Netta Immonen

Syväoppimiskehyksen hyödyntäminen chattibotin toteutuksessa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Ohjelmistotuotanto

Insinöörityö

7.11.2022

Tiivistelmä

Tekijä:	Netta Immonen
Otsikko:	Syväoppimiskehyksen hyödyntäminen chattibotin toteutuksessa
Sivumäärä:	36 sivua
Aika:	7.11.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Vesa Ollikainen

Chattibotteja käytettään yhä useammassa paikassa kommunikoimaan asiakkaiden kanssa. Chattibottien määrä on viimevuosien aikana kasvanut räjähdysmäisesti. Tekoälyn hyödyntäminen on nykyään mahdollista muillekin kuin tekoälyn tutkijoille ja suurille yrityksille, joilla on varaa rahoittaa tutkimusta. Syväoppimiskehykset ja tietokoneiden laskentatehon kasvu on demokratisoinut tekoälyn käytön ja mahdollistanut sen, että pienetkin toimijat voivat hyödyntää muun muassa chattibotteja toiminnassaan. Chattibotit mahdollistavat sen, että yrityksen asiakkaat voivat nopeasti saada oikeaa ja ajankohtaista tietoa yksinkertaisiin kysymyksiin botilta, jolloin sekä asiakkaan että asiakaspalvelijan aikaa säästyy.

Tämän työ tarkoituksena oli kehittää pieni chattibotti hyödyntämällä tarjolla olevia syväoppimiskehyksiä ja tuottaa dokumentaatio, jota seuraamalla kuka tahansa voi luoda oman chattibottinsa.

Chattibotti on tehty Python-ohjelmointikielellä. Käytetty syväoppimiskehys on PyTorch. Ympäristönä on Googlen Colaboratory, Googlen pilviympäristössä toimiva Jupyter-pohjainen kehitysympäristö. Botin opetusmateriaali on itse kirjoitettu json-tiedosto.

Työn lähteinä on käytetty verkosta löytyvää neuroverkkoihin ja chattibotteihin liittyvää aineistoa.

Avainsanat: Chattibotti, Neuroverkko, Syväoppimiskehys, Tekoäly, PyTorch

Abstract

Author:	Netta Immonen
Title:	Utilization of Deeplearning Framework in Implementation of Chatbot
Number of Pages:	36 pages
Date:	7 November 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technologies
Professional Major:	Software Engineering
Supervisors:	Vesa Ollikainen, Senior Lecturer

Chatbots are increasingly used in more and more places to communicate with customers. The number of chatbots has exploded in recent years. The use of AI is now open to more people than just AI researchers and large companies that can afford to fund research. Deeplearning frameworks and the increasing computing power of computers have democratised the use of AI and enabled even small players to use, for example, chatbotsin their operations. Chatbots enable a company's customers to quickly get accurate and timely information to simple questions from a bot, saving time for both the customer and the customer service agent.

The purpose of the present study was to develop a small chatbot using available deep learning frameworks and produce documentation that anyone can follow to create their own chatbot.
The chatbot is written in the Python programming language. The deep learning framework used is PyTorch. The environment is Google's Colaboratory, a Jupyter-based development environment running in Google's cloud environment. The teaching material for the bot is a self-written json file.

Keywords: Chatbot, Neural Net, Deeplearning Framework, AI, PyTorch

Sisällys

Lyhenteet

API: Ohjelmointirajapinta. Application Programming Interface.

CUDA-tensori:

CUDA-tensorit toimivat samalla tavalla kuin tensorit, mutta ne hyödyntävät GPU:ta.

CPU: Central Processing Unit. Yleissuoritin, joka on suunniteltu suorittamaan laaja-alaisesti erilaisia laskentatehtäviä.

GPU: Graphics Processing Unit. Grafiikkasuoritin. Ei yhtä joustava kuin CPU. Usein tehokkaampi, kun pitää suorittaa monta laskutoimitusta samanaikaisesti.

Json: JavaScript Object Notation. Tiedostomuoto tiedon tallentamista ja välitystä varten. Helppolukuinen ihmisille.

ReLU: Rectified Linear Unit. Aktivointifunktioita on useita erilaisia, joiden soveltuvuus eri tarkoituksiin vaihtelee. Aktivointifunktio määrittää, aktivoituuko neuroverkon solmu. Aktivointifunktio laskee solmun output-arvon. Mikäli output-arvon on suurempi kuin solmun kynnysarvo, solmu aktivoituu ja välittää arvonsa eteenpäin.

Tensori: Moniulotteinen matriisi, joka sisältää yhtä tietotyyppiä olevia elementtejä. Tensoreita käytetään datan säilyttämiseen neuroverkoissa.

1 Johdanto

Erilaiset chattibotit ovat nykyään yleisiä ja niitä käytetään nykyään moniin eri tarkoituksiin. Yksi yleisimmistä sovelluksista on yrityksen tai palveluntarjoajan verkkosivulla oleva asiakaspalveluchattibotti, joka osaa vastata asiakkaiden esittämiin yksinkertaisiin kysymyksiin palvelusta. Chattibottien taustalla on yleensä tekoäly neuroverkon muodossa, mikä mahdollistaa sen, että chattibotti osaa ymmärtää ihmisten käyttämää luonnollista, vaihtelevaa ja joskus epätarkkaa kieltä.

Tekoäly on konseptina ja teorian tasolla ollut olemassa jo 1950-luvulta asti, mutta vasta viimeisen vuosikymmenen teknologian kehitys ja tietokoneiden laskentatehon paraneminen on tehnyt siitä mahdollisen käytännössä. Tekoälyä hyödynnetään lukuisissa paikoissa monilla eri tavoilla. Asiaan perehtymätön ei välttämättä edes huomaa, kuinka yleistä tekoälyn käyttö on. Tekoälyn yleistyminen on myös tarkoittanut sitä, että on kehitetty yksinkertaisia työkaluja tekoälyä hyödyntävien sovellusten luomiseen. Syväoppimiskehykset sisältävät tarvittavat kirjastot ja työkalut neuroverkkopohjaisen tekoälyn luontiin. Näitä käyttämällä voi periaatteessa kuka tahansa luoda oman tekoälyä hyödyntävän sovelluksen helposti ja vaivattomasti joutumatta opettelemaan neuroverkkojen teoriaa.

Tämän työn tavoitteena on testata chattibottisovelluksen rakentamista syväoppimiskehyksen avulla. Työn on myös tarkoitus toimia ohjeistuksena kenelle tahansa, joka haluaa itse kokeilla chattibotin tekemistä.

2 Tekoäly ja chattibotit

2.1 Historia

Tekoäly on tietokoneohjelma, joka kykenee toimimaan älykkäänä pidetyllä tavalla. Tekoälyn historian voidaan katsoa alkavan brittiläisen matemaatikko Alan Turingin vuonna 1950 julkaisemasta artikkelista Computing machinery and intelligence, jossa Turing esittää kysymyksen: ”Voivatko koneet ajatella?”. Artikkelissa Turing esitteli kokeen, joka nykyään tunnetaan nimellä Turingin testi. 50-luvun alkupuolella tekoälykokeilujen toteutuksen tiellä oli kuitenkin teknologia ja hinta. Tietokoneen vuokra saattoi olla jopa 200 000 dollaria kuukaudessa. (1.)

Vuonna 1956 julkistettu Logic Theorist oli Allen Newellin, Cliff Shawn, and Herbert Simonin kehittämä tekoäly, joka ratkoi matemaattisia ongelmia ja oli ensimmäinen tietokone ohjelma, joka pyrki jäljittelemään tapaa, jolla ihminen ratkoo ongelmia (2).

Maailman ensimmäinen chattibotti oli Joseph Weizenbaumin kehittämä ELIZA. 1960-luvun puolivälissä kehitetty ELIZA on luonnollisen kielen prosessointia hyödyntävä ohjelma, jonka on tarkoitus imitoida vastauksia, joita käyttäjä voisi saada terapeutilta. ELIZA tunnisti saamastaan inputista avainsanoja ja lauseita, joiden perusteella se valitsi sopivan vastauksen valmiiksi luodulta listalta. (3.)

Chattibottien kehitys jatkui hitaasti tekniikan kehittyessä. Viimevuosina chattibottien suosio ja määrä on kasvanut räjähdysmäisesti. Chattibotit mahdollistava teknologia on suhteellisen halpaa ja helposti ihmisten saatavilla.

2.2 Turingin testi

Turingin testi on brittiläisen matemaatikko Alan Turingin vuonna 1950 kirjoittamassa artikkelissa Computing machinery and intelligence ideoima testi, jonka tarkoituksena on määrittää, onko tietokoneohjelma älykäs. Testin idea on, että testaaja, ihminen, keskustelee koneen kanssa tietämättä, onko keskustelun

vastapuoli kone vai toinen ihminen. Mikäli testaaja ei pysty keskustelun perusteella päättelemään, että keskustelukumppani oli kone, voidaan konetta pitää älykkäänä.

Viime vuosikymmenen aikana on uutisoitu muutamia kertoja, että jokin ohjelma olisi läpäissyt Turingin testin. Esimerkiksi vuonna 2014 Eugene Goostman -niminen tietokoneohjelma onnistui uskottelemaan 33 prosentille Royal Societyn tuomareista olevansa 13-vuotias poika ukrainasta Readingin yliopiston järjestämässä tapahtumassa (4). Kesällä 2022 Googlella työskennellyt insinööri väitti, että Googlen LaMDA-tekoäly on saavuttanut älykkyyden. Kaikki tunnetut väitteet koneiden todellisesta älykkyydestä ovat kuitenkin vahvasti kiistanalaisia ja yksi kritiikeistä Turingin testiä kohtaan on, että sen läpäiseminen ei todista älykkyyttä vaan koneen kyvyn teeskennellä ihmisyyttä. (5.)

2.3 Vahva ja heikko tekoäly

Turingin testi testaa, onko kone ihmiseen verrattavalla tavalla älykäs. Se testaa niin sanottua vahvaa tekoälyä. Vahva tekoäly on tietoinen itsestään, kykenee ongelmaratkaisuun, oppimiseen ja tulevan suunnitteluun. Vahva tekoäly on tutkimuksen kohteena yliopistoissa ja yrityksessä. Oikeaa vahvaa tekoälyä ei vielä ole olemassa ja joidenkin mielestä sellaisen kehittäminen ei ole edes mahdollista. (6.)

Heikko tekoäly on kapea-alainen tekoäly, joka on luotu suorittamaan tietty tehtävä. Heikkoja tekoälyjä on nykyään monissa eri paikoissa suorittamassa hyvin tarkasti rajattuja tehtäviä. Heikot tekoälyt ovat tehtävissään usein jopa parempia kuin ihminen olisi vastaavassa tehtävässä ja voivat ulkoapäin katsottuna vaikuttaa aidosti älykkäiltä. (7.) Heikkoa tekoälyä käytetään muun muassa seuraaviin tarkoituksiin:

- Kuvantunnistus: Käytössä esim. sosiaalisen media sovelluksissa automaattisesti tunnistamaan kuvissa mukana olevat ihmiset (7) ja itseajavissa autoissa tunnistamaan ympäristö (8).

- Chattibotit ja virtuaaliset avustajat: Sekä yksinkertaiset asiakaspalvelu chattibotit että monimutkaisemmat virtuaaliset avustajat, kuten Applen Siri ja Amazonin Alexa ovat heikkoja tekoälyjä (7).

- Suosittelujärjestelmät: Esimerkiksi verkkokauppojen käyttämiä järjestelmiä, jotka suosittelevat asiakkaille uusia tuotteita aiemman ostokäyttäytymisen perusteella (8).

2.4 Koneoppiminen, syväoppiminen ja neuroverkot

Tekoälystä puhuttaessa ja varsinkin sellaista kehitettäessä tulee äkkiä vastaan termit koneoppiminen (machine learning), syväoppiminen (deep learning) ja neuroverkko (neural net). Niitä käytetään välillä sekaisin tai toistensa tilalla, mikä voi johtaa sekaannukseen termien merkityksestä.

Koneoppiminen on tekoälyn osa-alue, johon kuuluu tilastollisen oppimisen ja optimointimenetelmien käyttö, joiden avulla tietokone voi analysoida datakokonaisuuksia ja tunnistaa malleja. Koneoppimistekniikoissa hyödynnetään tiedonlouhintaa datassa olevien mallien tunnistamiseksi ja tulevien mallien muodostamiseksi. (9.)

Neuroverkot ovat koneoppimisen osa-alue. Neuroverkkojen tarkoituksena on jäljitellä aivojen toimintaa. Ne koostuvat lukuisista solmuista, jotka on järjestetty kerroksiin ja joiden solmut ovat yhteydessä toisiin solmuihin. Jokaisella solmulla on paino ja kynnysarvo, ja mikäli solmun output-arvo ylittää kynnysarvon solmu aktivoituu ja välittään output-arvonsa eteenpäin. Neuroverkossa on aina input-kerros, yksi tai useampi piilokerros ja output-kerros. (10.)

Syväoppiminen on koneoppimisen osa-alue, joka liittyy kiinteästi neuroverkkoihin. Syväoppimisen ”syvä” viittaa käytetyn neuroverkon syvyyteen. Neuroverkkoa, jossa on enemmän kuin kolme kerrosta, input ja output mukaan lukien, voi kutsua syväoppimisalgoritmiksi. (10.)

Koneoppimisalgoritmit voidaan karkeasti jakaa kolmeen: ohjattu oppiminen, ohjaamaton oppiminen ja osin ohjattu oppiminen (11).

Ohjatussa oppimisessa opetusmateriaalina käytetty data on nimikoitu. Algoritmi kehittyy paremmaksi mittaamalla omaa tarkkuuttaan nimikoitua dataa vasten. (12.) Nimikoidulla datalla tarkoitetaan, että data on jo valmiiksi luokiteltu ja siihen on kiinnitetty arvo, joka kertoo neuroverkolle, miten se on luokiteltu.

Ohjaamattomassa oppimisessa algoritmille annettua dataa ei ole nimikoitu. Algoritmi luokittelee sen itse datassa olevien piirteiden ja piilossa olleiden mallien perusteella ilman ihmisapua. (12.)

Osittain ohjatussa oppimisessa osa algoritmille annetusta datasta on nimikoitua. Nimikoitua dataa käytetään laajemman nimikoimattoman datan luokitteluun. (11.)

2.5 Chattibotit

Chattibotti on tietokoneohjelma, joka simuloi keskustelua ihmisen kanssa. Erilaisia chattibotteja käytetään nykyisin hyvin monessa eri paikassa. Yksi yleisimmistä chattibottityypeistä on asiakaspalvelubotti, joka ilmestyy esille ruudun alareunasta monilla internetsivuilla. Asiakaspalvelubotin tarkoitus on vastata asiakkaiden esittämiin yleisiin kysymyksiin. Bottien käyttäminen hyödyttää sekä asiakasta että yritystä. Asiakas saa välittömästi vastauksen kysymykseensä päivästä ja vuorokaudenajasta riippumatta ja yritys hyötyy, kun työntekijöiden aikaa ei kulu kaikkein yleisimpiin ja yksinkertaisimpiin kysymyksiin vastaamiseen.

Chattibotit voidaan karkeasti jakaa kahteen ryhmään: tekstiä generoiva chattibotti ja tekstin hakuun perustuva chattibotti.

Tekstiä generoiva chattibotti nimensä mukaisesti generoi vastauksia, kun sen kanssa keskustelee. Tällainen botti vaatii laajan ja monipuolisen tekstiaineiston opetusmateriaaliksi. PyTorch.org-sivuilla olevan tutoriaali tämänkaltaisesta chattibotista käyttää aineistonaan Cornell Movie-Dialogs Corpusta, joka kattaa

yli 200 000 keskustelua yli 10 000 elokuvahahmon välillä (13). Tällä tavalla opetettu chattibotti pystyy ehkä keskustelemaan sujuvasti ihmisen kanssa, mutta se ei välttämättä sovellu tehtäviin, jossa asiakkaalle tulee antaa tarkka ja yksityiskohtainen vastaus.

Tekstin hakuun perustuvan chattibotin toiminta perustuu siihen, että se oppii ymmärtämään asiakkaiden kysymyksiä, mutta sen sijaan, että se generoisi itse vastauksen kysymykseen, se hakee tietokannastaan sopivan valmiiksi kirjoitetun vastauksen. Hyöty tässä on se, että vastaukset on kirjoittanut ihminen, jolloin voidaan olla varmoja, että botti antaa asiakkaille oikeaa, yksityiskohtaista ja ymmärrettävää tietoa.

3 Kehykset

3.1 Miksi käyttää kehystä?

Neuroverkon rakentaminen alusta lähin itse on täysin mahdollista. Tekemällä kaiken alusta lähtien itse oppii luonnollisesti enemmän siitä, miten neuroverkot ja syväoppiminen toimivat. Neuroverkon luominen itse on kuitenkin aikaa vievä prosessi. Jos tavoitteena ei ole oppia neuroverkkojen teoriaa vaan luoda toimiva neuroverkko helposti, on olemassa lukuisia kehyksiä, jotka auttavat tässä. Syväoppimiskehys tarjoaa työkalut neuroverkon rakentamiseen ja käyttämiseen. Kehyksiä on useita. Tunnetuimmat ovat TensorFlow, Keras ja PyTorch.

3.2 TensorFlow, Keras ja PyTorch

TensorFlow on Googlen Brain Teamin kehittämä avoimen lähdekoodin alusta koneoppimiselle. TensorFlow on alkuperin julkaistu vuonna 2015. TensorFlow lupaa kattavan ja joustavan työkalujen ja kirjastojen ekosysteemin. Lisäksi TensorFlow lupaa sekä aloittelijoille että asiantuntijoille soveltuvia työnkulkuja, joissa on intuitiiviset, korkean tason API:t koneoppimismallien luomiseen eri kielillä. TensorFlow’ta käyttävät monet Googlen tuotteet ja palvelut kuten Haku,

Gmail, Translate, Maps, Android, Kuvat, Speech, YouTube, and Play. (14.) TensorFlow'lla on oma YouTube-kanava, josta löytyy tutoriaaleja, jotka opastavat TensorFlow käytössä (15).

Keras on Googlen kehittämä korkean tason API neuroverkkojen toteuttamiseen. Keras on korkean abstraktionsa takia käyttäjäystävällinen ja helposti opittavissa. Kerasia voi käyttää eri kehysten kanssa ja kehykset, joita Keras tukee, ovat Theano, PlaidML, MXNet, CNTK ja TensorFlow. (16.) TensorFlow on adoptoinut Kerasin osaksi itseään ja Keras tulee paketoituna TensorFlow 2:n mukana tensorflow.keras:na (17).

PyTorch on alun perin FaceBookin (nykyään Meta) kehittämä kehys. Syksyllä 2022 Meta AI ilmoitti, että PyTorchin hallinnointi siirtyy PyTorch-säätiön vastuulle. PyTorch Foundation on osa Linux-säätiötä. (18.) PyTorch lupaa nopeutta, joustavuutta ja tehokkuutta. PyTorch lupaa myös helppokäyttöisen käyttöliittymän, hajautetun opetuksen ja kirjastojen ja työkalujen ekosysteemin. PyTorchin sivuilla on paljon aloittelijoille sopivia tutoriaaleja ja materiaalia PytTorchin opetteluun. (19.)

4 Google Colaboratory

4.1 Mikä on Google Colab?

Google Colabolatory on Googlen ylläpitämä ja kehittämä Jupyter-notebook ympäristö, jossa voi suorittaa Python-koodia. Colab toimii selaimessa eikä edellytä käyttäjältä minkäänlaista asennusta tai latausta. Colab on ilmainen, ja koska Colabin suorittama koodi pyörii Googlen palvelimilla, Colab pienentää niitä laitteisto rajoituksia, joita esimerkiksi neuroverkkojen testaamisella ja kehittämisellä on ja tuo suuremman laskentakapasiteetin yksityiskäyttäjien ulottuviin. (20.)

Koska Colab on ilmainen, se ei aina voi taata resurssien saatavuutta, mutta itselleni ei ole vielä tullut vastaan tilannetta, että resursseja ei olisi ollut saatavilla.

Colabilla on myös Colab Pro, maksullinen versio, jossa rajoitukset ovat lievemmät, ja lisäksi Google Cloud Platform Marketplace, jossa voi ostaa itselleen aikaa virtuaalikoneella, joka soveltuu parhaiten sille suunniteltuun tarkoitukseen. (20.)

Colab on tarkoitettu interaktiivisia käyttäjiä varten, joten esimerkiksi tiedostojen tallennus ja jako, median tarjonta ja muu verkkopalvelujen tarjonta sekä kryptovaluutan louhinta on kielletty. Myös toisiin haitallisesti vaikuttavat asiat kuten Deepfake-väärennösten luominen on kielletty. (21.)

4.2 Colabin valinta

Colab valikoitui käytettäväksi sen helppouden takia, koska se ei edellytä minkäänlaista latausta tai asennusta. Colab on erittäin kevyt myös vanhemmalla tietokoneella, koska koodi pyörii Googlen pilvessä. Colabissa on myös valmiina mm. Keras/Tensorflow ja PyTorch. Kehysten käyttöönotto onnistuu yksinkertaisesti importilla koodin alussa. Kuvassa 1 on oikealla importattu PyTorch ja vasemmalla puolella kuvassa on Tensorflow import toisesta projektista. Kehysten käyttöönoton helppouden ansiosta on myös helppoa lähteä kokeilemaan eri kehysten käyttöä.

```
import tensorflow as tf
pip install tflearn
import tflearn
```

```
import torch
import torch.nn as nn
```

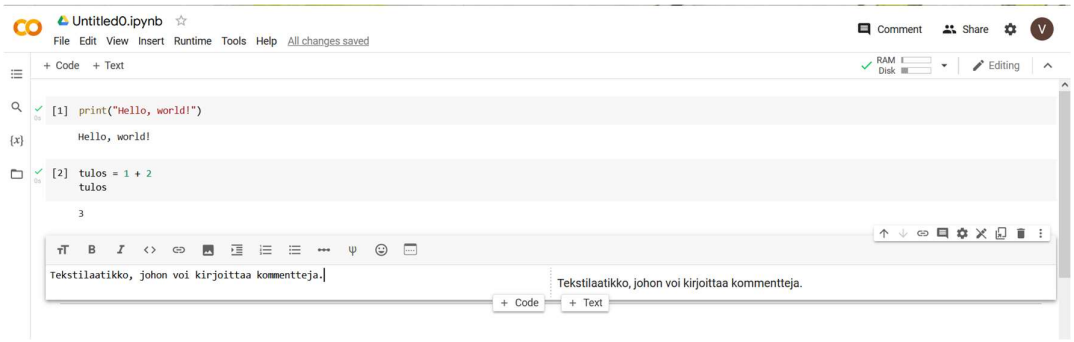
Kuva 1. Vasemmalla TensorFlow import -lauseet oikealla PyTorch import -lauseet

4.3 Erot muihin ohjelmointiympäristöihin verrattuna

Suurin ero Colabin ja aiemmin käyttämieni ohjelmistoympäristöjen kanssa oli rakenne. Muissa käyttämissäni ympäristöissä koodin osat on eritelty omiin tiedostoihinsa: jokainen luokka on omassa tiedostossaan ja ohjelmalla on puumainen

tiedostorakenne. Colabissa ei ole erillisiä tiedostoja eri luokille vaan koko koodi on samassa .ipynb-tiedostossa.

Colab notebookissa on Code- ja Text-laatikot, joihin koodi ja kommentit tulevat. Jokaisen Code-laatikon sisällä olevan koodin voi suorittaa erikseen laatikon yläkulmassa olevasta play-painikkeesta. Kuvassa 2 ”Hello, world!” ja laskutoimitus ovat toisistaan täysin riippumattomia, joten niiden koodit voi suorittaa erikseen. Mikäli eri laatikoissa olevat koodin osat ovat riippuvaisia edeltävissä laatikoissa olevista koodin osista, on edeltävät laatikot luonnollisesti suoritettava ensin. Yläreunan työkalupalkin Runtime-valinnan alta löytyy Run all ,joka suorittaa koko koodin järjestyksessä.



Kuva 2.Colabin käyttöliittymä

Code- ja Text-laatikoita voi lisätä, joko vasemmasta yläreunasta tai valittuna olevan laatikon alareunasta. Kannattaa huomioida se, että lisätty laatikko tulee aina valittuna olevan laatikon alle eikä koodin loppuun.

4.4 Käyttö

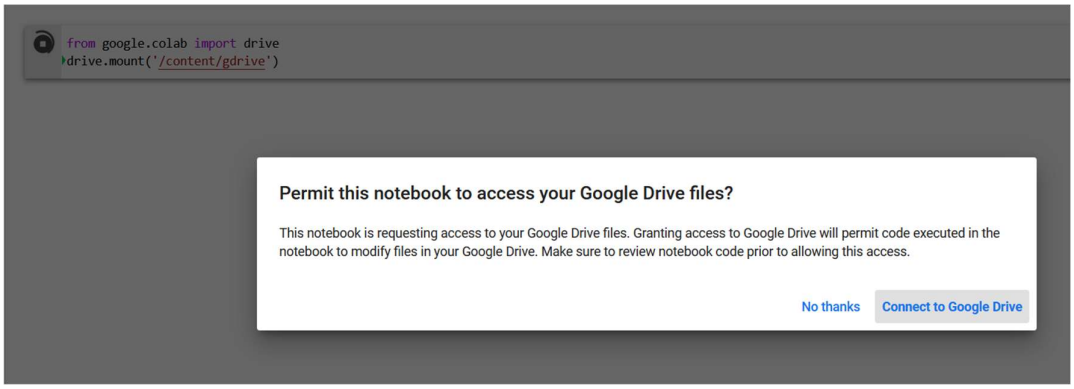
Colabin käyttöä varten tarvitsee vain Google-tilin. Jokaisella, jolla on jo Gmail-osoite, YouTube-tili, Google-drive tai käytössään ylipäätään mikään Googlen palvelu, on jo olemassa Google-tili.

Colabiin pääsee kirjautumaan osoitteesta https://colab.research.google.com/ vaihtoehtoisesti voi suoraan oman Google-driven kautta luoda uuden colab-notebookin. Edellä olleen kuvan 2 esimerkissä oleva koodi ei lukenut eikä kirjoittanut tiedostoja. Mutta mikäli kirjoitettavan koodin on tarkoitus pystyä näin tekemään, tulee notebookin olla yhdistettynä Driveen. Notebookin ja Driven yhdistäminen tapahtuu esimerkkikoodin 1. mukaisesti

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Esimerkkikoodi 1. Colab-muistikirjan yhdistäminen Google-driveen.

Komennon suorittamisen jälkeen Google pyytää lupaa yhdistää muistikirjan Google-driveen alla olevan kuvan 3 mukaisesti.



Kuva 3. Google Driven yhdistäminen Colabiin

5 Chattibotin toteutus

Chattibotin toteutuksen ensimmäinen tehtävä on päätös tekstiä generoivan ja tekstin hakevan botin välillä. Koska tässä projektissa oli tarkoitus tehdä asiakaspalvelubotti, päädyin tekstiä hakevaan chattibottiin edellisessä luvussa määriteltyjen syiden perusteella

Toinen päätös oli kehyksen valinta. Lopullinen valinta tapahtui TensorFlow/Kerasin ja PyTorchin välillä. Valintaan vaikutti suuresti chattibotti tutoriaalien saatavuus. Suurin osa Tensorflow/Keras tutoriaaleista oli tekstiä generoivia botteja. PyTorchille oli saatavilla enemmän tekstin hakuun perustuvia esimerkkejä. Eri tutoriaaleja tutkimalla kävi kuitenkin hyvin nopeasti selväksi, että kovin monta tapaa tehdä chattibottia ei ole ja suurin osa esimerkeistä on hyvin samanlaisia. PyTorch-esimerkit muistuttivat kaikki erittäin paljon YouTuberin Python Engineer tutoriaalisarjaa Chat Bot With Pytorch – NLP And Deep Learning (22). Koska tämä oli kaikkein kattavin esimerkki PyTorchilla tehdystä chattibotista päädyin käyttämään sitä pohjana omalle PyTorch-chattibotilleni.

Vaikka suurin osa TensorFlow/Keras-pohjaisista chattiboteista oli tekstiä generoivia, löysin riittävästi esimerkkejä siitä, miten tekstiä hakeva chattibotti tehdään TensorFlow/Kerasilla (ChattibottiTK), että päätin kokeilla myös tämän vaihtoehdon toteutusta. ChattiBottiTK:n suurin ero PyTorchilla toteutettuun ChattiBottiPT:n oli neuroverkon harjoittamisen hitaus. PyTorch on matalamman tason kirjasto kuin Keras ja eron neuroverkkoja harjoittaessa huomasi selvästi. Tämän eron lisäksi PyTorch-koodi on jonkin verran pidempi. Neuroverkon luomiseen menee enemmän rivejä PyTorchilla tehdessä kuin TensorFlow/Kerasilla. Koska suurin ero oli kuitenkin ChattibottiTK:n harjoittamisen hitaus, oli lopullinen valinta PyTorchilla tehty ChattiBottiPT.

5.1 JSON-tiedosto

Json-tiedosto toimii sekä chattibotin opetusmateriaalina että lähteenä, josta se hakee vastaukset sille esitettyihin kysymyksiin. Chattibotti opetetaan ohjatun opetuksen tekniikalla. Koodiesimerkissä 2 ”tag” on ”greeting”, mikä kertoo chattibotille, että tämän intentin patterns-listassa määritetyt tekstit ovat esimerkkejä tervehdyksistä, joita botti saattaa kohdata ja jotka sen pitäisi pystyä tunnistamaan nimenomaisesti tervehdyksiksi. Responses-lista määrittää vastaukset, joista chattibotti valitsee, kun se tunnistaa, että sitä tervehditään.

```
"intents": [
        {
            "tag" : "greeting",
            "patterns": [
                "Hi",
                "Hey",
                "How are you",
                "Good morning",
                "Good evening",
                "Hello",
                "Good day"
            ],
            "responses": [
                "Hello",
                "Hello, what can I do for you?",
                "Hello, how can I help you?"
            ]
        },
```

Esimerkkikoodi 2. Json-tiedostossa oleva lista esimerkkitervehdyksistä ja niiden vastaukset. (22.)

Chattibotin pitää luonnollisesti pystyä tekemään muutkin kuin tervehtimään käyttäjää. Tässä vaiheessa tuli ajankohtaiseksi päättää, mitä tässä projektissa tehtävän chattibotin pitäisi osata tehdä. Alkuperäisenä ideana oli, että botti olisi asiakaspalvelu chattibotti, ja se idea tarkentui nyt siten, että botista tulisi kuvitteellisen kukkakaupan asiakaspalvelubotti. Kukkakaupalla olisi pari kivijalkakauppaa ja verkkokauppa ja botin tulisi osata vastata yksinkertaisiin kysymyksiin kaupan sijainnista, aukioloajoista, laskutuksesta, toimituksesta ja ongelmista toimituksessa.

Json-tiedostoa valmistellessa suurimmaksi ongelmaksi osoittautui se, miten helposti botti ymmärsi väärin sille annetun kysymyksen, jos patternien, eli esimerkkilauseiden valinnan kanssa ei ollut huolellinen. Esimerkiksi sana ”sell” osoittautui ongelmaksi, jos sitä esiintyi kahden eri tagin alla olevissa esimerkkilauseissa, koska botti meni sekaisin sen suhteen, kumpaa tarkoitettiin.

5.2 Alkutoimet

Kuten jo aiemmin mainittiin Colabissa koko koodi on samassa tiedostossa. Käytännössä tämä tarkoittaa sitä, että kaikki koodissa tarvittavat importit voi laittaa koodin alkuun. Koodin eriluokissa määriteltyjä funktioita ei myöskään tarvitse

tuoda import-lauseella niihin luokkiin, missä niitä käytetään. Importit tulee vain määritellä ennen kuin niitä tarvitaan. Itse katsoin helpoimmaksi määrittää kaikki import-lauseet koodin alussa esimerkkikoodin 3 mukaisesti.

```
import numpy as np
import random
import json
import nltk
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from nltk.stem.porter import PorterStemmer
nltk.download('punkt')

stemmer = PorterStemmer()
```

Esimerkkikoodi 3. Chattibotin edellyttämät import-lauseet. (22.)

Näistä importeista numpy ja random ovat metematiikkakirjastoja torch on itse PyTorch ja nltk, eli Natural Language Tool Kit on tekstin käsittelyä varten.

5.3 Tekstin käsittely

Chattibotin on tarkoitus luoda illuusio siitä, että käyttäjä keskustelee jonkun kanssa tekstin välityksellä. Chattibotti ei kuitenkaan kykene ymmärtämään tekstiä samalla tavalla kuin ihminen.

Jotta chattibotti voisi käsitellä sille syötettyä tekstiä, se on ensin muokattava numeeriseen muotoon. Tekstin käsittelyyn käytetään kolmea funktiota: tokenisaatio, typistäminen ja sanapussi.

5.3.1 Tokenisaatio

Tokenisaatio on luonnollisen kielen käsittelyn menetelmä, joka jakaa teksti pienempiin tokeneiksi kutsuttuihin osiin. Tokenit voivat olla joko sanoja, merkkejä

tai sanan osia. Yleisin tapa muodostaa tokeneja on käyttää erottimena sanavä-liä. (23.) Esimerkkikoodin 4 tokenize()- funktio ottaa String-olion, jakaa sen to-keneihin ja palauttaa String-taulukon.

```
def tokenize(sentence):
    return nltk.word_tokenize(sentence)
```

Esimerkkikoodi 4. Tokenize()-funktio. (22.)

Esimerkkikoodi 5 näyttää tokenize() muodostaa taulukon lauseesta.

```
a = "Tokenize this sentence"
print(a)
a = tokenize(a)
print(a)

Tokenize this sentence
['Tokenize', 'this', 'sentence']
```

Esimerkkikoodi 5. Tokenize()-funktion toiminta. (22.)

5.3.2 Typistäminen

Stemmaus tai typistäminen on toinen luonnollisen kielen käsittelyn menetelmä. Typistämisen tarkoitus on poistaa sanasta sanan perässä olevat sanaliitteet, jol-loin jäljelle jää vain sanan vartalo. Typistäminen on kuitenkin melko karkea heuristinen prosessi, joka katkaisee sanojen päät siinä toivossa, että tämä ta-voite saavutetaan useimmiten oikein. (24.) Typistämistä varten on olemassa eri algoritmeja, joilla saattaa saavuttaa eri tuloksen. Yksi käytetyimmistä englannin-kielisistä typistysalgoritmeista on Porterin algoritmi, jonka on empiirisesti todis-tettu olevan hyvin tehokas. (24.) Typistäjä, jota käytin tässä projektissa, on Por-terStemmer, joka käyttää Porterin algoritmia (25). Typistämisen lisäksi tämä funktio huolehtii siitä, että kaikki sanat on kirjoitettu pienellä. Esimerkkikoodissa 6 näkyy myös typistämisen heikkous, kun ideasta ”to organize” tulee sama kuin ”organ”.

```
def stem(word):
    return stemmer.stem(word.lower())

words = ["Organize", "organizes", "organizing"]
stemmed_words = [stem(w) for w in words]
print(stemmed_words)


['organ', 'organ', 'organ']
```

Esimerkkikoodi 6. Stem()-funktio typistää sanat jättäen jäljelle vain sanan vartalon. (22.)

5.3.3 Sanapussi

Viimeinen tekstin käsittelyn vaihe on luoda sanapussi, bag of words. Sanapussi on keino kuvata sanojen esiintymistä tekstissä. Sitä kutsutaan pussiksi, koska sanojen järjestyksellä ei ole väliä, vain sanan esiintyminen tekstissä merkitsee. (26)

```
"Do you take credit cards?",
"Do you accept Mastercard?",
```

Jos otetaan yllä olevat lauseet, niin niissä on yhteensä seitsemän uniikkia sanaa, jotka ovat do, you, take, credit, cards, accept, mastercard. Nämä seitemän sanaa ovat tässä esimerkkitapauksessa koko sanalista. Varsinaisessa chattibotissa sanalista muodostuu kaikista json-tiedoston esimerkkilauseista.

Otamme toisen yllä olevista lauseista ja vertaamme sanalistaa siihen. Mikäli sanalistan sana löytyy lauseesta, merkitään 1 ja jos ei merkitään 0. Lauseen ”Do you accept Mastercard” esitys on siis [1, 1, 0, 0, 0, 1, 1]. Taulukossa 1 on selkeämpi esitys siitä, miten sanapussi muodostuu ja esimerkkikoodissa 7 bag_of_word()-funktio, jolla tämä toteutetaan.

Taulukko 1. Bag of Words -esitys taulukkomuodossa

do	you	take	credit	cards	accept	master-card
1	1	0	0	0	1	1

```
def bag_of_words(tokenized_sentence, words):
    sentence_words = [stem(word) for word in tokenized_sentence]
    bag = np.zeros(len(words), dtype=np.float32)
    for idx, w in enumerate(words):
        if w in sentence_words:
            bag[idx] = 1

    return bag
```

Esimerkkikoodi 7. Bag_of_words()-funktio muokkaa luonnollisen kielen lauseet muotoon, jota chattibotti voi ymmärtää. (22.)

5.4 Opetusdatan valmistelu

Tekstin käsittelyn jälkeen on vuorossa harjoitusdatan luominen. Harjoitus datana toimii aiemmin luotu json.tiedosto, jonka avaaminen esimerkkikoodi 8 mukaisesti.

```
with open ('/content/gdrive/MyDrive/jsontesti.txt') as json_data:
  intents = json.load(json_data)
```

Esimerkkikoodi 8. Json-tiedoston avaaminen Colabissa

Sulkujen sisällä oleva osoite on sijainti, jossa jsontesti.txt-tiedosto on. Tämän kutsun on tapahduttava ennen kuin seuraavat koodin osat suoritetaan.

Luodaan tyhjät taulukot all_words, tags ja xy.

```
all_words = []
tags = []
xy = []
```

Esimerkkikoodi 9. Luodaan tyhjät taulukot all_words, tags ja xy. (22.)

Json-tiedosto sisältää ”intenttejä” eli ”aikeita”/”tarkoituksia”. Jokainen aikomus, eli intent, sisältää nimikointi tiedon ”tag” ja esimerkkilauseet sisältävän patterns-listan sekä responses-listan, joka sisältää vastaukset näihin esimerkkilauseisiin. Esimerkkikoodi 10 mukaisesti for-toistorakenteella käydään läpi json-tiedosto intent kerrallaan. Jokaisen intentin tag-arvo otetaan talteen ja lisätään tags-taulukkoon. Minkä jälkeen käydään toistorakenteella läpi intenttien sisällä olevat pattern-esimerkkilauseet.

```
for intent in intents['intents']:
    tag = intent['tag']
    tags.append(tag)
    for pattern in intent['patterns']:
        w = tokenize(pattern)
        all_words.extend(w)
        xy.append((w, tag))
```

Esimerkkikoodi 10. Intenttien läpikäyminen sisäkkäisissä toistorakenteissa. (22.)

Nämä esimerkkilauseet tokenisoidaan edellä määritetyllä Tokenize()-funktiolla ja tokensoidut esimerkkilauseet lisätään all_words-taulukkoon.

Tässä käytetään appendin sijasta extendiä, sillä tarkoitus ei ole luoda taulukoita taulukoiden sisällä.

Pattern ja siihen kuuluva tag lopuksi lisätään taulukkoon xy. Lopputuloksena on siis kolme taulukkoa: yksi, joka sisältää tokenisoidut patternit, yksi, joka sisältää tagit ja yksi, joka sisältää tokenisoidut patternit ja niihin liittyvät tagit.

Sanojen joukosta täytyy myös poistaa välimerkit, joten esimerkkikoodin 11 mukaisesti luodaan ignore_words-taulukko, jossa määritetään sanat, eli tässä tapauksessa välimerkit, joita ei huomioida. Tämän jälkeen syötetään all_words-taulukko aiemmin määritetylle stem()-funktiolle.

Set-funktio poistaa duplikaatit ja sorted()-funktio tekee siitä järjestetyn listan. Tags-taulukossa ei tässä tapauksessa ole duplikaatteja, mutta sen voi myös varmuuden vuoksi tarkistaa.

```
ignore_words = ['?', '.', '!', ',']
all_words = [stem(w) for w in all_words if w not in ignore_words]
all_words = sorted(set(all_words))
tags = sorted(set(tags))
```

Esimerkkikoodi 11. Välimekkien poistaminen ja sanalistan järjestäminen. (22.)

Luodaan tyhjät x_train- ja y_train-taulukot. Esimerkkokoodi 11 mukaisesti Xy-taulukossa oleva data käydään toistorakenteenavulla läpi. Taulukossa olevat pattern-esimerkkilauseet ja all_words-taulukko annetaan aiemmin määritetylle bag_of_words()-funktiolle, jonka palauttama numeerinen data lisätään x_train-taulukkoon. Xy-taulokossa olevien tagit muutetaan numeerisiksi ja nämä label-arvot lisätään y_train-taulukkoon.

```
X_train = []
y_train = []
for (pattern_sentence, tag) in xy:
    bag = bag_of_words(pattern_sentence, all_words)
    X_train.append(bag)
    label = tags.index(tag)
    y_train.append(label)
```

Esimerkkikoodi 12. X_train- ja y_train-taulukoiden luominen ja täyttäminen. (22.)

Tämän jälkeen x_train- ja y_train-taulukot muutetaan numpy-muotoon, kuten alla oleva esimerkkikoodi 13 näyttää.

```
X_train = np.array(X_train)
y_train = np.array(y_train)
```

Esimerkkikoodi 13. X_train- ja y_train-taulikoiden muokkaus numpy-muotoon. (22.)

Seuraavaksi luodaan luokka ChatDataset, joka perii Datasetin. ChatDatasetin sisään tulee kolme funktiota. Init ja len saavat parametrinä vain itsensä, getitem saa myös indexin.

Init-funktion sisään tulee, esimerkkikoodin 14 mukaisesti, n_samples-muuttujaan x_train-taulukon pituus ja x_train- ja y_train-taulukot muuttujiin x_data ja y_data.

```
class ChatDataset(Dataset):

    def __init__(self):
        self.n_samples = len(X_train)
        self.x_data = X_train
        self.y_data = y_train
```

Esimerkkikoodi 14. Init-funktio. (22.)

Getitem-funktion, esimerkkikoodi 15, takaa sen, että dataan pääsee käsiksi index-arvojen avulla

```
    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]
```

Esimerkkikoodi 15. Getitem-funktio. (22.)

Len-funktio, esimerkkikoodi 16, palauttaa n_samples-muuttujan arvon, eli x_train-taulukon pituuden.

```
    def __len__(self):
        return self.n_samples
```

Esimerkkikoodi 16. Len-funktio. (22.)

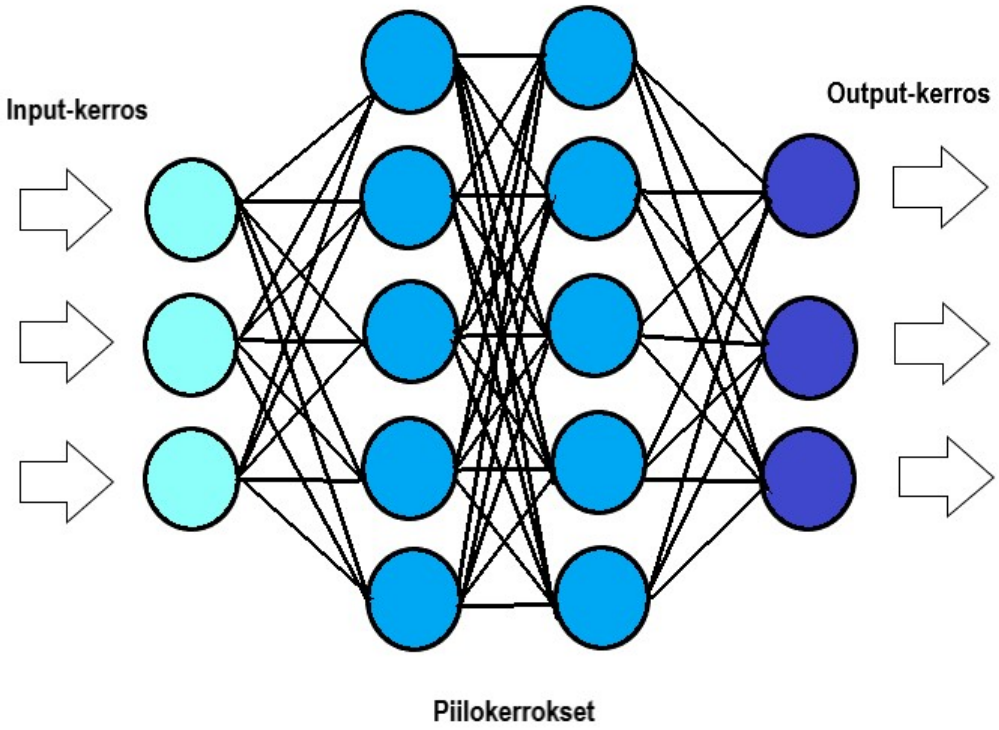
Tämän jälkeen, esimerkkikoodi 17, luodaan dataset ja dataloader.

```
dataset = ChatDataset()
train_loader = DataLoader(dataset=dataset,
                          batch_size=batch_size,
                          shuffle=True,
                          num_workers=0)
```

Esimerkkikoodi 17. Dataset ja dataloader ovat PyTorchista import-lauseella tuotuja työkaluja. (22.)

5.5 Malli

Tämä chattibotti perustuu feedforward-tyyppiseen neuroverkkoon. Feedfoward-neuroverkoissa data liikkuu vain yhteen suuntaan. Data saattaa kulkea usean solmun läpi, mutta toisin kuin recurrent-tyyppisissä neuroverkoissa sitä ei kierrätetä. Kuvassa 4 näkyy feedforward-tyyppisen neuroverkon rakenne.



Kuva 4. Neuroverkon rakenne

Mallia varten luodaan luokka NeuralNet, jonka tulee periytyä nn.Modulesta. Nn.Module tulee jälleen PyTorchista. Esimerkkikoodissa 18 neuroverkon luominen luokan sisälle tuleva init-funktio saa parametrinä itsensä lisäksi input- ja piilokerrosten koon ja luokkien määrän.

Init-funktion sisällä määritetään neuroverkon kerrosten määrä. Ensimmäinen kerros saa parametrinä inputkerroksen koon ja piilokerroksen koon, välissä olevat kerrokset saavat molempina parametreinä piilokerroskoon, ja viimeinen kerros saa ensimmäisenä parametrinä piilokerroksen koon ja toisena luokkien määrän. Luokkien määrällä tarkoitetaan tässä json-tiedostossa olevien tagien määrää. Output on siis neuroverkon päätelmä siitä, mitä sille on sanottu.

Loppuun lisätään aktivointifunktio, joka tässä chattibotissa on ReLU eli Rectified Linear Unit. Aktivointifunktioita on useita erilaisia, joiden soveltuvuus eri tarkoituksiin vaihtelee. Aktivointifunktio määrittää, aktivoituuko neuroverkon solmu. Aktivointifunktio laskee solmun output-arvon, mikäli output-arvon on suurempi kuin solmun kynnysarvo, solmu aktivoituu ja välittää arvonsa eteenpäin.

```
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.l1 = nn.Linear(input_size, hidden_size)
        self.l2 = nn.Linear(hidden_size, hidden_size)
        self.l3 = nn.Linear(hidden_size, hidden_size)
        self.l4 = nn.Linear(hidden_size, num_classes)
        self.relu = nn.ReLU()
```

Esimerkkikoodi 18. Neuroverkon luominen. (22.)

Tämä jälkeen toteutetaan forward()-funktio, esimerkkikoodi 19, joka saa parametrina itsensä ja x. Jokainen kerros, joka määritettiin edellisessä vaiheessa, tulee huomioida tässä. Tämä funktio palauttaa outputin.

```
    def forward(self, x):
        out = self.l1(x)
        out = self.relu(out)
        out = self.l2(out)
        out = self.relu(out)
        out = self.l3(out)
        out = self.relu(out)
        out = self.l4(out)
        return out
```

Esimerkkikoodi 19. Forward()-funktio. (22.)

Tämän jälkeen luodaan malli. Jos käytössä olisi toisenlainen ohjelmointiympäristö mallin luominen tapahtuisi training-luokassa, mutta Colabissa riittää, että se on luotu määrittelyn jälkeen.

Model saa parametreinaan neuroverkon kerrosten koot. Device-koodilla, esimerkkikoodi 20, testataan, onko käytössä GPU ja mikäli on .to(device) määrittää koodin käyttämään GPU:ta. CUDA-tensorit toimivat samalla tavalla kuin CPU-tensorit, mutta ne hyödyntävät GPU:ta (27).

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = NeuralNet(input_size, hidden_size, output_size).to(device)
```

Esimerkkikoodi 20. Device ja model. (22.)

CrossEntropyLoss on funktio, jota on tässä chattibotissa käytetty laskemaan neuroverkon loss-arvo. Mallia opettaessa jokaisen ajokierroksen jälkeen loss-funktio vertaa mallin antamaan output-arvoa odotettuun. Opetuksen tarkoituksena on minimoida näiden välinen ero, loss. Optimizer-funktio muuttaa mallin painoja loss-arvon perusteella eron pienentämiseksi. Tämä tunnetaan myös nimellä takaisinkytkentä. Esimerkkikoodissa 21 on loss-funktio ja optimizer.

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

Esimerkkikoodi 21. Määritetään loss-funktio ja optimizer. (22.)

5.6 Varsinainen opettaminen

Neuroverkon opetus tapahtuu toistorakenteessa, esimerkkikoodi 22. Toistojen määrä on määritelty num_epochs-hyperparametrissä.

```
for epoch in range(num_epochs):
    for (words, labels) in train_loader:
        words = words.to(device)
        labels = labels.to(dtype=torch.long).to(device)
```

Esimerkkikoodi 22. Mallin opettaminen toistorakenteessa. (22.)

Eteenpäin syöttö tapahtuu kutsumalla modelia, ja loss lasketaan kutsumalla criterionia. (Esimerkkikoodi 23).

```
        # Forward
        outputs = model(words)
        loss = criterion(outputs, labels)
```

Esimerkkikoodi 23. Eteenpäinsyöttö ja loss-arvon laskeminen. (22.)

Takisinkytkennän, alla esimerkkikoodi 24, laskemista varten optimizer pitää ensin tyhjentää, mikä tapahtuu zero_grad()-funktiolla, tämä jälkeen loss.backward()-funktio laskee takaisin kytkennän eli kuinka verkon painoja tulee muuttaa. Viimeisänä oleva step()-funktio tekee muutokset.

```
        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Esimerkkikoodi 24. Neuroverkon takaisinkytkentä. (22.)

Esimerkkikoodi 25 tulostaa näytölle opetuksen etenemistä sen seurannan helpottamiseksi. Tämän voi luonnollisesti jättää pois, jos haluaa.

```
    if (epoch+1) % 100 == 0:
        print (f'Epoch [{epoch+1}/{num_epochs}],
                    Loss: {loss.item():.4f}')
```

Esimerkkikoodi 25. Opetuksen etenemisen tulostus. (22.)

5.7 Mallin tallentaminen

Neuroverkon malli tallennettaan muuttujaan, joka on tässä nimeltään data. Esimerkkikoodissa 26 näkyvät tallennettavat tiedot.

```
data = {
"model_state": model.state_dict(),
"input_size": input_size,
"hidden_size": hidden_size,
"output_size": output_size,
"all_words": all_words,
"tags": tags
}
```

Esimerkkikoodi 26. Neuroverkon mallin tallentaminen käyttöä varten. (22.)

Colabia käyttäessäni tämä data-muuttujan luonti oli samassa koodilaatikossa opetustoistorakenteen kanssa, mutta seuraavat koodin osiot, jotka näkyvät alla esimerkkikoodissa 27, olivat omassa laatikossaan.

```
FILE = "data.pth"
torch.save(data, FILE)

print(f'training complete. file saved to {FILE}')
```

Esimerkkikoodi 27. Data.pth-tiedoston tallentaminen. (22.)

Tiedoston pääte .pth tarkoittaa, että kyseessä on PyTorch tiedosto.

5.8 Neuroverkon käyttäminen chattibotissa

Json-tiedostoon on jo luotu yhteys koodin alussa, joten Colabissa sitä ei ole tarpeen luoda uudelleen tässä vaiheessa. Mikäli käytössä olisi toisenlainen ympäristö, tulisi json-tiedosto avata tässä. Tässä tarvitsee sen sijaan vain avata aiemmin tallennettu tiedosto ja hakea siitä arvot vastaaviin muuttujiin ja luoda ja ladata malli, kuten esimerkkikoodissa 28 näkyy.

```
FILE = "data.pth"
data = torch.load(FILE)

input_size = data["input_size"]
hidden_size = data["hidden_size"]
output_size = data["output_size"]
all_words = data['all_words']
tags = data['tags']
model_state = data["model_state"]

model = NeuralNet(input_size, hidden_size, output_size).to(device)
model.load_state_dict(model_state)
model.eval()
```

Esimerkkikoodi 28. Data.pth-tiedoston avaaminen. (22.)

Viimeinen vaihe chattibotin luomisessa on itse chattifunktion toteuttaminen.

Tässä botille annetaan nimi ja aloitetaan chatin toistorakenne, josta pääsee poistumaan kirjoittamalla chatti-ikkunaan ”quit”. Koodiesimerkissä 29 sentence-muuttuja on käyttäjän antama syöte. Syöte tokenisoidaan ja muutetaan mallin

edellyttämään muotoon, minkä jälkeen se muutetaan torch-tensoriksi. Tensori on moniulotteinen matriisi, joka sisältää yhtä tietotyyppiä olevia elementtejä (28). Tensoreita käytetään datan säilyttämiseen neuroverkoissa.

```
bot_name = "Bot"
print("Ready! (type 'quit' to exit)")
while True:
    sentence = input("You: ")
    if sentence == "quit":
        break

    sentence = tokenize(sentence)
    X = bag_of_words(sentence, all_words)
    X = X.reshape(1, X.shape[0])
    X = torch.from_numpy(X).to(device)
```

Esimerkkikoodi 29. Chatin toistorakenteen aloitus ja syötteen käsittely. (22.)

Jos malli on riittävän varma, eli yli 75 prosentin todennäköisyydellä tunnistaa, mitä sille on sanottu, käydään läpi json-tiedostossa olevat intentit ja tagit ja tulostetaan satunainen vastaus oikean intentin alla olevan response-listan vastauksista. Mikäli varmuus on alle 75 prosenttia, chattibotti tulostaa kovakoodatun ”I’m sorry, but I don’t understand” -vastauksen. Alla koodiesimerkissä 30.

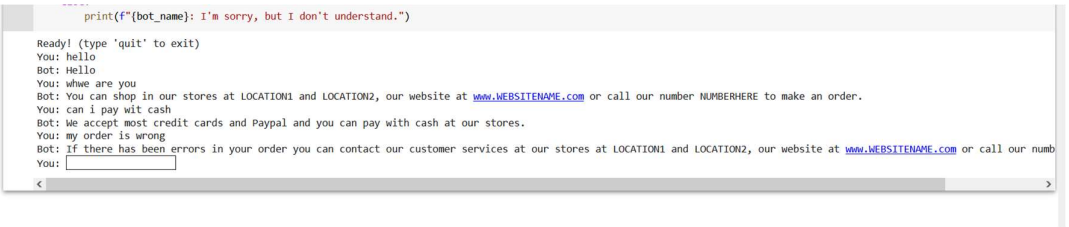
```
    output = model(X)
    _, predicted = torch.max(output, dim=1)

    tag = tags[predicted.item()]

    probs = torch.softmax(output, dim=1)
    prob = probs[0][predicted.item()]
    if prob.item() > 0.75:
        for intent in intents['intents']:
            if tag == intent["tag"]:
                print(f"{bot_name}: {random.choice(intent['respons-
es'])}")
    else:
        print(f"{bot_name}: I'm sorry, but I don't understand.")
```

Esimerkkikoodi 30. Mikäli malli tunnistaa annetun syötteen yli 75 prosentin todennäköisyydellä, tulostetaan satunnainen vastaus oikean intentin alta responses-listalta. (22.)

Tämän jälkeen chattibotti toimii ja siltä voi kysellä tietoja kukkakaupan sijainnista, inventaariosta, aukio-oloajoista ja toimituksesta. Kuvassa 5 chattibotti on vastannut kysymyksiin siitä, mitä kauppa tekee, missä se sijaitsee ja myydäänkö kaupassa narsisseja.

Kuva 5. Onnistunut keskustelu chattibotin kanssa

Kuvassa 6 näkyy, että chattibotti myös ymmärtää yksinkertaisia kirjoitusvirheitä sille annetuissa syötteissä ja on osannut vastata oikein sijaintia ja maksuvälineitä koskeviin kysymyksiin. Kuvassa 7 näkyy, että botti ymmärsi, että ”montra” on tarkoittanut monsteraa. Jos botti ei ymmärrä sille annettua syötettä, se vastaa käyttäjälle ”I’m sorry, but I don’t undestand.



Kuva 6. Chattibotti ymmärtää yksinkertaisia kirjoitusvirheitä annetussa syötteessä.

```
        for intent in intents['intents']:
            if tag == intent["tag"]:
                print(f"{bot_name}: {random.choice(intent['responses'])}")
    else:
        print(f"{bot_name}: I'm sorry, but I don't understand.")
```

```
Ready! (type 'quit' to exit)
You: do you sell montra
Bot: Yes, you can see our full and up to date inventory at our website www.WEBSITENAME.com
You: llkkkl
Bot: I'm sorry, but I don't understand.
You:
```

Kuva 7. Vastaus, kun botti ei ymmärrä, mitä sille sanotaan

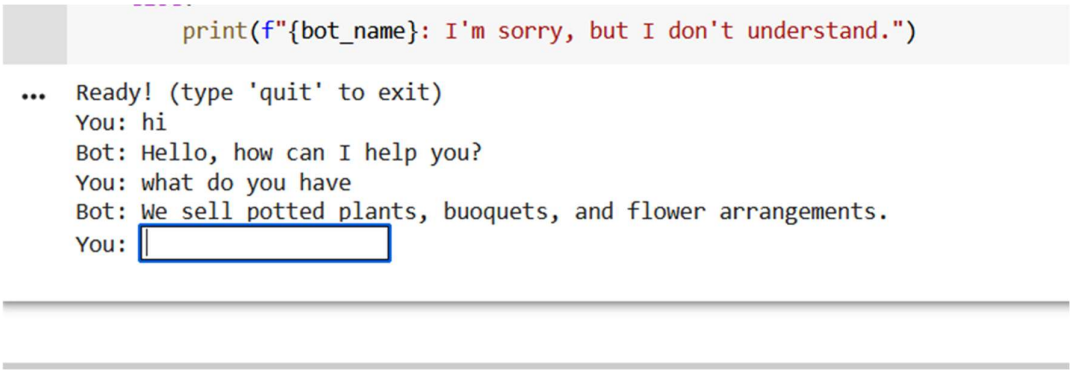
Näitä kuvia ottaessa chattibotti ymmärsi kirjoitusvirheet erittäin hyvin. Enemmän ongelmia tuli sen kanssa, että botti arvioi annetun syötteen väärin kuten kuvassa 8 ja vastasi väärältä responses-listalta.

```
        for intent in intents['intents']:
            if tag == intent["tag"]:
                print(f"{bot_name}: {random.choice(intent['re
    else:
        print(f"{bot_name}: I'm sorry, but I don't understand
```

```
Ready! (type 'quit' to exit)
You: hi
Bot: Hello
You: what do you have
Bot: Bye! Thank you for visiting.
You:
```

Kuva 8. Väärä vastaus

Nämä ongelmat väärien vastausten kanssa enimmäkseen johtuvat json-tiedostosta ja korjaantuvat päivittämällä ja laajentamalla sitä. Ne ovat ehkä ennemminkin json-tiedoston tekijän virhe kuin ongelma botissa itsessään. Yllä oleva ongelma korjautui muuttamalla json-tiedostoa, kuten kuvassa 9 alla näkyy.



Kuva 9. Pieni muutos json-tiedostoon ja chattibotti ymmärtää, mitä siltä halutaan.

5.9 Hyperparametrit

Hyperparametrit määrittävät neuroverkon rakenteen. Koodissa ne ovat training-funktioden jälkeen, mutta tuntui selkeimmältä käsitellä ne erikseen. Kun chattibotin on saanut toimimaan, näitä muuttujia muuttamalla voi muuttaa sitä, miten botti oppii. Osa, kuten output_size ja input_size, määräytyy json-tiedoston perusteella, mutta muita voi muuttaa ilman, että koodi hajoaa. Esimerkkikoodissa 31. on chattibotin käyttämät hyperparametrien arvot.

```
num_epochs = 1000
batch_size = 8
learning_rate = 0.001
input_size = len(X_train[0])
hidden_size = 8
output_size = len(tags)
```

Esimerkkikoodi 31. Chattibotin käyttämät hyperparametrit. (22.)

Epoch-muuttuja määrittää, kuinka monta kertaa algoritmi käy läpi koko datasetin. Batch_size määrittää, kuinka monta datanäytettä käydään läpi ennen sisäisen mallin parametrien päivittämistä (29). Learning_rate määrittää, kuinka paljon mallin painoja muutetaan, kun sitä päivitetään. Input_size on kunkin sanapussin pituus, mikä on sama kuin all_words-taulukon pituus. Tässä esimerkissä on käytetty ensimmäistä x_train[0]-sanapussia. Hidden_size on piiloker-

rosten koko. Tätä voi halutessaan muuttaa, ja output_size on json-tiedoston tagien lukumäärä. Json-tiedoston tagien määrä määrittää sen, kuinka monta erilaista kysymystä/kommunikaatiotyyppiä chattibotti ymmärtää.

6 Yhteenveto

Työn tarkoituksena oli testata chattibotin tekemistä syväoppimiskehyksen avulla ja tuottaa dokumentaatio, jota seuraamalla kuka tahansa voi tehdä saman. Chattibotin tekeminen itsessään oli suhteellisen yksinkertaista. Saatavilla olevat tutoriaalit ovat hyviä, joskin kaikki täysin samanlaisia ja Colab on näin pienessä projektissa hyvin toimiva ympäristö.

Työn perusteella voin sanoa, että kehystä käyttämällä chattibotin tekeminen oli yksinkertaista. Kehyksien käyttäminen edellyttää jonkin verran koodaamistaustaa ja jonkin verran ymmärrytä neuroverkoista, mutta huomattavasti vähemmän kuin jos pyrkisin tekemään saman itse. Jos on tarkoitus käyttää Pythonia, ainakin pieni kokemus kielestä on hyödyllinen, muttei täysin välttämätön. Näin pienessä projektissa sillä, mitä kehystä loppujen lopuksi käytti, ei ollut kovin paljon merkitystä, ja joku toinen olisi voinut tulla samoilla perusteluilla toiseen lopputulokseen.

Suurin haaste itse työssä oli json-tiedoston muokkaaminen sellaiseen muotoon, että se on riittävän kattava, jotta chattibotti ymmärtää mahdollisimman paljon, mutta että eri intenteissä olevat samat sana eivät sekoita bottia. Työhön liittyvä, mutta osin sen ulkopuolinen haaste, jolla oli suuri vaikutus työhön, oli se, että kehysten asentaminen omalle tietokoneelleni osoittautui täysin mahdottomaksi tehtäväksi. Sekä TensorFlow'lla että PyTorchilla on erittäin hyvät asennusohjeet omilla sivuillaan, ja vaikka koneeni on jo vanhempi, siinä pitäisi olla riittävästi tilaa asennuksille. Asennusyritysten jatkuva epäonnistuminen oli se tekijä, joka sai minut etsimään tapaa suorittaa koodia selaimessa ja johti Google Colabin löytymiseen.

Kuten sanottu Colab on näin pienessä projektissa oikein toimiva vaihtoehto. Mutta jos tavoitteena olisi tehdä isompi työ, tulisi ehdottomasti löytää tapa jakaa koodia erillisiin osiin sen sijaan, että kaikki on vain samassa pötkössä. Koodi ja kommenttilaatikot eivät mielestäni ole riittäviä pitämään isomman projektin rakenteen kasassa.

Lähteet

1. The History of Artificial Intelligence. 2017. Verkkoaineisto. Harvard University blog. Rockwell Anyoha. < https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/ > Luettu 27.07.2022.

2. Logic Theorist Explained – Everything You Need To Know. 2021 Vekkoaineisto. History Computer. < https://history-computer.com/logic-theorist/> Luettu 2.11.2022.

3. ELIZA--A Computer Program For the Study of Natural Language Communication Between Man and Machine. 1966. Verkkoaineisto. Joseph Weizenbaum. <http://www.universelle-automation.de/1966_Boston.pdf> Luettu 2.11.2022.

4. Computer passes Turing test in 'world first'. 2014. Verkkoartikkeli. BBC. <https://www.bbc.com/news/technology-27762088> Luettu 27.07.2022.

5. Google's AI passed a famous test — and showed how the test is broken. 2022. Verkkoaineisto. The Wahsington Post. Will Oresmus. <https://www.washington-post.com/technology/2022/06/17/google-ai-lamda-turing-test/ >Luettu 27.07.2022.

6. Strong AI. 2020. Verkkoaineisto. IBM Cloud Education. < https://www.ibm.com/cloud/learn/strong-ai > Luettu 27.07.2022.

7. narrow AI (weak AI). 2021. Verkkoaineisto. TechTarget. Mark Lagge, Ivy Wigmore <https://www.techtarget.com/searchenterpriseai/definition/narrow-AI-weak-AI > Luettu 27.07.2022.

8. Artificial Intelligence (AI). 2020. Verkkoaineisto. IBM Cloud Education. <https://www.ibm.com/cloud/learn/what-is-artificial-intelligence> Luettu 27.07.2022.

9. What Is Machine Learning (ML)? 2020. Verkkoaineisto. Berkeley School of Information. < https://ischoolonline.berkeley.edu/blog/what-is-machine-learning/ > 27.07.2022.

10. Neural Networks. 2020. Verkkoaineisto. IBM Cloud Education. < https://www.ibm.com/cloud/learn/neural-networks > Luettu 27.07.2022.

11. Machine Learning. 2020. Verkkoaineisto. IBM Cloud Education. < https://www.ibm.com/cloud/learn/machine-learning > Luettu 27.07.2022.

12. Supervised vs. Unsupervised Learning: What's the Difference? 2021. Verkkoartikkeli. IBM. Julianna Delua. < https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning > Luettu 27.07.2022.

13. Chatbot Tutorial. 2022.Verkkoaineisto. PyTorch. Matthew Inkawhich <https://pytorch.org/tutorials/beginner/chatbot_tutorial.html?highlight=chatbot%20tutorial > Luettu 25.10.2022.

14. Google Open Source. TensorFlow. < https://opensource.google/projects/tensorflow > Luettu 25.10.2022.

15. TensorFlow-kanava. Verkkoaineisto. < https://www.youtube.com/channel/UC0rqucBdTuFTjJiefW5t-IQ > Luettu 25.10.2022.

16. What Is Keras: The Best Introductory Guide To Keras. 2021. Verkkoaineisto. SimpliLearn < https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-keras > Luettu 25.10.2022.

17. About Keras. 2022. Verkkoainesto. Keras. <https://keras.io/about/> Luettu 25.10.2022.

18. Meta spins off PyTorch Foundation to make AI framework vendor neutral. 2022. Verkkoartikkeli. Ars Technica. Benj Edwards < https://arstechnica.com/information-technology/2022/09/meta-spins-off-pytorch-foundation-to-make-ai-framework-vendor-neutral/> Luettu 2.11.2022.

19. PyTorch. 2022. Verkkoaineisto. PyTorch. <https://pytorch.org/features/> Luettu 2.11.2022.

20. Colaboratory Frequently Asked Questions. Verkkoaineisto. Google. < https://research.google.com/colaboratory/faq.html> Luettu 25.10.2022.

21. Colaboratory Instructions for starting a GCE VM on Colab via GCP Marketplace. Verkkoaineisto. Google. <https://research.google.com/colaboratory/marketplace.html?utm_source=faq&utm_medium=link&utm_campaign=why_arent_resources_guaranteed> Luettu 25.10.2022.

22. Chatbot with PyTorch – NLP and Deep Learning. 2020. Verkkoaineisto. Python Engineer. < https://www.youtube.com/watch?v=RpWeNzfSUHw&list=PLqnslRFeH2UrFW4AUgneY37qOAWQpJyg> Luettu 2.11.2022.

23. What is Tokenization in NLP? Here's All You Need To Know. 2022. Verkkoaineisto AnalyticsVidhya. Aravindpai Pai. < https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/ > Luettu 27.07.2022.

24. Stemming and lemmatization. 2008. Verkkoaineisto. Cambridge University Press. < https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html > Luettu 27.07.2022.

25. PorterStemmer. 2022. Verkkoaineisto. NLTK < https://www.nltk.org/_modules/nltk/stem/porter.html > Luettu 27.07.2022.

26. A Gentle Introduction to the Bag-of-Words Model. 2019. Machine Learning Mastery. Jason Brownlee. < https://machinelearningmastery.com/gentle-introduction-bag-words-model/ > Luettu 27.07.2022.

27. torch.cuda. 2022. Verkkoaineisto. PyTorch. <https://pytorch.org/docs/stable/cuda.html> Luettu 5.11.2022.

28. torch.Tensor. 2022. Verkkoaineisto. PyTorch < https://pytorch.org/docs/stable/tensors.html> Luettu 5.11.2022.

29. Difference Between a Batch and an Epoch in a Neural Network. 2022.Verkkoaineisto. Machine Learning Mastery. Jason Brownlee. 2022. < https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/> Luettu 25.10.2022.