Kraków 26.06.2024

dr hab. inż. Ernest Jamro prof. AGH
Instytut Elektroniki
Wydział Informatyki Elektroniki i Telekomunikacji
AGH, Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
al. Mickiewicza 30, 30-059 Kraków

Recenzja

rozprawy doktorskiej mgra inż. **Grzegorza Rafała Deca** pt. *Sprzętowa implementacja sieci LSTM*

Niniejszą recenzje opracowano na wniosek Rady Dyscypliny Informatyka Techniczna i Telekomunikacja Politechniki Rzeszowskiej. Promotorek jest dr hab. inż. Zbigniew Hajduk, prof. PRz.

Struktura pracy

Praca składa się z pięciu rozdziałów. W rozdziale pierwszym: „Wprowadzenie" Doktorant definiuje cele, zakres oraz tezy pracy. Jako tezy pracy podano:

1) Sprzętowa realizacja sieci neuronowych typu LSTM w strukturach FPGA lub realizacja sprzętowo-programowa pozwala uzyskać znacząco większą szybkość działania w odniesieniu do realizacji całkowicie programowych, w tym z wykorzystaniem GPU.
2) Możliwa jest sprzętowa realizacja procesu uczenia sieci LSTM umożliwiająca istotne skrócenie czasu uczenia sieci.

Odnośnie tezy pierwszej, już we wstępie cytując referencję [18] Doktorant podaje, że implementacja w układach FPGA sieci LSTM do analizy obrazu daje 50 przyspieszenie w stosunku do GPU. Szkoda, że w dalszej części pracy Doktorant szerzej nie opisuje sposób implementacji sieci LSTM w [18] .

W rozdziale drugim: „Narzędzia i technologie wykorzystywane w pracy" Doktorant opisuje na samym początku technologię układów FPGA, ale również już bardzo stare i niewykorzystywane w pracy technologie PLA, PAL i CPLD. Podrozdział 2.3 (ponad 6 stron) opisuje języki opisu sprzętu, głównie język Verilog z podaniem operatorów (Tab. 2.1) a nawet ich priorytetów (Tab. 2.2). Podrozdziały 2.4 i późniejsze poświęcono układom Zynq, Versal oraz narzędziom projektowym.

W rozdziale trzecim: „Sieci LSTM" na samym początku opisano sieci neuronowe LSTM. W podrozdziale 3.2 podano założenia implementacyjne, między innymi użycie arytmetyki zmiennoprzecinkowej czy też założonej precyzji obliczeń. Podrozdział 3.3 jest poświęcony realizacji funkcją aktywacji: funkcji tangensa hiperbolicznego oraz sigmoidalnej. Jedną z przyjętych strategii implementacji było najpierw obliczenie eksponenty (z użyciem algorytmu CORDIC) a następnie zastosowanie wzorów (3.7) lub (3.8). Do obliczenia eksponenty (e^x) dodatkowo zastosowano arytmetykę stałoprzecinkową. Niestety nie użyto ograniczenia dziedziny funkcji do zakresu [0, 1) poprzez zastosowanie wzoru: $x \log_2(e) = x_2 = x_i + x_f$ (część całkowita x_i i część ułamkowa x_f), oraz $e^x = 2^{x2} = 2^{xi} e^{\ln 2\, xf}$. W konsekwencji obliczanie eksponenty ogranicza się do: $e^{\ln 2\, xf}$, co umożliwia dużo szybsze i łatwiejsze obliczenie funkcji eksponenty. W rozdziale 3.3 opisano użycie aproksymacji funkcji aktywacji z użyciem wielomianów Czebyszewa. Jest to poprawne podejście, ale lepszy rezultat można uzyskać używając funkcji minmax dostępnej w MATLABie do aproksymacji dowolnej funkcji w określonym przedziale i określonym stopniem wielomianu. Po drugie przy doborze aproksymacji funkcji być może należy

uwzględniać nie tylko błąd maksymalny ale również np. ciągłość funkcji. W podrozdziale 3.5 pokazano przykład praktyczny zastosowania sieci LSTM w procesie kucia na zimno. Na uwagę zasługuje tutaj sprawdzenie dokładności klasyfikacji dla konkretnego przypadku. Jest to unikalny rozdział pracy ponieważ rzadko spotyka się studium implementacji sprzętowej w konkretnym rozwiązaniu. W podrozdziale 3.6 opisano implementację sieci LSTM w przypadku ograniczonych zasobów sprzętowych występujących na platformie Zynq, gdzie w jednym układzie scalonym znajdują się zarówno struktura układu FPGA jak i procesor ARM.

Rozdział 4: „Uczenie sieci LSTM z wykorzystaniem układów FPGA" opisuje implementacje algorytmu uczenia sieci LSTM. Na uwagę zasługuje fakt, że uczenie sieci jest dużo bardziej skomplikowane od wnioskowania, w konsekwencji istnieje niewiele podobnych implementacji w układach FPGA. Doktorant w tym rozdziale prezentuje swoje oryginalne osiągnięcie. Niniejszy rozdział jest szczególnym osiągnięciem Doktoranta, szkoda, że skala skomplikowania architektury jest na tyle duża, że uniemożliwia prześledzenia i bardziej szczegółowego opisania osiągniętego rezultatu.

Rozdział 5 zawiera podsumowanie pracy oraz największych osiągnięć Doktoranta. Podsumowując, układ i struktura pracy jest poprawna i typowa dla pracy doktorskich.

Piśmiennictwo. Praca zawiera 127 odwołań do literatury. W pracy widoczna jest dogłębna analiza literatury. Niestety zakres pracy jest bardzo duży oraz w literaturze dostępne jest dużo publikacji na relatywnie niskim poziomie, co poskutkowało, że Doktorant w wielu miejscach porównywał się do miernej jakości rozwiązań.

Cele pracy. Cele pracy zostały określone na stronie 12. Doktorant wymienia trzy cele pracy, które zostały jasno i klarownie postawione. Według podanych przez Doktoranta wyników implementacji, rezultaty pracy zostały osiągnięte. Niestety podane wyniki budzą wątpliwości co zostało wskazane w uwagach krytycznych.

Zastosowane metody badawcze. Doktorant zastosował metody badawcze typowe dla prac związanych z implementacją sprzętową w układach FPGA. Prace praktyczne polegały głównie na kodowaniu w języku Verilog z wykorzystaniem gotowych modułów wykonujących operacje zmiennoprzecinkowe oraz stałoprzecinkowe. Dyplomant korzystał z dostępnych narzędzi oraz standardów (np. magistrali AXI Stream) dostarczonych przez firmę AMD (dawniej Xilinx), poprawnie używał projektowania systemów wbudowanych. Używał również języka programowania wysokiego poziomu Vivado HLS, które jak wiadomo zdecydowanie przyśpiesza czas projektowania niestety kosztem zwiększonych zasobów sprzętowych, co zostało wskazane w wynikach. Doktorant wykonał również implementacje na procesorach CPU oraz GPU, z użyciem języków Python i odpowiednich bibliotek oraz języka C/C++. Doktorant każde doświadczenie naukowe zakończył podaniem wyników oraz wnioskami.

Omówienie wyników. W pracy Doktorant wykonał bardzo wiele różnych implementacji w wielu różnych wariantach, co skutkowało podaniem bardzo wielu wyników, które zostały dogłębnie omówione. Wadą pracy jest wątpliwa jakość podanych wyników, brak krytycznego podejścia do nich, czy też przeoczenie lepszych rozwiązań. Zagadnienie to zostało rozszerzone w uwagach krytycznych. Oczywistym jest fakt, że w pracy naukowej (a przede wszystkim doktorskiej, na wstępnym etapie działalności naukowej) nie wszystkie osiągnięte wyniki są na najwyższym światowym poziomie i można je ulepszyć. Dlatego Doktorant powinien próbować obiektywniej (z większą samokrytyką) podchodzić do osiągniętych wyników oraz wskazać ich niedociągnięcia i ewentualne formy ulepszenia. Niestety w podsumowaniu pracy „możliwe dalsze kierunki rozwoju i tematyki pracy" zostały spisane tylko na sześciu liniach. Podsumowując, mniej krytyczny czytelnik na podstawie niniejszej pracy mógłby odnieść wrażenie, że implementacja sieci neuronowych w układach FPGA jest najlepszym rozwiązaniem, dużo lepszym niż w procesorach CPU i GPU. Niestety, z mojego doświadczenia, układy FPGA wykorzystywane do

implementacji sieci neuronowych stanowią raczej rozwiązanie stosowane tylko w szczególnych sytuacjach.

Praktyczne zastosowanie uzyskanych wyników badań. Użycie uczenia maszynowego jest dzisiaj wiodącym trendem zarówno w badaniach naukowych jak i w praktycznym zastosowaniu. Jednym z głównych ograniczeń uczenia maszynowego są ograniczone zasoby obliczeniowe. Dlatego akceleracja obliczeń w sieciach neuronowych jest bardzo gorącym tematem. Jest to widoczne chociażby w wycenie producenta procesorów graficznych: Nvidia, który jest najdroższą korporacją na świecie o kapitalizacji około pięciokrotnie większej niż PKB Polski. Praktyczna implementacja sieci neuronowych w układach FPGA, zarówno wnioskowania jak i trenowania stanowi bardzo ważny aspekt tej pracy. Warto podkreślić, że zakres pracy doktorskiej jest bardzo rozległy i w związku z tym pewne szczegóły pracy nie zostały dogłębnie zbadane. Niemniej *kluczowym elementem pracy jest fakt, że sieci neuronowe zostały praktycznie zaimplementowane zarówno w procesie wnioskowania jak i uczenia*, podano wyniki działania sieci dla praktycznego zastosowania: kłucia na zimno.

Uwagi krytyczne i pytania do Doktoranta

1) Operacje zmiennoprzecinkowe. W rozdziale 3.2 założono użycie tylko operacji zmiennoprzecinkowych: „ze względu na chęć podkreślenia wpływu zmiany platformy, na której wykonywane są obliczenia związane z siecią LSTM, przyjęto, że zarówno na FPGA jak i w programach przygotowanych w C/C++ wykorzystana zostanie arytmetyka zmiennoprzecinkowa pojedynczej precyzji zgodna z IEEE754". Powyższa motywacja jest niejasna lub zbyt skromnie zmotywowana. Warto podkreślić, że operacje zmiennoprzecinkowe w układach FPGA zajmują dużo więcej zasobów sprzętowych niż operacje stałoprzecinkowe. W przypadku implementacji sieci neuronowych nawet operacje stałoprzecinkowe są najczęściej dalej redukowane przez zmniejszanie precyzji obliczeń (szerokości bitowej) lub kwantyzację wag [18]. Dlatego często operacja mnożenia jest zastępowana operacją dodawania. Stosuje się nawet zerowanie mniej znaczących wag. Często po zmianie precyzji wag (zerowaniu wag) stosuje się dodatkowe douczanie sieci neuronowej. Pewnym uzasadnieniem użycia operacji zmiennoprzecinkowych jest trenowanie sieci, które niestety wymaga większej precyzji. Niemniej coraz częściej wykorzystuje się w takim wypadku operacje zmiennoprzecinkowe połówkowej precyzji (16-bitów) lub nawet mniejszej. Proszę o lepsze zmotywowanie przyjętego założenia oraz o teoretyczne porównanie zajmowanych zasobów sprzętowych układu FPGA przez układy mnożące i dodające zmiennoprzecinkowe, stałoprzecinkowe oraz stałoprzecinkowe o różnej precyzji.

2) Realizacja funkcji aktywacji. W podrozdziale 3.3.1 na około pięciu stronach Doktorant opisał studium literatury na temat implementacji funkcji aktywacji w układach FPGA, funkcje: tangensa hiperbolicznego oraz funkcję sigmoidalną. Studium wydaje się być wyczerpujące, niemniej ostateczne wnioski wyciągnięte z tego studium są błędne. W rezultacie Doktorant w rozdziale 3.3 opisuje własne badania nad różnymi implementacjami funkcji aktywacji. Co więcej zaimplementowane przez Doktoranta funkcje aktywacji są zarówno wolne jak i zajmują dużo zasobów sprzętowych układu FPGA. W rezultacie Doktorant wyciąga według mnie mylny wniosek (również na podstawie innych publikacji), że funkcja aktywacji jest jedną z najbardziej skomplikowanych i czasochłonnych części implementacji sieci neuronowych w układach FPGA. Doktorant na podstawie [91] wnioskuje, że bezpośrednia aproksymacja funkcji aktywacji może znacząco zredukować ilość wykorzystywanych zasobów oraz przyśpieszyć czas obliczeń, w przeciwieństwie do implementacji wymagających obliczania funkcji eksponenty. Niemniej w dalszej części pracy wniosek ten ignoruje i implementuje funkcję aktywacji z użyciem eksponenty, jako jednego z możliwych rozwiązań.

Niestety Doktorant cytuje, ale później ignoruje np. publikację [86], która opisuje implementacje z użyciem aproksymacji liniowej i liczb stałoprzecinkowych. Według mnie, aproksymacja liniowa lub kwadratowa umożliwia osiągnięcie dowolnie małego błędu wyjściowego kosztem wielkości stosowanej pamięci (liczby przedziałów). W pracy w Tab. 3.9 przedstawiono dyskusję na temat liczby przedziałów. Na stronie 80 wyciągnięto poprawne wnioski odnośnie liczby przedziałów i dokładności: czy bardziej opłacalne jest stosowanie wysokiego stopnia wielomianu (długiego czasu obliczeń i liczby użytych modułów arytmetycznych) ale małej liczby przedziałów (małej pamięci), czy też stosowanie prostej aproksymacji kwadratowej kosztem zwiększenia liczby przedziałów. W pracy Doktorant skupia się na implementacji używającej tylko 15 przedziałów, a w Tab. 3.9 ogranicza się tylko do 244 przedziałów. Warto podkreślić, że pamięci LUT w układach FPGA mają pojemność 64×1-bit, czyli bez dodatkowych zajmowanych zasobów można użyć 64 przedziałów. Używając pojedynczej pamięci BRAM o pojemności 18kb w konfiguracji 512×32 bity można uzyskać 512 przedziałów. Liczbę przedziałów można dodatkowo zwiększyć stosując więcej bloków pamięci BRAM. Jeśli dla liczby przedziałów rzędu 2048 i aproksymacji liniowej dokładność obliczeń jest za mała można próbować użyć aproksymacji kwadratowej. Używając arytmetyki stałoprzecinkowej i aproksymacji liniowej można nie tylko zmniejszyć liczbę zajmowanych zasobów sprzętowych ale również zmniejszyć latencje (opóźnienie potokowe) oraz zwiększyć częstotliwość zegara. Według moich szacunków, możliwe jest uzyskanie częstotliwości rzędu 200 MHz i latencji rzędu 2-3 taktów zegara dla takiej implementacji funkcji aktywacji. Przy zastosowaniu architektury potokowej wynik możemy otrzymywać co takt zegara. Proszę Doktoranta o potwierdzenie moich szacunków. Dlaczego w dalszej części pracy (rozdział 3.3.5) Doktorant ignoruje własne wyniki z Tab. 3.9. Podobnie w Tab. 3.14, publikacja [86] jest skomentowana jako: brak danych dotyczących wykorzystywanych zasobów i czasu obliczeń i dalej ignorowana. Na obronę wykonanych przez Doktoranta badań mogą posłużyć liczne cytowane publikacje, np. [9], które w większości popełniają podobne błędy i nieoptymalnie implementują funkcję aktywacji.

3) Architektura potokowa i równoległa. Na stronie 27-28 Doktorant opisuje zasadę działania architektury potokowej (ang. pipeline). Wymowny jest Rys. 2.10, który pokazuje zdecydowane przyśpieszenie obliczeń z użyciem architektury potokowej. Zalety stosowania architektury potokowej są znaczące: umożliwia ona zwielokrotnienie przepustowości kosztem niewielkich dodatkowych zasobów głównie rejestrów potokowych oraz niestety kosztem skomplikowania projektu. W dalszej części pracy (z wyłączeniem odwołań do literatury) nie wspomina się o tej architekturze. Na przykład w Tab. 3.7 podano wyniki dla różnej liczby cykli zegara na operację (czyli różnej liczbie etapów potokowych: latencji). Podobnie jest w Tab. 3.22 i komentarzu do latencji modułów arytmetycznych na początku strony 140. Sprawia to wrażenie, że Doktorant nie rozumie zasady działania architektury potokowej, która z definicji nie zmniejsza czasu od podania danej wejściowej do otrzymania danej wyjściowej (latencji). Jednak zdecydowanie zwiększa przepustowość, czyli liczbę przetworzonych danych w jednostce czasu przy założeniu zapełnienia potoku. W przypadku realizacji funkcji aktywacji (ale również mnożenia i dodawania w sieci neuronowych) używano automatu stanu do kontroli przepływu danych, czyli realizowano zadania sekwencyjnie bez zapełniania potoku. Nawet przy architekturze sekwencyjnej możliwe jest zastosowanie architektury potokowej i można wykonywać obliczenia dla 2-8 danych w potoku równocześnie – co skutkowałoby przyśpieszeniem obliczeń około 2-8 razy. Przykładem mogą być wydajne procesory CPU, które zawsze wykorzystują architekturę potokową i o ile to możliwe zapełniają wszystkie etapy potokowe. Innym (prostszym w projektowaniu) rozwiązaniem jest stosowanie w pełni równoległej architektury, dla której realizacja np. operacji aproksymacji wielomianowej wymaga tyle układów dodających i mnożących jaki jest stopień wielomianu. Ułatwia to projektowanie układu z użyciem architektury potokowej: dane wchodzą co takt zegara i wychodzą co takt zegara z odpowiednim opóźnieniem potokowym (latencją). Taka architektura według mnie jest lepsza ponieważ np. dla Tab. 3.7 i drugiego stopnia wielomianu umożliwia otrzymanie przepustowości jednej danej (wejściowej /

wyjściowej) na takt zegara zamiast jednej danej na 34 takty zegara. Wiąże się to z podwojeniem zajmowanych zasobów sprzętowych (dwa układy dodające i mnożące) ale 34 krotnym przyspieszeniem obliczeń. Pojedyncza funkcja aktywacji może być wtedy współdzielona przez wiele neuronów. Podobny sposób obliczeń mógłby być stosowany do sumowania iloczynów wag i wejść w ramach neuronów.

3) Porównanie czasów obliczeń z platformą CPU i GPU. Na stronie 113 (Tab. 3.20, Tab. 3.21, Tab. 3.23) przedstawiono czasy obliczeń dla platform CPU, CPU + GPU oraz procesora ARM. Wyniki obliczeń dla platformy FPGA pozornie wyglądają bardzo dobrze. Jednak opisom obliczeń na platformie CPU oraz CPU + GPU poświęcono bardzo mało miejsca w pracy. Nie wiadomo na ile zostały one zoptymalizowane. Warto zauważyć, że czas obliczeń (wielkość obliczeń) jest relatywnie krótki: 6,8 ms do 46 ms. W konsekwencji znacząca część czasu obliczeń mogła być poświęcona wywołaniu programu, odczytowi danych z pamięci dyskowej, dostarczenie danych do pamięci cache, itd. Znamienne jest porównanie czasu obliczeń: platforma CPU (47 ms), platforma Raspherry Pi (11 ms) lub Zynq PS (6.8 ms) – czym mniej wydajna platforma i niższy poziom skomplikowania systemu (operacyjnego) tym mniejszy czas obliczeń. Doktorant powinien program napisany w C/C++ skompilować na platformie CPU (x86) i wykonać obliczenia wielokrotnie tak, aby dane znajdowały się w pamięci cache. Warto zwrócić uwagę również na użycie jednostki wektorowej AVX. Optymalizacji może ulec również implementacja samej funkcji aktywacji tak aby błąd obliczeń był porównywalny jak na platformie FPGA (np. instrukcja asemblera VEXP2PS lub nawet aproksymacja linowa/kwadratowa). Warto również teoretycznie uwzględnić liczbę rdzeni CPU dostępnych we współczesnych procesorach CPU. Proszę również o wskazanie teoretycznej szczytowej mocy obliczeniowej (FLOPS) procesorów CPU / GPU oraz uzyskanej w realizacji sprzętowej na platformie FPGA. Jeśli procesor CPU nie wykorzystuje swojej pełnej mocy obliczeniowej proszę o wskazanie przyczyny. W jakich warunkach (np. wielkości sieci LSTM, dostępności do danych wejściowych) układy FPGA wykonują obliczenia szybciej / wolniej niż platforma zoptymalizowana CPU.

4) Optymalizacja struktury sieci neuronowej. Przykładem szeroko stosowanej optymalizacji sieci neuronowych implementowanych w układach FPGA jest użycie operacji stałoprzecinkowych, zmniejszenie precyzji wag, zastępowanie operacji mnożenia operacjami dodawania lub nawet obcinanie (zerowanie) wybranych wag. W takim wypadku proces uczenia (na CPU / GPU) przebiega dwuetapowo: standardowe trenowanie oraz powtórne douczania sieci po optymalizacji jej struktury pod kątem implementacji w układach FPGA. Czy podobnie jest z dokładnością obliczania funkcji aktywacji? Dokładność funkcji aktywacji ma mniejsze znaczenie, gdy w procesie (powtórnego) uczenia sieci i wnioskowania wykorzystywany jest taki sam model funkcji aktywacji?

Doktorant w swojej pracy założył użycie operacji zmiennoprzecinkowych, dzięki temu sieć neuronowa nauczona na platformie CPU / GPU może w prosty sposób być przeniesiona na platformę układów FPGA. Jest to zdecydowana zaleta zaproponowanej architektury pod warunkiem, że konieczne jest częste douczanie/zmiana parametrów sieci zgodnie z dotrenowaniem na platformie CPU. Doktorant o tym fakcie nie wspomina w pracy. Proszę o komentarz.

5) Wykorzystanie pamięci (BRAM i Distributed RAM) układów FPGA. W rozdziale 3.6 zaproponowano implementacje sieci LSTM na platformie Zynq, dla której dane wejściowe, wagi sieci oraz wyjścia każdego neuronu były za każdym razem przesyłane do części logicznej (PL) z części procesora CPU (PS). Transfer był realizowany za pomocą szybkiego modułu DMA, jednak stanowił on wąskie gardło systemu do tego stopnia, że użycie części logicznej (PL) nie dawało żadnej akceleracji. W układach FPGA występują pamięci blokowe BRAM (opisane na stronie 26 pracy). Dlaczego Doktorant z nich nie korzystał? Pamięci BRAM mogły służyć do pamiętania współczynników wag oraz wejść / wyjść w poszczególnych iteracjach sieci. W konsekwencji transmisja danych byłaby znacznie ograniczona – jak przy pełnej implementacji (dużego układu FPGA). Podobnie mogło być z użyciem automatu stanu do zarządzania modułami arytmetycznymi i podawaniem współczynników wag. Standardowo, jeśli w języku Verilogu nie użyjemy dedykowanej składni, program syntezujący użyje przerzutników zamiast pamięci rozproszonej

(w tablicach LUT) lub pamięci BRAM w przypadku większych pamięci. Wiąże się to ze zdecydowanym wzrostem zajmowanych zasobów sprzętowych

6) **Brak dogłębnej analizy teoretycznej zaimplementowanych układów.** Implementacja sprzętowa w układach programowalnych wymaga długiego czasu projektowania. Dlatego w wielu przypadkach sensowne jest podawanie analizy teoretycznej układu np. liczby operacji dodawania, mnożenia, funkcji aktywacji, liczby taktów zegara również na poszczególne operacje arytmetyczne, zajmowanych zasobów DSP, pamięci. Skraca to zdecydowanie czas projektowania ponieważ już na poziomie wstępnych założeń możliwe jest odrzucanie gorszych (bezsensownych) rozwiązań. Analiza teoretyczna pomaga również zrozumieć i sprawdzić sensowność otrzymanych wyników. W pracy skupiono się głównie na implementacji i podawaniu gotowych wyników dużej liczby alternatywnych rozwiązań, bez ich gruntownej teoretycznej analizy. Przykładem może być implementacja funkcji aktywacji, a szczególnie jej implementacja z użycie funkcji eksponenty (ze wzorów 3.7 i 3.8). Implementacja ta wymaga obliczania funkcji eksponenty i dzielenia, które są skomplikowane i wymagają wielu taktów zegara. Czy w takim razie jest sens dalszego ulepszania takiej implementacji i dalszego stosowania w strukturze sieci neuronowych skoro istnieją szybsze i prostsze algorytmy np. aproksymacji liniowej / kwadratowej. W przypadku implementacji sieci LSTM warto podać wyniki implementacji (jednego najlepszego rozwiązania) z podaniem ile zajmują zasobów / taktów zegara poszczególne podbloki.

Podobnie jest z implementacją algorytmu uczącego. Na stronie 158 podano, że zajmowane zasoby logiczne LUT wzrosły 5.27 razy w porównaniu z siecią wnioskującą, mimo tego, że zasoby obliczeniowe (DSP) pozostały bez zmian. Układy arytmetyczne były współdzielone, czyli obliczenia były wykonywane sekwencyjnie. Złożoność obliczeniowa (wnioskowanie i uczenie) została przełożona na dłuższy czas obliczeń. Doktorant próbuje na str. 158 tłumaczyć wynik zajmowanych zasobów, niemniej tłumaczenie to jest mało przekonujące. Czy układ nie można dodatkowo ulepszyć stosując pamięć, lepiej zoptymalizować użyty automat stanu. Zrozumiałym jest, że użyty algorytm jest bardzo skomplikowany i trudno dokładnie przeanalizować wyniki zajmowanych zasobów. Niemniej przy tak zaskakujących wynikach warto podać ile zasobów sprzętowych i taktów zegara zajmują elementy składowe układu.

Oryginalne rozwiązania problemu badawczego. W podsumowaniu pracy a szczególnie na stronie 182 i 183 podano syntetyczne podsumowanie. Dlatego wymienię tylko najważniejsze osiągnięcie Doktoranta. Za najważniejsze oryginalne osiągnięcie Doktoranta uważam implementację algorytmu uczenia w sieci LSTM. W porównaniu z wnioskowaniem uczenie sieci jest dużo bardziej skomplikowane ale niestety implementacja w układach FPGA nie daje tak znaczących efektów w szybkości obliczeń. Niemniej w niektórych sytuacjach: brak połączenia sieciowego, jego niepewność lub krytyczny czas reakcji, uczenie sieci bezpośrednio w systemie wbudowanym (systemie brzegowym) może być niezastąpione. Warto podkreślić, że sieci LSTM są często wykorzystywane do predykcji szeregów czasowych, dlatego douczanie sieci dla sygnałów niestacjonarnych lub też w warunkach anomalii może być kluczowe.

Dużą zaletą prezentowanej pracy jest sprawdzenie poprawności działania sieci LSTM dla konkretnego zadania klasyfikacji kucia na zimno pod kątem dokładności implementacji funkcji aktywacji. Z reguły w literaturze implementacja sieci neuronowej jest oderwana od konkretnego zastosowania. Warto pokreślić wniosek ze strony 118 stwierdzający, że forma dostarczania danych do platformy sprzętowej jest kluczowa. W konsekwencji użycie układów FPGA umożliwia łatwą integrację np. przetworników analogowo-cyfrowych, dzięki temu akwizycja danych oraz obliczenia mogą być prowadzone na tej samej platformie. Jest to bardzo ważne w przypadku przetwarzania brzegowego (ang. edge computing).

W pracy przeprowadzono bardzo wiele eksperymentów badawczych i podano wiele oryginalnych wniosków. W tym miejscu należy podkreślić, że projektowanie układów FPGA jest bardzo czasochłonne. Dlatego wielość zaproponowanych rozwiązań wymagała znaczącego nakładu pracy Doktoranta. Analizując tą pracę nie można jej porównywać z efektami osiągniętymi np. z użyciem języka Python, gdzie łatwość projektowania pozwala na szersze studium literatury oraz podawania wielu różnych rozwiązań.

Za największą wadę tej pracy uważam brak krytycznego podejścia do osiągniętych wyników, szerszego opisania dalszych możliwych ulepszeń lub też teoretycznego wpływu przyjętej metody na osiągnięte rezultaty.

Podsumowanie

Podsumowując recenzję stwierdzam, że Pan mgr inż. Grzegorz Rafał Dec w rozprawie doktorskiej pt. *Sprzętowa implementacja sieci LSTM* zrealizował cel rozprawy. Zaprezentowane rezultaty - choć niepozbawione niedociągnięć przedstawionych wcześniej - stanowią oryginalny wkład Doktoranta w dyscyplinę Informatyka Techniczna i Telekomunikacja. Doktorant wykazał się umiejętnością samodzielnej pracy badawczej, znajomością literatury światowej i wiedzą w zakresie projektowania układów FPGA oraz uczenia maszynowego. Recenzowana praca spełnia wymagania ustawy z dnia 20 lipca 2018 roku Prawo o szkolnictwie wyższym i nauce w dyscyplinie naukowej Informatyka Techniczna i Telekomunikacja. Wnoszę o jej dopuszczenie do dalszych etapów postępowania doktorskiego.

Jamro Ernest

.....

dr hab. inż. Ernest Jamro prof. AGH