بیستمین کنفرانس ملی دانشجویی مهندسی برق ایران
دانشگاه رازی کرمانشاه
دانشگاه رازی
انجمن علمی مهندسی برق دانشگاه رازی
20th IRANIAN STUDENT CONFERENCE ON ELECTRICAL ENGINEERING
بیستمین کنفرانس ملی دانشجویی مهندسی برق ایران
NSSOEE سازمان علمی دانشجویی مهندسی برق کشور تاسیس ۱۳۷۷
اتحادیه انجمن‌های علمی دانشجویی مهندسی برق
ISC Islamic World Science Citation Center

ابزارتولید و بهینه‌سازی شبکه‌های یادگیری عمیق برای پیاده‌سازی روی FPGA ها

نیما شیرین‌زاده[۱] ، مهدی امینیان[۲]

۱-دانشجوی کارشناسی ارشد، گروه مهندسی کامپیوتر، دانشکده فنی و مهندسی، دانشگاه گیلان، رشت

nima.shirinzadeh@gmail.com

۲-استادیار، گروه مهندسی کامپیوتر، دانشکده فنی و مهندسی، دانشگاه گیلان، رشت

mahdi.aminian@guilan.ac.ir

خلاصه

یکی از انواع مهم شبکه‌های عصبی، شبکه‌های عصبی بازگشتی هستند. به طور کلی از این شبکه‌ها می‌توان برای پیدا کردن الگوها در داده‌هایی که توالی در آن‌ها وجود دارد، استفاده کرد. از جلمه کاربردهای شبکه‌های عصبی بازگشتی می‌توان به تشخیص صوت، ترجمه‌گرها، تشخیص دست‌خط، تشخیص حالت‌های انسانی، کنترل ربات‌ها، تولید موسیقی و کارهای مختلف دیگر اشاره کرد. اما یکی از چالش‌های مهم برای این نوع الگوریتم‌ها پیاده‌سازی آن‌ها روی سخت‌افزار است. پیاده سازی این الگوریتم‌ها روی CPU و GPU و FPGA امکان پذیر است. پیاده‌سازی با استفاده از CPU و GPU به دلیل سهولت پیاده‌سازی و وجود راهکارهای نرم‌افزاری آسان، روش‌های پیاده‌سازی شناخته شده‌تر هستند. از طرفی، امروزه FPGAها به دلیل مصرف توان پایین و امکان طراحی بهینه و مناسب هر الگوریتم و امکان پیاده‌سازی موازی می‌تواند در بخش‌های مختلف جایگزین مناسبی برای CPU و GPU باشد. با توجه به ویژگی‌های ذکر شده FPGA می-تواند ابزار مناسبی برای پیاده‌سازی شبکه‌های عصبی نیز باشد. با توجه به ماهیت پیچیده‌ی الگوریتم‌های شبکه‌های عصبی، پیاده‌سازی این نوع الگوریتم‌ها روی FPGA کاری زمان‌بر است و از طرفی برای پیاده‌سازی بهینه و مناسب الگوریتم‌های شبکه‌های عصبی به تجربه کافی در زمینه پیاده‌سازی سخت‌افزاری الگوریتم‌ها و همچین تسلط کافی به ماژول‌های تشکیل دهنده یک شبکه‌ی عصبی نیاز است. اما برای رفع این مشکلات و امکان پیاده‌سازی سریع و بهینه شبکه‌های عصبی روی FPGA می‌توان با ترکیب راهکارهای سخت‌افزاری و نرم‌افزاری نظیر روش‌های بهینه‌سازی مختلف ماژول‌های تشکیل دهنده شبکه‌های عصبی این امکان را فراهم نمود تا شبکه‌ی عصبی پیاده شده در پلتفرم‌های نرم‌افزاری مانند Keras و Pytorch را به شبکه‌های سخت‌افزاری بهینه‌سازی شده و قابل پیاده‌سازی روی FPGAها تبدیل کرد. یکی از خروجی‌های سخت‌افزاری ممکن، خروجی در زبان‌های سطح بالا مانند C یا C++ برای پیاده‌سازی از طریق HLS[۱]ها و سنتز سطح بالا است. که به این ترتیب می‌توان فرایند زمانبر پیاده‌سازی و بهینه‌سازی سخت‌افزاری یک شبکه را در کوتاه‌ترین زمان ممکن طی نمود و همچنین این امکان فراهم می‌شود تا استفاده‌کنندگان حوزه‌های نرم‌افزاری هم بدون نیاز به دانش سخت‌افزاری بالا بتوانند به راحتی یک شبکه‌ی عصبی روی سخت‌افزار اجرا کنند.

---

[۱] High-Level Synthesis

کلمات کلیدی: شبکه‌های عصبی، سنتز سطح بالا، شتاب‌دهنده‌های سخت‌افزاری، بهینه‌سازی، Framework، LSTM[۲]، HLS

۱.مقدمه

در گذر زمان و با پیشرفت تکنولوژی‌های سخت‌افزاری، در زمینه‌هایی مانند افزایش قدرت پردازشی و افزایش فضای ذخیره‌سازی، باعث شد تا الگوریتم‌های هوش مصنوعی[۳]مورد توجه قرار بگیرند و امکان استفاده‌ی عمومی آن‌ها فراهم شود، تا با استفاده از این الگوریتم‌ها امکان حل مسائلی که برای انسان غیرممکن بود فراهم شود. از جمله الگوریتم‌های مهم هوش‌مصنوعی و یادگیری ماشین، الگوریتم‌های یادگیری عمیق[۴]و شبکه‌های عصبی[۵]هستند که امروزه نقش مهمی را در بین الگوریتم‌های هوش‌مصنوعی بازی می کنند. یکی از نوآوری‌های مهم شبکه‌های عصبی، شبکه‌های عصبی بازگشتی[۶] هستند. این نوع شبکه‌ها به گونه‌ای طراحی شده اند که از فرایند یادگیری نورون[۷]های مغز انسان تقلید می‌کنند. به همین علت می‌توانند روی داده‌های توالی‌پذیر مانند صوت یا متن پردازش انجام دهند و مانند یک انسان به یادگیری و پیدا کردن الگوهای موجود در داده‌ها و پیش‌بینی بپردازدند. از مهمترین کاربردهای شبکه‌های عصبی بازگشتی، پردازش زبان‌های طبیعی [1]، سیستم تشخیص گفتار [2]، سیستم تشخیص صوت [6] [3]، و سیستم تشخیص دست‌خط [4] است.

امروزه از شبکه‌های عصبی بازگشتی ساده به دلیل حافظه بسیار کوتاه آن که باعث از بین رفتن گرادیان می‌شود استفاده نمی‌شود.[5] دو نوع مهم شبکه‌های عصبی که امروزه به صورت گسترده مورد استفاده قرار می‌گیرند شبکه‌های LSTM و[۸]GRU هستند. این شبکه‌های عصبی مشکل شبکه‌های عصبی ساده‌تر را ندارند و ارتباط بین داده‌های متوالی را حفظ می‌کنند.

برای پیاده‌سازی این شبکه‌های عصبی از GPU ها استفاده می‌شود. GPU ها با توجه به امکان اجرای موازی، بستر خوبی برای اجرای الگوریتم‌های شبکه‌های عصبی هستند. اما برای اجرای این الگوریتم‌ها می‌توان از FPGA ها نیز استفاده کرد. FPGAها هم مانند GPUها امکان اجرای موازی الگوریتم‌ها را دارند. و با توجه به اینکه FPGAها مصرف توان کمتری دارند و این امکان را می‌دهند تا برای هر الگوریتم معماری مناسب و بهینه طراحی کنیم، آن‌ها را گزینه مناسب‌تری برای پیاده‌سازی شبکه‌های عصبی بازگشتی می‌کند. ولی به علت اینکه اجرای الگوریتم‌های پیچیده مانند الگوریتم‌های هوش‌مصنوعی و بهینه‌سازی آن‌ها کاری زمان‌بر و نیاز به تجربه کافی در زمینه پیاده‌سازی سخت‌افزاری الگوریتم‌ها دارد، عموما برای پیاده‌سازی الگوریتم‌های شبکه‌های عصبی از بسترهای نرم‌افزاری موجود و GPU استفاده می‌شود.

در این مقاله، قصد داریم تا با ترکیب راهکارهای سخت افزاری و نرم افزاری پیاده‌سازی شبکه‌های عصبی این امکان را فراهم کنیم تا بتوان شبکه‌ی عصبی تولید شده در سطح نرم‌افزار را به شبکه عصبی بهینه‌سازی شده و قابل پیاده‌سازی روی سخت‌افزار تبدیل کنیم. بدین ترتیب علاوه بر تولید سریع شبکه، شبکه بهینه و مناسب برای پیاده‌سازی سخت‌افزاری

---

[۲] Long short-term Memory
[۳] Artificial Intelligence
[۴] Deep Learning
[۵] Neural Network
[۶] Recurrent Neural Network
[۷] Neuron
[۸] Gated Recurrent Units

تولید می‌گردد. همچین از طریق این راهکار، این امکان فراهم می‌شود تا کاربران نرم‌افزاری هم بتوانند شبکه‌ی عصبی خود را با کمترین دانش سخت‌افزاری روی FPGA پیاده‌سازی کنند.

در ادامه این مقاله، بخش ۲ به بررسی کارهای پیشین در زمینه‌های مرتبط می‌پردازد. در بخش ۳، به مفاهیم اولیه‌ی شبکه‌های عصبی و سخت‌افزار و روش‌های به‌کار گرفته‌شده در پیاده‌سازی آن پرداخته می‌شود. در بخش ۴، چالش‌های پیش رو در این زمینه و ایده‌های حل مساله مطرح می‌شود. سپس در بخش ۵ روش پیشنهادی و چگونگی اجرای آن توضیح داده شده است؛ و در بخش ۶، ارزیابی و نتایج آن ارائه شده است. در نهایت، در بخش ۷، نتیجه‌گیری حاصل از اجرای روش پیشنهادی بیان می‌شود.

۲.کارهای پیشین

امروزه پیاده‌سازی GRU/LSTM روی FPGA یکی از موضوعات مناسب تحقیقاتی است. برای مثال در مقاله‌های [7][8] یک LSTM با استفاده از زبان C طراحی شده است که با استفاده از HLS به RTL[۹] تبدیل شده است. در مقاله [9] و در مقاله‌ی [10]، LSTM در سطح RTL طراحی شده است و در پلتفرم Xilinx zynq پیاده سازی شده است. استفاده از این روش زمان بیشتری صرف نموده است ولی خروجی آن می‌تواند بهتر از خروجی HLS باشد.

استفاده از HLS برای پیاده‌سازی شبکه‌های عصبی یک انتخاب مناسب است ولی استفاده از این ابزار به تنهایی نمی‌تواند فرایند پیاده‌سازی شبکه‌های عصبی روی FPGA را کامل کند. یک شبکه‌ی عصبی LSTM/GRU از المان‌های مختلفی مانند تابع‌های فعال‌ساز[۱۰] و ضرب ماتریسی تشکیل شده است. برای پیاده‌سازی بهینه شبکه‌های عصبی روی سخت افزار باید این المان‌ها به گونه‌ای پیاده‌سازی شوند تا مناسب پیاده سازی روی سخت افزار باشد.

در شبکه‌های عصبی تابع‌های فعال‌ساز Sigmoid، Tahn و Softmax تابع‌هایی هستند که بیشترین استفاده را در شبکه‌های عصبی بازگشتی دارند. اما این توابع به دلیل وجود محاسبات توانی[۱۱] و تقسیم اعداد شناور[۱۲] مقدار زیادی از منابع سخت‌افزاری را مصرف می‌کنند. برای پیاده سازی تابع‌های فعال‌ساز باید از پیاده‌سازی تقریبی این المان‌ها روی سخت‌افزار استفاده کرد تا با مصرف منابع کمتر به خروجی مورد انتظار رسید. در مقاله [7] [11] از روش PLAN برای پیاده‌سازی Sigmoid استفاده شده است. در مقاله [12] برای پیاده‌سازی Tanh رابطه‌ای تقریبی ارائه شده است که برای محاسبه نتیجه از Sigmoid استفاده می‌کند. همچین در مقاله [13] برای پیاده‌سازی یک لایه Softmax معماری جدیدی ارائه شده است که عملیات تقسیم اعداد شناور را از پیاده سازی حذف کرده است. در مقاله [14] برای بهینه‌سازی مصرف BRAM[۱۳] از Tiling استفاده شده تا به این ترتیب این امکان فراهم شود تا شبکه‌ی بزرگ‌تری را بتوان روی FPGA پیاده‌سازی کرد.

در مقالات ذکر شده سعی شد با استفاده از روش‌های بهینه‌سازی مختلف، بخش‌های مختلف را به صورت کارآتر برای سخت‌افزار طراحی کنند. در روش پیشنهادی این مقاله با ترکیب روش‌های بهینه‌سازی موجود در قالب یک پلتفرم و

---

[۹] Register-transfer Level
[۱۰] Activation Function
[۱۱] Exponentiation
[۱۲] Floating-point Divisions
[۱۳] Block RAMs

علاوه‌بر آن، با بهره‌گیری از راهکارهای نرم‌افزاری سعی در تولید سریع و بهینه‌ی یک شبکه سخت‌افزاری از خروجی شبکه پیاده‌سازی شده از Keras با استفاده از Python و HLS را داریم.

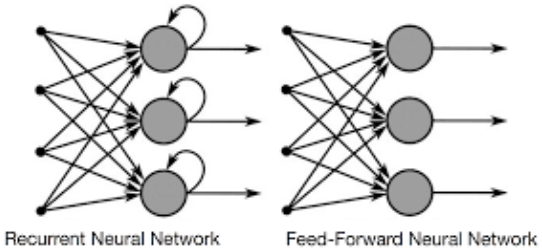
به طور خلاصه، این مقاله موارد زیر ار ارائه می‌دهد:

- جامع بودن پلت‌فرم پیشنهادی نسبت به موارد موجود از نظر دسترسی به تابع های فعال‌ساز مختلف و نوع شبکه‌های قابل پیاده‌سازی متفاوت
- ایجاد شبکه‌ی عصبی از خروجی نرم‌افزاری و صرفه‌جویی در زمان
- بهینه‌سازی تابع‌های فعال‌ساز و مساحت[۱۴] و تاخیر آن‌ها
- بهینه‌سازی مصرف BRAM و مساحت آن‌ها

۳.مفاهیم اولیه

۳. ۱. شبکه‌های عصبی بازگشتی

شبکه‌ی عصبی بازگشتی یک شبکه‌ی عصبی عمیق است. این نوع شبکه‌ها به گونه‌ای طراحی شده‌اند که می‌توانند با داده‌هایی که با هم ارتباط دارند کار کنند. بر خلاف شبکه‌های Feed-Forward [15] در این شبکه‌ها خروجی تولید شده در زمان قبلی را به عنوان ورودی در زمان حال استفاده می‌کنند. که بدین ترتیب می‌تواند توالی و ارتباط داده‌ها در یک مجموعه داده حفظ شود. در شکل ۱ ساختار شبکه‌ی RNN[۱۵] و Feed-Forward نشان داده شده است.



شکل ۱- ساختار شبکه ی عصبی RNN و Feed-Forward

رابطه‌های زیر نحوه‌ی عملکرد و محاسبات یک شبکه‌ی عصبی ساده را نشان می‌دهد. که در آن X بردار ورودی، H بردار خروجی در زمان قبلی، و Y بردار خروجی زمان فعلی است.

$$h_t = \sigma(w_{xh} + w_{hh} + b_h) \quad (1)$$
$$y_t = w_{hy} + b_y \quad (2)$$

w_{xh} ماتریس وزنی است که لایه‌ی خروجی را به لایه‌ی نهان مرتبط می‌کند، w_{hy} ماتریس وزنی است که لایه‌ی نهان را به لایه‌ی خروجی مرتبط می‌کند، و w_{hh} ماتریس وزن بازگشتی است که ورودی لایه‌ی نهان در زمان قبلی را به لایه‌ی

---

[۱۴] Area

[۱۵] Recurrent Neural Network

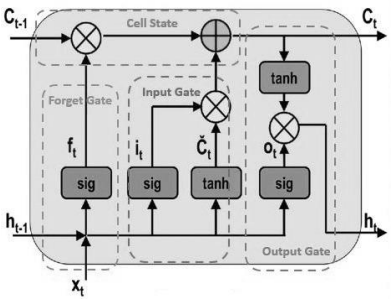
نهان در زمان فعلی مرتبط می‌کند. بردارهای b_h و b_y بردارهای بایاس هستند. همچنین، σ بیان‌گر یک تابع فعال‌ساز است که می‌تواند Sigmoid یا Tanh باشد.

۳. ۲. شبکه‌های عصبی بازگشتی

در شبکه‌های عصبی ساده تنها این امکان وجود دارد که از ورودی زمان قبلی یادگیری انجام دهد و این امکان وجود ندارد تا ورودی زمان‌های دورتر نیز در تصمیم‌گیری در داده‌های مرتبط در نظر گرفته شود. همچنین در شبکه‌های عصبی ساده مشکل Vanishing Gradient نیز وجود دارد. این حالت ممکن است در هنگام آموزش یک شبکه با چند لایه رخ دهد و باعث شود که تغییر در وزن ها هنگام آموزش به سختی انجام شود. برای حل این مشکلات، معماری جدیدی برای شبکه-های عصبی به نام LSTM [5] ارائه شد. در شکل ۲ معماری یک سلول LSTM نشان داده شده است. در زمان t خروجی‌های زیر از گیت‌های LSTM تولید می شود:

$$i_t = \sigma(w_{ix} x_t + w_{ir} y_{t-1} + w_{ic} c_{t-1} + b_i) \quad (3)$$
$$f_t = \sigma(w_{fx} x_t + w_{fr} y_{t-1} + w_{fc} c_{t-1} + b_f) \quad (4)$$
$$g_t = \sigma(w_{cx} x_t + w_{cr} h_{t-1} + b_c) \quad (5)$$
$$c_t = f_t \circ x_t + g_t \circ i_t \quad (6)$$
$$o_t = \sigma(w_{ox} x_t + w_{or} y_{t-1} + w_{oc} c_{t-1} + b_o) \quad (7)$$
$$m_t = o_i \circ h(c_t) \quad (8)$$
$$y_t = w_{ym} m_t \quad (9)$$

در روابط (3) تا (9)، W و U ماتریس وزن را نشان می‌دهند و b نشان‌دهنده‌ی بردارهای بایاس هستند. پارامترهای I، f، O و c به‌ترتیب گیت‌های ورودی، گیت فراموشی، گیت خروجی و سلول فعالیت حافظه‌ی طولانی مدت را نشان می‌دهد. عملگر (نقطه تو خالی) ضرب عنصر به عنصر را نشان می‌دهد و عملگر + جمع عنصر را نشان می‌دهد. علامت سیگما تابع Sigmoid را نشان می‌دهد و h تابع Tanh را نشان می‌دهد. هدف از تابع Sigmoid و Tanh نرمال‌سازی ورودی‌ها به منظور جلوگیری از تاثیر بیش از حد برخی ورودی‌ها نسبت به دیگر ورودی‌ها است. در میان تمام عملگرها ضرب ماتریس بردار بیشترین عملگر به کار رفته در حافظه و محاسبات است. ابعاد به کار رفته در x_t، y_t و c_t یکسان هستند.



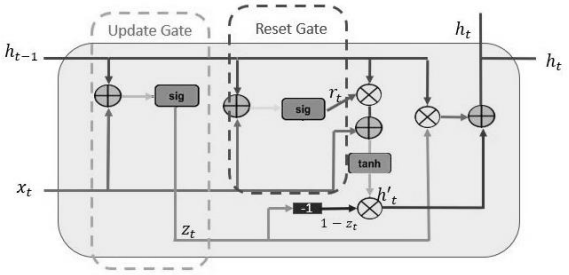
شکل ۲- معماری یک سلول LSTM

۳. ۳. شبکه عصبی GRU

نوع دیگری از معماری شبکه‌های عصبی، GRU ها هستند. معماری سلول GRU مشابه معماری سلول LSTM است با این تفاوت که تعداد پارامترهای یک سلول GRU کمتر از یک سلول LSTM است. عملکرد GRU در برخی از کارها مانند مدل‌سازی موسیقی و پردازش زبان‌های طبیعی مانند عملکرد LSTM است. از این رو در این نوع کارها استفاده از GRU کمک به کاهش حجم محاسبات و فضای مورد استفاده می‌کند.[16] معماری یک سلول GRU در شکل ۳ نشان داده شده است. در زمان t خروجی‌های زیر از گیت‌های GRU تولید می شود:

$$z_t = \sigma(w_z x_t + u_z h_{t-1} + b_z) \quad (10)$$
$$r_t = \sigma(w_r x_t + u_r h_{t-1} + b_r) \quad (11)$$
$$h'_t = \sigma(w_h x_t + u_h (r_t \circ h_{t-1}) + b_h) \quad (12)$$
$$h_t = \sigma((1-z_t) \circ h_{t-1} + z_t \circ h'_t) \quad (13)$$
$$y_t = h_t \quad (14)$$

همانطور که مشخص است یک سلول GRU به LSTM شباهت دارد اما از تعداد گیت‌های کمتری تشکیل شده است. z و r به ترتیب گیت به روزرسانی و گیت تنظیم مجدد را نشان می‌دهد.



شکل۳- معماری یک سلول GRU

۳. ۴. شتاب دهنده‌های سخت افزاری

یکی از بسترهای مناسب برای پیاده‌سازی شبکه‌های عصبی GPU است. GPU به دلیل داشتن تعداد هسته‌های زیاد می‌تواند قابلیت اجرای موازی داشته باشد به همین منظور یک گزینه‌ی خوب برای پیاده سازی شبکه‌های عصبی است. اما نوع دیگری از سخت‌افزار که مناسب پیاده‌سازی شبکه‌های عصبی است FPGAها هستند. این نوع سخت‌افزار هم به دلیل امکان اجرای موازی گزینه‌ی مناسبی برای پیاده‌سازی شبکه های عصبی است.

اما FPGA تفاوت‌هایی با GPU دارد که آن را به گزینه‌ی مناسب‌تری برای پیاده‌سازی شبکه‌های عصبی تبدیل می‌کند. FPGAها با فرکانس پایین‌تری کار می‌کنند، از این رو مصرف توان کمتر را با عملکرد یکسان بخاطر قابلیت

موازی‌سازی بالاتر دارند. همچنین به دلیل وجود بلوک‌های داخلی حافظه (BRAM) می‌توان به بیشترین حد موازی سازی در FPGAها رسید. نکته‌ی مهم دیگر انعطاف‌پذیر بودن FPGAها است که باعث می‌شود بتوان برای هر الگوریتمی، مداری بهینه طراحی کرد. به بیان دیگر امکان تغییر سریع معماری، یا حتی تغییر بخشی از آن در حین کار باقی بخش‌ها، با توجه به کاربرد وجود دارد تا الگوریتم‌ها به کارایی بالاتری برسند. [17] اما یکی از معایب پیاده‌سازی یک شبکه‌ی عصبی روی FPGA زمان‌بر بودن پیاده‌سازی الگوریتم است. همچنین پیاده‌سازی بهینه الگوریتم نیاز به تجربه کافی در پیاده‌سازی سخت‌افزاری الگوریتم ها مورد نیاز دارد و به دلیل اینکه این کار به صورت دستی انجام می‌شود باعث افزایش زمان پیاده‌سازی و تست یک شبکه روی FPGA می‌شود.

۳. ۵. روش‌های پیاده‌سازی سخت‌افزاری

یکی از راه‌های پیاده‌سازی الگوریتم‌ها روی سخت‌افزار استفاده از HDLها[۱۶](زبان‌های توصیف سخت‌افزار) است. اما برای پیاده‌سازی با این روش بسیاری از جزئیات پیاده‌سازی نیز باید توسط طراح پیاده‌سازی شود که این زمان به زمان پیاده‌سازی الگوریتم اصلی و بهینه‌سازی اضافه می‌شود. روش دیگر پیاده‌سازی استفاده از HLSها (زبان‌های سنتز سطح بالای مدارهای دیجیتال) است که از زبان‌های سطح بالا مانند C یا C++ برای پیاده سازی استفاده می‌کنند.

مزیت پیاده‌سازی با HLS کاهش زمان پیاده‌سازی و تست است. با استفاده از HLS حدود ۲ برابر سریع‌تر می‌توان الگوریتم‌ها را پیاده‌سازی کرد. اما مشکل استفاده از HLS این است که کارایی[۱۷]آن نسبت به کارایی خروجی HDL کمتر است، اما با این حال می تواند تا حد قابل قبولی به کارایی روش دستی HDL در زمان بسیار کوتاه‌تر نزدیک شود.

بنابراین HLS در بسیاری از موارد می‌تواند گزینه مناسبی برای پیاده‌سازی بسیاری از الگوریتم‌ها روی FPGA باشد. در این مقاله نیز برای پیاده‌سازی شبکه‌های عصبی از HLS استفاد شده‌است. تا علاوه بر اینکه پیاده‌سازی و تست الگوریتم سریع‌تر انجام شود این امکان برای کابران نرم‌افزاری نیز فراهم شود تا شبکه‌ی خود را روی سخت‌افزار پیاده‌سازی و تست کنند. [18]

۴.طرح مسئله

GPU و FPGA به دلیل دارا بودن قابلیت اجرای موازی از جمله بسترهای مناسب برای اجرای الگوریتم‌های شبکه‌های عصبی هستند. اما FPGA با توجه به انعطاف‌پذیری بالا و مصرف توان کمتر گزینه بهتری نسبت به GPUهاست. اما برای پیاده‌سازی شبکه‌ی عصبی روی GPU ها رابط‌های نرم‌افزاری متنوعی وجود دارد که این امکان را به کاربر می‌دهد تا در سریع‌ترین زمان و به آسانی بتواند یک شبکه را پیاده‌سازی کند و همچنین از قابلیت اجرای موازی آن استفاده کند. به همین دلایل GPUها گزینه‌ی پر استفاده‌تری هستند.

با وجود اینکه FPGAها یکی از مناسب‌ترین ابزارها برای پیاده‌سازی شبکه‌ی عصبی هستند، اما پیاده‌سازی شبکه-های عصبی روی FPGAها با چالش‌هایی همراه است و به راحتی پیاده‌سازی روی GPU یا CPUها نیست. یکی از

---

[۱۶] Hardware Description Language
[۱۷] Performance

مشکلات پیاده‌سازی شبکه‌های عصبی روی FPGAها نبود پلتفرم‌هایی مانند Keras یا Pytorch است تا بتوان سریعا یک شبکه برای سخت افزار تولید کرد.

برای این منظور باید جزییات شبکه‌های عصبی را برای رسیدن به یک پلتفرم پیاده‌سازی سخت‌افزاری کاملا بررسی نمود. یک شبکه‌ی عصبی از ماژول‌ها و بخش‌های مختلفی تشکیل شده است، و به دلیل این‌که ماژول‌های شبکه های عصبی منابع زیادی برای انجام محاسبات خود مصرف می‌کنند، امکان پیاده‌سازی یک شبکه‌ی عصبی بزرگ روی FPGA بصورت همزمان کار غیرممکنی است. بنابراین تنها تسلط روی جزییات شبکه برای پیاده‌سازی کافی نیست و ماژول‌ها باید بهینه‌سازی نیز شوند. برای بهینه کردن هر ماژول باید از روش‌های متفاوت و مختلفی با توجه به نوع ماژول استفاده کرد. بنابراین، یکی دیگر از مشکلات، علاوه بر زمان مورد نیاز برای پیاده‌سازی، پیچیدگی و زمان مورد نیاز بالا برای بهینه‌سازی یک شبکه‌ی عصبی برای اجرا روی FPGA است.

برای بهینه‌سازی شبکه های عصبی دو چالش مهم وجود دارد. یکی از این چالش‌ها، وجود عملیات‌هایی مانند توان‌رسانی و تقسیم اعداد شناور در توابع فعال‌سازی پر استفاده است که منابع محاسباتی زیاد از جملهٔ DSP[۱۸]، LUT[۱۹] و FF[۲۰]ها را مصرف می‌کند. چالش مهم دیگر، مصرف بالای حافظه BRAM در شبکه‌های عصبی است. یک شبکه‌ی عصبی از پارامترهای فراوانی در قالب ماتریس تشکیل شده است که ذخیره‌سازی آن‌ها با استفاده از بلوک‌های BRAM انجام می‌شود. مقدار داده‌های ورودی یک شبکه‌ی عصبی بسیار بیشتر از تعداد بلوک‌های BRAM موجود روی FPGA است، بنابراین پیاده‌سازی اکثر شبکه‌ها روی FPGA امکان پذیر نیست.

همچنین، برای پیاده سازی یک شبکه روی FPGA نیاز به یک رابط نرم‌افزاری جامع وجود دارد تا این امکان را فراهم کند که بتوان در سریع‌ترین زمان به یک خروجی قابل پیاده‌سازی رسید که عملکرد بهینه‌ای داشته‌باشد. در نتیجه باید راهکاری ارائه کرد که پیاده‌سازی شبکه‌ی عصبی روی FPGA را آسان‌تر کند، تا استفاده از آن برای کسانی که تجربه کافی در زمینه پیاده‌سازی سخت‌افزاری الگوریتم‌ها را ندارند نیز امکان‌پذیر باشد. این مقاله قصد دارد با ارائه یک واسط سخت‌افزاری نرم‌افزاری، و ایجاد امکانات پیاده‌سازی سخت‌افزاری ماژول‌های مهم شبکه‌های عصبی، قابلیت تطبیق و تبدیل شبکه‌های مختلف را برای پیاده‌سازی روی FPGA ایجاد کند. این رابط، شبکه‌های توصیف شده در پلتفرم‌های نرم‌افزاری مانند KERAS را به کدهای سخت‌افزاری تبدیل می‌کند، علاوه بر بهینه‌سازی کد، امکان زمان‌بندی اجرای لایه‌های مختلف روی بستر سخت‌افزاری را فراهم می‌کند.

۵. روش پیشنهادی

در روش پیشنهادی هدف فراهم آورن امکان پیاده سازی بهینه و سریع شبکه های عصبی شامل LSTM، GRU و DENSE با استفاده از شبکه‌ی آموزش دیده شده در Keras است. برای این منظور باید بخش‌های مهم این شبکه‌ها به صورتی پیاده‌سازی شوند که مناسب پیاده سازی روی سخت‌افزار باشند. در شبکه‌های عصبی توابع فعال‌ساز و ضرب‌های ماتریسی بخش هایی هستند که بیشترین منابع را مصرف می‌کنند. بنابراین در ابتدا راهکارهایی ارائه می‌دهیم تا مصرف منابع برای توابع فعال‌ساز مهم بهینه شود. سپس مصرف منابع فرآیند کلی یک شبکه‌ی عصبی و ضرب ماتریسی را بهینه می‌کنیم و در آخر راهکاری ارائه می‌دهیم تا شبکه‌ی تولید شده در ابزار نرم‌افزاری Kears را با استفاده از واسط

---

[۱۸] Digital Signal Processing
[۱۹] LookUp Table
[۲۰] Flip-flop

سخت‌افزاری نرم‌افزاری و ماژول های بهینه سازی شده‌ی شبکه‌های عصبی برای سخت‌افزار، به شبکه‌ی بهینه‌ی قابل پیاده سازی روی سخت‌افزار تبدیل کنیم.

۵. ۱. پیاده‌سازی تابع‌های فعال‌ساز

یکی از ماژول‌های مهم شبکه‌های عصبی تابع‌های فعال‌ساز غیر خطی هستند. در اکثر این نوع ماژول‌ها عملیات توان رسانی و تقسیم اعداد شناور وجود دارد. پیاده سازی این دسته از عملیات روی سخت‌افزار منابع زیادی مصرف می‌کند و همچنین باعث افزایش تاخیر می‌شود. بنابراین، برای پیاده‌سازی این ماژول‌ها باید از روش‌های تقریبی استفاده کرد. در پیاده‌سازی با روش‌های تقریبی علاوه بر بهبود تاخیر، به اندازه‌ی قابل ملاحظه‌ای در منابع مصرفی صرفه‌جویی می‌شود. برای این پلتفرم توابع Sigmoid [11] و Tanh [12] در قالب یک ماژول تقریبی پیاده‌سازی شده است. همچنین Softmax به صورت یک لایه Softmax تقریبی ارائه شده است. از Relu به دلیل اینکه ماهیتا مصرف منابع کمی دارد بدون تغییر در پیاده سازی استفاده شده است.

۵. ۱. ۱. پیاده‌سازی Sigmoid

یکی از انواع پر کاربرد توابع فعال‌ساز ، تابع فعال‌ساز Sigmoid است. رابطه(15)، تابع Sigmoid را نشان می‌دهد که شامل توان‌رسانی و تقسیم اعداد شناور است. برای پیاده‌سازی این ماژول از روش PLAN[11] استفاده شده است. در این روش عملیات‌های پرهزینه مانند توان‌رسانی و تقسیم اعداد اعشاری حذف شده است و این تابع با استفاده از Mux[۲۱]ها و Shift Registerها پیاده‌سازی شده است.

در شکل ۴ پیاده سازی Sigmoid با روش PLAN نشان داده شده است. در این روش در بخش مثبت ۷ نقطه‌ی کلی روی نمودار انتخاب می‌شود. و بین هر دو نقطه یک معادله خط تشکیل می‌شود و در نهایت ۴ پاره خط بر اساس نقاط روی نمودار اصلی بدست می‌آیند که به صورت تقریبی می‌تواند به ازای هر ورودی، خروجی نزدیک به نمودار اصلی را تولید کند. نقاط به گونه‌ای انتخاب شده‌اند که معادلات خط بدست آمده تنها با استفاده از یک عمل شیفت و یک عمل جمع می‌تواند خروجی تقریبی تابع Sigmoid را تولید کند. تفاوت مصرف منابع و تاخیر با استفاده از روش PLAN نسبت به پیاده‌سازی اصلی در جدول ۲ نشان داده شده است.

همانطور که در شکل ۵ مشخص شده نمودار Sigmoid یک نمودار متقارن است و خط تقارن در نقطه‌ی x=0 قرار دارد. بنابراین کافی است محاسبات فقط برای بخش مثبت انجام شود. برای بدست آوردن مقدار خروجی برای ورودی‌های منفی کافی است از خروجی محاسبه شده با استفاده از |x|، یک واحد کم کنیم.

$$sigmoid\ (x) = \frac{1}{1+e^{-x}} \tag{15}$$

---

[۲۱] Multiplexer

```
const Output sigmoid(Input input)
{
    Input y = 1;
    Input twoscomp =  -input;
    bool signbit = input[0];
    if (signbit){
        input = twoscomp;
    }
    if((Input) 2.375 <= input && input < (Input) 5){
        y = (input >> 5) + Input(0.84375);
    }else if((Input) 1 <= input && input < (Input) 2.375){
        y = (input >> 3) + Input(0.625);

    }else if( (Input) 0 <= input && input < (Input) 1){
        y = (input >> 2) + Input(0.5);
    }

    if(signbit) {
        y = Input(1) - y;
    }

    return Output(y);
}
```

شکل ۴- پیاده سازی sigmoid با استفاده ازروش PLAN در HLS

۵. ۱. ۲. پیاده‌سازی Tanh

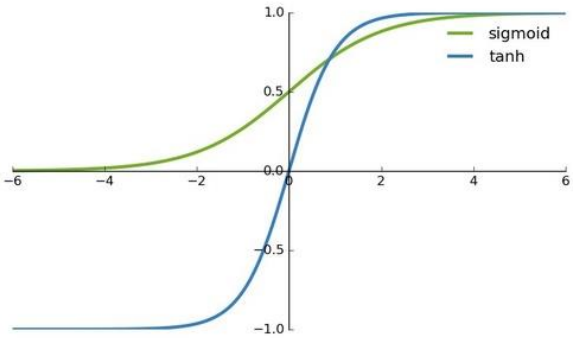
یکی دیگر از توابع فعال‌ساز مهم در شبکه‌های عصبی بازگشتی، تابع فعال‌ساز Tanh است. همانطور که در رابطه(16) مشخص است این تابع از ۴ عملیات توان‌رسانی اعداد شناور تشکیل شده است. بنابراین پیاده‌سازی این تابع نسبت به تابع Sigmoid در رابطه(15) از تاخیر بیشتری برخوردار است و مصرف منابع به مراتب بالاتری در حدود ۳ برابر بشتر از Sigmoid را داراست.

$$tanh\ (x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{16}$$

همانطور که در شکل ۵ نشان داده شده است نمودار دو تابع Sigmoid و Tanh مشابه یک دیگر است. [12] بنابراین با توجه به قوانین انتقال نمودارها می‌توان نقطه بدست آمده روی نمودار Sigmoid را به نقطه‌ای روی نمودار Tanh انتقال داد. با توجه به رابطه (17) می توان با استفاده از تابع Sigmoid پیاده‌سازی شده با روش PLAN و عملیات Shift، تابع فعال‌ساز Tanh را به صورت تقریبی پیاده‌سازی کرد.

در رابطه (17) زمانی که ورودی x دوبرابر می‌شود نمودار Sigmoid جمع‌تر شده و در واقع شیب نمودار بیشتر شده و به نمودار Tanh نزدیک‌تر می‌شود. برای انتقال نمودار در جهت محور y به سمت پایین یک واحد از تابع Sigmoid کم شده است و برای این که بازه تابع در جهت محور y دو برابر شود نیاز است تا خروجی تابع Sigmoid دو برابر شود. به این ترتیب خروجی تقریبی تابع Tanh با استفاده اعمال شیفت و تفریق روی تابع Sigmoid قابل محاسبه است. همچنین در جدول ۳ تفاوت در مقدار منابع مصرفی و تاخیر تابع تقریبی پیاده سازی شده نسبت به پیاده سازی اولیه نشان داده شده است.

$$tanh(x) = 2\sigma(2x) - 1 \tag{17}$$



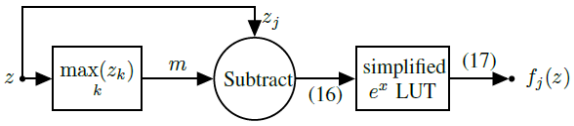
شکل ۵- مقایسه نمودار Sigmoid و Tanh

۵. ۱. ۳. پیاده سازی Softmax

برای این ابزار لایه Softmax در قالب یک لایه تقریبی [13] پیاده‌سازی شده است که برای عمل طبقه‌بندی[۲۲] در انتهای شبکه مورد استفاده قرار می‌گیرد.

$$softmax\ (x_i) = \frac{e^x}{\sum e^x} \tag{18}$$

با در نظر گرفتن رابطه (18) برای بهبود منابع مصرفی این تابع از معماری ارائه شده در شکل ۶ استفاده شده است در این معماری عملیات تقسیم شناور حذف شده و تنها به یک عملیات توان رسانی اعداد شناور نیاز است. برای محاسبه Softmax در این روش ابتدا بیشترین مقدار در بین اعداد پیدا می‌شود. سپس، بعد از تفریق کردن بیشترین مقدار ورودی از اعداد ورودی، این اعداد در عملیات توان رسانی عدد e شرکت داده می‌شوند. باید در نظر داشت که معماری ارائه شده در صورتی که در لایه‌های میانی استفاده شود احتمال اینکه خطای شبکه را بالا ببرد وجود دارد. اما در لایه آخر، با توجه به اینکه مقدار خطا ناچیز است این معماری می‌تواند یکی از مناسب‌ترین پیاده‌سازی‌ها برای تابع Softmax باشد.

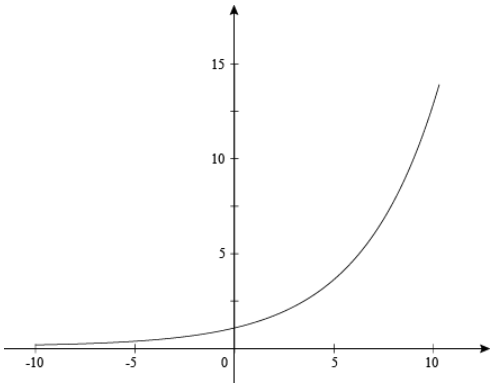


شکل ۶- معماری پیشنهادی یک لایه softmax [13]

---

[۲۲] Classification

وظیفه اصلی لایه Softmax دادن امتیاز بیشتر به ورودی بزرگ‌تر است و معمولا در لایه‌ی آخر برای تعیین نتیجه نهایی استفاده می‌شود. در معماری ارائه شده ابتدا در بین ورودی‌هایی که به لایه Softmax وارد می‌شوند بزرگترین ورودی انتخاب می‌شود. سپس ورودی بزرگ از تمام ورودی‌ها کم می‌شود. اعداد به دست آمده بعد از عمل تفریق در بازه بین صفر تا منفی بی‌نهایت قرار می‌گیرند. با اعمال یک تابع exp روی این ورودی‌ها می‌توان خروجی نهایی را به دست آورد. با توجه به شکل ۷ ورودی‌ها بعد از عمل تفریق در سمت چپ محور y قرار می‌گیرند. بنابراین خروجی‌ها در بازه ۰ تا ۱ قرار دارند و بزرگترین مقدار ممکن عدد ۱ به ازای ورودی صفر است. در این معماری این حالت زمانی اتفاق می‌افتد که بزرگترین ورودی از خودش کم می‌شود. به این ترتیب به ورودی‌ها از بزرگ به کوچک امتیاز اختصاص می‌یابد و ورودی بزرگ‌تر تعیین می‌شود.



شکل ۷- نمودار یک تابع exp

۵. ۲. روش Tiling

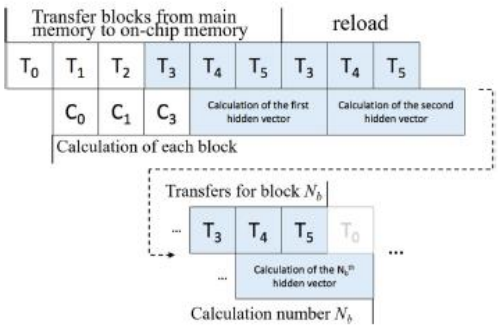
یکی از منابع مهم برای پیاده‌سازی شبکه‌های عصبی روی سخت‌افزار بلوک‌های BRAM هستند. از این منابع نسبت به LUTها، FFها تعداد کمتری وجود دارد و همین طور فضای ذخیره‌سازی این حافظه نیز نسبت به حافظه‌های خارجی کمتر است. همان طور که در بخش ۳ دیدیم یک لایه LSTM یا GRU از تعداد زیاد ماتریس وزن، بایاس و ورودی تشکیل شده است. بنابراین اگر تعداد نورون‌های یک لایه یا تعداد ویژگی‌های ماتریس ورودی زیاد شود اندازه ماتریس وزن و بایاس نیز تغییر می‌کند. که در این حالت به دلیل تعداد کم و فضای ذخیره‌سازی محدود BRAMها امکان پیاده سازی شبکه‌های بزرگ روی FPGA وجود ندارد. بنابراین یکی از چالش های مهم، بهینه‌سازی مصرف BRAM در پیاده‌سازی سخت‌افزاری شبکه است.

یکی از روش‌های موجود و پرکاربرد برای بهبود مصرف BRAM، استفاده از روش Tiling [14] در کنار روش Streaming ورودی‌ها از حافظه خارجی و Pipelining عملیات خواندن از حافظه خارجی و عملیات محاسباتی است. با ترکیب این روش‌ها می‌توان وزن‌ها و بایاس‌ها را روی حافظه خارجی DDR ذخیره‌سازی کرد و برای انتقال سریع آن‌ها به مدار از AXIS[۲۳] استفاده کرد. اما به دلیل اینکه خواندن از حافظه خارجی به تاخیر عملیات محاسباتی اضافه می‌کند، همان‌طور که در شکل ۸ نشان داده شده است، بخشی که شامل خواندن ورودی از حافظه خارجی و سپس اجرای عملیات

[۲۳] AXI4-Stream interface

اصلی است به صورت Pipeline پیاده‌سازی می‌شود. در این روش، در حین اجرای عملیات اصلی با ورودی‌های خوانده شده در t، ورودی‌ها برای زمان t+1 نیز خوانده می‌شود تا تاخیر خواندن و زمان اجرای عملیات همپوشانی پیدا کرده و از ایجاد تاخیر بخاطر خواندن از حافظه خارجی جلوگیری شود.

با استفاده از این روش تنها بخشی از ورودی‌ها که برای محاسبات است خوانده می‌شود. این ورودی از یک حافظه خارجی انتقال می‌یابد. همچنین این امکان وجود دارد تا تعداد ورودی‌های لازم برای هر خواندن نیز تعیین شود تا به این ترتیب علاوه به صرفه جویی در مصرف BRAM، تاخیر مدار نیز مطابق نیاز تنظیم شود.



شکل ۸- نحوه عملکرد روش Tiling

۵. ۳. واسط سخت‌افزاری نرم‌افزاری

در بخش‌های قبل توابع فعال‌ساز پرکاربرد شبکه‌های عصبی با استفاده از روش‌های تقریبی به گونه‌ای طراحی شد که با مصرف کمترین منابع، بهبود در تاخیر و دارا بودن دقت کافی، مناسب برای پیاده‌سازی روی سخت‌افزار و قابل استفاده برای شبکه‌های عصبی باشد. همچنین با استفاده از روش Tiling، Streaming و Pipelining فرایند کلی محاسبات در شبکه‌های عصبی به‌گونه‌ای طراحی شد که باعث کاهش مصرف BRAM در پیاده‌سازی سخت‌افزاری شبکه‌های عصبی گردد. بهینه‌سازی توابع فعال‌ساز و بهینه‌سازی مصرف BRAM این امکان را فراهم کرد تا شبکه‌های بزرگ نیز با تاخیر مناسب و مساحت کمتر روی FPGA قابل پیاده‌سازی باشند.

با وجود ماژول‌های قابل پیاده‌سازی شبکه‌های عصبی روی FPGA می‌توان با استفاده از یک واسط سخت‌افزاری و نرم‌افزاری، شبکه‌های تولید شده در پلتفرم‌های نرم‌افزاری Keras یا Pytorch را به شبکه های قابل پیاده‌سازی برای FPGA تبدیل کرد. واسط نرم افزاری ارائه شده با استفاده از زبان Python پیاده‌سازی شده است و این امکان را دارد تا لایه‌های LSTM، GRU و DENSE را از شبکه‌ی نرم‌افزاری آموزش داده شده تولید کند، که برای این کار از ماژول‌های بهینه شده مانند Sigmoid و باقی ماژول‌ها و روش‌های توضیح داده شده استفاده می‌کند. خروجی تولید شده کد C++ هر لایه برای HLS است که این امکان را می‌دهد تا هر لایه به صورت IP[۲۴] جداگانه با استفاده از viviado hls پیاده‌سازی شود و ورودی و خروجی هر لایه قابل استفاده و ذخیره سازی باشد. با استفاده از فایل های C++ تولید شده

---

[۲۴] Intellectual Property

در vivado hls می‌توان IP های مورد نیاز هر لایه را بدست آورد، سپس با استفاده از ابزار vivado ارتباط بین IPها و پردازنده ایجاد می‌شود.

در ابتدا این واسط نرم افزاری فایل خروجی پلتفرم نرم‌افزاری را بررسی کرده، و به ازای هر لایه، تعداد ورودی، تعداد نورون‌ها و نوع تابع‌های فعال‌ساز را تعیین می‌کند. سپس با استفاده از اطلاعات بدست آمده اقدام به تولید کد HLS هر لایه با توجه تعداد ورودی و نورون و نوع تابع فعال‌ساز می‌کند.

این واسط به ازای هر IP، ورودی‌های وزن و بایاسی که باید بر اساس روش Tiling به سیستم وارد شود را نیز تولید می‌کند که می‌توان از این فایل‌ها برای تست شبکه و در زمان پیاده‌سازی به عنوان ورودی استفاده کرد. بدین ترتیب در سریع‌ترین زمان ممکن شبکه قابل پیاده‌سازی روی FPGA تولید شده که قابلیت تست و تغییرات سریع را نیز به کاربر می‌دهد.

نرم‌افزار فعلی تولید شبکه‌های عصبی شامل لایه‌های شبکه‌های LSTM، GRU و DENSE است. در تولید IP، هر لایه نرم‌افزار از توابع فعال‌ساز مورد نیاز استفاده می‌کند و بر اساس اندازه‌های هر لایه، اندازه‌های TILING و ضرب ماتریسی شبکه را تنظیم می‌کند. همچنین این امکان وجود دارد که ۴ نوع لایه‌ی DENSE با استفاده از ۴ تابع فعال ساز ارائه شده تولید کند که بنابراین در واسط سخت‌افزاری نرم‌افزاری فعلی امکان تولید ۶ نوع لایه مختلف بهینه شده وجود دارد.

۶.ارزیابی روش پیشنهادی

برای بررسی نتایج این مقاله از برد pynq-z1 به عنوان برد مبنا استفاده شده و برای ارائه نتایج از آمار و ارزیابی‌های خروجی نرم‌افزار HLS استفاده شده است. در نتایج بدست آمده مشخص شد که بدون روش‌های بهینه‌سازی اعمال شده، فقط پیاده‌سازی لایه با تعداد ۲۰۰ ورودی و ۵۱۲ نورون روی برد pynq-z1 امکان پذیر است و با این اندازه شبکه، حدود ۲۶۴ عدد از ۲۸۰ عدد BRAM موجود مصرف می‌شود. اما استفاده از روش Tiling در فرایند کلی محاسباتی شبکه‌ی عصبی این امکان را فراهم کرد تا بتوان لایه ای با ۳۰۰ برابر ورودی و ۱۲۰ برابر نورون بیشتر نیز قابل پیاده‌سازی باشد. همچنین پیاده‌سازی تابع‌های فعال‌ساز با روش‌های تقریبی کمک کرد تا در استفاده از DSP، FF و LUT به مقدار قابل توجهی صرفه جویی شود. در جدول ۱ منابع موجود روی برد pynq-z1 نشان داده شده است.

جدول ۱- منابع موجود روی برد pynq-z1

BRAM_18k	DSP48E	FF	LUT
280	220	106400	53200

همانطور که در جدول ۲ و جدول ۳ مشخص شده است استفاده از روش‌های تقریبی برای پیاده‌سازی تابع‌های فعال‌ساز Sigmoid و Tanh باعث بهبودی ۹۰ درصدی در مصرف منابع این دو تابع شده است. در جدول ۴ مقایسه منابع مصرفی لایه Softmax نشان داده شده است که با توجه به آمار حدود ۱۶ درصد در مصرف منابع بهبود داده شده است.

جدول ۲- مقایسه منابع مصرفی تابع فعال ساز Sigmoid با روش تقریبی پیشنهادی (Proposed Approx.) و بدون روش تقریبی (no Approx.)

Sigmoid	BRAM_18k	DSP48E	FF	LUT
No Approx.	0	7	907	2519
Proposed Aprox.	0	0	17	229
Improvement	0	100%	98%	99%

جدول ۳- مقایسه منابع مصرفی تابع فعال ساز Tanh با روش تقریبی پیشنهادی(Proposed Approx.) و بدون روش تقریبی (No Approx.)

Tanh	BRAM_18k	DSP48E	FF	LUT
No approx.	0	14	1742	4719
Proposed Aprox.	0	0	25	398
Improvement	0	100%	99%	99%

جدول ۴- مقایسه منابع مصرفی یک لایه softmax با روش تقریبی پیشنهادی (Proposed Approx.) و بدون روش تقریبی (No Approx.)

Softmax	BRAM_18k	DSP48E	FF	LUT
No Approx.	0	7	1384	3964
Proposed Aprox.	0	6	1058	3564
Improvement	0	15%	25%	10%

قابل ذکر است که پیاده‌سازی شبکه با HLS نسبت به HDL دو برابر سریع‌تر انجام می‌شود، و زمان تست و پیاده-سازی نیز به مقدار چشم‌گیری پایین است. پیاده‌سازی شبکه با استفاده از این واسط سخت‌افزاری نرم‌افزاری زمان مورد نیاز برای پیاده‌سازی دستی را حذف کرده و در سریع‌ترین زمان ممکن یک شبکه تولید می‌کند. علاوه بر این، این امکان را فراهم می‌کند که شبکه‌ی بدست امده کارآیی بالاتری داشته باشد و همچنین قابلیت تغییر شبکه، توسط طراح نیز همچنان وجود داشته باشد. علاوه بر طراحان سخت‌افزاری، امکان استفاده از این پلتفرم برای طراحان نرم‌افزاری هم وجود دارد تا بدون نیاز به دانش سخت‌افزاری بالا شبکه‌ی خود را روی سخت‌افزار پیاده سازی کنند.

۷.نتیجه گیری

در این مقاله، واسطی سخت‌افزاری نرم‌افزاری ارائه شد که می‌تواند با بهره‌گیری از روش‌های تقریبی و روش‌های بهینه‌سازی حافظه on-chip، شبکه های بهینه و قابل پیاده سازی روی FPGA را از شبکه‌های تولید شده در پلتفرم‌های نرم‌افزاری تولید کند. نتایج بررسی‌های واسط پیشنهادی مشخص کرد که بیشتر شبکه‌های تولید شده در پلتفرم‌های نرم‌افزاری با روش‌های بهینه سازی به کار گرفته شده در این واسط قابل پیاده سازی روی FPGA هستند. همچنین با توجه به اینکه تولید کد سخت‌افزاری در یک زبان سطح بالا و توسط ماشین انجام می‌شود، علاوه بر بالا بردن سرعت پیاده-

سازی، امکان تغییرات بیشتر برای استفاده‌کننده را نیز در حین کار فراهم می‌کند و همچنین طراحان و کاربران نرم‌افزاری هم می‌توانند بدون داشتن دانش بالا در زمینه پیاده‌سازی سخت افزاری از این ابزار استفاده کنند.

۸.مراجع

1. Chowdhury, Gobinda G. "Natural language processing." Annual review of information science and technology 37, no. 1 (2003): 51-89.

2. Chiu, Chung-Cheng, Tara N. Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan et al. "State-of-the-art speech recognition with sequence-to-sequence models." In 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 4774-4778. IEEE, 2018.

3. Aleksic, Petar, and Pedro J. Moreno Mengibar. "Voice recognition system." U.S. Patent 10,049,666, issued August 14, 2018.

4. Wigington, Curtis, Chris Tensmeyer, Brian Davis, William Barrett, Brian Price, and Scott Cohen. "Start, follow, read: End-to-end full-page handwriting recognition." In Proceedings of the European Conference on Computer Vision (ECCV), pp. 367-383. 2018.

5. Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9, no. 8 (1997): 1735-1780.

6. Hasim Sak, Andrew Senior, and Fran¸coise Beaufays. Long short term memory recurrent neural network architectures for large scale acoustic modeling. In Proceedings of the Annual Conference of International Speech Communication Association (INTERSPEECH), 2014.

7. Guan, Y., Yuan, Z., Sun, G., Cong, J.: FPGA-based accelerator for long short-term memory recurrent neural networks. In: 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC),pp. 629–634(2017).

8. Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates," in Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2017, pp. 152–159.

9. Andre Xian Ming Chang, Berin Martini, and Eugenio Culurciello. Recurrent neural networks hardware implementation on fpga. arXiv preprint arXiv:1511.05552, 2015.

10. J. C. Ferreira, J. Fonseca, "An fpga implementation of a long short-term memory neural network" in ReConFigurable Computing and FPGAs (ReConFig) 2016 International Conference on, IEEE, pp. 1-8, 2016.

11. Amin, H., et all: "Piecewise linear approximation applied to nonlinear function of a neural network", *IEE Proc. Circuits, Devices Sys*., 144, (6), pp. 313–317J, 1997.

12. Pedro Manuel Afonso Costa " Customized Hardware for Long-Short Term Memory Networks in Embedded Systems", IEEE, 2020.

13. I. Kouretas, V. Paliouras "Simplified Hardware Implementation of the Softmax Activation Function", 8th International Conference on Modern Circuits and Systems Technologies, 2019.

14. Zhiqiang Que, Yongxin Zhu, Hongxiang Fan, Hongxiang Fan, Jiuxi Meng, Xinyu Niu & Wayne Luk, "Mapping Large LSTMs to FPGAs with Weight Reuse", Journal of Signal Processing Systems,2020.

15. D. Svozil, V. Kvasnicka, J. Pospichal "Introduction to multi-layer feedforward neural networks", Chemometrics Intell Lab Systems, 39 (1997), pp. 43-62.

16. Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio, “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”, Presented in NIPS 2014 Deep Learning and Representation Learning Workshop, 2014.

17. Asano, Shuichi, Tsutomu Maruyama, and Yoshiki Yamaguchi. "Performance comparison of FPGA, GPU and CPU in image processing." In 2009 international conference on field programmable logic and applications, pp. 126-131. IEEE, 2009.

18. Tétrault Marc-André, “Two FPGA Case Studies Comparing High Level Synthesis and Manual HDL for HEP applications”, IEEE NPSS Real Time Conference Record, 2018.