0,0141	0,0091	0,0335	0,0043	0,0028	0,0008	0,0046	0,0014
RBP(10)	0,003	0,0009	0,0052	0,0019	0,0022	0,0006	0,0035	0,0015
LMLS(10)	0,0038	0,0012	0,0063	0,0019	0,0029	0,0008	0,0048	0,0019
ARBP(10)	0,0025	0,0005	0,0033	0,0017	0,0029	0,0008	0,0048	0,0019

w literaturze zalety tej odpornej metody: dla danych czystych i z niewielkim szumem błędy były podobnego rzędu, jak dla metod ze stałą funkcją kryterialną, natomiast dla ciągu uczącego zawierającego duże zakłócenia, okazały się mniejsze nawet ponad czterokrotnie od algorytmu tradycyjnego i o około 30% od algorytmu LMLS.

Podobna sytuacja miała miejsce w przypadku wyników działania sieci uczonych algorytmem ARBP. Dla architektury z pięcioma neuronami w warstwie ukrytej rezultaty symulacji okazały się najgorsze spośród wszystkich testowanych metod. Również dla tej metody jej odporność daje o sobie znać, gdy sieć ma strukturę większą od minimalnej. I tak dla danych bez zakłóceń uzyskuje ona błędy podobnego rzędu, co inne algorytmy, natomiast dla danych zawierających błędy grube okazuje się rzeczywiście najlepsza. Błąd struktury nauczonej klasyczną metodą BP jest ponad pięciokrotnie większy, niż w przypadku metody ARBP.

Widać więc, że ocena istniejących odpornych algorytmów uczenia nie jest sprawą prostą i trudno jednoznacznie powiedzieć, który z nich jest algorytmem najlepszym, pomimo, że autorzy każdego próbowali wykazać wyższość swojej metody. Najbardziej uniwersalne wydaje się być podejście ze stałą funkcją kryterialną LMLS, które dawało zadawalające wyniki niezależnie od rozmiaru sieci. Dwa pozostałe algorytmy odporne wymagają struktury o rozmiarze odpowiednio dużym, ponadto ważny jest w nich właściwy dobór parametrów, oraz, ewentualnie, oszacowanie maksymalnej zawartości błędów w danych uczących.

Tabela 3.2: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami typu "on-line" na danych czystych.((5), (10) - oznacza 5 lub 10 neuronów ukrytych)

Algorytm	Dane czyste			
	Srednia	σ	Max	Min
BP(5)	0,0008	0,0002	0,0013	0,0005
RBP(5)	0,0028	0,0013	0,0063	0,0019
LMLS(5)	0,0008	0,0002	0,0013	0,0005
ARBP(5)	0,0048	0,0032	0,011	0,0022
BP(10)	0,0005	0,0001	0,0006	0,0004
RBP(10)	0,0008	0,0001	0,0011	0,0005
LMLS(10)	0,0004	0,0001	0,0006	0,0003
ARBP(10)	0,0012	0,0002	0,0014	0,0008

Dodatkowa zaleta algorytmu z kryterium LMLS to możliwość jego stosowania również z metodami uczenia typu "batch", choć nie wspomina o tym jego autor. Pozwala to potencjalnie na przyspieszenie procesu uczenia, choć rozważyć tu trzeba możliwą utratę części właściwości algorytmu, w którym nie są już rozpatrywane wartości funkcji celu zależne od pojedynczych residuów, lecz od ich całości.

Niestety, autorzy algorytmu TAO nie podali wszystkich parametrów początkowych i warunków, w jakich był on testowany. Z tego względu powtórzenie symulacji okazało się niemożliwe. Podjęte w tym kierunku próby wykazały, że algorytm w formie przedstawionej w artykule bardzo często zbiega do minimów lokalnych, dlatego nie został uwzględniony w porównaniu. Należy zatem zapewne założyć, że wynik uzyskane przy uczeniu sieci tym właśnie algorytmem nie odbiegają znacznie od wartości podanych przez jego autorów, którzy porównywali go jedynie z metodą LMLS, wykazując na przykładzie wyższość nad tą ostatnią w przypadku błędów o niesymetrycznym rozkładzie.

Rozdział 4

Opracowane odporne algorytmy uczenia

4.1 Wprowadzenie

Jak można zauważyć analizując przytoczone w poprzednim rozdziale, istniejące przed rozpoczęciem badań mających na celu powstanie tej pracy, odporne algorytmy uczenia sieci neuronowych, literatura dotycząca tej tematyki jest bardzo uboga i zamyka się w kilku zaledwie publikacjach [7, 9, 34, 46]. Przedstawione metody mają ponadto kilka wad, spośród których wymienić należy przede wszystkim oparcie mechanizmu uodparniania tylko i wyłącznie na stosowaniu odpornych funkcji błędów (związanych z odpornymi M-estymatorami), jak również przeznaczenie ich do współpracy z algorytmami typu "on-line", co determinuje użycie wolniejszych metod optymalizacyjnych pierwszego rzędu.

Przedstawione dalej, nowe algorytmy uczenia sieci, wykorzystują różnorakie mechanizmy uodparniania przeciw błędom grubym. Biorą one pod uwagę również inne typy zakłóceń niż tylko związane z tzw. *gross error model*, omówionym w sekcji 1.2. Dodatkowo większość z nich może być stosowana również z metodami uczenia skumulowanego, co umożliwia stosowanie algorytmów drugiego rzędu.

Opracowane w niniejszej pracy metody uczenia podzielić można, jak wspomniano we wstępie, na kilka grup. Do pierwszej z nich zaliczają się algorytmy ze zmodyfikowaną funkcją kryterialną. Co prawda zastosowany mechanizm jest tu podobny, jak

w cytowanych metodach, jednak wybór postaci funkcji z przycinaniem błędów, sprawia, że ich działanie jest inne, a w niektórych przypadkach (dla błędów w wektorze wejściowym) zdecydowanie lepsze.

Druga grupa, to algorytmy oparte na zmianach współczynnika uczenia. Pierwszy z nich, przeznaczony do uczenia "on-line" zmienia współczynnik uczenia w prosty sposób w zależności od aktualnego błędu, drugi zaś wykorzystuje w tym celu minimalizację pewnej funkcji błędu.

Kolejne podejście to wprowadzenie zmiennej funkcji aktywacji neuronów. Za pomocą zmian kształtu owej funkcji uzyskać można zmniejszanie w procesie uczenia wpływu błędów grubych.

Osobno potraktować można algorytm drugiego rzędu dedykowany dla kryterium LMLS. Jest on przykładem na to, że również dla metod odpornych konstruować można szybkie w działaniu i nieskomplikowane w implementacji procedury minimalizacji funkcji celu.

Ostatni z opracowanych algorytmów opiera się na wstępnym oznaczeniu danych podejrzanych o bycie błędami grubymi. Stosować go można z odpowiednio zmodyfikowaną funkcją kryterialną.

Warto w tym miejscu przypomnieć, sformułowane na wstępie, postulaty dotyczące odpornych algorytmów uczenia. Najkrócej rzecz ujmując można powiedzieć, że odporny algorytm uczenia sieci powinien:

- uczyć sieć prawidłowo na danych czystych i zanieczyszczonych niewielkim białym szumem;
- uczyć sieć możliwie dobrze dla danych zawierających duże zakłócenia różnego typu;
- posiadać mechanizm "uodparniania" wbudowany w schemat uczenia.

Dodatkowo założono, że dla algorytmu nie jest dostępna wiedza na temat typu zakłóceń, jakie wystąpią w danych, natomiast może on mieć informacje dotyczące maksymalnej ich zawartości w ciągu uczącym. Dokładna znajomość rozkładu implikuje bowiem możliwość stworzenia algorytmu dedykowanego do konkretnej sytuacji, np. przy wykorzystaniu metody największej wiarogodności, jak pokazano w poprzednim rozdziale. Jest to, oczywiście, sytuacja raczej teoretyczna i niemająca zwykle

miejsca w praktyce, należy więc założyć, że zakłócenia pojawiają się, a ich rozkład jest nieznany.

Wszystkie opracowane w ramach niniejszej pracy algorytmy przeznaczone są do uczenia sieci jednokierunkowych sigmoidalnych. Potencjalne bogactwo klas umożliwia naturalnie tworzenie odpornych metod uczenia innych typów sieci, niemniej jednak rozbudowałoby to znacznie rozpatrywany problem. Uzasadnić zajęcia się właśnie tym rodzajem SSN można przytaczając raz jeszcze argumenty przedstawione przy omawianiu istniejących odpornych algorytmów. Otóż, sieci jednokierunkowe są najbardziej popularne, najlepiej opisane, mają szeroki wachlarz zastosowań, ponadto można znaleźć związki pomiędzy procesem ich uczenia a pewnymi pojęciami z dziedziny statystyki. Dodatkowo, jak już wspominano, sieci sigmoidalne są z natury bardziej odporne od większości sieci RBF [36].

4.2 Odporny algorytm LTS

Wprowadzenie

Przedstawione w poprzednim rozdziale odporne algorytmy uczenia opierały się na wprowadzeniu zmodyfikowanej funkcji błędu, wykorzystującej funkcje odpornych M-estymatorów [21, 28, 42], która pozwalała na zmniejszenie wpływu dużych zakłóceń na proces uczenia. Opracowany w ramach niniejszych badań i przedstawiony dalej odporny algorytm, opiera się z kolei na innym odpornym estymatorze zwanym estymatorem najmniejszych przycinanych kwadratów (czasem spotykana jest nazwa "trymowanych kwadratów"), w którym część błędów nie jest w ogóle brana pod uwagę.

Odporny estymator LTS

Odporny estymator nazwany estymatorem najmniejszych przycinanych kwadratów *least trimmed squares* (LTS) wprowadzony został przez Rousseuw w pracach [50, 51]. Jest to klasyczny juz estymator posiadający wysoki punkt załamania (*break-down point*, BDP) [42, 28]. W swoich własnościach podobny jest nieco do najmniejszej mediany kwadratów (*least median of squares* - LMS) [50], charakteryzuje się jednak lepszą zbieżnością dla rozkładu normalnego.

Estymator ten doczekał się wielu różnych zastosowań w obszarach takich, jak problemy regresji zarówno liniowej, jak i nieliniowej, problemy małej liczby próbek, analiza wrażliwościowa, czy wykrywanie dużych zakłóceń, a więc punktów odstających mogących występować w danych. Dodatkową jego zaletą jest stosunkowo niska złożoność obliczeniowa, również dla problemów wielowymiarowych, w porównaniu z innymi estymatorami o podobnych właściwościach, jak np. RM (*repeated median*) [61], oraz prostsza implementacja w porównaniu z innymi estymatorami [74, 67].

Różnica pomiędzy zarówno estymatorem LTS, jak i LMS, a typową minimalizacją sumy kwadratów błędów, polega na zmianie operacji wykonywanej na residuach i zastąpieniu sumowania medianą, bądź przycinaną sumą. Warto zauważyć, że jest to procedura zupełnie inna, niż w przypadku M-estymatorów, w których pomimo zamiany funkcji kwadratowej na bardziej skomplikowaną, sama operacja sumowania pozostaje bez zmian.

Rozważmy uogólniony model regresji nieliniowej:

$$y_i = \eta(x_i, \theta) + \epsilon_i, \quad i = 1, \ldots, N, \tag{4.1}$$

gdzie y_i oznacza zmienną zależną, $x_i = (x_{i1}, \ldots, x_{ik})$ niezależny wektor wejściowy systemu, a $\theta \in R^n$ reprezentuje poszukiwany wektor parametrów. Losowe zakłócenie ϵ_i jest ciągiem zmiennych losowych typu iid posiadających funkcję gęstości. Nieliniowy estymator najmniejszych przycinanych kwadratów LTS zdefiniowany jest wtedy jako:

$$\hat{\theta} = \arg\min_{\theta \in R^n} \sum_{i=1}^{h} (r^2)_{i:N}, \tag{4.2}$$

gdzie $(r^2)_{1:N} \leqslant \ldots \leqslant (r^2)_{N:N}$ to uporządkowane niemalejąco błędy kwadratowe $r_i^2(\theta) = \{y_i - \eta(x_i, \theta)\}^2$. Stała trymowania h przyjmuje wartości całkowite z przedziału $N/2 < h \leqslant N$, aby zapewnić, że $N - h$ obserwacji o największych residuach bezpośrednio nie wpływa na estymator. Dla nieliniowego LTS pokazano [63], że przy $\eta(x, \theta) = g(x^T\theta)$, przy czym $g(t)$ jest nieograniczone dla $t \to \infty$, punkt załamania estymatora dąży do 1/2 przy $h = [N/2] + 1$, gdzie $[N]$ oznacza część całkowitą z N, oraz do 0 przy $h = N$. W takim przypadku estymator ten staje się równoważny nieliniowemu estymatorowi najmniejszych kwadratów. Dla innych klas obliczono dotychczas jedynie dolne i górne ograniczenia.

W przypadku regresji liniowej i liniowego estymatora LTS, przy pewnych, niezbyt silnych założeniach, granica załamania wynosi $([(N-n)/2]+1)/N$ dla $h = [N/2]+[(n+1)/2]$, oraz $([N/2]-n+2)/N$ przy $h = [N/2]+1$, $n > 1$ [52]. Oznacza to, że graniczny BDP wynosi 1/2, co jest oczywiście najwyższą teoretycznie oczekiwaną wartością. Niezerowy punkt załamania odpornego estymatora implikuje odporność nie tylko na zakłócenia pionowe (w wektorze y), ale także na tzw. *leverage points*, czyli mocno zniekształcone wartości x. O wyższości LTS nad LMS decyduje lepsza asymptotyczna zbieżność dla rozkładu normalnego, wynosi ona $N^{-1/2}$, co jest wartością wykazywaną przez M-estymator zwany Huber skipped mean. Oczywistą korzyścią przemawiającą za zastosowaniem LTS, a nie LMS jest znacznie gładsza postać funkcji celu, co daje nadzieję na możliwość użycia optymalizacji gradientowej.

Opis algorytmu LTS

Dla skupienia uwagi, rozważmy prostą sieć neuronową zawierającą jedną warstwę ukrytą (struktura taka zwykle nazywana jest siecią dwuwarstwową). Sieć trenowana jest na ciągu N par uczących $\{(\vec{x}_1, \vec{t}_1), (\vec{x}_2, \vec{t}_2), \ldots, (\vec{x}_N, \vec{t}_N)\}$, gdzie $\vec{x}_i \in R^p$, oraz $\vec{t}_i \in R^q$. Dla danego wektora wejściowego $\vec{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^T$, wyjście j-ego neuronu warstwy ukrytej może być zapisane jako:

$$z_{ij} = f_1(\sum_{k=1}^{p} w_{jk}x_{ik} - b_j) = f_1(inp_{ij}), \quad \text{dla } j = 1, 2, \ldots, l, \tag{4.3}$$

przy czym $f_1(\cdot)$ jest funkcją aktywacji neuronów warstwy ukrytej (dla uproszczenia przyjęto, że wszystkie neurony w warstwie mają tę samą funkcję aktywacji), w_{jk} oznacza wagę pomiędzy k-tym wejściem a j-tym neuronem, natomiast b_j próg (*bias*) j-tego neuronu. Ważoną sumę wejść j-tego neuronu dla obrazu uczącego i oznaczono tu jako inp_{ij}. Wtedy wektor wyjściowy sieci $\vec{y}_i = (y_{i1}, y_{i2}, \ldots, y_{iq})^T$ dany jest jako:

$$y_{iv} = f_2(\sum_{j=1}^{l} w'_{vj}z_{ij} - b'_v) = f_2(inp_{iv}), \quad \text{dla } v = 1, 2, \ldots, q. \tag{4.4}$$

Funkcję aktywacji neuronów drugiej warstwy oznaczono tu jako $f_2(\cdot)$, w'_{vj} jest wagą pomiędzy vth neuronem warstwy wyjściowej a j-ym neuronem warstwy ukrytej, natomiast b'_v jest progiem aktywacji v-tego neuronu. Ważoną sumę wejść v-tego neuronu dla obrazu uczącego i oznaczono tu jako inp_{iv}.

Wprowadźmy teraz odporne kryterium błędu LTS bazujące na estymatorze najmniejszych przyciętych kwadratów. Nowa postać funkcji błędu zdefiniowana jest następująco:

$$E_{LTS} = \sum_{i=1}^{h} (r^2)_{i:N}. \tag{4.5}$$

W tym przypadku $(r^2)_{1:N} \leqslant \ldots \leqslant (r^2)_{N:N}$ to podniesione do kwadratu residua postaci:

$$r_i^2 = \{\sum_{v=1}^{q} |(y_{iv} - t_{iv})|\}^2. \tag{4.6}$$

Od wielkości stałej trymującej h zależy ilość obrazów uczących, które będą traktowane jako punkty odstające, dlatego sposób jej doboru ma istotne znaczenie dla właściwości nowego kryterium.

Dla uproszczenia załóżmy, że wagi sieci modyfikowane są metodą największego spadku (warto jednak zwrócić uwagę, że kryterium to może zostać zastosowane również w innych algorytmach gradientowych, np. metodzie gradientów sprzężonych). Przy takim założeniu otrzymujemy następującą zmianę wag, kolejno dla pierwszej i drugiej warstwy, w każdym kroku procesu uczenia:

$$\Delta w_{jk} = -\eta \frac{\partial E_{LTS}}{\partial w_{jk}} = -\eta \frac{\partial \sum_{i=1}^{h} (r^2)_{i:n}}{\partial r_i} \frac{\partial r_i}{\partial w_{jk}}, \tag{4.7}$$

$$\Delta w'_{vj} = -\eta \frac{\partial E_{LTS}}{\partial w'_{vj}} = -\eta \frac{\partial \sum_{i=1}^{h} (r^2)_{i:n}}{\partial r_i} \frac{\partial r_i}{\partial w'_{vj}}, \tag{4.8}$$

gdzie

$$\frac{\partial r_i}{\partial w_{jk}} = f_2'(inp_{iv}) w'_{vj} f_1'(inp_{ij}) x_{ik}, \tag{4.9}$$

oraz

$$\frac{\partial r_i}{\partial w'_{vj}} = f_2'(inp_{iv}) z_{ij}. \tag{4.10}$$

Poważnym problemem, jaki możemy napotkać realizując powyższy schemat uczenia, jest nieciągłość gradientu funkcji celu E_{LTS}, co mogłoby potencjalnie utrudnić, albo wręcz uniemożliwić uczenie metodami gradientowymi. Doświadczenia autora niniejszej pracy, związane z opracowaniem odpornych na uszkodzenia sieci jednokierunkowych z medianą, jako funkcją wejściową neuronu [58], pokazały jednak, że problem taki jest do przezwyciężenia. Dotyczyło to przypadku bardziej nieciągłej funkcji, a mianowicie mediany, przy czym zastąpiono jej gradient, pochodną sumy, aby zapewnić jego ciągłość.

W tym wypadku stopień nieciągłości gradientu zależy od h, a sam gradient funkcji może być zapisany jako:

$$\frac{\partial \sum_{i=1}^{h}(r^2)_{i:n}}{\partial r_i} = \begin{cases} 2r_i & \text{for } r_i^2 \leqslant (r^2)_{i:h} \\ 0 & \text{for } r_i^2 > (r^2)_{i:h} \end{cases} \tag{4.11}$$

Jak pokazały symulacje, taka jego postać jest wystarczająca do prawidłowego działania procesu uczenia i nie ma konieczności zastępowania go żadnym przybliżeniem.

Przy stosowaniu znanych z literatury odpornych algorytmów uczenia pojawia się problem doboru właściwego punktu startowego procedury, co ma największe znaczenie w algorytmach, które opierają się na funkcji celu modyfikowalnej w zależności od postaci danych uczących. Problem ów podzielić można na dwa podproblemy: wybór właściwych parametrów początkowych (wag) sieci, oraz wybór skali, czyli przedziału wartości błędów, lub ogólniej danych, które odporny algorytm będzie traktował, jako prawidłowe. Oczywistym jest, że nieprawidłowy dobór wspomnianych wartości skutkować może dla procesu uczenia podążeniem w niewłaściwym kierunku, czy też zwiększeniem ryzyka utknięcia w minimum lokalnym. Pierwsze zagadnienie rozwiązać można np. poprzez ustalenie wag początkowych po przeprowadzeniu jednego, lub dwóch kroków minimalizacji algorytmem klasycznym (a więc z kryterium kwadratowym) i tak najczęściej czynią autorzy podobnych algorytmów [7]. Taki sposób inicjalizacji wag przyjęto również dla kryterium LTS.

Drugi problem w tym wypadku sprowadza się do wyboru trymującego parametru h. W niniejszej pracy zaproponowano dwa sposoby jego ustalania. Pierwszy, prymitywniejszy, polega na dostosowaniu h do założonego a priori maksymalnego udziału danych odstających w zbiorze danych uczących, czyli wybraniu pewnej wartości h, która ogranicza z góry największą spodziewaną względną ilość dużych zakłóceń w zbiorze danych. Naturalnie pociąga to za sobą konieczność dodatkowej wiedzy, na temat tego, jak będą owe dane wyglądały, widać więc, że strategia ta nie jest zbyt użyteczna. Ponadto trzeba pamiętać, że im więcej obrazów uczących traktowanych będzie jako błędy grube, tym mniejsza będzie efektywność metody dla danych czystych lub z gaussowskim szumem [50, 51].

Drugie podejście to znane z odpornej statystyki oszacowanie za pomocą mediany błędów [42, 28]:

$$h = \|\{r_i : |r_i| < c * \text{median}(|r_i|), i = 1 \ldots n\}\|, \tag{4.12}$$

gdzie stała $c = 6$. Tak opisany estymator skali, w którym rozrzut definiowany jest jako wielokrotność mediany jest stosunkowo często spotykany w literaturze.

Stałą trymowania można również konstruować na podstawie popularnego odpornego estymatora MAD (*median of absolute deviation*) [28] jako:

$$h = \|\{r_i : |r_i| < c * \text{median}\left(|r_i| - \text{median}(|r_i|)\right), i = 1 \ldots n\}\|, \tag{4.13}$$

przy czym, dla zgodności z rozkładem normalnym $c = 1.483$. Estymator ten szacuje medianę bezwzględnych odchyleń od mediany pomiarów (w tym wypadku są nimi błędy). Residua w powyższym wzorze obliczone są dla sieci z zainicjalizowanymi wagami, po zakończeniu działania algorytmu bez modyfikacji. Jak można zauważyć, obliczenie parametru h odbywa się tylko na początku procesu uczenia (po przeprowadzeniu wstępnego nauczenia sieci) i jego wartość pozostaje stała do końca. W ramach niniejszej pracy próbowano również stosować zmienny parametr h zmniejszający swoją wartość w czasie. Jak jednak pokazały eksperymenty dla różnych schematów wyżarzania, podejście takie skutkowało często niepożądanymi i trudnymi do przewidzenia zaburzeniami w procesie minimalizacji, a co za tym idzie nieprawidłowym uczeniem sieci. Z tego względu zrezygnowano z tego pomysłu pozostając przy dwóch opisanych właśnie sposobach szacowania stałej trymowania.

Algorytm LTS został przez autora niniejszej pracy przedstawiony szczegółowo w [56].

Algorytm LTS - procedura

W skrócie rzecz ujmując, schemat postępowania w opracowanym odpornym algorytmie LTS można zapisać w sposób następujący:

1. Wylosuj wartości początkowe parametrów sieci
2. Wykonaj wstępne uczenie algorytmem RBP
3. Oblicz wartość h.
4. Uszereguj kwadraty błędów w porządku rosnącym.
5. Oblicz błąd Lts dla wszystkich wyjść sieci, oraz błędy warstw ukrytych (propagacją wsteczną). Jeśli kryterium stopu spełnione, zakończ uczenie.

6. Wykonaj krok uczenia aktualizując wagi sieci (np. zależności (4.7) i (4.8) i wróć do punktu czwartego.

Podsumowanie algorytmu LTS

Opracowany w niniejszej pracy odporny algorytm z kryterium LTS, różni się od cytowanych metod przede wszystkim postacią funkcji kryterialnej, a w szczególności obecnością w niej przycinania, dzięki czemu nie uwzględnia się w każdej epoce uczenia pewnej liczby największych błędów kwadratowych. Podobnie, jak w przypadku odpornego estymatora LTS, spodziewać się można, że metoda ta będzie zwiększała odporność nie tylko na błędy wartości opisujących zadane wyjścia sieci, ale również zakłócenia pojawiające się w wektorze podawanym na jej wejście.

Z opracowanych dwóch wariantów algorytmu, pierwszy, w którym zakłada się odgórnie maksymalną zawartość błędów w ciąg uczącym, jest z założenia wrażliwy na jej właściwy dobór. Zbyt mała jej wartość powoduje, iż część zakłóceń może być uwzględniona w procesie uczenia, zbyt duża zaś może powodować wolne i nieefektywne działanie algorytmu. Gdy jednak można dokładnie oszacować ilość błędów w danych, metoda ta powinna radzić sobie lepiej niż wariant drugi.

Drugi sposób to szacowanie zawartości błędnych obrazów za pomocą jednego z odpornych estymatorów wariancji. W tym przypadku ustalenie parametrów odpowiadających za punkt odcięcia odbywa się automatycznie, co sprawia, że metoda jest bardziej uniwersalna, liczyć się jednak należy z tym, że uzyskiwane nią rezultaty będą gorsze od tych, przy prawidłowo oszacowanej ilości błędów w wariancie pierwszym.

Metoda przeznaczona jest do uczenia skumulowanego.

4.3 Odporny algorytm LTLS

Wprowadzenie

Jako, że zastosowanie algorytmu LTS do uczenia sieci przyniosło dobre rezultaty, również dla danych zawierających błędy w wektorze wejściowym, postanowiono zastosować ideę przycinania (trymowania) do funkcji innej niż kwadratowa. Idea zastosowania przycinania pewnej ilości danych, do estymatorów innych, niż najmniejszych

kwadratów wzmiankowana była również na gruncie odpornej statystyki, aczkolwiek nie ma gotowych odpowiedzi na pytanie, jak zrobić to najefektywniej.

Opis algorytmu LTLS

W niniejszej pracy, abstrahując od teorii odpornych estymatorów, zajęto się, dla skupienia uwagi, prostą funkcją $\rho(r_i)$ zaczerpniętą z kryterium LMLS, daną zależnością 3.12, którą połączono z przycinaniem w sposób analogiczny, jak to miało miejsce w algorytmie LTS. Funkcja ta ma prostą postać, sprawdza się zarówno w odpornych M-estymatorach, jak również jako kryterium uczenia sieci.

Minimalizowane w tym algorytmie kryterium zapisać więc można jako:

$$E_{LTLS} = \sum_{i=1}^{h} (\log(1 + \frac{1}{2} {r_i}^2))_{i:N}. \tag{4.14}$$

Nowe kryterium nazwano LTLS (*Least Trimmed Log Squres*). Różnica w stosunku do algorytmu LTS jest taka, że sumowane są tu nie uszeregowane kwadraty błędów, lecz ich wartości poddane dodatkowemu przekształceniu. Wartość gradientu jak i zmianę wag sieci obliczyć można w sposób analogiczny, jak w przypadku LTS, a więc:

$$\Delta w_{jk} = -\eta \frac{\partial \sum_{i=1}^{h} (\log(1 + \frac{1}{2} r^2))_{i:N}}{\partial r_i} \frac{\partial r_i}{\partial w_{jk}}, \tag{4.15}$$

$$\Delta w'_{vj} = -\eta \frac{\partial \sum_{i=1}^{h} (\log(1 + \frac{1}{2} r^2))_{i:N}}{\partial r_i} \frac{\partial r_i}{\partial w'_{vj}}, \tag{4.16}$$

oraz:

$$\frac{\partial \sum_{i=1}^{h} (\rho(r_i))_{i:N}}{\partial r_i} = \begin{cases} \frac{r_i}{1+\frac{1}{2}{r_i}^2} & \text{dla } \log(1 + \frac{1}{2}{r_i}^2) \leqslant (\log(1 + \frac{1}{2} r^2))_{i:h} \\ 0 & \text{dla } \log(1 + \frac{1}{2}{r_i}^2) > (\log(1 + \frac{1}{2} r^2))_{i:h} \end{cases} \tag{4.17}$$

Pozostałe składniki obliczane są podobnie jak w zależnościach (4.9) i (4.10). Warto zauważyć, że mamy tu teraz zarówno przycinanie pewnej ilości błędów niezależnie od ich wielkości, jak i eliminację błędów przekraczających zadany poziom. W związku z tym algorytm powinien filtrować dane odstające zarówno wtedy, gdy nie powodują one dużych błędów, jak i wtedy, gdy jest ich więcej, niż początkowo założono i takie błędy powodują. Przyjęto tutaj stałą wartość $h/N = 0.9$, co jak się okazało, dawało stosunkowo dobre rezultaty dla różnych ilości dużych zakłóceń w ciągu uczącym.

Algorytm LTLS - procedura

Dla algorytmu opracowanego algorytmu LTLS procedura postępowania przedstawia się podobnie, jak w przypadku metody LTS:

1. Wylosuj wartości początkowe parametrów sieci.
2. Uszereguj wartości LMLS błędów w porządku rosnącym.
3. Oblicz błąd LTLS dla wszystkich wyjść sieci, oraz błędy warstw ukrytych (propagacją wsteczną). Jeśli kryterium stopu spełnione, zakończ uczenie.
4. Wykonaj krok uczenia aktualizując wagi sieci i wróć do punktu drugiego.

Podsumowanie algorytmu LTLS

Algorytm LTLS powstał poprzez zastosowanie w uczeniu sieci kryterium błędu będącego połączeniem przycinanego błędu średniokwadratowego (LTS) z funkcją LMLS. W ten prosty sposób uzyskano funkcję kryterialną, która powinna umożliwić eliminowanie dużych zakłóceń w dwojaki sposób: po pierwsze, przez usuwanie częście największych błędów, po drugie zaś poprzez odpowiednie obcinanie ich za pomocą funkcji stosowanej w jednym z M-estymatorów.

Metoda przeznaczona jest do uczenia skumulowanego.

Użyty w algorytmie sposób modyfikacji wag, a w szczególności obliczania gradientu, jest analogiczny jak dla metody LTS. Można przypuszczać, że algorytm ten będzie się sprawdzał przy dużej zawartości zakłóceń, natomiast gorsze może być jego działania dla danych czystych.

4.4 Odporny algorytm ze zmiennym współczynnikiem uczenia

Wprowadzenie

W pierwszym, z historycznego punktu widzenia, algorytmie uczenia sieci, opartym na zasadzie wstecznej propagacji błędu, współczynnik uczenia ustalany był przed rozpoczęciem działania metody minimalizacyjnej i pozostawał niezmieniony przez cały czas działania algorytmu. Jego wartość dobierana była najczęściej w sposób

eksperymentalny, tak, aby zapewnić, że proces uczenia w większości przypadków odpowiednio zbliża się do poszukiwanego ekstremum funkcji celu. Oczywiście dziś podejście to, mimo jego niewątpliwej prostoty, nie jest już praktycznie stosowane. Wynika to z faktu, że przebieg procesu uczenia, jest bardzo wrażliwy na dobór odpowiedniej wartości współczynnika kroku - najprościej rzecz ujmując, za duża jego wartość spowodować może "przeskakiwanie" minimów, zbyt mała zaś implikuje niską zbieżność i zwiększa ryzyko utknięcia w ekstremach lokalnych.

By poradzić sobie z tym problemem powstało wiele metod polegających na doborze współczynnika uczenia dynamicznie w trakcie samej procedury uczenia sieci. Wymieniając jedynie najważniejsze z nich, wspomnieć należy o metodzie adaptacyjnego doboru współczynnika uczenia, w której jest on zwiększany o zadaną wartość, jeśli bieżący błąd sieci przekracza błąd poprzedni (proces jest daleko od optimum) i zmniejszany w przypadku przeciwnym [66]. Przy odpowiednio ustalonych warunkach zmian współczynnika, ten sposób postępowania pozwala na przyspieszenie działania algorytmu. Istnieje również kilka innych sposobów adaptacyjnego doboru współczynnika uczenia, opisanych np. w [4] i [5].

Bardziej zaawansowane podejście polega z kolei na poszukiwaniu, w każdej epoce algorytmu, współczynnika uczenia, który minimalizuje funkcję celu w zadanym kierunku [12]. Aby przyspieszyć działanie metody stosuje się tu często bezgradientowe sposoby poszukiwań minimum w kierunku, np. bisekcji, Fibbonaciego, kwadratowej aproksymacji funkcji celu, etc.

W metodzie delta-bar-delta [29] natomiast, dla każdej wagi sieci dobierany jest osobny współczynnik uczenia. Znak gradientu funkcji błędu dla pojedynczej wagi porównywany jest z jego znakiem w poprzedniej iteracji i na tej podstawie, w zależności od kierunku zmian gradientu, dokonuje się zwiększenia, bądź redukcji współczynnika uczenia.

Warto podkreślić, że żadna z opisanych w literaturze metod doboru współczynnika uczenia nie bierze pod uwagę potencjalnego wpływu danych odstających. Zastanówmy się więc, w jaki sposób zależność taką można uwzględnić. W tym celu przedstawiony zostanie, opracowany w ramach niniejszej pracy algorytm nazwany VLR (*variable learning rate*).

Opis algorytmu VLR

Najogólniej rzecz ujmując, wielkość kroku algorytmu optymalizacji, uczącego sieć neuronową poprzez poszukiwanie minimum funkcji celu metodą największego spadku, zależy od dwóch czynników: gradientu funkcji celu w danym punkcie, oraz wartości współczynnika uczenia. Można zauważyć, że aby zmniejszyć wpływ pojedynczej danej odstającej na kierunek uczenia sieci trzeba, dla konkretnego residuum, podejrzanego o bycie takim błędem, zmniejszyć którąś z dwóch wymienionych wielkości. Uwzględnienie wpływu pojedynczych danych wymaga oczywiście zastosowania procedury uczenia on-line, zajmijmy się zatem takim procesem uczenia, w którym adaptacja wag odbywa się po prezentacji każdego ze wzorców uczących w sposób następujący: do wagi w_j po zaprezentowaniu i-tego elementu ciągu uczącego dodawana jest wielkość:

$$\Delta w_{ij} = -\eta \frac{\partial \rho\left(r_i\right)}{\partial r_i} \frac{\partial r_i}{\partial w_j}, \tag{4.18}$$

przy czym η oznacza współczynnik uczenia.

Zatem, aby zmienić wielkość kroku algorytmu nie ingerując w postać funkcji celu, rozważyć należy modyfikację η. Jeśli chcemy, by współczynnik uczenia pomagał w dyskryminacji błędów grubych, powinien on przyjmować relatywnie niedużą wartość dla obrazów uczących powodujących największe błędy, aby zmiany wag zależały od nich w niewielki sposób. Wprowadźmy więc zmienny współczynnik uczenia zdefiniowany jako:

$$\eta\left(r_i\right) = \eta_0 \phi\left(r_i\right), \tag{4.19}$$

gdzie η_0 oznacza pewną stałą wartość początkową, natomiast $\phi(r_i)$ jest funkcją kosztu współczynnika. Funkcja ta powinna być symetryczna, posiadać swoje jedyne minimum w zerze, oraz dążyć do zera w nieskończoności. Jako, że jej zadaniem jest zmniejszanie wartości współczynnika uczenia dla dużych r_i, warto dla prostoty przyjąć $\phi(0)=1$, aby $\eta(0) = \eta_0$. Funkcja kosztu może być wtedy, np. kształtu Gaussowskiego:

$$\phi\left(r_i\right) = e^{-\left(\frac{1}{a\sqrt{2}} r_i\right)^2}, \tag{4.20}$$

gdzie $\pm a$ jest punktem przegięcia. Stała ta (dobrana eksperymentalnie jako 0.35) determinuje także przedział obcinający residua podejrzane o bycie błędami grubymi.

Działanie opisanej modyfikacji algorytmu uczenia jest więc bardzo proste: współczynnik uczenia ważony jest w niej przez pewną funkcję błędu, która przyjmuje stosunkowo małe wartości dla dużych argumentów. Jako, że jest to algorytm, w którym aktualizacja wag następuje po prezentacji każdego obrazu uczącego, dlatego funkcja zmniejszająca współczynnik uczenia powinna być stała, gdyż nie można powiązać jej z aktualnymi wartościami rozrzutu błędów. Ewentualnie, można by próbować wprowadzić pewien schemat zmian jej postaci w czasie, ale próby takie nie zostały w niniejszej pracy podjęte.

Algorytm VLR - procedura

Procedura postępowania dla algorytmu VLR może więc zostać zapisana, jak poniżej:

1. Wylosuj wartości początkowe parametrów sieci.
2. Oblicz błąd dla wszystkich wyjść sieci, oraz błędy warstw ukrytych (propagacją wsteczną). Jeśli kryterium stopu spełnione, zakończ uczenie.
3. Oblicz nową wartość współczynnika uczenia $\eta(r_i)$.
4. Wykonaj krok uczenia aktualizując wagi sieci i wróć do punktu drugiego.

Omawiana metoda została przedstawiona bardziej szczegółowo przez autora niniejszej pracy w [55].

Uwagi

Oczywiście opisane powyżej postępowanie niesie za sobą kilka niebezpieczeństw: po pierwsze, proces uczenia uwzględnia również błędy grube, choć przesuwa się w kierunku zależnym od nich bardzo powoli, po drugie zaś, nie ma gwarancji, że zakłócenia procedury minimalizacyjnej nie spowodują jej błędnego działania dla czystych danych uczących. Pierwszy z wymienionych problemów rozwiązać częściowo można przez zastosowanie w procesie uczenia funkcji kryterialnej, która sama wprowadza już częściowe obcinanie danych odstających. Drugi problem jest niestety charakterystycznym zjawiskiem dla wszystkich odpornych algorytmów, choć również i tu, odpowiednia postać funkcji celu zapobiega w większości przypadków rozbieżności

i prowadzi do znalezienia rozwiązania bliskiego właściwemu. Z tego powodu, algorytm z adaptacyjnie dobieranym współczynnikiem uczenia należy używać łącznie ze zmodyfikowaną funkcją celu, najlepiej mającą swój rodowód w odpornych estymatorach. Na potrzeby niniejszej pracy przyjęto przy testowaniu metody, opisaną wcześniej funkcję błędu LMLS (sekcja 3.2.2).

Dodatkowo powstać może pytanie o sens wprowadzania algorytmu pierwszego rzędu, jeżeli, jak wspominano, algorytmy drugiego rzędu działają znacznie efektywniej. Otóż, jak się okazuje, istnieją również sytuacje, w których niezbędne jest wykorzystanie właśnie sposobu uczenia typu "on-line", który w zasadzie determinuje stosowanie metody największego spadku, ewentualnie jej modyfikacji. By wspomnieć, wśród typowych przykładów, filtry adaptacyjne, regulatory neuronowe, przetwarzanie danych strumieniowych i inne zastosowania, w których sieć powinna uczyć się w miarę dostępności informacji o modelowanym procesie.

Warto jednak zauważyć, że przy trochę zmienionej definicji problemu, kiedy dostępne staje się przechowywanie pewnej ilości prezentowanych danych uczących w ograniczonym buforze, możliwe by było zastosowanie w wymienionych sytuacjach również algorytmów z uczeniem skumulowanym. Modyfikacja wag sieci i obliczenie błędów, mogłyby bowiem mieć miejsce po zaprezentowaniu owej stałej ilości elementów uczących, co z kolei pozwoliłoby na stosowanie metod uczenia drugiego rzędu.

Podsumowanie algorytmu VLR

Odporny algorytm VLR w prosty sposób wykorzystuje zmianę współczynnika uczenia do niwelowania wpływu błędów grubych. Jest on przeznaczony do uczenia typu "obraz po obrazie", nie może być więc stosowany z metodami drugiego rzędu.

4.5 Odporny algorytm uczenia drugiego rzędu ze zmiennym współczynnikiem uczenia

Wprowadzenie

Przedstawiony wcześniej odporny algorytm uczenia sieci ze zmiennym współczynnikiem uczenia, pomimo swojej wykazanej empirycznie odporności na błędy w danych jest, niestety, mało praktyczny w użyciu. Wynika to przede wszystkim z faktu, że w opisanym sposobie uczenia korzystać należy z metody największego spadku, a

modyfikacja wag następować musi po każdej prezentacji obrazu uczącego. Jak wiadomo, znacznie szybciej działają procedury uczenia oparte na algorytmach optymalizacyjnych drugiego rzędu, które muszą być łączone z aktualizacją wag dopiero po prezentacji całego ciągu uczącego.

Pewną przeszkodą w przeniesieniu idei uodparniania sieci za pomocą zmiennego współczynnika uczenia na algorytmy typu "batch-learning" jest fakt, że taki sposób postępowania uniemożliwia analizę poszczególnych elementów ciągu uczącego, pod względem powodowanych przez nie błędów.

Warto jednak zauważyć, że również metody opierające się na zmianie kryterium uczenia de facto nie biorą pod uwagę konkretnych residuów, lecz błąd całkowity dla całego ciągu uczącego. W ich przypadku lepsze, jeśli rozważyć odporność, rezultaty uzyskuje się przy metodzie uczenia on-line, choć, jak widać, również drugi ze sposobów uczenia zmniejsza wrażliwość na błędy w danych. Dlatego zdecydowano się na zaproponowanie metody uczenia opartej na algorytmie drugiego rzędu, w której współczynnik uczenia dobierany jest tak, aby dane odstające miały jak najmniejszy wpływ na proces optymalizacji. Algorytm ten nazwano ALR (*adaptive learning rate*).

Opis algorytmu ALR

W metodzie tej współczynnik kroku znów związany jest z zadaną funkcją zależną od błędów popełnianych przez sieć. Sytuacja różni się tu jednak tym, że w każdej epoce uczenia współczynnik ów wyznaczany jest na podstawie minimalizacji podanej funkcji w kierunku wyznaczonym przez główny algorytm uczenia. I tak, dla danego kierunku p_t w epoce t szukane jest:

$$\eta_t = \min_{\eta} E(\vec{w_t} + p_t \eta(\vec{r})). \tag{4.21}$$

Minimalizacja jest tu dokonywana po wartościach pewnej funkcji całego wektora błędów $\eta(\vec{r})$, która może być dana np. przez zależność (4.20). Istotne jest, aby jej postać odzwierciedlała założenie dotyczące błędów grubych, innymi słowy zmniejszała wpływ dużych residuów. Metoda ta nie spowalnia tak procesu optymalizacji, jak opisana poprzednio (sekcja 4.4), ponadto można ją stosować z różnymi algorytmami poszukiwania kierunku (w niniejszej pracy testowana była w połączeniu z algorytmem gradientów sprzężonych).

Z kolei poszukiwanie minimum funkcji w obliczonym kierunku odbywać się może bez obliczania gradientu funkcji (w pracy za pomocą aproksymacji wielomianem drugiego stopnia). Dodatkowo sposób ten nie powoduje "zbaczania" algorytmu w stronę niewłaściwych ekstremów, dlatego też nie wymaga modyfikowania funkcji celu na jedną z postaci mających zapewnić odporność uczenia. Oczywiście zastosowanie jednej z takich funkcji (np. danej zależnością (3.12)) może również polepszyć działanie metody, gdyż błędy grube brane są wtedy pod uwagę już na etapie wyznaczania kierunku poszukiwań, choć jednocześnie mogłoby to skutkować ograniczaniem wpływu na proces uczenia zbyt dużej ilości obrazów uczących.

Algorytm ALR - procedura

Algorytm z adaptacyjnie dobieranym współczynnikiem uczenia ALR, przedstawia się w skrótowym zapisie następująco:

1. Wylosuj wartości początkowe parametrów sieci.
2. Oblicz błąd dla wszystkich wyjść sieci, oraz błędy warstw ukrytych (propagacją wsteczną). Jeśli kryterium stopu spełnione, zakończ uczenie.
3. Znajdź kierunek zmiany wag.
4. Wykonaj minimalizację kierunkową, aby otrzymać wartość współczynnika uczenia η_t, wykonaj krok uczenia, wróć do punktu 2.

Podsumowanie algorytmu ALR

Nowy odporny algorytm uczenia ALR wykorzystuje mechanizm zmian współczynnika uczenia w celu zmniejszania wpływu błędów grubych na model tworzony przez sieć. W każdej epoce uczenia przeprowadzana jest minimalizacja pewnej funkcji błędu, na podstawie której dobierana jest nowa wartość współczynnika uczenia.

Metoda ta przeznaczona jest do stosowania z uczeniem typu skumulowanego, można ją również łączyć z uodpornioną funkcją kryterialną.

4.6 Odporny algorytm uczenia ze zmienną funkcją aktywacji neuronów

Wprowadzenie

Jak już wspomniano wcześniej, krok algorytmu uczenia sieci neuronowej zależny jest od wartości gradientu funkcji celu w danym punkcie, oraz wielkości współczynnika uczenia. Opisano już na poprzednich stronach (sekcja 4.4 i 4.5) w jaki sposób zmodyfikowany może zostać współczynnik uczenia, aby jakość działania algorytmu w obecności dużych zakłóceń, była lepsza niż w metodzie klasycznej. Jeżeli nie chce się ingerować w funkcje błędu minimalizowaną przez algorytm, pozostaje jeszcze zmiana kształtu funkcji aktywacji. Na tej właśnie zasadzie opiera się proponowany w niniejszej pracy odporny algorytm ATF (*adaptive transfer function*).

Zastosowana w strukturze sieci neuronowej postać funkcji aktywacji neuronów może, przynajmniej teoretycznie, zmieniać jej odporność na błędy grube pojawiające się w danych. Jeżeli przyjmiemy, że funkcje aktywacji mają kształt zbliżony do skokowego (przy zachowaniu różniczkowalności), nietrudno zauważyć, że szybkość przebiegu procesu uczenia zależeć będzie od wielkości obszaru, poza którym neurony wchodzą w nasycenie, jak i od tego, jak strome jest zbocze funkcji. Różnica w tolerancji błędów grubych w danych uczących powinna być więc dostrzegalna już przy porównaniu sieci wykorzystujących funkcje sigmoidalne i oparte na tangensie hiperbolicznym. Jak pokazały symulacje, rzeczywiście tak się dzieje, a więc obserwować możemy, że w zależności od tego, którą z dwóch najpopularniejszych funkcji aktywacji zastosujemy, zmienia się wrażliwość sieci na dane odstające pojawiające się w trakcie uczenia. Zmiana ta jest jednak bardzo niewielka, a przy małej ilości porównywanych przykładów testowych, praktycznie niezauważalna. Wynika to z faktu, iż obie funkcje są do siebie podobne w przedziale pomiędzy obszarami nasycenia, a ich pochodne poza nim, tak naprawdę, też niewiele od siebie odbiegają.

Opis algorytmu ATF

Aby zatem osiągnąć wyraźniejszą poprawę w odporności sieci, można zastosować zmienną w czasie, a więc zależną od postępów uczenia, funkcję aktywacji. Zaproponowano sparametryzowanie postaci funkcji poprzez wprowadzenie dodatkowej wielkości odpowiadającej za jej zmianę w miarę przebiegu algorytmu uczenia:

$$f(x) = \frac{1}{1 + \exp(-\beta x)}, \tag{4.22}$$

gdzie

$$\beta = \frac{1}{1 + c * t^2}, \tag{4.23}$$

przy czym c oznacza pewną niewielką stałą, która została eksperymentalnie dobrana jako $c = 5 * 10^{-4}$, natomiast t to numer epoki procesu uczenia. Jak można zauważyć, postać funkcji aktywacji neuronów zmienia się tu adaptacyjnie w miarę postępu procesu uczenia sieci, w bezpośredniej zależności od przebytych epok (iteracji) algorytmu uczenia. Zmiana ta polega na "rozciągnięciu" funkcji wzdłuż jej dziedziny tak, by jej pochodna pomiędzy obszarami nasycenia zmniejszała się w czasie. Skutkuje to wprowadzeniem dodatkowego tłumienia gradientu, a co za tym idzie również kroku procedury optymalizacyjnej. Negatywnym efektem tego zjawiska może być przesunięcie się obszarów nasycenia ku nieskończoności, ale jako, że dzieje się to dopiero w końcowej fazie uczenia,w praktyce problem taki nie istnieje.

Dzięki temu, że funkcja aktywacji uzależniona została od czasu, nie zaś od aktualnego błędu, zależności dotyczące algorytmu uczenia sieci, nie ulegają zmianie. Dotyczy to w szczególności również kierunku, jak i wielkości kroku procedury minimalizacyjnej, co pozwala na nieskomplikowaną implementację opisanej modyfikacji dla różnych sposobów uczenia sieci sigmoidalnych.

Algorytm ATF - procedura

Dla algorytmu ATF procedurę postępowania można zapisać następująco:

1. Wylosuj wartości początkowe parametrów sieci.
2. Oblicz błąd dla wszystkich wyjść sieci, oraz błędy warstw ukrytych (propagacją wsteczną). Jeśli kryterium stopu spełnione, zakończ uczenie.
3. Oblicz nowe postacie funkcji aktywacji.

4. Wykonaj krok uczenia i wróć do punktu 2.

Uwagi

Analogicznie jak to miało miejsce w przypadku algorytmu z adaptacyjnie dobieranym współczynnikiem uczenia, również przy stosowaniu zmiennej w czasie funkcji aktywacji neuronów proponowana modyfikacja może doprowadzić do niepożądanych zakłóceń procesu minimalizacji, w związku z tym należy ją stosować łącznie z jedną z funkcji kryterialnych opartych na odpornych M-estymatorach. W niniejszej pracy posłużono się więc opisaną wcześniej funkcją LMLS (3.12).

Warto w tym miejscu dodać, iż zmienna sparametryzowane w ten sposób funkcje aktywacji neuronów, odpowiadają w pewnym stopniu zmianom współczynnika uczenia, gdyż ich celem również jest zmiana rozmiaru kroku algorytmu uczącego dla większych residuów. Niemniej jednak wykorzystywany tu mechanizm jest o tyle inny, że polega na modyfikacji różnych elementów algorytmu. Jakie ma to znaczenie, opisano przy rozważaniach tyczących się nieparametrycznego ujęcia sieci neuronowych podczas omawiania potencjalnych dalszych kierunków badań.

Podsumowanie algorytmu ATF

Zaproponowany odporny algorytm ATF wykorzystuje zmianę kształtu funkcji aktywacji do zmniejszania wpływu błędów grubych na proces uczenia. Zmiany nachylenia funkcji przeprowadzane są zgodnie z zadanym schematem zależnym od aktualnych postępów uczenia sieci. Metoda ta może być stosowana z uczeniem typu skumulowanego i algorytmami drugiego rzędu.

4.7 Odporny algorytm dedykowany dla kryterium LMLS

Wprowadzenie

Idea tego algorytmu, zwanego dalej w skrócie DLMLS (*dedicated to least mean log squares* oparta jest na opisanym wcześniej algorytmie Levenberga-Marquardta (sekcja 2.2.3), który w istotny sposób przyspiesza proces uczenia sieci neuronowych w porównaniu z algorytmem największego spadku, bez konieczności obliczania macierzy hesjanu. W metodzie L-M założono, że minimalizowana funkcja błędu ma

postać kwadratową i na tej podstawie określono zależności umożliwiające uzyskanie gradientu i aproksymacji hesjanu w zależności od macierzy jakobianu. Założenie to uniemożliwia niestety połączenie algorytmu L-M z odpornymi kryteriami będącymi podstawą działania odpornych metod uczenia sieci, co sprawia, że w pracy tej stosowano najczęściej algorytm gradientów sprzężonych, aby nie ograniczać się do historycznej już dziś metody największego spadku.

Opis algorytmu DLMLS

Jak się jednak okazuje, podobne jak w metodzie L-M rozumowanie przeprowadzić można również dla wspominanej już wielokrotnie funkcji danej zależnością (3.12). Kluczowe jest tu znalezienie odpowiednika macierzy jakobianu, oraz zależności pozwalającej na aproksymowanie hesjanu funkcji kryterialnej. Zależność taka powinna być stosunkowo prosta i nie może wymagać obliczania wyższych pochodnych. Analogicznie, jak we wzorze (2.15) zdefiniujmy więc funkcję celu jako:

$$E(\vec{w}) = \sum_{k=1}^{N}\sum_{i=1}^{m} \log(1 + \frac{1}{2} {r_{ki}}^2(\vec{w})), \tag{4.24}$$

gdzie, dla przypomnienia, m jest ilością wyjść sieci, N liczbą obrazów uczących.

Dla wektora błędów:

$$r(\vec{w}) = \left[\begin{array}{cccccccccc} r_{11} & \dots & r_{N1} & r_{12} & \dots & r_{N2} & \dots & r_{1m} & \dots & r_{Nm} \end{array} \right]^T \tag{4.25}$$

wprowadźmy teraz pomocniczą macierz p:

$$p(\vec{w}) = \left[\begin{array}{ccccc} \frac{r_{11}}{1+\frac{1}{2}{r_{11}}^2} & \frac{r_{21}}{1+\frac{1}{2}{r_{21}}^2} & \frac{r_{31}}{1+\frac{1}{2}{r_{31}}^2} & \dots & \frac{r_{Nm}}{1+\frac{1}{2}{r_{Nm}}^2} \end{array} \right]^T. \tag{4.26}$$

Jak widać powstała ona po nieskomplikowanych algebraicznych przekształceniach dokonanych na wektorze residuów. Pamiętając, że jakobian zdefiniowany jest zależnością 2.16, możemy więc zapisać gradient funkcji celu:

$$\nabla E(\vec{w}) = J(\vec{w})^T p(\vec{w}). \tag{4.27}$$

Warto zwrócić uwagę, że dzięki temu otrzymaliśmy zależność dokładną.

Zapiszmy teraz drugą macierz pomocniczą:

$$q(\vec{w}) = \left[\begin{array}{ccccc} \frac{1}{1+\frac{1}{2}{r_{11}}^2} - \frac{{r_{11}}^2}{(1+\frac{1}{2}{r_{11}}^2)^2} & \frac{1}{1+\frac{1}{2}{r_{21}}^2} - \frac{{r_{21}}^2}{(1+\frac{1}{2}{r_{21}}^2)^2} & \dots & \frac{1}{1+\frac{1}{2}{r_{Nm}}^2} - \frac{{r_{Nm}}^2}{(1+\frac{1}{2}{r_{Nm}}^2)^2} \end{array} \right]^T. \tag{4.28}$$

Znów wykonane zostały jedynie operacje arytmetyczne na elementach wektora r. Na tej podstawie zdefiniujmy dodatkową macierz JQ:

$$JQ(\vec{w}) = \begin{bmatrix} \frac{\partial r_{11}}{\partial w_1} q_{11} & \frac{\partial r_{11}}{\partial w_2} q_{11} & \cdots & \frac{\partial r_{11}}{\partial w_n} q_{11} \\ \frac{\partial r_{21}}{\partial w_1} q_{22} & \frac{\partial r_{21}}{\partial w_2} q_{22} & \cdots & \frac{\partial r_{21}}{\partial w_n} q_{22} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial r_{N1}}{\partial w_1} q_{N1} & \frac{\partial r_{N1}}{\partial w_2} q_{N1} & \cdots & \frac{\partial r_{N1}}{\partial w_n} q_{N1} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial r_{1m}}{\partial w_1} q_{1m} & \frac{\partial r_{1m}}{\partial w_2} q_{1m} & \cdots & \frac{\partial r_{1m}}{\partial w_n} q_{1m} \\ \frac{\partial r_{2m}}{\partial w_1} q_{2m} & \frac{\partial r_{2m}}{\partial w_2} q_{2m} & \cdots & \frac{\partial r_{2m}}{\partial w_n} q_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial r_{Nm}}{\partial w_1} q_{Nm} & \frac{\partial r_{Nm}}{\partial w_2} q_{Nm} & \cdots & \frac{\partial r_{Nm}}{\partial w_n} q_{Nm} \end{bmatrix}, \tag{4.29}$$

która powstała przez pomnożenie każdego wiersza jakobianu przez odpowiadający mu element $q(\vec{w})$.

Na tej podstawie, można pokazać, że macierz hesjanu, dana jest w tym przypadku równaniem następującym:

$$H(\vec{w}) = J(\vec{w})^T JQ(\vec{w}) + S(\vec{w}). \tag{4.30}$$

Uwaga. Reszta S, którą pominiemy w procedurze przybliżonej zależy bezpośrednio od drugich pochodnych i zapisana może być jako:

$$S(\vec{w}) = \sum_{k=1}^{N} \sum_{i=1}^{m} p_{ki}(\vec{w}) \nabla^2 r_{ki}(\vec{w}) \tag{4.31}$$

W procedurze zastosujemy czynnik regularyzacyjny u_k, otrzymując ostateczną przybliżoną zależność opisującą hesjan:

$$H(\vec{w}) = J(\vec{w})^T JQ(\vec{w}) + u1. \tag{4.32}$$

W zależności powyższej, podobnie, jak w algorytmie L-M, aby uzyskać aproksymowaną wartość hesjanu, mnoży się dwie, oparte w prosty sposób na jakobianie, macierze, a następnie dodaje do nich dodatkowy składnik. Zmiana wag następuje według zależności:

$$\vec{w}_{t+1} = \vec{w}_t - \left[J(\vec{w}_t)^T JQ(\vec{w}_t) + u_t 1 \right]^{-1} \nabla E(\vec{w}_t), \tag{4.33}$$

przy czym t oznacza epokę procesu uczenia. Czynnik regularyzacyjny zmienia się tu natomiast w sposób opisany wcześniej, przy omawianiu algorytmu L-M, a więc maleje w przebiegu całej metody, wzrastając tylko w ramach jednej epoki aż do osiągnięcia poprawy funkcji celu.

Uwagi

Trzeba zwrócić uwagę, że przybliżenie funkcji celu jest tu podobne, jak w przypadku algorytmu L-M, gdyż powstaje tu dodatkowa niedokładność związana z rozwinięciem oszacowania hesjanu ($S(\vec{w})$), nadal jednak można zauważyć, że jest ono wystarczająco dobre do prawidłowego przebiegu procesu uczenia.

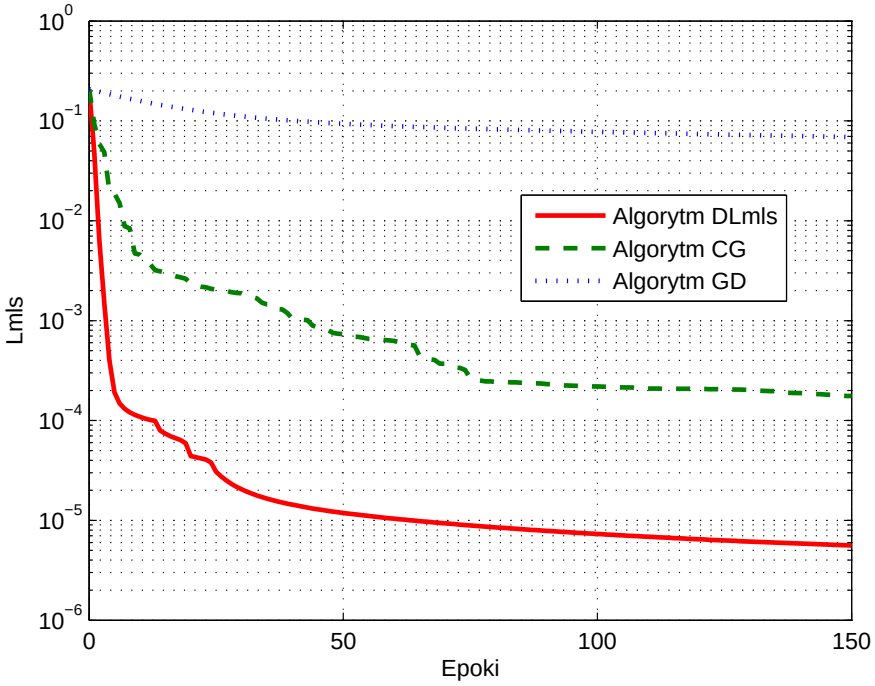
Jak pokazały symulacje, odporny algorytm dedykowany do kryterium LMLS rzeczywiście znacznie przyspiesza uczenie sieci, zwłaszcza jeśli porówna się go z pierwszorzędowym algorytmem największego spadku. W ramach niniejszej pracy zaproponowano także pewne dalej idące uproszczenia w szacowaniu hesjanu. Pierwsze z nich, to przybliżone oszacowanie elementów macierzy q. Dane ona była zależnością (4.28), czyli można jej elementy zapisać jako:

$$q_{ki} = \frac{1}{1 + \frac{1}{2}{r_{ki}}^2} - \frac{{r_{ki}}^2}{(1 + \frac{1}{2}{r_{ki}}^2)^2}. \tag{4.34}$$

Jeżeli przyjmiemy, że wartości r są dostatecznie małe (a założenie takie jest o tyle słuszne, że dla dużych błędów najczęściej czynnik regularyzacyjny jest stosunkowo duży, a więc metoda opiera się tylko na gradiencie), wtedy pierwszy człon równania jest w przybliżeniu równy jeden, a więc q upraszcza się do postaci:

$$q_{ki} = 1 - \frac{{r_{ki}}^2}{(1 + \frac{1}{2}{r_{ki}}^2)^2} = 1 - {p_{ki}}^2. \tag{4.35}$$

Wspomnieć tu należy o jeszcze jednej, istotnej z punktu widzenia uczenia z zakłóceniami, sprawie. Wydaje się mianowicie, że większa niedokładność może, w przypadku uczenia przy udziale błędów grubych, okazać się zaletą, gdyż wolniejsze uczenie w pobliżu rozwiązania może pomagać w nie uwzględnianiu błędów powodowanych przez duże zakłócenia. Jednak problem ten jest na tyle skomplikowany, że w niniejszym algorytmie skupiono się jedynie na efektywnej minimalizacji funkcji kryterialnej, bez prób osiągnięcia odporności przez świadome niedokładności w szacowaniu macierzy drugich pochodnych.,

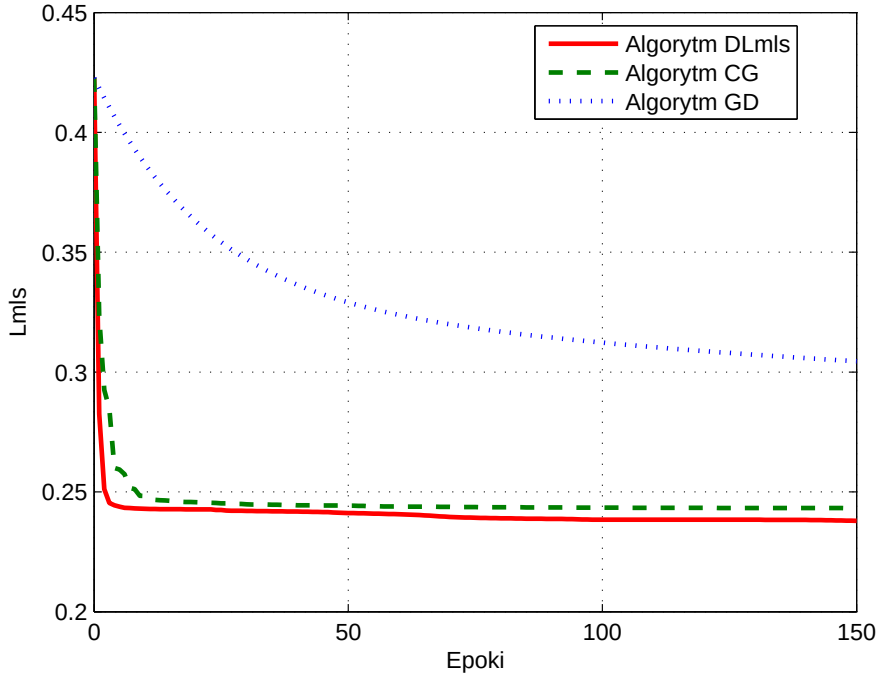


Rysunek 4.1: Porównanie przebiegu uczenia, przy kryterium LMLS, dla algorytmu największego spadku (GD), gradientów sprzężonych (CG), oraz dedykowanego algorytmu DLMLS (dane uczące bez błędów grubych).

Na rysunkach 4.1 i 4.2 pokazano przykładowy przebieg procesu uczenia algorytmami największego spadku, gradientów sprzężonych, oraz DLMLS, na danych czystych i zawierających zakłócenia.

Podsumowanie algorytmu DLMLS

Odporny algorytm dedykowany dla kryterium LMLS, jest algorytmem drugiego rzędu opierającym się na optymalizacji wykorzystującej przybliżoną wartość hesjanu. Dla funkcji kryterialnej LMLS pozwala on na przyspieszenie procesu uczenia w porównaniu z metodą największego spadku. Opracowana w niniejszej pracy metoda jest odpornym odpowiednikiem, przeznaczonego do minimalizacji błędu średniokwadratowego, algorytmu Levenberga-Marquardt'a.



Rysunek 4.2: Porównanie przebiegu uczenia, przy kryterium LMLS, dla algorytmu największego spadku (GD), gradientów sprzężonych (CG), oraz dedykowanego algorytmu DLMLS (dane z błędami).

4.8 Odporny algorytm uczenia ze wstępną analizą danych

Wprowadzenie

Jak już wspomniano, w niniejszej pracy nie zajmowano się odpornymi metodami statystycznymi mającymi na celu analizę danych uczących przed wykorzystaniem ich do trenowania sieci neuronowych. Powód takiego postępowania był oczywisty: sposoby owe są bowiem uniwersalne w tym sensie, że nie biorą pod uwagę przeznaczenia danych, a ich jedynym celem jest identyfikacja zakłóceń i punktów, co do których zakłada się, że są danymi o prawidłowej zawartości, zatem trudno mówić, że istnieje ich bezpośredni związek z procesem uczenia sieci. Ponadto założono na wstępie, że odrzucanie błędnych danych ma się odbywać już w samych algorytmach uczących, co oczywiście nie ma miejsca w przypadku prostego zastosowania odpornych metod statystycznych.

W tym paragrafie zostanie jednak pokrótce przedstawiona metoda, która, co prawda, opiera się na wstępnej analizie danych, ale uzyskane za jej pomocą wyniki stosowane są bezpośrednio w algorytmie uczącym.

Estymator MCD

Do wstępnego przetworzenia danych uczących posłużono oparto się tutaj na estymatorze MCD (*Minimum Covariance Determinant*), który zaproponowany został w pracy [50], a dokładniej opisany, m.in. w [39]. Jak sama nazwa wskazuje, w estymatorze tym poszukiwany jest w zbiorze wszystkich N danych postaci $x_i = (x_{i1}, \ldots, x_{in})$, podzbiór $h > N/2$ obserwacji taki, że jego macierz kowariancji posiada najmniejszy wyznacznik. Zachowany tu musi być oczywiście również warunek $n < h$, aby macierze kowariancji były nieosobliwe, oraz $N/n > 5$, aby uwzględnić przypadek próbek koplanarnych. Dla znalezionego podzbioru oblicza się następnie klasyczne estymatory: średnią $\hat{\mu}$ i macierz kowariacji $\hat{\Sigma}$, które służą do zidentyfikowania centrum i rozproszenia danych czystych, obliczenia odpowiednich odległości i oddzielenia danych odstających.

Podobnie jak dla estymatora LTS największą teoretyczną odporność na duże zakłócenia w danych, uzyskuje się przyjmując $h = [(N + n + 1)/2]$, co naturalnie powoduje pogorszenie zbieżności estymatora dla przypadku rozkładu normalnego.

Największa trudność przy konstruowaniu estymatora MCD polega tu na efektywnej minimalizacji macierzy kowariancji po wszystkich h-elementowych podzbiorach. Na szczęście, opracowany został również odpowiedni algorytm, który pomaga przyspieszyć ten proces [53] i nie wymaga przejrzenia wszystkich możliwych podzbiorów, niemniej zadanie pozostaje złożone obliczeniowo.

Po znalezieniu odpowiedniego podzbioru danych, oraz obliczeniu estymatorów $\hat{\mu}$ i $\hat{\Sigma}$ można już zająć się identyfikacją danych odstających poprzez zmierzenie ich odległości od środka zbioru danych czystych. Robi się to poprzez obliczenie odpornej odległości (*Robust Distance*) [43] dla każdego elementu:

$$RD_i = \sqrt{(x_i - \hat{\mu})^T \hat{\Sigma}^{-1} (x_i - \hat{\mu})}, \tag{4.36}$$

który jak widać jest prostym uodpornieniem znanej odległości Mahalanobisa:

$$MD_i = \sqrt{(x_i - \bar{x})^T S^{-1} (x_i - \bar{x})}, \tag{4.37}$$

gdzie S oznacza macierz kowariancji całej próby. Po obliczeniu poszczególnych odległości dla wszystkich elementów zbioru, identyfikuje się jako dane odstające punkty, których odporna odległość jest większy niż pewna zadana wartość, a więc leżą dalej niż maksymalna dopuszczalna odległość od centrum zbioru danych uznanych za czyste.

Opis algorytmu ze wstępną analizą danych

Powyższą procedurę można, po pewnych modyfikacjach, przenieść na grunt uczenia sieci neuronowych, w celu skonstruowania algorytmu bardziej odpornego na błędy w danych. W niniejszej pracy zaproponowano, aby przed rozpoczęciem właściwego algorytmu uczenia obliczyć dla wszystkich obrazów uczących współczynniki wagowe użyte później do konstrukcji funkcji kryterialnej. Stwórzmy więc macierz z składającą się z wartości x podawanych na wejście sieci, oraz odpowiadających im żądanych wartości wyjść t w taki sposób, że:

$$z_i = \left[\begin{array}{cccccccc} x_{i1} & x_{i2} & \ldots & x_{ip} & t_{i1} & t_{i2} & \ldots & t_{ik} \end{array} \right], \tag{4.38}$$

gdzie $i = 1 \ldots N$. Dla macierzy z obliczamy estymator MCD, natomiast dla poszczególnych $(p+k)$-elementowych wektorów (w praktyce najczęściej $k = 1$), otrzymanych na podstawie parametrów MCD, liczymy odporne odległości dane zależnością (4.36). Takie postępowanie powinno umożliwić wykrycie danych odstających zarówno pod względem wartości wejść, jak i targetów. W niniejszej pracy przekazanie uzyskanej w ten sposób informacji do algorytmu uczenia odbywa się poprzez wprowadzenie opartych na RD współczynników wagowych:

$$u_i = \min\{1, \frac{\chi^2_{0.975,p+k}}{RD_i^2}\}, \tag{4.39}$$

gdzie wartość statystyki χ^2 wybrano przy założeniu normalności rozkładu danych. Minimalizowana w procesie uczenia sieci funkcja celu przyjmuje postać średniej ważonej:

$$E = \frac{1}{N}\sum_{i=1}^{N} u_i \rho\,(r_i), \tag{4.40}$$

przy czym ρ może być zarówno funkcją kwadratową, jak i jedną z funkcji "odpornych" opartych na jednokrokowych M-estymatorach.

Warto zauważyć, że w przeciwieństwie do opisanej wcześniej metody statystycznej, nie tracimy tu informacji niesionych przez obrazy uczące podejrzane o bycie błędami, natomiast ich wpływ na kierunek uczenia sieci jest zmniejszany.

Algorytm ze wstępną analizą danych - procedura

Dla opracowanego algorytmu ze wstępną analizą danych, możemy więc przedstawić całą procedurę w sposób następujący:

1. Wylosuj wartości początkowe parametrów sieci.
2. Znajdź estymator MCD zbioru danych uczących, oblicz odległości i wagi dla każdego obrazu uczącego.
3. Oblicz błąd dla wszystkich wyjść sieci, oraz błędy warstw ukrytych (propagacją wsteczną). Jeśli kryterium stopu spełnione, zakończ uczenie.
4. Wykonaj krok uczenia i wróć do punktu 3.

Podsumowanie algorytmu ze wstępną analizą danych

Algorytm ze wstępną analizą danych opiera się na wyryciu obrazów podejrzanych o bycie błędami grubymi, przed rozpoczęciem właściwego procesu uczenia. Każdemu elementowi ciągu uczącego przyznawana jest pewna waga, która następnie wykorzystywana jest przy konstruowaniu funkcji kryterialnej.

Metoda przeznaczona jest do uczenia skumulowanego i może być łączona z algorytmami drugiego rzędu.

Rozdział 5

Metodyka badania odporności na błędy w danych

Jako, że w ramach tworzenia niniejszej pracy przeprowadzono bardzo wiele eksperymentów symulacyjnych, postarano się możliwie dokładnie przedstawić całą procedurę mającą na celu testowanie opracowanych algorytmów uczenia.

5.1 Sformułowanie problemu badania odporności algorytmów

Przetestowanie zaproponowanych odpornych algorytmów uczenia sieci neuronowych wymagać musi ustalenia procedury postępowania mającej na celu porównanie jakości działania poszczególnych metod. Wbrew pozorom zadanie to jest dość złożone, ze względu na ilość parametrów niezbędną do ustalenia w przypadku badania sztucznych sieci neuronowych.

Na początek należy uzmysłowić sobie dokładnie, jakie cechy analizowanych algorytmów mają priorytetowe znaczenie z punktu widzenia odporności na błędy w danych uczących, a co za tym idzie jaka będzie miara jakości ich działania. Otóż, przypomnieć trzeba, że odporne metody uczenia mają za zadanie, po pierwsze, uczyć sieć możliwie jak najlepiej w sytuacji, gdy w danych uczących pojawiają się duże zakłócenia. Po drugie, ich działanie nie powinno zbyt mocno odbiegać od działania klasycznych metod również dla obrazów uczących czystych i pozbawionych zakłóceń. Po trzecie wreszcie, choć nie jest to najistotniejsze, dobrze by było, gdyby algorytmy takie wymagały możliwie niewielkich zmian w typowej metodzie uczenia. Warto dodać, że sformułowanie "dobrze nauczyć sieć" oznacza po prostu, że sieć nauczona

danym algorytmem działa zgodnie z założeniami, innymi słowy właściwie modeluje zadaną zależność.

To ostatnie stwierdzenie implikuje oczywisty, być może, fakt, że badanie odpornych algorytmów uczenia w rzeczywistości powinno sprowadzać się do badania jakości działania wytrenowanych przez nie sieci. Tak też uczyniono w niniejszej pracy: miarą tego, jak działa dana metoda uczenia, było to, jakie wyniki uzyskuje się testując sieć nauczoną za jej pomocą na danych różnych typów. Przy ogólnej ocenie metod nie brano pod uwagę jedynie szybkości działania, gdyż nietrudno wyobrazić sobie przypadek, w którym właściwe rezultaty są ważniejsze od czasu, w jakim zostały uzyskane. Z drugiej strony, bardzo duża złożoność obliczeniowa algorytmu może prowadzić do tego, że staje się on niepraktyczny, dlatego też wybierając odporną metodę trzeba mieć na względzie nakład obliczeniowy potrzebny do jej realizacji. W większości proponowanych metod nie zmienia się on znacznie, gdy jednak do tego dochodzi jest to zaznaczone przy opisie metody.

Podsumowując, cel eksperymentów numerycznych przedstawiał się więc następująco:

- przebadanie odpornych na błędy w danych algorytmów uczenia sieci,
- opisanie i demonstracja ich wad i zalet w porównaniu z algorytmem klasycznym i innymi algorytmami odpornymi.

Pomimo jasno przedstawionego celu, należy się dodatkowo zastanowić nad problemami dotyczącymi owych eksperymentów, które w skrócie zapisać można jako:

- problem parametrów, czyli co, na jakich zadaniach, przy jakich warunkach początkowych symulować;
- jaką przyjąć ilość prób eksperymentalnych i przy jakich założeniach;
- jak opracować zebrane wyniki i jakie przyjąć kryteria oceny algorytmów.

Warto zdać sobie sprawę, ile parametrów ustalić należy dla pojedynczej symulacji, czyli jednego nauczenia i przetestowania sieci, aby w pełni zobrazować potencjalną ilość różnych wariantów przeprowadzenia podobnych badań. Z grubsza można owe zmienne pogrupować jako dotyczące struktury sieci, zadania testowego, algorytmu uczenia, sposobu symulacji i przetwarzania wyników.

Struktura sieci

Jako, że opracowane algorytmy przeznaczone są do uczenia sieci jednokierunkowych sigmoidalnych, zawęża to problem doboru sieci do szczególnej ich klasy. Pomimo to, problemem pozostaje rozmiar testowanych sieci, a więc ilość warstw ukrytych, ich wielkość (przy założeniu, że rozmiary warstw zewnętrznych determinowane są przez specyfikę zadań testowych), oraz użyte funkcje aktywacji neuronów (z wyjątkiem algorytmu ze zmienną funkcją aktywacji).

Zadanie testowe

Niezbędny jest wybór jednego z możliwych, opisanych wcześniej, typów zastosowań SSN, takich jak aproksymacja funkcji, rozpoznawanie wzorców [32], klasyfikacja, etc. Następnie konieczne staje się dobrania parametrów konkretnego zadania, czyli np., jaką funkcję aproksymować, w ilu wymiarach, na jakim obszarze, jak próbkowanym, przy znormalizowanych wartościach, czy też wartościach rzeczywistych. Gdy zadanie zostanie już właściwie sformułowane, trzeba zająć się modelem zakłóceń, które zostaną wprowadzone do danych uczących. W szczególności ustalić należy, jakiego typu są zakłócenia, w jakiej ilości się pojawiają, jaki jest ich rozkład i czy dotyczą zmiennej zależnej, czy niezależnej.

Algorytm uczenia

Dla niektórych metod konieczny jest wybór pomiędzy uczeniem skumulowanym, a typu "on-line", choć dla większości z nich jest to zdeterminowane przez sam sposób działania metody. Jeżeli nie jest to określone trzeba również podjąć decyzję dotyczącą algorytmu minimalizacji (pierwszego, czy drugiego rzędu) i ustalić jego parametry, przykładowo: kryteria stopu, minimalny krok, błąd, gradient, ilość epok, parametry początkowe, metody poszukiwań w kierunku, etc. Z kolei dla sieci konieczny jest wybór wag początkowych, które mogą być ustalone lub w jakiś sposób losowane.

Przebieg symulacji

W tym punkcie niezbędne staje się techniczne podejście do problemu: trzeba zdecydować, w jaki sposób zakłócenia będą wprowadzane do danych uczących, gdyż dziać

się to może poprzez losowanie ich z zadanym prawdopodobieństwem, lub używanie zawsze pewnej zadanej ilości zakłóceń. Dalej trzeba dobrać odpowiednio dane testujące (zwykle różniące się od czystych danych uczących), oraz ustalić, czy parametry, takie jak wagi początkowe i wartości zakłóceń będą takie same dla wszystkich przebiegów uczenia, czy za każdym razem losowane od nowa. Na koniec potrzebne jest wytypowanie ilości powtórzeń symulacji jednego typu, których wyniki będą potem uśredniane, a więc np. dla jednego rozmiaru sieci, zadania testowego i typu, oraz ilości zakłóceń.

Przed rozpoczęciem symulacji niezbędna jest naturalnie decyzja dotycząca tego, jakie dane podlegają akwizycji, przy czym w najbardziej ogólnym przypadku mogą to być: odpowiedź sieci, jej wagi początkowe i końcowe, parametry algorytmu, dane uczące, ewentualnie dane (błędy, lub wagi) dotyczące przebiegu procesu uczenia.

Opracowanie danych eksperymentalnych

Przy założeniu, że porównywanym kryterium jest jakość działania sieci dla danych testowych, należy zdefiniować konkretny typ błędu (np. mse, rms), który posłuży do liczbowych porównań. Ponadto trzeba wprowadzić pewien sposób uśredniania, oraz zdecydować jakiego typu wyniki będą prezentowane (np. średnia, odchylenie standardowe) i w jaki sposób (tabele, wykresy).

5.2 Sposób przeprowadzania symulacji

Na podstawie powyższej analizy opracowano procedurę testowania algorytmów, w której miarą jakości ich działania były błędy popełniane przez nauczone za ich pomocą sieci. Schemat przeprowadzania symulacji przedstawiał się w niej dla konkretnego algorytmu następująco:

1. Ustal strukturę sieci (ilość warstw, neuronów, funkcje aktywacji), zadanie testowe (w tym ilość i typ zakłóceń) i parametry algorytmu.

2. Jeśli maksymalna ilość powtórzeń eksperymentu nie została przekroczona, losuj zakłócenia i wagi startowe sieci , w przeciwnym wypadku wróć do kroku pierwszego.

3. Przeprowadź proces uczenia sieci na danych uczących.

4. Zasymuluj działanie sieci dla danych testowych, zapisz wyniki symulacji, idź do kroku 2.

5.3 Sposób wyboru sieci poprawnie nauczonej

Jak pokazały wyniki symulacji, praktycznie wszystkie z testowanych i analizowanych algorytmów w pewnych warunkach działały znacznie gorzej od innych metod i okazywały się rozbieżne. Nawet, gdy ze statystycznego punktu widzenia sytuacja taka miała miejsce stosunkowo rzadko (np. 2-3 razy na 500 prób symulacji), nie dawało to przecież żadnej gwarancji, że przy konkretnym przeprowadzeniu procesu uczenia uzyskany wynik będzie zgodny z założeniami. Z tego też powodu pojawił się praktyczny problem wyboru właściwej sieci spośród wszystkich przypadków wytrenowania konkretnej struktury danym algorytmem przy określonym typie zakłóceń. Zadanie to ma wiele wspólnego ze znanymi z literatury próbami stosowanie układów sieci, które na drodze głosowania lub konkurencji mogą typować właściwe rozwiązanie. Zasadnicza różnica polega jednak na tym, że dopuszczalne jest tu pojawianie się w danych uczących dużych zakłóceń mających potencjalnie duży wpływ na przebieg procesu uczenia. Problem ten można sformułować następująco: przeprowadzamy n-krotnie trenowanie sieci na danych zawierających zakłócenia o nieznanym rozkładzie przy różnych wartościach parametrów początkowych. Otrzymujemy w ten sposób n różnie nauczonych sieci, których jakości działania nie można porównać, ze względu na to, że nie dysponujemy czystymi obrazami, na których trzeb by je testować. Jedyne, co można na ich temat powiedzieć, to jak dobrze dopasowały się do danych uczących.

Załóżmy teraz, że w każdym z ciągów uczących zakłócenia były generowane w podobny sposób, ale nie takie same. Tak właśnie przedstawia się sytuacja z wynikami zebranymi podczas testowania opracowanych w niniejszej pracy odpornych algorytmów uczenia. Jeżeli nie jest możliwa ocena działania sieci na podstawie popełnianych przez nią błędów w przypadku, gdy nie są znane dane bez zakłóceń, wyboru dokonać można analizując samą sieć. Zaproponowano więc, by wszystkie n sieci powstałych przez trenowanie tą samą metodą przy różnych warunkach początkowych, potraktować w sposób parametryczny, czyli jako zbiór punktów (przy założeniu, że są to struktury o tej samej architekturze). Dla tak skonstruowanego

zbioru należy obliczyć jego centrum, a następnie przyjąć, że sieć mu najbliższa jest najbardziej reprezentatywna dla danej sytuacji. Jeżeli przez θ_i oznaczymy wektor parametrów i-tej sieci, to dla każdego takiego wektora, analogicznie do zależności (4.37), obliczamy odległość:

$$D_i = \sqrt{(\theta_i - \bar{\theta})^T S^{-1} (\theta_i - \bar{\theta})}, \tag{5.1}$$

a następnie wybieramy $\hat{\theta}$, takie, że:

$$D(\hat{\theta}) = \min_i D_i. \tag{5.2}$$

Oczywiście metoda taka ma sens jedynie dla odpornych algorytmów, które w większości przypadków uczą sieci prawidłowe i jedynie w szczególnych warunkach są przyczyną błędnego wytrenowania struktury. Opisana procedura znalazła zastosowanie przy analizie danych eksperymentalnych jako jeden z elementów miary jakości odpornych algorytmów uczenia.

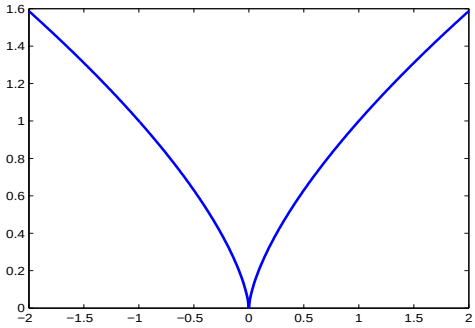
5.4 Zadania testowe

Spośród różnych zadań, do których trenować można sieci jednokierunkowe, najbardziej oczywistym do testowania odpornych algorytmów uczenia, jest zadanie aproksymacji funkcji. Wynika to z dwóch powodów. Po pierwsze, sam sposób wprowadzania modyfikacji mającej służyć uodpornieniu sieci, który w przypadku algorytmów obecnych w literaturze polega na zmianie minimalizowanego podczas uczenia kryterium, sugeruje, że odporność osiągana tu jest poprzez eliminowanie niektórych elementów z potencjalnie ciągłego zbioru wartości mogących pojawić się na wyjściu sieci. Po drugie, zadanie to pozwala na łatwe uzyskiwanie danych czystych, jak również możliwość stopniowania wprowadzonych zakłóceń, nie tylko pod względem ilości, ale i skali.

Aproksymacja funkcji jednej zmiennej

Do testowania algorytmów w zadaniu aproksymacji funkcji w pracach dotyczących odpornych algorytmów stosowana jest prawie zawsze funkcja jednej zmiennej pokazana na rysunku 5.1 i dana zależnością:

$$y = |x|^{-2/3}. \tag{5.3}$$



Rysunek 5.1: Aproksymowana funkcja $y = |x|^{-2/3}$.

Jej zaletą jest punkt nieróżniczkowalności, który, przy odpowiednio rzadkim próbkowaniu może "oszukiwać" algorytmy swym podobieństwem do danej odstającej.

Aproksymacja funkcji dwóch zmiennych

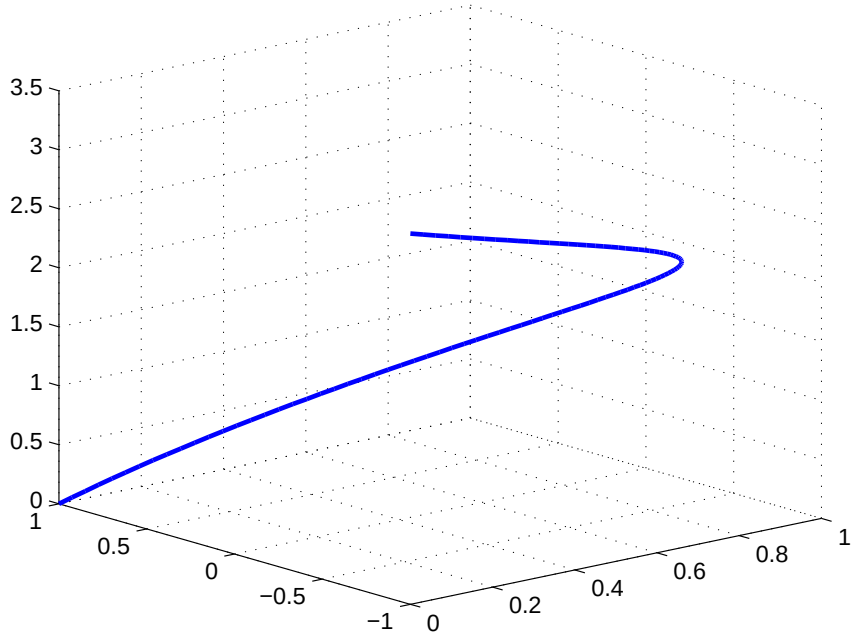
W niniejszej pracy do testowania wybrano również funkcję dwóch zmiennych opisaną jako

$$\begin{cases} x = \sin y \\ z = \cos y \end{cases} \tag{5.4}$$

Oczywiście jest to zależność funkcyjna jedynie na określonych obszarach (pokazanych na rysunku 5.2) i tak też była traktowana. Zaletą jej kształtu jest skok, który również może zostać błędnie rozpoznany jako błąd gruby.

Problem dwóch spiral

Jak już wcześniej wspomniano, opracowane odporne algorytmy uczenia przeznaczone są do sytuacji, w których wyjście sieci przybierać może ciągłe wartości. W związku z tym, ich przydatność w zadaniach klasyfikacji i rozpoznawania wzorców może być mocno ograniczona. Oczywiście im większa ilość klas w zadaniu, tym bardziej spodziewać się można, że odporne metody będą dawały rezultaty lepsze niż algorytm klasyczny.



Rysunek 5.2: Aproksymowana funkcja dwóch zmiennych $x = \sin y, z = \cos y$.

Aby jednak pokazać, że niektóre z nich mogą polepszać jakość uczenia sieci nawet w przypadku rozpoznawania obrazów pochodzących z zaledwie dwóch klas, posłużono się znanym zadaniem testowym, tzw. problemem dwóch spiral. Do skonstruowania klas testowych użyto więc dwóch przeplatających się spiral Archimedesa.

Spiralę taką definiuje się jako krzywą zakreślaną przez punkt poruszający się ze stałą prędkością v, po półprostej obracającej się wokół swego początku ze stałą prędkością kątową ω. Opisać można ją we współrzędnych biegunowych poniższym prostym wzorem:

$$\rho = v/\omega * \phi, \tag{5.5}$$

gdzie ρ oznacza promień, zaś ϕ współrzędną kątową. Sieć uczona była na ciągu, obrazów, z których część należała do jednej klasy - spirali, część zaś do drugiej. W danych uczących nie pojawiały się obrazy nienależące do żadnej z klas.

Dane strumieniowe - predykcja szeregów czasowych

Jak już wspomniano przy omawianiu odpornego algorytmu z adaptacyjnie zmiennym współczynnikiem uczenia VLR, znajduje on zastosowanie jedynie w metodzie

uczenia typu "on-line", która, poza szczególnymi przypadkami, jest w rzeczywistości rzadko stosowana. Gdybyśmy brali pod uwagę jedynie zadania rozpoznawania obrazów, czy aproksymacji funkcji, tworzenie takiego algorytmu uznane być by mogło za mało praktyczne, w porównaniu z algorytmami drugiego rzędu. Jeśli jednak sięgniemy do trochę innych zastosować okazuje się on być bardzo przydatny.

Stworzone do przetestowania metody VLR zadanie testowe polegało na predykcji dwóch szeregów czasowych. Symulować może ono prymitywny przypadek tzw. danych strumieniowych, w których mamy do czynienia z koniecznością ciągłego przetwarzania informacji w miarę jej napływania, bez możliwości gromadzenia większej ich ilości przekraczającej zadany rozmiar bufora pamięci. Dane takie są często wielowymiarowe i wymagają działania na nich na bieżąco, dlatego wszelkie poprawki mające na celu zwiększenie odporności algorytmu na błędy również powinny spełniać postulat prawidłowego działania w takich warunkach. Podobnie jak w przypadku danych strumieniowych, w zadaniu predykcji szeregów czasowych kolejne wartości szeregu napływają stale w dyskretnych momentach czasu. Muszą być one na bieżąco wykorzystywane do wytworzenia na wyjściu sieci wartości odpowiadającej następnemu elementowi szeregu. Sieć wykorzystuje do jej wygenerowania, oprócz wartości bieżącej, również kilka wartości poprzednich, co symuluje skończoną pamięć bufora. Jeżeli dodamy do tego wektor, jako wejście i wyjście sieci, można śmiało stwierdzić, iż jest to prosty przykład przetwarzania danych strumieniowych.

W tym przypadku zadanie sformułowano następująco: mamy do czynienia z dwoma sygnałami sinusoidalnymi trwającymi 5 sekund próbkowanymi z częstotliwością 20HZ. Po upływie 2.5 sekundy częstotliwość w pierwszym z dwukrotnie się zwiększa, w drugim zaś dwukrotnie zmniejsza. Sieć powinna adaptować się do tych zmian przewidując, na podstawie podanych w każdej epoce na jej wejścia, 5 ostatnich wartości, przyszłe wartości sygnałów.

5.5 Typy błędów

Błędy dla zadania aproksymacji

Przy symulowaniu błędów mogących pojawić się w danych mających służyć uczeniu sieci neuronowych, zastosowano bardzo prostą procedurą. Mianowicie, do odpowiednio przygotowanych danych, całkowicie pozbawionych zakłóceń, dodawano błędy

wygenerowane sztucznie na podstawie założonych rozkładów, tak jak to opisano przy omawianiu różnych typów błędów w danych (sekcja 1.2).

Technicznie rzecz ujmując, polega to na dodaniu do każdego obrazu uczącego błędu wylosowanego dla niego zgodnie z pewnym schematem. W literaturze poświęconej odpornym algorytmom uczenia sieci, błędy takie dodawane były jedynie do wektora targetu, a więc zadanych wartości wyjść. W taki też sposób stworzone zostały typy błędów I i II. W niniejszej pracy rozszerzono możliwość występowania błędów na dane podawane na wejście sieci (błędy typu III).

Dodatkowo symulowano błędy powstałe poprzez zastąpienie 49% danych uczących szumem o jednorodnym rozkładzie na obszarze ograniczonym poprzez wartości i argumenty aproksymowanej funkcji (błędy typu IV).

Błędy dla zadania klasyfikacji

W przypadku opisanego dalej problemu dwóch spiral, wartości etykiet klas ograniczały się do zbioru $\{0, 1\}$, w związku z tym powyższe modele błędów nie mogły mieć zastosowania. Z tego powodu wprowadzono dla wymienionego zadania dodatkowy sposób zakłócania ciągu uczącego, który polegał na tym, że zadaną z pewnym zadanym prawdopodobieństwem δ, podobnie jak w innych przypadkach, zmieniano etykietę klasy obrazu uczącego na przeciwną [18, 30].

Ilość wprowadzanych zakłóceń

Zakłócenia wprowadzane były do danych uczących z założonymi prawdopodobieństwami, najczęściej δ=0.1, 0.2 i 0.3, a nie w ustalonych wcześniej ilościach. Należy przez to rozumieć, że dla danych, w których zakłócenia pojawiają się z konkretną wartością δ, nie mamy pewności ile w rzeczywistości dużych zakłóceń znalazło się w konkretnym ciągu uczącym, możemy jedynie stwierdzić, z jakim prawdopodobieństwem zostały one wylosowane w trakcie przygotowywania danych do testowania algorytmów.

Rozdział 6

Wyniki eksperymentów symulacyjnych

6.1 Wyniki symulacji

Rezultaty uzyskane podczas testowania różnych algorytmów uczenia zebrane zostały w formie tabel umożliwiających porównanie działania sieci uczonych poszczególnymi metodami. Jak już wspomniano, jako miarę jakości działania sieci neuronowej, a co za tym idzie algorytmu, którym została wytrenowana, przyjęto błąd średniokwadratowy pomiędzy wartościami żądanymi, a uzyskanymi na wyjściu sieci dla ciągu testowego. Tabele powstały w oparciu o przeprowadzenie serii opisanych uprzednio symulacji polegających na wielokrotnym uczeniu sieci na różnych danych uczących, generowanych w odpowiedniej ilości według zadanego schematu, przy różnych warunkach początkowych. Obliczany był następnie błąd średniokwadratowy dla każdej z sieci testowanych na czystych danych testujących.

Uśredniony dla dużej ilości symulacji (konkretne ilości podane w odpowiednich podrozdziałach), błąd popełniany przez sieci, przedstawiony został w kolumnie pierwszej. Można uznać, że niesie on najistotniejszą informację na temat tego, jak dany algorytm sprawdza się w porównaniu z innymi (najlepsze w danym przypadku wartości zostały zaznaczone pogrubioną czcionką). Dodatkowo, w kolejnych kolumnach pokazane zostały: odchylenie standardowe, wartość maksymalna i minimalna z wszystkich prób, jak również trzy podstawowe kwantyle stopni 0.25, 0.50 i 0.75 (oznaczone jako q=0.25, etc). Ich analiza potencjalnie umożliwia zaobserwowanie tego, czy rozpatrywany algorytm np. działa dobrze dla większości przypadków, choć

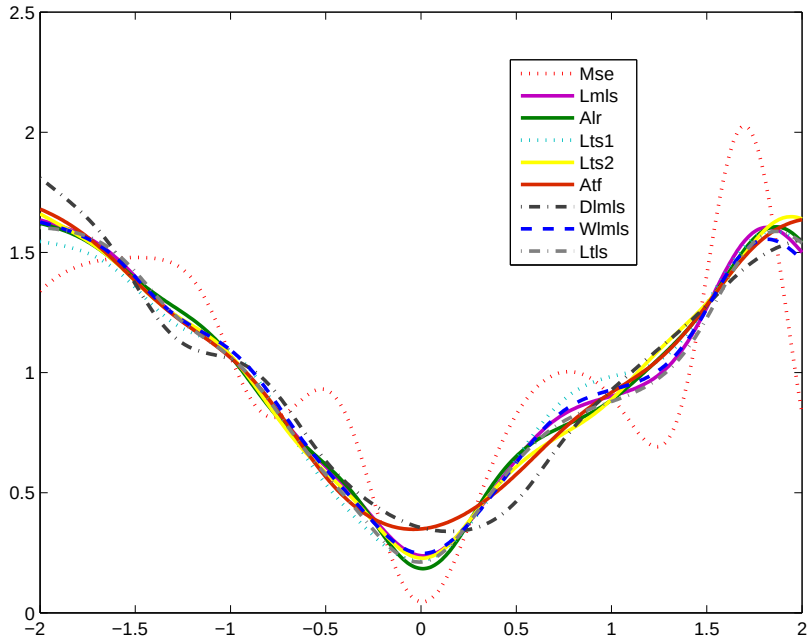
w pewnych warunkach jest rozbieżny. Sytuacja taka ma miejsce, gdy średnia zakłócana jest przez kilka prób nie zakończonych sukcesem, podczas gdy algorytm mógł dawać lepsze wyniki niż inne metody dla pozostałych przypadków.

Ostatnia kolumna tabel zawiera błąd średniokwadratowy sieci wybranej ze wszystkich sieci nauczonych w ustalonych warunkach danym algorytmem, na podstawie jej odległości od centrum zbioru tworzonego przez parametry wszystkich innych sieci, tak jak to opisano wcześniej (sekcja 5.3).

Do porównania, oprócz opracowanych w ramach niniejszej pracy algorytmów, dodano wyniki uzyskane za pomocą algorytmu klasycznego z kryterium mse, oraz algorytmu odpornego z kryterium LMLS, w tabelach pokazano więc w sumie wyniki działania 9 metod uczenia. Zostały one oznaczone następującymi skrótami:

- ALR (*adaptive learning rate*) - algorytm drugiego rzędu z adaptacyjnie dobieranym współczynnikiem uczenia,
- ATF (*adaptive transfer function*) - algorytm z adaptacyjnym doborem funkcji aktywacji neuronów,
- WLMLS (*weighted LMLS*) - algorytm ze wstępną analizą danych,
- DLMLS (*dedicated LMLS*) - dedykowany algorytm dla kryterium LMLS,
- LTLS - algorytm z kryterium Ltls,
- LTS1 - algorytm z kryterium LTS z apriorycznie przyjmowanym h,
- LTS2 - algorytm z kryterium LTS z h na podstawie MAD,
- MSE - algorytm z kryterium Mse,
- LMLS - algorytm z kryterium LMLS.

Dla porządku należy wspomnieć, że w opisie rezultatów działania badanych metod używano, dla poprawy stylu, określeń takich jak "błąd popełniany przez algorytm", "błąd powodowany przez algorytm" itp. Wszystkie one oznaczać mają to samo i należy rozumieć je w ten sam sposób, a więc jako błąd popełniany przez sieć uczoną konkretnym algorytmem (tak, jak opisano to wcześniej, chodzi o uśredniony



Rysunek 6.1: Porównanie działania odpornych algorytmów, uczących aproksymacji funkcji jednej zmiennej, przy prawdopodobieństwie pojawienia się dużych zakłóceń $\delta = 0.1$ (błędy typu I).

z kilkuset symulacji błąd średniokwadratowy sieci popełniany dla danych testujących). Podobnie rzecz się ma z używanymi zamiennie słowami rezultaty, czy wyniki, oznaczającymi wzmiankowane błędy.

6.1.1 Aproksymacja funkcji jednej zmiennej

W tabelach 6.1-6.9 przedstawiono wyniki symulacji polegających na uczeniu sieci zadania aproksymacji funkcji jednej zmiennej danej zależnością (5.3). Uśrednianie następowało tu po przeprowadzeniu 500 prób uczenia i testowania każdej metody uczenia dla każdego z rozważanych typów danych uczących.

Wyniki uzyskane przez poszczególne odporne algorytmy dla danych zawierających niewielką ilość błędów generowanych według modelu I (tabele 6.1, 6.2) są do siebie zbliżone. Największe błędy pojawiały się po uczeniu sieci odpornym algorytmem z kryterium LMLS, nie były one jednak znacząco wyższe od błędów powodowanych przez nowe odporne metody powstałe na potrzeby tej pracy. Błąd popełniany przez

Tabela 6.1: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji jednej zmiennej przy prawdopodobieństwie pojawienia się dużych zakłóceń $\delta = 0.1$ (błędy typu I)

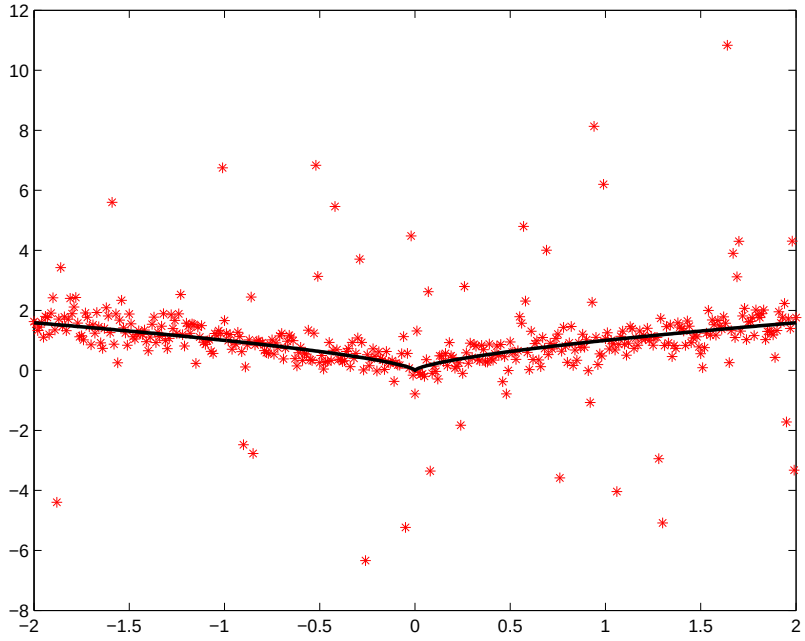
Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.0061	0.0019	0.0118	0.0028	0.0047	0.0058	0.0073	0.0028
ALR	**0.0046**	0.0019	0.0113	0.0015	0.0031	0.0043	0.0056	0.0027
ATF	**0.0047**	0.0019	0.0150	0.0013	0.0033	0.0044	0.0057	0.0048
WLMLS	0.0051	0.0017	0.0118	0.0025	0.0039	0.0048	0.0060	0.0058
DLMLS	0.0054	0.0022	0.0147	0.0024	0.0039	0.0049	0.0067	0.0036
LTLS	**0.0047**	0.0017	0.0136	0.0016	0.0037	0.0042	0.0055	0.0025
LTS1	0.0054	0.0018	0.0108	0.0019	0.0040	0.0053	0.0068	0.0025
LTS2	0.0049	0.0017	0.0090	0.0014	0.0037	0.0046	0.0058	0.0030
MSE	0.0398	0.0195	0.0924	0.0094	0.0270	0.0353	0.0492	0.0170

Tabela 6.2: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji jednej zmiennej przy prawdopodobieństwie pojawienia się dużych zakłóceń $\delta = 0.2$ (błędy typu I)

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.0088	0.0035	0.0216	0.0019	0.0064	0.0083	0.0111	0.0019
ALR	**0.0055**	0.0021	0.0125	0.0016	0.0042	0.0052	0.0069	0.0016
ATF	0.0064	0.0027	0.0148	0.0021	0.0042	0.0062	0.0079	0.0072
WLMLS	0.0071	0.0027	0.0146	0.0025	0.0052	0.0066	0.0083	0.0042
DLMLS	0.0070	0.0030	0.0175	0.0026	0.0046	0.0066	0.0083	0.0050
LTLS	0.0062	0.0025	0.0157	0.0022	0.0045	0.0058	0.0076	0.0078
LTS1	0.0056	0.0022	0.0108	0.0021	0.0038	0.0050	0.0067	0.0036
LTS2	0.0067	0.0024	0.0151	0.0031	0.0049	0.0063	0.0076	0.0034
MSE	0.0809	0.0351	0.1923	0.0230	0.0546	0.0716	0.1034	0.0551

Tabela 6.3: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji jednej zmiennej przy prawdopodobieństwie pojawienia się dużych zakłóceń $\delta = 0.3$ (błędy typu I)

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.0114	0.0046	0.0248	0.0029	0.0082	0.0108	0.0139	0.0051
ALR	**0.0074**	0.0029	0.0208	0.0029	0.0052	0.0073	0.0088	0.0041
ATF	0.0089	0.0034	0.0197	0.0020	0.0065	0.0086	0.0109	0.0020
WLMLS	0.0111	0.0050	0.0294	0.0027	0.0072	0.0105	0.0137	0.0043
DLMLS	0.0087	0.0042	0.0262	0.0024	0.0059	0.0080	0.0104	0.0044
LTLS	0.0101	0.0043	0.0216	0.0034	0.0067	0.0093	0.0126	0.0035
LTS1	0.0076	0.0030	0.0165	0.0027	0.0055	0.0069	0.0095	0.0057
LTS2	0.0100	0.0052	0.0408	0.0022	0.0066	0.0090	0.0132	0.0022
MSE	0.1096	0.0416	0.2184	0.0334	0.0771	0.1066	0.1355	0.0334



Rysunek 6.2: Aproksymowana funkcja i dane uczące przy prawdopodobieństwie pojawienia się dużych zakłóceń $\delta = 0.2$ (błędy typu II).

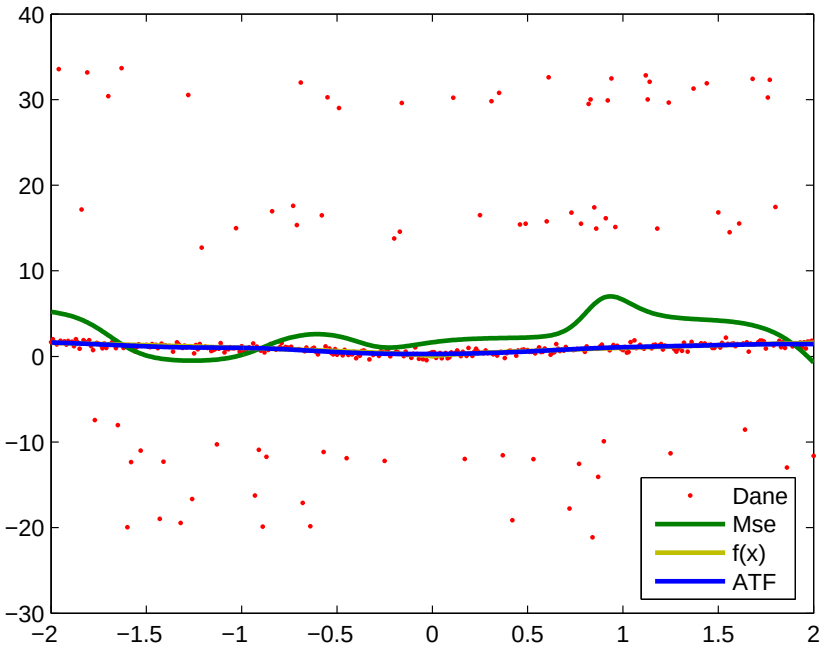
sieci uczone kryterium klasycznym (mse) był jednak zdecydowanie (kilkukrotnie) wyższy.

Przy $\delta = 0.3$ sytuacja jest już bardziej zróżnicowana: najlepsze rezultaty uzyskane zostały przez algorytmu LTS1 i ALR. Trochę gorzej poradziły sobie z uczeniem DLMLS i ATF, natomiast w ostatniej grupie plasują się pozostałe odporne algorytmy, przy czym znów największy błąd (nie licząc algorytmu klasycznego) uzyskiwały sieci uczone algorytmem LMLS. Różnice w błędach nie przekraczały tu jednak ok. 50%. W algorytmie WLMLS poziom błędu był, jak można zauważyć, najbardziej zbliżony do LMLS - łatwo to wytłumaczyć zakładając, że dla większości danych uczących obliczone wagi związane z identyfikacją dużych zakłóceń były równe 1.

Przykładowe dane uczące typu II pokazano na rysunku 6.2. Dla zakłóceń tego typu (tabela 6.4), przy najmniejszej ich ilości, najgorzej z algorytmów odpornych spisuje się metoda LTS1. Popełniany przez nią błąd jest nawet kilkukrotnie wyższy niż dla innych odpornych algorytmów. Tłumaczyć to można tym, że algorytm LTS1 opiera się na założonej apriorycznie maksymalnej ilości błędów, jeśli zaś mamy do czynienia z rozkładem zakłóceń typu II, dużych błędów może być więcej dlatego, że każdy błąd tego rodzaju może mieć sporą wartość, natomiast przy generowaniu ich z prawdopodobieństwem δ ich rzeczywista ilość może być większa niż δ, a więc przekroczyć ustalony próg.

Najlepiej w tym zestawieniu wypadły algorytmy ATF (rysunki 6.3 i 6.4), DLMLS, oraz LTLS. Dla $\delta = 0.2$ (tabela 6.5 sytuacja wygląda bardzo podobnie, z tą różnicą, że błąd dla algorytmu ALR jest kilka rzędów wielkości większy od innych. Jak pokazuje jednak analiza wartości kwantyli, spowodowane jest to jego rozbieżnością i bardzo dużym błędem dla jednego przypadku, którego wyeliminowanie plasować go może na poziomie podobnym, co LTS1. Zjawisko rozbieżności wystąpiło tu tylko dla jednego z 500, konkretnego zestawu parametrów początkowych i składu zakłóceń, co nie powinno dyskwalifikować metody, zwłaszcza, że w innych sytuacjach uzyskiwała ona znacznie lepsze rezultaty.

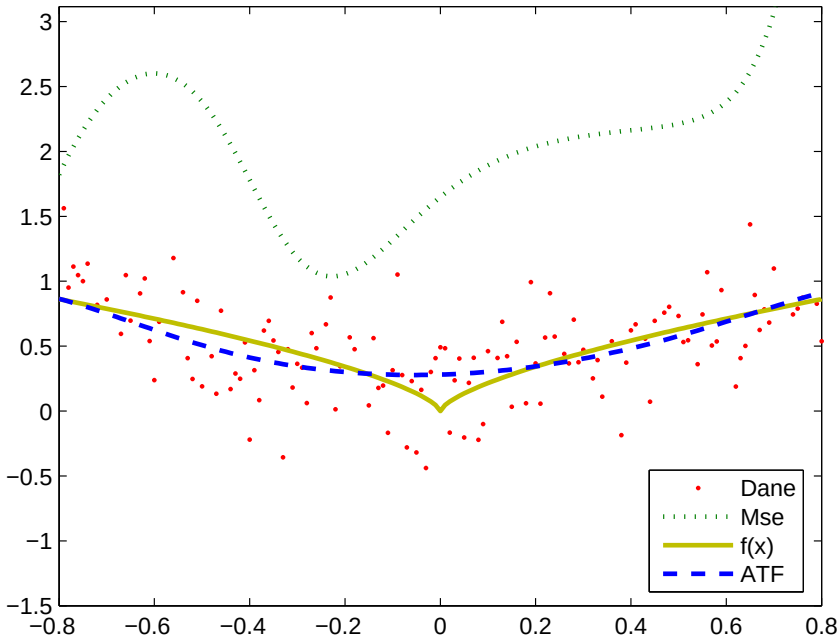
Warto zauważyć, że sieci wybrane jako leżące w centrum pod względem parametrów, reprezentują stosunkowo niski poziom błędów również w przypadku tych dwóch odpornych metod. Przy maksymalnej ilości zakłóceń (tabela 6.6) do algorytmów, których błąd jest największy, dołącza metoda LTS2, a także, choć w mniejszym



Rysunek 6.3: Najlepszy z odpornych algorytmów w porównaniu z algorytmem klasycznym (dane uczące z $\delta = 0.2$ błędów typu II).

stopniu WLMLS, której błąd jest dwukrotnie większy niż dla pozostałych odpornych algorytmów. Widać, że w tym przypadku gorzej radzą sobie metody, w których wykrywa się na podstawie analizy błędów ich zawartość w ciągu uczącym.

Wprowadzenie niewielkiej ilości zakłóceń w wektorze wejściowym powoduje nieco inne skutki (tabele 6.7 - 6.9). Po pierwsze, okazuje się, że różnice między wynikami uzyskanymi dla różnych algorytmów są stosunkowo niewielkie i po raz pierwszy algorytm z kryterium mse okazuje się lepszy od niektórych metod odpornych. Najlepiej działają tu algorytmy z przycinaną funkcją celu (Ltls, LTS1, LTS2), najgorzej ATF i DLMLS. Wynika to zapewne z tego, że są one oparte na idei odpornego estymatora Lts, który, w przeciwieństwie do odpornych M-estymatorów, pomaga również w eliminacji wpływu zakłóceń pojawiających się w wektorze wejściowym. Identycznie sytuacja przedstawia się zarówno dla $\delta = 0.2$ jak i 0.3, przy czym warto podkreślić, że praktycznie nie ma tu żadnej różnicy pomiędzy kryterium LMLS i mse, a więc prosta modyfikacja funkcji celu nie powoduje tu uodpornienia.



Rysunek 6.4: Najlepszy z odpornych algorytmów w porównaniu z algorytmem klasycznym (dane uczące z $\delta = 0.2$ błędów typu II) - powiększenie rysunku 6.3.

Tabela 6.4: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji jednej zmiennej przy prawdopodobieństwie pojawienia się dużych zakłóceń $\delta = 0.1$ (błędy typu II)

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.0050	0.0015	0.0098	0.0021	0.0040	0.0049	0.0058	0.0041
ALR	0.0052	0.0048	0.0467	0.0012	0.0031	0.0045	0.0056	0.0049
ATF	**0.0037**	0.0014	0.0083	0.0013	0.0026	0.0035	0.0043	0.0032
WLMLS	0.0049	0.0020	0.0133	0.0018	0.0036	0.0047	0.0057	0.0030
DLMLS	0.0043	0.0019	0.0103	0.0012	0.0030	0.0038	0.0052	0.0025
LTLS	0.0041	0.0015	0.0097	0.0014	0.0030	0.0040	0.0048	0.0030
LTS1	0.0632	0.1169	0.8188	0.0025	0.0045	0.0068	0.0801	0.0050
LTS2	0.0051	0.0018	0.0105	0.0021	0.0038	0.0049	0.0060	0.0025
MSE	1.7929	0.6893	3.8039	0.5333	1.2493	1.7341	2.1771	1.7562

Tabela 6.5: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji jednej zmiennej przy prawdopodobieństwie pojawienia się dużych zakłóceń $\delta = 0.2$ (błędy typu II)

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.0053	0.0016	0.0094	0.0020	0.0041	0.0050	0.0063	0.0033
ALR	15.2474	56.8152	330.7377	0.0030	0.0055	0.0102	0.0386	0.0031
ATF	0.0046	0.0019	0.0148	0.0008	0.0034	0.0045	0.0054	0.0034
WLMLS	0.0069	0.0028	0.0173	0.0028	0.0051	0.0061	0.0088	0.0057
DLMLS	0.0048	0.0018	0.0107	0.0017	0.0035	0.0046	0.0056	0.0042
LTLS	**0.0045**	0.0018	0.0107	0.0013	0.0032	0.0044	0.0054	0.0037
LTS1	0.1454	0.2514	1.1103	0.0024	0.0058	0.0115	0.1828	0.0052
LTS2	0.0062	0.0046	0.0430	0.0012	0.0043	0.0053	0.0067	0.0060
MSE	4.0996	1.6899	10.8478	0.8996	2.8594	3.9274	5.2043	3.2179

Tabela 6.6: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji jednej zmiennej przy prawdopodobieństwie pojawienia się dużych zakłóceń $\delta = 0.3$ (błędy typu II)

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.0056	0.0021	0.0123	0.0024	0.0041	0.0053	0.0066	0.0026
ALR	0.0952	0.1644	1.0226	0.0000	0.0000	0.0002	0.1409	0.0000
ATF	**0.0049**	0.0020	0.0112	0.0014	0.0033	0.0048	0.0060	0.0019
WLMLS	0.0108	0.0046	0.0313	0.0029	0.0075	0.0098	0.0139	0.0068
DLMLS	0.0054	0.0021	0.0120	0.0017	0.0039	0.0049	0.0065	0.0031
LTLS	0.0056	0.0023	0.0118	0.0017	0.0039	0.0054	0.0068	0.0019
LTS1	0.1876	0.3165	1.5563	0.0026	0.0066	0.0142	0.2498	0.0070
LTS2	0.1112	0.3512	2.2469	0.0027	0.0051	0.0070	0.0115	0.0061
MSE	5.9862	2.5492	14.9487	1.5837	4.0991	5.6400	7.2759	4.6236

Tabela 6.7: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji jednej zmiennej przy prawdopodobieństwie pojawienia się dużych zakłóceń $\delta = 0.1$ (błędy typu I w wektorze wejściowym)

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.0151	0.0123	0.1267	0.0070	0.0119	0.0137	0.0154	0.0108
ALR	0.0133	0.0023	0.0188	0.0068	0.0119	0.0129	0.0142	0.0182
ATF	0.0301	0.0172	0.1328	0.0116	0.0191	0.0257	0.0367	0.0174
WLMLS	0.0138	0.0024	0.0199	0.0091	0.0119	0.0133	0.0152	0.0144
DLMLS	0.0328	0.0222	0.1286	0.0089	0.0169	0.0268	0.0427	0.0469
LTLS	0.0106	0.0023	0.0167	0.0068	0.0090	0.0102	0.0118	0.0095
LTS1	**0.0104**	0.0022	0.0170	0.0069	0.0089	0.0101	0.0116	0.0080
LTS2	0.0112	0.0023	0.0188	0.0066	0.0097	0.0107	0.0125	0.0107
MSE	0.0140	0.0023	0.0215	0.0093	0.0123	0.0137	0.0151	0.0119

Tabela 6.8: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji jednej zmiennej przy prawdopodobieństwie pojawienia się dużych zakłóceń $\delta = 0.2$ (błędy typu I w wektorze wejściowym)

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.0177	0.0116	0.1286	0.0121	0.0146	0.0162	0.0183	0.0150
ALR	0.0159	0.0028	0.0239	0.0107	0.0138	0.0156	0.0176	0.0127
ATF	0.0464	0.0186	0.1211	0.0182	0.0339	0.0437	0.0565	0.0318
WLMLS	0.0169	0.0027	0.0242	0.0117	0.0149	0.0168	0.0187	0.0138
DLMLS	0.0381	0.0258	0.1718	0.0137	0.0207	0.0304	0.0464	0.0352
LTLS	**0.0115**	0.0021	0.0188	0.0078	0.0101	0.0113	0.0124	0.0120
LTS1	0.0120	0.0120	0.1284	0.0075	0.0091	0.0103	0.0121	0.0090
LTS2	0.0149	0.0117	0.1273	0.0087	0.0116	0.0133	0.0163	0.0111
MSE	0.0180	0.0032	0.0288	0.0121	0.0155	0.0178	0.0199	0.0214

Tabela 6.9: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji jednej zmiennej przy prawdopodobieństwie pojawienia się dużych zakłóceń $\delta = 0.3$ (błędy typu I w wektorze wejściowym)

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.0216	0.0041	0.0349	0.0136	0.0188	0.0209	0.0234	0.0171
ALR	0.0197	0.0037	0.0308	0.0124	0.0173	0.0191	0.0221	0.0174
ATF	0.0633	0.0275	0.1510	0.0268	0.0413	0.0574	0.0835	0.0373
WLMLS	0.0213	0.0037	0.0317	0.0146	0.0187	0.0214	0.0237	0.0239
DLMLS	0.0514	0.0338	0.1668	0.0165	0.0276	0.0416	0.0632	0.0417
LTLS	0.0132	0.0030	0.0282	0.0094	0.0113	0.0126	0.0140	0.0115
LTS1	**0.0129**	0.0044	0.0344	0.0075	0.0097	0.0123	0.0147	0.0085
LTS2	0.0191	0.0043	0.0301	0.0106	0.0162	0.0184	0.0217	0.0192
MSE	0.0245	0.0044	0.0409	0.0166	0.0211	0.0239	0.0271	0.0361

Zastąpienie części danych uczących jednorodnym szumem w obszarze ograniczonym maksymalnymi wartościami argumentów i wartości aproksymowanej funkcji (tabele 6.4 - 6.6) również skutkuje pogorszeniem jakości działania nauczonej na takich danych sieci. Widać jednak, że wpływ tego typu błędów, nawet jeśli stanowią one prawie połowę zbioru uczącego, jest istotnie mniejszy, niż w przypadku typowych dużych zakłóceń. Wszystkie algorytmy powodują, że uczone nimi sieci osiągają podobny rząd błędu. Uważa się, że w praktyce tego typu szum nie powinien mieć wpływu na proces budowania modelu, jednak, jak można zauważyć, błąd największy (algorytm Mse i DLMLS) jest pięciokrotnie wyższy od błędu najmniejszego uzyskiwanego przez algorytm Ltlmls. Złe działania metody DLMLS w porównaniu z LMLS tłumaczyć można tym, że proces uczenia dla niej mógł posunąć się bliżej minimum funkcji. Spowodować to mogło, pomimo uzyskiwania niskich wartości błędu LMLS, zbyt dobre dopasowanie sieci do danych uczących.

Oprócz algorytmu LTLMS najlepiej radzą sobie metody LTS1 i ALR, oraz LTS2 i WLMLS. Pozostałe algorytmy proces uczenia przeprowadziły zdecydowanie gorzej, niemniej stwierdzić można, że również w tym przypadku zastosowanie odpornych metod pozwala na polepszenie jakości działania sieci.

Wspominano już, że odporne algorytmy uczenia sieci, prócz zapewnienia odporności na pojawiające się w danych wartości odstające i błędy grube, powinny

Tabela 6.10: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji jednej zmiennej przy prawdopodobieństwie pojawienia się danych zastąpionych jednorodnym szumem $\delta = 0.49$

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.0407	0.0091	0.0699	0.0231	0.0347	0.0404	0.0449	0.0424
ALR	0.0315	0.0085	0.0637	0.0170	0.0256	0.0303	0.0364	0.0319
ATF	0.0402	0.0080	0.0604	0.0216	0.0340	0.0405	0.0459	0.0499
WLMLS	0.0386	0.0086	0.0638	0.0204	0.0327	0.0368	0.0438	0.0333
DLMLS	0.0554	0.0273	0.2187	0.0246	0.0422	0.0495	0.0608	0.0353
LTLS	**0.0148**	0.0051	0.0364	0.0061	0.0114	0.0137	0.0172	0.0107
LTS1	0.0261	0.0359	0.2238	0.0006	0.0039	0.0117	0.0337	0.0011
LTS2	0.0363	0.0132	0.0726	0.0053	0.0254	0.0365	0.0452	0.0156
MSE	0.0515	0.0106	0.0831	0.0292	0.0435	0.0511	0.0575	0.0553

również działać możliwie dobrze dla danych uczących, które nie zostały w żaden sposób zanieczyszczone. Z tego powodu niezbędne było również sprawdzenie, jak omawiane zadanie aproksymacji wykonywane było przez sieci uczone na prawidłowym zbiorze danych. Wyniki symulacji zebrane w tabeli 6.11 pokazują, że prawie wszystkie odporne metody radziły sobie z taką sytuacją stosunkowo dobrze. Jedynie w przypadku kryterium Ltls, błąd popełniany przez sieci jest prawie o rząd większy niż dla najlepszych metod, natomiast wśród pozostałych pojawia się maksymalnie dwukrotna różnica wartości błędu. Pomimo dużych różnic względnych, trzeba zaznaczyć, że bezwzględnie uzyskane wartości są na tyle małe, iż można uznać, że wszystkie metody pozwalają przeprowadzić proces uczenia prawidłowo. Prawdopodobnie, lepsze rezultaty przy zastosowaniu metody LTLS uzyskano by zwiększając rozmiar ciągu uczącego, gdyż z natury swojej ma ona skłonność do eliminowania największej liczby obrazów uczących.

Z oczywistych względów nie przedstawiono w tabeli wyników działania metody LTS1, która w tym przypadku równoznaczna jest z metodą mse, jak również wybranej sieci, gdyż odchylenie standardowe jest tu zawsze bardzo niewielkie.

Tabela 6.11: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji jednej zmiennej na danych bez zakłóceń

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75
LMLS	0.0007	0.0001	0.0012	0.0005	0.0006	0.0007	0.0008
ALR	0.0007	0.0002	0.0013	0.0005	0.0006	0.0007	0.0008
ATF	0.0015	0.0009	0.0044	0.0004	0.0008	0.0012	0.0018
WLMLS	0.0010	0.0003	0.0019	0.0005	0.0007	0.0009	0.0011
DLMLS	0.0014	0.0013	0.0092	0.0005	0.0007	0.0010	0.0018
LTLS	0.0065	0.0098	0.0687	0.0007	0.0020	0.0033	0.0058
LTS2	0.0013	0.0017	0.0121	0.0005	0.0007	0.0008	0.0011
MSE	0.0007	0.0001	0.0012	0.0005	0.0007	0.0007	0.0008

6.1.2 Aproksymacja funkcji dwóch zmiennych

Wyniki symulacji sieci uczonych aproksymacji funkcji dwóch zmiennych danej zależnością 5.4 zebrane zostały w tabelach 6.12 - 6.20. Są to wartości średnie z 500 eksperymentów przeprowadzonych dla każdej ilości zakłóceń danego typu.

Dla danych uczących zawierających najmniejszą ilość zakłóceń typu I, najgorzej, nie licząc algorytmu klasycznego, wypada odporny algorytm z kryterium LMLS. Średni błąd popełniany przez uczone nim sieci jest prawie czterokrotnie większy niż w przypadku najlepszego tutaj algorytmu ATF. Dobrze wypada również metoda DLMLS, natomiast pozostałe odporne algorytmy powodują błędy podobnej wielkości, różniące się o kilkanaście procent. Po zwiększeniu ilości zakłóceń (tabela 6.13), sytuacja przedstawia się bardzo podobnie: nadal najlepsze rezultaty dają metody ATF, oraz DLMLS, podczas gdy algorytm z kryterium LMLS wydaje się być najgorszy w tym gronie. Przy maksymalnej ilości błędów tego typu, praktycznie wszystkie, z wyjątkiem wspomnianych już dwóch metod, tracą skuteczność. Najsłabsze w zestawieniu okazują się algorytmy z trymowaną funkcją kryterialną, jak również WLMLS. Można zaryzykować więc stwierdzenie, że tam, gdzie proces uczenia zatrzymywany jest wcześniej uzyskano rezultaty najlepsze.

Z kolei dla danych uczących o najmniejszej zawartości błędów typu drugiego (tabela 6.15) najgorszy okazuje się algorytm LTS1. Co prawda średni błąd obliczony dla sieci trenowanych algorytmem ALR jest tu największy, ale jak pokazuje wartość kwantyla 0.75, wynikać to musi z niekorzystnych warunków w jednej, lub najwyżej

Tabela 6.12: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji dwóch zmiennych przy prawdopodobieństwie pojawienia się dużych zakłóceń $\delta = 0.1$(błędy typu I)

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.0584	0.0441	0.2285	0.0148	0.0323	0.0430	0.0648	0.0374
ALR	0.0274	0.0134	0.1110	0.0109	0.0196	0.0249	0.0344	0.0118
ATF	**0.0129**	0.0067	0.0384	0.0023	0.0076	0.0113	0.0170	0.0072
WLMLS	0.0253	0.0101	0.0661	0.0079	0.0179	0.0241	0.0307	0.0193
DLMLS	0.0171	0.0076	0.0464	0.0060	0.0114	0.0149	0.0214	0.0082
LTLS	0.0242	0.0099	0.0646	0.0092	0.0177	0.0221	0.0275	0.0169
LTS1	0.0305	0.0159	0.1298	0.0107	0.0207	0.0266	0.0345	0.0357
LTS2	0.0284	0.0107	0.0617	0.0090	0.0196	0.0272	0.0349	0.0223
MSE	0.3967	0.2990	2.2687	0.0408	0.2122	0.3089	0.5461	0.0488

Tabela 6.13: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji dwóch zmiennych przy prawdopodobieństwie pojawienia się dużych zakłóceń $\delta = 0.2$ (błędy typu I)

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.1442	0.1882	1.5775	0.0154	0.0571	0.0831	0.1408	0.0871
ALR	0.0550	0.0486	0.3405	0.0143	0.0272	0.0414	0.0620	0.0316
ATF	**0.0206**	0.0117	0.0552	0.0053	0.0126	0.0174	0.0255	0.0094
WLMLS	0.0464	0.0268	0.1728	0.0099	0.0265	0.0400	0.0604	0.0107
DLMLS	0.0259	0.0172	0.1401	0.0056	0.0154	0.0219	0.0304	0.0257
LTLS	0.0448	0.0346	0.2848	0.0104	0.0263	0.0350	0.0549	0.0123
LTS1	0.0335	0.0154	0.1338	0.0091	0.0244	0.0312	0.0402	0.0247
LTS2	0.0534	0.0651	0.6393	0.0155	0.0280	0.0407	0.0571	0.0316
MSE	0.7722	0.3739	1.9482	0.2480	0.5124	0.7152	0.9344	0.3920

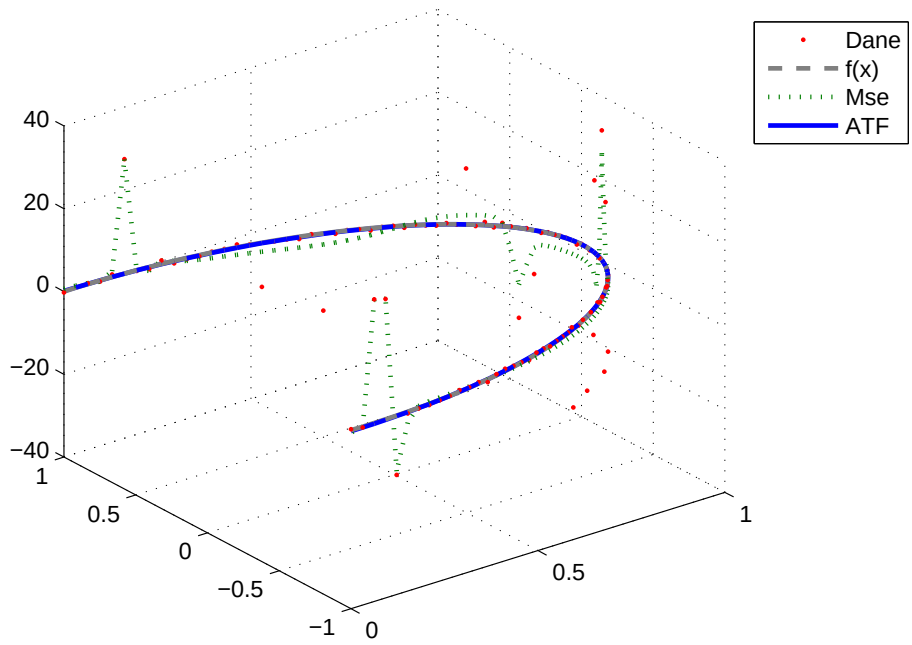
Tabela 6.14: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji dwóch zmiennych przy prawdopodobieństwie pojawienia się dużych zakłóceń $\delta = 0.3$ (błędy typu I)

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.0579	0.0331	0.1721	0.0077	0.0333	0.0499	0.0773	0.0596
ALR	0.0920	0.1035	0.6995	0.0168	0.0401	0.0585	0.0864	0.0390
ATF	**0.0253**	0.0162	0.0866	0.0024	0.0131	0.0213	0.0335	0.0378
WLMLS	0.1009	0.0919	0.5524	0.0134	0.0441	0.0751	0.1206	0.0354
DLMLS	0.0382	0.0208	0.1228	0.0121	0.0229	0.0332	0.0493	0.0363
LTLS	0.1011	0.0682	0.4566	0.0203	0.0617	0.0810	0.1205	0.0375
LTS1	4.3195	6.5798	40.0671	0.0119	0.0367	1.2513	6.5647	0.0499
LTS2	0.1357	0.1438	1.0785	0.0160	0.0501	0.0927	0.1795	0.0409
MSE	45.5307	12.7130	81.3433	20.1623	36.7137	43.8255	54.3685	33.2337

w kilku symulacjach. Najmniejsze błędy uzyskują, podobnie, jak poprzednio, metody ATF (rysunek 6.5) i DLMLS, natomiast LMLS, okazuje się znów gorszy od pozostałych odpornych algorytmów z wyjątkiem LTS1. Podobnie, jak w przypadku poprzedniego zadania wyjaśnić to można pojawianiem się ilości grubych zakłóceń przekraczających założony w algorytmie próg.

Nieoczekiwanie, dla większej ilości zakłóceń, błędy popełniane przez sieci uczone niektórymi algorytmami zmniejszają się. Dotyczy to metod: LMLS i Ltls. Powtarza się natomiast sytuacja z metodą ALR, która uzyskuje niskie błędy w większości przypadków, natomiast błąd średni ma najwyższy. Warto zauważyć, że dla wszystkich odpornych metod, sieci wybrane jako środkowe, popełniają stosunkowo niewielkie błędy. Ranking algorytmów nie zmienia się przy $\delta = 0.3$ - najlepsze znów są ATF i Dlmls, za nimi plasuje się Wlmls, natomiast pozostałe metody dają wyniki zdecydowanie gorsze.

Przedstawione w tabelach 6.18 - 6.19 błędy otrzymane dla sieci uczonych na danych zawierających zakłócenia w wektorze wejściowym przedstawiają się zupełnie inaczej. Najgorsza (uzyskują błąd większy nawet niż algorytm z kryterium mse) okazuje się, najlepsza uprzednio, metoda ATF. Wynika to zapewne z tego, że proces uczenia jest w niej spowalniany arbitralnie, bez wykorzystywania informacji i aktualnym zachowaniu sieci. Pozostałe, z wyjątkiem LTS1, prezentują zrównoważony



Rysunek 6.5: Najlepszy z odpornych algorytmów w porównaniu z algorytmem klasycznym (dane uczące z $\delta = 0.2$ błędów typu II).

Tabela 6.15: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji dwóch zmiennych przy prawdopodobieństwie pojawienia się dużych zakłóceń $\delta = 0.1$ (błędy typu II)

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.0326	0.0439	0.4347	0.0076	0.0192	0.0253	0.0340	0.0190
ALR	0.5144	4.9243	49.2647	0.0085	0.0165	0.0215	0.0260	0.0128
ATF	**0.0096**	0.0049	0.0261	0.0014	0.0056	0.0087	0.0123	0.0052
WLMLS	0.0202	0.0082	0.0452	0.0050	0.0138	0.0190	0.0251	0.0088
DLMLS	0.0152	0.0066	0.0370	0.0043	0.0105	0.0142	0.0186	0.0224
LTLS	0.0218	0.0079	0.0485	0.0051	0.0154	0.0216	0.0272	0.0161
LTS1	1.6276	3.0223	18.2707	0.0107	0.0226	0.0322	2.3183	0.0519
LTS2	0.0311	0.0570	0.5898	0.0079	0.0194	0.0253	0.0309	0.0218
MSE	23.6145	8.6535	45.1408	3.9663	18.0965	23.8920	29.4470	3.9663

Tabela 6.16: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji dwóch zmiennych przy prawdopodobieństwie pojawienia się dużych zakłóceń $\delta = 0.2$ (błędy typu II)

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.0140	0.0061	0.0404	0.0029	0.0100	0.0131	0.0170	0.0091
ALR	15.2774	80.4489	602.3634	0.0063	0.0177	0.0235	0.0328	0.0206
ATF	**0.0113**	0.0065	0.0332	0.0024	0.0064	0.0102	0.0147	0.0070
WLMLS	0.0231	0.0087	0.0530	0.0035	0.0172	0.0215	0.0285	0.0157
DLMLS	0.0161	0.0057	0.0450	0.0075	0.0120	0.0155	0.0197	0.0111
LTLS	0.0250	0.0102	0.0660	0.0064	0.0178	0.0240	0.0290	0.0220
LTS1	1.6980	4.7115	38.7084	0.0080	0.0229	0.0336	1.2948	0.0283
LTS2	0.2326	1.0905	8.8763	0.0096	0.0221	0.0269	0.0352	0.0366
MSE	36.6818	11.8470	66.6210	13.4001	28.1746	35.5839	45.4512	14.9619

Tabela 6.17: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji dwóch zmiennych przy prawdopodobieństwie pojawienia się dużych zakłóceń $\delta = 0.3$ (błędy typu II)

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.0889	0.1575	0.8548	0.0066	0.0361	0.0504	0.0738	0.0367
ALR	70.8095	212.1080	1765.4	0.0075	0.0255	0.0384	62.5158	0.0309
ATF	**0.0140**	0.0077	0.0461	0.0036	0.0088	0.0123	0.0173	0.0139
WLMLS	0.0448	0.0430	0.3149	0.0105	0.0239	0.0330	0.0453	0.0188
DLMLS	0.0194	0.0089	0.0470	0.0045	0.0122	0.0182	0.0234	0.0271
LTLS	0.3821	1.4328	10.1932	0.0096	0.0262	0.0357	0.0529	0.0109
LTS1	3.0370	5.2931	24.6491	0.0123	0.0288	0.0516	3.5929	0.0185
LTS2	9.2864	12.6741	62.8841	0.0128	0.0344	3.3251	15.0370	0.0228
MSE	46.2942	12.1067	83.6027	24.8605	38.2440	43.8599	54.0970	34.9213

Tabela 6.18: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji dwóch zmiennych przy prawdopodobieństwie pojawienia się dużych zakłóceń $\delta = 0.1$ (błędy typu I w wektorze wejściowym)

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.0173	0.0121	0.0664	0.0012	0.0086	0.0141	0.0214	0.0220
ALR	**0.0154**	0.0084	0.0432	0.0026	0.0095	0.0135	0.0190	0.0081
ATF	0.0852	0.0483	0.2717	0.0272	0.0531	0.0726	0.1037	0.1046
WLMLS	0.0161	0.0080	0.0466	0.0041	0.0099	0.0150	0.0202	0.0179
DLMLS	0.0272	0.0140	0.0646	0.0047	0.0168	0.0231	0.0365	0.0310
LTLS	0.0159	0.0076	0.0435	0.0029	0.0109	0.0137	0.0205	0.0060
LTS1	0.0424	0.0441	0.2255	0.0015	0.0114	0.0251	0.0612	0.0195
LTS2	0.0201	0.0306	0.2105	0.0019	0.0089	0.0135	0.0195	0.1394
MSE	0.0177	0.0119	0.0586	0.0017	0.0100	0.0141	0.0220	0.0027

poziom błędów. Przy zwiększonej ilości zakłóceń najmniejsze błędy otrzymuje się po użyciu metod WLMLS, LTLS i ALR, natomiast reszta odpornych algorytmów (z wyjątkiem ATF) uzyskuje błędy podobne, jak metoda klasyczna. Gdy w wektorze wejściowym znajduje się $\delta = 0.3$ dużych zakłóceń, sytuacja wśród najlepszych metod nie zmienia się, natomiast, traci wiarygodność algorytm z kryterium LMLS. Sieci wybrane, jako leżące najbliżej centrum, mają stosunkowo niski błąd jedynie dla algorytmów, dla których średni błąd nie jest zbyt wysoki.

Dla przypadku, w którym prawie połowę elementów ciągu uczącego zastąpiono jednorodnym szumem (tabela 6.21, najlepsze okazały się algorytmy LTLS i ALR. Również tutaj, podobnie jak w przypadku aproksymacji funkcji jednej zmiennej, błędy uzyskiwane przez sieci uczone wszystkimi algorytmami charakteryzuje podobny rząd wielkości, choć różnica pomiędzy najmniejszymi a największym (dla Mse) jest prawie czterokrotna. Z algorytmów odpornych najgorzej wypadły tu ATF, DLMLS i LMLS. Pozostałe powodowały uzyskiwanie błędów spomiędzy tych wartości. Nadal zastosowanie każdego z algorytmów odpornych poprawia działanie nauczonej nim sieci w stosunku do algorytmu klasycznego.

Wyniki uczenia sieci metodami odpornymi na danych nie zawierających zakłóceń, ani błędów grubych, zawarto w tabeli 6.22. Jak można zauważyć, wszystkie

Tabela 6.19: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji dwóch zmiennych przy prawdopodobieństwie pojawienia się dużych zakłóceń $\delta = 0.2$ (błędy typu I w wektorze wejściowym)

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.0235	0.0144	0.0731	0.0038	0.0133	0.0205	0.0291	0.0406
ALR	**0.0184**	0.0105	0.0633	0.0036	0.0113	0.0158	0.0229	0.0253
ATF	0.1499	0.0825	0.4525	0.0413	0.0854	0.1366	0.1871	0.1281
WLMLS	0.0163	0.0088	0.0462	0.0021	0.0109	0.0135	0.0196	0.0115
DLMLS	0.0391	0.0180	0.0963	0.0084	0.0278	0.0381	0.0503	0.0396
LTLS	0.0167	0.0090	0.0501	0.0025	0.0102	0.0160	0.0210	0.0135
LTS1	0.0216	0.0102	0.0589	0.0050	0.0140	0.0203	0.0282	0.0106
LTS2	0.0238	0.0161	0.0918	0.0032	0.0121	0.0200	0.0333	0.0158
MSE	0.0206	0.0144	0.0922	0.0044	0.0108	0.0179	0.0250	0.0132

Tabela 6.20: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji dwóch zmiennych przy prawdopodobieństwie pojawienia się dużych zakłóceń $\delta = 0.3$ (błędy typu I w wektorze wejściowym)

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.2477	0.1369	0.6504	0.0185	0.1330	0.2290	0.3382	0.4533
ALR	0.0218	0.0121	0.0810	0.0038	0.0137	0.0191	0.0264	0.0232
ATF	0.2251	0.1104	0.6164	0.0540	0.1430	0.2180	0.2776	0.3236
WLMLS	0.0210	0.0109	0.0706	0.0043	0.0130	0.0194	0.0260	0.0132
DLMLS	0.0518	0.0226	0.1347	0.0110	0.0353	0.0502	0.0682	0.0590
LTLS	**0.0174**	0.0075	0.0441	0.0029	0.0114	0.0172	0.0218	0.0229
LTS1	0.0305	0.0254	0.1692	0.0067	0.0164	0.0230	0.0379	0.0162
LTS2	0.0258	0.0140	0.0699	0.0037	0.0170	0.0225	0.0317	0.0070
MSE	0.0325	0.0184	0.0867	0.0047	0.0196	0.0271	0.0411	0.0079

Tabela 6.21: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji dwóch zmiennych przy prawdopodobieństwie pojawienia się danych zastąpionych jednorodnym szumem $\delta = 0.49$

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.0745	0.0680	0.5181	0.0123	0.0356	0.0515	0.0928	0.1720
ALR	**0.0327**	0.0336	0.2583	0.0034	0.0152	0.0231	0.0368	0.0148
ATF	0.0956	0.0528	0.2717	0.0273	0.0580	0.0798	0.1188	0.0585
WLMLS	0.0626	0.0675	0.4826	0.0039	0.0231	0.0413	0.0819	0.0075
DLMLS	0.0857	0.0572	0.4398	0.0239	0.0503	0.0738	0.1049	0.0505
LTLS	0.0380	0.0800	0.6810	0.0019	0.0085	0.0192	0.0381	0.0060
LTS1	0.0535	0.0783	0.3391	0.0001	0.0041	0.0190	0.0729	0.0093
LTS2	0.0661	0.0677	0.3879	0.0005	0.0217	0.0398	0.0951	0.0005
MSE	0.1274	0.0902	0.5952	0.0143	0.0686	0.1053	0.1616	0.0143

algorytmy wykazują zadowalającą zdolność do uczenia się na danych czystych. Podobnie, jak w przypadku aproksymacji funkcji jednej zmiennej, największy błąd uzyskany został dla metody Ltls, mimo to, jest to nadal niewielki i bliski zera średni poziom błędu.

6.1.3 Problem dwóch spiral

Dla zadania klasyfikacji, odmiennie niż w innych przypadkach, w tabelach zamiast błędu średniokwadratowego umieszczone zostały względne ilości błędnych klasyfikacji sieci (tabele 6.23 - 6.25). Są to wynik uśrednione z 500 symulacji. Sieci uczone tego zadania miały dwa wyjścia. Konkretną klasę oznaczał sygnał wysoki (1) na jednym z wyjść i niski (0) na drugim.

Przy testowaniu uznawano, że sieć rozpoznała obraz prawidłowo, jeżeli żadne z wyjść nie różniło się od wartości zadanej o więcej niż ± 0.4. Jeśli dla co najmniej jednego z nich różnica przekroczyła zadany próg, uznawano klasyfikację za błędną. W tabelach uwzględniono jedynie część odpornych algorytmów, gdyż pozostałe dawały wyniki bardzo zbliżone do algorytmu bez mechanizmu uodporniania.

Dla danych uczących zawierających najmniejszą ilość błędów (tabela 6.23, okazuje się, że ilość błędnych klasyfikacji wynosi w każdym przypadku poniżej 10%. Największy błąd uzyskuje tutaj algorytm klasyczny, natomiast testowane algorytmy

Tabela 6.22: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami aproksymacji funkcji dwóch zmiennych na danych bez zakłóceń

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
LMLS	0.0004	0.0002	0.0015	0.0000	0.0002	0.0003	0.0005	0.0004
ALR	0.0007	0.0005	0.0028	0.0001	0.0004	0.0006	0.0010	0.007
ATF	0.0021	0.0012	0.0068	0.0006	0.0012	0.0017	0.0028	0.0012
WLMLS	0.0004	0.0002	0.0016	0.0000	0.0002	0.0003	0.0005	0.0007
DLMLS	0.0006	0.0003	0.0021	0.0000	0.0003	0.0005	0.0007	0.0006
LTLS	0.0128	0.0236	0.1455	0.0003	0.0019	0.0046	0.0112	0.0114
LTS2	0.0006	0.0020	0.0172	0.0000	0.0000	0.0001	0.0002	0.0001
MSE	0.0003	0.0002	0.0011	0.0001	0.0002	0.0003	0.0004	0.0003

Tabela 6.23: Średnia względna ilość błędnych klasyfikacji sieci uczonych odpornymi algorytmami na problemie dwóch spiral przy prawdopodobieństwie pojawienia się błędów w obrazach uczących $\delta = 0.1$

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75
MSE	0.0763	0.0281	0.1762	0.0238	0.0571	0.0714	0.0905
LMLS	0.0696	0.0284	0.1952	0.0143	0.0524	0.0667	0.0857
ALR	0.0633	0.0312	0.2476	0.0095	0.0452	0.0619	0.0762
LTS1	**0.0594**	0.0254	0.1524	0.0095	0.0429	0.0571	0.0738
LTLS	0.0697	0.0310	0.1524	0.0143	0.0476	0.0619	0.0952

Tabela 6.24: Średnia względna ilość błędnych klasyfikacji sieci uczonych odpornymi algorytmami na problemie dwóch spiral przy prawdopodobieństwie pojawienia się błędów w obrazach uczących $\delta = 0.2$

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75
MSE	0.1802	0.0517	0.3286	0.0667	0.1452	0.1738	0.2048
LMLS	0.1660	0.0492	0.3429	0.0429	0.1286	0.1667	0.1952
ALR	0.1549	0.0445	0.2714	0.0571	0.1214	0.1524	0.1857
LTS1	0.1640	0.0627	0.3381	0.0476	0.1119	0.1643	0.2048
LTLS	**0.1527**	0.0436	0.2714	0.0667	0.1214	0.1476	0.1810

Tabela 6.25: Średnia względna ilość błędnych klasyfikacji sieci uczonych odpornymi algorytmami na problemie dwóch spiral przy prawdopodobieństwie pojawienia się błędów w obrazach uczących $\delta = 0.3$

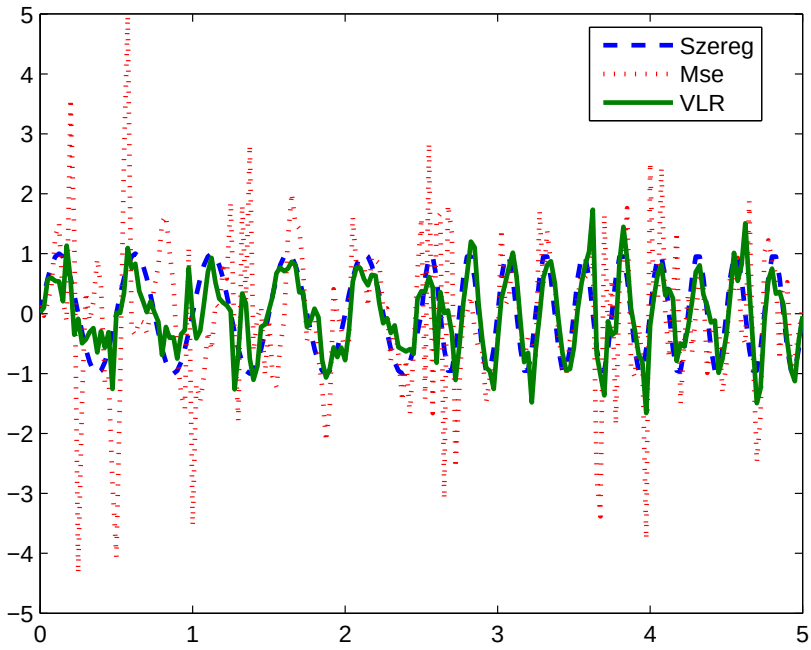
Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75
MSE	0.3331	0.0688	0.5571	0.1762	0.2857	0.3357	0.3905
LMLS	0.3064	0.0580	0.4667	0.1667	0.2619	0.3048	0.3476
ALR	**0.2670**	0.0579	0.3905	0.1429	0.2333	0.2571	0.3071
LTS1	0.3215	0.0981	0.5524	0.0952	0.2667	0.3190	0.3857
LTLS	0.2833	0.0629	0.4667	0.1571	0.2310	0.2738	0.3190

reprezentują wyrównaną jakość działania, mając od poniżej 6 do 7% nieprawidłowo rozpoznanych obrazów testowych.

Tabela 6.24 pokazuje, że w tym przypadku znów algorytm klasyczny odstaje niewiele, bo o 2-2.5 punktu procentowego niepoprawnych klasyfikacji od algorytmów odpornych, które nadal reprezentują podobny poziom. Najlepiej wypada tu algorytm ALR, najgorzej zaś LMLS.

Podobna sytuacja zachodzi dla sieci uczonych na danych z maksymalną ilością błędów (tabela 6.25). Również w tym przypadku najlepszy z algorytmów odpornych to ALR, najgorzej zaś działa metoda klasyczna. Różnica pomiędzy nimi to aż 6.5% błędnych klasyfikacji mniej dla sieci uczonych metodą ALR.

Widać, że pomimo, iż opracowane algorytmy nie zostały skonstruowane do uodparniania procesu uczenia dla zadania klasyfikacji, zwłaszcza z małą ilością klas, okazuje się, że niektóre z nich mogą spowodować kilkuprocentowy wzrost skuteczności uczonych nimi sieci, w porównaniu z algorytmem klasycznym. Zauważyć jednak



Rysunek 6.6: Odporny algorytm VLR w porównaniu z algorytmem klasycznym (Mse), (predykcja szeregów czasowych, $\delta = 0.1$).

należy, że odnotowane zmiany są niezbyt duże, a względny spadek poziomu błędnych klasyfikacji wynosi ok. 20%.

6.1.4 Predykcja szeregów czasowych

Do adaptacji sieci mającej przewidywać kolejne wartości zadanych sygnałów użyto algorytmu klasycznego, oraz odpornego algorytmu ze zmiennym współczynnikiem uczenia VLR. Odporny algorytm łączony był zarówno z kryterium Mse, jak i LMLS. Wyniki będące średnią z 500 symulacji przedstawiono w tabelach 6.26 - 6.28. Jako, że zebrane błędy liczone był po 10 prezentacjach wszystkich obrazów uczących wpływ dużych zakłóceń był zwielokrotniony. W tabelach nie umieszczono błędów dla sieci uczonych algorytmem klasycznym, a jedynie podano ich poziom w podpisach, gdyż ich rząd kilkusetkrotnie nawet przekraczał poziom błędów popełnianych przez sieci uczone algorytmami odpornymi. Przykładowy rezultat działania sieci przestawiono na rysunku 6.6.

Tabela 6.26: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami predykcji szeregów czasowych przy prawdopodobieństwie pojawienia się dużych zakłóceń w obrazach uczących $\delta = 0.1$. Poziom błędu algorytmu klasycznego 1E+62.

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
VLR (Mse)	0.3353	0.0845	0.5507	0.1803	0.2672	0.3330	0.3912	0.1148
VLR (LMLS)	0.3168	0.0683	0.4935	0.1940	0.2616	0.3139	0.3625	0.0827

Tabela 6.27: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami predykcji szeregów czasowych przy prawdopodobieństwie pojawienia się dużych zakłóceń w obrazach uczących $\delta = 0.2$. Poziom błędu algorytmu klasycznego 1E+251.

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
VLR (Mse)	0.4584	0.0939	0.6606	0.2546	0.3996	0.4630	0.5257	0.3309
VLR (LMLS)	0.3711	0.0545	0.5393	0.2674	0.3250	0.3646	0.4111	0.1145

Analizując uzyskane dane, stwierdzić można, że w tym przypadku propagujący się w czasie kilku prezentacji całego ciągu uczącego wpływ błędów grubych na działanie sieci adaptowanej algorytmem klasycznym jest najwyraźniejszy. Błędy w tym przypadku sięgnęły poziomu $10^3 00$ (oczywiście chodzi o błąd kwadratowy). Obie wersje odpornego algorytmu radzą sobie niewspółmiernie lepiej uzyskując błąd rzędu 0.3-0.4, przy czym mniejsze wartości uzyskano dla metody z kryterium LMLS.

Na tym przykładzie dobrze widać, że nieznaczna nawet modyfikacja algorytmu uczenia skutkować może istotną poprawą uzyskanych rezultatów. Jest to szczególnie ważne, gdy, podobnie, jak w tym przypadku, do ustalenia wyjścia sieci wykorzystywanych jest kilka wartości obrazów uczących (w tym wypadku elementów szeregów czasowych). Jeżeli bowiem, kilka wejść sieci jest obarczonych błędem, można się spodziewać, że również jej wyjście będzie nieprawidłowe w stopniu znacznie większym. Zastosowanie nawet prostego uodpornienia metody adaptacji, pozwala na osiągnięcie znacznie lepszych, a więc zbliżonych do zadanych, wartości wyjść sieci.

Tabela 6.28: Błąd średniokwadratowy sieci uczonych odpornymi algorytmami predykcji szeregów czasowych przy prawdopodobieństwie pojawienia się dużych zakłóceń w obrazach uczących $\delta = 0.3$. Poziom błędu algorytmu klasycznego 1E+300.

Algorytm	Średnia	σ	Max	Min	q=0.25	q=0.50	q=0.75	S.centr.
VLR (Mse)	0.4886	0.0916	0.7340	0.3237	0.4222	0.4768	0.5344	0.1335
VLR (LMLS)	0.4264	0.0489	0.5333	0.2974	0.3901	0.4254	0.4583	0.1601

6.2 Podsumowanie wyników symulacji

Analizując zebrane w trakcie przeprowadzania eksperymentów numerycznych wyniki, pokusić się można o kilka stwierdzeń dotyczących opracowanych odpornych algorytmów uczenia i rezultatów ich działania w różnych sytuacjach.

Po pierwsze, stwierdzić należy, że wszystkie badane odporne metody uczenia spełniły pierwszy z postawionych na wstępie postulatów, a więc potrafiły nauczyć sieć stosunkowo dobrze na danych czystych i pozbawionych jakichkolwiek błędów. W praktyce działają one w takim przypadku podobnie, jak algorytm klasyczny, a więc można stosować je bez ryzyka, gdy nie ma pewności, czy zakłócenia rzeczywiście mogą wystąpić. Najgorsze wyniki uzyskiwał tu algorytm LTLS, w którym odrzucana jest stosunkowo duża ilość obrazów uczących, nawet jeśli nie zawierają one błędów. Jego działania poprawia jednak zwiększenie ilości obrazów w ciągu uczącym, choć trzeba pamiętać, że nie zawsze jest to możliwe.

Co do spełnienia postulatu drugiego, można jedynie powiedzieć, że rzeczywiście opracowane metody działają lepiej niż algorytm klasyczny. Testowany, dla porównania z nowymi metodami, algorytm LMLS, radził sobie, w większości przypadków, zdecydowanie gorzej niż opracowane metody i nigdy nie okazał się najlepszy. Trzeba jednak pamiętać, że w wersji oryginalnej był on przeznaczony do uczenia "on-line" i w niniejszej pracy wypróbowywany był przy uczeniu skumulowanym jedynie dla porównania.

Bardzo często najlepsze rezultaty dawały metody, w których dodatkowo tłumiony jest postęp procesu uczenia, a więc ATF i ALR. Pierwsza z wymienionych metod nie sprawdzała się przy uczeniu na danych zawierających błędy w wektorze wejściowym, choć w innych przypadkach błąd uzyskiwany przez uczone nią sieci był najmniejszy.

Trzeba pamiętać, że w algorytmie tym kluczowy jest dobór prędkości zmian funkcji aktywacji. Druga z metod, ALR, mimo, iż czasem najlepsza, dość często dawała zupełnie błędne wyniki. Mimo więc, że można za jej pomocą osiągnąć, w większości przypadków, znaczną poprawę działania sieci, trzeba się liczyć z niebezpieczeństwem, że może okazać się rozbieżna i wtedy rezultaty jej działania będą nie do przewidzenia.

Warte uwagi są rezultaty uzyskane przy uczeniu sieci algorytmem WLMLS. W żadnym przypadku nie uzyskuje on najmniejszych błędów, ale należy zwrócić uwagę na fakt, że poziom błędów jest dla niego bardzo stabilny. Zwykle mieści się wśród lepszej połowy metod, nigdy zaś nie okazuje się najgorszy. Można więc stwierdzić, że stosować go należy, gdy bardziej zależy nam na tym, aby sieć nauczona była w miarę dobrze w każdym przypadku, niż by osiągnąć bardzo niski średni błąd przy wielu uczeniach.

Należy przypomnieć, że testowany algorytm DLMLS, jest ulepszoną wersją metody LMLS. Minimalizuje on to samo kryterium, jednak w jego przypadku proces uczenia odbywa się szybciej. Kluczowe więc okazuje się kryterium stopu, które w niniejszych symulacjach przyjęto jako maksymalną ilość epok (oczywiście były również inne, dodatkowe kryteria, dotyczące polepszenia funkcji kryterialnej, bądź wartości jej gradientu, które mogły przerwać proces uczenia wcześniej). Omawiany algorytm doprowadzał sieć bliżej poszukiwanego optimum, w porównaniu z LMLS, co zwykle powodowało błąd mniejszy, choć czasem większy, niż w tym drugim przypadku.

Wśród algorytmów z przycinaniem błędów, najlepsze rezultaty uzyskiwał LTLS. W jego przypadku bowiem zaangażowane zostały dwa mechanizmy usuwania największych błędów, co jak można zauważyć, zaowocowało wzrostem skuteczności w porównaniu z metodą LTS. W przypadku tej ostatniej wariant LTS1 okazał się bardzo wrażliwy na właściwy dobór maksymalnego ograniczenia błędów w danych, tak więc należy polecić raczej stosowanie metody LTS2. Trzeba jednak zaznaczyć, że wraz z LTLS zawodzą one przy dużych ilościach zakłóceń typu II.

Badany na zadaniu predykcji szeregów czasowych algorytm VLR, pokazał, że nawet tak prosta modyfikacja sposobu uczenia umożliwia znaczne zwiększenie odporności sieci. Jest to szczególnie widoczne w sytuacji, gdy sieć wykorzystuje w jednym momencie informacje pochodzące z kilku elementów ciągu uczącego.

6.3 Inne czynniki wpływające na odporność sieci

Wielokrotnie już wspominano, że nie wiadomo, co modeluje sieć neuronowa uczona na mocno zakłóconych danych i jak otrzymane w wyniku jej trenowania odpowiedzi mają się do wartości prawidłowych. Opisane i przetestowane odporne algorytmy mogą, jak się okazało, dość znacznie poprawić jakość uczenia sieci w takich warunkach, niemniej należy uzmysłowić sobie, że poza metodami uczenia, istnieją również inne czynniki, które mogą mieć potencjalnie wpływ na odporność sieci na błędy grube.

Większość parametrów algorytmu uczenia, takich jak funkcja celu, współczynnik uczenia, czy funkcje aktywacji neuronów, którymi można sterować w celu uzyskania odporności na duże zakłócenia, opisana została już w rozdziale poświęconym opracowanym odpornym algorytmom uczenia sieci (Rozdział 4). Tutaj skupimy się jedynie na pozostałych czynnikach.

Po pierwsze, wydaje się, że wpływ na to, jak dobrze sieć dopasowuje się do danych uczących, a zarazem, jakie są jej zdolności uogólniania, ma jej rozmiar. W przypadku ciągu uczącego zawierającego błędy grube, dokładne dopasowanie sieci do danych uczących prowadzi w rezultacie do nieprawidłowego działania, można się więc spodziewać, że im większy będzie rozmiar sieci, tym większa możliwość przeuczenia, a w związku z tym większy błąd popełniany dla danych czystych. W tabeli 6.29 pokazano średni błąd z 500 symulacji dla sieci o różnej liczbie neuronów ukrytych popełniany dla danych zawierających grube zakłócenia[1]. Na podstawie uzyskanych wyników stwierdzić można, że rzeczywiście istnieje tu prosta zależność mówiąca, że im większa (poczynając od minimalnej dla danego zadania) liczba neuronów, tym gorsze rezultaty uzyskano ucząc sieć na danych z zakłóceniami.

Drugi problem, jaki pojawia się przy uczeniu z błędami grubymi, to rozmiar ciągu uczącego. Znów, wydaje się, że gdy ciąg uczący jest stosunkowo niewielki, sieć będzie dopasowywała się dokładnie do wszystkich obrazów, a więc wpływ błędów grubych będzie największy. Z kolei dla dużego ciągu, sieć powinna modelować tylko główny trend w danych, a więc efekty działania zakłóceń nie będą tak wyraźne. Jak się okazuje, założenie takie jest uzasadnione, co obrazuje tabela 6.30. W przypadku istnienia bardzo wielu obrazów uczących błąd sieci uczonej klasycznym algorytmem staje się kilkudziesięciokrotnie mniejszy niż dla ciągu najkrótszego.

[1]Dokładny opis sposobu uzyskania wyników znajduje się w rozdziale 5

Tabela 6.29: Zależność średniego błędu sieci od ilości neuronów ukrytych przy uczeniu aproksymacji funkcji jednej zmiennej z $\delta = 0.1$ błędów w wektorze wejściowym

Il. neuronów	Mse	σ	Max	Min
25	0.0851	0.0335	0.1905	0.0276
20	0.0657	0.0270	0.1405	0.0217
15	0.0544	0.0222	0.1101	0.0198
10	0.0398	0.0195	0.0924	0.0094
5	0.0312	0.0133	0.0755	0.0117

Tabela 6.30: Zależność średniego błędu sieci od wielkości ciągu uczącego, przy uczeniu aproksymacji funkcji jednej zmiennej z $\delta = 0.1$ błędów w wektorze wejściowym

Wielkość ciągu	Mse	σ	Max	Min
81	0.2216	0.1560	0.9603	0.0220
201	0.0758	0.0415	0.2185	0.0101
801	0.0199	0.0083	0.0550	0.0058
2001	0.0079	0.0027	0.0168	0.0033
4001	0.0039	0.0017	0.0111	0.0015

Kolejne zagadnienie, również związane w pewien sposób z możliwością przeuczenia sieci, to dobór odpowiedniej długości procesu uczenia, a więc ustawienie kryteriów stopu. W tabeli 6.31 pokazano wpływ ilości epok uczenia sieci na błąd popełniany, przy założeniu, że dane uczące zawierały błędy grube. Ogólny trend pokazuje, że zbyt późne przerwanie trenowania skutkuje zwiększonym poziomem błędów. Trzeba pamiętać, że również w tym przypadku badanie rozpoczęto od wartości minimalnej ilości epok, którą uznano za wystarczającą do nauczenia sieci na danych czystych.

Dodatkowo można zastanawiać się, czy np. techniki odpowiedzialne za zapobieganie zjawisku przeuczenia, takie jak wczesne przerywanie treningu, czy regularyzacja mogą pozwolić na zwiększenie odporności sieci na błędy w danych. Jest to dość złożony problem i w niniejszej pracy nie zajmowano się nim dokładnie. Wykonano jedynie badania numeryczne, które nie wykazały, aby stosowanie podziału ciągu uczącego na dane uczące i walidacyjne, służące przerywaniu uczenia, gdy różnica pomiędzy błędami dla tych dwóch podzbiorów zacznie się zwiększać miało jakikolwiek wpływ na odporność na błędy grube.

Tabela 6.31: Zależność średniego błędu sieci od długości procesu trenowania, przy uczeniu aproksymacji funkcji jednej zmiennej z $\delta = 0.1$ błędów w wektorze wejściowym

Epok	Mse	σ	Max	Min
10	0.0595	0.0242	0.1378	0.0208
30	0.0744	0.0347	0.2056	0.0207
40	0.0742	0.0277	0.1419	0.0223
50	0.0798	0.0290	0.2104	0.0372
70	0.0832	0.0333	0.2135	0.0208
90	0.0950	0.0383	0.2183	0.0309
110	0.0938	0.0365	0.2462	0.0303
130	0.0990	0.0398	0.2509	0.0418

Z drugiej strony, nie stwierdzono również, że przy stosowaniu opracowanych odpornych algorytmów uczenia, sieć trudniej się przeucza. Trudno więc jednoznacznie stwierdzić, czy taki związek zachodzi, a jeśli tak, to jak wygląda owa relacja.

Rozdział 7

Zakończenie

7.1 Podsumowanie

W pracy niniejszej z powodzeniem zajęto się opracowaniem, odpornych na błędy w danych, algorytmów uczenia sieci neuronowych jednokierunkowych. Grube błędy mogące wystąpić w ciągu uczącym są o tyle niebezpieczne, że niewykryte mogą, choć nie muszą, spowodować całkowicie błędne działanie tradycyjnych metod uczących. Sieć tworzy wtedy model daleki od prawidłowego, odpowiadający charakterystyce zakłóceń, które pojawiły się w danych. Grube błędy pojawiające się często w danych eksperymentalnych (pochodzących np. z nauk przyrodniczych, społecznych, medycyny) spowodowane mogą być różnymi czynnikami i powstawać na każdym etapie, a więc w trakcie procesu akwizycji, przetwarzania, składowania, czy interpretacji wyników pomiarów.

Aby pokonać ten problem, zdefiniowano cel pracy, jakim było opracowanie algorytmów uczenia sieci neuronowych, które byłby odporne na błędy mogące pojawić się w ciągu uczącym. Sformułowano więc trzy postulaty, jakie powinny zostać spełnione przez odporną metodę, mianowicie: każda z nich powinna działać dobrze dla danych pozbawionych zakłóceń, jak również działać (znacznie) lepiej niż algorytm klasyczny dla danych zawierających duże zakłócenia. Ponadto wszystkie one, muszą mieć wbudowane elementy, pomagające w uzyskiwaniu odporności, w samą strategię uczenia.

Podczas realizacji tego zadania zaprojektowano szereg nowych odpornych algorytmów uczenia, wykorzystujących różne mechanizmy w celu zmniejszenia wrażliwości na pojawiające się w ciągu uczącym zakłócenia. Można stwierdzić, że dla każdego

z nich założenia postawione na wstępie zostały spełnione. Najkrócej rzecz ujmując, można powiedzieć, że opracowane nowe odporne metody:

- umożliwiają zwiększenie odporności na błędy w danych uczących;
- działają lepiej niż algorytm klasyczny w obecności błędów w danych;
- działają lepiej niż istniejące algorytmy odporne, umożliwiając uczenie skumulowane algorytmami drugiego rzędu;
- uwzględniają różne typy zakłóceń;
- wykorzystują różne mechanizmy zapewniające odporność.

Zaproponowane algorytmy przeznaczone są do uczenia sieci jednokierunkowych sigmoidalnych. Przy ich tworzeniu wykorzystano metody ze zmodyfikowaną funkcją kryterialną opartą na estymatorze LTS, metody ze zmiennym współczynnikiem uczenia i zmienną funkcją aktywacji neuronów, dedykowany algorytm drugiego rzędu dla zmodyfikowanej funkcji błędu LMLS, oraz metodę ze wstępną analizą danych estymatorem MCD.

Aby zweryfikować działanie opracowanych algorytmów uczenia, przeprowadzono wiele tysięcy cykli symulacji, w których sieci neuronowe trenowane były na danych zawierających różne ilości zakłóceń różnych typów, przy zmiennych warunkach początkowych. Działanie opracowanych metod porównywano między sobą i z algorytmem klasycznym.

Jak wykazały eksperymenty symulacyjne, opracowane metody uczenia, umożliwiają uzyskanie stosunkowo dobrych rezultatów działania sieci, nawet w przypadku, gdy uczone one były przy maksymalnych, a więc pojawiających się z prawdopodobieństwem $\delta = 0.3$ dużych zakłóceniach. Jest to o tyle ważny wynik, że w rzeczywistych zastosowaniach dane bardziej zanieczyszczone spotyka się niezwykle rzadko. Dodatkowo, mimo, iż algorytmy owe nie były do tego celu projektowane, część z nich stosowana może być również przy zadaniach klasyfikacji o niewielkiej ilości klas, choć uzyskana poprawa działania nie jest wtedy jednak aż tak znaczna.

Na koniec warto przypomnieć, że przed rozpoczęciem realizacji tej pracy stan wiedzy dotyczący odpornych algorytmów uczenia zamykał się w zaledwie kilku publikacjach, a tematyka ta nie była i nadal jeszcze nie jest, odpowiednio opisana

i przebadana. Niemniej, jak pokazały zamieszczone w pracy rezultaty, czasem nawet niewielkie modyfikacje algorytmów uczenia mogą znaczącą wpłynąć na ich działanie przy trenowaniu sieci na mocno zanieczyszczonych danych uczących.

7.2 Proponowane dalsze kierunki badań

Nietrudno stwierdzić, że niniejsze praca nie wyczerpuje bardzo rozległej i potencjalnie dającej spore możliwości, tematyki uodparniania algorytmów uczenia sieci neuronowych. Mimo, iż obrazowo mówiąc, opracowane w ramach niniejszych badań odporne algorytmy uczenia, stanowią ponad połowę istniejących rozwiązań, wydaje się, że dziedzina ta ma przed sobą jeszcze duże perspektywy rozwoju.

Inne typy sieci

Po pierwsze, opracowane odporne algorytmy dotyczą jedynie sieci neuronowych jednokierunkowych sigmoidalnych, podczas, gdy obecnie można wymienić jeszcze co najmniej kilka innych ich klas. Najprostsze podejście polegać by mogło na zastosowaniu odpowiednio uodpornionych nowych funkcji kryterialnych, wszędzie tam, gdzie mamy do czynienia z minimalizacją pewnej funkcji błędu.

Inne typy problemów

Opracowane algorytmy przeznaczone są do sytuacji, w których zbiór dopuszczalnych wartości wyjścia nauczonej sieci jest ciągły lub bardzo liczny. Można się zastanawiać nad stworzeniem dedykowanych algorytmów mających na celu poprawne uczenie np. zadana klasyfikacji przy niewielkiej liczbie klas.

Sieć neuronowa w ujęciu semiparametrycznym a zagadnienie odporności

Zastanawiając się nad uodparnianiem sieci przejść można również do problemu nieparametrycznego. Sieć neuronowa semiparametryczna zdefiniowana może zostać jako sieć zawierająca w swej strukturze zarówno komponenty parametryczne, jak i nieparametryczne. Aby wprowadzić do sieci element nieparametryczny, załóżmy więc, że funkcje aktywacji neuronów ukrytych nie mają sprecyzowanego kształtu, innymi słowy nie są ustalone i mogą podlegać zmianom w procesie uczenia sieci, przy

utrzymaniu niezbędnych założeń dotyczących ich gładkości. W tej sytuacji traktować możemy problem uczenia jako minimalizację pewnej funkcji kryterialnej zależnej nie tylko od wektora parametrów (wag) sieci, ale i od kształtu funkcji aktywacji neuronów.

Istotna wydaje się tu konkluzja, że odpowiedni dobór owych funkcji niewątpliwie może zwiększyć odporność sieci na występowanie danych odstających (uproszczenie problemu poprzez sparametryzowanie postaci funkcji aktywacji przedstawiono w niniejszej pracy w algorytmu ATF).

Zdolności uogólniania i rozmiar sieci

Przy omawianiu czynników mających wpływ na uczenie sieci na danych zanieczyszczonych, wzmiankowano jedynie parametry takie, jak odpowiednia liczba neuronów ukrytych, ilość elementów ciągu uczącego, czy wreszcie problem kryterium stopu algorytmu uczącego. Dalsze badania mogłyby dotyczyć metod uzyskujących odporność na podstawie automatycznego ustalania odpowiedniej (a więc, z punktu widzenia problemu błędów grubych, nie za dużej) wielkości uczonej struktury.

Dobór ilości elementów ciągu uczącego nie zawsze jest dowolny, często bowiem dysponujemy skończoną ilością danych z eksperymentów, warto jednak pamiętać, że im jest ona większa, tym mniejszy wpływ na uczenie mogą mieć zakłócenia.

Połączenie zaproponowanych mechanizmów uodparniania

Na podstawie zaproponowanych w niniejszej pracy mechanizmów służących zwiększaniu odporności, udało by się być może sporządzić pewną ich syntezę umożliwiającą uzyskiwanie jeszcze lepszych rezultatów. Pomimo, iż część z nich była już łączona w opracowanych algorytmach, istnieje jeszcze wiele niewykorzystanych kombinacji, z których część, być może, ma w sobie potencjał do wykorzystania w odpornych algorytmach uczenia.

Bibliografia

[1] Andreou, P.C., Charalambous C., Martzoukos, S.H.: Robust Artificial Neural Networks for Pricing of European Options, Computational Economics (2006) 27: pp. 329–351, Springer 2006

[2] Arbib, M.(ed.): The handbook of brain theory and neural networks, MIT Press, 1995

[3] Battiti, R.: First- and second-order methods for learning between steepest descent and newton's method, Neural Computation, vol.4(2), pp.141-166, 1992

[4] Battiti, R.: Accelerated backpropagation learning: Two optimization methods, Complex Systems, vol. 3, pp. 331-342, 1989

[5] Chan, L.W., Fallside, F.: An adaptive training algorithm for backpropagation networks, Computers, Speech and Language, vol. 2, pp. 205-218, 1987

[6] Charalambous, C.: Conjugate gradient algorithm for efficient training of artificial neural networks, IEE Proceedings-G, Vol. 139, No. 3,pp. 301-310, 1992

[7] Chen, D.S., Jain, R.C.: A robust back propagation learning algorithm for function approximation, IEEE Transactions on Neural Networks, vol. 5, pp. 467-479, May 1994

[8] Chen, S., Cowan, C.F.N., Grant, P. M.: Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks, IEEE Trans. on Neural Networks, vol. 2, no. 2, pp. 302-309, 1991

[9] Chuang, C., Su, S., Hsiao C.: The Annealing Robust Backpropagation (ARBP) Learning Algorithm, IEEE Transactions on Neural Networks, vol. 11, pp.1067-1076, September 2000

[10] Chuang, C.C., Jeng, J.T., Lin, P.T.: Annealing robust radial basis function networks for function approximation with outliers, Neurocomputing, vol. 56, pp. 123–139, 2004

[11] David Sanchez, V.A.: Robustization of a learning method for RBF networks, Neurocomputing, vol.9, pp. 85-94, 1995

[12] Dennis, J.E., Schnabel, R.B.: Numerical methods for unconstrained optimization and nonlinear equations, Prentice-Hall, NJ, 1983

[13] Elman, J. L.: Finding structure in time, Cognitive Science, vol. 14, pp. 179-211, 1990

[14] Fahlman, S.E.: Faster learning variations on backpropagation: empirical study, Proc. Connesctionist Models Summer School, Morgan Kaufmann, 1988

[15] Fletcher, R., and Reeves, C.M.: Function minimization by conjugate gradients, Comput. J., vol. l, pp. 149-154, 1964

[16] Gill, P., Murray, W., Wright, M.: Practical Optimization, Academic Press, NY, 1981

[17] Gori, M., Tesi, A.: On the problem of local minima in backpropagation, IEEE Tans. Pattern Analysis and Machine Intelligence, vol. 14, pp. 76-65, 1992

[18] Greblicki, W.: Learning to recognize patterns with a probabilistic teacher, Pattern Recogn., vol.12, pp. 159-164, 1980

[19] Hagan, M. T., Demuth, H. B., Beale, M. H.: Neural Network Design, Boston, MA: PWS Publishing, 1996

[20] Hagan, M.T., Menhaj, M.B.: Training Feedforward Networks with the Marquardt Algorithm, IEEE Trans. on Neural Networks, vol. 5, no. 6, 1994

[21] Hampel, F.R., Ronchetti, E.M., Rousseeuw, P.J., Stahel, W.A.: Robust Statistics the Approach Based on Influence Functions, John Wiley & Sons, New York, 1986

[22] Hawkins, D.M.: Identification of Outliers, Chapman and Hall, London, 1980

[23] Haykin, S.: Neural Networks - A Comprehensive Foundation, Second Edition, Prentice Hall, N.J., 1999

[24] Hebb, D.O.: The Organization of Behavior, Wiley, NY, 1949

[25] Hertz, J., Krogh, A., Palmer, R.G.: Wstęp do teorii sieci neuronowych, WNT, Warszawa, 1993

[26] Hopfield, J.J.: Neural networks and physical systems with emergment colective computational abilities, Proc. of the National Academy of Science, USA 79, pp. 2554-2558, 1982

[27] Hornik, K., Stinchconbe, M., White, H.: Multilayer feedforward networks are universal approximators, Neural Networks, vol. 2, pp. 359-366, 1989

[28] Huber, P.J.: Robust Statistics, Wiley, New York, 1981

[29] Jacobs, R.A.: Increased rates of convergence through learning rate adaptation, Neural Networks, vol. 1, pp. 295-307, 1988

[30] Kearns, M., Li, M.: Learning in the presence of malicious errors, Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, pp. 267-280, 1988

[31] Kohonen, T.: Self-Organization and Associative Memory, Springer-Verlag, 1987

[32] Kurzyński, M.: Rozpoznawanie obiektów - metody statystyczne, Oficyna Wydawnicza PWr, Wrocław, 1997

[33] Li, J., Michel, A. N., Porod, W.: Analysis and synthesis of a class of neural networks: linear systems operating on a closed hypercube, IEEE Trans. on Circuits and Systems, vol. 36, no. 11, pp. 1405-1422, 1989

[34] Liano, K.: Robust error measure for supervised neural network learning with outliers, IEEE Transactions on Neural Networks, vol. 7, pp. 246-250, Jan. 1996

[35] Lippmann, R.: An introduction to computing with neural nets, IEEE ASSP Magazine, vol. 4, pp. 4-22, 1987

[36] Liu, J., Gader, P.: Outlier Rejection with MLPs and Variants of RBF Networks, pp.680-683, IEEE 2000

[37] Marquardt, D.: An algorithm for least squares estimation of non-linear parameters, J. Soc. Ind. Appl. Math., pp. 431-441, 1963

[38] McCulloch, W.S., Pitts, W.H.: A logical of calculus ideas immanent in nervous activity, Bull. Math. Biophysics, vol. 5, pp. 115-119, 1943

[39] Mili, L., Cheniae, M., Vichare, N.S., Rousseeuw, P.J.: Robust State Estimation Based on Projection Statistics, IEEE Transactions on Power Systems, Vol. 11, No.2, May 1996.

[40] Minsky, M.L., Papert, S.A.: Perceptrons, MIT Press, 1969

[41] Moller, M.: A scaled conjugate gradient algorithm for fast supervised learning, Neural Networks, vol. 6(4), pp. 525-533, 1993

[42] Olive, D.J.: Applied Robust Statistics, praca niepublikowana, 2007

[43] Olive, D.J., Hawkins, D.M.: Robustifying Robust Estimators, 2007

[44] Osowski, S.: Sieci neuronowe w ujęciu algorytmicznym, WNT, Warszawa 1996

[45] Parker, D.B: Optimal algorithms for adaptive networks: Second order backpropagation, second order direct propagation, and second order hebbian learning, In Proc. of the IEEE International Conference on Neural Networks, vol. 2, pp. 593-600, 1987

[46] Pernia-Espinoza, A.V., Ordieres-Mere, J.B., Martinez-de-Pison, F.J., Gonzalez-Marcos, A.: TAO-robust backpropagation learning algorithm, Neural Networks, vol.18, pp. 191-204, 2005

[47] Polak, E.: Computational methods in optimisation: a unified approach, Academic Press, NY, 1971

[48] Riedmiller, M., Braun, H.: RPROP - a fast adaptive learning algorithm, Technical Report, University Karlsruhe, 1992

[49] Rosenblatt, F.: Principles of Neurodynamics, Spartan, NY, 1962

[50] Rousseeuw, P.J.: Least median of squares regression, Journal of the American Statistical Association, 79, pp. 871-880, 1984

[51] Rousseeuw, P.J.: Multivariate Estimation with High Breakdown Point, Mathematical Statistics and Applications, vol. B, Reidel, the Netherlands, 1985

[52] Rousseeuw, P.J., Leroy, A.M.: Robust Regression and Outlier Detection. Wiley, New York, 1987

[53] Rousseeuw, P.J., Van Driessen, K.: A Fast Algorithm for the Minimum Covariance Determinant Estimator, Technometrics vol. 41, pp. 212-223., 1999

[54] Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation, in Parallel Distribution Processing: Explorations in the Microstructures of Cignition, vol. 1, MIT Press, Cambridge, 1986

[55] Rusiecki, A.L.: Robust Learning Algorithm with the Variable Learning Rate, ICAISC 2006, Artificial Intelligence and Soft Computing, pp. 83-90, Warszawa 2006

[56] Rusiecki, A.L.: Robust LTS Backpropagation Learning Algorithm, IWANN 2007, LNCS, vol.4507, pp. 102-109, Springer-Verlag 2007

[57] Rusiecki, A.L.: Badanie odpornych algorytmów uczenia sieci neuronowych, Raport PWr, preprinty nr 23/2005, Wrocław 2005

[58] Rusiecki, A.L.: Fault tolerant feedforward neural network with median neuron input function, Electronics Letters, Vol. 41, No. 10, pp. 603-605, 2005

[59] Rutkowska, D., Piliński, M., Rutkowski, L.: Sieci neuronowe, algorytmy genetyczne i systemy rozmyte, Wydawnictwo Naukowe PWN, Warszawa-Łódź, 1997

[60] Saito, K., Nakano, R.: Partial BFGS and efficient step-length calculation for three-layer neural networks, Neural Computation vol. 9(1), 1997

[61] Siegel, A.F.: Robust Regression using Repeated Medians, Biometrika, 69, pp.242-244, 1982

[62] Skubalska-Rafajłowicz, E.: Samoorganizujące sieci neuronowe, Biocybernetyka i Inżynieria Biomedyczna, Tom 6, Sieci neuronowe, Akademicka Oficyna Wydawnicza PLJ, Warszawa, 2000

[63] Stromberg, A.J., Ruppert, D.: Breakdown in nonlinear regression. J. Amer. Statist. Assoc. 87, pp. 991–997, 1992

[64] Tadeusiewicz, R.: Sieci neuronowe, Akademicka Oficyna Wydawnicza, Warszawa, 1993

[65] Van der Smagt, P.P.: Minimization methods for training feedforward neural networks, Neural Networks, vol. 7, pp. 1-11, 1994

[66] Vogl, T.P., Mangis, J.K., Rigler, A.K., Zink, W.T., Alkon, D.L.: Accelerating the convergence of the backpropagation method, Biological Cybernetics, vol. 59, pp. 256-264, 1988

[67] Wang, H., Suter, D.: A Novel Robust Method for Large Numbers of Gross Errors, ICARCV '02, Singapore, 2002

[68] Wang, J.H., Jiang, J.H., Yu, R.Q.: Robust back propagation algorithm as a chemometric tool to prevent the overfitting outliers, Chemometrics and Intelligent Laboratory Systems, vol.34, pp. 109-115, 1996

[69] Walczak B.: Neural networks with robust backpropagation learning algorithm, Analytica Chimica Acta 322, pp. 21-29, 1996

[70] Widrow, B.: Generalization and Information Storage in Networks of Adaline 'Neurons', Self-Organizing Systems 1962, 435-461, 1962

[71] Widrow, B., Hoff, M.E.: Adaptive switching circuits, 1960 IRE WESCON Convention Record, part 4, pp. 96-104, NY, 1960

[72] Wilamowski, B.M., Iplikci, S., Kaynak O., Efe, M.O.: An Algorithm for Fast Convergence in Training Neural Networks, IEEE, pp. 1778-1782, 2001

[73] Yohai, V., Zamar, R.: High breakdown-point estimates of regression by means of the minimization of an efficient scale, Journal of the American Statistical Association, 83(402), pp. 406–413, 1988

[74] Yu, X., Bui, T.D. i Krzyzak, A.: Robust Estimation for Range Image Segmentation and Reconstruction, IEEE Trans. Pattern Analysis and Machine Intelligence, 16(5), pp.530-538, 1994