Appunti di Sistemi Digitali D

Sas

Filippo Lenzi e Guglielmo Palaferri

14 febbraio 2022

Indice

Capitolo 1

Intro

Versione attuale del doc: trascrizione videolezioni circa, da mettere meglio a posto in seguito

1.1 22 settembre 2021

Scopo del corso: introduzione a sistemi embedded

Diverse famiglie: risorse limitate con meno consumo, o viceversa

Metodologie di sviluppo per questi dispositivi

Più vicino a hardware con linguaggi di basso o alto livello, o anche smartphone con linguaggi di più alto livello

Si userà Python, linguaggio ad oggi più utilizzato (doubt)

In contesto anche applicativo: usato in applicazioni reali

Modulo 1: studio dispositivi (come FPGA) su cui si possono usare linguaggi di basso livello, vengono usati nel corso framework di più alto livello come HLS, che usa variante di C/C++, non Verilog e VHDL che sarebbero molto più vicini all'hardware

Inoltre si definiscono modalità di calcolo / progetto del sistema stesso in modulo 1

Modulo 2: come usare architetture esistenti, e anche board con bridge in Python per manipolare hardware sottostante

Python è interpretato ed è meno performante di C, avendo più overhead

Info esame

esame MASSIMO 2 VOLTE

Sistemi Embedded

Sistemi digitali includono tutti i dispositivi a ogni livello di consumo; i **sistemi embedded** sono compatti e a basso consumo energetico; quindi, la complessità delle funzioni da svolgere è un fattore limitante, dovendo spesso gestire una batteria limitata, e evitare altri problemi come surriscaldamento in base al caso d'uso.

Board: scheda con diversi accessori, con un piccolo dispositivo centrale. Più specializzata, architettura progettata ad hoc per la funzione. Hanno comunque delle librerie per interfacciare con accessori in usecase comuni (fotocamere, usb, ecc.).

Alcuni esempi di dispositivi:

- Robot di pulizia casalinga, con software di navigazione in base a sensori.
- Occhiali per persone ipovedenti, con fotocamera che elabora informazioni per fornire info su semafori, testo e altro in forma audio, che necessita di lunga autonomia e limitazione del calore.
- Droni contengono sistema embedded per ricevere il controllo remoto e tradurlo in movimento delle eliche.
- Sblocco del telefono con fotocamera: gestione batteria, fotocamera deve rimanere in "semi-idle" per poter rilevare volto, senza consumare troppa batteria.

Discorso di bilanciamento tra ottimizzare tempo performance mettendo cose in memoria, consumando però più memoria

Computer vision: elaborazione immagini. La mole di dati da elaborare può essere molto grande, essendo matrici. Spesso basato su machine learning. Esiste hardware dedicato al processamento di immagini.

Elaborazione di stream video: mole di dati dipende da framerate oltre che da risoluzione

Due strategie viste nel corso per elaborare video/immagini:

- Modulo 1: Board con ARM + FPGA, paradigma di programmazione classico di basso livello, anche con astrazione via Python in certi casi. Architettura progettata apposta per il problema. Non general purpose, programmazione più complessa.
- Modulo 2: Architettura hardware già progettata e immutabile (smartphone, GPU), con programmazione e tool di più alto livello, più simili a PC desktop. Controllo sull'hardware limitato da API sul dispositivo, e quindi limite alla complessità dei programmi dettato da esse.

1.2 24 settembre 2021

GPU è progettata per velocizzare operazioni grafiche di processori esistenti; ARM e FPGA costituiscono un sistema integrato con una acceleratore progettato apposta per essa, e con la quale si scambia informazioni su una linea molto rapida, consumando meno energia di una GPU general purpose.

Alcune problematiche attuali

Un telefono di fascia medio-alta spesso ha 3-5 fotocamere: questa visuale stereo può essere usata per ottenere diverse informazioni, per esempio una mappa della distanza di ogni pixel dalla telecamera, o il riconoscimento semantico di diversi oggetti. Questi problemi hanno un consumo di risorse anche alto, e non necessariamente una rete pensata per un contesto funzionerà bene in altri.

L'analisi della profondità ha molti usi, anche semplicemente il calcolo dell'occlusione (oggetti reale davanti che nasconde oggetto virtuale dietro) per realtà aumentata.

Molti casi comuni sono pensati per essere eseguiti su GPU con consumi molto alti e quindi non adatti a sistemi embedded. Lo scopo è avere reti utilizzabili su sistemi embedded con prestazioni comparabili (sia framerate/velocità, che accuratezza e qualità).

Esistono framework di ottimizzazione di sistemi per dispositivi fatti da diversi produttori (come Apple e Intel): per esempio, un'ottimizzazione comune è evitare float e usare interi.

Esempio: adattamento di rete pensata per GPU a FPGA e telefono. Lo scopo è quindi usare questi framework per ottimizzare reti (soprattutto di Deep Learning) a dispositivi a basso consumo; per algoritmi fatti dal programmatore, l'ottimizzazione verte più su di lui.

Un sistema basato su deep learning richiede calcoli anche pesanti, e ha un problema in base alla disponibilità dei dati di training: se allenato su un contesto specifico, non sarà in grado di operare su altri contesti. Si cercano di creare in quel caso sistemi che apprendono dai nuovi dati, aggiustando parametri della rete in base a nuovi ambienti.

Elaborazione stream video: milioni di pixel su schermo, ognuno elaborabile in diverso modo, e quindi originalmente GPU nate per questo, ma si presta bene anche per reti neurali.

Processori ARM hanno minori consumi, inizialmente usati per telefoni, poi introdotti a PC date le prestazioni comparabili.

Oggi la frequenza di clock di una CPU è meno importante, visto che comporta consumi più elevati. Spesso la frequenza è modulata in base al carico ad oggi.

Alcuni esempi di architettura a consumo ridotto:

- Movidius Intel: acceleratore per immagini, una sorta di micro-GPU, presente su telefoni e anche in formato USB.
- JevoIs Smart Machine Vision: anche esso molto piccolo, contiene processore ARM, con telecamera integrata, lunga autonomia.
- Jetson Nvidia: sistema con piccola GPU.
- TPU Google: accelera calcoli di reti machine learning, pure qua ARM. Usato da loro per calcoli correlati invece dei server general purpose, per cose come il riconoscimento vocale ecc. TPU sta per Tensor Processing Unit, è una architettura custom per massimizzare la performance nei datacenter.
- Brainwave di Microsoft: non architettura custom completamente, ma utilizzando board con FPGA con design originale della rete. Usato non per training, ma per inferenza di reti.

Gli esempi precedenti non sono programmabili a livello di architettura, solo nel software. Negli ultimi anni diversi produttori hanno reso disponibili dei chip, basati su una serie di processori (solitamente ARM) e un FPGA, che permettono di definire qualunque architettura. La CPU permette di fare qualunque task, anche se non al meglio, mentre la FPGA è totalmente progammabile per ottenere uno scopo specifico, accelerando i calcoli. Addirittura, è possibile definire delle architetture in modo automatico a partire da

una descrizione fornita in C o C++. Lato negativo è quindi richiedere più tempo per definire l'architettura, rispetto ai sistemi già configurati dei vari produttori.

RISC-V: serie di istruzioni open-source per CPU RISC, supportato da molte aziende (a parte ARM). Set di istruzioni libero ed estendibile, con un comitato affinchè l'ISA non cambi. ARM come azienda vende progetti, non hardware, anche a aziende rivali. Il suo possibile acquisto da parte di NVIDIA sta aumentando l'interesse in RISC-V.

Altri sistemi embedded con architettura non programmabile, linguaggio spesso Python: raspberry, jetson Nvidia, smartphone, ecc. Con Python, il framework di machine learning più comune è Tensorflow di Google.

Sistemi programmabili: un esempio trattato nel corso architettura ZYNQ prodotta da Xilinx, appunto con dual core ARM e un FPGA, collegati tramite bus ad alta efficienza AXI; possiede inoltra memoria condivisa tra CPU e FPGA. Su slide presenti le caratteristiche tecniche (PS = Processing System) della CPU. Usa sistema operativo Linux di base, nel corso si usa OS "bare metal" cioè senza protezioni, molto basilare. All'avvio una delle CPU decide quale bitstream usare per configurare FPGA. Altra architettura più recente Ultrascale, con prestazioni elevate e una simile idea generale di progettazione. Infine, la PYNQ, simile a ZYNQ; si crea l'architettura con linguaggi di alto livello (C) come con ZYNQ, e su di essa si può scrivere codice con Python. Queste board sono simili a quelle basate su Xilinx-700 (ZedBoard), ma si può usare un linguaggio di alto livello.

È possibile simulare queste board su software, nel caso non si posseggano.

Su slide un esempio di progetto su ZYNQ.

In conclusione: Modulo 1 basato su architetture custom, Modulo 2 strumenti di ottimizzazione per sfruttare architetture esistenti.

Capitolo 2

Modulo 1

Capitolo 3

Modulo 2

3.1 06 dicembre 2021

Introduzione al Deep Learning

Vedremo una panoramica sul mondo del deep learning, in particolare dal punto di vista dello sviluppatore, senza soffermarsi troppo sui dettagli teorici.

Due framework principali: Tensorflow e PyTorch.

Due strade possibili:

- utilizzare una rete neurale già addestrata, integrandola all'interno del proprio progetto software, oppure
- addestrare autonomamente una rete neurale per poi andarla ad utilizzare

Per deep learning si intende una famiglia di metodi basati su reti neurali che tramite l'apprendimento da insiemi di dati etichettati possono consentire di risolvere problemi generalizzati su un qualsiasi insieme di dati.

Teorizzati inizialmente negli anni '80, hanno trovato ampia diffusione solo in tempi recenti, quando è stato possibile soddisfare due requisiti fondamentali per l'utilizzo di reti neurali:

- Elevata potenza di calcolo
- Grandi quantità di dati per l'addestramento

Le reti neurali convoluzionali (CNNs) sono tra i framework più popolari, consentono di risolvere problemi di classificazione e regressione nell'ambito dell'elaborazione delle immagini. Due fasi: Training e Testing

Una rete neurale è definita da una sequenza di moduli (livelli), ciascuno caratterizzato da un insieme di pesi (ognuno associato ad un neurone). I pesi definiscono il comportamento di ogni modulo e di conseguenza della rete nella sua interezza. Durante la fase di Training, la rete processa il dato fornito in input (nel nostro caso un'immagine) e predice un risultato. Tale risultato viene confrontato con l'etichetta reale assegnata all'immagine. Misurando la differenza tra il valore predetto dalla rete e l'etichetta possiamo

quantificare l'errore ed ottimizzare la rete per minimizzarlo, in particolare aggiustando i valori dei vari pesi della rete.

La sequenza di moduli che definiscono la rete può essere percorsa in avanti (**forward pass**) per ottenere l'output a partire dall'input, oppure all'indietro (**backward pass**) per ottenere l'espressione dell'uscita in funzione dei pesi della rete (poiché l'uscita è funzione composta delle uscite dei singoli moduli).

A livello matematico, la differenza tra il risultato della rete e l'etichetta viene modellata tramite una funzione L (*loss function*), calcolata come la differenza tra l'output (funzione degli N pesi della rete) e l'etichetta. La differenza viene minimizzata muovendo i pesi in direzione opposta rispetto al gradiente della funzione L.

espressione matematica del peso θ_j

Framework per il deep learning quali Tensorflow e PyTorch gestiscono automaticamente l'aggiornamento dei pesi nella fase di training della rete.

esempi di problemi risolvibili con una CNN: object detection, instance segmentation, depth estimation ecc.

Vedremo ora il funzionamento dei due framework citati, mostrando infine come sia possibile eseguire il porting su dispositivi mobili (in particolare Android).

Tensorflow

Sviluppato da Google, esistono due versioni principali: 1.x e 2.x. In Tensorflow 1.x, lo sviluppo del codice era diviso in due fasi: costruzione del grafo (trainig) ed esecuzione (testing). In Tensorflow 2.x al contrario, non è prevista la separazione tra le due fasi.

Tensorflow si basa sul concetto di **tensore**, ovvero un array multidimensionale: array (1-D), matrici (2-D), array di matrici (3-D) ecc. Possiamo pensare ad un'immagine come ad un tensore 3-D (altezza, larghezza e canale RGB).

La struttura della rete viene rappresentata tramite un **grafo** unidirezionale, il quale rappresenta l'esatta successione dei layer. Questo consente in modo efficiente sia di percorrere il grafo per ottenere l'output in fase di forward pass, sia di recuperare le dipendenze tra i diversi layer in fase di backward pass. In questo modo si modella la rete tramite un'astrazione ad alto livello.