ŠOLSKI CENTER CELJE
SŠ KER

Generator glasbe

(Raziskovalna naloga, Mladi za Celje 2018)

Mentor:
mag. Boštjan Resinovič, univ. dipl. inž. rač. inf.

Avtorji:
Tadej Tržan, R4b
Kevin Šarlah, R4b
Beno Volavšek, R4b

Celje, 4. 3. 2018
Šolsko leto 2017/18

Kazalo vsebine

Povzetek ... 3
Abstract ... 3
Ključne besede ... 3
Keywords ... 3
Kratice in okrajšave ... 4
1 UVOD ... 5
1.1 Opis / predstavitev problema ... 5
1.2 Hipoteze ... 5
1.3 Cilji ... 5
1.4 Raziskovalne metode ... 6
1.5 Pripomočki ... 6
1.5.1 Strojna oprema ... 6
1.5.2 Programska oprema ... 6
2 Teoretične osnove ... 7
2.1 Strojno učenje ... 7
2.1.1 Nevronske mreže ... 7
2.2 TensorFlow ... 9
2.3 Python ... 9
2.4 Anaconda ... 10
3 Potek dela ... 11
3.1 Kaj dela generator ... 11
3.1.1 Delovanje TensorFlow ... 12
3.1 Glavni program ... 20
4 Rezultati raziskovalnega dela ... 23
4.1 Anketa ... 23
4.1.1 Rezultati ankete ... 23
4.1.2 Zaključki iz ankete ... 24
4.2 Drugi zaključki in hipoteze ... 25
5 ZAKLJUČEK ... 26
6 VIRI IN LITERATURA ... 27
PRILOGE ... 28

Povzetek

Glavni cilj naše naloge je bil ustvariti delujoči generator glasbe, ki uporablja strojno učenje, da generira novo glasbo na osnovi primerov za učenje. Nastali generator temelji na orodju TensorFlow, posredno na jeziku Python, in MIDI datotečnem formatu glasbe.
V nalogi je zapisan rezultat raziskovalnega dela, opisane so teoretične osnove nevronskih mrež, orodja TensorFlow in drugih uporabljenih orodij. Opisan je potek našega dela in rezultat tega.

Abstract

The main goal of our task was to create a functioning music generator that uses machine learning to generate new music based on learning examples. The created generator is based on the TensorFlow toolkit, indirectly also the Python language, and the MIDI file format for music. The paper summarizes the results of the research work, describes the theoretical basics of Neural Networks, the TensorFlow toolkit and other tools used. We describe our workflow and its results.

Ključne besede

strojno učenje, nevronska mreža, TensorFlow, generator glasbe

Keywords

machine learning, neural network, TensorFlow, music generator

Kratice in okrajšave

- NN – Neural Network (nevronska mreža)
- MIDI – Musical Instrument Digital Interface (tehnični standard, ki opisuje komunikacijski protokol, digitalni vmesnik in električne priključke)
- TF – TensorFlow (odprtokodna knjižnica programske opreme za numerično računanje in splošno delo z NN in strojnim učenjem)
- API – ApplicationProgramming Interface (niz definicij podprogramov, protokolov in orodij za izdelavo aplikacijske programske opreme)
- op – Operation (Kratica za operacijo znotraj okolja TensorFlow)
- CPU – Central Processing Unit (centralna procesna enota - procesor)
- GPU – Graphics Processing Unit(grafična procesna enota – grafična kartica)

1 UVOD

V zadnjih letih se je priljubljenost sistemov, ki uporabljajo strojno učenje izjemno povečala. Tudi mi smo opazili ta trend in opazovali razvoj izjemnih programov in orodij, kot sta Googlov DeepDream - program za prepoznavanje in razvrstitev slik, program AlphaGo, ki igra igro Go in je v njej celo premagal enega najboljših svetovnih igralcev.
Ko smo si morali izbrati temo za našo raziskovalno in maturitetno nalogo, smo se vprašali, kaj so naši skupni interesi. Odkrili smo, da imamo zelo različne interese in le nekaj reči, ki nas povezujejo, a ena od teh, če ne morda kar najmočnejša, je naša ljubezen do glasbe.
Vedeli smo torej, da želimo narediti nekaj povezanega z glasbo in računalništvom. Izmenjali smo nekaj idej in kmalu ugotovili, da nas vse zanima prej omenjeno strojno učenje in da bi bilo mogoče raziskati sposobnosti tega v kontekstu ustvarjanja glasbe.

1.1 Opis / predstavitev problema

Na podlagi strojnega učenja se da generirati novo glasbo in to izboljšati z učenjem sistema. Presodili smo, da je najlažji način upravljanja z glasbo in generiranja glasbe MIDI datotečni format.
Naš cilj je torej bil ustvariti program, preučiti in uporabiti prej obstoječa orodja za izdelavo delujoče strukture strojnega učenja, s katero lahko preko MIDI datotek učimo strukturo in generiramo novo glasbo.

1.2 Hipoteze

Postavili smo si hipoteze glede izdelave in uporabe našega generatorja in kako je generirana glasba podobna človeško ustvarjeni, oziroma ali lahko ljudje razločijo, katera je generirana in katera ni.

- Za izdelavo generatorja bo potrebno večinoma izvenšolsko znanje.
- Večina anketiranih bo ustvarjeno glasbo prepoznalo kot računalniško generirano glasbo.
- Ob uporabi večjega števila ponovitev učenja na posamezni skladbi in večjega števila skritih nevronov nastane boljša ali pa vsaj bolj človeška glasba.

1.3 Cilji

Postavili smo si cilje, in sicer da se naučimo uporabljati TensorFlow, ustvarimo delujočo strukturo strojnega učenja in to strukturo uporabimo za generiranje nove glasbe.

1.4 Raziskovalne metode

Delali smo v timu, torej smo si delo razdelili in sodelovali med seboj s točno določenimi nalogami posameznika. Raziskali smo že obstoječe izdelke in na katerih orodjih in teoretičnih osnovah ti temeljijo. Za preverjanje naše druge hipoteze smo uporabili kratko anketo in anketirali sošolce.

1.5 Pripomočki

Ker naša naloga temelji na izdelku s področja računalništva, smo uporabili v njeni izdelavi strojno in programsko opremo.

1.5.1 Strojna oprema

Našo strojno opremo sta predstavljala dva računalnika. Ker naš izdelek deluje na CPU in ne GPU je vredno omeniti, da sta to bila delovna postaja z Intel i7 4790K in prenosnik z Intel i7 7700HQ procesorjem.

1.5.2 Programska oprema

Ker naš izdelek temelji na orodju TensorFlow, ki je odprtokodna knjižnica programske opreme za numerično računanje s pomočjo grafov podatkovnih tokov, smo morali uporabiti tolmačen programski jezik visokega nivoja za splošne namene Python in relevantne knjižnice jezika. Z jezikom Python smo delali preko distribucije Anaconda, ki omogoča obsežno obdelavo podatkov, napovedno analitiko in znanstveno računanje.

2 Teoretične osnove

V izdelavi našega izdelka smo uporabili veliko različnih računalniških področij in orodij. Najpomembnejše smo opisali v tem poglavju. Te so: strojno učenje, orodje TensorFlow, programski jezik Python in Python distribucija Anaconda.

2.1 Strojno učenje

Strojno učenje je področje računalništva, ki se ukvarja s sposobnostmi učenja računalniških sistemov. To takšnim sistemom omogoča postati boljši v reševanju nekaterih nalog, ne da bi bilo specifično sprogramirani zanje s točno določenimi algoritmi. Da je to učenje mogoče, so potrebni ustrezni podatki. Strojno učenje lahko na podlagi teh podatkov dela tudi napovedi. Vsako strojno učenje deluje preko neke strukture oz. modela, ki nastane na podlagi vhodnih podatkov. Strojno učenje se uporablja v številnih računalniških problemih, kjer je oblikovanje in programiranje algoritmov z ustrezno dobrim delovanjem težko ali neizvedljivo. Strojno učenje podpira tudi delovanje umetne inteligence in splošne umetne inteligence.

2.1.1 Nevronske mreže

Ena od bolj priljubljenih in raziskanih struktur za strojno učenje je umetna nevronska mreža, zato je tudi temelj našega sistema. Umetne nevronske mreže temeljijo na nevronskih mrežah v živih bitjih. Pogosto se ukvarja s problemi, kot so prepoznavanje slik, prepoznavanje glasu in medicinske diagnoze.

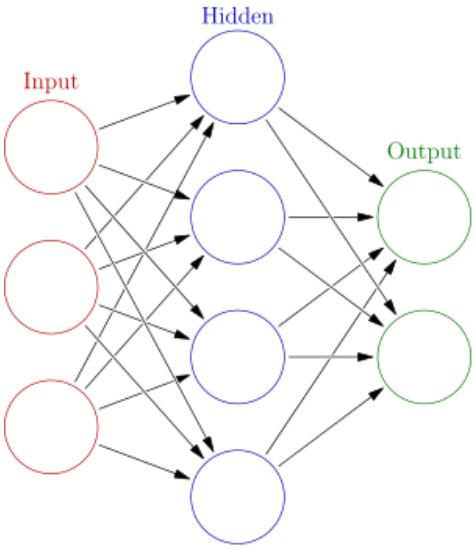
2.1.1.1 Nevronske mreže nasploh

Nevron ali živčna celica je električno razdražljiva celica, ki sprejema, obdeluje in prenaša informacije preko električnih in kemičnih signalov. Ti signali med nevroni potujejo preko povezav, ki se imenujejo sinapse. Nevroni se lahko povezujejo med seboj, da tvorijo nevronske mreže (NN). Nevroni so glavna sestavina osrednjega živčnega sistema in omogočajo učenje, napredek in razmišljanje.

2.1.1.2 Umetne nevronske mreže

»Umetne nevronske mreže so eno od glavnih orodij, ki se uporabljajo pri strojnem učenju. Kot nakazuje "nevronski" del njihovega imena, so to sistemi, ki jih navdihujejo možgani, namenjeni nadomestitvi človeškega načina učenja. Nevronske mreže sestavljajo vhodni in izhodni sloji, pa tudi (v večini primerov) skriti sloji, sestavljeni iz enot, ki transformirajo vhod v nekaj, kar lahko uporabi izhodna plast. So izvrstna orodja za iskanje vzorcev, ki so preveč zapleteni ali številni za človeškega programerja, da bi jih izvlekel in učil stroj prepoznati.« (Dormehl, 2018)

ANN temeljijo na zbirki povezanih enot ali vozlov/vozlišč (nodes), imenovanih umetni nevroni (artificial neurons). Vsaka povezava (sinapse) med umetnimi nevroni lahko prenaša signal od enega do drugega. Umetni nevron, ki sprejema signal, ga lahko obdeluje in nato posreduje umetnim nevronom, ki so z njim povezani.
V ANN je signal na povezavi med umetnimi nevroni (weight) realno število (med 1 in -1) in izhod vsakega umetnega nevrona se izračuna z nelinearno funkcijo (Sigmoid, ReLU) vsote njihovih vložkov pomnoženih z utežmi. Umetni nevroni in povezave imajo utež, ki se prilagaja kot učno sredstvo. Vrednost uteži poveča ali zmanjša moč signala pri povezavi in s tem pomembnost te. Umetni nevroni imajo lahko mejo, tako da je signal poslan le če signal agregata prečka to mejo. Značilno je, da so umetni nevroni organizirani v plasteh. Različne plasti lahko na svojih vhodnih podatkih izvedejo različne vrste transformacij. Signali potujejo od prve (vhodne) do zadnje (izhodne) plasti, po možnosti po večkratnem premikanju slojev.



Osnovni graf nevronske mreže 1

Najpomembnejše in vplivne operacije se izvajajo v vmesnih (skritih) plasteh.
Prvotni cilj ANN je bil rešiti probleme na enak način, kot bi ga lahko ljudje. Sčasoma se je pozornost osredotočila na ujemanje določenih nalog, kar je vodilo do odstopanj od biologije.

2.2 TensorFlow

TensorFlow je odprtokodna knjižnica programske opreme za numerično računanje s pomočjo grafov podatkovnih tokov. Vozlišča v grafu predstavljajo matematične operacije, medtem ko robovi grafov predstavljajo večdimenzionalne podatkovne matrike – tenzorje (n-dimenzionalne matrike), posredovane med njimi. Fleksibilna arhitektura omogoča, da na računalniku, strežniku ali mobilni napravi z enim samim API uvedete izvajanje računanja na enega ali več procesorjev ali grafičnih kartic. TensorFlow so razvili raziskovalci in inženirji, ki so delali v Google Brain Teamu v Googlovem raziskovalnem središču Machine Intelligence (Strojna Inteligenca) za namene strojnega učenja in raziskovanja globokih nevronskih mrež, vendar je sistem dovolj splošen, da se lahko uporablja na številnih drugih področjih.

Podjetja, ki uporabljajo TensorFlow:
NVidia, Airbnb, Uber, DeepMind, Dropbox, Ebay, Google, Snapchat, Intel, Coca Cola, Twitter, Lenovo, IBM in še veliko drugih.

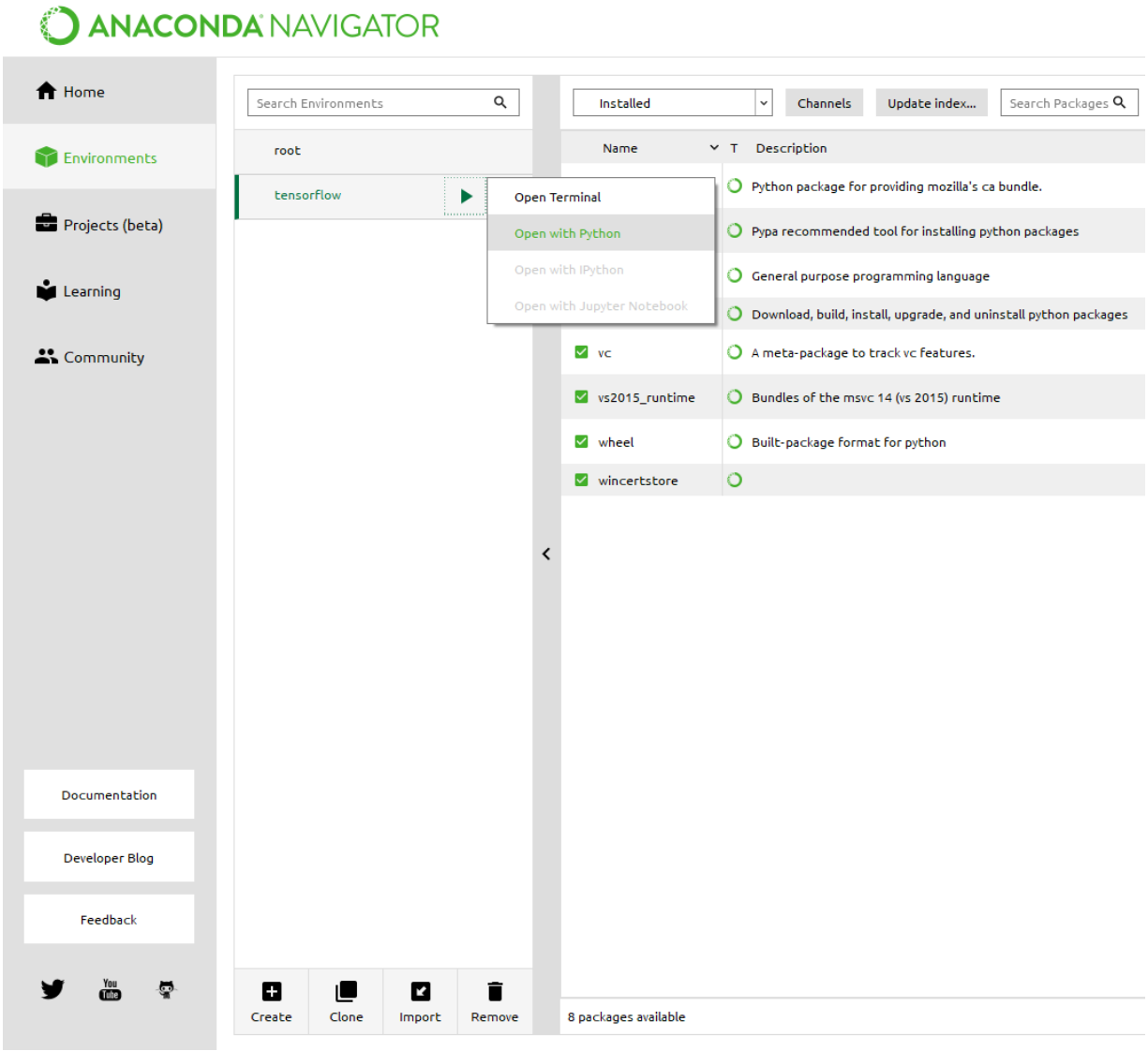
Logotip TensorFlow 1

2.3 Python

Čeprav TensorFlow ponuja funkcionalnost v mnogih programskih jezikih (C++, Java, Go, Haskell in Rust), je Python trenutno edini jezik, ki ga podpira TensorFlow API. Čeprav je naše znanje tega jezika pred to nalogo bilo omejeno, nismo videli druge poti, kot da se ga naučimo uporabljati.
Python je tolmačen programski jezik visokega nivoja za splošne namene. Poudari berljivost kode in sintakso, ki programerjem omogoča, da izrazijo koncepte v manj vrsticah kode. Nekaj programerjev v šali Python imenuje tudi kar »izvedljiva psevdokoda«. Zaradi naštetih lastnosti nas nov jezik ni ustavil na poti do naših ciljev.
V tem projektu je uporabljen Python 3.5.4, saj verzija 2.7 (najbolj priljubljena in podprta starejša verzija) ni kompatibilna s TensorFlow.

2.4 Anaconda

Anaconda je odprtokodna distribucija Python in R programskih jezikov za obsežno obdelavo podatkov, napovedno analitiko in znanstveno računanje. Paketne različice upravlja sistem upravljanja paketov Conda.



Za Anacondo smo se odločili, ker ima uporabniku prijazen vmesnik, vgrajeno shranjevanje v oblaku in navodila za uporabo za nameščanje in uporabo TensorFlow.

3 Potek dela

Problema smo se sprva lotili z iskanjem rešitev za probleme podobne našemu problemu in pri tem opazili relativno razširjeno orodje TensorFlow za delo s strojnim učenjem. Čeprav ta deluje na podlagi jezika Python nas to ni ustavilo na poti do cilja, saj smo vedeli, da ima že Python sam veliko (morda celo kar največ v primerjavi z drugimi programskimi jeziki) podpore za strukture strojnega učenja.

Ko smo se dokončno odločili za uporabo TF, smo že našli tudi poizkuse drugih za skrajno podobne operacije, kot smo jih želeli izvajati sami. Najbolj mamljiv primer teh že obstoječih rešitev je orodje Magenta, ki znotraj TF obdeluje (in torej tudi generira) glasbo na podlagi MIDI formata datotek glasbe. Čeprav smo mislili, da smo našli rešitev, ki smo jo potrebovali, smo z Magento prišli do težav pri implementaciji, katerih nismo uspeli razrešiti. Poizvedbo za rešitev naših težav smo sicer posredovali na nekaj programerskih forumov in na podporo Magente, a smo se po dvodnevnem čakanju na produktiven odgovor odločili, da bo naš čas bolje uporabljen s preučevanjem TF in generiranjem glasbe neposredno z njim, torej izven Magente.

Po tej odločitvi smo se še vedno strinjali da želimo da naš program deluje z MIDI datotekami, kot to počne tudi Magenta, saj manipulacija drugih zvočnih formatov, kot je npr. mp3, zahteva veliko večjo natančnost in podrobno pozornost za strojno učenje in je tako rekoč problem zase.

Uspelo nam je ustvariti algoritem za učenje strukture, ki temelji na osnovi najdenega programa za delo z MIDI.

Na internetu smo iskali primerne MIDI skladbe pop zvrsti v pravilnem MIDI formatu, ki ima sled le enega inštrumenta. Našli smo 126 skladb, katerih obsežnost sklepamo da je zadosti za učenje našega NN.

Ko smo imeli izdelane vse funkcionalne dele, smo začeli testirati in učiti izdelek.

3.1 Kaj dela generator

Naš izdelek uči nevronsko mrežo ustvarjeno v TensorFlow in na podlagi učenja iz datotek MIDI formata generira nove dele glasb, ki jih lahko združi v celotno skladbo (prav tako v obliki MIDI).

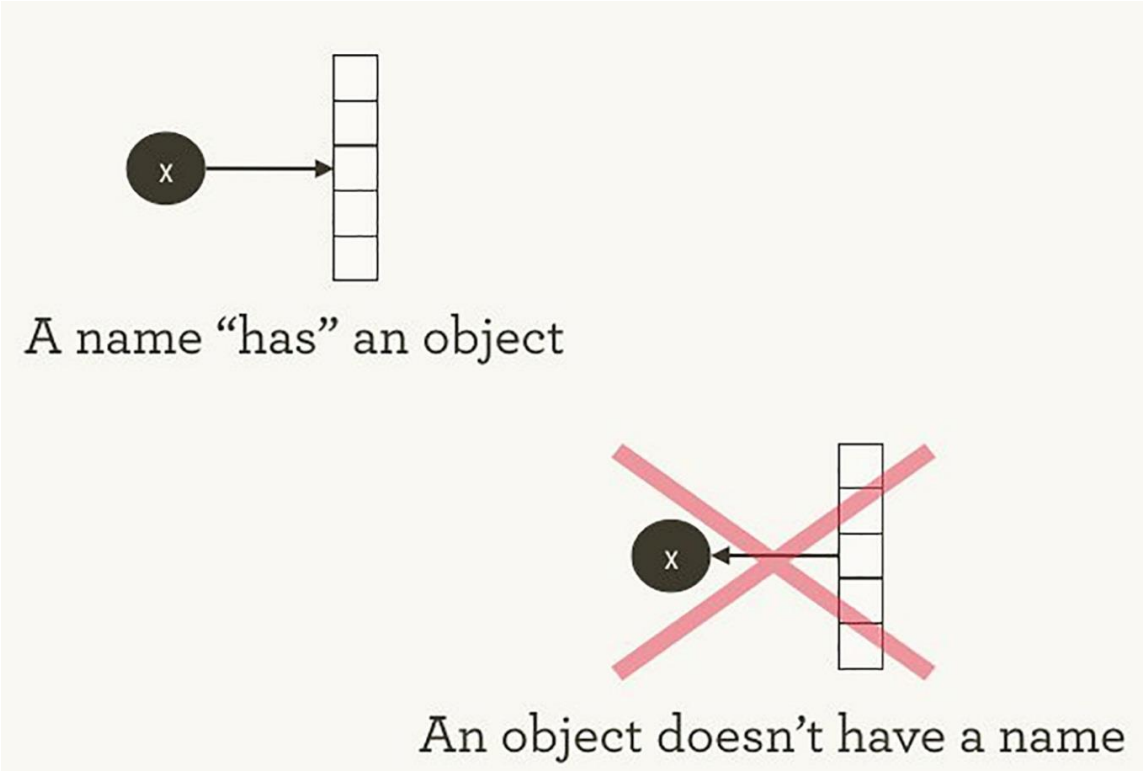
V tem poglavju sledi razlaga kako generator to dela.

3.1.1 Delovanje TensorFlow

Kako deluje TensorFlow? Razložili bomo graf toka podatkov, ki definira izračune, izvajane na podatkih, kako trenirati modele z uporabo TensorFlow gradientnih spustov.

3.1.1.1 Imena in izvajanje v Python in TensorFlow

Način, kako TensorFlow upravlja z računanjem, ni povsem drugačen od načina, ki ga uporablja Python. Pri obeh torej objekt nima imena, ampak ima ime objekt. Imena v Pythonu niso v resnici to kar predstavljajo, temveč le kažejo na zastopane objekte. interno, Python upravlja z vsemi vašimi predmeti in spremlja imena vaših spremenljivk in na katere objekte se nanašajo. Graf TensorFlow predstavlja še eno plast te vrste upravljanja; Python imena se nanašajo na objekte, ki se povezujejo z bolj granularnimi in upravljanimi operacijami grafov TensorFlow.



Python izraze računa sproti, kot so mu ti dani. To dela tudi, če je nek izraz uporabljen le enkrat. Alternativa temu bi bila le zapomniti si izraz, in če je uporabljen kdaj kasneje, bi se to zgodilo takrat in ne prej. To je bližje obnašanju TensorFlow, kjer je opredelitev odnosov popolnoma ločena od računanja rezultatov. TensorFlow ločuje definicijo izračunov od njihovega izvajanja še bolj, saj se zgodijo na različnih mestih: graf definira operacije, vendar se operacije izvedejo le znotraj seje. Grafi in seje so ustvarjeni neodvisno.

»Graf je kot načrt in seja je kot gradbišče.« (Schumacher, 2018)

V Pythonu lahko referenca kaže sama nase. To strukturo si lahko predstavljamo kot graf z enim vozliščem, ki kaže samo nase. Takšni gnezdeni seznami so eden od načinov za predstavitev grafov, kot je graf izračunov TensorFlow. Resnični TensorFlow grafi seveda niso sestavljeni le iz enega gnezdenega seznama, temveč so mnogo bolj kompleksni.

3.1.1.2 Najenostavnejši graf v TensorFlow

Za začetek vključimo TensorFlow.

```
>>> import tensorflow as tf
```

S tem je TensorFlow že začel za nas upravljati veliko stvari v ozadju. Na primer že obstaja implicitni privzeti graf. Privzeti graf je v _default_graph_stack, vendar nimmo neposrednega dostopa do tega. Uporabljamo funkcijo za pridobitev grafa tf.get_default_graph ().

```
>>> graph = tf.get_default_graph()
```

Vozlišča grafov TensorFlow se imenujejo "operacije" ali kar »op« v kodi. Z grafičnim elementom graph.get_operations() lahko vidimo, katere operacije so v grafu.

```
>>> graph.get_operations()
## []
```

Zaenkrat v grafu še ni operacij ali česarkoli drugega. Vse, kar želimo da TensorFlow računa, moramo postaviti v ta graf. Začnimo z enostavno, vhodno konstanto z vrednostjo ena.

```
>>> input_value = tf.constant(1.0)
```

Ta konstanta je zdaj v grafu v obliki vozlišča, operacije. Ime spremenljivke input_value se posredno nanaša na to operacijo, a je operacija sama tudi v grafu.

TensorFlow interno uporablja protokolne odbojnike (Protocol Buffers). (Protokolni odbojniki so podobni JSON v Googlu). Tiskanje node_def za operacijo konstante spodaj prikazuje, kaj je predstavlja protokolnegi odbojnik TensorFlow za prvo številko.

```
>>> operations = graph.get_operations()
>>> operations
## [<tensorflow.python.framework.ops.Operation at
0x1185005d0>]
>>> operations[0].node_def
## name: "Const"
## op: "Const"
## attr {
##   key: "dtype"
##   value {
##     type: DT_FLOAT
##   }
## }
## attr {
##   key: "value"
##   value {
##     tensor {
##       dtype: DT_FLOAT
##       tensor_shape {
##       }
##       float_val: 1.0
##     }
##   }
## }
```

TensorFlow je zelo močno orodje, vendar lahko deluje samo s tem, kar mu je bilo izrecno dano. To velja tudi za eno samo konstanto.
Če pregledamo input_value, vidimo, da je konstanten 32-bitni float tensor brez dimenzije: samo ena številka.

```
>>> input_value
## <tf.Tensor 'Const:0' shape=() dtype=float32>
```

Same vrednosti input_value ne izvemo. Če želimo ovrednotiti input_value in dobiti številčno vrednost, moramo ustvariti sejo (session/sess), kjer je mogoče ovrednotiti operacije grafa in nato izrecno zahtevati ovrednotenje ali zagon (run) input_value.
Seja privzeto vzpostavi privzeti graf, če ni rečeno drugače.

```
>>> sess = tf.Session()
>>> sess.run(input_value)
## 1.0
```

Nenavadno je zagnati konstanto, ampak ni tako drugače od ocenjevanja izraza v navadnem Pythonu. Razlika je le, da TensorFlow upravlja svoj lastni prostor stvari – računalniški graf – in ima svojo lastno metodo vrednotenja.

3.1.1.3 The simplest TensorFlow neuron

Ko je ustvarjena seja s preprostim grafom, lahko ustvarimo nevrone. Za obrazložitev delovanja bo to nevron s samo enim parametrom – utežjo. Celo preprosti nevroni imajo pogosto tudi pristranskost in razne funkcije, vendar jih bomo v tem primeru izpustili, da bo razlaga bolj jasna in strnjena.

Utež nevrona ne bo konstantna. Pričakujemo, da se bo spremenila, da bi se učili na podlagi "resničnega" vhoda in izhoda, ki ga uporabljamo za učenje. Teža bo TensorFlow spremenljivka. Tej spremenljivki bomo dali začetno vrednost 0,8.

```
>>> weight = tf.Variable(0.8)
```

Lahko bi pričakovali, da bi dodajanje spremenljivke grafu dodalo eno operacijo, a bi se motili. Ta ena vrstica bo dodala štiri operacije. Lahko pregledamo vsa imena operacij:

```
>>> for op in graph.get_operations(): print(op.name)
## Const
## Variable/initial_value
## Variable
## Variable/Assign
## Variable/read
```

Dodamo še računski postopek pridobitve izhodne vrednosti output_value.

```
>>> output_value = weight * input_value
```

Sedaj je v grafu šest operacij, zadnje je to množenje.

```
>>> op = graph.get_operations()[-1]
>>> op.name
## 'mul'
>>> for op_input in op.inputs: print(op_input)
## Tensor("Variable/read:0", shape=(), dtype=float32)
## Tensor("Const:0", shape=(), dtype=float32)
```

Na zgornji sliki lahko vidimo, kako operacija množenja sledi, od kod prihajajo njeni vhodi: prihajajo iz drugih operacij v grafu. Če želimo razumeti celoten graf, sledenje referenc na ta način hitro postane dolgočasno in preveč kompleksno za človeka. Vizualizacija grafikonov TensorBoard je zasnovana tako, da je v pomoč.

Kako ugotovimo, kaj je produkt? Moramo "zagnati" operacijo output_value. Toda ta operacija je odvisna od spremenljivke – uteži. TensorFlowu smo povedali, da mora biti začetna vrednost teže 0,8, vendar vrednost še ni nastavljena v trenutni seji. Funkcija tf.initialize_all_variables () ustvari operacijo, ki bo inicializirala vse naše spremenljivke (v tem primeru le eno), nato pa lahko zaženemo to operacijo.

```
>>> init = tf.initialize_all_variables()
>>> sess.run(init)
```

Rezultat tf.initialize_all_variables () bo vključeval inicializatorje vseh spremenljivk, ki so trenutno v grafu. Če bi dodali več spremenljivk, bi morali ponovno uporabiti tf.initialize_all_variables (). Statični, nespremenjen init ne bi vključeval novih spremenljivk.

Zdaj lahko zaženemo operacijo output_value.

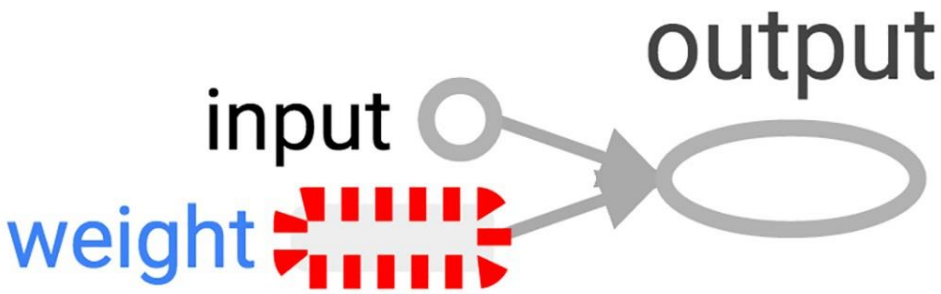
```
>>> sess.run(output_value)
## 0.80000001
```

0,8 * 1,0 z 32-bitnimi floati. Ti imajo težave z 0,8. 0,80000001 je najbližji približek.

Uporaba tf.mul tukaj je enaka naši prejšnji uporabi * za množenje, vendar nam omogoča, da nastavimo ime operacije.

```
>>> x = tf.constant(1.0, name='input')
>>> w = tf.Variable(0.8, name='weight')
>>> y = tf.mul(w, x, name='output')
```

Naš graf bi sedaj zgledal nekako takole:



3.1.1.4 Učenje nevrona

Kako se bo učil naš nevron? Nastavili smo vhodno vrednost 1,0. Recimo, da je pravilna izhodna vrednost nič. To pomeni, da imamo zelo preprost "vadbeni niz" samo enega primera z eno funkcijo, ki ima vrednost ena, in eno oznako, ki je nič. Hočemo, da se nevron nauči funkcije, ki spremeni ena v nič.

Trenutno sistem sprejme vhodno vrednost in vrne 0,8, kar ni pravilno. Potrebujemo način, kako izmeriti, kako daleč od pravilnega odgovora je sistem. To mero napačnosti bomo označili kot izgubo (loss) in našemu sistemu dali cilj zmanjšati izgubo. Če je izguba lahko negativna, potem bi jo lahko zmanjkalo, tako da je izguba kvadratna razlika med trenutnim izhodom in želenim izhodom.

```
>>> y_ = tf.constant(0.0)
>>> loss = (y - y_)**2
```

Doslej nič v grafu ne izvaja nobenega učenja. Za to potrebujemo optimizator. Uporabili bomo optimizator spuščanja gradienta, tako da bomo lahko posodobili utež na podlagi vrednosti izpeljane iz izgube. Optimizator ima stopnjo učenja, da zmeri velikost posodobitev, ki jo bomo določili na 0.025.

```
>>> optim =
tf.train.GradientDescentOptimizer(learning_rate=0.025)
```

Optimizator je neverjetno pameten. Samodejno lahko izračuna in uporabi ustrezne gradiente skozi celo omrežje, pri čemer se izvaja vzvratni korak za učenje.

Poglejmo, kako je videti gradient za naš preprost primer.

```
>>> grads_and_vars = optim.compute_gradients(loss)
>>> sess.run(tf.initialize_all_variables())
>>> sess.run(grads_and_vars[1][0])
## 1.6
```

Zakaj je vrednost gradienta 1.6? Naša izguba je napaka na kvadrat, in derivat tega je dvakratna napaka. Trenutno sistem kaže 0.8 namesto 0, zato je napaka 0.8, dvakrat 0.8 pa je 1.6; torej deluje!

Za bolj zapletene TensorFlow izračuna in nato samodejno uporabi te gradiente.

Uporabimo gradient, da dokončamo vzvratno učenje.

```
>>> train_step =
tf.train.GradientDescentOptimizer(0.025).minimize(loss)
>>> for i in range(100):
>>>     sess.run(train_step)
>>>
>>> sess.run(y)
## 0.0044996012
```

Ob uporabi mnogih učnih korakov, utež in izhodna vrednost sta sedaj zelo blizu nič. Nevron se je naučil!

3.1 Glavni program

Namen programa je na osnovi glasbe za učenje generirati datoteke nove glasbe.
Program pridobi vso glasbo za učenje iz danega direktorija, na podlagi te uči strukturo strojnega učenja preko TensorFlow metod, za katere prej primerno določi parametre kot so število nevronov v plasti, število ponovitev učenja v enem setu in hitrost učenja. Določi se še TensorFlow spremenljivke: spremenljivke podatkov, matriko uteži, vektor skrite in vektor vidne plasti. Po korakih učenja preko gibbs verige se posodobijo vrednosti TensorFlow spremenljivk. Ustvari se graf strukture in se na osnovi tega generirajo nove glasbe v MIDI formatu, ki so poimenovane glede na parametre, uporabljene ob generiranju te datoteke.

```
1.  ## -*- coding: utf-8 -*-
2.  #"""
3.  #Uporabljene knjižnice:
4.
5.  import numpy as np
6.  import pandas as pd
7.  import glob
8.  import tensorflow as tf
9.
10. from tqdm import tqdm
11. from tensorflow.python.ops import control_flow_ops
12.
13. import midi_manipulation
14.
15.
16. #Pridobi vso glasbo v nekem direktoriju
17.
18. def get_songs(path):
19.     files = glob.glob('{}/*.mid*'.format(path))
20.     songs = []
21.     for f in tqdm(files):
22.         try:
23.             song = np.array(midi_manipulation.midiToNoteStateMatrix(f))
24.             if np.array(song).shape[0] > 50:
25.                 songs.append(song)
26.         except Exception as e:
27.             raise e
28.     return songs
29.
30. #### Pomožne funkcije
31.
32. #Funkija nam omogoča lahkotno vzorčenje iz vektorja verjetnosti (probability vector)
33. def sample(probs):
34.     #Vhod je vektor verjetnosti, izhod pa naključni vektor ničel in enic vzorčenih i
    z vhodnega vektorja
35.     return tf.floor(probs + tf.random_uniform(tf.shape(probs), 0, 1))
36.
37. #Funkcija izvrši gibbs verigo
38. def gibbs_sample(k):
39.     Izvrši k - korakov (step) gibbs verig za vzorčenje verjetnostne distribucije
    RBM, ki ga definirajo W, bh, bv
40.     def gibbs_step(count, k, xk):
```

```
        #Izvrši gibbs korak. Vidne vrednosti so inicializirane na xk
        hk = sample(tf.sigmoid(tf.matmul(xk, W) + bh)) #Razporejanje vidnih vrednosti za vzorčenje skritih vrednosti
        xk = sample(tf.sigmoid(tf.matmul(hk, tf.transpose(W)) + bv)) #Razporejanje skritih vrednosti za vzorčenje vidnih vrednosti
        return count+1, k, xk

    #Izvrši gibbs korak za k iteracij
    ct = tf.constant(0) #števec
    [_, _, x_sample] = control_flow_ops.while_loop(lambda count, num_iter, *args: count < num_iter,
                                            gibbs_step, [ct, tf.constant(k), x])
    #To ni nujno potrebno pri tej izvedbi, če pa bi želeli kodo prilagoditi uporabi enega od TensorFlow optimizatorjev, je to potrebno da ustavimo tensorflow od propagacijskih gradientov nazaj skozi korake gibbs
x_sample = tf.stop_gradient (x_sample)
    return x_sample
# Velikost skrite plasti
n_hidden = 100

# Število ponovitev/dob ucenja v enem setu
num_epochs = 300

songs = get_songs('Pop_Muske') #datoteke glasbe so že pretvorjene iz MIDI v msgpack
print ("{} glasbe najdene!".format(len(songs)))

### Konstantni parametri naših midi filov

lowest_note = midi_manipulation.lowerBound
highest_note = midi_manipulation.upperBound
note_range = highest_note - lowest_note

# Število primerov (sample) glasb, ki bodo generirani
sample_count = 20

# Število postopkov, ki bodo ustvarjeni
num_timesteps = 15

# Število primerkov, ki jih pošljemo naenkrat
batch_size = 100

# Hitrost učenja
learning_rate = tf.constant(0.001, tf.float32)

# Velikost vidne plasti
n_visible = 2 * note_range * num_timesteps

### TensorFlow spremenljivke

# Spremeniljivka v kateri imamo podatke
x = tf.placeholder(tf.float32, [None, n_visible], name="x")

# Matrika v kateri hranimo naše uteži (Weight)
W = tf.Variable(tf.random_normal([n_visible, n_hidden], 0.01), name="W")

# Vektor skritega lajerja (bias_hidden)
bh = tf.Variable(tf.zeros([1, n_hidden], tf.float32, name="bh"))

# Vektor vidnega lajerja (bias_visible)
bv = tf.Variable(tf.zeros([1, n_visible], tf.float32, name="bv"))

x_sample = gibbs_sample(1)

# Primer skritih vozlišč (node), začenjši z vidnim stanjem x
h = sample(tf.sigmoid(tf.matmul(x, W) + bh))
```

```
# Primer skritih vozlišč (node), začenjši z vidnim stanjem x_sample
h_sample = sample(tf.sigmoid(tf.matmul(x_sample, W) + bh))

# Posodobimo naše spremenljivke W, bh in bv glede na razlike samplov, ki smo jih upo
rabili in originalne vrednosti

size_bt = tf.cast(tf.shape(x)[0], tf.float32)
W_adder = tf.multiply(learning_rate/size_bt, tf.subtract(tf.matmul(tf.transpose(x),
h), tf.matmul(tf.transpose(x_sample), h_sample)))
bv_adder = tf.multiply(learning_rate/size_bt, tf.reduce_sum(tf.subtract(x, x_sample)
, 0, True))
bh_adder = tf.multiply(learning_rate/size_bt, tf.reduce_sum(tf.subtract(h, h_sample)
, 0, True))

updt = [W.assign_add(W_adder), bv.assign_add(bv_adder), bh.assign_add(bh_adder)]

# Nariše graf
with tf.Session() as sess:

    init = tf.global_variables_initializer()
    sess.run(init)

    for epoch in tqdm(range(num_epochs)):
        for song in songs:
            song = np.array(song)
            song = song[:int(np.floor(song.shape[0]/num_timesteps)*num_timesteps)]
            song = np.reshape(song, [song.shape[0]//num_timesteps, song.shape[1]*num
_timesteps])

            for i in range(1, len(song), batch_size):
                tr_x = song[i:i+batch_size]
                sess.run(updt, feed_dict={x: tr_x})

    sa*mple = gibbs_sample(1).eval(session=sess, feed_dict={x: np.zeros((sample_coun
t, n_visible))})
    mySongs = np.zeros((0,156))
    for i in range(sample.shape[0]):
        if not any(sample[i,:]):
            continue

        S = np.reshape(sample[i,:], (num_timesteps, 2*note_range))
        mySongs = np.concatenate((mySongs,S))
        #midi_manipulation.noteStateMatrixToMidi(S, "generated_chord_{}".format(i))
    #Oblika imenovanja datoteke ob generiranju, glede na parametre generiranja
    midi_manipulation.noteStateMatrixToMidi(mySongs, "Steps{1}NV{2}NH{3}Epoch{4}".fo
rmat(sample_count, num_timesteps, n_visible, n_hidden, num_epochs, batch_size))
```

4 Rezultati raziskovalnega dela

Generirali smo več skladb na podlagi različnih parametrov učenja.
Potrdili smo, da ob učenju nastane boljša glasba in da se glasba boljša s številom nevronov in številom korakov/dob učenja, a smo opazili, da se v nekaterih primerih učenja kvaliteta tudi poslabša, ker se zaradi začetnih naključnih parametrov lahko učenje tudi izneveri.

4.1 Anketa

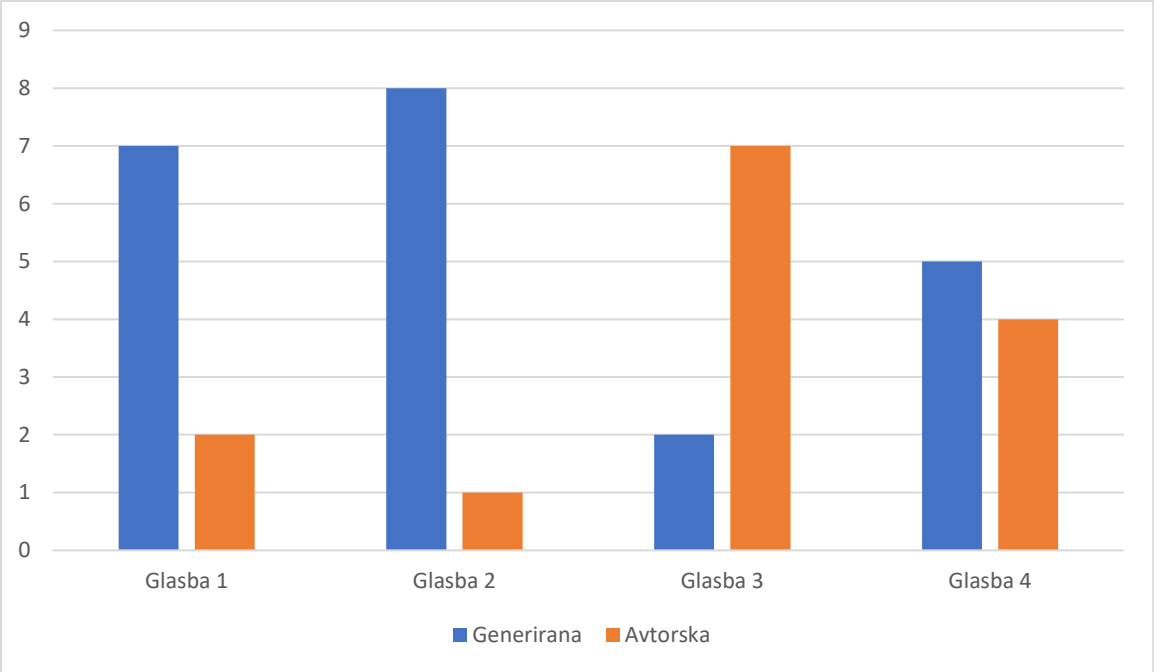
Namen ankete je bil preveriti resničnost naše hipoteze, kako je generirana glasba podobna človeško ustvarjeni, oziroma ali lahko ljudje razločijo, katera je generirana.

4.1.1 Rezultati ankete

Anketirali smo 9 sošolcev. Predvajali smo jim štiri glasbe, od katerih je bila ena MIDI verzija versa glasbe, ki jo je ustvaril človek klasično (ne generirano) Taylor Swift – Blank Space, drugi trije pa so računalniško generirani. Glasbi 1 in 2 smo generirali z našim programom z različnimi parametri učenja. Glasba 1 je Steps15NV2340NH50Epoch150.mid, torej generirana s 15 koraki, 2340 vidnimi nevroni, 50 skritimi nevroni in 150 učnimi obdobij. Glasba 2 je Steps15NV2340NH50Epoch1000.mid torej generirana z 15 koraki, 2340 vidnimi nevroni, 50 skritimi nevroni in 1000 učnimi obdobji.
Glasba 4 je generirana z Googlovo vnaprej naučeno nevronsko mrežo.
Za vsako glasbo smo anketirane vprašali, ali menijo, da je računalniško generirana ali ne.

1. Ali je glasba računalniško generirana?

Glasba/Anketirani	1	2	3	4	5	6	7	8	9
1 (generirana)	Ne	Da	Da	Da	Da	Da	Ne	Da	Da
2 (generirana)	Ne	Da	Da	Da	Da	Da	Da	Da	Da
3 (avtorska)	Ne	Da	Ne	Ne	Ne	Ne	Ne	Da	Ne
4 (generirana)	Da	Da	Ne	Ne	Ne	Da	Da	Da	Ne



4.1.2 Zaključki iz ankete

Kljub manjšemu vzorcu menimo, da je pridobljena statistika reprezentativna. Anketirani so bili naši sošolci, ki so jim blizu računalniške teme in so zato kvečjemu bolj natančni pri identifikaciji generirane glasbe.

Le ena oseba je pravilno določila izvor vseh štirih predvajanih glasb. Iz tega sklepamo, da je razlika med generirano glasbo in glasbo, ki jo ustvarjajo ljudje v osnovni obliki posnetka, komaj opazna.
Kdor je za prvi dve glasbi menil, da sta ti avtorski in je dodal tudi, da se mu je glasba zdela napisana na višjem nivoju (jazzovski vpliv).
Hipotezo, da bo večina anketiranih glasbo prepoznala kot računalniško generirano, je tako potrjena, a ne v celoti, saj so ob zadnji glasbi, generirani na vnaprej naučeni strukturi, rezultati različni in imajo ob tem bolj uspešnem učenju in generiranju glasbe ljudje velik problem razločevanja izvora.

4.2 Drugi zaključki in hipoteze

Glede naše druge hipoteze o uporabi skoraj izključno izvenšolskega znanja, menimo, da smo potrdili tudi to. Uporabljali smo Python, ki ni v rednem šolskem programu, orodje TensorFlow in z njim povezano strojno učenje, ki prav tako ni del našega šolskega programa in znanje za upravljanje z MIDI zvočnim formatom, kar ni del učnega programa, vsaj izven posameznih projektov, kot je naš.

Kljub vsem tem dejstvom pa moramo priznati, da nas je naša izobrazba vseeno podprla, pa čeprav v manj konkretnih načinih. Od rednega šolskega programa smo se naučili načrtovati in razvijati algoritme, kar nam je pri projektu in izdelavi programa prišlo prav, a morda še bolj v razumevanju algoritmov v TF nasploh. Pri mnogih praktičnih urah in projektnem delu v skupinah smo se naučili timskega dela in učinkovitega iskanja podatkov na internetu.

Da povzamem zadnji odstavek: hipotezo smo potrdili, a smo vseeno hvaležni šoli za izjemno podlago za nadaljnje učenje in pridobivanje znanja.

5 ZAKLJUČEK

Naloga nam je pokazala, koliko napredka je bilo v zadnjih letih na področju strojnega učenja in kako dostopno je to področje postalo.
Pri izdelavi generatorja smo sicer imeli nekaj težav, a so se sčasoma in z veliko truda odpravile, ali pa smo jih vsaj obšli in nastal je izdelek, na katerega smo upali.
Menimo, da je naša podrobna raziskava že obstoječih izdelkov podobnega namena zelo pomagala pri izdelavi našega generatorja, saj smo vedeli, na kaj moramo biti pozorni in smo imeli smernice širše javnosti in s tem potrdila o napredku in pravi smeri tega.
Dosegli smo vse svoje zadane cilje in potrdili hipotezo, da bo za izdelavo generatorja potrebno večinoma izvenšolsko znanje. Hipotezo, da bo večina anketiranih ustvarjeno glasbo prepoznalo kot računalniško generirano glasbo smo potrdili le delno, saj so pri zadnji glasbi v anketi, ki je bila generirana z najboljše učenim sistemom, prevarala kar nekaj nekateri menili, da je klasično ustvarjena. Potrdili smo tudi, da ob uporabi večjega števila ponovitev učenja na posamezni skladbi in večjega števila skritih nevronov nastane boljša glasba. Kombinacija obeh teh lastnosti je ustvarila naše najboljše rezultate.

Pridobljeno znanje o strojnem učenju in nevronskih mrežah je splošno uporabno na sorodnih področjih. Uporaba jezika Python je prav tako splošno uporabna, saj lahko z njim delamo tako rekoč kakršenkoli visokonivojski algoritem. TensorFlow je uporaben specifično na področju strojnega učenja, to področje pa je zelo široko in v porastu.

Na tem področju obstaja še mnogo možnosti za raziskavo in spodbujamo vse, ki jih vsaj bežno zanima strojno učenje, da se podajo na novo, dostopno in uporabno področje.
Neposredno dodatek k naši nalogi bi bila vizualizacija TensorFlow grafa, saj bi to omogočilo boljšo in lažjo predstavitev izdelka in postopka delovanja. Ta dodatek smo sicer želeli vključiti v nalogo že v osnovi, a nam to ni uspelo zaradi časovne omejitve.
S področja naše naloge bi lahko raziskovanje nadaljevali z že obstoječim orodjem znotraj TensorFlow, Magenta (posebej namenjeno strojnemu učenju povezanemu z glasbo), ki nam ga prav tako ni uspelo uporabiti. Uporabno, čeprav najverjetneje bolj kompleksno nadaljevanje raziskovanja, je nedvomno obdelava formatov, ki niso MIDI, temveč delujejo z opisom valov zvoka, kot je npr. .mp3 format.

6 VIRI IN LITERATURA

Anaconda. (12. 2 2018). *Anaconda*. Pridobljeno iz Anaconda.org: https://anaconda.org/

DeepMind. (10. 2 2018). *DeepMind collaborations with Google*. Pridobljeno iz deepmind.com: https://deepmind.com/applied/deepmind-for-google/

Dormehl, L. (1. 3 2018). *What is an artificial neural network? Here's everything you need to know*. Pridobljeno iz digitaltrensds.com: https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/

Michael, G. (11. 2 2018). *Install Python on windows - Anaconda*. Pridobljeno iz medium.com: https://medium.com/@GalarnykMichael/install-python-on-windows-anaconda-c63c7c3d1444

Schumacher, A. (1. 3 2018). *Hello, TensorFlow!* Pridobljeno iz oreilly.com: https://www.oreilly.com/learning/hello-tensorflow

TensorFlow. (13. 2 2018). *TensorFlow*. Pridobljeno iz GitHub.com: https://github.com/tensorflow

TensorFlow. (15. 2 2018). *TensorFlow Architecture*. Pridobljeno iz tensorflow.org: https://www.tensorflow.org/extend/architecture

TensorFlow. (16. 2 2018). *TensorFlow: API: Reading data*. Pridobljeno iz TensorFlow: https://www.tensorflow.org/api_guides/python/reading_data

TensorFlow. (13. 2 2018). *TensorFlow: Get started*. Pridobljeno iz tensorflow.org: https://www.tensorflow.org/get_started/

TensorFlow. (15. 2 2018). *TensorFlow: Placeholders*. Pridobljeno iz tensorflow.org: https://www.tensorflow.org/api_docs/python/tf/placeholder

TensorFlow. (15. 2 2018). *TensorFlow: Programmers guide: Tensors*. Pridobljeno iz tensorflow.org: https://www.tensorflow.org/programmers_guide/tensors

wikipedia. (11. 2 2018). *Artificial neural networks*. Pridobljeno iz wikipedia.org: https://en.wikipedia.org/wiki/Artificial_neural_network

wikipedia. (11. 2 2018). *Biological neural network*. Pridobljeno iz wikipedia: https://en.wikipedia.org/wiki/Biological_neural_network

wikipedia. (11. 2 2018). *Machine learning*. Pridobljeno iz wikipedia.org: https://en.wikipedia.org/wiki/Machine_learning

wikipedia. (13. 2 2018). *Nevronska mreža*. Pridobljeno iz Wikipedia.org: https://sl.wikipedia.org/wiki/Nevronska_mreža

PRILOGE

Glavni program:

temp.py

Program za delo z MIDI datotekami:

midi_manipulation.py

Program za združevanje MIDI datotek:

merge.py

Zbirka glasb za učenje:

Pop_Muske

IZJAVA*

Mentor (-ica) ,BOŠTJAN RESINOVIČ, v skladu z 2. in 17. členom Pravilnika raziskovalne dejavnosti »Mladi za Celje« Mestne občine Celje, zagotavljam, da je v raziskovalni nalogi naslovom

GENERATOR GLASBE,

katere avtorji (-ice) so KEVIN JARLAH, BENO VOLAUŠEK, TADEJ TRŽAN:

- besedilo v tiskani in elektronski obliki istovetno,
- pri raziskovanju uporabljeno gradivo navedeno v seznamu uporabljene literature,
- da je za objavo fotografij v nalogi pridobljeno avtorjevo (-ičino) dovoljenje in je hranjeno v šolskem arhivu,
- da sme Osrednja knjižnica Celje objaviti raziskovalno nalogo v polnem besedilu na knjižničnih portalih z navedbo, da je raziskovalna naloga nastala v okviru projekta Mladi za Celje,
- da je raziskovalno nalogo dovoljeno uporabiti za izobraževalne in raziskovalne namene s povzemanjem misli, idej, konceptov oziroma besedil iz naloge ob upoštevanju avtorstva in korektnem citiranju,
- da smo seznanjeni z razpisni pogoji projekta Mladi za Celje

Celje, 12. 3. 2018

žig šole

ŠOLSKI CENTER CELJE
ŠOLA ZA KEMIJO, ELEKTROTEHNIKO IN RAČUNALNIŠTVO
SREDNJA ŠOLA 1 CELJE, Pot na Lavo 22

Podpis mentorja(-ice)

Podpis odgovorne osebe

*

POJASNILO

V skladu z 2. in 17. členom Pravilnika raziskovalne dejavnosti »Mladi za Celje« Mestne občine Celje je potrebno podpisano izjavo mentorja(-ice) in odgovorne osebe šole vključiti v izvod za knjižnico, dovoljenje za objavo avtorja(-ice) fotografskega gradiva, katerega ni avtor(-ica) raziskovalne naloge, pa hrani šola v svojem arhivu.