Laboration i Artificiell Intelligens Nivå 1

Partille Gymnasium

Viktor Arohlén

Datum

Introduktion

Denna laboration består av två delar som utförs under två separata tillfällen. Den första delen fokuserar på att förstå hur algoritmer som Word2Vec kan identifiera mönster i skrivet språk. Den andra delen introducerar konceptet prompt engineering och hur man kan förbättra interaktionen med AI-modeller. Slutligen skrivs en avslutande reflektion av hur Word2Vec och prompt engineering kan användas tillsammans för att förstå och förbättra interaktionen med AI-modeller.

Lärarkommentarer i sista avsnittet.

1 Tillfälle 1: Word2Vec och mönster i språk

1.1 Syfte

Att förstå hur Word2Vec-algoritmen kan användas för att identifiera och representera mönster i skrivet språk, samt utforska hur den kan hitta liknande ord baserat på deras betydelse.

1.2 Material

- Dator med Python installerat (eller Chromebook med tillgång till webben)
- Gensim-biblioteket för Word2Vec (för Python-användare)
- En text att analysera (ex. en nyhetsartikel, en låt eller utdrag från en bok)
- Webbaserat alternativ: https://remykarem.github.io/word2vec-demo/

1.3 Instruktioner

1.3.1 För Python-användare

1. Installera Gensim-biblioteket genom att köra följande kommando i terminalen:

```
pip install gensim
```

2. Ladda ner eller skapa en text som du vill analysera. Du kan använda en enkel textfil med några meningar som exempel.

3. Skriv ett Python-skript som använder Word2Vec för att träna en modell på din text. Här är ett exempel:

```
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess

# Ladda textkorpus
with open('din_korpus.txt', 'r', encoding='utf-8') as f:
    text = f.read()

# Förbearbeta texten
sentences = [simple_preprocess(text)]

# Träna Word2Vec-modellen
model = Word2Vec(sentences, vector_size=100, window=5,
   min_count=1, workers=4)

# Spara modellen
model.save("word2vec.model")
```

4. Använd antingen kodexemplet ovan och testa att justera parametrarna, alternativt sätt dig in mer i Gensim genom dess dokumentation

5. Nedan följer en förklaring av de olika parametrarna:

   - **vector_size**:
     - **Vad det är**: Storleken på vektorerna som representerar varje ord.
     - **Hur man kan ändra**: Öka för mer detaljerade representationer (t.ex. 200), minska för snabbare träning (t.ex. 50).
     - **Effekt**: Högre värden ger mer detaljerade ordrepresentationer men kräver mer minne.

- **window**:
  - **Vad det är**: Antalet ord före och efter ett ord som modellen tar hänsyn till.
  - **Hur man kan ändra**: Öka för bredare kontext (t.ex. 10), minska för smalare kontext (t.ex. 2).
  - **Effekt**: Högre värden ger en bredare kontext, men kan göra träningen långsammare.
- **min_count**:
  - **Vad det är**: Det minsta antalet gånger ett ord måste förekomma för att inkluderas i modellen.
  - **Hur man kan ändra**: Öka för att utesluta sällsynta ord (t.ex. 5), minska för att inkludera alla ord (t.ex. 1).
  - **Effekt**: Högre värden förbättrar modellens kvalitet genom att fokusera på vanliga ord.
- **workers**:
  - **Vad det är**: Antalet trådar som används för att träna modellen.
  - **Hur man kan ändra**: Öka för snabbare träning på flerkärniga datorer (t.ex. 8), minska för att spara resurser (t.ex. 1).
  - **Effekt**: Högre värden snabbar upp träningen om datorn har tillräckligt med kärnor.

6. Analysera modellen genom att utforska ordvektorer och hitta liknande ord. Till exempel:

```
# Ladda modellen
model = Word2Vec.load("word2vec.model")

# Hitta ord som är lika 'kung'
similar_words = model.wv.most_similar("kung")
print(similar_words)
```

   Modellen kommer presentera värden mellan 0 till 1 för att rangordna hur *lika* orden är.

7. **Uppgift:** Testa att hitta liknande ord för minst tre olika ord (t.ex. 'bil', 'skola', 'musik'). Dokumentera resultaten och fundera över hur Word2Vec bestämmer vilka ord som är *lika*.

1.3.2 För Chromebook-användare (Webbbaserat alternativ)

1. Öppna webbläsaren och gå till: https://remykarem.github.io/word2vec-demo/.
2. Kopiera in din text eller börja att experimentera med den förvalda demotexten (Hey there, Delilah!).
3. **Vilken modell ska ni använda?**
   I det webbaserade Word2Vec-verktyget finns två alternativ: **CBOW** (Continuous Bag of Words) och **Skip-gram**. Här är en förklaring av skillnaderna och vilken modell som rekommenderas:

   - **CBOW**:
     - Förutsäger ett målord baserat på dess kontextord.
     - Snabbare att träna men kanske inte lika bra för sällsynta ord.
   - **Skip-gram**:
     - Förutsäger kontextord baserat på ett målord.
     - Bättre för sällsynta ord och ger ofta bättre resultat för större datamängder.

   Rekommendation: Eftersom Python-biblioteket Gensim använder Skip-gram som standard, bör du välja **Skip-gram** i det webbaserade verktyget för att få resultat som är mest lika Python-modellen. Experimenta gärna med båda modellerna om du har tid.
4. **Inställningar för att träna modellen**

   För att få resultat som är så lika som möjligt med Python-skriptet (Gensim), rekommenderas följande inställningar i det webbaserade verktyget:

   - **Window size**: 5 (samma som `window=5` i Gensim).
   - **Embedding size**: 100 (samma som `vector_size=100` i Gensim).
   - **Optimiser**: SGD (Stochastic Gradient Descent).
   - **Learning rate**: 0.03 (liknande Gensims standardvärde `alpha=0.025`).
   - **Epochs**: 10 (samma som `epochs=10` i Gensim).

   Dessa inställningar säkerställer att modellen tränas på ett sätt som är konsekvent med Python-skriptet och ger jämförbara resultat.
5. **Inställning för t-SNE simulering** För att visualisera Word2Vec-vektorer med t-SNE rekommenderas följande inställningar:

   - **Learning rate**: 200 (hur snabbt algoritmen konvergerar mot en lösning)
   - **Perplexity**: 30 (hur nära orden algoritmen arbetar (se window))
   - **Iterations**: 500 (hur många gånger algoritmen körs)

   Dessa inställningar ger en meningsfull visualisering av hur ord är relaterade till varandra i vektorrummet. Experimentera gärna med olika värden för att se hur det påverkar resultatet!
6. **Uppgift:** Undersök med hjälp av de olika inställningarna hur nära ord hamnar varandra. Dokumentera resultaten och fundera över hur Word2Vec bestämmer vilka ord som är *lika*.

1.4 Jämförande av modellerna

För den som använt sig av båda modellerna för Word2vec eller som jämfört med en klasskamrat förklaras här likheterna och skillnaderna i resultatet.

- **Liknande ord**:
  - I Python-skriptet får vi en lista av ord som är mest lika ett visst ord (t.ex. "kung").
  - I t-SNE-visualiseringen kan vi leta upp samma ord och se om de ligger nära varandra i 2D-rummet.
- **Kluster av ord**:
  - I t-SNE-visualiseringen kan vi identifiera kluster av ord som har liknande betydelser.
  - I Python-skriptet kan vi bekräfta detta genom att kontrollera om dessa ord också har höga likhetsscore med varandra.
- **Avvikande ord**:
  - Om ett ord ligger långt ifrån andra ord i t-SNE-visualiseringen, kan vi använda Python-skriptet för att kontrollera om det har låga likhetsscore med andra ord.

Genom att jämföra resultaten på detta sätt kan vi bekräfta att t-SNE-visualiseringen och Python-skriptet ger konsekventa resultat.

2 Tillfälle 2: Prompt Engineering

2.1 Syfte

Att förstå och tillämpa prompt engineering för att förbättra interaktionen med en AI-modell.

2.2 Material

- Tillgång till en AI-modell (t.ex. ChatGPT eller annan språkmodell)
- Resultat från Tillfälle 1 (Word2Vec-modellen)

2.3 Instruktioner

1. Börja med att skapa ett grundläggande prompt för att interagera med AI-modellen. Till exempel:

   ```
   "Beskriv hur Word2Vec fungerar."
   ```

2. Testa ditt prompt och notera svaret från AI-modellen.

3. Förbättra ditt prompt genom att lägga till mer kontext eller specifika instruktioner. Till exempel:

   ```
   "Beskriv hur Word2Vec fungerar och ge ett exempel
      på hur det kan användas för att hitta liknande
      ord i en text."
   ```

4. **Uppgift:** Jämför svaren från AI-modellen med ditt förbättrade prompt. Skriv ner hur svaren förändras och reflektera över hur prompt engineering kan påverka kvaliteten på svaren.

5. **Uppgift:** Skapa ett eget exempel där du använder prompt engineering för att få AI-modellen att förklara ett annat koncept relaterat till AI (t.ex. 'neuronnät' eller 'maskininlärning'). Dokumentera ditt prompt och resultatet.

3 Reflekterande uppgift

Efter att ha utforskat Word2Vec och prompt engineering i denna laboration är det dags att reflektera över vad du har lärt dig och hur det hänger ihop med större frågor om artificiell intelligens. Svara på följande frågor utifrån dina erfarenheter och den dokumentation du har skapat under laborationen.

1. Word2Vec och språkförståelse

- Beskriv hur Word2Vec fungerar för att hitta liknande ord. Använd dina resultat från laborationen som exempel.
- Vilka fördelar och begränsningar ser du med Word2Vec? Tänk på hur väl det fungerade för att hitta liknande ord i din text.
- Jämför hur Word2Vec hittar liknande ord med hur du själv skulle göra det. Är det på samma sätt eller annorlunda?

2. Prompt engineering och interaktion med AI

- Beskriv hur du använde prompt engineering för att förbättra resultatet från en AI-modell. Ge exempel från laborationen.
- Vilka utmaningar stötte du på när du skapade prompts?
- Hur kan prompt engineering påverka hur vi använder AI i framtiden? (Både positiva och negativa konsekvenser)

3. AI och mänsklig intelligens

- Jämför hur Word2Vec och prompt engineering fungerar med hur människor förstår och använder språk. Vad är AI bra på, och vad är människor fortfarande bättre på?
- Finns det situationer där du tycker att AI är överlägset människan? Ge exempel från laborationen eller från dina egna erfarenheter.

Lärarkommentar

Laborationen är en omarbetad version av 3.2 (Foundation models - Laborera) från kursen TIG133. Den andra delen föjer till stort sätt samma struktur, men är något förenklad. Valet gjordes eftersom jag själv upplevde laborationen lärorik och dessutom utforskade fler Word2Vec modeller och fann dem intressanta.

Den största skillnaden hittas i den första delen av laborationen, där eleverna får möjligheten att arbeta med Python-biblioteket Gensim för arbeta med en lokal Word2Vec modell. Instruktionerna här är även tänkta at vara så pass detaljerade att elever även utan programmeringsvana, men viss datorvana enkelt kan kopiera instruktioner och skript.

Att inkludera två olika Word2Vec modeller ger även möjlihet att jämföra och utveckla en mer fördjupad kunskap om hur de fungerar.

Laborationen är tänkt att uföras över 2-3 tillfällen.

- **Tidsåtgång**:
  - Tillfälle 1 (Word2Vec): Cirka 60–90 minuter.
  - Tillfälle 2 (Prompt Engineering): Cirka 60 minuter.
  - Tillfälle 3 (Reflekterande uppgift); Cirka 60 minuter eller eventuell hemuppgift.
- **Centralt innehåll**: De praktiska laborationerna tar delvis upp följande centrala innehåll:
  - Vikten av data, datakvalitet för AI och val av data.
  - Översikt av tekniker för AI, däribland sökning, klassificering och objektigenkänning.
  - Enklare typ av problemlösning med hjälp av AI, till exempel klassificering, objektigenkänning, prediktion, tolkning och bearbetning av naturligt språk (NLP), enklare maskininlärning och användning av spelagent.
  - Metoder för enklare träning av AI.
- **Reflekterande uppgift**: uppgiften bedöms främst i form av den reflekterande uppgiften som ska ta vara på dokumentationen från de båda laborationstillfällena. Den reflekterande uppgiften täcker även följande centrala innehåll:
  - Jämförelse mellan hur enklare lösningar med AI fungerar och hur en människa löser samma problem.
  - Några situationer där AI är överlägsen människan och tvärtom.