Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/02BE04Mitrovic

RAZVOJNO OKRUŽENJE ZA PROJEKTOVANJE I GENERISANJE KODA NEURONSKIH MREŽA

NEURAL NETWORK SOFTWARE DEVELOPMENT ENVIRONMENT WITH CODE GENERATION CAPABILITIES

Aleksandra Mitrović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu je predstavljeno razvojno okruženje za kreiranje modela neuronskih mreža i generisanje koda na osnovu istih. Takođe, objašnjen je i grafički DSL koji se koristi za izradu modela i prikazan je primer upotrebe okruženja.*

Ključne reči: *Neuronske mreže, generator koda, DSL, IDE.*

Abstract – *This paper presents a neural network model development environment with code generation capabilities. This paper also describes a DSL used for specifying neural network model and shows an example of its use.*

Keywords: *Neural networks, code generator, DSL, IDE.*

1. UVOD

Upotreba mašinskog učenja za rešavanje kompleksnih problema je najčešći izbor u poslednjih nekoliko godina. Popularnost ovog pristupa je značajno porasla u poslednjoj deceniji kada su računarske komponente postale dostupnije i efikasnije. Jedan od često korišćenih tipova mašinskog učenja predstavljaju neuronske mreže, čiji je domen upotrebljen za kreiranje jezika specifičnog za domen (eng. DSL). Cilj ovog jezika jeste da omogući brže kreiranje modela ekspertima, ali isto tako da i onim korisnicima koji tek zalaze u domen neuronskih mreža omogući kreiranje modela bez potrebe za poznavanjem nekog programskog jezika.

Kako arhitektura neuronskih mreža predstavlja usmeren graf, gde čvorovi predstavljaju različite operacije koje se sprovode nad ulaznim podacima, a veze predstavljaju prosleđivanje izlaza jedne operacije na ulaz drugih, kreiranje grafičkog DSL-a bi doprinelo boljem razumevanju toka podataka i redosledu primene operacija. Vizuelizacija jezika je omogućena u već postojećem rešenju FLCT [1] koji predstavlja razvojno okruženje za projektovanje i generisanje koda fazi logičkih kontrolera. Ovo okruženje je za DSL iz domena fazi logičkih kontrolera omogućilo vizuelizovano kreiranje šeme kontrolera, validaciju šeme, kao i generisanje koda spram zadate šeme. Rezultirajući kod se može upotrebiti za neki drugi program ili direktno postaviti na kontroler nekog fizičkog uređaja.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bila dr Gordana Milosavljević, van.prof.

Uključivanjem još jednog atraktivnog domena u ovo okruženje otvara mogućnost kombinovanja dva jezika, jer su bazirani na osnovnom modelu jezika, gde se dodaju samo domenske specifičnosti.

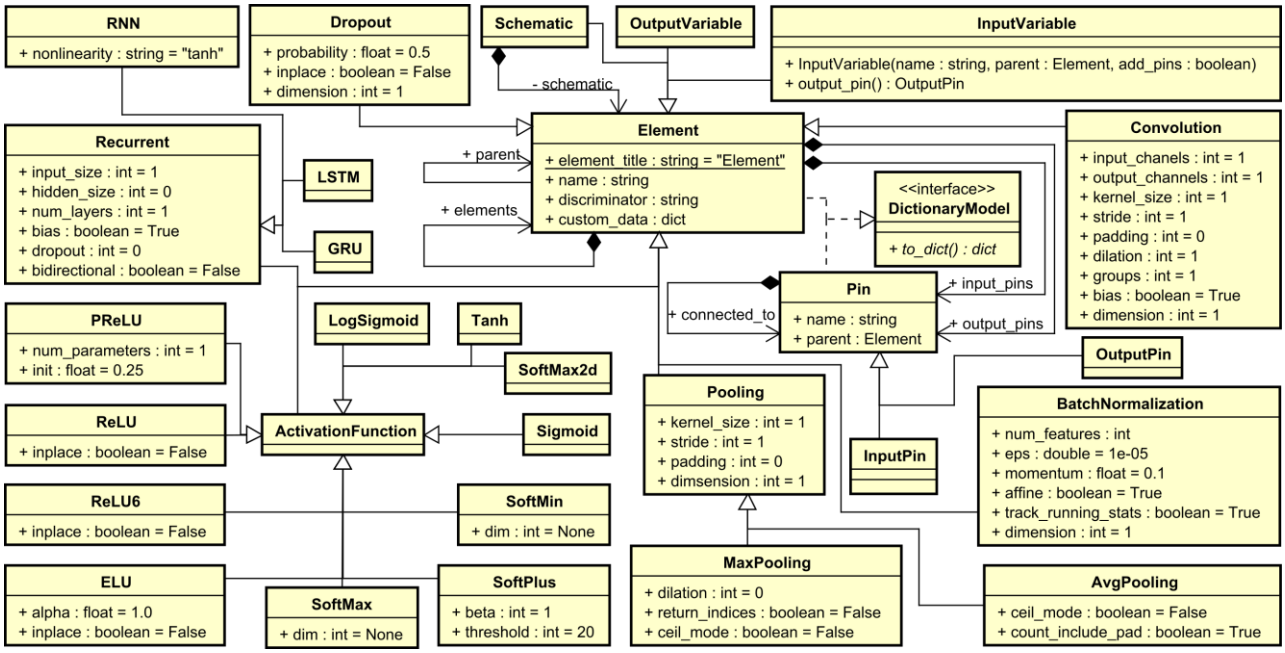
2. DSL ZA NEURONSKE MREŽE

Za formiranje ovog jezika u obzir je uzeta struktura i model prethodnog jezika za fazi logičke kontrolere, ali i biblioteka za rad sa neuronskim mrežama PyTorch [2].

2.1. Gramatika i činioci jezika

Za formiranje jezika koji se oslanja na strukturu grafa, potrebno je definisati osnovne pojmove koji odgovaraju čvoru i vezi. U ovom jeziku to predstavljaju *Element* i *Pin* redom. Uvođenjem termina iz neurosnkih mreža koji predstavljaju operacije se kreiraju specijalizacije *Element*-a. Dok za *Pin* postoje samo dve specijalizacije koje bliže određuju da li se podaci prosleđuju elementu ili se prosleđuju iz elementa. Pomoću ovog jezika moguće je formirati hijerarhijsku strukturu, i imati ugnježdene elemente. Na vrhu hijerarhije se nalazi *Shematic* (srp. šema) koja predstavlja forward-pass modula neuronske mreže. Za dobavljanje podataka i isporuku koriste se *InputVariable* i *OutputVariable* (srp. ulazna i izlazna varijabla), koje predstavljaju specijalizacije *Element*-a. Činioce jezika, pored osnovnih koji su prethodno navedeni, predstavljaju i:

- aktivacione funkcije – služe za određivanje pobuđenosti neurona na osnovu ulaza, u ovoj verziji jezika podržane su samo neke, često korišćene, aktivacione funkcije;
- konvolucioni slojevi – primenjuju konvolutivnu neuronsku mrežu na ulazne podatke i rezultat isporučuju na izlaz elementa;
- dropout slojevi – služe za regulazaciju ulaznih podataka, koji u procesu obučavanja sprečavaju preobučenost;
- linearni slojevi – primenjuju linearnu transformaciju ulaznih podataka;
- slojevi za normalizaciju – vrše mapiranje ulaznih vrednosti na prethodno definisan skup vrednosti, pomaže u obučavanju kod previše velikih ili previše malih vrednosti i njihovih oscilacija;
- pooling slojevi – služe za redukciju dimenzionalnosti uzlaznih podataka, i
- rekurentni slojevi – primenjuju rekurentnu neuronsku mrežu na ulazne podatke.



Slika 1. *Dijagram klasa modela*

Svi prethodno navedeni činioci predstavljaju operacije koje se izvršavaju na ulaznim podacima, dobijenih na ulazni pin elementa, a rezultat primene date operacije, prosleđuju na izlazni pin elementa. Na slici 1. dat je model činilaca jezika.

2.2. Model

Uz oslonac na prethodni DSL je kreiran i DSL za neuronske mreže. Generičke činioce poput *Element*, *Pin*, *InputVariable*, *OutputVariable* i *Schematic* su izdvojene kao zajedničke i predstavljaju osnovu za svaku drugu verziju DSL-a.

Kako je ideja ovog jezika bila kreiranje modula neuronske mreže za PyTorch biblioteku, i sami elementi su kreirani u skladu sa tim da podrže operatore iz ove biblioteke i njoj sličnih. Svaki *Element* sadrži sledeće podatke:

- name – predstavlja naziv datog elementa, služi za imenovanje varijable u programskom kodu, ukoliko se generiše,
- parent – definiše hijerarhiju uvođenjem roditeljskog elementa,
- elements – predstavlja kolekciju dece datog elementa,
- discriminator – omogućava tipizaciju elemenata,
- custom_data – definiše druge metapodatke koji će dopuniti svojstva elementa,
- input_pins – predstavlja kolekciju svih ulaznih pinova,
- output_pins – predstavlja kolekciju svih izlaznih pinova,
- element_title – predstavlja naslov čvora, služi za vizuelizaciju i najčešće opisuje tip čvora.

Pin predstavlja konektor elementa, i omogućava mu da se kroz njega dostave ili isporuče podaci.

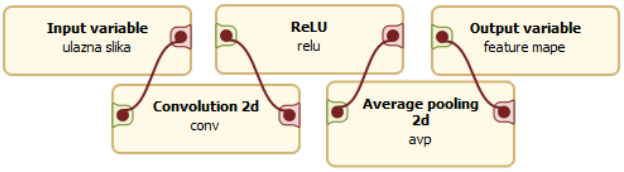
Atributi ove klase su *name*, *parent*, čije je značenje isto kao i kod *Elementa*, i *connected_to* koji predstavlja kolekciju pinova sa kojima je dati pin povezan.

Ove dve klase implementiraju *DictionaryModel* i redefinišu metodu *to_dict* koja treba da omogući serijalizaciju objekata ovih klasa radi perzistencije.

Još jednu specifičnu klasu predstavlja *Schematic* koja predstavlja arhitekturu neuronske mreže, odnosno forward-pass modula. Takođe, ima i ulaznu i izlaznu varijablu u kojoj dostavlja ulazne podatke ostalim elementima šeme, a rezultat primene svih operacija čuva u izlaznoj varijabli. Ostale klase, predstavljaju specijalizacije elementa i specifične operacije u domenu neuronskih mreža. Detaljnije objašnjenje se može pronaći u master radu.

2.3. Implementacija

Konkretna sintaksa jezika kreirana je po uzoru na prethodni jezik za fazi logiku, odnosno fazi logičke kontrolere. Tehnologije upotrebljene za kreiranje te sintakse su *Python* programski jezik, *PySide* biblioteka. Za ovu verziju aplikacije i samog jezika upotrebljena je nova verzija *PySide2* [3] i izvršena je neophodna migracija. Za vizuelizaciju ovog modela napravljena je wrapper klasa *SchematicModel* koja je kreirana kao *editable model* po smernicama Qt-a [4]. Na slici 2. dat je primer vizuelizacije odabranih elemenata jezika.



Slika 2. *Primer šeme*

Ova slika prikazuje čvorove, odnosno operacije, u obliku zaobljenih pravougaonika ispunjenih krem bojom. Pinovi se nalaze sa strana i na obodu čvora, i u zavisnosti da li je ulazni ili izlazni ima zelenu ili crvenu boju, redom. Veze između tih čvorova formiraju se u ulaznom i izlaznom pinu i predstavljene su kubnom linijom bordo boje.

Podebljan tekst u čvoru predstavlja njegov naslov, a tekst ispod naslova predstavlja naziv čvora. U zavisnosti od pravila jezika, čvorovi mogu imati više od jednog pina. Ovaj jezik je formiran tako da podaci uvek idu od izlaznog ka ulaznom pinu nekog drugog elementa.
Za svaki čvor na šemi moguće je vršiti izmenu boje ispunjenosti elementa, kao i izmenu boje bordure. Naslov elementa, odnosno čvora, je fiksan, ali je naziv, kao i ostale specifične parametre, moguće promeniti.

4. APLIKACIJA

FLCT je aplikacija koja predstavlja okruženje za generisanje šema i koda na osnovu šeme fazi logičkog kontrolera [1]. Aplikacija je proširiva, uz pomoć plug-in framework-a koji je kreiran sa ciljem da bude minimalistički.

Proširenje aplikacije se trenutno ogleda u dodavanju novih generatora koda, ali proširenje može biti izvršeno i dodavanjem novih editora i drugih komponenti, što je u planu za dalji razvoj aplikacije. Grafički korisnički interfejs (GUI) je kreiran u *Python* programskom jeziku i uz korišćenje *PySide2* biblioteke.

Prikaz aplikacije dat je na slici 3. Generisanje koda na osnovu šeme omogućeno je uz pomoć biblioteke *Jinja2* [5].

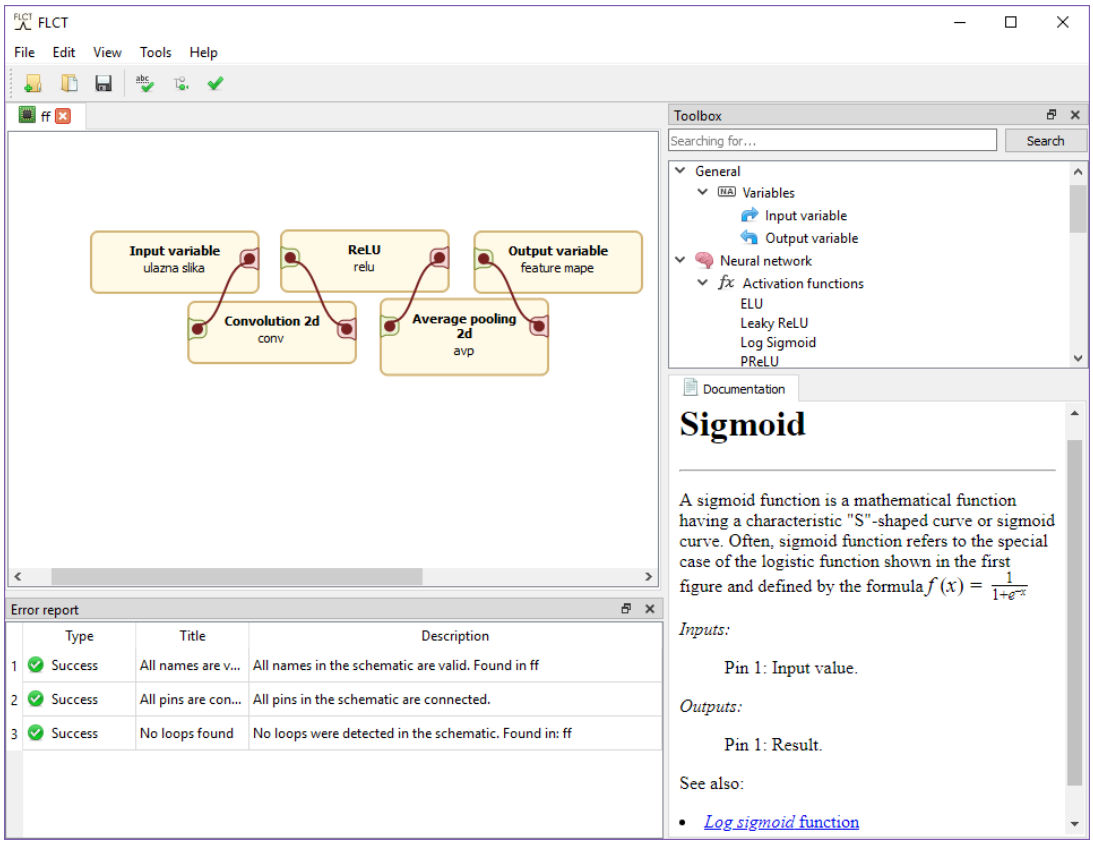
Kreiranje šeme vrši se odabirom lokacije i naziva datoteke u kojoj će se šema čuvati. Populisanje novokreirane šeme vrši se primenom *drag&drop* operacija sa palete alata, koja se nalazi u *toolbox-u*, sa desne strane editora šeme. Izmena podataka o elementu vrši se dvoklikom na željeni element, nakon čega se prikazuje dijalog sa svim parametrima koje ima dati element, a varira u zavisnosti od konkretnog tipa elementa.

U procesu izmene čvora, moguće mu je prilagoditi i izgled, izmenom boja za ispunu čvora i oboda. Kreiranje se može izvršiti i uz pomoć prethodno definisanih primera, koji prestavljaju poznate arhitekture neuronskih mreža, i korisne su prilikom manjih izmena istih ili kod sagledavanja strukture, za kreiranje nove.

U aplikaciji je omogućena validacija šeme na osnovu zadatih pravila, što u slučaju ovog jezika predstavlja jedinstvenost imenovanja, isključivanje petlji, povezanost svih pinova i postojanje ulaznih i izlaznih varijabli. Validaciju je moguće vršiti samo za čvorove ili samo za veze. Takođe, u opciji je i potpuna validacija koja uključuje prethodno navedene validacije.

Još jednu funkcionalnost predstavlja opciono generisanje koda na osnovu šeme. Preduslov za generisanje predstavlja kompletnu ispravnost šeme na osnovu koje se želi generisati kod. Prilikom generisanja koda, vrši se odabir ciljnog programskog jezika.

U toku generisanja vrši se popunjavanje predefinisanih *Jinja2* šablona i kreira se odgovarajuća struktura direktorijuma i paketa. Generisani kod je spreman za upotrebu u nekom programu, ili za direktno izvršavanje na nekoj mašini uz neke preduslove, poput postojanja kompajlera ili interpretera za ciljni jezik.



Slika 3. *Prikaz aplikacije*

Čuvanje i učitavanje šeme omogućava njenu ponovnu upotrebu. U planu za dalji razvoj predstavlja učitavanje šeme kao podšeme glavne šeme, čime će biti omogućeno dekomponovanje i mogućnost ponovne upotrebe.

Za svaku operaciju ili tip elementa koji je moguće dodati na šemu postoji i odgovarajuća dokumentacija, koja dodatno objašnjava dati element, primer njegove upotrebe i njegovu strukturu. U ovoj verziji dokumentacija predstavlja veb pretraživač, čime je omogućeno linkovanje na druge relevantne elemente i korišćenje javascript-a, kao npr. za bolji prikaz matematičkih formula.

5. ZAKLJUČAK

Prednost kreiranja šema uz pomoć vizuelizacije jeste bolje razumevanje i jednostavnost dodavanja, jer postoji već prethodno definisana paleta alata.

U pogledu vremena potrebnog za kreiranje koda ili potrebnog za kreiranje šeme, često je rezultat da je manje vremena neophodno za kreiranje šeme, pogotovo kod složenijih šema. FLCT se pokazao kao dobro rešenje za generisanje koda šema fazi logičkih kontrolera i samim tim je postao dobar izbor za dalja proširenja i unapređivanja.

Dodavanje DSL-a za neuronske mreže je pokrilo još jednu oblast koja se često koristi.

Generisanje koda predstavlja samo opcioni deo procesiranja ovog DSL-a. Šema se može upotrebiti i za skciranje arhitekture mreže, koja može biti deljena drugim korisnicima, koji na osnovu nje mogu izvršiti svoju implementaciju. Validacija šeme predstavlja korisnu opciju koja će otkloniti potrebu za validaciju šeme od strane korisnika, i time napraviti još jednu uštedu vremena u kreiranju šeme.

Takođe, moguće je vršiti upotrebu oba DSL-a u jednoj šemi. To može biti korisno da se npr. neuronska mreža koristi za izbor parametara funkcija pripadnosti u fazi logičkim kontrolerima. Drugi primer bi bio upotreba fazi logike za preprocesiranje i postprocesiranje podataka za obučavanje neuronskih mreža.

Dalja proširenja jezika bi mogla da omoguće i proces obučavanja neuronskih mreža, testiranje neuronskih mreža, kao i dodavanje komponente za prikaz rezultata, poput *TensorBoard*-a [6].

Pored toga, kreiranje i drugih reprezentacija DSL-a bi omogućila veći izbor kreiranja šema u skladu sa korisnikovom naklonjenošću za određene DSL-ove.

6. LITERATURA

[1] A. Mitrović, "Razvojno okruženje za projektovanje i generisanje koda fazi logičkih kontrolera," Bachelor thesis, University of Novi Sad, Novi Sad, 2017.

[2] "PyTorch," 2018. [Na mreži]. Dostupno na: https://pytorch.org/. [Pristupljeno: 11-Sep-2018].

[3] "Qt for Python - Qt Wiki," 2018. [Na mreži]. Dostupno na: https://wiki.qt.io/Qt_for_Python. [Pristupljeno: 11-Sep-2018].

[4] "Model/View Programming | An editable model," 2018. [Na mreži]. Dostupno na: http://doc.qt.io/qt-5/model-view-programming.html#an-editable-model. [Pristupljeno: 18-Sep-2018].

[5] "Welcome to Jinja2 — Jinja2 Documentation (2.10)," 2018. [Na mreži]. Dostupno na: http://jinja.pocoo.org/docs/2.10/. [Pristupljeno: 11-Sep-2018].

[6] "TensorBoard: Graph Visualization," *TensorFlow*, 2018. [Na mreži]. Dostupno na: https://www.tensorflow.org/guide/graph_viz. [Pristupljeno: 19-Sep-2018].

Kratka biografija:

Aleksandra Mitrović rođena je u Loznici 1994. god. Osnovne studije iz oblasti Elektrotehnika i računarstvo – softversko inženjerstvo i informacione tehnologije je završila 2017. godine kada je upisala i master studije iz iste oblasti.