

İSTANBUL TEKNİK ÜNİVERSİTESİ
ELEKTRİK-ELEKTRONİK FAKÜLTESİ

**AÇIK KAYNAKLI İŞLEMCİ ÜZERİNDE DERİN SİNİR
AĞLARINI HIZLANDIRICI DONANIM TASARIMI**

LİSANS BİTİRME TASARIM PROJESİ

Nazım Altar Koca

Berkay Yıldız

Yusuf Caner Demirkol

ELEKTRONİK VE HABERLEŞME MÜHENDİSLİĞİ BÖLÜMÜ

HAZİRAN, 2021

İSTANBUL TEKNİK ÜNİVERSİTESİ
ELEKTRİK-ELEKTRONİK FAKÜLTESİ

**AÇIK KAYNAKLI İŞLEMCİ ÜZERİNDE DERİN SİNİR
AĞLARINI HIZLANDIRICI DONANIM TASARIMI**

LİSANS BİTİRME TASARIM PROJESİ

Nazım Altar Koca
(040160006)

Berkay Yıldız
(040160116)

Yusuf Caner Demirkol
(040160044)

Proje Danışmanı: Prof. Dr. Sıddıka Berna Örs Yalçın

ELEKTRONİK VE HABERLEŞME MÜHENDİSLİĞİ BÖLÜMÜ

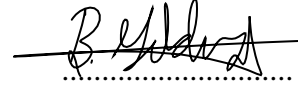
HAZİRAN, 2021

İTÜ, Elektronik ve Haberleşme Mühendisliği Bölümü'nün ilgili Bitirme Tasarım Projesi yönergesine uygun olarak tamamen kendi çalışmamız sonucu hazırladığımız "AÇIK KAYNAKLI İŞLEMCİ ÜZERİNDE DERİN SİNİR AĞLARINI HIZLANDIRICI DONANIM TASARIMI" başlıklı Bitirme Tasarım Projesi'nin Ara Raporu'nu sunmaktayız. Bu çalışmayı intihal olmaksızın hazırladığımızı taahhüt eder; intihal olması durumunda bitirme tasarım projesinin başarısız sayılacağını kabul ederiz.

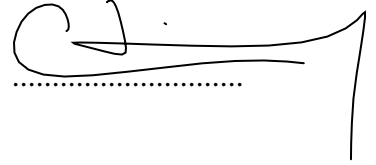
Nazım Altar Koca
(040160006)



Berkay Yıldız
(040160116)



Yusuf Caner Demirkol
(040160044)



ÖNSÖZ

Bitirme projemiz boyunca bizim yanımızda olan ve her hafta vaktini ayıran, bize olan desteğini hiç esirgemeyen sayın Prof.Dr. Sıddıka Berna ÖRS YALÇIN'a çok teşekkür ederiz.

İstanbul Teknik Üniversitesi lisans eğitimi hayatımız boyunca bize katkı sağlayan, bakış açımızı genişleten değerli hocalarımıza ve arkadaşlarımıza teşekkür ederiz.

Proje boyunca karşılaştığımız zorluklarda bize moral ve motivasyon desteği sunan tasarımcı Corey DAVIS'e teşekkür ederiz.

Bitirme projemizde 2209-A kapsamında desteklerinden dolayı Tübitak BİLGEM TÜTEL laboratuvarına teşekkür ederiz.

Son olarak, her anımızda bize destek olan ve hiçbir şeylerini esirgemeyen değerli ailelerimize sonsuz teşekkürlerimizi sunarız.

HAZİRAN 2021

Nazım Altar Koca
Berkay Yıldız
Yusuf Caner Demirkol

İÇİNDEKİLER

Sayfa

ÖNSÖZ	iv
İÇİNDEKİLER	v
KISALTMALAR	vii
TABLO LİSTESİ	viii
ŞEKİL LİSTESİ	ix
ÖZET	xi
SUMMARY	xii
1. GİRİŞ	1
1.1 Genel Bakış	1
1.1.1 Yapay Sinir Ağları	2
1.1.1.1 Perceptron	2
1.1.1.2 Adaline	4
1.1.1.3 Çok Katmanlı Algılayıcı	5
2. MLP Algoritmasının Python ve C Seviyesinde Gerçeklenmesi	7
2.1 Çok Katmanlı Algılayıcının Python İle Gerçeklenmesi	7
2.2 Fashion-Mnist Veri Kümesinin MLP ile Gerçeklenmesi	9
2.2.1 Veri Seti	9
2.2.1.1 Genel Bakış	9
2.2.1.2 Veri Kümesinin Teknik Özellikleri	9
2.2.1.3 MLP ile Sınama Sonuçları	12
2.2.2 Hazır Kütüphane ile Problemin Çözülmesi	13
2.2.3 Özel Hazırlanan Kütüphane ile Problemin Çözülmesi	16
2.2.3.1 Sonuçların Analizi	18
2.2.3.2 Confussion Matrix	20
2.2.3.3 RELU Aktivasyon Fonksiyonu ve Dropout	21
2.2.3.4 Softmax Aktivasyon Fonksiyonu ve CrossEntropyLoss	23
2.3 MLP Algoritmasının C Gerçeklenmesi	26
3. RISC-V VE MLP MODELİNİN FPGA UYGULAMALARI	30
3.1 Uygulama Kodunu Fpga Kartına Yüklenir Hale Getirmek	30
3.2 MAC Ünitesi	32
3.3 Aktivasyon Fonksiyonu	36
3.3.1 Relu	36
3.3.2 Sigmoid	37
3.3.3 Tanh	38
3.4 Riscv İşlemcisi İle Fpga Kartı Üzerinde Uart Uygulamaları	40
3.4.1 Yazı Uygulaması	40
3.4.1 Tam Sayı Uygulaması	40
3.5 Riscv İşlemcisi İle Fpga Kartı Üzerinde Float İşlem Uygulaması	41
3.6 Yapay Zeka Modelinin Soft Float İle Fpga Üzerinde Gerçeklenmesi	42
4. DAVRANIŞSAL VE ZAMANLAMA ANALİZİ	46

4.1 Karşılaşılabilecek Potansiyel Hatalar Ve Çözümleri	47
4.2 Simülasyon Adımları	49
4.3 Soft-Hard Float	52
5. CUSTOM HARDWARE	54
5.1 AXI Arayüzü	54
5.2 Yerel Hafıza Yapısı	58
5.3 Custom IP Tasarımı	59
6. GERÇEKÇİ KISITLAR, SONUÇLAR VE ÖNERİLER	65
6.1 Çalışmanın Uygulama Alanı	65
6.2 Gerçekçi Tasarım Kısıtları	65
6.2.1 Maliyet	65
6.2.2 Standartlar	65
6.2.3 Sosyal, Çevresel ve Ekonomik Etki	65
6.2.4 Sağlık ve Güvenlik RİSKLERİ	65
6.3 Sonuçlar	66
6.4 Geleceğe Yönelik Öneriler	66
KAYNAKLAR.....	67
EKLER	69
EK 1. Perceptron Python Kodu	69
EK 2. Adaline Python Kodu	70
EK 3. ÇKA Python Kodu	72
EK 4. ÇKA C Kodu.....	78
ÖZGEÇMİŞLER.....	82

KISALTMALAR

PULP	: Parallel Ultra Low Power
RISC	: Reduced Instruction Set Computer
FPU	: Floating Point Unit
UART	: Universal Asynchronous Receiver Transmitter
FPGA	: Field-Programmable Gate Array
VHDL	: Very High Speed Integrated Circuit Hardware Description Language
MLP	: Multi Layer Perceptron
MAC	: Multiply Accumulate Operation
ReLU	: Rectified Linear Unit
ÇKA	: Çok Katmanlı Algılayıcı
DNN	: Deep Neural Network
ADALINE	: Adaptif Lineer Nöron
CPU	: Central Process Unit
TPU	: Tensor Processing Unit
AXI	: Advanced eXtensible Interface

TABLO LİSTESİ

Sayfa

Tablo 2.1: Fashion-Mnist Veri Sınıfları[18].	20
Tablo 2.2: Farklı Aktivasyon Fonksiyonları ve Katman Sayıları ile Fashion-Mnist(sol) ve Mnist(sağ) MLP Sınaması[18]	21
Tablo 2.3: Ağırlık İlk Koşullarının Etkisi	22
Tablo 2.4: Öğrenme Hızının ve Momentum Teriminin Etkisi	34
Tablo 4.1: Hard float çıktıları.....	22
Tablo 4.2: Soft float çıktıları.	23
Tablo 4.3: Frekansın simülasyon süresine etkisi.	44

ŞEKİL LİSTESİ

Sayfa

Şekil 1.1: Doğrusal ve doğrusal olmayan ayırıştırma.....	3
Şekil 1.2: Genlikte Ayrık Algılayıcı.....	4
Şekil 1.3: Genlikte Sürekli Algılayıcı için Öğrenme Kuralı.....	4
Şekil 1.4: Çok Katmanlı Algılayıcı Modeli.....	5
Şekil 2.1: Çok Katmanlı Algılayıcı Yapısının Iris Veriseti ile Sınanması	7
Şekil 2.2: Billing Sistemi Çıktıları(Kırmızı) ile Ağ Çıktıları(Mavi).....	8
Şekil 2.3: Fashion-Mnist Veri Kümesi Genel Bakış	10
Şekil 2.4: Veri Kümesinin Şekillendirme Aşaması.....	10
Şekil 2.5: Fashion-Mnist ile Mnist Veri kümelerinin Karşılaştırması.....	12
Şekil 2.6: Kapsamlı Fashion-Mnist MLP Sınaması	13
Şekil 2.7: Keras Kütüphanesi ile Elde Edilen Çıktılar	14
Şekil 2.8: Keras – İterasyon ve Doğruluk Karşılaştırması	15
Şekil 2.9: Keras – İterasyon ve Kayıp Karşılaştırması	15
Şekil 2.10: Kümelerin Ayrılması-Normalizasyon	16
Şekil 2.11: Aksivasyon Fonksiyonlarının Tanımlanması	17
Şekil 2.12: İleri Yol Yayılım.	17
Şekil 2.13: Geri Yol Yayılım	18
Şekil 2.14: Ortalama Hata	19
Şekil 2.15: Hata Vektörü ve Doğru Sınıflandırılan Veri Sayısı	19
Şekil 2.16: Confussion Matrix Örneği	20
Şekil 2.17: Ağ Yapımız için Confussion Matrix Sonucu	21
Şekil 2.18: RELU Aktivasyon Fonksiyonu ve Türevi.....	21
Şekil 2.19: İdeal ve Aşırı Öğrenme Eğrileri.	22
Şekil 2.20: Dropout Fonksiyonu.....	22
Şekil 2.21: RELU ve Dropout ile Eğitim Hatasının Değişimi	23
Şekil 2.22: Softmax Fonksiyonu.	24
Şekil 2.23: Cross Entropy Hata ve Tahmin Değişim Grafiği	24
Şekil 2.24: Cross Entropy Gradyen Hesapları	25
Şekil 2.25: Softmax ve Cross Entropy Hata vs İterasyon Grafiği.....	25
Şekil 2.26: Python Kodunun Çıktısı	27
Şekil 2.27: C Kodunun Çıktısı	28
Şekil 2.28: Gcc Derleyicisi Çıktısı	29
Şekil 3.1: FPGA Uygulaması için Eklenen Tasarım Dosyaları	30
Şekil 3.2: FPGA Kartının Projeye Tanıtılması	31
Şekil 3.3: Optimizasyon Seviyesinin Ayarlanması	31
Şekil 3.4: Vivado Flash Scriptleri	32
Şekil 3.5: MAC Ünite VHDL Şematik	33
Şekil 3.6: MAC Ünite Simülasyon Çıktısı(2 adet 32 bit giriş ile).....	34
Şekil 3.7: MAC Ünite Simülasyon Çıktısı(1000 adet giriş için).....	34
Şekil 3.8: Strachpad Bellek Bloğu Simülasyon Çıktısı.	35
Şekil 3.9: RELU Aktivasyon Fonksiyonu Simülasyonu.	36
Şekil 3.10: Gerçek Sigmoid-Yakınsama Fonksiyonu Grafiği.....	37
Şekil 3.11: Sigmoid Fonksiyonu Simülasyon Sonuçları.	38
Şekil 3.12: Tanh Aktivasyon Fonksiyonu	38
Şekil 3.13: Top_module Hiyerarşisi	39
Şekil 3.14: Tanh Simülasyon Sonuçları.....	40

Şekil 3.15: UART Üzerinden Yazı Gönderilmesi ...	40
Şekil 3.16: UART Üzerinden Tam Sayı İşlem Sonucu Görüntüleme	41
Şekil 3.17: Led ile Float İşlem Kontrolü	42
Şekil 3.18: Donanımsal Olarak Dataram Boyutunu Değiştirme.....	43
Şekil 3.19: Block Ram Boyutunu Büyütmek için Değiştirilmesi Gereken TCL.....	43
Şekil 3.20: Pulpino Hafıza Hiyerarşisi	44
Şekil 3.21: link.common.ld Dosyası İçerisinde Değişim	45
Şekil 3.22: link.boot.ld Dosyası İçerisinde Değişim	45
Şekil 3.23: s19toslm.py Dosyası İçerisindeki Değişim	45
Şekil 4.1: Make vcompile Çıktısı.	47
Şekil 4.2: Make helloworld Terminal Çıktısı.	47
Şekil 4.3: .baschrc Eklenen Pathler.	49
Şekil 4.4: Basit c Kodu	49
Şekil 4.5: Uart Çıktısı.	49
Şekil 4.6: ModelSim Arayüzü	50
Şekil 4.7: Basit Float Kodu	51
Şekil 4.8: Wave Float İşlem Çıktıları.	51
Şekil 4.9: Ana Model Çıktısı	52
Şekil 4.10: Performans Sayıcı Çıktıları	53
Şekil 5.1: AXI Full Veri Yazma Aksiyonu	54
Şekil 5.2: AwSIZE Sinyali Byte Bilgileri[1].....	55
Şekil 5.3: AXI Full Veri Okuma Aksiyonu	55
Şekil 5.4: AXI Full Sinyal Tanımları	56
Şekil 5.5: Top Modülde Yapılan Değişiklikler	56
Şekil 5.6: Custom IP Port Dizilimi	58
Şekil 5.7: Ağırlıkların Verilog formatına dönüştürülmesi.....	58
Şekil 5.8: MAC İşlem Birimi	59
Şekil 5.9: MAC ve Aktivasyon Fonksiyonu ile Gerçekleşmiş Nöron Yapısı	60
Şekil 5.10: Sonuçların Hassasiyetini Karşılaştırmak için Yazılan Python Kodu	61
Şekil 5.11 : Simülasyon Sonucu İlk Katman Çıkışı.....	61
Şekil 5.12: Python Modeli Sonucu İlk Katman Çıkışı	61
Şekil 5.13: İlk Katman Aktivasyon Fonksiyon Çıkışı	62
Şekil 5.14: Simülasyon Sonucu İkinci Katman Çıkışı.....	62
Şekil 5.15: Python Modeli Sonucu İkinci Katman Çıkışı.....	63
Şekil 5.16: İkinci Katman Aktivasyon Fonksiyon Çıkışı	63
Şekil 5.17: Simülasyon Sonucu Çıkış Katmanı Sonuçları	64
Şekil 5.18: Python Modeli Sonucu Çıkış Katmanı Çıktıları.....	64
Şekil 5.19: Donanım Bloğunun Zamanlama Analizi.....	64

Açık Kaynak İşlemci Üzerinde Derin Sinir Ağlarını Hızlandırıcı Donanım Tasarımı

ÖZET

Günümüzde otonom araçlar, robotik ve IoT alanlarındaki gelişmeler ile birlikte sahada yapay zeka uygulamalarına ihtiyaç artmıştır. Sıkça karşılaşılan problemler arasında olan sınıflandırma problemleri araştırmacılar tarafından çokça ilgi çekmekte ve gerek yazılım gerek donanım tarafında yeni çalışmalara kapı açmaktadır. Donanım tarafında transistör boyutu ve çalışma frekanslarında sınır noktalara eriştikçe alan özelinde mimarilere(Domain Specific Architecture - DSA) ilgi artmaktadır ve birçok irili ufaklı firma bu problemlere çözüm geliştirmeye başlamıştır.

Sınıflandırma problemleri için en sık kullanılan iki ağ yapısı Konvolüsyonel Sinir Ağları(Convolutional Neural Networks - CNN) ve Çok Katmanlı Algılayıcı(MultiLayer Perceptron - MLP) yapılarıdır. Bu çalışmada hem matematiğinin kolaylığı hem de hala birçok problemin çözümünde kullanılan MLP yapısı esas alınmıştır. Python ve C seviyesinde sıfırdan kütüphaneler geliştirilerek yaygın kullanılan bir karşılaştırma(benchmark) problemi ile yazılan kodlar test edilmiştir.

Çalışmanın asıl amacı olarak üst dillerde hazırlanan bu modellerin doğruluk oranlarından bir şey kaybetmeden işlemde paralelleştirme ve yerel hafıza yaklaşımları ile özel tasarlanan donanımlar ile daha hızlı ve verimli şekilde gerçekleşmesi olmuştur. Çalışma platformu olarak son yıllarda açık kaynak donanımlar arasında öne çıkan RISC-V ortamı tercih edilmiştir. Bu işlemci ortamı FPGA kartı üzerinde gerçekleştirilerek gerçek hayat kısıtları gözlenmiştir. Seçilen RISC-V ortamına evrensel veri hatları yardımıyla özel donanım eklenmiş ve sonuçlar doğrulanarak karşılaştırmalı olarak sunulmuştur.

Bu doğrultuda gerçek sayılarla gerçekleştirilen model önce tamsayı(integer) komutları ile ardından ondalıklı sayı(floating) komutları ile işlemci üzerinde gerçekleştirilmiş son olarak da tasarlanan özel donanım bloğu üzerinde algoritma gerçekleştirilerek aradaki dramatik gelişme karşılaştırmalı olarak sunulmuştur. Bu çalışma sonucunda amaca özel donanım tasarımının sinir ağları gibi veri yoğun uygulamalarda ne kadar fazla iyileştirmeye neden olabileceği net şekilde gözlenmiştir.

Deep Neural Network Hardware Accelerator Design on Open Source RISC-V Processor

SUMMARY

There are more need now for edge ai devices with the rapid development of autonom cars, robotics and IOT. Image classification is one of the most faced problems in this area. To overcome these issues, the software and hardware co-design approach is needed. With the real constraints in transistor size and frequency, domain specific architectures(DSA) become so popular. Specially, the general purpose computing units and DSA units are used together in chip design to get the advantage of both side. There are lots of company which aim to develop a solution to these type of data-driven systems like Intel Mobileye, Cerebras Sys, etc.

Convolutional Neural Networks(CNNs) and Multi Layer Perceptrons(MLPs) are the most used network to solve classification problems. In this project MLP network is chosen to be implemented for the sake of simplicity of its Math. In both Python and C level, the model code is written from scratch to understand the math behind it. For benchmark, Fashion-Mnist classification problem is selected. It is more complicated then classic Mnist hand-written digit problem. So thus, more feasible results are obtained.

Main focus of this projects is to develop an efficient software and hardware architecture to implement the models mentioned above. To get the best possible weights in the network, many network concepts like(dropout, momentum effect etc.) are added in software level. On the other hand, in hardware, it is aimed to get same accuracy as in software with the efficient implementation of it.

Many hardware concepts are reviewed to design higly parallel and efficient design. Local memory approach is used to prevent from high memory access latency. A scratchpad memory is added to keep the weights of the network in local memory. And the input pixels are given with the AXI bus serially to obtain the reusability.

RISC-V environment which is a novel open instruction set architecture, is chosen to work on. Pulpino microcontroller project from ETH Zurich is used along with this project. It's FPGA implementation and simulation analysis on Modelsim are presented on this thesis. The custom hardware design is added to this project with AXI Full bus. In this way, custom ip and core are communicated with each other easily.

As a result, the full model of MLP which is implemented on real numbers, is realized firstly on soft-float concept with integer and multiplication instruction set. After that hard-float unit is activated. Both float instruction and Float Point Unit(FPU) is used to implement model. Finally custom hardware is added to processor environment. The main computing and memory units are handled locally and the results were are dramatically better than general purpose computing. With this main result, many skills are obtained through this project like handling an RTL projetc, script-tcl writing, simulation and analysis of RTL design, FPGA implementation of open source project and modelling the problem on high level languages like Python and C.

1. GİRİŞ

1.1 Genel Bakış

Günümüzde görüntü işleme, ses işleme, sınıflama gibi problemlerle daha sık karşılaşmaya başladığımız için bu problemlere çözüm üreten sistemlere duyulan ihtiyaç da artmaktadır. Bu tür problemlerin çözümünde kendini kabul ettirmiş derin sinir ağlarının(Deep Neural Networks - DNN) karmaşık ve çözmesi zaman alan yapısı bu konuda çalışanlar için ciddi bir engel olarak durmaktadır. Günümüz bilgisayar tasarımında kendini öne çıkaran uygulama özelinde mimariler(Domain Specific Architecture) ile DNN problemleri donanım katmanında daha hızlı ve enerji verimli şekilde gerçekleştirilebilmektedir.

Bu donanımlar için ayrık ve çok sayıda işlem biriminin kendi hafıza yapıları ile birçok hesabı paralel olarak yapmaları genelde temel alınmış olsa da bazı yapılar işlemciler ile iç içe tasarlanarak genel amaçlı hesaplama birimlerinin de yararlarından faydalanmaktadır.[1] Bu projede ikinci yapı üzerinden ilerlenecektir. Temel işlem ünitesi olarak son 10 yılda kendini ön plana çıkaran RISC-V mimarili işlemci kullanılacaktır.

RISC-V California Berkeley üniversitesindeki araştırmacılar tarafından derslerde ve araştırma projelerinde kullanılmak üzere geliştirilen açık kaynak bir komut seti mimarisidir.[2] Projenin eriştiği alanlar sonradan genişleyerek tüm dünyada kabul edilen bir mimari haline dönüşmüştür. Temel özellikleri dondurularak(frozen set) yazılım platformları ve geliştiriciler için güvenli bir bölge oluşturmanın yanında esnek mimarisi sayesinde birçok uygulama özelinde genişletilmesi için çalışmalar da yapılmıştır. Bu projenin de amaçladığı bu donanımların sinir ağları için

özelleştirilmesi üzerinde özellikle ETH Zurich Üniversitesinden Pulp Platform grubu yoğun olarak çalışmış ve yayınları ile bunları duyurmuştur.[12]

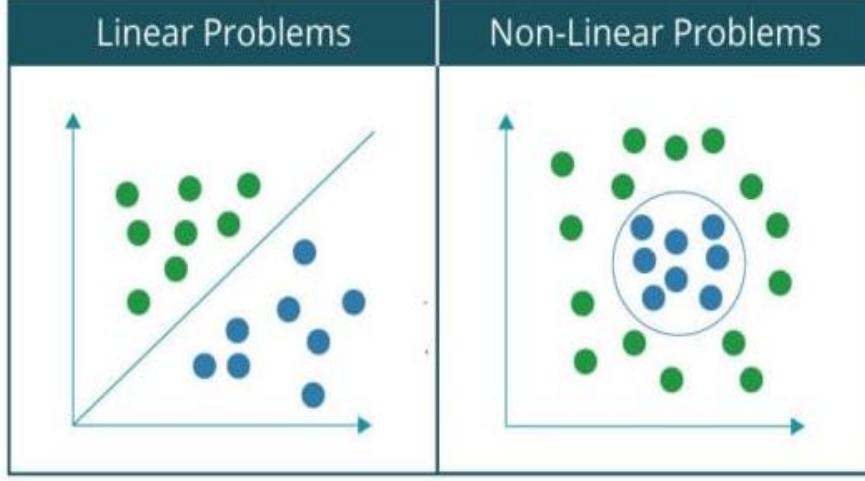
Projenin amaçları doğrultusunda birçok örneği bulunan[13] RISC-V işlemcilerinde çarpıcı ve 32 bit float sayı desteği sağlayan komut setleri üzerinde durulmuştur. Bu doğrultuda yapılan araştırmalarda şu an Open Hardware Group tarafından geliştirilen RISCY işlemcisi çalışmak için seçilmiştir.[14]

1.1.1 Yapay Sinir Ağları

Yapay sinir ağları temek olarak iki yapıya ayrılırlar, bunlar eğitici ve eğitici olmayan sinir ağlarıdır. Eğitici ağlar ağın olması gereken çıktısını bilirler ve olması gereken çıktı ile alınan çıktı arasındaki farka bakarak ağı eğitirler. Bu çalışmada eğitici ağlar üzerinden devam edilecektir. Çalışmanın nihayetinde Çok Katmanlı Algılayıcı yapısının donanım üzerinde hızlandırılması amaçlanmaktadır. Çok Katmanlı Yapının temellerini oluşturan Perceptron, Adaline ve Çok Katmanlı Algılayıcının temel özellikleri kısaca aşağıdaki gibi açıklanabilir.

Perceptron:

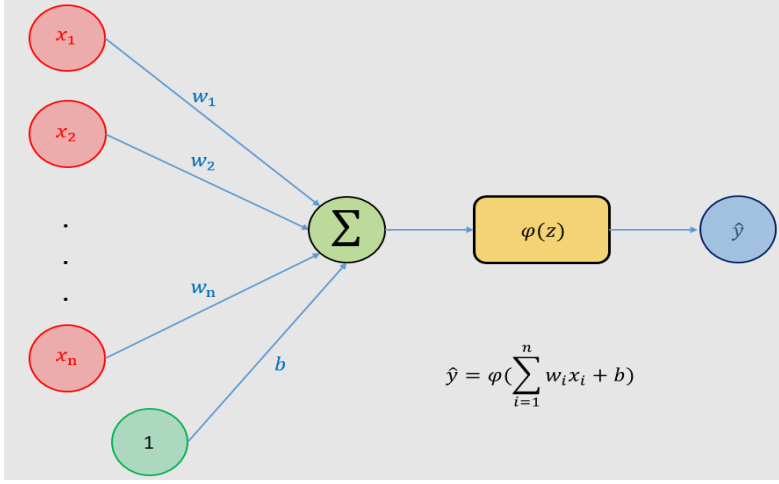
Perceptron genlikte ayrık algılayıcı algoritması Frank Rosenblatt tarafından 1958 yılında geliştirilmiştir.[3] Perceptron doğrusal bir sınıflandırma algoritmasıdır. Temel olarak sınıflandırma algoritmaları, doğrusal olarak ayrıştırılabilir algoritmalar ve doğrusal olmayan ayrıştırılabilir algoritmalar olmak üzere ikiye ayrılmaktadır. Veri setlerini iki boyutta bir çizgi, üç boyutta bir karar düzlemi kullanarak ayrıştırabildiğimiz problemler doğrusal olarak ayrıştırılabilirken, doğrusal olmayan problemlerde bu yaklaşım başarılı olmamaktadır. Şekil 1.1'de görülen doğrusal sınıflandırma problemleri için perceptron algoritması kullanılmaktadır.



Şekil 1.1: Doğrusal ve doğrusal olmayan ayrıştırma

(<https://devhunteryz.wordpress.com/2018/06/30/derin-ogrenme-perceptron-ogrenme-algoritmasi/>)

Perceptron algoritmasına göre, n boyutlu giriş vektörümüz ağırlık vektörüyle çarpılıp toplandıktan sonra $\phi(z)$ aktivasyon fonksiyonundan geçerek y çıkışları elde edilmektedir. Ağırlık vektörü, alınan girişlerin nöron üzerindeki etkisini belirleyen katsayılardan oluşmaktadır. Şekil 1.2’de b ile gösterilen bias terimi, orijinden geçen bir doğruyla eğitim kümemizin ayrıştırılamadığı durumlar için eklenmektedir. Perceptron öğrenme algoritması beklenen çıkış değerleriyle hesaplanan çıkış değerleri arasındaki farka göre ağırlıkların değiştirilmesiyle eğitilmektedir. Hesaplanan sonuçla beklenen sonuç arasındaki fark pozitifse ağırlık değerleri azaltılır, fark negatifse ağırlık değerleri artırılır, fark sıfırsa ağırlık değerleri değiştirilmez. Tüm eğitim setine bu işlem uygulanır ve iterasyon hesaplanan çıkışla beklenen çıkış aynı olana kadar sürer. Ayrıca sonsuz döngü olması durumu göz önüne alınarak belirli bir iterasyon sınırı belirlenir.

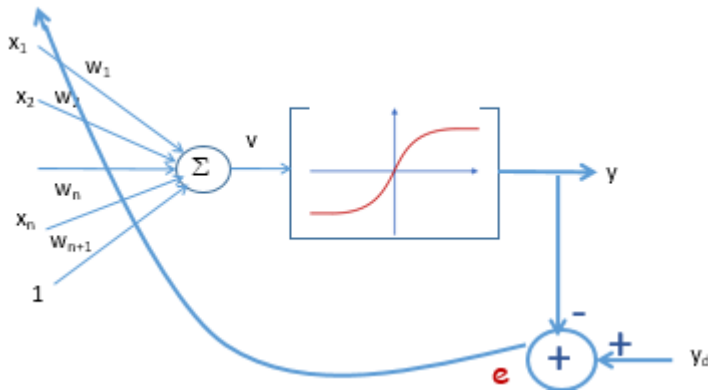


Şekil 1.2: Genlikte Ayrık Algılayıcı

Perceptron yapısı için yazılan ve gerçekleştirilen Python koduna Ekler 1. Kısımında erişilebilir.

Adaline:

Adaline(Adaptif Lineer Nöron) 1959 yılında Stanford Üniversitesinden Bernard Widrow ve Ted Hoff tarafından geliştirilmiştir.[4] Adaline, perceptron gibi tek katmanlı bir doğrusal ayrıştırıcıdır, ancak öğrenme kuralı Perceptron'dan farklıdır. Birçok yapay sinir ağı öğrenme methodu vardır ancak Perceptron ve Adaline eğitimci öğrenme methodunu kullanmaktadır. Bu methotta hem giriş hem de beklenen çıkış değerleri eğitim seti olarak ağa verilmektedir. Beklenen çıkış değeriyle alınan çıkış değeri arasındaki farkı, yani hatayı belirleyerek bir hata fonksiyonu oluşturulur. Amaç hatayı azaltacak ağırlıkları belirlemektir. Şekil 1.3'de Adaline öğrenme kuralı görülmektedir.



Şekil 1.3: Genlikte Sürekli Algılayıcı için Öğrenme Kuralı

(<https://www.linkedin.com/pulse/yapay-sinir-a%C4%9Flar%C4%B1-ve-tek-katmanl%C4%B1-a%C4%9Flarda-%C3%B6%C4%9Frenme-tanju-do%C4%9Fan/>)

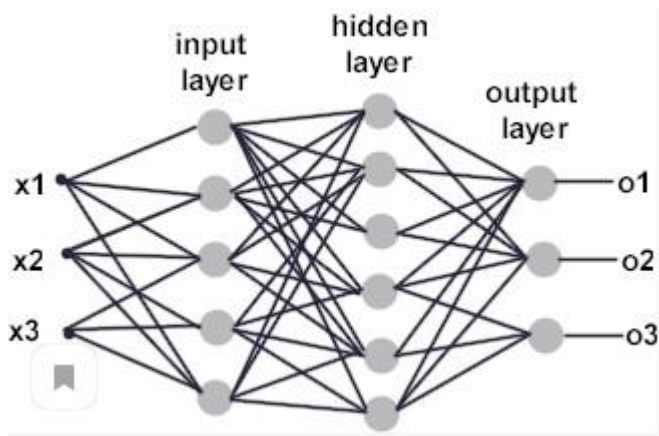
Adaline öğrenme kuralı ortalama karesel hatayı azaltmayı hedefleyerek ağırlıkların güncellenmesini sağlar.

Özetle, Perceptron ve Adaline doğrusal ayrıştırıcı olarak kullanılan tek katmanlı algılayıcılardır. Çıkış değerleri Perceptron’da $\{0,1\}$, Adaline’da $[-1,1]$ ’dir. Elde edilen çıkış değerleriyle tanımlanan sınıflar temsil edilir. Amaç iki sınıfı ayıran bir doğru veya düzlem bulmaktır. Eğitim yapay sinir ağına gelen girişlere ait ağırlık değerlerinin güncellenmesiyle gerçekleşmektedir. Ağırlıkların güncellenmesi, beklenen çıkış değeriyle hesaplanan çıkış değeri arasındaki fark kullanılarak gerçekleştirilir. Başarı ölçütü aranan ayrıştırıcı doğru veya düzlemin iki sınıfı ayırmasıyla belirlenir.

Adaline algılayıcısı için yazılan Python modeline Ekler 2’den ulaşılabilir.

Çok katmanlı algılayıcı:

Bir diğer eğitici öğrenme methodu çok katmanlı algılayıcıdır. Çok katmanlı ağ yapısı Şekil 1.4’te görüldüğü gibi giriş katmanı, çıkış katmanı ve ara katmanlardan oluşmaktadır. Geriye yayılım adı verilen yöntemle eğitici öğrenme gerçekleşmektedir.



Şekil 1.4: Çok Katmanlı Algılayıcı Modeli

(http://www.nuhazginoglu.com/2018/05/15/cok-katmanli-algilayicilar-multi-layer-perceptron/?doing_wp_cron=1611426089.1552031040191650390625)

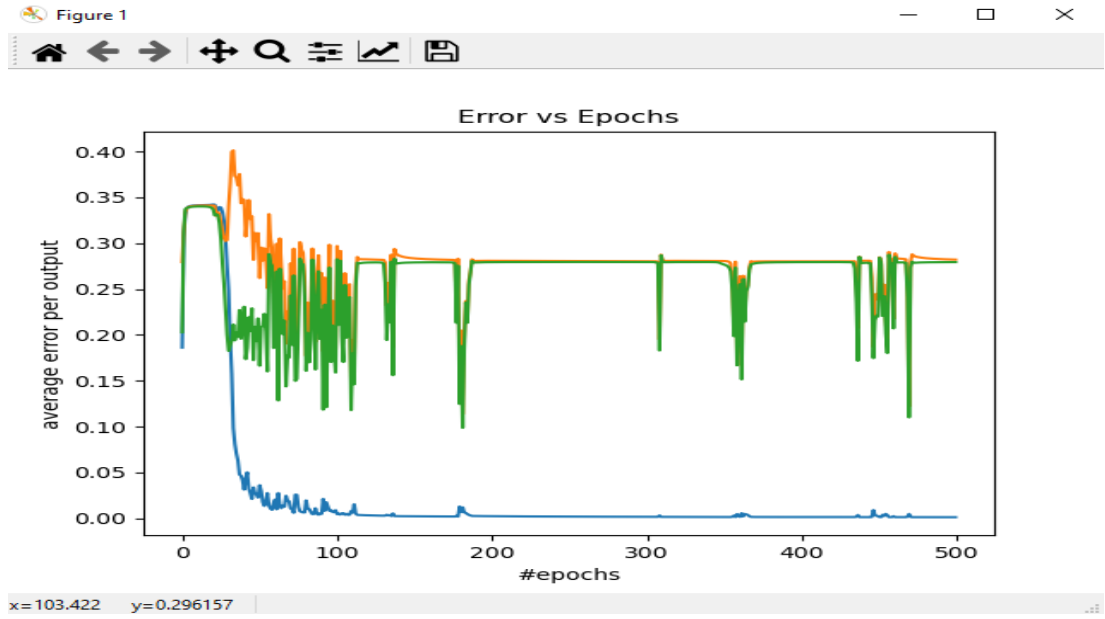
İleri yayılımda giriş değerleri ağırlıklarla çarpıldıktan sonra aktivasyon fonksiyonundan geçirilerek bir sonraki katmana aktarılır. Ara katman sayısı en az birdir ve gelen bilgiler bir sonraki katmana aktarılarak çıkış katmanına kadar yayılım devam eder. Çok katmanlı algılayıcıların öğrenme metodu Delta öğrenme olarak adlandırılır. İleri yol tamamlandıktan sonra çıkış katmanından önceki katmanlara doğru yayılım başlar. Çıkış katmanında hesaplanan toplam hata geriye doğru gradyanları hesaplamak için dağıtılır. Bir optimizasyon yöntemi olan gradyan iniş, geriye doğru ağırlıkları güncelleyerek hatayı azaltmayı amaçlamaktadır.

Çok Katmanlı Ağ Yapısı için oluşturulan Python modeline Ekler 3'ten erişilebilir.

2. MLP ALGORİTMASININ PYTHON VE C SEVİYESİNDE GERÇEKLENMESİ

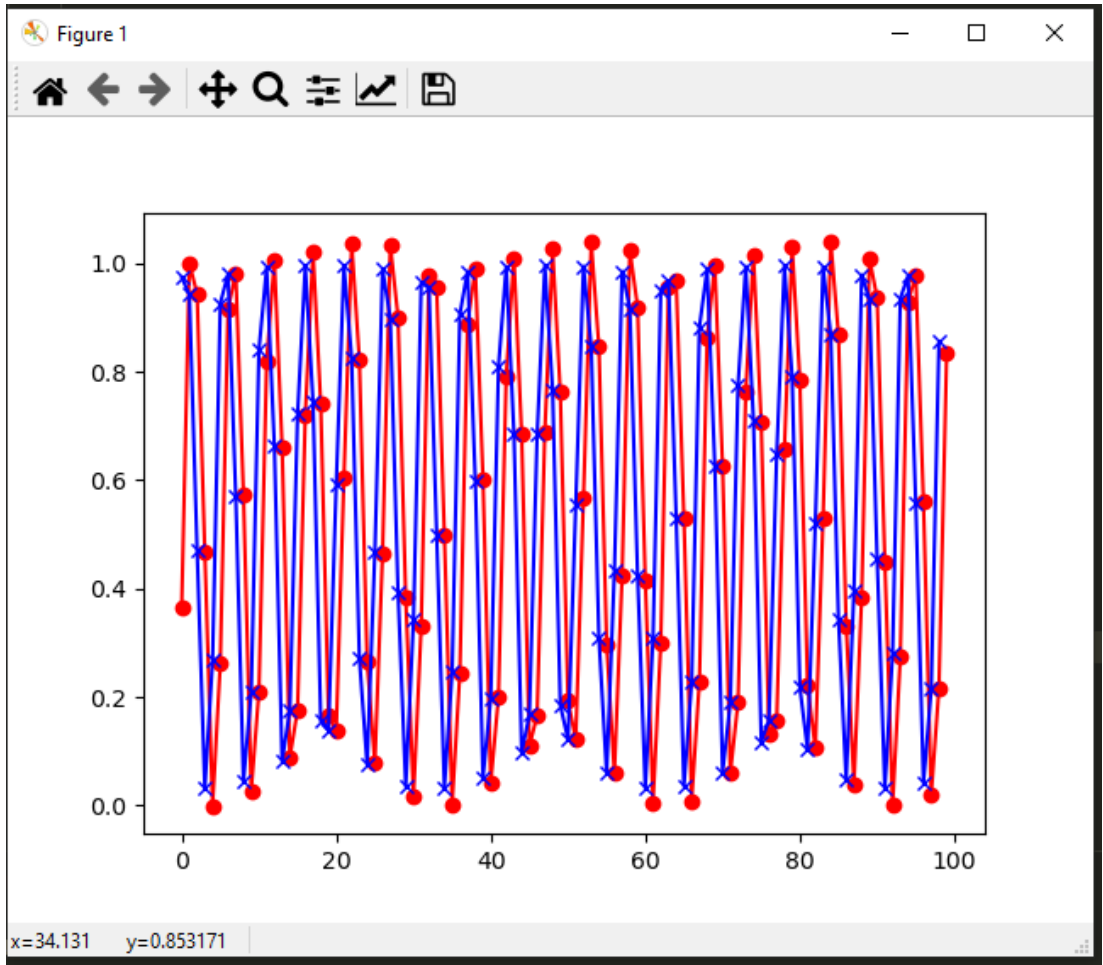
2.1 Çok Katmanlı Algılayıcının Python ile Gerçeklenmesi

Çok katmanlı algılayıcı yapısının matematiksel temellerini daha iyi anlayabilmek adına hazır kütüphane kullanmadan python ile gerçekleştirilmiştir. Kodda iki katmanlı nöron sayısı değiştirilebilir olarak tasarlanan ağ yapısı geriye yayılım algoritması ile öğrenmesini gerçekleştirmektedir. Ağın testleri iki yaygın sınaama yöntemi ile gerçekleştirilmiştir. İlki Iris çiçeğinin türleri arasında ayırım yapması üzerinde gerçekleştirilen testin sonuçları Şekil 2.1’de verilmiştir.



Şekil 2.1: Çok Katmanlı Algılayıcı Yapısının Iris Veriseti ile Sınanması

Diğer bir test yöntemi olarak da dinamik Billing sisteminin tahmini seçilmiştir. Ağın çıktısı ile sistemin olması gereken çıktıların karşılaştırılması Şekil 2.2’de verilmiştir. Görüldüğü gibi model başarılı şekilde çalışmaktadır.



Şekil 2.2: Billing Sistemi Çıktıları(Kırmızı) ile Ağ Çıktıları(Mavi)

Modelin Python kodlarına Ekler 3'ten erişilebilir.

2.2 Fashion-Mnist Veri Kümesinin MLP ile Gerçeklenmesi

2.2.1 Veri Seti

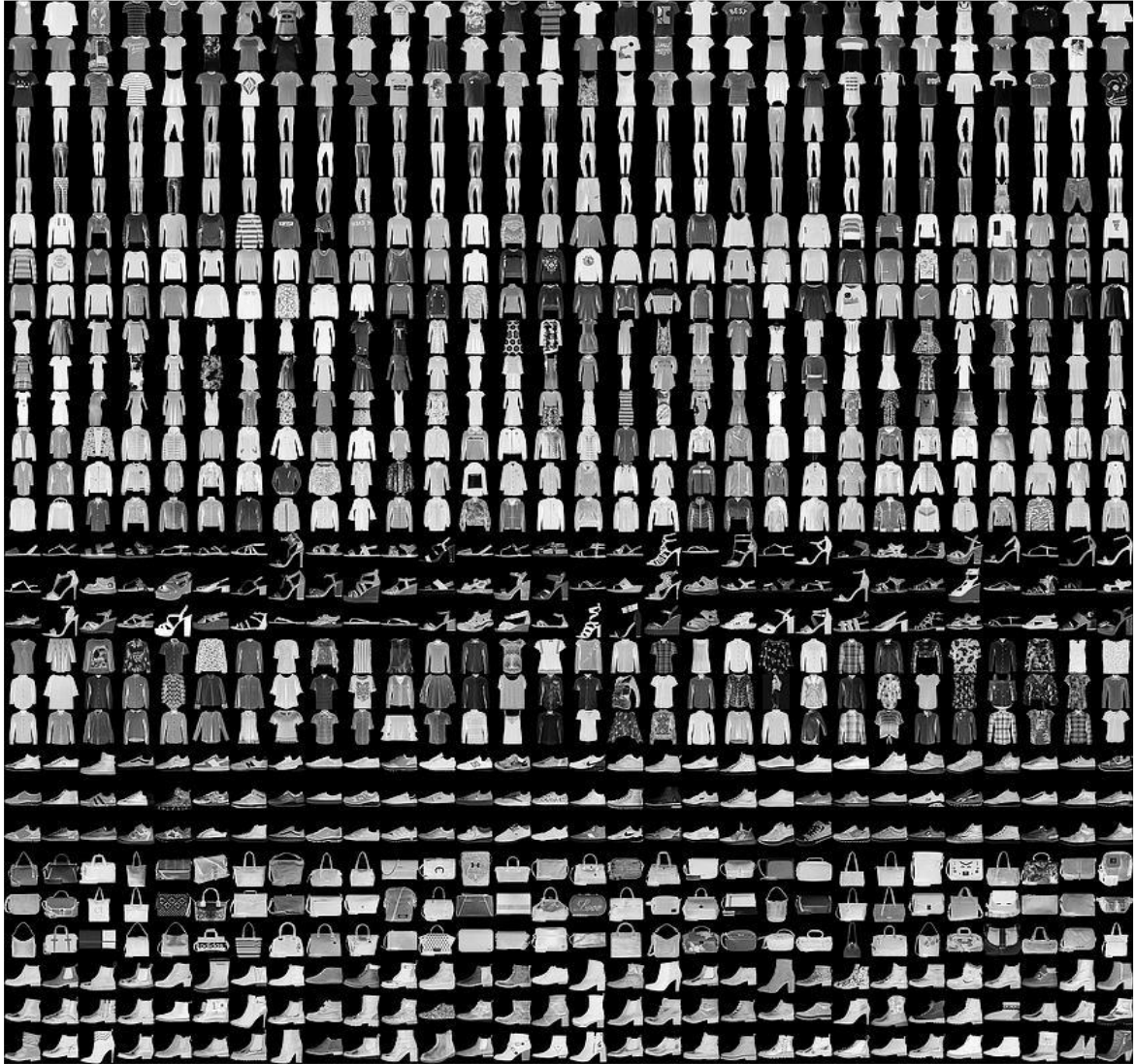
2.2.1.1 Genel Bakış

Fashion-Mnist adıyla bilinen sınaama(benchmark) amaçlı oluşturulan veri kümesinin Çok Katmanlı Algılayıcı yapısı kullanılarak görüntü tanıma-sınıflandırma probleminin çözülmesi bu projede amaçlanmıştır. Fashion-Mnist veri kümesi Alman Zalando firmasının araştırmacıları tarafından 2017 yılında geliştirilen bir veri kümesidir. Veri kümesi adından da anlaşılacağı üzere LeCun'un ünlü Mnist veri kümesinden yola çıkarak geliştirilmiştir. Yaratıcılarından Han Xiao'nun belirttiğine göre[16] Mnist veri kümesinin oluşturulan makine öğrenmesi modellerinin sınanmasında çok yetersiz kalmasından ötürü(CNN algoritmaları ile %99.7 oranında doğruluklara ulaşmak mümkün[17]) aynı format ve büyüklükte daha zor bir test yöntemi geliştirme hevesinden ortaya çıkmıştır. Araştırmacılar tarafından hızla kabul edilen bu sınaama yöntemi kısa zamanda birçok karşılaştırmada kullanılmıştır.

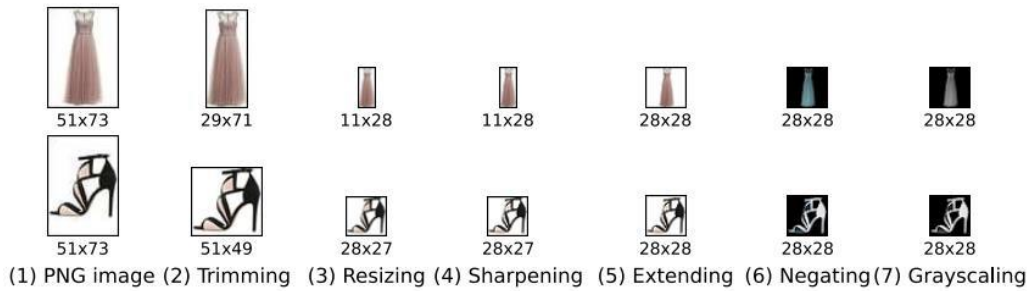
2.2.1.2 Veri Kümesinin Teknik Özellikleri

Veri kümesi 28x28 piksel boyutunda siyah beyaz görsellerden oluşmaktadır. 10 adet farklı giysi sınıfından her birinden 7 bin adet olmak üzere 70 bin adet veri içermektedir. Bunlardan 60 bin tanesi eğitim için 10 bin tanesi test için ayrılmıştır. Şekil 2.14'de veri setinin sınıfları görülmektedir. Giysiler profesyonel fotoğrafçılar 5

tarafından çekilmiş ve Şekil 2.4'te verilen adımlardan geçirilerek aynı formata sokulmuştur.[18]













Şekil 2.3: Fashion-Mnist Veri Kümesi Genel Bakış



Şekil 2.4: Veri Kümesinin Şekillendirme Aşaması

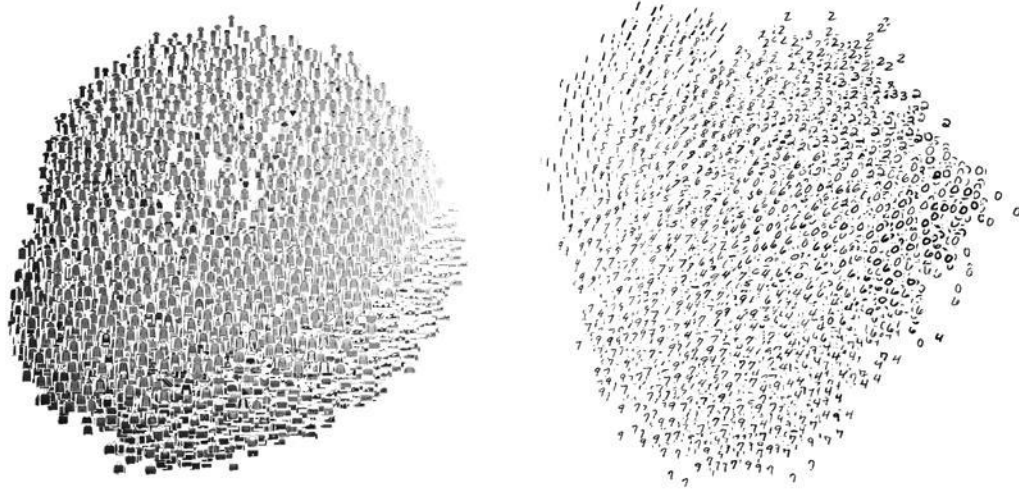
Veri kümesinin sınıfları Tablo 1’de gösterildiği gibidir.

Tablo 1: Fashion-Mnist Veri Sınıfları[18]

Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Şekil 2.5’te Fashion-Mnist ile Mnist veri setinin görsel karşılaştırması görülmektedir.[17]

PCA on Fashion-MNIST (left) and original MNIST (right)



Şekil 2.5: Fashion-Mnist ile Mnist Veri Kümelerinin Karşılaştırması

2.2.1.3 MLP ile Sınama Sonuçları

Veri setinin geliştiricileri tarafından uygulanan MLP ile sınama sonuçları tablo 2’de verilmiştir.

Tablo 2: Farklı Aktivasyon Fonksiyonları ve Katman Sayıları ile Fashion-Mnist(sol) ve Mnist(sağ) MLP Sınaması[18]

MLPClassifier			
	activation=relu hidden_layer_sizes=[100]	0.871	0.972
	activation=relu hidden_layer_sizes=[100, 10]	0.870	0.972
	activation=tanh hidden_layer_sizes=[100]	0.868	0.962
	activation=tanh hidden_layer_sizes=[100, 10]	0.863	0.957
	activation=relu hidden_layer_sizes=[10, 10]	0.850	0.936
	activation=relu hidden_layer_sizes=[10]	0.848	0.933
	activation=tanh hidden_layer_sizes=[10, 10]	0.841	0.921
	activation=tanh hidden_layer_sizes=[10]	0.840	0.921

Görüldüğü gibi aynı aktivasyon fonksiyonu ve katman sayısına rağmen Mnist üzerinde %97 doğruluk oranlarına ulaşan ağ Fashion-Mnist için %87'lerde kalmıştır.

Daha detaylı karşılaştırmalar için sınama sitesi test edilebilir.[19] Siteden MLP için alınan sonuçlar Şekil 2.6'deki gibidir.

Name	Parameter	Accuracy (mean)	Accuracy (std)	Training time	Repeats	Job start	Job Done
MLPClassifier	{'activation':'relu','hidden_layer_sizes':[100]}	0.850	0.003	0:02:58	5	3 years ago	3 years ago
MLPClassifier	{'activation':'relu','hidden_layer_sizes':[10,10]}	0.851	0.003	0:03:19	5	3 years ago	3 years ago
MLPClassifier	{'activation':'tanh','hidden_layer_sizes':[100]}	0.870	0.004	0:19:23	5	3 years ago	3 years ago
MLPClassifier	{'activation':'tanh','hidden_layer_sizes':[100,10]}	0.865	0.006	0:17:58	5	3 years ago	3 years ago
MLPClassifier	{'activation':'tanh','hidden_layer_sizes':[10,10]}	0.840	0.002	0:04:51	5	3 years ago	3 years ago
MLPClassifier	{'activation':'relu','hidden_layer_sizes':[100]}	0.877	0.005	0:16:03	5	3 years ago	3 years ago
MLPClassifier	{'activation':'relu','hidden_layer_sizes':[100,10]}	0.874	0.002	0:15:40	5	3 years ago	3 years ago
MLPClassifier	{'activation':'tanh','hidden_layer_sizes':[10]}	0.841	0.010	0:04:45	5	3 years ago	3 years ago

Şekil 2.6: Kapsamlı Fashion-Mnist MLP Sınaması

Bu tablodan ayrıca seçilen aktivasyon fonksiyonlarının ve katman sayılarının eğitim sürecine etkileri de gözlenebilmektedir.

2.2.2 Hazır Kütüphane ile Problemin Çözülmesi

Fashion-Mnist veri kümesi Python Keras modülü içinde bulunduğu için kolayca kod ortamına çekilmiş ve testleri yapılmıştır. Keras Ortamı MLP komutlarını ve ağ yapısını oluşturmak için kullanılmıştır. Giriş katmanında giriş sayısı 28x28 pikselden dolayı 728 olarak belirlenmiş, çıkış katmanında ise 10 farklı sınıftan dolayı 10 nöron konulmuştur. Ağ yapısı toplamda 1 giriş katmanı, 1 çıkış katmanı ve 3 gizli katmandan oluşmaktadır. Ara katmanların nöron sayıları sırasıyla 256-128-100

şeklinde belirlenmiştir. 100 iterasyon sonucunda elde edilen veriler Şekil 2.19'daki gibidir. Ağın kritik parametleri şöyledir: lr=0.01, momentum=0.975, epochs=100 ve batch_size=100. Öğrenme optimizasyon yöntemi ise Stochastic Gradient Descent'dir.

```
Epoch 100/100
750/750 - 3s - loss: 0.3424 - accuracy: 0.8827 - val_loss: 0.3531 - val_accuracy: 0.8882
Model: "sequential"

```

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 256)	200960
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32896
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 100)	12900
dropout_2 (Dropout)	(None, 100)	0
dense_3 (Dense)	(None, 10)	1010

```

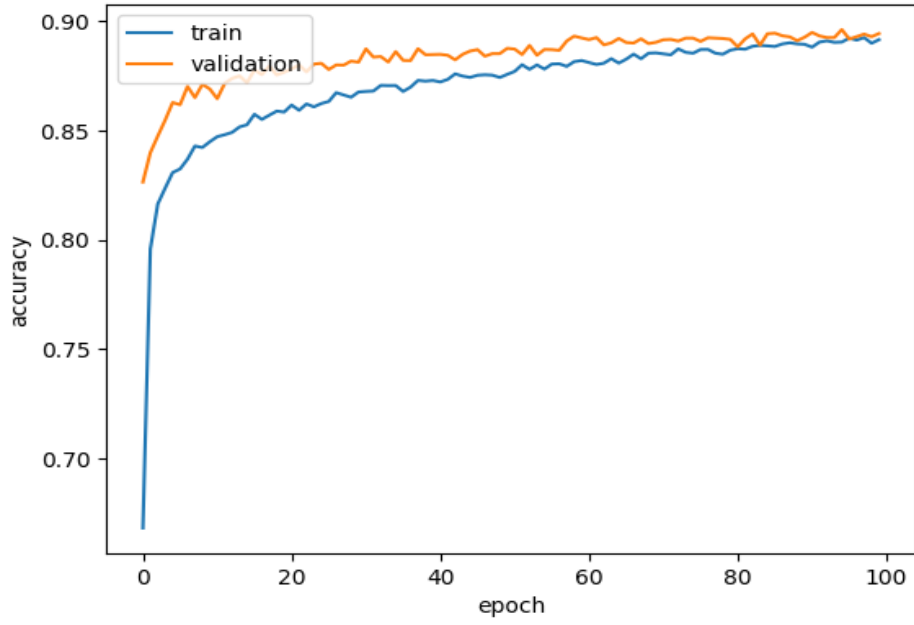
Total params: 247,766
Trainable params: 247,766
Non-trainable params: 0
None
Test loss: 0.384175568819046
Test top 1 accuracy: 0.87680000667572

```

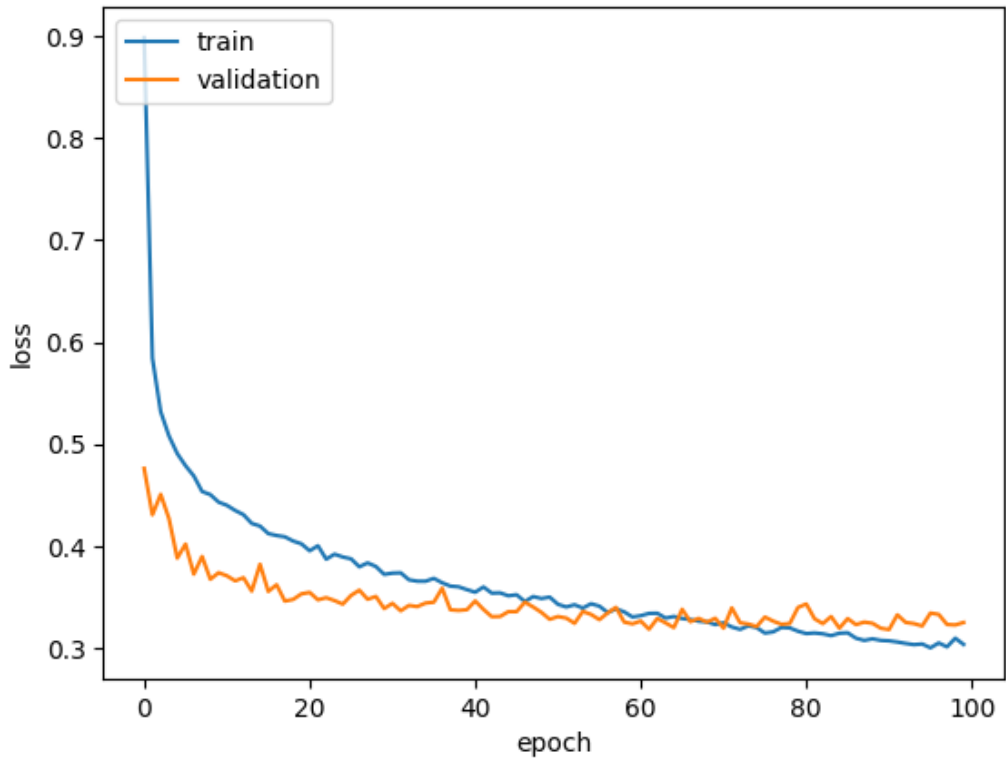
Şekil 2.7: Keras Kütüphanesi ile Elde Edilen Çıktılar

Görüldüğü gibi kayıp fonksiyonu 0.38 değerlerine kadar gerilemiş ve doğruluk oranı %87'lere kadar ulaşmıştır. Şekil 2.8 ve Şekil 2.9'de sırasıyla doğruluk ve kayıp

fonksiyonunun iterasyon sayısına bađlı olarak deđişimi g r lmektedir. Kodlar hazırlanırken kaynakta verilen Github Repo'sunun kaynakları kullanılmıřtır.[20]



řekil 2.8: Keras – İterasyon ve Doğruluk Karşılaştırması



řekil 2.9: Keras – İterasyon ve Kayıp Karşılaştırması

2.2.3 Özel Hazırlanan Kütüphane ile Problemin Çözülmesi

Fashion-Mnist probleminin çözümü için sınıflandırma becerisi yüksek olan çok katmanlı algılayıcının kullanılmasına karar verildi. Veri kümesi Python Keras modülü içinde bulunduğu için kolayca koda eklendi.

Eğitim ve test için veri kümemiz Şekil 2.10'da görüldüğü gibi ayrılarak boyut hatalarını önlemek için yeniden boyutlandırılmıştır. Ardından taşma(overflow)'yı önlemek için eğitim ve test kümelerimiz normalize edilmiştir.

```
23 # the data, split between train and test sets
24 x_train, y_train, x_test, y_test = [], [], [], []
25 (x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
26 x_train = x_train.reshape(train_set_num, x_num, 1)
27 x_test = x_test.reshape(validation_set_num, x_num, 1)
28
29 # normalize the inputs to avoid overflow
30 x_train = x_train.astype('float32')
31 x_test = x_test.astype('float32')
32 x_train /= 255
33 x_test /= 255
```

Şekil 2.10: Kümelerin Ayrılması-Normalizasyon

Veri kümemizde bulunan 10 adet küme için one-hot encoding yöntemi kullanılarak etiketler her bir küme için matris şeklinde tanımlanmıştır.

Ağ için gerekli parametreler ilklendirilmiştir. Ağırlık matrisi her bir katman için tanımlanan başlangıç değerleriyle rastgele olarak, eklenen bias terimi ve boyut düzenlemesiyle birlikte oluşturulmuştur..

```

#sigmoid function
# takes vector and applies element-wise sigmoid function
def sigmoid(vec_x):
    vec_y = []
    for i in range(len(vec_x)):
        vec_y.append(1 / (1 + np.exp(-vec_x[i])))
    return vec_y

def sigmoid_single(x):
    return (1 / (1 + np.exp(-x)))

#derivative of sigmoid
def der_sigmoid(vec_x):
    vec_y = []
    for i in range(len(vec_x)):
        vec_y.append(sigmoid_single(vec_x[i])*(1-sigmoid_single(vec_x[i])))
    return vec_y

```

Şekil 2.11: Aksivasyon Fonksiyonlarının Tanımlanması

Şekil 2.11’de görüldüğü gibi ileri yayılım için sigmoid fonksiyonu tanımlanırken, geriye yayılım için sigmoid fonksiyonunun türevi tanımlanmıştır. Koda daha sonrasında RELU ve SOFTMAX aktivasyon kodları eklemiştir.

```

while iteration < iter_upper_limit:

    iteration = iteration + 1
    sum_e = 0
    vec_train = []
    vec_tyd = []

    for i in range(train_set_num):

        #first layer
        f_v = f_weights_matrix.dot(np.concatenate([train_set[i][:].reshape(28*28 ), [1])))

        f_y = sigmoid(f_v)

        #second layer
        s_v = s_weights_matrix.dot(np.concatenate([f_y, [1]]))

        s_y = sigmoid(s_v)

        #output layer
        out_v = out_weights_matrix.dot(np.concatenate([s_y, [1]]))

        out_y = sigmoid(out_v)
        # end of forward-propagation

        vec_train.append(out_y)
        vec_tyd.append(yd[i])

        #error vector
        vec_e = np.subtract(yd[i], out_y)#, train_set[i][2])

        # sum_e = sum_e + (vec_e**2)
        sum_e = sum_e + np.absolute(vec_e) ** 2

```

Şekil 2.12: İleri Yol Yayılım

Şekil 2.12’de görüldüğü gibi iki ana döngü içinde eğitim aşaması tanımlanmıştır. Katmanlar arasında sigmoid aktivasyon fonksiyonu kullanılarak ileri yol yayılımı oluşturulmuştur. Hata vektörü ve toplam hata için de tanımlamalar yapılmıştır.

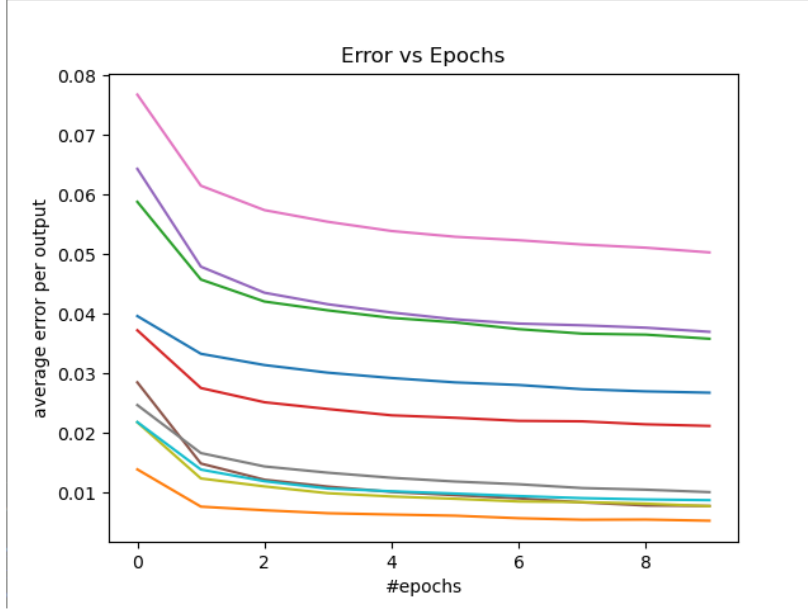
```
### back-propagation
# local gradients
out_gradient = der_sigmoid(out_v)
out_sigma = np.multiply(vec_e, out_gradient)
s_gradient = der_sigmoid(s_v)
s_sigma = np.transpose(out_weights_matrix[:, :s_cell_num]).dot(out_sigma) * s_gradient
f_gradient = der_sigmoid(f_v)
f_sigma = np.transpose(s_weights_matrix[:, :f_cell_num]).dot(s_sigma) * f_gradient
```

Şekil 2.13: Geri Yol Yayılım

Şekil 2.13’de görüldüğü gibi geriye yayılımda sigmoid fonksiyonunun türevi kullanılarak yayılım sağlanmıştır.

2.2.3.1 Sonuçların Analizi

Bilgisayarlarımızın performans kısıtları nedeniyle 10 iterasyon ile çalıştırılan algoritmanın %83-84’lere kadar başarıma ulaştığı görülmüştür. Şekil 2.15’de %83 başarı sağlayan bir örnek görülmektedir, ayrıca test sonucunda her bir küme için hata vektörü de gözlenmektedir. Eğitim sonucunda her bir sınıf için hata vektörünün iterasyon boyunca değişimi ise Şekil 2.14’de gözlenmektedir. Buna göre bazı kümelerin ayrışmakta diğerlerine göre nerdeyse 10 kat daha başarılı olduğu görülmektedir.



Şekil 2.14: Ortalama Hata

```
inference error [0.05647427 0.01113423 0.07399704 0.05266247 0.07848166 0.02011675
0.09778474 0.0261776 0.02026124 0.01959329]
8309
```

Şekil 2.15: Hata Vektörü ve Doğru Sınıflandırılan Veri Sayısı

Tablo 3'te ağırlık ilk koşullarının eğitim ve teste ortalama etkileri incelenmiştir.

Tablo 3: Ağırlık İlk Koşullarının Etkisi(C= 0.4, M=0.7 için)

Ağırlık İlk Koşulu	Eğitim Ortalama Hata	Doğruluk Oranı
-0.15 - 0.15	0.038139841193861636	72
-1.5 - 1.5	0.03352144905625841	78
-15 - 15	0.18000003181038882	12

Görüldüğü gibi küçük değerlerden çok daha iyi sonuçlar alınmıştır.

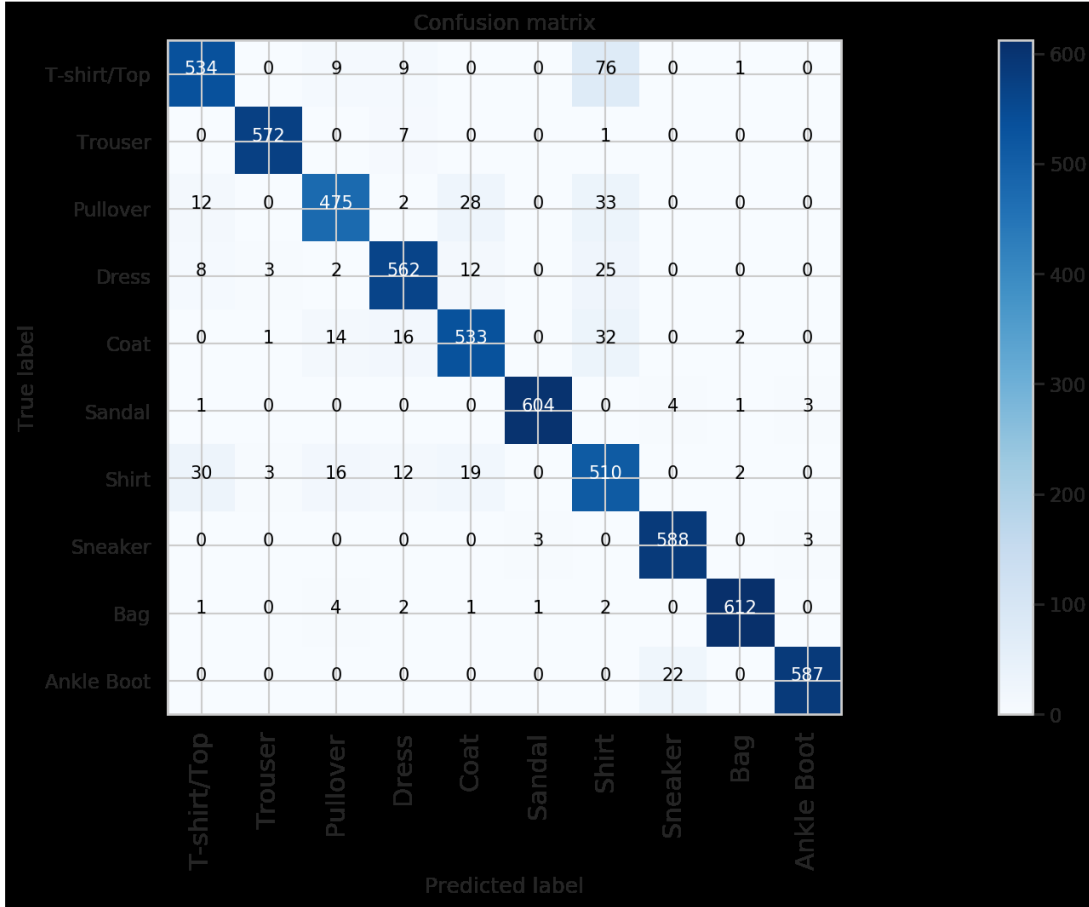
Öğrenim hızının eğitim ve test sürecine etkisi Tablo 4'te görülmektedir. Görüldüğü gibi genel olarak öğrenme hızı arttıkça eğitim süresi azalmış ama aynı zamanda doğruluk oranı da azalmıştır.

Tablo 4: Öğrenme Hızının ve Momentum Teriminin Etkisi(Ağırlıklar -.15/.15)

Momentum	Öğrenme Oranı	Eğitim için geçen süre (dakika)	Doğruluk
0.3	0.1	05:20	8451
0.6	0.4	05:29	7960
0.9	0.7	05:25	1000
0.9	0.9	05:19	1000

2.2.3.2 Confussion Matrix

Confussion Matrix makine öğrenmesinde sonuçların değerlendirilmesi aşamasında kullanılan etkili bir yöntemdir. Kısaca çalışma mantığı her bir veri için satırlarda beklenen değerleri sütunlarda o beklenen değere karşılık hangi değer üretildiğini tutan matristir. Yani matrisin diagonalını takip ettiğimizde istenen değere karşılık doğru değer üretildiğini gözlemleriz. Onun dışındaki dağılım ağırlıkların yanlışları gösterir. Şekil 2.16’da Fashion Mnist için görselleştirilmiş bir confussion matrix görülmektedir.



Şekil 2.16: Confussion Matrix Örneği

<https://www.kaggle.com/fuzzywizard/fashion-mnist-cnn-keras-accuracy-93>

Ağ yapısına sklearn kütüphanesinin hazır fonksiyonunu[21] kullanarak eklenen confusion matrix sonucu %72 başarımla sağlanan bir örnek için Şekil 2.17’de görüldüğü gibidir.

```
7225
[[660  0  8 10  0  0 112  0  0  0]
 [ 5 935  1 18  2  0  2  0  1  0]
 [ 55  0 499 18 205  0 411  0 26  0]
 [135 33 10 822 35  0 104  0  5  0]
 [ 28 30 410 112 698  1 139  0  4  0]
 [  1  0  0  1  0 759  0 11  7 14]
 [ 91  0 58 14 54  0 186  0  3  0]
 [  0  0  0  0  0 169  0 969  7 235]
 [ 23  2 13  5  6  7 46  0 947  1]
 [  2  0  1  0  0 64  0 20  0 750]]
```

Şekil 2.17: Ağ Yapımız için Confussion Matrix Sonucu

2.2.3.3 RELU Aktivasyon Fonksiyonu ve Dropout

İşlem yükünün azlığı ve donanımda gerçekleşmesinin kolay olması gibi sebeplerden dolayı RELU aktivasyon fonksiyonu günümüzde sıkça kullanılmaktadır.

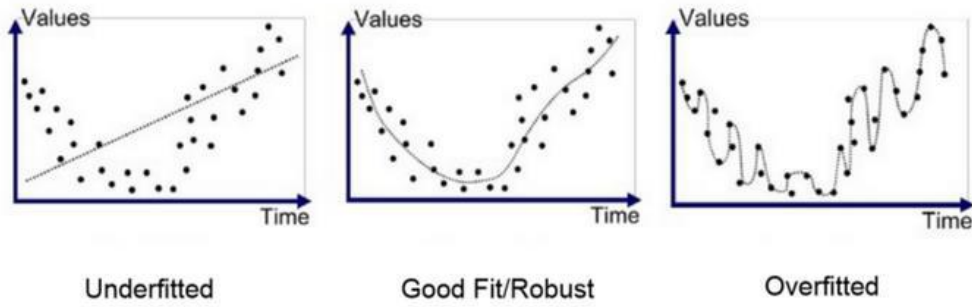
Gerçeklenmesi basitçe $\max(0,x)$ (x nöronun v çıkışı olmak üzere) dayalı olan fonksiyonun kendisinin ve türevinin python ile gerçekleşmesi Şekil 2.18’deki gibidir.

```
#Relu function
def relu(z):
    return np.maximum(0,z)

#Derivative of Relu func.
def d_relu(vec_x):
    vec_y = []
    for i in range(len(vec_x)):
        if vec_x[i] > 0:
            vec_y.append([1])
        else:
            vec_y.append([0])
    return np.array(vec_y)
```

Şekil 2.18: RELU Aktivasyon Fonksiyonu ve Türevi

Dropout fonksiyonu makine öğrenmesinde aşırı öğrenme(overfitting) probleminin baş göstermesiyle 2012 yılında ortaya atılmış bir teoridir. Bu teoriye göre eğitim sırasında her iterasyonda ara katmanlardaki farklı kombinasyondaki hücreler bir sonraki katmana etki etmektedir. Bu rastgelelik ile ağırlıklı verilen kümeyi ezberlemesi değil bir nevi gidişata ayak uydurması sağlanmaktadır. Bir görsel yoluyla tarif etmek gerekirse, az öğrenme, ideal öğrenme ve aşırı öğrenme Şekil 2.19’teki gibi gösterilebilir.



Şekil 2.19: İdeal ve Aşırı Öğrenme Eğrileri

Dropout fonksiyonun belirtilen olasılık oranındaki rastgele nöronların sıfırlanarak gerçekleşmesi Şekil 2.20’de görülmektedir.

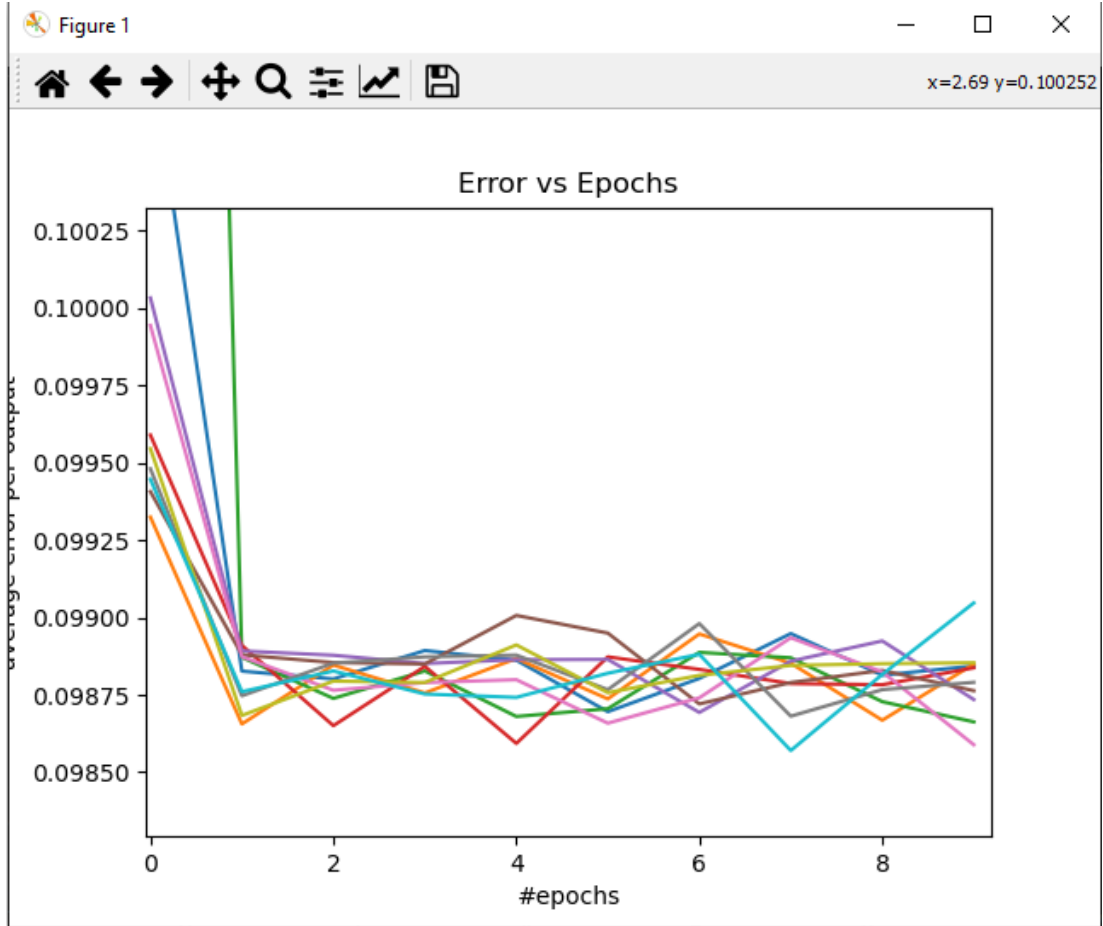
```
#Dropout function
def dropout(X, drop_probability):
    keep_probability = 1 - drop_probability
    mask = np.random.uniform(0, 1.0, X.shape) < keep_probability
    #####
    # Avoid division by 0 when scaling
    #####
    if keep_probability > 0.0:
        scale = (1/keep_probability)
    else:
        scale = 0.0
    return mask * X * scale
```

Şekil 2.20: Dropout Fonksiyonu

Dropout fonksiyonu basitçe RELU çıkışındaki y değerlerinin bu fonksiyona belli bir olasılık değeri ile(denemelerimiz 0.5 değeri ile yapıldı) gönderilmesiyle gerçekleşmektedir. İlk ara katman için fonksiyonlar aşağıdaki gibi kullanılmıştır.

```
f_y = relu(f_v)
f_y = dropout(f_y, dropout_constant)
```

RELU ve Dropout uygulamamız sonucunda eğitimdeki hatanın değişimi Şekil 2.21'de görüldüğü gibi gözlenmiştir.



Şekil 2.21: RELU ve Dropout ile Eğitim Hatasının Değişimi

2.2.3.4 Softmax Aktivasyon Fonksiyonu ve Cross Entropy Loss

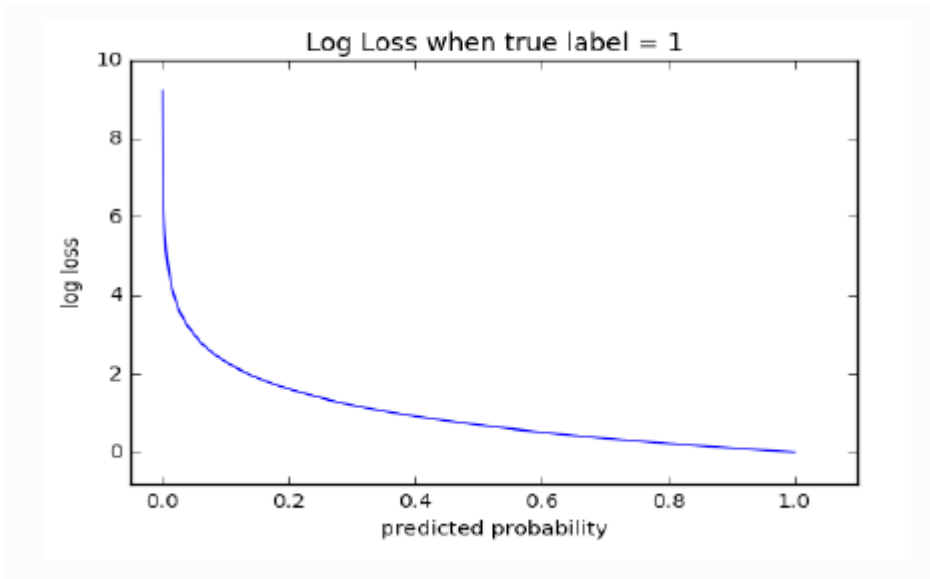
Sinir ağlarında çıkış katmanının aktivasyon fonksiyonu olarak sigmoid fonksiyonu yerine softmax fonksiyonu kullanmak da yaygın bir yöntemdir. Burada giriş vektörünün elemanlarını ayrı ayrı exponansiyel fonksiyonların toplam exponansiyel fonksiyona oranı softmax sonucunu vermektedir. Şekil 2.22'de Softmax fonksiyonunun Python ile gerçekleştirilmesi görülmektedir.

```
#Softmax Function
def softmax(X):
    exps = np.exp(X)
    return exps / np.sum(exps)
```

Şekil 2.22: Softmax Fonksiyonu

Sınıf sayısı 2’den fazla olan uygulamalarda kayıp fonksiyonu olarak Cross Entropy yönteminin kullanılması yine sinir ağlarında yaygın olarak karşımıza çıkmaktadır.[21] Basitçe aşağıda gösterildiği şekilde ifade edilen kayıp fonksiyonu olasılık dağılımına dayanan bir yöntemdir ve Şekil 2.23’de görüldüğü gibi kayıp azaldıkça fonksiyonun doğru çıktı üretme olasılığı artmaktadır.

$$-\sum_{c=1}^M y_{o,c} \log(p_{o,c})$$



Şekil 2.23: Cross Entropy Hata ve Tahmin Değişim Grafiği

https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

[21]’deki formüller temel alarak uygulanan cross entropy geri yayılım yönteminde çıkış katmanın gradyen hesabı basitçe yd değerinden y değeri çıkarılarak elde edilmiştir. Ardından bu gradyen değeri ara katmanların da yerel gradyen hesabında kullanılmıştır. Şekil 2.24’de bu hesabın Python uygulaması görülmektedir.

```

out_sigma = np.subtract(yd[i], out_y)

s_gradient = der_sigmoid(s_v)

s_sigma = np.transpose(out_weights_matrix[:, :s_cell_num]).dot(out_sigma) * s_gradient

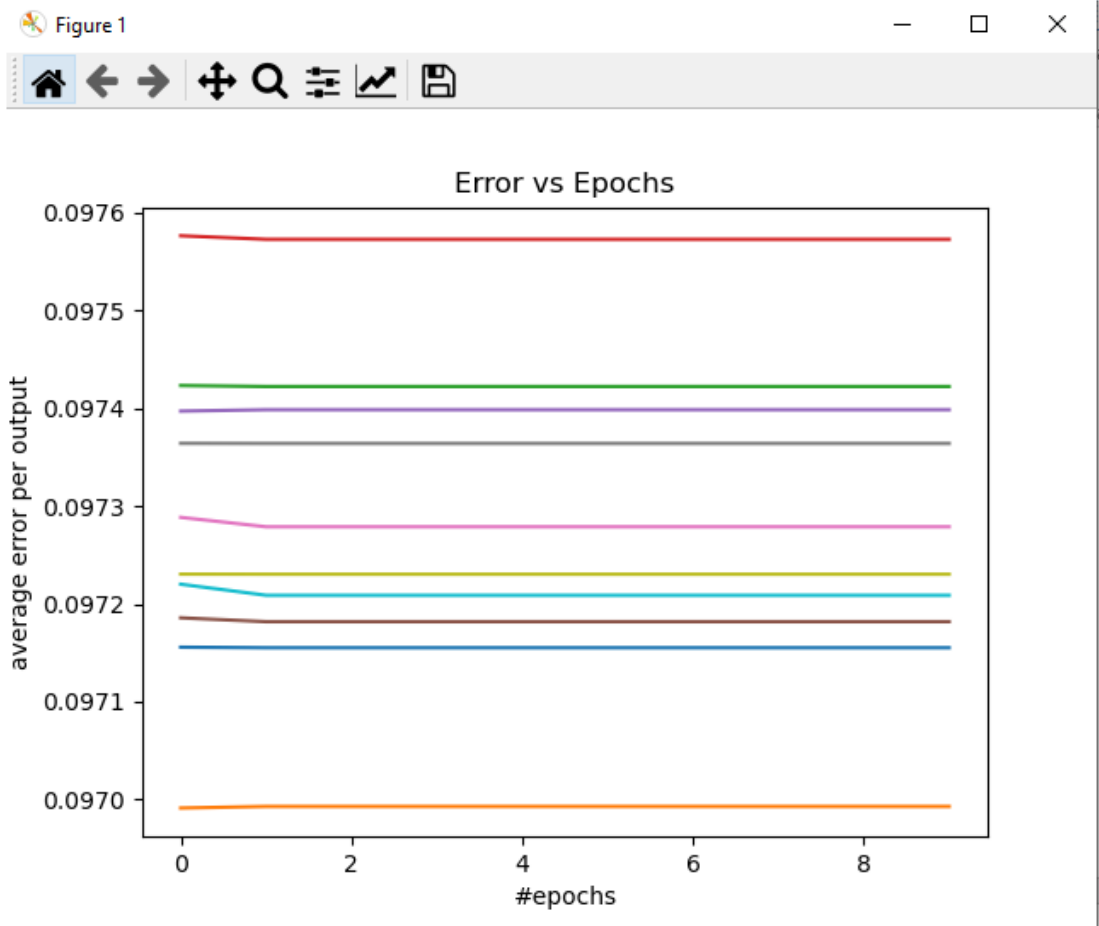
f_gradient = der_sigmoid(f_v)

f_sigma = np.transpose(s_weights_matrix[:, :f_cell_num]).dot(s_sigma) * f_gradient

```

Şekil 2.24: Cross Entropy Gradyen Hesapları

Çıkış katmanında softmax ve Cross Entropy kayıp fonksiyonu kullanılarak eğitilen ağın eğitim kaybı ve iterasyon grafiği Şekil 2.25'deki gibi elde edilmiştir.



Şekil 2.25: Softmax ve Cross Entropy Hata vs İterasyon Grafiği

2.3 MLP Algoritmasının C Gerçeklenmesi

Ek 3.3’de verilen mlp algoritması saf C koduna çevrildi. Bu süreçte hafıza biriminin kullanım miktarı ve kodun hızlı çalışması göz önünde bulundurularak C kodu yazıldı. C kodunda `<stdio.h>`, `<stdlib.h>`, `<math.h>`, `<stdint.h>` ve son olarak `<string.h>` kütüphaneleri kullanıldı. C kodunda doğrulama kümesinin boyutu 100×784 olacak şekilde seçildi. Python tarafında üretilmiş ağırlıklar txt dosyasına dönüştürülüp koda aktarıldı. Sırasıyla ilk katman ağırlıkları 16×784 ’lük ikinci katman ağırlıkları 16×17 ’lik ve çıkış katmanı ağırlıklarını oluşturan matrix 10×17 boyutunda oluşturuldu. Python üzerindeki modelde oluşturulduğu gibi katman katman model tasarlandı. İlk katman için doğrulama kümesinin verileri ilk katman ağırlıklarıyla çarpılıp bir dizi üzerinde tutuldu. Ardından ilk katmanın çıktısına bias terimi eklenip ikinci katmanın ağırlıklarıyla çarpılıp sigmoid fonksiyonuna sokulduktan sonra elde edilen veriler başka bir dizi üzerinde tutuldu. Son katmana geldiğinde önceki adımlara benzer işlemler tekrarlandı. Ardından ekrana çıkış matrixinin en büyük elemanın indeksi ekrana yazıldı. Sırasıyla python ve C üzerinde yazılmış kodların çıktıları şekil 2.26 ve şekil 2.27’de verilmiştir. Çıktılar detaylıca incelendiğinde C tarafında float işlemlerin verebileceği maksimum hassasiyet boyutunda çıktılar python çıktıları ile bire bir eşleşmiştir. Bu sonuçlar eşliğinde elde edilen C kodu Linux ortamında gcc derleyicisinde derlenmiştir. Derleme için kullanılan terminal komutu `gcc isim.c -o isim -lm` komutu ile gerçekleştirilmiştir. `-lm` komutu sayesinde matematiksel kütüphaneler aktive edilmiştir. Gcc derleyicisinin çıktısı şekil 2.28’de verilmiştir. Gcc derleyicisinin çıktısı incelendiğinde python kodunun vermiş olduğu çıktı ile aynı sonuçları hassasiyet sınırları içerisinde vermiştir. C kodunda genel olarak pointer işlemleri yapıldı. Pointer aritmatığı sayesinde hafızada gereksiz kopyalama maliyetlerine neden olabilecek problemlerin

önüne geçildi. Python üzerinde hazırlanan modeldeki adımların bir bir takip edildiği C kodu ek 3.4'de verilmiştir.

```
0.9039729565389941
0.9265558267884457
0.997248228109678
0.9726358168608665
0.5699557834279471
0.9907208744235233
0.8504898287220821
0.1600608626853771
0.5135833341523146
0.5135833341523146
0.9885415189932841
0.8504898287220821
0.7218142994480747
0.862908930104943
0.9718347406539768
0.8504898287220821
0.9897913361402042
0.9265558267907571
0.6597544653042741
0.9987687649337371
```

Şekil 2.26: Python Kodunun Çıktısı

```
C:\Users\caner\Desktop\BİTİRME\Yazîmler\validation_set_100.exe
0. = 0.903973
1. = 0.926556
2. = 0.997248
3. = 0.972636
4. = 0.569956
5. = 0.990721
6. = 0.850490
7. = 0.160061
8. = 0.513583
9. = 0.513583
10. = 0.988542
11. = 0.850490
12. = 0.721814
13. = 0.862909
14. = 0.971835
15. = 0.850490
16. = 0.989791
17. = 0.926556
18. = 0.659755
19. = 0.998769
20. = 0.946761
21. = 0.278129
22. = 0.526403
23. = 0.963575
24. = 0.963575
25. = 0.993450
26. = 0.926556
27. = 0.659755
28. = 0.470424
29. = 0.903973
```

Şekil 2.27: C Kodunun Çıktısı

```
caner@caner: ~/Documents/pulpino/anka_utils
caner@caner:~/Documents/pulpino/anka_utils$ gcc validation_set_100.c -o validation_set_100 -lm
caner@caner:~/Documents/pulpino/anka_utils$ ./validation_set_1000. = 0.903973
1. = 0.926556
2. = 0.997248
3. = 0.972636
4. = 0.569956
5. = 0.990721
6. = 0.850490
7. = 0.160061
8. = 0.513583
9. = 0.513583
10. = 0.988542
11. = 0.850490
12. = 0.721814
13. = 0.862909
14. = 0.971835
15. = 0.850490
16. = 0.989791
17. = 0.926556
18. = 0.659755
19. = 0.998769
20. = 0.946761
21. = 0.278129
22. = 0.526403
23. = 0.963575
24. = 0.963575
25. = 0.993450
26. = 0.926556
27. = 0.659755
28. = 0.470424
29. = 0.903973
30. = 0.391226
31. = 0.996042
32. = 0.993193
```

Şekil 2.28: Gcc Derleyicisi Çıktısı

3. RISC-V VE MLP MODELİNİN FPGA UYGULAMALARI

3.1 Uygulama Kodunu Fpga Kartına Yüklenir Hale Getirmek

Pulpino ortamının test edilen ve tavsiye edilen gereksinimleri indirilir:

Vivado 2015.1
Cmake > 2.6
GCC > 5.2
Python > 2.7
Riscv gnu toolchain

İlk olarak projede uygulama kodunu kolayca değiştirerek yüklemek için bootloader uygulaması yapılmasına karar verildi. Boot kodu projede sağlanan script'ler ile boot_code.sv dosyasına dönüştürülmeli. Github'dan elde edilen projeye FPGA uygulamasını yapabilmek için top modül ve clock manager eklendi. Ardından kullanılacak kartın constraint dosyaları projete eklendi. Sonrasında eklenen tasarım dosyaları fpga/pulpino/tcl/src_files.tcl dosyasına Şekil 3.1'de görüldüğü gibi eklenmeli.

```
# pulpino
set SRC_PULPINO " \
  $RTL/axi2apb_wrap.sv \
  $RTL/periph_bus_wrap.sv \
  $RTL/core2axi_wrap.sv \
  $RTL/axi_node_intf_wrap.sv \
  $RTL/axi_spi_slave_wrap.sv \
  $RTL/axi_slice_wrap.sv \
  $RTL/axi_mem_if_SP_wrap.sv \
  $RTL/core_region.sv \
  $RTL/instr_ram_wrap.sv \
  $RTL/sp_ram_wrap.sv \
  $RTL/boot_code.sv \
  $RTL/boot_rom_wrap.sv \
  $RTL/peripherals.sv \
  $RTL/ram_mux.sv \
  $RTL/pulpino_top.sv \
  $RTL/clock_gen.sv \
  $FPGA_RTL/pulpino_wrap.v \
  $FPGA_RTL/PULP_TOP.v \
  $FPGA_RTL/clock_wiz_0.v \
  $FPGA_RTL/clock_wiz_0_clock_wiz.v \
  $FPGA_RTL/main_clock_generator.v \
"
```

Şekil 3.1: FPGA Uygulaması için Eklenen Tasarım Dosyaları

Aynı dizin içerisinde run.tcl dosyası kartın özellikleri doğrultusunda güncellenmeli. Şekil 3.2'de bu ayarlar görülmektedir.

```

if { ![info exists ::env(BOARD) ]} {
    #set ::env(BOARD) "zedboard"
    #set ::env(BOARD) "neyxs4DDR"
}

if { ![info exists ::env(XILINX_PART) ] } {
    #set ::env(XILINX_PART) "xc7z020c1g484-1"
    set ::env(XILINX_PART) "xc7a100tcsq324-1"
}

if { ![info exists ::env(XILINX_BOARD) ] } {
    #set ::env(XILINX_BOARD) "em.avnet.com:zynq:zed:c"
}

```

Şekil 3.2: FPGA Kartının Projeye Tanıtılması

Ardından pulpino/fpga dizini içerisinde terminal açılarak “make” komutu çalıştırılıp sentez aşaması tamamlandıktan sonra implementasyon aşaması için vivado açılıp implementasyon aşaması tamamlanıp bitstream oluşturulmalı. Gerekli olan Xilinx Nexys4 fpga kartı vivadoya eklenir. Ardından vivado üzerinden hardware manager kısmında “auto connect” butonuna basarak fpga kartı vivadoya tanıtılır. Ardından “Program device” butonu ile bitstream dosyası pulpino/fpga/pulpino/pulpino.runs/impl_1 dosyası altında seçilerek fpga kartına yüklenir.

Öncelikle yazılan “bootloader” kodu ile işlemci en başta bu boot kodu için derlenmiş proje olarak fpga kartına yüklenecek ve flash’a yazılacak kodu alıp kendi hafızasındaki kod bölmesine ekleyerek çalıştırmayı bekliyor. Flash’a yazılacak kod ise asıl uygulama kodu olup pulpino için hazırlanan sürücüler kullanılarak yazılıyor.

Yazılan uygulama koduna derleyici tarafından uygulanacak olan optimizasyon seviyesini istenilen şekilde ayarlamak için pulpino/sw/build dizini içerisindeki “cmake_configure.riscvfloat.gcc” dosyasının içerisine girilerek TARGET_C_FLAGS’ın bulunduğu satırda tırnak içerisinde optimizasyon seviyesi değiştirilir. O0 optimizasyonun hiç uygulanmaması anlamına gelirken O3 en fazla optimizasyonun uygulanacağı anlamına gelir. Os olarak ayarlamak ise boyut anlamında optimizasyon uygulanacağı anlamına gelir. Aynı satırda “-Wl, -Map= isim.map” komutu optimizasyon seviyesinden sonra virgül konularak yazılırsa pulpino/sw/build/apps/helloworld dizininde hafıza map dosyası oluşturulur ve hangi adreste hangi değişkenlerin yazıldığı görülebilir. İlgili kısım ilgili kod **şekil 3.3’de** verilmiştir.

```
TARGET_C_FLAGS="-O3 -m32 -g -fsingle-precision-constant -mfpdouble=float -Werror=double-promotion -Wl,-Map=out.map"
```

Şekil 3.3: Optimizasyon Seviyesinin Ayarlanması

Ardından uygulama kodu sağlanan script’ler ile slm dosyasına çevrilir ve vivado’nun flash komutları ile flash hafıza bölmesine yüklenir. Gerekli scriptler Şekil 3.4’te verilmiştir.

```

create_hw_cfgmem -hw_device [lindex [get_hw_devices] 0] -mem_dev [lindex [get_cfgmem_parts {s25fl128sxxxxx0-spi-x1_x2_x4}] 0]
set_property PROGRAM.BLANK_CHECK 0 [ get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]]
set_property PROGRAM.ERASE 1 [ get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]]
set_property PROGRAM.CFG_PROGRAM 1 [ get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]]
set_property PROGRAM.VERIFY 1 [ get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]]
refresh_hw_device [lindex [get_hw_devices] 0]

set FLASH_IMAGE "/home/altarkoca/Documents/bitirme/pulpino/fpga/pulpino/pulpino.runs/s25fl128s.bin"
set BITFILE /home/altarkoca/Documents/bitirme/pulpino/fpga/pulpino/pulpino.runs/impl_1/FULP_TOP.bit

set_property PROGRAM.ADDRESS_RANGE {use_file} [ get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]]
set_property PROGRAM.FILES {list $FLASH_IMAGE } [ get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]]
set_property PROGRAM.UNUSED_PIN_TERMINATION {pull-none} [ get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]]
set_property PROGRAM.BLANK_CHECK 0 [ get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]]
set_property PROGRAM.ERASE 1 [ get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]]
set_property PROGRAM.CFG_PROGRAM 1 [ get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]]
set_property PROGRAM.VERIFY 1 [ get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0 ]]
startgroup
if {[string equal [get_property PROGRAM.HW_CFGMEM_TYPE [lindex [get_hw_devices] 0]] [get_property MEM_TYPE [get_property CFGMEM_PART [get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0]]]} {
program_hw_cfgmem -hw_cfgmem [get_property PROGRAM.HW_CFGMEM [lindex [get_hw_devices] 0]]
endgroup

set_property PROBES.FILE {} [lindex [get_hw_devices] 0]
set_property PROGRAM.FILE {/home/altarkoca/Documents/bitirme/pulpino/fpga/pulpino/pulpino.runs/impl_1/FULP_TOP.bit} [lindex [get_hw_devices] 0]
program_hw_devices [lindex [get_hw_devices] 0]
refresh_hw_device [lindex [get_hw_devices] 0]

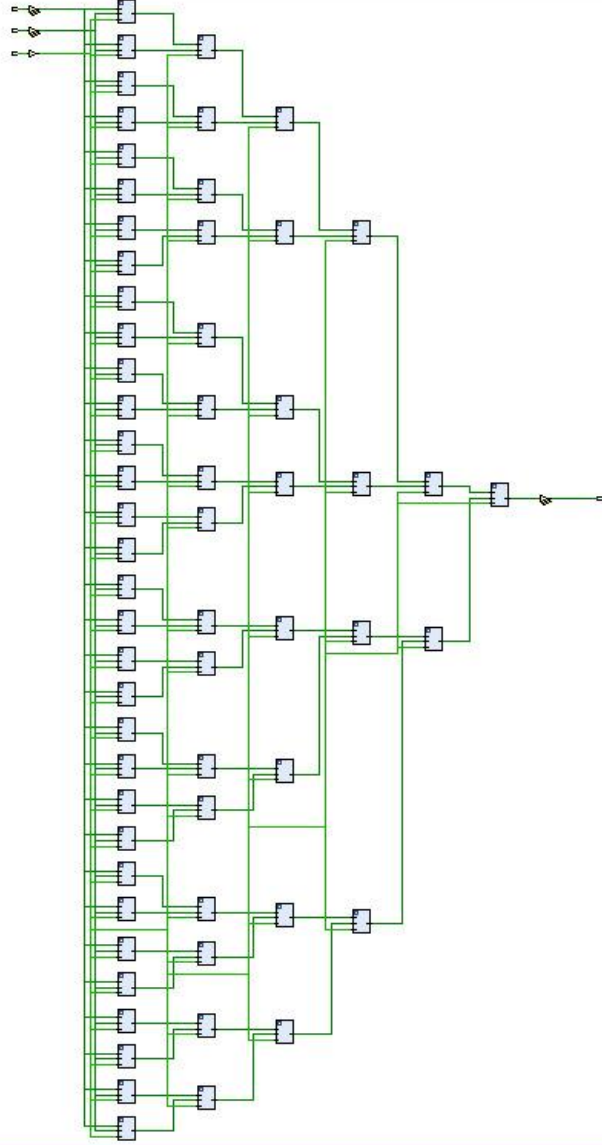
```

Şekil 3.4: Vivado Flash Scriptleri

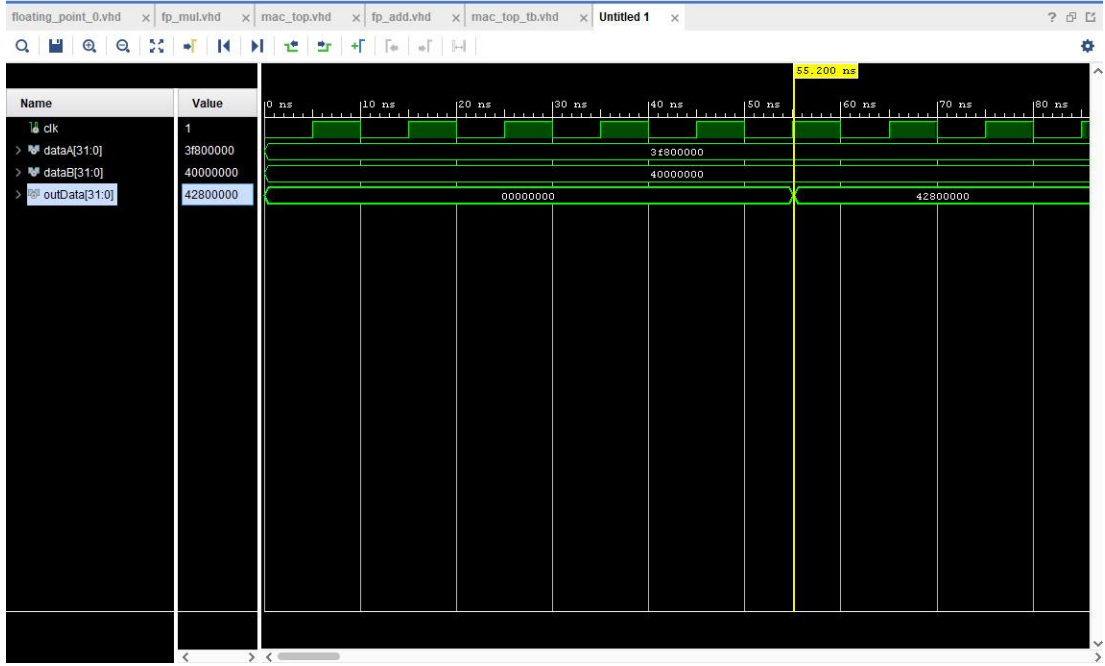
3.2 MAC Ünitesi

Projede kullanılan MAC(Çarpıcı-toplayıcı) ünite üzerine çalışmalar yapıldı. MAC ünite birçok yapı tarafından kullanılacağı için performansı bütün sistem açısından çok kritik öneme sahiptir. Ayrıca tasarımda ince noktalara sahiptir. Bu noktada paralel çarpıcı model üzerine çalışmalar yapıldı. Bu tasarımın en önemli özelliklerinden biri pipeline sistemi üzerine inşa edilmesidir. MAC ünite modelinin VHDL şematığı Şekil 3.5’de verilmiştir.

Buradaki yapı toplama ağacı şeklinde olmaktadır. Sistem 32 çarpıcı olacak şekilde tasarlandı. İki adet 32 bit girişi çarpıp 5 adımda toplayarak sonuca ulaşılmaktadır. Eğer giriş boyutu 32 bitten daha küçük olur ise kalan çarpıcılara giriş olarak 0 biti verilir. Sistem pipeline temelli olduğu için her bir saat döngüsünde bir çıktı verir. Şekil 3.6’de sisteme 2 adet 32 bit giriş verildiği ve 6 saat döngüsünde çıktı üretebildiği görülmektedir. Şekil 3.3’de verilen simülasyon çıktısında da görüldüğü üzere Şekil 3.5’de ki her bir adım 1 saat döngüsünde başarılı bir şekilde gerçekleşmektedir.

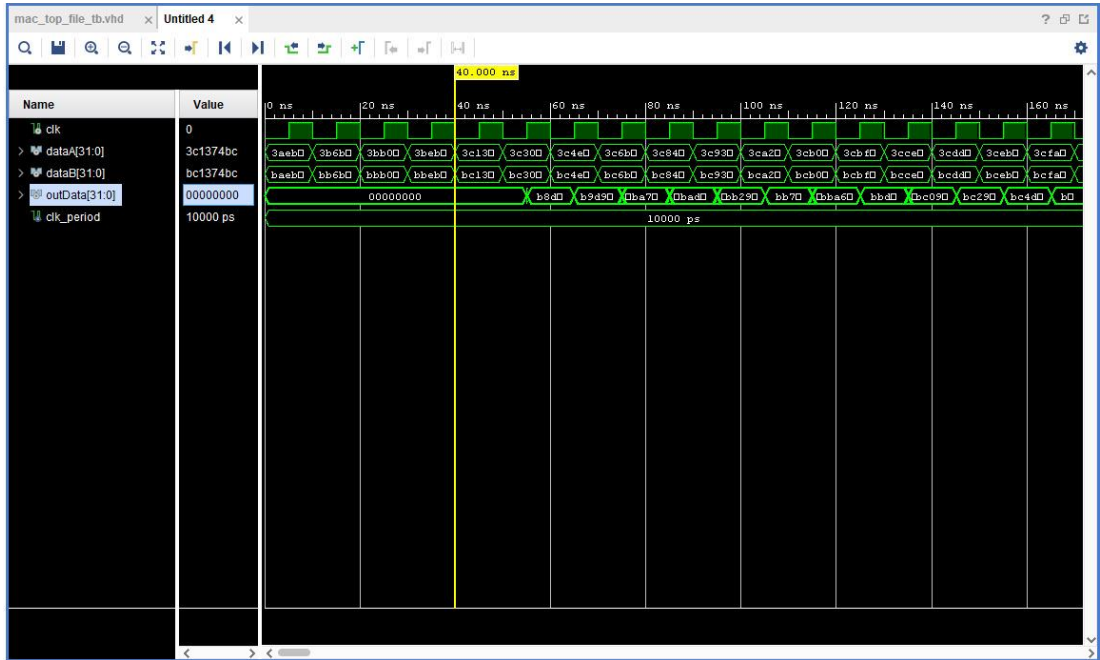


Şekil 3.5: MAC Unite VHDL Şematik



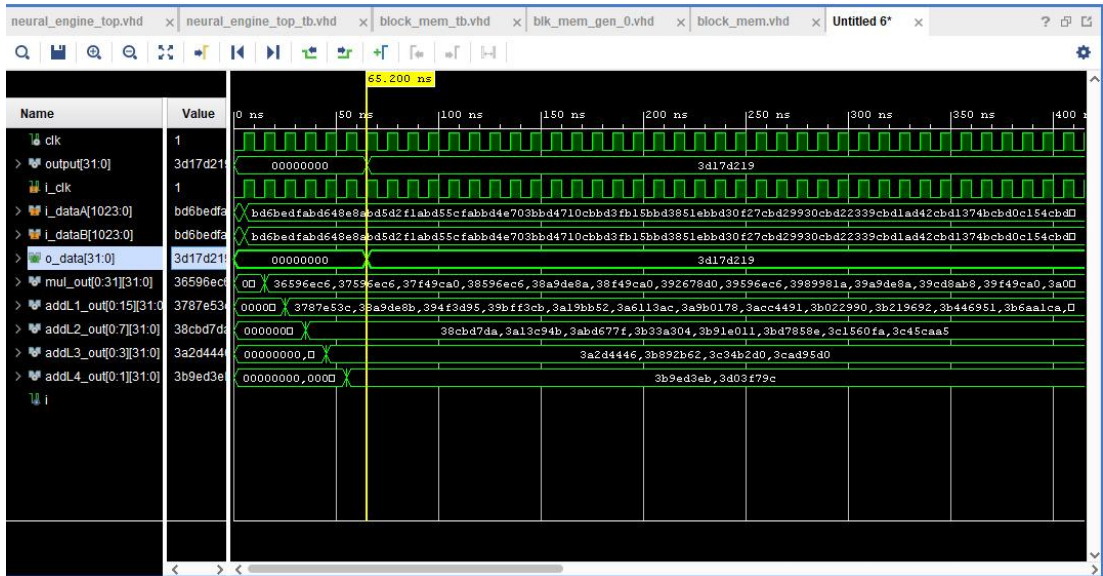
Şekil 3.6: MAC Ünite Simülasyon Çıktısı(2 adet 32 bit giriş ile)

Aynı model kullanılarak 1000 adet 32 bit ile yapılan testler sonucunda gerekli 6 saat döngüsünün ardından her bir saat döngüsünde başarılı bir şekilde çıkış alına bildiği görülmektedir. Bu başarıya pipeline sisteminin düzgün bir şekilde oturtulması sayesinde ulaşıldı. Bu simülasyonun çıktısı Şekil 3.7’te verilmiştir.



Şekil 3.7: MAC Ünite Simülasyon Çıktısı(1000 adet giriş için)

Bu projede performansı en çok etkileyecek meselelerden bir tanesi de verinin bellekten alınıp işlenip tekrar belleğe yazılması işlemidir. Burada ana bellek bloğundan verinin alınıp işlenip tekrar ana bellek bloğuna döndürülmesi performans açısından epey maliyetli bir sonuç doğuracaktır. Bunun yerine scratchpad bellek alanı bir diğer adıyla yerel depolama oluşturularak kaybedilme ihtimali olan performans kaybının önüne geçilecektir. Bunun etkisi şöyle olacaktır, tüm işlemler sonucunda elde edilecek sonuç çok hızlı bir şekilde bellek bloğuna yazılabilecek ve o yerel bellek bloğundan çok hızlı bir şekilde veriler tekrardan geri çekilecektir. Şekil 3.8’te 1024 bit girişi ve 32 bit çıkışı olan scratchpad bellek bloğu simülasyonu verilmiştir. 6 saat döngüsünde çıkışı verebilmektedir.



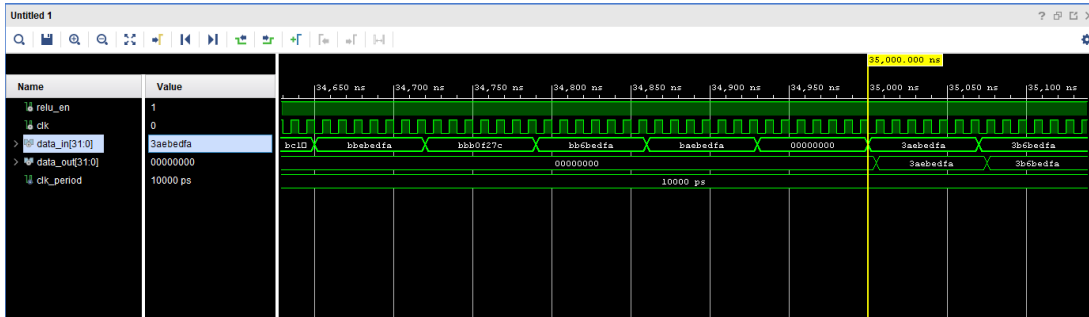
Şekil 3.8: Strachpad Bellek Bloğu Simülasyon Çıktısı

3.3 Aktivasyon Fonksiyonu

Yapay sinir ağlarında aktivasyon fonksiyonları, doğrusal olmayan dış dünya verilerini kullanmamızı sağlar. Doğrusal olmayan durumlar da yapay sinir ağıımız tarafından aktivasyon fonksiyonu sayesinde öğrenilebilir hale gelir. En sık kullanılan aktivasyon fonksiyonları relu, sigmoid ve tanh aktivasyon fonksiyonlarıdır. Öğrenmenin önemli bir katmanı olan aktivasyon katmanı projemizde bu üç aktivasyon fonksiyonunu içermektedir. Aktivasyon fonksiyonları VHDL dilinde Vivado programında donanım olarak tasarlanacaktır.

3.3.1 Relu

RELU(Rectified Linear Unit) fonksiyonu negatif girişler için 0 çıkışı üretirken, pozitif girişler için doğrusal bir çıktı vermektedir. Relu aktivasyon fonksiyonu tasarımının analizleri Vivado’da yapıldı. 1000 adet giriş verilerek yapılan analizde beklenen çıktılar elde edildi. Relu fonksiyonuna göre negatif girişler için 0 çıkışı, pozitif girişler için girişin aslı çıkış olarak üretildi. Analizde girişlerle çıkışlar arasında gecikme olmamaktadır. Şekil 3.9’da simülasyon çıktıları görülmektedir.

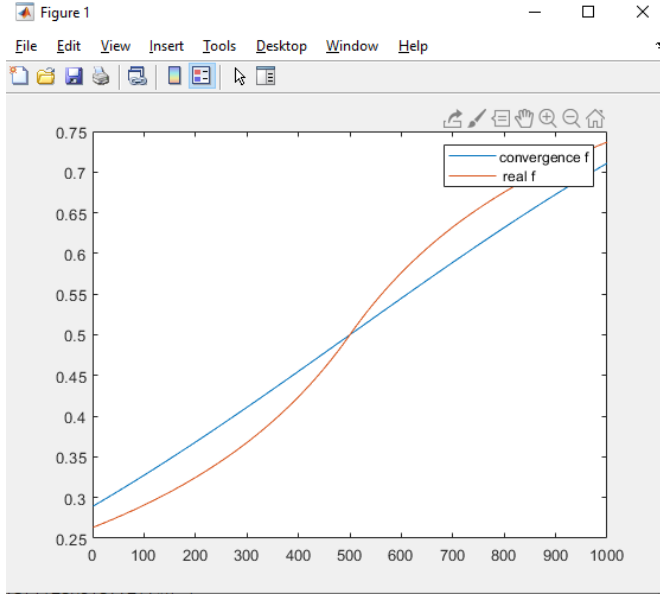


Şekil 3.9: RELU Aktivasyon Fonksiyonu Simülasyonu

3.3.2 Sigmoid

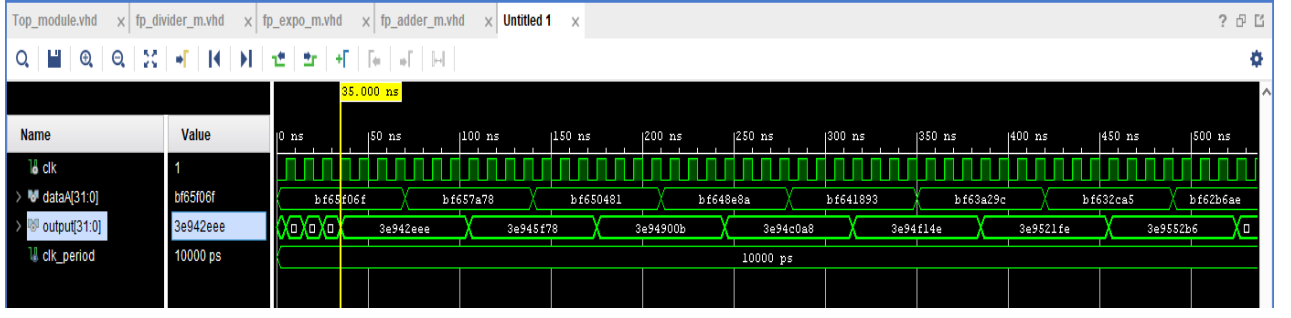
Sigmoid fonksiyonu $(-\infty, \infty)$ aralığında deęer olarak $[0,1]$ aralığında çıkış üretmektedir. Doğrusal bir fonksiyon olmaması, gerçek hayatta verilerin çoęunlukla doğrusal olmayışından, yapay sinir aęlarında popüler olmasını sağlamaktadır. Öncelikle, donanımda sigmoid fonksiyonuna yakınsadığı düşünölen $f(x)$ fonksiyonu geręeklenmiş ancak beklenen yakınsama Şekil 3.10'de göröldüğü gibi geręekleşmemiştir.

$$f(x) = \frac{1}{2} \cdot \left[\frac{x}{1 + \text{abs}(x)} + 1 \right]$$



Şekil 3.10: Geręek Sigmoid-Yakınsama Fonksiyonu Grafięi

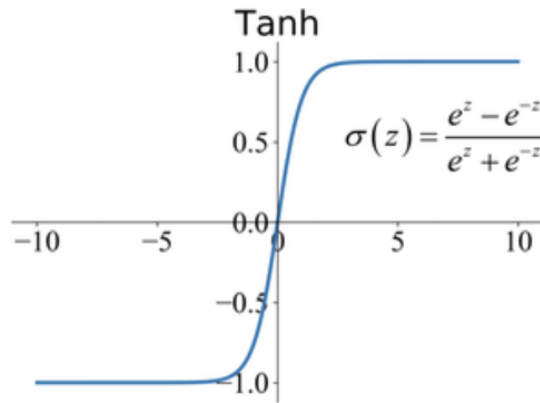
Şekil 3.11'de görölen sigmoid fonksiyonu ifadesi 32-bit floating point IP kullanılarak Vivado'da geręeklenmiştir. Simölasyonda giriş deęerlerine karřılık 3 saat döngüsü sonunda çıkış deęeri alındı. $(-1,1)$ arasında 1000 deęerden oluřan deęer kümesi giriş olarak verildięinde, Şekil 3.11'de göröldüğü gibi çıktıları doęru şekilde elde edildi.



Şekil 3.11: Sigmoid Fonksiyonu Simülasyon Sonuçları

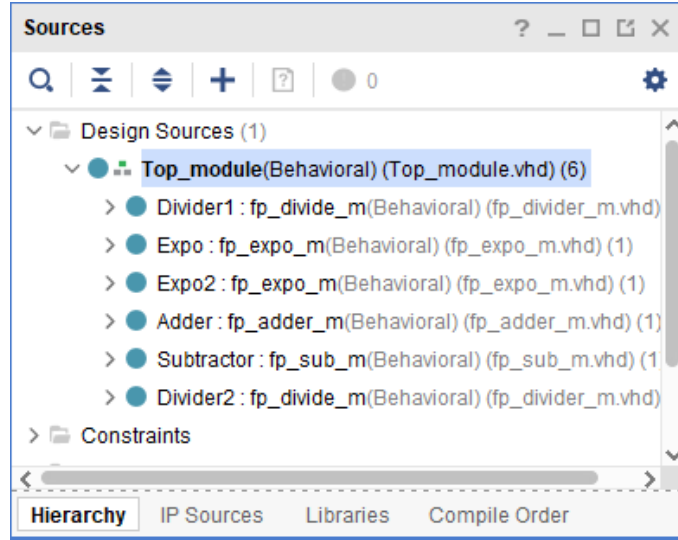
3.3.3 Tanh

Sıklıkla kullanılan bir diğer aktivasyon fonksiyonu da Tanh'dır. Şekil 3.12'da fonksiyonun ifadesi görülmektedir. Sigmoid fonksiyonuna çok benzemekle birlikte aralığı sigmoid'den farklıdır. (-1,1) aralığında değer alır ve türevi sigmoid'e göre daha diktir. Böylece daha çok değer alarak öğrenmenin daha verimli olmasını sağlamaktadır.



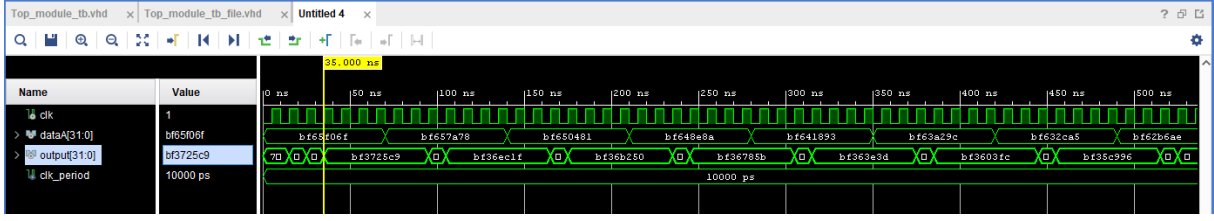
Şekil 3.12: Tanh Aktivasyon Fonksiyonu

Tanh doğrusal olmayan bir fonksiyondur ve üstel ifadeler içermektedir. Fonksiyonun donanımda gerçekleştirilmesi için 32-bit floating point IP'ler kullanılmıştır. Şekil 3.13'de görüldüğü gibi her bir ifadenin oluşturulması, gereken işlemlerin modül olarak tasarlanıp birbirine bağlanarak elde edilmesiyle sağlanmıştır. Böylece Top_module hiyerarşisi oluşturulmuştur. Her işlem için modüller tasarlanmıştır. Kullanılan modüller toplama modülü, çıkarma modülü, üstel ifade modülü ve bölme modülüdür.



Şekil 3.13: Top_module Hiyerarşisi

Simülasyonda verilen 32-bit giriş için alınan çıkışlar analiz edilmiş ve 3 saat döngüsü gecikmeyle doğru sonuç elde edilmiştir. (-1,1) aralığında oluşturulan 1000 değer in sırayla giriş olarak verildiği Şekil 3.14'de simülasyon sonuçları gözlenebilir.



Şekil 3.14: Tanh Simülasyon Sonuçları

3.4 Riscv İşlemcisi İle Fpga Kartı Üzerinde Uart Uygulamaları

3.4.1 Yazı Uygulaması

İlk olarak şekil 3.15’de verilen c kodu yazıldı. Bu kod üzerinde “uart_set_cfg” fonksiyonu yardımıyla uart üzerinden haberleşilecek baud rate ayarlanır. Yazılan kod pulpino/anka_utils dizinine kopyalanır ve konsol açılarak “sudo ./createImage uart_test.c” komutu yardımıyla uygulama kodu flash a yüklenecek bin dosyası haline getirilir. Bu aşamadan sonra 3.1’de bahsedilen adımlar takip edilerek uygulama kodu riscv pulpino işlemcisi üzerine yüklenir. Ve başarılı bir şekilde seri monitör üzerinde uart üzerinden aktarılan veri elde edilir.

```
#include <stdio.h>
#include <uart.h>
#include <utils.h>
#include <pulpino.h>

int main()
{
    uart_set_cfg(0, 10);
    printf("UART DENEMESİ");
    return 0;
}
```

Şekil 3.15: UART Üzerinden Yazı Gönderilmesi

3.4.2 Tam Sayı Uygulaması

Şekil 3.16’da verilen uygulama kodu yazıldı. Bu kod içerisinde 2 farklı tam sayı değişkeni tanımlanarak çarpılıp yeni tanımlanan bir tam sayı değişkenine yazılır. Bu

tam sayı deęiřkeni de UART üzerinden seri monitöre yazılır. Bu kod ilk olarak anka_utils dizininde “createImage” komutu yardımıyla bin dosyasına dönüřtürülür ve fpga kartına yüklenicek řekli alır.

```
#include <gpio.h>
#include <uart.h>
#include <utils.h>
#include <pulpino.h>
#include <stdio.h>

int main()
{
    uart_set_cfg(0, 10);
    int var1 = 2;
    int var2 = 3;
    int var3 = var1 *var2;

    printf("%d\n", var3);
    return 0;
}
```

řekil 3.16: UART Üzerinden Tam Sayı İşlem Sonucu Görüntüleme

3.5 Riscv İşlemcisi İle Fpga Kartı Üzerinde Float İşlem Uygulaması

Bu aşamada ilk olarak bölüm 2.7’de anlatılan řekilde float işlem ünitesi aktif hale getirilir. UART üzerinden float veri aktarılamadıęı için sonucun doęru řekilde çalıştıęı yazılan led yakma kodu ile kontrol edilir. Bu uygulama kodu řekil 3.17’de verildięi řekilde yazılır. Bu kod içerięinde 2 adet float sayı tanımlanıp çarpım işlemi gerçekleştirilerek yeni tanımlanan bir float deęiřkende tutulur. Ardından bu deęiřkenin doęru olması durumunda 8 led yanlıř olması durumunda 3 led yakılır. Ardından anka_utils dizininde bulunan uygulama kodu “createImage” yardımıyla fpga kartına yüklenecek bin dosyası haline dönüřtürülür ve pulpino işlemcisine yüklenir. Yapılan testler sonucunda bu işlem sonunda 8 led yakarak doęru sonuca ulařıldı görülür.

```

#include <gpio.h>
#include <uart.h>
#include <utils.h>
#include <math.h>
#include <pulpino.h>
#include <stdio.h>

void altk_delay(int input)
{
    int x = 0;
    while(x++ < input) asm volatile("nop");
}

void writeGpio(int input)
{
    *(volatile int*) (GPIO_REG_PADOUT) = input;
}

int main()
{
    float var1 = 2.7;
    float var2 = 3.3;
    float var3 = var1* var2;

    if(var3 == 8.91 ){
        writeGpio(0xFF);
        altk_delay(10000000);}
    else{
        writeGpio(0x07);
        altk_delay(10000000);}

    return 0;
}

```

Şekil 3.17: Led ile Float İşlem Kontrolü

3.6 Yapay Zeka Modelinin Soft Float İle FPGA Üzerinde Gerçeklenmesi

Yapay zeka modelinin boyutu pulpino işlemcisinin boyutu göz önünde bulundurulduğunda yaklaşık 70kB kadar büyük gelir. Bundan dolayı dataram kısmını genişletme işlemi gerçekleştirilir. Bu genişletme işlemi iki kısımdan oluşmaktadır. İlk kısım donanım kısmında dataram alanını genişletmek ikinci kısım yazılım seviyesinde dataramı genişletmektir. Donanım seviyesinde vivado ile pulpino projesi açıldığında PULP_TOP/pulpino_platform/pulpino_i/core_region_i modülü içerisindeki "DATA_RAM_SIZE" değişkeninin değeri byte cinsinden 80000 olarak tanımlanır. Bununla ilgili görsel şekil 3.18'de yer almaktadır.


```

12`include "axi_bus.sv"
13`include "config.sv"
14
15module core_region
16#(
17    parameter AXI_ADDR_WIDTH      = 32,
18    parameter AXI_DATA_WIDTH      = 64,
19    parameter AXI_ID_MASTER_WIDTH = 10,
20    parameter AXI_ID_SLAVE_WIDTH  = 10,
21    parameter AXI_USER_WIDTH      = 0,
22    parameter DATA_RAM_SIZE      = 80000, // in bytes
23    parameter INSTR_RAM_SIZE      = 32768, // in bytes 32768
24    parameter USE_ZERO_RISCY      = 0,
25    parameter RISCY_RV32F         = 0,
26    parameter ZERO_RV32M          = 1,
27    parameter ZERO_RV32E         = 0
28)

```

Şekil 3.18: Donanımsal Olarak Dataram Boyutunu Değiştirme

Bununla beraber FPGA için Xilinx Block Ram IP'si kullanıldığı için bu IP'nin boyu script yardımıyla değiştirilmelidir(fpga/ips/xilinx_mem_x/tcl/run.tcl). Şekil 3.19'da ilgili kısım gösterilmiştir.

```

if { ![info exists ::env(XILINX_PART)] } {
    set ::env(XILINX_PART) "xc7z020c1g484-1"
}

if { ![info exists ::env(XILINX_BOARD)] } {
    set ::env(XILINX_BOARD) "em.avnet.com:zynq:zed:c"
}

set partNumber $::env(XILINX_PART)
set boardName $::env(XILINX_BOARD)

create_project xilinx_mem_32768x32 . -part $partNumber
set_property board $boardName [current_project]
create_ip -name blk_mem_gen -vendor xilinx.com -library ip -module_name xilinx_mem_32768x32
set_property -dict {list CONFIG.Memory_Type {Single Port RAM} CONFIG.Use_Byte_Write_Enable {true} CONFIG.Byte_Size {8} CONFIG.Write_Width_A {32}
CONFIG.Write_Depth_A {32768} CONFIG.Register_PortA_Output_of_Memory_Primitives {false} CONFIG.Use_RSTA_Pin {true}} [get_ips xilinx_mem_32768x32]
generate_target all [get_files ./xilinx_mem_32768x32.srcs/sources_1/ip/xilinx_mem_32768x32/xilinx_mem_32768x32.xci]
create_ip_run [get_files -of_objects [get_fileset sources_1] ./xilinx_mem_32768x32.srcs/sources_1/ip/xilinx_mem_32768x32/xilinx_mem_32768x32.xci]
launch_run -jobs 8 xilinx_mem_32768x32_synth_1
wait_on_run xilinx_mem_32768x32_synth_1

```

Şekil 3.19: Block Ram Boyutunu Büyütmek için Değiştirilmesi Gereken TCL

Ardından rtl dosyası içindeki sp_ram_wrap.sv dosyası içine
`include"/projeYolu/pulpino/fpga/ips/xilinx_mem_32768x32/ip/xilinx_mem_32768x32_stub.v"
eklenerek olası bir sentez probleminde
read_checkpoint xilinx_mem_32768x32.dcp
komutu ile tasarım dosyası vivado'ya tanıtılmalıdır.

Yine donanım tarafında herhangi bir hafıza boyutu ya da yeri değiştirilmek istenildiğinde Pulpino hafıza hiyerarşisine dikkat edilmelidir. Şekil 3.20'de görülen Datasheet'te verilmiş olan hafıza yapısında ROM'un yeri yanlış gösterilmiş olup proje içinde 0x00008000 komuna yerleştirilmiştir.

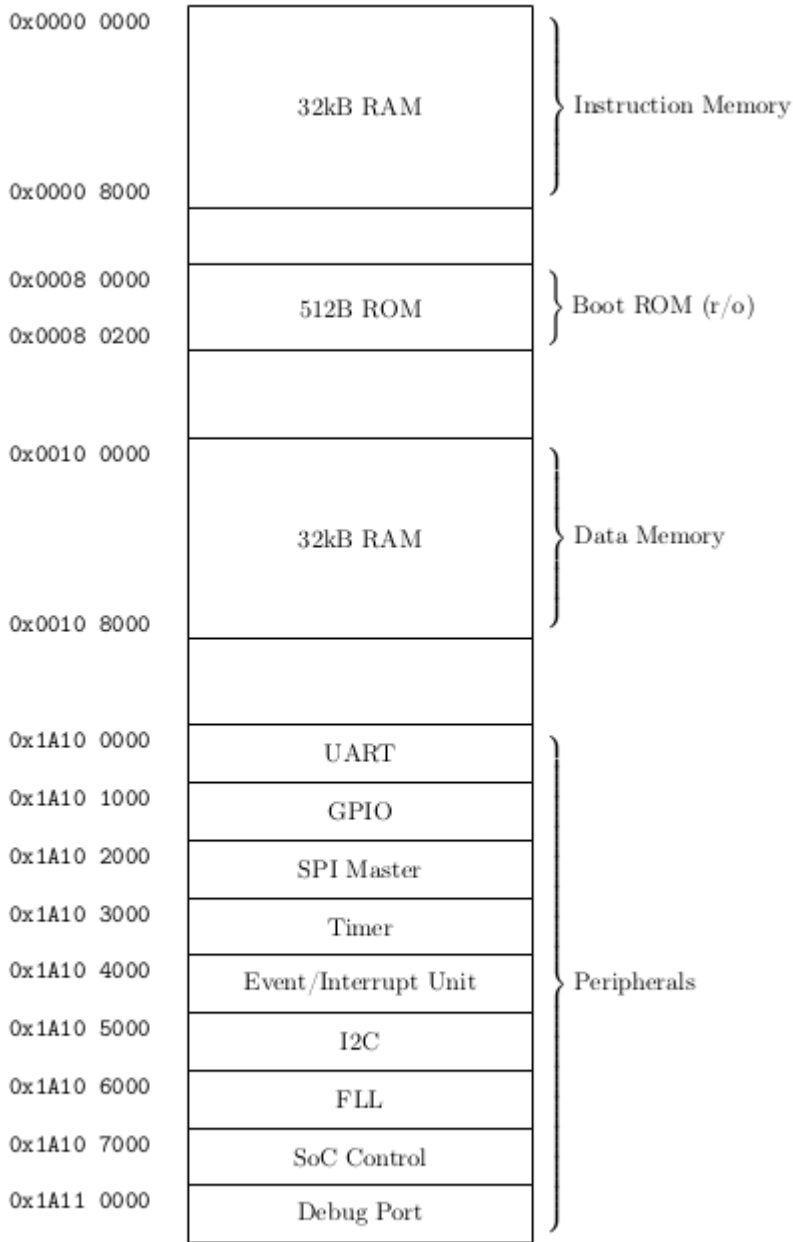


Figure 2.1: PULPINO memory map.

Şekil 3.20: Pulpino Hafıza Hiyerarşisi(Datasheet'ten alınmıştır.)

Yazılımsal olarak dataram alanını genişletmek için öncelikle pulpino/sw/ref dizini içerisinde bulunan “link.common.ld” linker dosyası içerisindeki dataram alanı bitişi ve boyutu ve stack’in başlangıcı uygun biçimde ayarlanır. Aynı dizin içerisindeki “link.boot.ld” dosyası içerisinde de stack’in başlangıcı uygun biçimde ayarlanır. Son olarak pulpino/sw/utills dizini içerisindeki “s19toslm.py” dosyası açılır ve “tcdm_bank_size” değişkeni istenilen boyutlarda uygun biçimde değiştirilir. Yazılımsal olarak dataram kısmının genişletilmesiyle ilgili gerekli görseller sırasıyla şekil 3.21, 3.22 ve 3.23’de verilmiştir.

```

MEMORY
{
    instram      : ORIGIN = 0x00000000, LENGTH = 0x8000
    dataram      : ORIGIN = 0x00100000, LENGTH = 0x1E000
    stack        : ORIGIN = 0x0011E000, LENGTH = 0x2000
}

```

Şekil 3.21: link.common.ld Dosyası İçerisinde Değişim

```

MEMORY
{
    rom          : ORIGIN = 0x00008000, LENGTH = 0x2000
    stack        : ORIGIN = 0x0011E000, LENGTH = 0x2000
}

```

Şekil 3.22: link.boot.ld Dosyası İçerisinde Değişim

```

if(len(sys.argv) < 2):
    print "Usage s19toslm.py FILENAME"
    quit()

l2_banks      = 1
l2_bank_size  = 8192 # in words (32 bit)
l2_start      = 0x00000000
l2_end        = l2_start + l2_banks * l2_bank_size * 4 - 1

tcdm_banks    = 1
tcdm_bank_size = 122880 # in words (32 bit)
tcdm_start    = 0x00100000
tcdm_end      = tcdm_start + tcdm_banks * tcdm_bank_size * 4 - 1
tcdm_bank_bits = int(math.log(tcdm_banks, 2))

```

Şekil 3.23: s19toslm.py Dosyası İçerisindeki Değişim

Dataram ile ilgili donanımsal ve yazılımsal olarak tüm bu değişiklikler yapıldıktan sonra proje tekrar derlenir ve dataram başarılı bir şekilde genişletilmiş olur. Dataram genişletilmesinden sonra

ek 3.4’de verilen modelin c kodu değişiklikler yapılarak Pulpino işlemcisine yüklenecek hale getirilir. Bu aşamada Pulpino işlemcisi için kullanılan özel fonksiyonlardan yararlanmak için “pulpino.h” kütüphanesi , gpio ve uarttan yararlanmak için bunların özel kütüphaneleri koda dahil edilir. Ardından “createImage” komutu ile uygulama kodu bin dosyasına çevrilerek fpga kartına yüklenir ve test edilir. Yapılan testlerin sonucunda soft float modelin sonucu beklenen sonuca eşit olarak fpga kartı üzerinde ledler yakılarak gözlemlendi.

4. DAVRANIŞSAL VE ZAMANLAMA ANALİZİ

Tasarım kodları ModelSim ortamı kullanılarak analiz edilebilmektedir. ModelSim tasarım kodlarının derlendiği ve simüle edildiği bir arayüz sunmaktadır. Vivado'da tasarlanan donanımların pulpino ortamında gözlenmesi ModelSim ile mümkündür. Tasarlanan .c kodu derlendikten sonra tüm sinyaller ModelSim ortamında gözlenerek analiz edilebilir. Oluşturulan C/C++ kodları RI5CY toolchain ile makine koduna dönüştürülür ve davranışsal simülasyon ModelSim ortamında gerçekleşir. ModelSim ortamının Ubuntu ortamında kurulumu ve Pulpino ortamında çalıştırılabilmesi için izlenecek adımlar ve karşılaşılabilecek potansiyel hataların çözümlerinden bahsedilecektir. Pulpino ortamında make ve build komutlarını çalıştırmak için CMake kullanılabilir. Kurulum için CMake kurulum dosyası sitelerinden indirilip arşivden çıkarıldıktan sonra aşağıdaki komut çalıştırılmalıdır ve bin dosyası sistem PATH'ine eklenmelidir.

```
./bootstrap make make install  
<installationfolder>/cmake-3.13.0-rc2/bin
```

Simülasyonlar ModelSim 18.0 versiyonuyla gerçekleştirilmiştir. ModelSim kurulumu gerçekleştirildikten sonra .bashrc'ye figürdeki gibi path eklemeleri yapılmalıdır.

```
PATH=$PATH:/home/berkay/intelFPGA_pro/18.0/modelsim_ase/bin  
PATH=$PATH:/home/berkay/intelFPGA_pro/18.0/modelsim_ase/linuxaloem  
PATH=$PATH:/home/berkay/ri5cy_gnu_toolchain/install/bin  
PATH=$PATH:/usr/local/bin/cmake
```

64-bit işletim sistemlerinde 32-bit işletim sistemleri için olan ModelSim'i çalıştırmak için ia32-libs ve g++multilib kütüphaneleri kurulmalıdır. Aşağıdaki komutlar izlenebilir.

```
sudo dpkg  
add architecture i386  
sudo apt-get install ia32-libs  
sudo apt-get update  
sudo apt-get install  
g++multilib
```

ModelSim kurulumu gerçekleştirildikten sonra, simülasyonları çalıştırmadan önce make vcompile komutu pulpino/sw/build konumunda çağırılmalı ve şekildeki çıktı hatasız alınmalıdır.

```
berkay@Berkay: ~/Desktop/pulpino/sw/build
17.07 Jul 26 2017
vmap pulpino_lib /home/berkay/Desktop/pulpino/vsim/modelsim_libs/pulpino_lib
Modifying modelsim.ini
Compiling component: PULPino

  Compiling for RISCv core
** Warning: /home/berkay/Desktop/pulpino/vsim/./././rtl/random_stalls.sv(72): (vlog-2186) SystemVerilog testbench feature
(randomization, coverage or assertion) detected in the design.
These features are only supported in Questasim.
** Warning: /home/berkay/Desktop/pulpino/vsim/./././rtl/random_stalls.sv(116): (vlog-2186) SystemVerilog testbench feature
(randomization, coverage or assertion) detected in the design.
These features are only supported in Questasim.
--> PULPino compilation complete!
--> Compiling work.tb...
Compiling component: work.tb

--> work.tb compilation complete!

--> PULPino platform compilation complete!

Built target vcompile
berkay@Berkay:~/Desktop/pulpino/sw/build$
```

Şekil 4.1: Make vcompile çıktısı

Yazılan .c kodları, pulpino/sw/apps dosyasının içindeki helloworld.c içine kopyalanarak sırasıyla make helloworld ve make helloworld.vsim komutları çağırılarak derlenip simülink arayüzünde gözlenebilir. Beklenen make helloworld terminal çıktısı şekildeki gibidir.

```
berkay@Berkay: ~/Desktop/pulpino/sw/build
(randomization, coverage or assertion) detected in the design.
These features are only supported in Questasim.
--> PULPino compilation complete!
--> Compiling work.tb...
Compiling component: work.tb

--> work.tb compilation complete!

--> PULPino platform compilation complete!

Built target vcompile
berkay@Berkay:~/Desktop/pulpino/sw/build$ make helloworld
[ 0%] Built target bench
[ 0%] Built target crt0
[ 0%] Built target string
[ 0%] Built target sys
[ 0%] Built target math_fns
[ 0%] Built target helloworld.elf
[ 0%] Built target helloworld.bin.cmd
[100%] Built target helloworld.slm.cmd
[100%] Generating vectors/stim.txt
[100%] Built target helloworld.stim.txt
[100%] Built target helloworld
berkay@Berkay:~/Desktop/pulpino/sw/build$
```

Şekil 4.2: Make helloworld terminal çıktısı

4.1 Karşılaşılabilecek Potansiyel Hatalar ve Çözümleri

-ModelSim kurulumunda kütüphane hatası alınırsa aşağıdaki komut çağrılabilir.

```
$ sudo apt-get install libxtst6:i386
```

-/libstdc++.so.6: version `CXXABI_1.3.11' not found hatası için sırasıyla aşağıdaki komutlar çağrılmalıdır. Ancak hangi kütüphanenin kurulu olduğu kontrol edilmelidir(örneğin 6.0.21).

```
cd <Installation Directory>/<version>/lib/lnx64.o
cp /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.21 .
mv libstdc++.so.6 libstdc++.so.6.bak
ln -s libstdc++.so.6.0.25 libstdc++.so.6
```

-bin/sh: 1: tcsh: not found hatası için kurulum komutu aşağıdaki gibidir.

```
apt-get install tcsh
```

-Kütüphane hatası alındığında modelsim_ase/bin dosyasında gksudo gedit vsim komutu çağırılarak, dir=`dirname "\$arg0"` in altına "export LD_LIBRARY_PATH=\${dir}/lib32" komutu eklenmelidir.

-Error: Cannot find "../linux_rh60/vsim" hatasıyla karşılaşırsa, modelsim_ase dosyasında gksudo gedit vco komutu ile linux_rh60, linux şeklinde düzeltilmelidir.

~/usr/local/cilk/bin/./lib32/pinbin: error while loading shared libraries:

libstdc++.so.6: cannot open shared object file: No such file or directory hatası alınırsa çözüm olarak apt-get install lib32stdc++6 komutu çağırılarak eksik kütüphane indirilebilir.

- -64 bayrağıyla ilgili bir hata alınırsa, /pulpino/vsim/tcl_files içindeki tüm dosyalarda, /pulpino/vsim/vcompile/rtl/vcompile_tb.sh dosyasında ve /pulpino/sw/apps/CMakeSim.txt dosyasında -64 bayrakları silinmelidir.

-jtag_dpi modülüyle ilgili hata alındığında pulpino/vsim/vcompile/rtl/vcompile_tb.sh dosyasında aşağıdaki kısımlar yoruma alınmalıdır.

```
#vlog -quiet -sv -work ${LIB_NAME} +incdir+${TB_PATH} -dpiheader
${TB_PATH}/jtag_dpi/dpiheader.h ${TB_PATH}/jtag_dpi.sv || goto
error
#vlog -quiet -work ${LIB_NAME} -ccflags "-I${TB_PATH}/jtag_dpi/" -
dpicpppath `which gcc` ${TB_PATH}/jtag_dpi/jtag_dpi.c || goto error
#vlog -quiet -sv -work ${LIB_NAME} +incdir+${TB_PATH}
+incdir+${RTL_PATH}/includes/ -dpiheader
${TB_PATH}/mem_dpi/dpiheader.h ${TB_PATH}/tb.sv || goto error
#vlog -quiet -work ${LIB_NAME} -ccflags "-I${TB_PATH}/mem_dpi/" -
dpicpppath `which gcc` ${TB_PATH}/mem_dpi/mem_dpi.c ||
goto error
```

-vsim vlog gibi komutların bulunamadığıyla ilgili bir hata alınırsa şekildeki gibi .bashrc'ye bu dosya uzantıları eklenmelidir.

```

#source /opt/Xilinx/Vivado/2015.1/settings64.sh
source /opt/Xilinx/SDK/2015.1/settings64.sh && source
/opt/Xilinx/Vivado/2015.1/settings64.sh
PATH=$PATH:/opt/Xilinx/Vivado/2015.1/bin
PATH=$PATH:/opt/Xilinx/Vivado/2015.1/bin:/opt/Xilinx/SDK/2015.1/bin
#PATH=$PATH:/opt/ri5cy_gnu_toolchain-master/install/bin
PATH=$PATH:/bin:/opt/ri5cy_gnu_toolchain-master/install/bin:
#export PATH=$PATH:/home/berkay/intelFPGA_pro/18.0/modelsim_ase/bin
PATH=$PATH:/home/berkay/intelFPGA_pro/18.0/modelsim_ase/bin
PATH=$PATH:/home/berkay/intelFPGA_pro/18.0/modelsim_ase/linuxaloem
PATH=$PATH:/home/berkay/ri5cy_gnu_toolchain/install/bin
PATH=$PATH:/usr/local/bin/cmake
PATH=$PATH:/home/berkay/intelFPGA_pro/18.0/modelsim_ase/linuxaloem/vlib
PATH=$PATH:/home/berkay/intelFPGA_pro/18.0/modelsim_ase/linuxaloem/vlog
PATH=$PATH:/home/berkay/intelFPGA_pro/18.0/modelsim_ase/linuxaloem/vmap
PATH=$PATH:/home/berkay/intelFPGA_pro/18.0/modelsim_ase/linuxaloem/vsim

```

Şekil 4.3: .baschrc Eklenen Pathler

4.2 Simülasyon Adımları:

-İlk olarak basit bir .c kodunun çıktıları modelsim ortamında sinyaller aracılığıyla gözlemlenmiştir. İki integer sayının toplandığı bu kodla aritmetik lojik ünitesinin çalıştığı gözlemlenmiştir. İzlenmesi gereken adımlar şöyledir; yazılan .c kodu sw/build içerisinde terminal açılarak make helloworld komutuyla derlenir ve make helloworld.vsim komutuyla modelsim açılır.

tb/top_i/core_region_i/CORE/RISCV_CORE/id_stage_i/registers_i objesinden "mem" wave'e eklenerek bellekte değerler gözlenir.

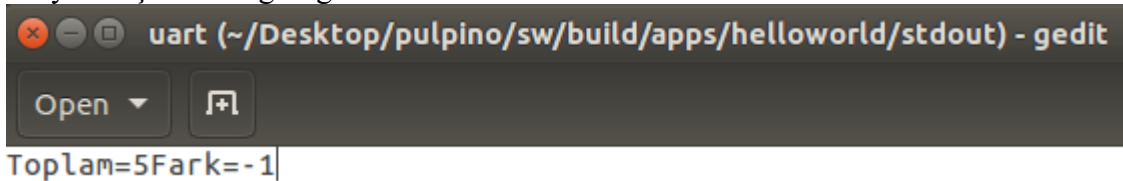
```

1  #include <stdio.h>
2  #include <gpio.h>
3  #include <uart.h>
4  #include <utils.h>
5  #include <pulpino.h>
6
7  int main()
8  {
9      int a=2;
10     int b=3;
11     int c;
12     int d;
13     c=a+b;
14     d=a-b;
15     printf("Toplam=%d",c);
16     printf("Fark=%d",d);
17     return 0;
18 }

```

Şekil 4.4: Basit c kodu

Printf satırlarındaki çıktıları "/home/berkay/Desktop/pulpino/sw/build/apps/helloworld/stdout" kısmındaki dosyadan şekildeki gibi gözlenebilir.



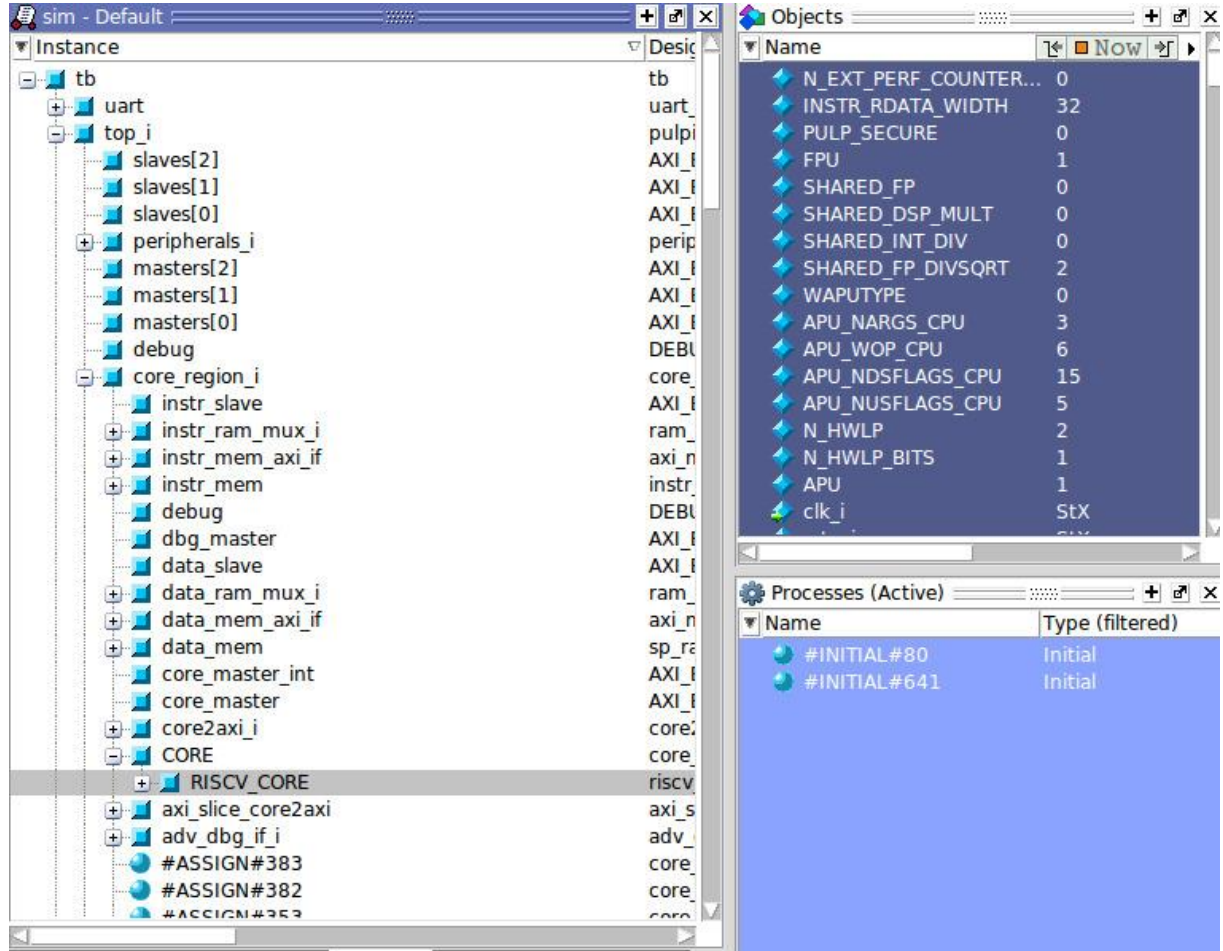
```

uart (~/.Desktop/pulpino/sw/build/apps/helloworld/stdout) - gedit
Toplam=5Fark=-1

```

Şekil 4.5: Uart çıktısı

-Basit float işlemlerini içeren .c kodu çalıştırılarak float işlem biriminin çalıştığı gözlemlenmelidir. Floating point unit'i aktive etmek için sw/build konumundaki cmake_configure.riscvfloat.gcc.sh dosyası düzenlenmelidir. "RISCY_RV32F=1" ve GCC_MARCH="IMFDXpulpv2" değişikliği yapılmalıdır. "tb" dosyasındaki RISCY_RV32F parametresi de 1'e eşitlenmelidir. Sw/build dosyasında değişiklikler yapıldıktan sonra "./cmake_configure.riscvfloat.gcc.sh" komutuyla derlenmelidir. Ardından make vcompile komutuyla işlemcinin FPU'su aktif hale gelir. Modelsim açıldığında tb/top_i/core_region_i/CORE/RISCV_CORE objesinde FPU=1 olarak şekildeki gibi gözlenmelidir.



Şekil 4.6: ModelSim Arayüzü

Modelsimde integer işlemler yapıldığında mem, float işlemler yapıldığında ise mem_fp gözlenmelidir.

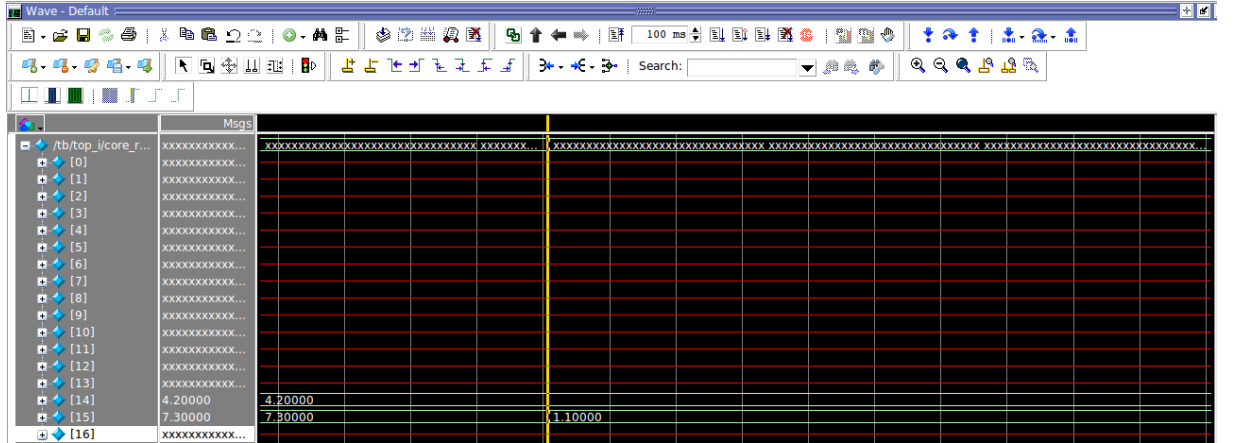
Basit float işlemin yapıldığı şekildeki .c kodu derlenip çalıştırılarak FPU'nun çalıştığından emin olunmalıdır. Modelsim üzerinde mem_fp wave'e eklenerek simülasyon bir süre çalıştırıldığında şekildeki gibi .c kodundaki işlemler register'larda gözlenebilir.


```
helloworld.c (~/Desktop)
Open [ ]

#include <stdio.h>
#include <gpio.h>
#include <uart.h>
#include <utils.h>
#include <pulpino.h>

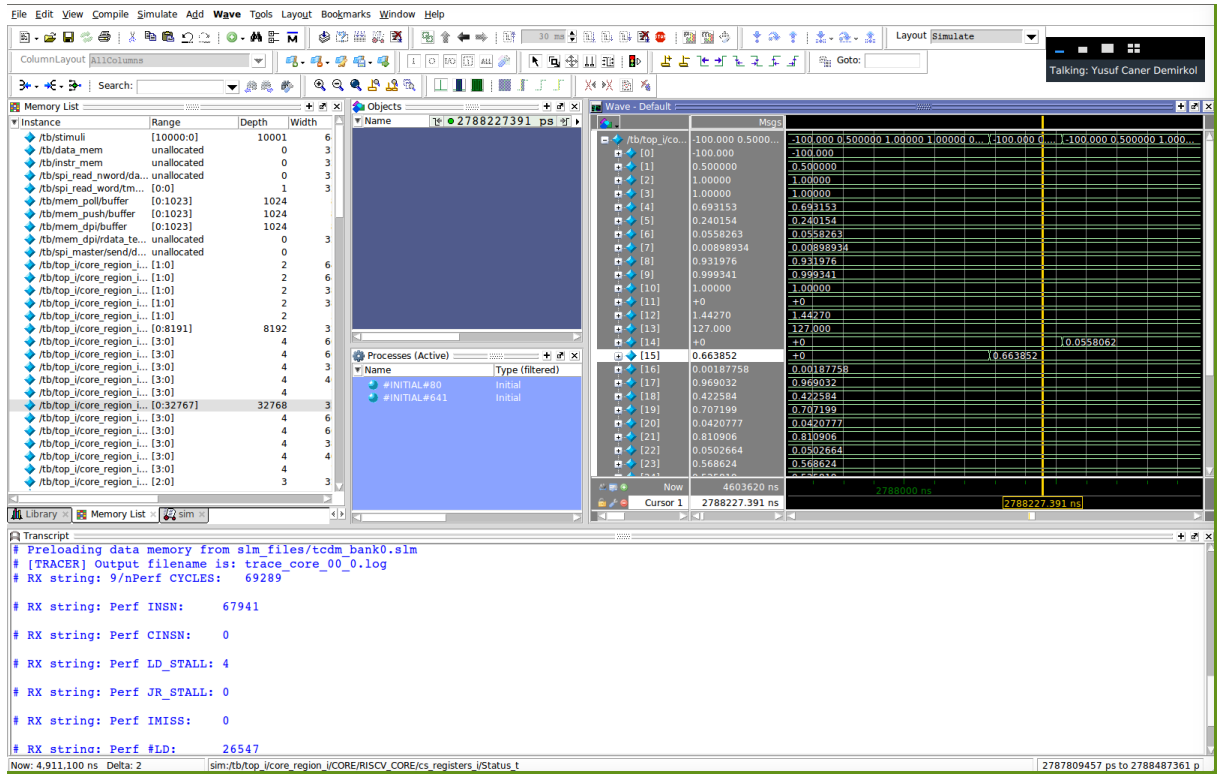
int main()
{
    float a=4.2;
    float b=3.1;
    float c;
    float d;
    c=a+b;
    d=a-b;
    printf("Toplam=%f",c);
    printf("Fark=%f",d);
    return 0;
}
```

Şekil 4.7: Basit float kodu



Şekil 4.8: Wave float işlem çıktıları

Çalışılacak ortamın tüm bileşenlerinin çalıştığından emin olunduktan sonra ana model dosyası çalıştırılarak şekildeki çıktı alınmıştır.



Şekil 4.9: Ana model çıktısı

.c model dosyasının beklenen çıktısı olan 0.663852 ağırlık değeri registerda simülasyon sonucunda gözlenmiştir.

4.3 Soft-Hard float

Soft float kavramı float işlem kullanılan kodlarda işlemlerin I ve M komut setleri kullanılarak gerçekleştirilmesi anlamına gelmektedir. Donanımda fpu birimini kullanmadığı için float işlemlerde hassas sonuç verse de çok uzun zaman gerektirmektedir. Hard float işlemde ise float işlemler doğrudan float komutlara dönüştürülür ve donanımda fpu birimini tetikleyerek zamandan kazanmamızı sağlar. Burada bir diğer hassas mesele ise Riscy işlemcisi sadece single float formatını desteklemekte ancak gcc derleyicisi ise double formatta komut üretmektedir. Bu yüzden üretilen komutların single forma çevrilmesi için GCC ayarlarından TARGET_C_FLAGS=""-O3 -m32 -g -fsingle-precision-constant -mfpdouble=float -Werror=double-promotion" şeklinde cmake dosyasında değiştirilmelidir.

Pulpino'nun sunduğu performans sayıcıları model koduna eklenerek Modelsim ortamında analiz edilmiştir. Elde edilen çıktılar şekildeki gibidir. Performans cycle'ı ve instruction sayıları dikkate alınarak farklı optimizasyon ayarlarında şekildeki analizler yapılmıştır. Soft ve hard float karşılaştırmalı olarak tabloda analiz edilebilir.

```

# RX string: Perf CYCLES: 3226647
# RX string: Perf INSN: 2609592
# RX string: Perf CINSN: 0
# RX string: Perf LD_STALL: 54371
# RX string: Perf JR_STALL: 26547
# RX string: Perf IMISS: 0
# RX string: Perf #LD: 457708
# RX string: Perf #ST: 323114
# RX string: Perf #JUMP: 225875
# RX string: Perf #BRANCH: 488321
# RX string: Perf #TAKEN: 132353
# RX string: Perf #RVC: 0

```

Şekil 4.10: Performans Sayıcı çıktıları

Tablo 1: Hard float çıktıları

Tür	Optimizasyon	Perf Cycles	INSN
Hard Float	O0	581824	441569
Hard Float	O3	69289	67941
Hard Float	Os	77157	73663

Tablo 2: Soft float çıktıları

Tür	Optimizasyon	Perf Cycles	INSN
Soft Float	O0	3723322	3016338
Soft Float	O3	3226647	2609592

Tb dosyasında işlemcinin periyodu, dolayısıyla frekansı değiştirilerek simülasyon süreleri kısaltılabilmektedir. Şekilden sonuçlar görülebilir.

Tablo 3: Frekansın simülasyon süresine etkisi

Tür	Optimizasyon	Frekans	Simülasyon süresi
Hard Float	O3	25MHz	5,672,680 ns
Hard Float	O3	50MHz	2,848,250 ns
Hard Float	O3	100MHz	1,436,225 ns

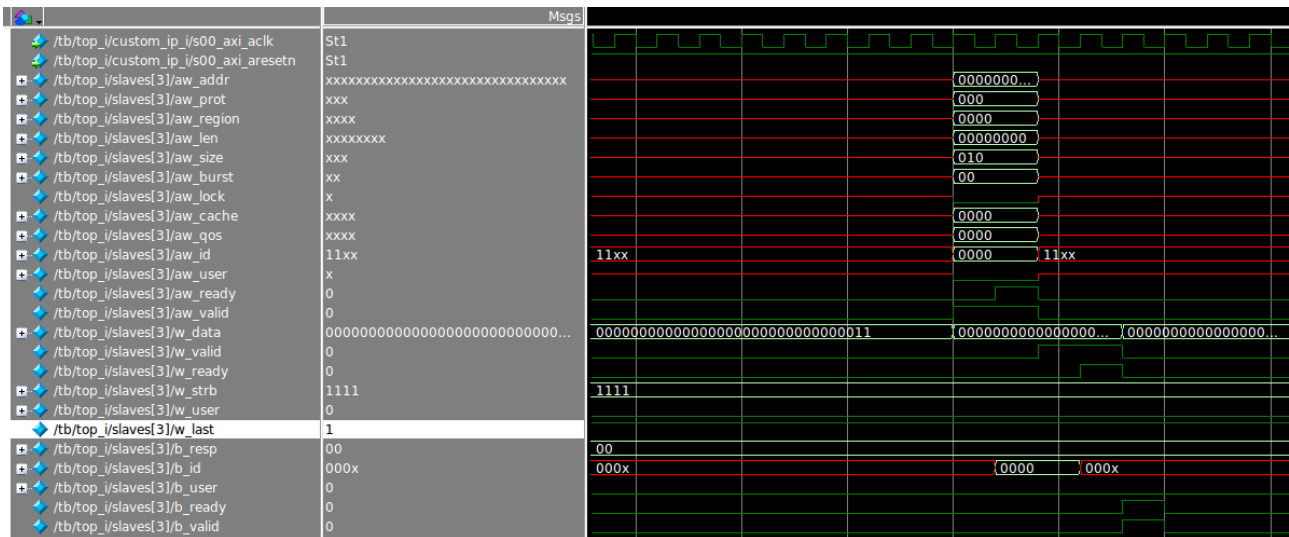
5. CUSTOM HARDWARE

Günümüzde git gide daha fazla kullanılmaya başlanan veri-yoğun yapılar özel mimarilere ihtiyaç artmaktadır. İrili ufaklı birçok donanım firması yapay zeka uygulamaları için özel IP ve çip tasarımlarına girişmektedir(Bknz. Intel Mobileye, Cerebras, Tenstorrent vs.). Bu tasarımlar arasında custom IP olarak bilinen ve bus haberleşmeleri vasıtasıyla işlem çekirdeğine bağlanan yapılar sıkça kullanılmaktadır. Tasarım için bus haberleşmesi olarak Pulpino ortamının desteklediği AXI haberleşmesi seçilmiştir.

5.1 AXI Arayüzü

Sıkça kullanılan arayüzlerden olan AXI farklı uygulamalar için farklı seçenekler sunmaktadır. Temelde üçe ayrılan mimari, hızlı veri haberleşmesi gerektiren uygulamalar için Full, görece daha yavaş ve çift taraflı haberleşme ihtiyacı duyan sistemler için Lite ve hızlı, kontrol karmaşasının az olmasına ihtiyaç duyan sistemler için Stream arayüzlerini sunmaktadır. Bu arayüzler sayesinde çevresellerin hafıza birimine yerleştirilmesi(memory-mapped) ve işlemcinin load-store komutları ile bu hafıza aralıklarına basitçe erişim imkanı sunulmaktadır. 2011 yılında tanıtılan AXI 4 mimarisi kısa sürede de facto mimarilerden biri haline geldi[1].

Temelde master ile slave birimlerinin valid ve ready sinyalleri ile birbirleriyle anlaşmasına dayalı olarak verinin iletimini gerçekleştiren mimari bunların yanında birçok yardımcı sinyal ile güvenli ve hızlı haberleşmenin garantisini sunmaktadır. 5 kanaldan oluşan mimaride tam bir veri alışverişi şu şekilde gerçekleşmektedir; Master tarafı önce yazılacak adres bilgisini ardından ise yazılacak veriyi gönderir. Sonrasında Slave tarafı veriyi sağlıklı şekilde aldığı bilgisini gönderir. Master tarafı veri okuma esnasında okunacak verinin adresini gönderir ve ardından o adresteki veri hatta yüklenerek okuma tamamlanır.



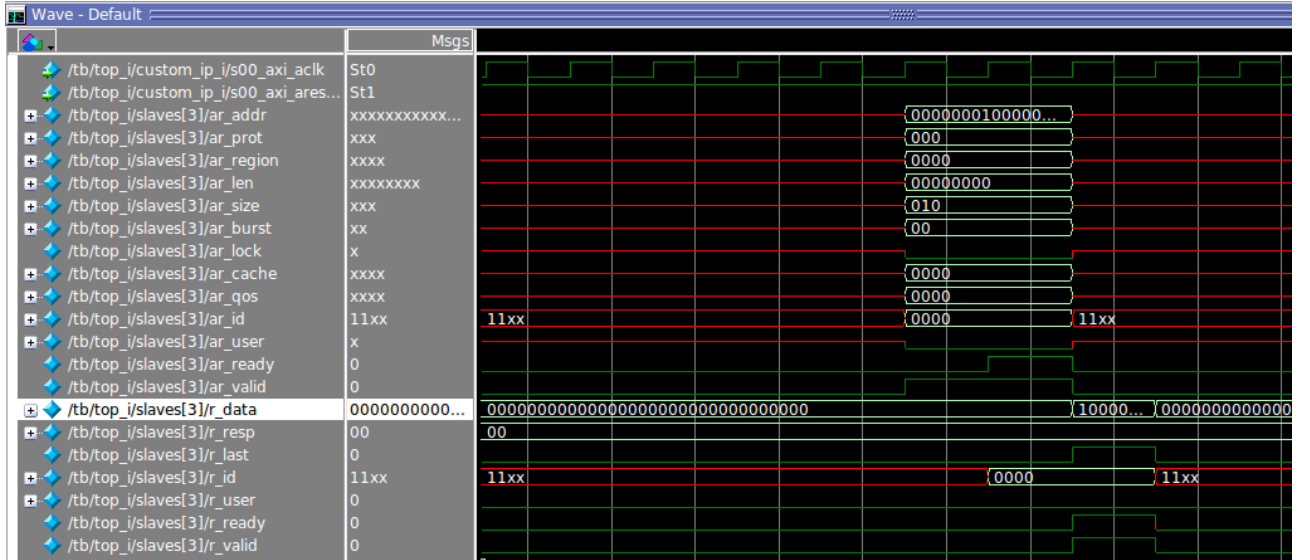
Şekil 5.1: AXI Full Veri Yazma Aksiyonu

Şekil 5.1’de görüldüğü üzere Master tarafı öncelikle veri okuyacağı adresi aw_addr hattı üzerinden gönderir. Bununla beraber aw_size sinyali ile kaç bitlik bir yazma işlemi yapacağını bildirir. Şekil 5.2’de görüldüğü gibi “010” bilgisi 4 byte’lık transferlere denk gelmektedir.

AxSIZE[2:0]	Bytes in transfer
0b000	1
0b001	2
0b010	4
0b011	8
0b100	16
0b101	32
0b110	64
0b111	128

Şekil 5.2: AwSIZE Sinyali Byte Bilgileri[1]

Şekil 5.3'te veri okuma dizini görülmektedir.



Şekil 5.3: AXI Full Veri Okuma Aksiyonu

AXI Full arayüz tasarımı Xilinx Vivado ortamındaki şablon üzerinde gerekli değişiklikler yapılarak kullanıldı. Pulpino ortamının desteklediği AXI arayüzüne uygun hale getirebilmek için tasarımda bazı değişikliklere ihtiyaç duyuldu. Şekil 5.4'te görüldüğü üzere ID sinyalleri birer cycle ötelendi ve USER sinyalleri sıfıra sabitlendi.

```

assign S_AXI_AWREADY = axi_awready;
assign S_AXI_WREADY = axi_wready;
assign S_AXI_BRESP = axi_bresp;
assign S_AXI_BVALID = axi_bvalid;
assign S_AXI_BID = axi_bid; //modified by altK
assign S_AXI_ARREADY = axi_arready;
assign S_AXI_RDATA = axi_rdata;
assign S_AXI_RRESP = axi_rresp;
assign S_AXI_RLAST = axi_rlast;
assign S_AXI_RUSER = 0; //modified by altK
assign S_AXI_RVALID = axi_rvalid;
assign S_AXI_RID = axi_rid; //modified by altK
assign aw_wrap_size = (C_S_AXI_DATA_WIDTH/8 * (S_AXI_AWLEN));
assign ar_wrap_size = (C_S_AXI_DATA_WIDTH/8 * (S_AXI_ARLEN));
assign aw_wrap_en = ((axi_awaddr & aw_wrap_size) == aw_wrap_size)? 1'b1: 1'b0;
assign ar_wrap_en = ((axi_araddr & ar_wrap_size) == ar_wrap_size)? 1'b1: 1'b0;
assign S_AXI_BUSER = 0;

```

Şekil 5.4: AXI Full Sinyal Tanımları

Arayüz sıkıntıları halledildikten sonra RTL tasarım alt modül olarak AXI şablonuna eklendi. Gelen verileri seri şekilde işleyen bir donanım tasarımı yapıldığı için bu aşamada buffer ya da register tarzı arabirimlere ihtiyaç duyulmadı.

AXI arayüzü sağlıklı şekilde kurulduktan sonra Pulpino ortamında yapılan değişiklikler sayesinde Core ile arasında iletişim sağlandı.

```

//-----//
// Axi node
//-----//

axi_node_intf_wrap
#(
    .NB_MASTER (4),
    .NB_SLAVE (4),
    .AXI_ADDR_WIDTH ( `AXI_ADDR_WIDTH ),
    .AXI_DATA_WIDTH ( `AXI_DATA_WIDTH ),
    .AXI_ID_WIDTH ( `AXI_ID_MASTER_WIDTH ),
    .AXI_USER_WIDTH ( `AXI_USER_WIDTH )
)
axi_interconnect_i
(
    .clk ( clk_int ),
    .rst_n ( rstn_int ),
    .test_en_i ( testmode_i ),

    .master ( slaves ),
    .slave ( masters ),

    .start_addr_i ( { 32'h0100_0000, 32'h1A10_0000, 32'h0010_0000, 32'h0000_0000 } ),
    .end_addr_i ( { 32'h010F_0000, 32'h1A11_FFFF, 32'h001F_FFFF, 32'h000F_FFFF } )
);

endmodule

```

Şekil 5.5: Top Modülde Yapılan Değişiklikler

Pulpino tasarım ortamında pulpino_top dosyası altında Şekil 5.5'te görüldüğü üzere NB_MASTER ve NB_SLAVE değişkenlerinin birer artırılması gerekmektedir. Ardından başlangıç ve bitiş adresleri hafıza aralıklarına eklenmelidir. Burada sağdan sola instruction ram, data ram, peripheral ve custom ip adres aralıkları yer almaktadır. Ardından oluşturulan array'ler ilgili master-slave kombinasyonlarına yerleştirilmelidir. Oluşturulan tasarım 4. slave olarak eklenmiş ve Şekil 5.6'da port dizilimi gösterilmiştir.

```

// Custom IP AXI Full
myipTest_v1_0
#(
.C_S00_AXI_DATA_WIDTH(AXI_DATA_WIDTH),
.C_S00_AXI_ADDR_WIDTH(AXI_ADDR_WIDTH),
.C_S00_AXI_ID_WIDTH(AXI_ID_SLAVE_WIDTH),

.C_S00_AXI_AWUSER_WIDTH(AXI_USER_WIDTH),
.C_S00_AXI_ARUSER_WIDTH(AXI_USER_WIDTH),
.C_S00_AXI_WUSER_WIDTH(AXI_USER_WIDTH),
.C_S00_AXI_RUSER_WIDTH(AXI_USER_WIDTH),
.C_S00_AXI_BUSER_WIDTH(AXI_USER_WIDTH)
)
custom_ip_i
(
.s00_axi_aclk(clk_int),
.s00_axi_aresetn(rstn_int),
.s00_axi_awaddr(slaves[3].aw_addr),
.s00_axi_awprot(slaves[3].aw_prot),
.s00_axi_awvalid(slaves[3].aw_valid),
.s00_axi_awready(slaves[3].aw_ready),
.s00_axi_wdata(slaves[3].w_data),
.s00_axi_wstrb(slaves[3].w_strb),
.s00_axi_wvalid(slaves[3].w_valid),
.s00_axi_wready(slaves[3].w_ready),
.s00_axi_bresp(slaves[3].b_resp),
.s00_axi_bvalid(slaves[3].b_valid),
.s00_axi_bready(slaves[3].b_ready),
.s00_axi_araddr(slaves[3].ar_addr),
.s00_axi_arprot(slaves[3].ar_prot),
.s00_axi_arvalid(slaves[3].ar_valid),
.s00_axi_arready(slaves[3].ar_ready),
.s00_axi_rdata(slaves[3].r_data),
.s00_axi_rresp(slaves[3].r_resp),
.s00_axi_rvalid(slaves[3].r_valid),
.s00_axi_rready(slaves[3].r_ready),

//full
.s00_axi_awid(slaves[3].aw_id),
.s00_axi_awlen(slaves[3].aw_len),
.s00_axi_awsz(slaves[3].aw_size),
.s00_axi_awburst(slaves[3].aw_burst),
.s00_axi_awlock(slaves[3].aw_lock),
.s00_axi_awcache(slaves[3].aw_cache),
.s00_axi_awqos(slaves[3].aw_qos),
.s00_axi_awregion(slaves[3].aw_region),
.s00_axi_awuser(slaves[3].aw_user),

.s00_axi_wlast(slaves[3].w_last),
.s00_axi_wuser(slaves[3].w_user),

.s00_axi_arid(slaves[3].ar_id),
.s00_axi_aren(slaves[3].ar_len),
.s00_axi_arsz(slaves[3].ar_size),
.s00_axi_arburst(slaves[3].ar_burst),
.s00_axi_arlock(slaves[3].ar_lock),
.s00_axi_arcache(slaves[3].ar_cache),
.s00_axi_arqos(slaves[3].ar_qos),
.s00_axi_arregion(slaves[3].ar_region),
.s00_axi_aruser(slaves[3].ar_user),

.s00_axi_rid(slaves[3].r_id),
.s00_axi_rlast(slaves[3].r_last),
.s00_axi_ruser(slaves[3].r_user)
);

```

Şekil 5.6: Custom IP Port Dizilimi

5.2 Yerel Hafıza Yapısı

Veri yoğun sistemlerde en önemli problemlerden olan verilerin hafızadan işlem birimine taşınması işlemi için hafıza bloklarının işlem birimlerine yakın ve özelleştirilmiş kontrol birimleriyle kullanılmasına karar verilmiştir. Bunun için scratchpad adı verilen hafıza blokları ilk ara katman, ikinci ara katman ve çıkış katmanını ağırlıkların tutmaktadır.

Tasarım içerisinde register şeklinde oluşturulan yapılar yazılan python script'i sayesinde oluşturulan model parametleri ile kolayca doldurulabilmektedir. Şekil 5.7'de script kodundan ilgili parça görülmektedir.

```
#function define to convert float data to hex form
def float_to_hex(f):
    return hex(struct.unpack('<I', struct.pack('<f', f))[0])

#opens separate txt filex to write coefficients
out_mem = open("out_mem.txt", 'w')

s_mem = open("s_mem.txt", 'w')

f_mem = open("f_mem.txt", 'w')

#conversion
for i in range(out_weights_matrix.size):
    out_mem.write("out_mem[%d] = %s;\n" % ( i, float_to_hex(out_weights_matrix[i]) ))

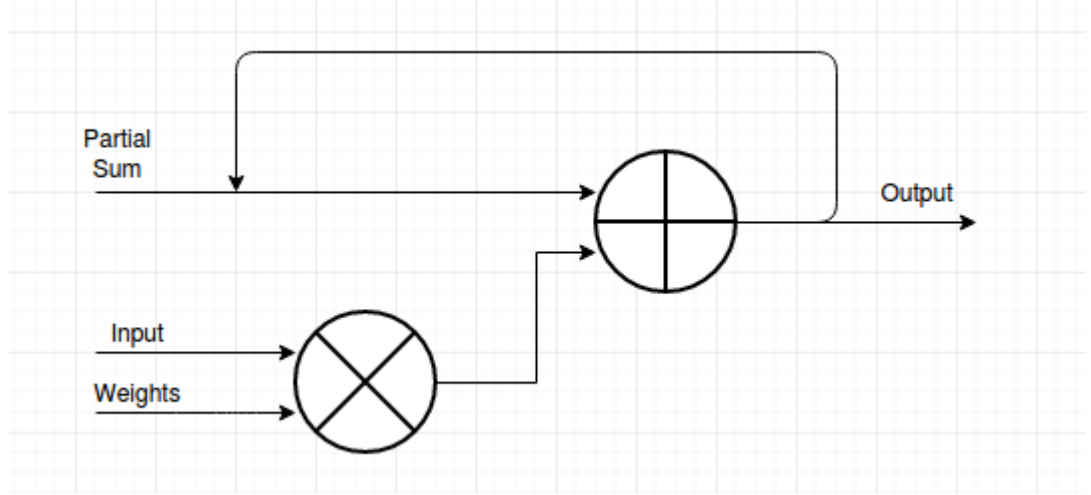
for i in range(s_weights_matrix.size):
    s_mem.write("s_mem[%d] = %s;\n" % ( i, float_to_hex(s_weights_matrix[i]) ))

for i in range(f_weights_matrix.size):
    f_mem.write("f_mem[%d] = %s;\n" % ( i, float_to_hex(f_weights_matrix[i]) ))
```

Şekil 5.7: Ağırlıkların Verilog formatına dönüştürülmesi

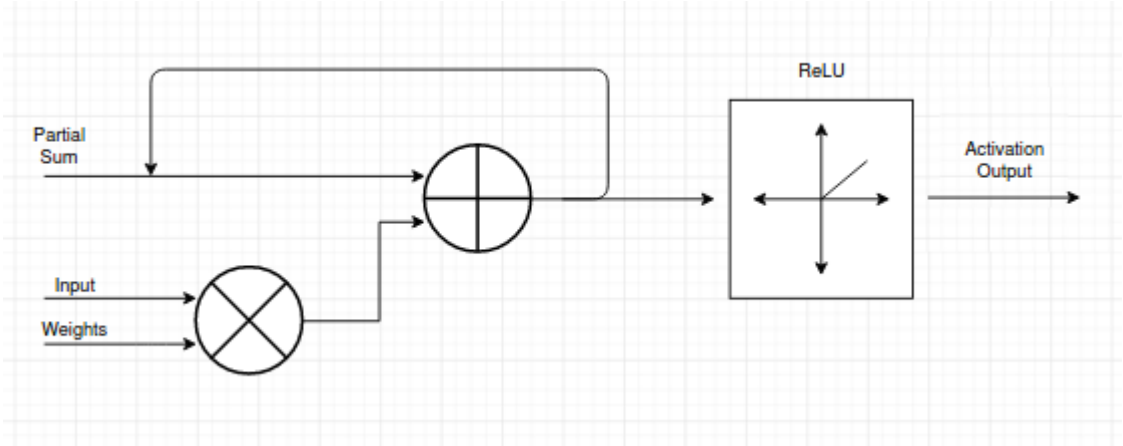
5.3 Custom IP Tasarımı

Custom IP tasarımı için temel bileşen olarak MAC ünitesi üzerinde karar kılındı. MAC işlemi geçmişteki toplam değeri üzerine yeni gelen iki girişi çarpılarak ekleyen basit bit işlemidir. Şekil 5.8’de işlem diyagramı görülmektedir.



Şekil 5.8: MAC İşlem Birimi

Sinir ağı gerçek sayılarla modellendiği için işlem birimi 32 bit IEEE float sayı formatı desteğinde hazırlanmıştır. Ardından bu işlem birimi neronları temsil edecek şekilde ağ yapısı kurulmuştur. Paralel olarak yerleştirilerek parametrik bir tasarım amaçlanmıştır.



Şekil 5.9: MAC ve Aktivasyon Fonksiyonu ile Gerçekleşmiş Nöron Yapısı

Sonraki adım olarak bu MAC ve Aktivasyon birimleri ağıın parametleri doğrultusunda çoğaltılarak paralel ve seri kombinasyonlarla dizilmiştir. Tasarım içerisinde alan ve eğitim zamanı gibi kısıtlar göz önüne alınarak ilk katman için 16 nöron, ikinci katman için 16 nöron ve çıkış katmanı için 10 nöron kullanılmasına karar verilmiştir. Katmanların içindeki nöronlar birbirlerine paralel işlem yapacak şekilde art arda gelen katmanlar ise seri şekilde donanıma aktarılmıştır.

Tasarımda ana motivasyon her katmanda giriş verilerinin(Şekil 5.9’da Input girişi) bütün paralel nöronlarda ortak olmasından dolayı yüksek paralelleştirme oranına ulaşabiliyor olmasından ötürü idi. Ağırlıklar ise yerel hafıza yapılarından paralel olarak çekilerek tek saat periyodu içerisinde nöronlardan çıktı alınması garanti edildi.

Tasarım sonuçları Modelsim ortamında yazılan Python Script’inin sonuçları ile karşılaştırılmıştır. Şekil 5.10’da Script’den ilgili parça görülebilir.

```
f_weights_matrix = np.loadtxt("f_weights_matrix.txt", delimiter=',')
s_weights_matrix = np.loadtxt("s_weights_matrix.txt", delimiter=',')
out_weights_matrix = np.loadtxt("out_weights_matrix.txt", delimiter=',')

validation_set = [0.0000000000000000e+00,0.0000000000000000e+00,0.0000000000000000e+00,0.00000000(
0000000000000000e+00,1.0000000000000000e+00,0.0000000000000000e+00,0.0000000000000000e+00,0.000(

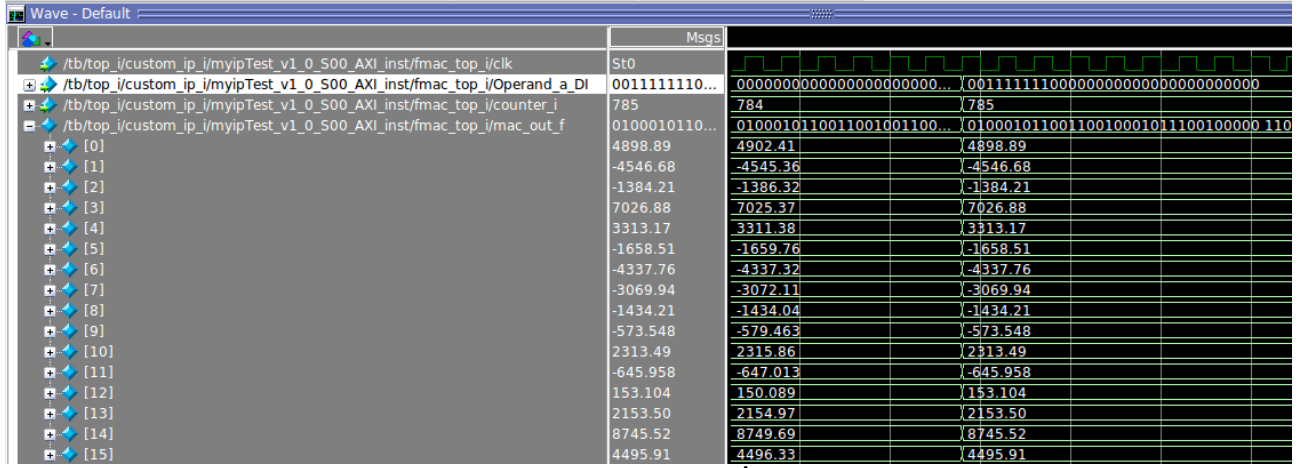
validation_set_n = np.array(validation_set)

def sigmoid(vec_x):
    vec_y = []
    for i in range(len(vec_x)):
        vec_y.append(vec_x[i] if vec_x[i]>0 else 0)
    return vec_y

f_v = f_weights_matrix.dot(validation_set_n.reshape(28*28+1 ))
print(f_v)
f_y = sigmoid(f_v)
print(f_y)
s_v = s_weights_matrix.dot(np.concatenate([f_y, [1]]))
print(s_v)
s_y = sigmoid(s_v)
print(s_y)
out_v = out_weights_matrix.dot(np.concatenate([s_y, [1]]))
print(out_v)
```

Şekil 5.10: Sonuçların Hassasiyetini Karşılaştırmak için Yazılan Python Kodu

Simülasyon sonucunda ilk katman çıkışında elde edilen sonuçlar Şekil 5.11 ve Python modelinde elde edilen sonuçlar Şekil 5.12’de verilmiştir. Girişler Pulpino işlemcisi üzerinden AXI ile seri şekilde gönderilmiş ve şekilde görüldüğü gibi son veriyi(784 giriş + 1 bias terimi) aldıktan sonra sonucu üretmektedir.

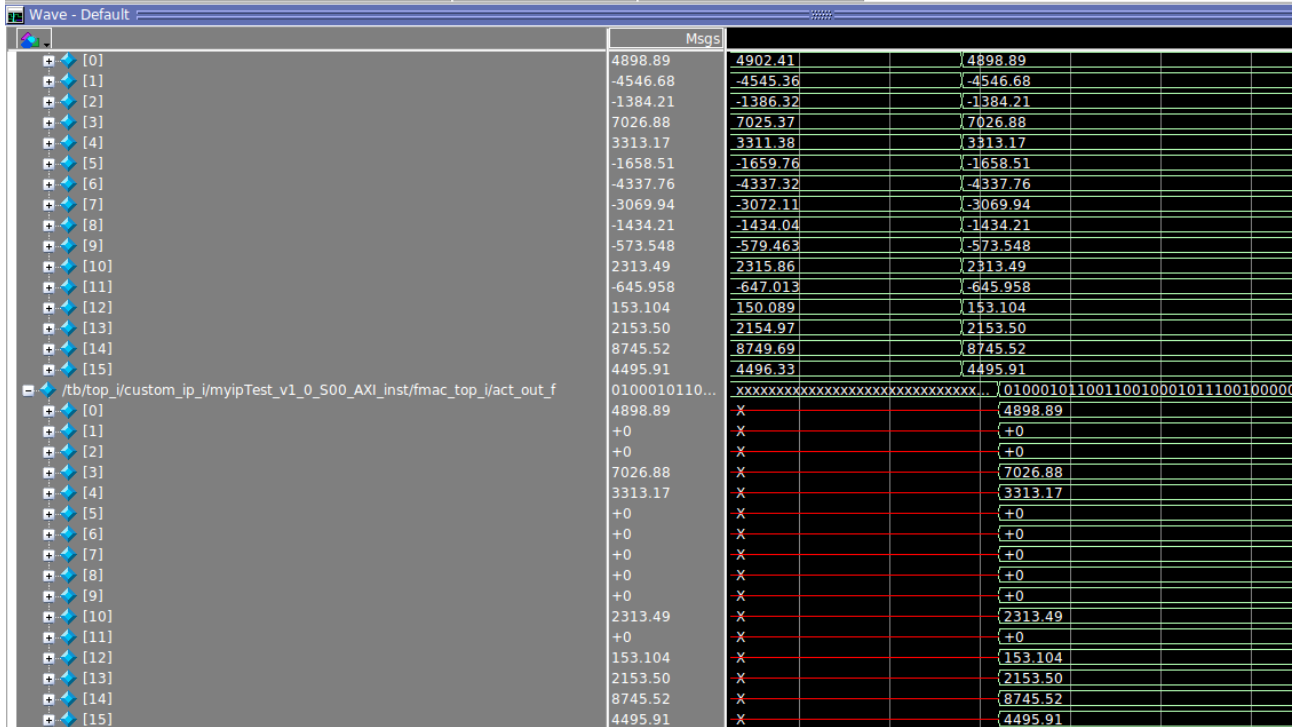


Şekil 5.11: Simülasyon Sonucu İlk Katman Çıkışı

```
[ 4898.89128902 -4546.67895828 -1384.21020193 7026.8834912
3313.16583611 -1658.51355201 -4337.75908976 -3069.94279922
-1434.20882864 -573.54849889 2313.48511374 -645.95844443
153.10340532 2153.50302695 8745.51552105 4495.9045083 ]
```

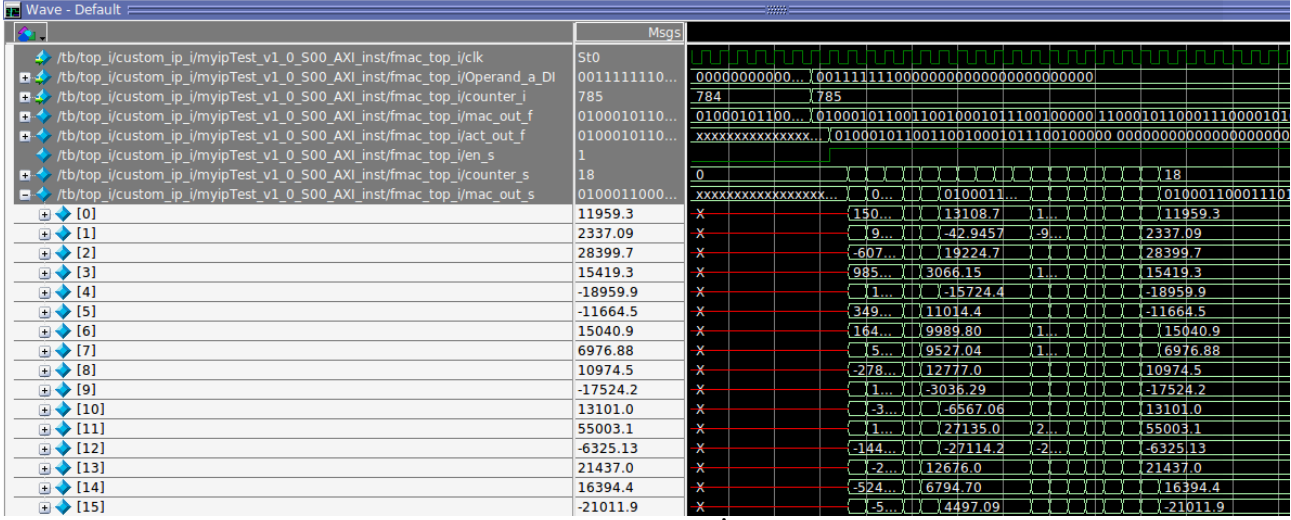
Şekil 5.12: Python Modeli Sonucu İlk Katman Çıkışı

Ardından ilk katman çıkışı aktivasyon fonksiyonuna girerek bi sonraki saat darbesinde ilk ara katman çıkışını oluşturmaktadır. Şekil 5.13'te bu durum görülebilmektedir.



Şekil 5.13: İlk Katman Aktivasyon Fonksiyon Çıkışı

İlk katman aktivasyon fonksiyonu çıkışları seri şekilde ikinci katmana verilmiş ve sonuçlar 17(16 ilk katman çıkışı + 1 bias terimi) saat darbesinden sonra elde edilmiştir. Sonuçlar Şekil 5.14'te görülmektedir.



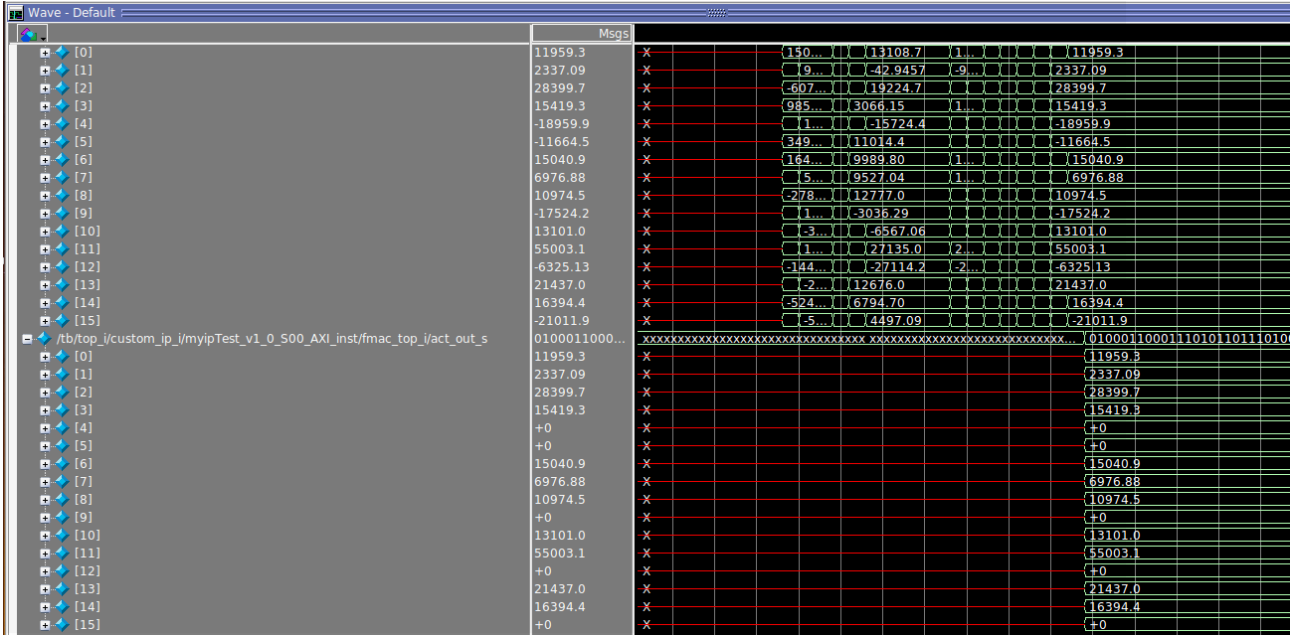
Şekil 5.14: Simülasyon Sonucu İkinci Katman Çıkışı

Şekil 5.15'te Python Modelinin ikinci katman çıkışları görülmektedir.

```
[ 11959.25026317   2337.09038258  28399.65089805  15419.33069404
 -18959.88770624  -11664.47887522  15040.92685273   6976.88177023
 10974.49363771  -17524.15321216  13101.03925404  55003.13812273
 -6325.13780383  21436.99545716  16394.44432927  -21011.94408115]
```

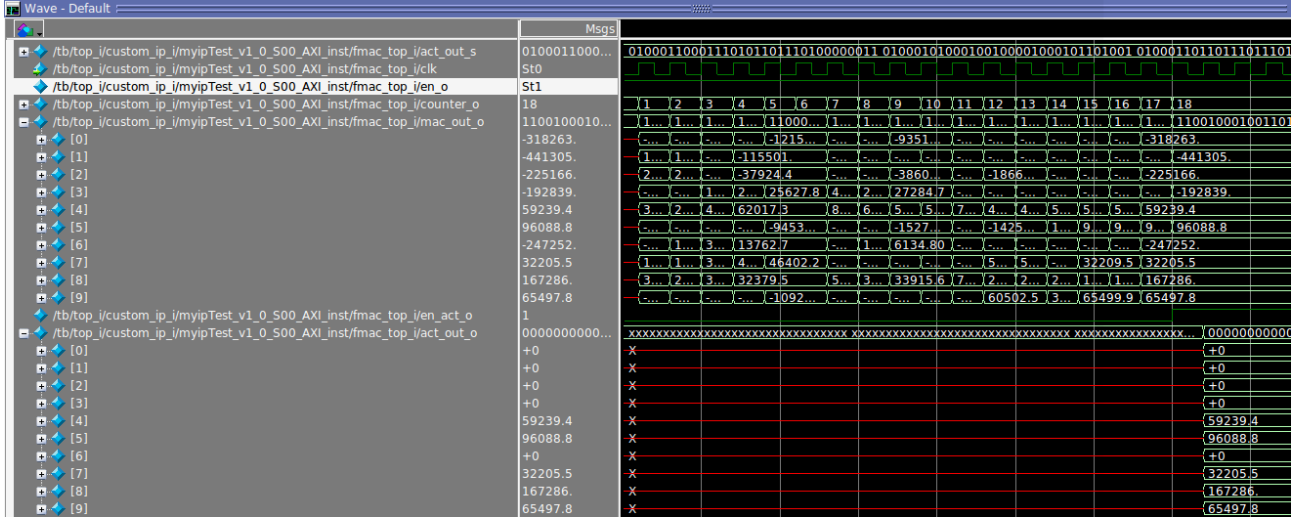
Şekil 5.15: Python Modeli Sonucu İkinci Katman Çıkışı

Şekil 5.16'da ikinci katmanın aktivasyon fonksiyonu çıkışı görülmektedir.

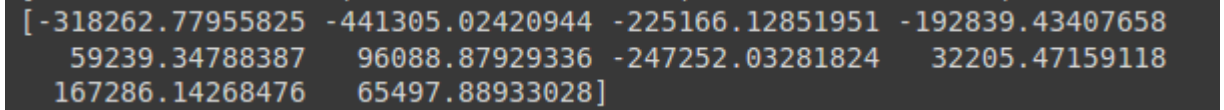


Şekil 5.16: İkinci Katman Aktivasyon Fonksiyon Çıkışı

Çıkış katmanına ise hazır olan ikinci katman aktivasyon çıkışları seri şekilde uygulanarak sonuçlar Şekil 5.17’de görüldüğü gibi elde edilmiştir. Burada aktivasyon fonksiyonu çıkışı artık ağıımızın sonucu olmakta ve şekilde de görüldüğü gibi bu giriş için çıkışta en yüksek değer olan 9. sınıfa ait bir veri olduğu anlaşılmaktadır. Çıkış katmanı için Python modeli çıktılarına Şekil 5.18’de erişilebilir.



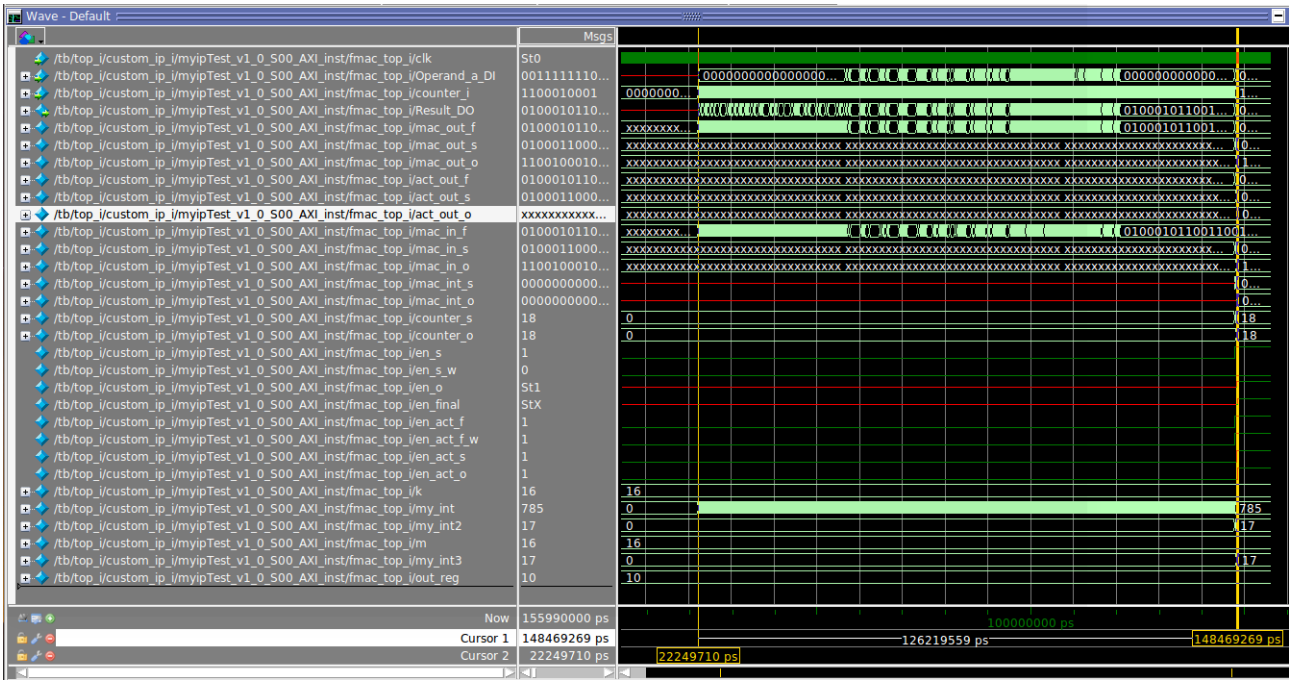
Şekil 5.17: Simülasyon Sonucu Çıkış Katmanı Sonuçları



Şekil 5.18: Python Modeli Sonucu Çıkış Katmanı Çıktıları

Python modeli ile donanım çıktıları şekillerden görüldüğü gibi çok yakın hassasiyet değerlerine ulaşmaktadır. Dolayısıyla Python modelinde elde edilen doğruluk oranlarına donanım üzerinde rahatlıkla erişilebileceği anlaşılmaktadır.

Eklenen donanım bloğu ile birlikte tek bir girdinin sonuçlanması Şekil 5.19’da görüldüğü üzere 126,2 us sürmektedir. Saat frekansı 50 Mhz’de çalıştırıldığı için tüm işlem 6311 cycle sürmektedir.



Şekil 5.19: Donanım Bloğunun Zamanlama Analizi

6. GERÇEKÇİ KISITLAR, SONUÇLAR VE ÖNERİLER

6.1 Çalışmanın Uygulama Alanı

Bu çalışmada temel olarak düşük güç tüketimine ve düşük gecikmeye sahip olması gereken sınır aiot aygıtları heeflenmiştir. Açık kaynak ve düşük tüketime sahip RISC-V işlemcisi ile alan yönünden verimli ve düşük hesaplama zamanına sahip özel donanım bu uygulama alanının ihtiyaçlarını karşılamaktadır.

6.2 Gerçekçi Tasarım Kısıtları

Tasarım esnasında birçok gerçekçi kısıtla karşılaşmıştır. Açık kaynak bir kodla çalışıldığı için tasarıma müdahale edebilmenin rahatlığına rağmen tasarımlarda birçok eksikle karşılaşılması zaman anlamında projeyi zorlamıştır.

6.2.1 Maliyet

Yapılan işlemler Nexys DDR4 kartı üzerinde test edilmiştir. Onun dışında çoğu işlem simülasyon ortamında çıktılarla ilerlediği için projede maliyet bakımından darı düşülmemiştir.

6.2.2 Standartlar

Proje boyunca donanım tasarımları IEEE'nin Verilog ve VHDL standartları temel alınarak ilerlemiştir. Aynı şekilde C modeli C99 standartını minimum referans olarak tamamlanmıştır.

6.2.3 Sosyal, çevresel ve ekonomik etki

Üzerinde çalışılan konu milyar dolarlık iot ve otonom araçlar sektörünü hedeflediği için etkili yazılım ve donanım tasarımları hem ülkeler hem de şirketler odağında ciddi kazançlara yol açabilir.

6.2.4 Sağlık ve güvenlik riskleri

Tasarımların hedef olarak kullanıcıyla etkileşimi olacak araçlarda kullanılacak olması insan hayatını riske atmaktadır. Tasarımlar için donanım ve yazılım güvenliği

birçok dışardan saldırıya karşı korumasız olması muhtemel olup daha ileriki arařtırmalarda bu konu üzerinde durulması iyi projeye kapı açabilir.

6.3 Sonular

Sonu olarak sıfırdan yazılan ve rekabeti bir oranda doėruluk sunan sinir aėı modelimiz donanım üzerinde gereklenmiř ve karřılařtırmalı sonular sunulmuřtur. Öncelile IEEE Single Float formatında gereklenen model iřlemcide I ve M komut setleri ile gereklenmiřtir. Daha sonra iřlemciye donanım katmanında FPU(Float Point Unit) eklenmiř ve kod F komut setini de kapsayacak řekilde derlenmiřtir. Optimizasyon seviyesi aynı derleyici ile bu iki gerekleme arasında yaklařık 45 katlık bir zamansal iyileřme ile karřılařıldı. Buna karřılık FPU biriminin aktifleřtirilmesi ip alanının artmasına sebep olmaktadır. Sonrasında FPU birimi aktif iken iřlem yk iřlemciye veri hattı ile baėlanmış zel donanım bloėuna kaydırılmıř ve burda da FPU’lu gereklemeye gre yaklařık 11 kat iyileřme elde edilmiřtir. Burdaki iyileřmenin ise esas sebebi yerel hafıza yapısının ve paralel iřlem biriminin getirdiėi avantajlar olmuřtur.

6.4 Geleceėe Ynelik neriler

Son tasarımda giriř verileri(resim pikselleri) AXI hattı üzerinden seri řekilde sunulmaktadır. Bu veriler zel DMA(Direct Memory Access) bloėu yardımıyla Ram üzerinden doėrudan iřlem birimine burst modu yardımıyla aktarılması saėlanabilir. Bu sayede AXI haberleřmesinde gecikmeye sebep olan “handshaking” kısıtlamasının nne geilebilir. Bununla beraber tasarımların detaylı alan ve g analizleri yapılarak daha saėlıklı karřılařtırmalar sunulabilir.

KAYNAKLAR

- [1] **V. Sze, Y. Chen, T. Yang and J. S. Emer**, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," in Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, Dec. 2017, doi: 10.1109/JPROC.2017.2761740.
- [2] **Asanovic, K., Patterson, D.**(2014). "Instruction Sets Should Be Free: The Case For RISC-V" Technical Report No. UCB/EECS-2014-146<http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-146.html>
- [3] **Rosenblatt, F.**(1958). "The perceptron: a probabilistic model for information storage and organization in the brain" Psychological Review 65 : 386-408
- [4] **B. Widrow and M. A. Lehr**, "30 years of adaptive neural networks: perceptron, Madaline, and backpropagation," in Proceedings of the IEEE, vol. 78, no. 9, pp. 1415-1442, Sept. 1990, doi: 10.1109/5.58323.
- [5] **A. Pedram, A. Gerstlauer and R. A. v. d. Geijn**, "A high-performance, low-power linear algebra core," ASAP 2011 - 22nd IEEE International Conference on Application-specific Systems, Architectures and Processors, Santa Monica, CA, 2011, pp. 35-42, doi: 10.1109/ASAP.2011.6043234.
- [6] **Jiao, Q., Hu, W., Wen, Y., Dong, Y., Li, Z., & Gan, Y.** (2020). "Design of a Convolutional Neural Network Instruction Set Based on RISC-V and Its Microarchitecture Implementation." In International Conference on Algorithms and Architectures for Parallel Processing (pp. 82-96). Springer, Cham.
- [7] **R. Banakar, S. Steinke, Bo-Sik Lee, M. Balakrishnan and P. Marwedel**, "Scratchpad memory: a design alternative for cache on-chip memory in embedded systems," Proceedings of the Tenth International Symposium on Hardware/Software Codesign. CODES 2002 (IEEE Cat. No.02TH8627), Estes Park, CO, USA, 2002, pp. 73-78, doi: 10.1145/774789.774805.
- [8] **RISC-V Foundation** (2019). The RISC-V Instruction Set Manual, 20190608-PrivMSU-Ratified
- [9] **Xilinx** (2020). Vivado Design Suite User Guide
- [10] **Digilent** (n.d) Arty A7 <https://reference.digilentinc.com/reference/programmable-logic/artty-a7/%20reference-manual>
- [11] **Digilent** (n.d) Nexys 4 <https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/%20reference-manual>
- [12] **Url-1** < <https://pulp-platform.org/publications.html> >, erişim tarihi 28.01.2021.

- [13] **Url-2** < <https://github.com/riscv/riscv-cores-list>>, erişim tarihi 28.01.2021.
- [14] **Url-3** <https://core-v-docs-verif-strat.readthedocs.io/projects/cv32e40p_um/en/latest/glossary.html>, erişim tarihi 28.01.2021.
- [15] **Url-4** < <https://github.com/pulp-platform/pulpino>>, erişim tarihi 28.01.2021.
- [16] **Url-5** < <https://hanxiao.io/2018/09/28/Fashion-MNIST-Year-In-Review/>>, erişim tarihi 29.01.2021.
- [17] **ZalandoResearch.**(2017). <https://github.com/zalandoresearch/fashion-mnist>
- [18] **Xiao, Han, Kashif Rasul, and Roland Vollgraf.** "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms." arXiv preprint arXiv:1708.07747 (2017).
- [19] **Url-6** < <http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/>>, erişim tarihi 29.01.2021.
- [20] **Url-7** < <https://github.com/heitorrapela/fashion-mnist-mlp>>, erişim tarihi 29.01.2021.
- [21] **Url-8** < https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html>, erişim tarihi 10.02.2021.
- [22] **Sadowski, P.** "Notes on Backpropagation"
<https://www.ics.uci.edu/~pjsadows/notes.pdf>
- [23] **Url-9** < <https://github.com/pulp-platform/pulpino>>, erişim tarihi 28.01.2021.
- [23] **Url-10** < https://github.com/pulp-platform/ri5cy_gnu_toolchain>, erişim tarihi 28.01.2021.

EKLER

EK 1. Perceptron Python Kodu

```
import numpy as np
import random

#object oriented model for perceptron
class Perceptron:

    #construction
    def __init__(self, numberOfTrainingPoints, numberOfTestPoints):
        # Random linearly separated data
        xA,yA,xB,yB = [random.uniform(-1, 1) for i in range(4)]
        self.V = np.array([xB*yA-xA*yB, yB-yA, xA-xB, xA, xB, yA])
        self.X = self.generate_points(numberOfTrainingPoints)
        self.training(numberOfTrainingPoints)
        self.Test = self.generate_points(numberOfTestPoints)
        self.inference(numberOfTestPoints)

    def generate_points(self, N):
        X = []
        for i in range(N):
            x1,x2,x3,x4,x5 = [random.uniform(-1, 1) for j in range(5)]
            x = np.array([x1,x2,x3,x4,x5,1])
            s = int(np.sign(self.V.T.dot(x)))
            X.append((x, s))
        return X

    def training(self, N):

        #initialize the weights
        self.weights = np.ones(6)

        #initialize the learning rate
        c = 0.5

        counter = 0
        iterations = 0
        while counter != N:
            # assigns zero to counter to avoid to conflict
```

```

counter = 0
iterations = iterations + 1
for i in range(N):

    var_v = self.weights.dot(self.X[i][0])

    #sign function as activation function
    var_y = int(np.sign(var_v))

    if(var_y == self.X[i][1]):
        counter = counter + 1
        self.weights = self.weights
    else:
        self.weights = self.weights + (1/2 * c * (self.X[i]
[1] - var_v)* self.X[i][0])
    print(self.weights)
    print(iterations)

def inference(self, N):

    hits = 0
    for i in range(N):
        var_v = self.weights.dot(self.Test[i][0])

        #sign function as activation function
        var_y = int(np.sign(var_v))

        if(var_y == self.Test[i][1]):
            hits = hits + 1
    print("number of hits:")
    print(hits)

p = Perceptron(25, 15) # give training and test number as param

```

EK 2. Adaline Python Kodu

```

f = []
f_inference = []
average_error = []

# yd set for training
for i in range(50):
    f.append(3*x1_training[i] + 2*np.cos(x2_training[i]))

```

```

    f[i] = f[i] / 5

#yd set for inference
for i in range(50):
    f_inference.append(3*x1_inference[i] + 2*np.cos(x2_inference[i]))
    f_inference[i] = f_inference[i] / 5

#sigmoid function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

#derivative of sigmoid
def der_sigmoid(x):
    return (sigmoid(x)*(1-sigmoid(x)))

#initializes the weight
weights = np.array([1,2,3])

def adaline_training():
    global weights
    c = 1.0

    iterations = 0
    while iterations != 1000:

        iterations = iterations + 1
        error = []
        for i in range(50):
            var_x = np.array([x1_training[i], x2_training[i], 1])
            var_v = np.transpose(weights).dot(var_x)
            #sigmoid as activation function
            var_y = sigmoid(var_v)
            e = f[i] - var_y
            error.append(e)

            #adaline learning rule
            weights = weights + c * (e * der_sigmoid(var_v) * var_x)

        print(weights)
        print(iterations)
        # average error track
        total = np.sum(error)
        total = total / 50
        average_error.append(total)
        print(total)
        if(np.absolute(total) < 0.1):
            break

def adaline_inference():

```

```

global weights

hits = 0
N = 50

for i in range(N):
    var_x = np.array([x1_inference[i], x2_inference[i], 1])
    var_v = np.transpose(weights).dot(var_x)

    #sigmoid function as activation function
    var_y = sigmoid(var_v)

    if (np.absolute(f_inference[i] - var_y) < 0.1):
        hits = hits + 1

print("number of hits:")
print(hits)

adaline_training()
adaline_inference()

```

EK 3. ÇKA Python Kodu

```

import numpy as np
import matplotlib.pyplot as plt

#import this module for the training and validation data set
import deneme1_berkay

# constants for the network
x_num = 3

f_cell_num = 15

s_cell_num = 7

out_num = 1

train_set_num = 300

validation_set_num = 100

```

```

iter_upper_limit = 1000

# must be added bias term, takes x_num+1 element and gives four outputs
# in this case
train_set = deneme1_berkay.V1

#validation set to test the network perf
validation_set = deneme1_berkay.V2

#learning rate
c = 0.4

#momentum term
m = 0.7

#error epsilon term
epsilon = 0.01

# weight matrice for first hidden layer, dim = fCell_num * x_num+1, plu
# s one for bias
f_weights_matrix = np.random.uniform(0.15, -
0.15, f_cell_num*(x_num+1)).reshape(f_cell_num,(x_num+1))
#f_weights_matrix = weights.firstW
old_f_weights_matrix = f_weights_matrix
firstW = f_weights_matrix

# weight matrice for first hidden layer, dim = fCell_num * x_num+1, plu
# s one for bias
s_weights_matrix = np.random.uniform(0.15, -
0.15, s_cell_num*(f_cell_num+1)).reshape(s_cell_num,(f_cell_num+1))
#s_weights_matrix = weights.secondW
old_s_weights_matrix = s_weights_matrix
secondW = s_weights_matrix

# weight matrice for first hidden layer, dim = fCell_num * x_num+1, plu
# s one for bias
out_weights_matrix = np.random.uniform(0.15, -
0.15, out_num*(s_cell_num+1)).reshape(out_num,(s_cell_num+1))
#out_weights_matrix = weights.outW
old_out_weights_matrix = out_weights_matrix
outW = out_weights_matrix

#sigmoid function
# takes vector and applies element-wise sigmoid function
def sigmoid(vec_x):

```

```

vec_y = []
for i in range(len(vec_x)):
    vec_y.append(1 / (1 + np.exp(-vec_x[i])))
return vec_y

def sigmoid_single(x):
    return (1 / (1 + np.exp(-x)))

#derivative of sigmoid
def der_sigmoid(vec_x):
    vec_y = []
    for i in range(len(vec_x)):
        vec_y.append(sigmoid_single(vec_x[i])*(1-
sigmoid_single(vec_x[i])))
    return vec_y

error_list = []

def mlp_training():

    global f_weights_matrix
    global s_weights_matrix
    global out_weights_matrix

    global old_f_weights_matrix
    global old_s_weights_matrix
    global old_out_weights_matrix

    iteration = 0

    while iteration < iter_upper_limit:

        iteration = iteration + 1

        sum_e = 0

        for i in range(train_set_num):

            #first layer
            f_v = f_weights_matrix.dot(np.concatenate([train_set[i][0],
[1]]))

            f_y = sigmoid(f_v)

            #second layer
            s_v = s_weights_matrix.dot(np.concatenate([f_y, [1]]))

            s_y = sigmoid(s_v)

```



```

#output layer
out_v = out_weights_matrix.dot(np.concatenate([s_y, [1]]))

out_y = sigmoid(out_v)
# end of forward-propagation

#error vector
vec_e = np.subtract(train_set[i][1], out_y)#, train_set[i][
2]])

# sum_e = sum_e + (vec_e**2)
sum_e = sum_e + np.absolute(vec_e)

### back-propagation

# local gradients

out_gradient = der_sigmoid(out_v)

out_sigma = np.multiply(vec_e, out_gradient)

s_gradient = der_sigmoid(s_v)

s_sigma = np.transpose(out_weights_matrix[:,s_cell_num]).dot(out_sigma) * s_gradient

f_gradient = der_sigmoid(f_v)

f_sigma = np.transpose(s_weights_matrix[:,f_cell_num]).dot(s_sigma) * f_gradient

# to keep old weight values
temp_out_weights_matrix = out_weights_matrix
temp_s_weights_matrix = s_weights_matrix
temp_f_weights_matrix = f_weights_matrix

# update weights via momentum
out_weights_matrix = out_weights_matrix + c * out_sigma.reshape((out_num,1)).dot(np.concatenate([s_y, [1]]).reshape((1,s_cell_num+1))) + m * (out_weights_matrix - old_out_weights_matrix)

s_weights_matrix = s_weights_matrix + c * s_sigma.reshape((s_cell_num,1)).dot(np.concatenate([f_y, [1]]).reshape((1,f_cell_num+1))) + m * (s_weights_matrix - old_s_weights_matrix)

```

```

        f_weights_matrix = f_weights_matrix + c * f_sigma.reshape((
f_cell_num,1)).dot(np.concatenate([train_set[i][0], [1]]).reshape((1,x_
num+1))) + m * (f_weights_matrix - old_f_weights_matrix)

        # to keep old weight values
        old_out_weights_matrix = temp_out_weights_matrix
        old_s_weights_matrix   = temp_s_weights_matrix
        old_f_weights_matrix   = temp_f_weights_matrix
        sum_e = sum_e / train_set_num
        error_list.append(sum_e)
        print(iteration)
        print(sum_e)
        if(all(x < epsilon for x in sum_e)):
            break
y_vec = []
def mlp_inference():

    global f_weights_matrix
    global s_weights_matrix
    global out_weights_matrix
    global inf_out_y
    hits = 0

    for i in range(validation_set_num-1):
        print(i)
        #first layer
        f_v = f_weights_matrix.dot(np.concatenate([validation_set[i
]][0], [1]]))

        f_y = sigmoid(f_v)

        #second layer
        s_v = s_weights_matrix.dot(np.concatenate([f_y, [1]]))

        s_y = sigmoid(s_v)

        #output layer
        out_v = out_weights_matrix.dot(np.concatenate([s_y, [1]]))

        inf_out_y = sigmoid_single(out_v)
        # end of forward-propagation
        y_vec.append(inf_out_y)
        #error vector
        vec_e = np.absolute(np.subtract(validation_set[i][1], inf_o
ut_y))#, train_set[i][2]])

        # sum_e = sum_e + (vec_e**2)
        sum_e = sum_e + np.absolute(vec_e)

```

```

        error = sum_e / validation_set_num

        print("error", error)
        print(firstW)
        print(secondW)
        print(outW)

mlp_training()
plt.plot(error_list)
plt.xlabel("#epochs")
plt.ylabel("average error per output")
plt.title("Error vs Epochs")
plt.show()
mlp_inference()
# plt.plot(y[:300])
# plt.show()

plt.plot(deneme1_berkay.y[300:400], color="red", marker="o")
plt.plot(y_vec, color="blue", marker="x")
plt.show()

```

EK 4. ÇKA C KODU

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <stdint.h>
5  #include <string.h>
6
7  #define SIZE 16
8  #define input_number 784
9  #define output_number 10
10 // Constants for the network
11 #define x_num 28*28
12 #define f_cell_num 16
13 #define s_cell_num 16
14 #define out_num 10
15 #define train_set_num 60000
16 #define validation_set_num 100
17 #define iter_upper_limit 10
18
19 #static float validation_set[100][input_number] = { ... }
119
120
121 void get_err(float *arr, size_t size) {
122     size_t i = 0;
123
124     for (i; i < size; ++i) {
125         arr[i] /= validation_set_num;
126     }
127 }
128
129 void matrix_add(float *pdest, float *psource, size_t size) {
130     size_t i = 0;
131
132     for (i; i < size; ++i) {
133         pdest[i] += psource[i];
134     }
135 }
136
137
```

```

136
137
138 void set_err_vec(float *pdest, float *yd_test, float *out_y, size_t size) {
139     size_t i = 0;
140
141     for (i; i < size; ++i) {
142         pdest[i] = fabs(yd_test[i] - out_y[i]);
143     }
144 }
145
146
147 void get_max_idx(int *max1_idx, int *max2_idx, float *p1, float *p2, size_t size) {
148     *max1_idx = 0;
149     *max2_idx = 0;
150
151     int i = 0;
152     for (i = 0; i < size; ++i) {
153         if (p1[i] > p1[*max1_idx]) {
154             *max1_idx = i;
155         }
156         if (p2[i] > p2[*max2_idx]) {
157             *max2_idx = i;
158         }
159     }
160 }
161
162
163 void adding_bias(float *pdest, float *psource, size_t size) {
164
165     size_t i = 0;
166
167     for (i; i < size; ++i) {
168         pdest[i] = psource[i];
169     }
170
171     pdest[i] = 1.0;
172 }
173

```

```

173
174 void copy_arr(float *pdest, float(*psource)[input_number], size_t size, int i) {
175     size_t j = 0;
176
177     for (j; j < size; ++j) {
178         pdest[j] = psource[i][j];
179     }
180     pdest[j] = 1.0;
181 }
182
183
184 void sigmoid(float *pdest, float *psource, size_t size) {
185     while (size--) {
186         *pdest = 1 / (1 + exp(-(*psource)));
187         ++pdest;
188         ++psource;
189     }
190 }
191
192 void dot_product_785(float *pdest, float(*psource1)[input_number + 1], float *psource2, size_t size) {
193     size_t i = 0;
194     size_t j = 0;
195     size_t k = 0;
196     float sum = 0;
197
198     for (i; i < size; ++i) {
199         for (j = 0; j < input_number + 1; ++j) {
200             sum += psource1[i][j] * psource2[j];
201         }
202         pdest[i] = sum;
203         sum = 0;
204     }
205 }
206

```

```

206
207 void dot_product_17(float *pdest, float(*psource1)[SIZE + 1], float *psource2, size_t size) {
208     size_t i = 0;
209     size_t j = 0;
210     size_t k = 0;
211     float sum = 0;
212
213     for (i; i < size; ++i) {
214         for (j = 0; j < SIZE + 1; ++j) {
215             sum += psource1[i][j] * psource2[j];
216         }
217         pdest[i] = sum;
218         sum = 0;
219     }
220 }
221
222 static float f_weights_matrix[SIZE][input_number + 1] = { ... }
223
224 static float s_weights_matrix[SIZE][SIZE + 1] = { ... }
225
226 static float out_weights_matrix[output_number][SIZE + 1] = { ... }
227
228 float yd_test[12] = { 0 };
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247

```

```

271 int main() {
272
273     float sum_e[output_number];
274     uint16_t hits = 0;
275     int i;
276
277     for (i = 0; i < validation_set_num; ++i) {
278
279         //first layer
280         float f_v[SIZE];
281         float validation_set_new[input_number + 1];
282         copy_arr(validation_set_new, validation_set, input_number, i);
283         dot_product_785(f_v, f_weights_matrix, validation_set_new, SIZE); //concanacate yap
284         float f_y[SIZE];
285         sigmoid(f_y, f_v, SIZE);
286
287         //second layer
288
289         float s_v[SIZE];
290         float f_y_new[SIZE + 1];
291         adding_bias(f_y_new, f_y, SIZE);
292         dot_product_17(s_v, s_weights_matrix, f_y_new, SIZE);
293         float s_y[SIZE];
294         sigmoid(s_y, s_v, SIZE);
295
296         //output layer
297
298         float out_v[output_number];
299         float s_y_new[SIZE + 1];
300         adding_bias(s_y_new, s_y, SIZE);
301         dot_product_17(out_v, out_weights_matrix, s_y_new, output_number);
302         float out_y[output_number];
303         sigmoid(out_y, out_v, output_number);
304
305         //end of forward-propagation
306
307         int out_y_max_idx = 0;
308         int yd_test_max_idx = 0;
309
310         get_max_idx(&out_y_max_idx, &yd_test_max_idx, out_y, yd_test, output_number);
311

```

```

311
312
313     if (out_y_max_idx == yd_test_max_idx) {
314         hits += 1;
315     }
316
317     // error vector
318     float vec_e[output_number];
319     set_err_vec(vec_e, yd_test, out_y, output_number);
320
321     // # sum_e = sum_e + (vec_e**2)
322     matrix_add(sum_e, vec_e, output_number);
323
324     printf("%d. = %f\n", i, out_y[out_y_max_idx]);
325
326     memset(validation_set_new, 0, (input_number + 1));
327 }
328 float error[output_number];
329 get_err(error, output_number);
330
331
332
333 return 0;
334 }
335

```

ÖZGEÇMİŞLER

Ad Soyad: Nazım Altar KOCA

Doğum yeri ve tarihi: Yozgat, 26.08.1998

E-mail: nazim-altarkoca@gmail.com

Nazım Altar Koca İstanbul Teknik Üniversitesi Elektronik ve Haberleşme Mühendisliği son sınıf öğrencisidir. Stajlarını İTÜ bünyesindeki Teknokent/Çekirdek, TÜBİTAK BİLGEM TÜTEL ve YONGATEK şirketlerinde yapmıştır.



Ad Soyad: Berkay YILDIZ

Doğum yeri ve tarihi: Tokat, 23.01.1998

E-mail: berkayyildiz77@gmail.com

Berkay YILDIZ İstanbul Teknik Üniversitesi Elektronik ve Haberleşme Mühendisliği son sınıf öğrencisidir. Stajlarını İTÜ bünyesindeki İTÜ GSTL(Gömülü Sistem Tasarımı Laboratuvarı), TÜBİTAK BİLGEM ve ASELSAN şirketlerinde yapmıştır. Şuanda ULAK Haberleşme şirketinde yazılım mühendisi olarak çalışmaktadır.



Ad Soyad: Yusuf Caner DEMİRKOL

Doğum yeri ve tarihi: İzmit, 05.03.1997

E-mail: canerdmrkl@gmail.com

Yusuf Caner Demirkol İstanbul Teknik Üniversitesi Elektronik ve Haberleşme Mühendisliği son sınıf öğrencisidir. Stajlarını İTÜ bünyesindeki İTÜ GSTL(Gömülü Sistem Tasarımı Laboratuvarı), YONGATEK ve TÜBİTAK BİLGEM şirketlerinde yapmıştır. Şuanda ULAK Haberleşme şirketinde yazılım mühendisi olarak çalışmaktadır.

