Machine learning – Appendix 2

I denne oppgaven var målet å skape en dypere forståelse for maskinlæring og kunstig intelligens gjennom å endre koden til en eksisterende chatbot. Et par i gruppen hadde noe tidligere erfaring med python, men uten innblanding av hverken kunstig intelligens eller maskinlæring. Vi startet med å gjennomgå eksempeloppgaven lokalt i IDLE, men oppleve problemer med å importere "numpy". Etter litt tid med research og prøving uten resultat gikk vi heller over til "colab.research" av Google.

Vi opplevde oppgaveteksten som vag, men tolket at vi skulle endre diverse parameter i koden for å se hvordan modifikasjoner hadde innvirkning på oppførselen til chatbotten. Dette så vi gjennom variablene "loss" og "accuracy".

Vår tolkning av uttrykk og variabler:
Accuracy: hvor god den er til å predikere seg selv til å forstå og treffe blank på hver setning i manuskriptet
Epoker: antallet av iterasjoner den bruker i treningen
Batch size: hvor mye data den ser på samtidig under treningen
Antall lag: antall lag med dotter i modellen (neural network)
Loss: antallet feil - Vi vet ikke hva "feil" er
Inputet: hva vi skriver til AI-en

Modellen som ble gitt er en simulering av det som er bedre kjent som en "AI-infused chatbot". Strukturen og arkitekturen av denne modellen er ukjent, utover den kjennskapen gruppen har til nevrale nettverk og hvordan deep learning fungerer.

Bildene under viser parameterne som vi har tatt fatt i.

```
import numpy as np
import tensorflow.keras

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, LSTM
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.preprocessing.text import Tokenizer
import random

batch_size = 32
max_words = 1000
epochs = 3
```

antall runder for trening

```
#Model
model = Sequential()
model.add(Dense(512, input_shape=(max_words,)))
model.add(Activation('relu'))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_split=0.1)
print("Finished training")
```

antall lag man definerer for modellen

Kartlegging av “accuarcy” og “loss” ble gjort gjennom ni eksekveringer hvor vi endret modellen til å kjøre med 100, 1000 og 10000 lag. I tillegg spesifiserte vi 3 “epochs”, altså iterasjoner, for computeringen.

Lag/Strofer	100	1000	10000
	1.loss = 3.3 *accuracy = 0.11* **2.loss = 3.14** *accuracy = 0.10* **3.loss = 2.9** *accuracy = 0.137*	**1.loss = 3.1** *accuracy = 0.15* **2.loss = 2.69** *accuracy = 0.17* **3.loss = 2.65** *accuracy = 0.156*	**1.loss = 3.11** *accuracy = 0.16* **2.loss = 2.68** *accuracy = 0.172* **3.loss = 2.64** *accuracy = 0.172*
	1.loss = 3.34 *accuracy = 0.134* **2.loss = 3.19** *accuracy = 0.172* **3.loss = 2.99** *accuracy = 0.172*	**1.loss = 3.13** *accuracy = 0.14* **2.loss = 2.69** *accuracy = 0.172* **3.loss = 2.64** *accuracy = 0.172*	**1.loss = 2.84** *accuracy = 0.171* **2.loss = 2.65** *accuracy = 0.163* **3.loss = 2.64** *accuracy = 0.154*
	1.loss = 3.31 *accuracy = 0.11* **2.loss = 3.12** *accuracy = 0.172* **3.loss = 2.91** *accuracy = 0.172*	**1.loss = 3.14** *accuracy = 0.15* **2.loss = 2.69** *accuracy = 0.172* **3.loss = 2.65** *accuracy = 0.164*	**1.loss = 2.82** *accuracy = 0.154* **2.loss = 2.65** *accuracy = 0.175* **3.loss = 2.64** *accuracy = 0.146*

Ut ifra de variablene som ble observert under de forskjellige eksekveringene så vi en klar sammenheng mellom økning av antall av lag, reduksjon av loss (tap) og høyere accuracy (nøyaktighet). Oppførselen til chatbotten var logisk ved at økt kompleksitet, i form av flere lag, førte til høyere nøyaktighet og mindre “tap”. Dette påvirket dog tiden programmet brukte på å kjøre. Altså så vi at flere lag gav et bedre utfall, men gikk på bekostning av tiden.