Tillämpning av förutbildad djupinlärningsmodell för text multiklassificering inom kundtjänst

Max Nordlund

Examensarbete

Informationsteknik

2023

EXAMENSARBETE	
Yrkeshögskolan Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	9183
Författare:	Max Nordlund
Arbetets namn:	Tillämpning av förutbildad djupinlärningsmodell för text multiklassificering inom kundtjänst
Handledare (Arcada):	Andrey Shcherbakov
Uppdragsgivare:	

Sammandrag:
Under de senaste åren har det tagits stora steg inom djupinlärning och används inom flera olika industrier för att automatisera arbeten som tidigare utförts för hand. En av dessa uppgifter är att klassificera ärenden inom IT-servicebranschen. Examensarbetets syfte är att forska möjligheten att finjustera en förutbildad djupinlärningsmodell som klarar av textklassificering på en godkänd nivå för att kunna implementeras i produktionsmiljö. Som data används en datasats av artiklar och deras kategori som är samlad från Suomi24 (2016–2022). Arbetet görs som en konceptvalidering och den slutliga modellen kommer inte att integreras till andra automationssystem. För att uppnå syftet så utförs det kvantitativa experiment för att välja verktyg och modell för att uppnå den bästa prestandan. Med hjälp av dessa experiment väljs det de noggrannaste parametrar, inlärningshastighet och antalet epoker. För att nå den bästa resultaten väljs det en förlustfunktion som använder sig av en uppvärmning och minskar gradvis på inlärning. Under inträningen samlas räknas det ut en F1-Score, intränings- och valideringsförlust för att analysera modellens prestanda. Den slutliga modellen uppnådde en F1-score på 0,899 och en 89,5% total precision. Dessa resultat tyder på att modellen är passlig för att användas för textklassifikation inom kundservice och kan integreras för att uppnå en obruten automation av arbetsförfrågan. För att minska på fel klassificerade ärenden kan det implementeras en tröskel för hur säker modellen är innan ärendet klassificeras. Dessa ärenden behandlas manuellt och kan användas senare för att vidareutveckla modellens prestanda.

Nyckelord:	FinBERT, BERT, djupinlärning, maskininlärning, Textklassifikation
Sidantal:	27
Språk:	Svenska
Datum för godkännande:	

DEGREE THESIS	
Arcada University of Applied Sciences	
Degree Programme:	Information Technology
Identification number:	9183
Author:	Max Nordlund
Title:	Application of pre-trained deep-learning model for text multiclassification in customer service
Supervisor (Arcada):	Andrey Shcherbakov
Commissioned by:	

Abstract:
In recent years, there have been significant advances in deep learning, and it is being used in several industries to automate tasks previously performed manually. One of these tasks is classifying ticket requests in the IT service industry. The purpose of this thesis is to research the possibility of fine-tuning a pre-trained deep-learning model that can classify text on an acceptable level for implementation in a production environment. A dataset of articles and their categories collected from Suomi24 (2016-2022) is used as data. The work is done as proof of concept and the final model will not be integrated into other automation systems. To achieve the goal, quantitative experiments are performed to select tools and a model to achieve the best performance. The most accurate parameters and the number of epochs are selected using these experiments. To achieve the best results, a loss function is chosen that uses a warm-up and gradually decreases the learning rate. During the training, an F1-Score is calculated and collected along with training and validation loss to analyze the model's performance. The final model achieved an F1-Score of 0.899 and an 89.5% overall precision. These results suggest that the model is suitable for text classification in customer service and can be integrated to achieve end-to-end automation of ticket requests. To reduce misclassified cases, a threshold can be implemented to determine how confident the model must be before classifying the request. These requests are handled manually and can be used later to further improve the model’s performance.

Keywords:	FinBERT, BERT, Deep learning, Machine learning, Text classification
Number of pages:	27
Language:	Swedish
Date of acceptance:	

OPINNÄYTE	
Arcada	
Koulutusohjelma:	Informaatiotekniikka
Tunnistenumero:	9183
Tekijä:	Max Nordlund
Työn nimi:	Esikoulutettun syväoppimismallin soveltaminen tekstin moniluokitukseen asiakaspalvelussa
Työn ohjaaja (Arcada):	
Toimeksiantaja:	

Tiivistelmä:
Viime vuosien syväoppimisessa on otettu suuria edistysaskeleita, ja sitä käytetään useilla eri toimialoilla automatisoimaan tehtäviä, jotka aiemmin suoritettiin manuaalisesti. Yksi tällainen tehtävä on IT-palvelualan työpyyntöjen luokittelu. Tämän opinnäytetyön tarkoituksena on tutkia mahdollisuutta soveltaa esikoulutettua syväoppimismallia, joka kykenee suorittamaan tekstinluokittelun hyväksyttävällä tasolla, jotta sitä voidaan käyttää tuotantoympäristössä. Aineistona käytetään artikkeleita ja niiden kategorioita, jotka ovat kerätty Suomi24:sta (2016–2022). Työ toteutetaan konseptitodisteena, eikä lopullista mallia integroida muihin automaatiojärjestelmiin. Tavoitteen saavuttamiseksi toteutetaan kvantitatiivisia kokeita, selvittääkseen työkalut ja malli, jotka saavuttavat parhaan suorituskyvyn. Näiden kokeiden avulla valitaan tarkimmat parametrit ja epookkien määrä. Parhaan tuloksen saavuttamiseksi valitaan häviöfunktio, joka käyttää lämmitystä aluksi ja vähentää oppimisnopeutta vähitellen. Oppimisen aikana lasketaan F1-pistemäärä, koulutus- ja validointihäviö, jotta voidaan analysoida mallin suorituskykyä. Lopullinen malli saavutti 0,899 F1-pistemäärän ja 89,5 % kokonaistarkkuuden. Nämä tulokset osoittavat, että malli sopii hyvin käytettäväksi tekstiluokittelussa asiakaspalvelussa ja voidaan integroida saavuttaakseen päästä päähän automaatio työpyynnön käsittelyssä. Virheellisten tapausten vähentämiseksi voidaan ottaa käyttöön kynnys, joka määrittää kuinka varma mallin on oltava ennen kuin tapaus luokitellaan. Nämä tapaukset käsitellään manuaalisesti ja niitä voidaan käyttää myöhemmin mallin suorituskyvyn parantamiseen.

Avainsanat:	FinBERT, BERT, Syväoppiminen, Koneoppiminen, Tekstiluokittelu
Sivumäärä:	27
Kieli:	Ruotsi
Hyväksymispäivämäärä:	

INNEHÅLL

FIGURER

TABELLER

FÖRKORTNINGAR

Application Programming Interface (API) **-** Applikationsprogrammeringsgränssnitt
Central Processing Unit (CPU) **-** Centralprocessor
Graphics Processing Unit (GPU) **-** Grafikprocessor
IT Service Management (ITSM) **-** IT-tjänsthantering.
Minimal Viable Product **(MVP) -** Minsta livskraftiga produkt
Natural Language Processing **(NLP) -** Naturlig språkbehandling
Proof of Concept **(POC) –** Konceptsvalidering

1 INLEDNING

Under de gångna åren har det varit mycket tal om maskininlärning och hur det kommer att påverka på vardagen och arbetslivet, detta har väckt ett intresse inom mig att förstå djupare vad det är som händer bakom abstraktionen och hur det skulle kunna utnyttjas inom det jag arbetar med.

Inom kundtjänst är det vanligt att servicepunktsagenter hanterar ett stort antal kontakter, både via e-post och samtal, varifrån en arbetsförfrågan eller problemärende skapas i ITSM-systemet för att dokumentera arbetet och lösningen. I dagens arbetsliv har e-posten etablerat en dominerande ställning tack vare sin snabbhet och användarvänlighet, vilket har lett till att majoriteten av arbetsförfrågningar skickas via e-post. Att registrera en biljett via e-post tar i genomsnitt 2–5 minuter. Registreringen av biljetten är en väsentlig del av behandlingen av arbetsförfrågningar, eftersom det ger den initiala informationen med vilken arbetsförfrågningen löses. Vid registreringen av ärendet ska agenten klassificera vilken kategori arbetsförfrågan tillhör. Klassificeringen görs utifrån e-postmeddelandets innehåll och rubrik. Detta påminner mycket om ett allmänt textklassificeringsproblem som har lösts i andra situationer med maskininlärning, till exempel skräppostfiltrering.

Antalet arbetsförfrågningar som skickas med e-post under en arbetsdag kan vara mycket stort, vilket leder till repetitivt arbete, ökar risken för mänskliga fel och tar upp arbetstid som kan användas för uppgifter som inte kan automatiseras.

Maskininlärning möjliggör användning av en modell som kan vidareutvecklas och att tränas med nya data om det finns behov av till exempel nya kategorier. En maskininlärnings modell möjliggör också kontinuerlig utveckling, eftersom modellen kan tränas om samtidigt som den används för klassificering.

1.1 Syfte

Syftet med den praktiska delen är att utveckla en POC av en djupinlärningsmodell. En POC är en demonstration eller test av genomförbarheten av ett koncept eller idé. Inom mjukvaruutveckling är det en prototyp eller MVP som utvecklas för att demonstrera kärnfunktionaliteten i en mjukvaruapplikation. I examensarbetet utförs detta genom att finjustera en modell med data från Suomi24 som innehåller artiklar och deras kategori. Valet

av data motiveras eftersom textens struktur är liknande till epost och det finns tillgång till kategori vilket används för att träna modellen.

Målet med examensarbetet är att skapa en djupinlärningsmodell som är når tillräcklig prestanda för att tillämpas i produktionsmiljö inom kundservice.

1.2 Metoder

En stor del av arbetet går ut på forskande om olika maskin- och djupinlärningsmodeller, hur de anpassar sig för textklassificering och hur lätta de är att implementera. Själva implementeringen utförs genom försök och misstag metoden (*trial and error*). Metoden innebär att det testas många olika sätt att implementera tills det hittas en lösning som ger tillgodokännande resultat för att kunna börja finjustera inträningen. Det används färdiga funktioner och moduler för att underlätta implementationen och för att nå så högt resultat som möjligt. Detta möjliggör också att man slipper och pröva många olika lösningar snabbare under utvecklingen.

1.3 Avgränsningar

Den finjusterade modellen kommer inte att integreras för att kunna hantera skickade e-poster. Allting kommer att köras manuellt i Python filer och Jupyter notebooks. De själva klasserna som används inom kundservicen nämns inte heller. De matematiska algoritmer inom djupinlärning och neuronnätet kommer inte beskrivas, utom om val av verktyg som är gjort på basis av algoritmen som används i bakgrunden. Arbetet kommer inte att gå djupare in på hur modellens prestanda kan förbättras.

1.4 Struktur

Strukturen för resten av slutarbetet är följande; Det andra kapitlet ger en grundlig teori bakom artificiell intelligens, djupinlärning, neurala nätverk och BERT-modellen. Tredje kapitlet presenterar data, dess struktur och vilka verktyg som använts för implementeringen. Det fjärde kapitlet beskriver implementeringen och hur modellen utvärderas. Det femte kapitlet beskriver resultatet av experimentet och olika testers resultat analyseras i

detalj. Det sjätte kapitlet ger en sammanfattande diskussion och slutsatser som baserar sig på resultaten.

2 TEORETISK BAKGRUND

2.1 Vad är NLP?

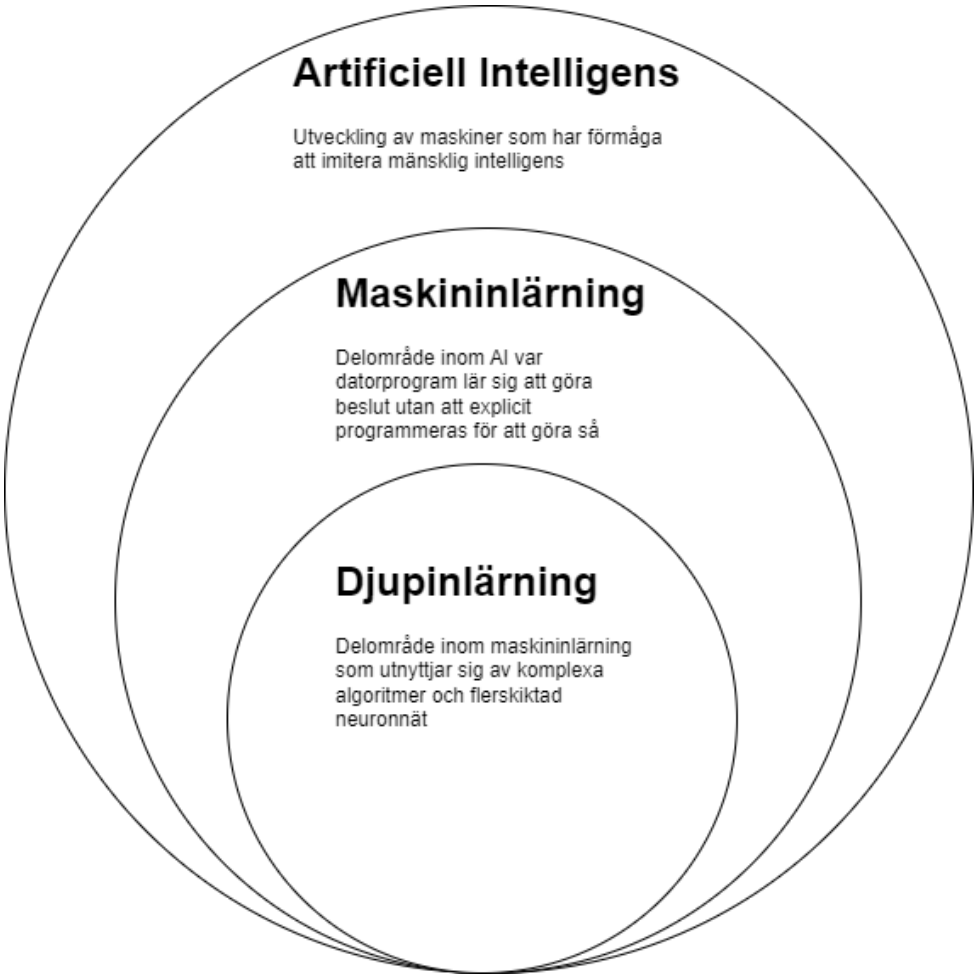
NLP är ett område som tillhör datavetenskap, artificiell intelligens och lingvistik. Syftet med NLP är att skapa interaktion mellan människors språk och datorer genom att möjliggöra datorer att förstå, tolka och generera mänskligt språk. NLP innefattar utveckling av algoritmer, beräkningsmodeller och verktyg som kan analysera, bearbeta och manipulera mänskliga språkdata i olika former, såsom text, tal och handstil. Några av huvudsakliga applikationer för NLP är maskinöversättning, sentimentanalys, textsammanfattning, taligenkänning, namngiven enhetsigenkänning och textklassificering.

2.2 Maskininlärning

Maskininlärning är ett delområde inom artificiell intelligens, vilket generellt definieras som att en maskin imiterar intelligent mänskligt beteende. Maskininlärning innebär att ett datorprogram och algoritmer används för att analysera data och utveckla mönster som kan användas för att göra förutsägelser eller fatta beslut utan att det explicit programmeras att göra så. Maskininlärning utnyttjar algoritmer och statistiska modeller för att göra det möjligt för datorer att ”lära sig” från data, vilket möjliggör att de kan göra beslut som baserar sig på tidigare erfarenheter.

2.3 Djupinlärning

Djupinlärning är en form av maskininlärning som är inspirerad av strukturen i den mänskliga hjärnan, där varje neuron är kopplad till flera andra neuron i flera lager. Detta nätverk kan sedan användas för att lära sig att känna igen mönster i data genom att justera parametrarna i nätverket baserat på felaktiga svar och förbättra prestandan övertiden.



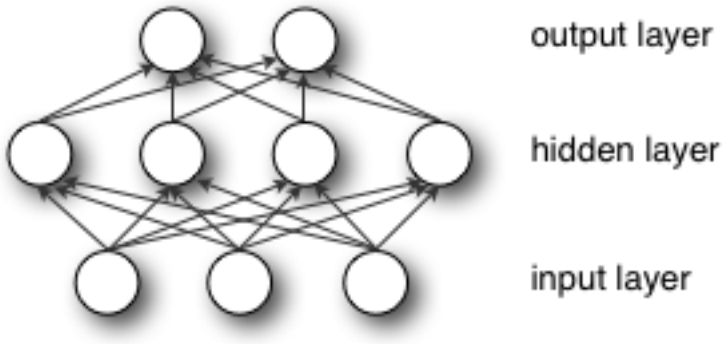
Figur 1 Skillnaden mellan artificiell intelligens, maskinginlärning och djupinlärning

2.4 Artificiella Neuronnät

Neurala Nätverk är en undergrupp inom maskininlärning och utgör kärnan i djupinlärningsalgoritmer. Dessa nätverk imiterar den mänskliga hjärnan och det sättet som biologiska neuroner signalerar till varandra. Artificiella neurala nätverk består av flera nodlager, inklusive ingångslager, dolda lager och responslager. Varje nod i nätverket ansluter till en annan och har en vikt och tröskel som är associerade med den. Om utgången från en nod överstiger tröskelvärdet, aktiveras noden och skickar data vidare till nästa nod i nätverket.

För att lära sig och förbättra sin noggrannhet över tid kräver neurala nätverket tränings-data. När inlärningsalgoritmerna är finslipade för hög noggrannhet, är de kraftfulla verktyg inom datavetenskap och artificiell intelligens som kan klassificera och klustra data på

hög hastighet. Till exempel kan uppgifter som rör taligenkänning och bildigenkänning utföras på betydligt kortare tid än vad det skulle ta för mänskliga experter att identifiera dem manuellt.



Figur 2 Artificiellt neuronnätets arkitektur (Pathmind)

2.5 Bidirectional Encoder Representations from Transformers

Bidirectional Encoder Representations from Transformers (BERT) är en djupinlärnings modell som är baserad på öppen källkod. Modellen baserar sig på transformer-arkitekturen och är utvecklad att förstå och lösa problem inom naturligt språk.

Modellen består av flera lager med transformerblock som arbetar tillsammans för att skapa en representationsvektor för varje ingående textsekvens. Dessa representationsvektorer är konstruerade för att bäst representera betydelsen av varje ord i sammanhanget av hela sekvensen, vilket innebär att modellen är kapabel att ta hänsyn till kontextuell information när den bearbetar text.

BERT -modellen tränas genom en process som kallas förutbildning där modellen utsätts för mängder oannoterad textdata. Under förutbildningen lär sig modellen att förutsäga ord som är maskerade från textsekvensen eller att försöka skilja på två meningar som antingen är relaterade eller inte relaterade till varandra. Denna förutbildning ger modellen en grundläggande förståelse för språkets struktur och samband. Efter förutbildningen kan modellen finjusteras för en specifik uppgift genom att lägga till ett sista lager av neuroner som är utformat att för att lösa den uppgiften. Till exempel kan modellen finjusteras för uppgifter som textklassificering, maskinöversättning eller frågesvar.

2.5.1 FinBERT

FinBERT är en version av Googles BERT som är utvecklad av Turun Yliopisto. Modellen bygger på idén om överföringsinlärning, där en förutbildad modell anpassas för ett språk genom att träna den ytterligare med relevant data. Genom att träna BERT på finska texter kan modellen fånga språkliga särdrag och nyanser specifika för finskan. Detta förbättrar FinBERT:s förmåga att utföra olika uppgifter inom finsk naturlig språkbehandling, såsom sentimentanalys, textklassificering, namngiven och entitetsigenkänning. FinBERT erbjuder en effektiv lösning för att analysera och bearbeta finskspråkig text, vilket kan vara användbart för företag och organisationer som vill automatisera och förbättra deras informationshantering och analyser inom finska domäner (Virtanen, Kanerva, Ilo, Luoma, Luotolahti, Salaskoski, Ginter & Pyysalo 2019).

2.6 Finjustering av FinBERT

2.6.1 Förarbetning av data

Första steget för att finjustera modellen är att städa upp data, i processen ingår det att byta ut förkortningar till deras riktiga mening, rensa bort alla specialtecken, länkar och siffror från texten. Efter städning så ges klasserna nya namn enligt deras index och datan delas till utbildning-, validerings- och testuppsättningar. Varje uppsättning ordsegmenteras vilket innebär att textsekvenserna delas till mindre delar. I BERT modellen används WordPiece algoritmen för att skapa ordsegment. Ordsegmenterings processen lägger till också special ordsegmenter till exempel [CLS] och [SEP] till sekvensen. Efter ordsegmenteringen så transformeras textsekvenserna till numeriska representationer som modellen kan tolka. Detta görs oftast i ett inbäddningslager var varje token mappas till en motsvarande vektor med fixad längd. Modellen kräver att all input är av samma längd så inputsekvenserna måste vadderas eller trunkeras till en fixad längd. Vadderingen innebär att tillägga nollor till slutet av sekvensen för att göra sekvensen lika lång som längre sekvenser, medan trunkering innebär att förkorta sekvensen för att vara samma längd som kortare sekvenser.

2.6.2 Finjustering av djupinlärningsmodellen

För att finjustera modellen så uppdateras vikterna för den förutbildade FinBERT modellen för att anpassa sig bättre till givna uppgiften Detta uppnås med att backpropagera felen genom modellen och uppdatera vikterna med hjälp av gradientnedstigning.

Modellen tränas på finjusteringsdataset med ett liknande tillvägagångssätt som förträning, men med en mindre inlärningshastighet (*learning rate*) för att undvika överanpassning. Under träning uppdateras modellens vikter baserat på felen mellan förutsagda utdata och de sanna etiketterna.

Finjusteringsprocessen är iterativ, och modellens hyperparametrar (till exempel inlärningshastighet, statsstorlek, antal epoker) justeras baserat på prestandan på valideringsuppsättningen. När modellen väl har uppnått tillfredsställande prestanda kan den distribueras för användning i den specifika nedströmsuppgiften.

Efter att datan har förarbetats så hämtas optimeringsalgoritmen. Efter att de har konfigurerats så laddas den förutbildade modellen och den sista lagern som används för att klassificera input textsekvenserna. Eftersom den förutbildade modellen finns färdigt på Hugging Face hub databanken så används Hugging Face biblioteket för att ladda modellen och att sätta upp den sista lagern. Med att ladda modellen från det biblioteket så försäkras det att modellen är kompatibel för finjusteringen.

```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(50105, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=7, bias=True)
)
```

Figur 3 Sammanfattning över BERT modellens arkitektur

2.6.3 Optimering

Som optimeringsmetod används AdamW som är en variant av Adam (*Adaptive Moment Estimation*) optimeringsalgoritmen som korrigerar viktnedgångsproblemet som händer i den ursprungliga Adam-algoritmen. Reglering av viktminskning är en vanlig teknik som används för att hindra överanpassning genom att lägga till en straffterm till förlustfunktionen som avskräcker stora vikter. Den ursprungliga Adam-optimeraren tillämpar

viktminskning på alla parametrar, inklusive de små gradienter, vilket kan resultera i suboptimal prestanda.

AdamW, på andra sidan, tillämpar viktminskning endast på viktparametrarna och inte biasparametrarna. Detta innebär att viktminsknings regulariseringen endast tillämpas på de parametrar som har en större effekt på förlustfunktionen, vilket gör att den kan frikopplas från den adaptiva inlärning hastigheten. (Graetz, 2018)

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda \boldsymbol{\theta}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

Figur 4 Jämförelse mellan Adam och AdamW algoritmerna (Loshchilov & Hutter 2019)

2.7 Utvärdering av modellen

Modellens prestanda utvärderas på valideringsuppsättningen under träningen för att övervaka överanpassning och justera modellens hyperparametrar därefter. Dessutom testas modellen också med en datauppsättning som inte har använts under finjusteringen.

2.7.1 F1-score

Ett sätt att mäta hur väl en modell presterar är att räkna ut *F1-score*. Detta görs genom att kombinera modellens *precision* och *recall* till ett enda värde. Precision mäter hur många positiva förutsägelser som var korrekta och många som var felaktiga. Detta indikerar hur noggrann modellen är när det gäller positiva förutsägelser. Precision räknas genom att dela antalet korrekta förutsägelser (*tp*) med summan av alla förutsägelser som modellen har identifierat som positiva (*tp* + *fp*).

$$Precision = \frac{tp}{tp + fp}$$

Recall mäter hur många positiva fall som modellen identifierade korrekt av alla positiva fall. Detta indikerar hur känslig modellen är när det gäller positiva fall. *Recall* beräknas genom antalet korrekta positiva förutsägelser med totala antalet positiva fall.

$$Recall = \frac{tp}{tp + fn}$$

F1-score beräknas med hjälp av dessa två talen enligt följande: 2 * (*precision* * *recall*) / (*precision* + *recall*). F1-score ger en bra översikt över modellens prestanda och hjälper till att identifiera situationer där modellen kan identifiera positiva fall bra, men också felaktigt identifierar många negativa fall (vilket minskar på precisionen), eller situationer där modellen missar många positiva fall (vilket minskar *recall*).

$$F_1 = \frac{2}{recall^{-1} + \ precision^{-1}} = 2\frac{precision * recall}{precision + recall} = \frac{2tp}{2tp + fp + fn}$$

3 DATA OCH VERKTYG

3.1 Datauppsättningen

Data som används för att lär modellen är artiklar från Suomi24 som är skrapat över årgångarna 2016–2022. Indexet i datasatsen är varje artikel och varje artikel består av 4 kolumner: ”*id*”, ”*topic*”, ”*article*” och ”*class*”. Under förarbetningen kombineras ”*topic*” och ”*article*” till en kolumn eftersom båda ger insikt om vad artikeln handlar om, liknande som med epost. I slutliga implementeringen används den kombinerade rubriken och meddelandet för träning och klassen för att validera klassificeringen.

3.2 Verktyg

3.2.1 Python

Python är ett generellt programmeringsspråk som är utformat för att vara lättläst och lättanvänt. Språket kan användas för att skapa allt från korta skript till komplexa applikationer. Eftersom Python har så brett användningsområde är det ett av det mest populära programmeringsspråket i dagens läge och används i många olika områden, inklusive webbutveckling, datavetenskap och maskininlärning. Python är också känd för sin förmåga att hantera data. Det är enkelt att arbeta med strängar, listor, tupler, och *dictionaries*, vilket gör det till ett utmärkt val för dataanalys och -hantering. Det finns också många ramverk och bibliotek som är utformade speciellt för datavetenskap och maskininlärning, såsom Pandas, Numpy, Tensorflow, och PyTorch.

3.2.2 Tensorflow

Tensorflow är ett ramverk med öppen källkod som används för maskininlärning och artificiell intelligens. Ramverket är utvecklad av Google Brain-teamet och ger en uppsättning av verktyg för att bygga och träna neurala nätverk. Namnet Tensorflow kommer från hur det representerar dataflödet i det neurala nätverket som en serie multidimensionella matriser även kallade för tensorer (Google for Developers, 2016). Tensorflow tillhandahåller ett programmeringsgränssnitt som möjliggör för utvecklare att skapa och manipulera dessa tensorer, samt en uppsättning av förbyggda funktioner som kan användas för att utveckla komplexa maskininlärningsmodeller. En av de viktigaste fördelarna med Tensorflow är dess förmåga att automatiskt distribuera och parallellisera beräkningen över flera CPU: er eller GPU: er, vilket drastiskt påskyndar tränings processen för storskaliga modeller. Tensorflow stöder en mängd olika programmeringsspråk, inklusive Python, C++ och Java, vilket gör det tillgängligt för ett brett spektrum av utvecklare (Tensorflow, 2021).

3.2.3 PyTorch

Pytorch är ett ramverk för maskininlärning med öppen källkod som främst används för att utveckla och träna djupa neurala nätverk. PyTorch är utvecklad av META

forskningsgrupp för artificiell intelligens 2016. PyTorch är baserat på Torch-biblioteket, som ursprungligen utvecklades med Lua (Yegulalp, 2017). PyTorch erbjuder flera funktioner såsom dess dynamiska beräkningsgraf, som möjliggör enklare felsökning och mer flexibel modellering, samt dess automatiska differentieringsförmåga som förenklar processen att beräkna gradienter. Dessa funktioner gör det till ett populärt val bland forskare och utvecklare. PyTorch använder sig också av tensorer för att representera data och dessa kan hanteras antingen med CPU eller GPU. Ramverket erbjuder olika färdigbyggda neurala nätverkslager och moduler, samt en omfattande samling av optimeringsalgoritmer, vilket gör det lättare att utveckla komplexa djupinlärningsmodeller. Dessutom stöder PyTorch implementering till olika plattformar, inklusive mobila enheter och molnet.

3.2.4 Transformers

Transformers är ett bibliotek med öppen källkod utvecklad av Hugging Face. Biblioteket erbjuder ett enkelt och enhetligt API för att använda förutbildade transformatormodeller som BERT, GPT-2 och T5 för ett brett utbud av NLP uppgifter såsom textklassificering, svar på frågor och språkgenerering. Transformatorer är modeller som använder självuppmärksamhetsmekanismer för att fånga långväga beroenden i sekventiella data, såsom text. Transformers bibliotek har uppnått toppmoderna resultat på många NLP-uppgifter de senaste åren (Hugging Face, u.å.). Transformers ger ett gränssnitt på hög nivå för att använda dessa modeller i NLP-applikationer, vilket gör det lättare för forskare och utvecklare att utnyttja kraften av transformatorer utan att behöva implementera komplexa modeller själva. Biblioteket innehåller förutbildade modeller som har finjusterats för NLP-uppgifter och som enkelt kan laddas och användas för specifika uppgifter. Biblioteket stöder många populära programmeringsspråk, inklusive Python, Java och JavaScript, och kan användas i ett brett utbud av applikationer, såsom chatbots, textklassifikation, sentimentanalys och maskinöversättning.

3.2.5 Val av verktyg

Slutliga modellen i examensarbetet är implementerat[1] i Python. För att träna modellen så används det huvudsakligen PyTorch, den förutbildade modellen samt ordsegmenteraren hämtas från Transformers bibliotek. Python valdes eftersom språket har massvis med

bibliotek och ramverk som underlättar utveckling av djupinlärningsmodeller. För att förarbeta datasuppsättningen används det Pandas för att läsa in data och att hantera det. För att göra finjusteringen snabbare så utnyttjas GPU: n med CUDA (*Compute Unified Device Architecture*). CUDA är en arkitektur som möjliggör parallellbearbetning av data med GPU.

Källkoden är tillgänglig på GitHub: https://github.com/nordluma/multiclass-textclassification-finbert

4 BESKRIVNING AV ARBETET

4.1 Förarbetning av datauppsättningen

Data förarbetas först genom att byta de vanligaste förkortningar till sin fulla längd, sedan rensa den från alla specialtecken, länkar och siffror. Efter att data ha rensats så kombineras rubriken och artikeln eftersom båda kommer att användas för att klassificera text. När detta är gjort så raderas alla kolumner som inte används för finjusteringen, det vill säga id-kolumnen. När förarbetningen är genomförd så delas data i två uppsättningar, 95% för inträningen och 5% som inte kommer att användas under inträningen utan endast för att utvärdera modellen efter finjusteringen. På detta sätt undviker vi att modellen har fått en bias av data. Till sist så sparas datauppsättningarna som CSV-filer för att möjliggöra finjustering med antingen en lokal Python skript eller på nätet med Google Colab.

4.2 Finjustering av modellen

För att finjustera en modell för sekvensklassificering, behöver vi först importera alla nödvändiga bibliotek. Sedan definierar vi enheter för konfiguration av träningsmiljö och väljer mellan GPU och CPU beroende på tillgänglighet. Nästa steg är att läsa in datasetet från en CSV-fil som innehåller det förarbetade data. Vi skapar en ordlista med möjliga *labels* för klasserna och ändrar klasserna i datasetet till index baserat *label*-ordlistan. Därefter definierar vi BERT modellen för sekvensklassificering. Vi delar upp datasetet i tränings- och valideringsuppsättningar med en 80/20 uppdelning och ordsegmenterar texten i dessa uppsättningar med hjälp av BertTokenizer som hämtas från Hugging Face. För att hålla data koncis så används det en fixad längd på 512 ordsegmenter om textsekvensen

är längre än detta så trunkeras det och om det är kortare så läggs det till ordsegment i slutet av textsekvensen. De ordsegmenterade kodningarna används sedan för att skapa PyTorch-tensorer för input och *labels* för tränings- och valideringsuppsättningar. Vi skapar Pytorch-dataset för tränings- och valideringsuppsättningarna och transformerar dem till *dataloaders*. För att uppdatera inträningsparametrarna i modellen under finjusteringen definierar vi en optimerare och en inlärningshastighets *scheduler* som gradvis minskar inlärningshastigheten. Vi definierar även funktioner för att beräkna *F1-score* och noggrannhet per klass, vilka används som prestandamått under finjusteringen. Vi definierar en funktion för att utvärdera modellens prestanda på valideringsuppsättningen under finjusteringen och på testuppsättningen efter att modellen har finjusterats. Slutligen definieras finjusterings *loop* som finjusterar modellen med träningsuppsättningen med hjälp av *dataloaders*, optimeraren och *scheduler:* n. Efter varje epok så beräknas modellens prestanda och returnerar valideringsförlusten för att veta när modellen börjar bli övertränad. Dessutom så sparas en version av modellen efter varje epok och en annan version om valideringsförlusterna är lägre än förra epoken. Efter att modellen har finjusterats så sparas data som samlats under finjusteringen för att kunna jämföra modellens prestanda med olika hyperpatametrar.

4.3 Utvärdering

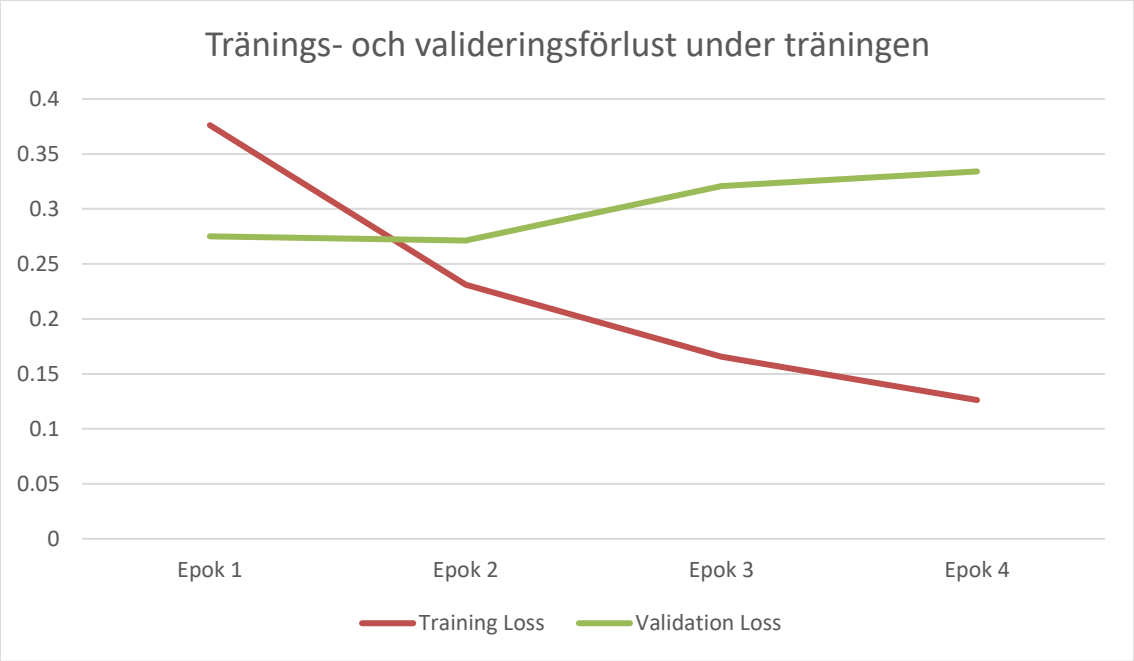
När modellen har finjusterats så hämtas en testuppsättning som används för att utvärdera modellens prestanda med data som inte har presenterats till den tidigare. Detta görs på samma sätt som med inträningsuppsättningen och samma validerings funktion används. Till sist räknas modellens noggrannhet per klass för att granska om det är någon speciell klass som orsakar sämre resultat.

5 RESULTAT

5.1 Inträningen

Under finjusteringen testades det olika mängder epoker för att se hur länge det går att träna modellen innan den blir övertränad. Från figur 6 kan det tydas att modellen börjar bli övertränad efter andra epoken. För att förhindra att modellen som sparas är övertränad

så sparas det en extra modell alltid då valideringsförlusten är den lägsta tills vidare under träningen. I resten av testerna kommer det användas 4 epoker för att träna modellen för att försäkra att modellen sparas om det fortsätter att förbättra dess inlärning. Om modellen blir överträned så används den senaste bästa versionen i stället.



Figur 5 Tränings- och valideringsförlust kurvor

5.2 Tester

För att få reda på bästa resultatet så körs det 5 tester med olika hyperparametrar. Modellen väljs på basis av dessa resultat.

5.2.1 Hyperparametrarna för testerna

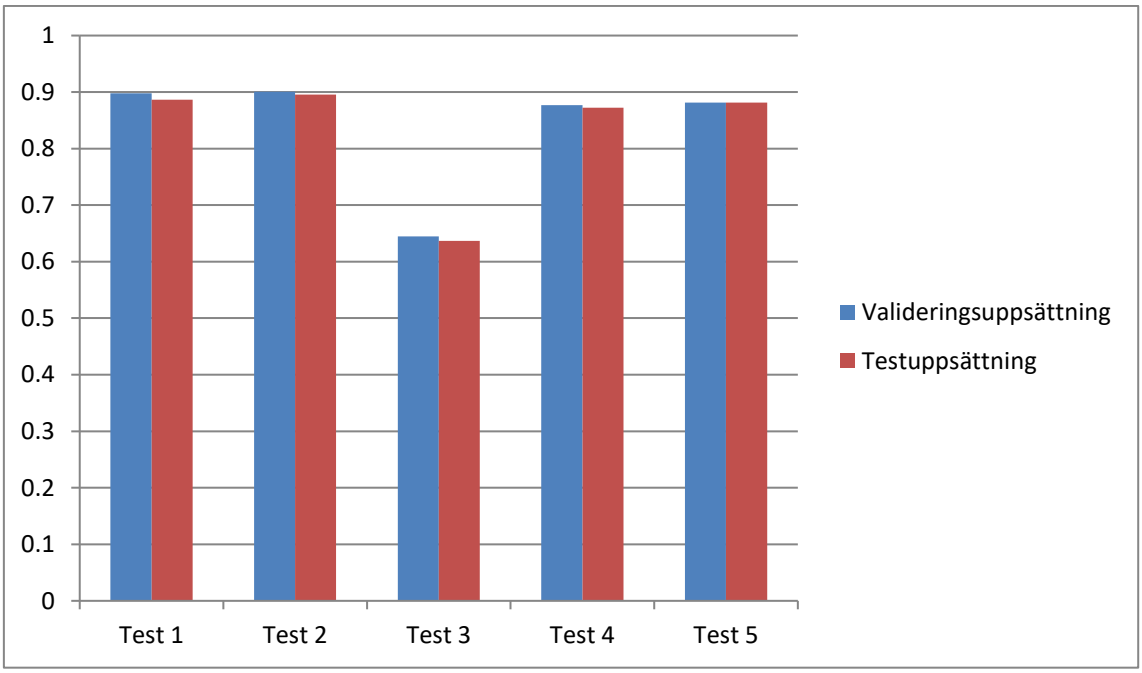
Valet av värden för hyperparametrarna baserar sig på Devlin et al. (2018) rekommendation och eget beslut för att pröva annan inlärningshastighet. Antalet epoker och max längd av tokens ändras inte.

Tabell 1. Hyperparametrarna som används i finjusteringen

Test nr.	**Batch Size**	**Learning Rate**
1	8	1e-5
2	16	1e-5
3	8	1e-4
4	8	5e-5
5	8	3e-5

5.3 Analys

För att utvärdera modellens prestanda så används det *F1-score* och noggrannhet per klass. Av alla körda tester gav test 2 de bästa resultaten så det kommer att analyseras noggrannare.



Figur 6 Totala precisionen testad på validerings- och testuppsättning

Resultaten för Test 2 gav en F1-Score på 0,899 och en noggrannhet på 89,5% vilket tyder på att modellen presterar relativt bra. Modellen är mycket kapabel att korrekt klassificera data till de olika klasserna. När resultaten granskas per klass ser man att klass 7 har en hög träffsäkerhet, där 1266 av 1278 klassificeringar vilket är en mycket hög precision på

99%. Modellen presterar också bra på klass, där det finns 1266 korrekta klassificeringar av totalt 1278 klassificeringar vilket är precision på 98,9%. Resultaten tyder på att dessa klasser innehåller unika egenskaper som modellen identifierar utmärkt.

Å andra sidan visar resultaten att modellen har svårigheter med att klassificera klass 9 och 2 med respektive noggrannheter 55,5% och 77,5% detta kan vara en indikation på att dessa klasser har likheter med andra klass som gör det svårt för modellen att särskilja dem. För att utreda varför dessa klasser har en så märkvärdig skillnad med alla andra klasser så granskades deras innehåll förhand. Båda klassernas innehåll handlar om bilar och olika bilmärken nämns i båda klasserna vilket gör det svårt för modellen att identifiera unika egenskaper för klasserna.

Sammanfattningsvis kan det tydas att modellen är kapabel att effektivt klassificera de olika klasserna med hög precision och träffsäkerhet. Vissa klasser kräver ännu förbättring innan modellen kan implementeras. För att förbättra modellens prestanda måste de klasser som presterade sämre undersökas manuellt för att identifiera vilka egenskaper om är signifikanta för klassificeringen och se om de kan optimeras under förarbetningen av datauppsättningen för att förbättra prestandan modellens prestanda för de mindre presterande klassernas.

Tabell 2. Antalet rätta klassificeringar per klass

Klass	**Korrekta klassificeringar**	**Totalt klassificeringar**	**Procentuell noggrannhet**
3	1092	1104	98,9%
2	640	826	77,5%
19	754	786	95,9%
0	605	684	88,4%
1	825	846	97,5%
9	333	600	55,5%

7	1266	1278	99%

6 SLUTSATSER

Denna studie har bevisat att klassificering med hjälp av djupinlärningsmodell, specifikt FinBERT är ett tillförlitligt alternativ, före det är möjligt att kunna implementera modellen i produktion så måstes den testas med riktiga data från arbetsförfrågan för att jämföra hur dess prestanda skiljer sig med denna modell.

6.1 Diskussion

Förberedningen innan modellen kan finjusteras kan vara svårt eftersom det kräver en bred kunskap över vad de olika ramverken, parametrar och funktioner gör och vilka är rätta för situationen. Examensarbetet gav en grundlig inblick om hur en BERT modell finjusteras för textklassifikation, vilka verktyg, ramverk, parametrar och funktioner kan användas för att skapa en modell som klarar av klassifikationsuppgiften.

Ett antal visualiseringar användes för att förklara den teoretiska bakgrunden, arkitekturen och resultaten. Inom ramarna för detta examensarbete undersöktes det flera ramverk och bibliotek för finjusteringen. Tensorflow användes först för att finjustera modellen men det visade sig vara för komplicerat och krävde att FinBERT modellen måste konverteras till en format som ramverket kan hantera. På grund av detta valdes PyTorch ramverket. Transformers bibliotek valdes eftersom FinBERT modellen stödde biblioteket och hade uppladdats till Hugging Face plattformen. Transformer förelättade finjusteringen märkvärdigt tack vare dess inbyggda funktioner, PyTorch: s stöd för CUDA förelättade också testande eftersom det kunde köras flera tester inom kortare tid.

6.2 Framtida arbeten

Detta examensarbete fokuserade på hur en förutbildad BERT modell finjusteras för att skapa en modell som är kapabel att klassificera text. I framtiden skulle det vara intressant att vidareutveckla på modellen med riktiga data från arbetsförfrågan för att se om det skulle kräva stora förändringar i förarbetningen av data. Dessutom borde antalet klasser

ökas, det skulle vara intressant att se hur dessa förändringar skulle påverka på modellens prestanda. En annan del som inte hanterades i examensarbetet är hur själva modellen skulle integreras till automationssystemen. Detta skulle vara intressant att implementera och bygga upp data pipelines för kontinuerlig finjustering.

KÄLLOR

Brown, S. (21 April 2021) MIT Sloan. *Machine learning, explained*. https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

Devlin, J., Chang, M.W., Lee, K. & Toutanova, K. (2018). *Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding.* https://arxiv.org/abs/1810.04805

Google for Development. (20 Maj 2016) *Machine Learning: Google's Vision – Google I/O 2016*. https://www.youtube.com/watch?v=Rnm83GqgqPE

Graetz, F. M. (3 Juni 2018) *Why AdamW matters. Towards Data Science.* https://towardsdatascience.com/why-adamw-matters-736223f31b5d

Hugging Face. (u.å.) *Transformers.* https://huggingface.co/docs/transformers/index

IBM. (u.å.) *What is a neural network?* https://www.ibm.com/topics/neural-networks

Loshchilov, I. & Hutter F. (2019) *Decoupled Weight Decay Regularization.* https://arxiv.org/abs/1711.05101

Nicholson, C. V. (u.å.) *A Beginner's Guide to Neural Networks and Deep Learning. Pathmind*. https://wiki.pathmind.com/neural-network

TechTarget, (2023) *Natural Language Processing.* https://www.techtarget.com/searchenterpriseai/definition/natural-language-processing-NLP

TensorFlow. (22 April 2021) *API Documentation.* https://www.tensorflow.org/api_docs/

Virtanen, A., Kanerva, J., Ilo, R., Luoma, J., Luotolahti, J., Salaskoski, T., Ginter, F. & Pyysalo, S. (2019) *Multilingual is not enough: BERT for Finnish.* https://arxiv.org/abs/1912.07076

Wood, T. (u.å.) *What is the F-score?* https://deepai.org/machine-learning-glossary-and-terms/f-score

Yegulalp, S. (19 Januari 2017) *Facebook brings GPU-powered machine learning to Python.* https://www.infoworld.com/article/3159120/facebook-brings-gpu-powered-machine-learning-to-python.html