Uppsala University

Machine Learning

Assignment 7

General information

- The recommended tool in this course is R (with the IDE R-Studio). You can download R **here** and R-Studio **here**. You can use Python and Jupyter Notebooks, although the assignments may use data available only through the R package, a problem you would need to solve yourself.
- Report all results in a single, `*.pdf`-file. *Other formats, such as Word, Rmd, Jupyter Notebook, or similar, will automatically be failed.* Although, you are allowed first to knit your document to Word or HTML and then print the assignment as a PDF from Word or HTML if you find it difficult to get TeX to work.
- The report should be written in English.
- If a question is unclear in the assignment. Write down how you interpret the question and formulate your answer.
- You should submit the report to **Studium**. Deadlines for all assignments are **Sunday 23.59**. See **Studium** for dates. Assignments will be graded within 10 working days from the assignment deadline.
- To pass the assignments, *you should answer all questions not marked with **, and get at least 75% correct.
- To get VG on the assignment, *also the questions marked with a ** should be answered. Sometimes you can choose between different VG assignments, then this is explicitly stated. To get VG you need 75% both on questions to pass and the VG part of the assignment. VG will only be awarded on the first deadline of the assignment.
- A report that does not contain the general information (see the **template**), will be automatically rejected.
- When working with R, we recommend writing the reports using R markdown and the provided **R markdown template**. The template includes the formatting instructions and how to include code and figures.
- Instead of R markdown, you can use other software to make the pdf report, but you should use the same instructions for formatting. These instructions are also available in **the PDF produced from the R markdown template**.
- The course has its own R package `uuml` with data and functionality to simplify coding. To install the package just run the following:

  1. `install.packages("remotes")`
  2. `remotes::install_github("MansMeg/IntroML", subdir = "rpackage")`

- We collect common questions regarding installation and technical problems in a course Frequently Asked Questions (FAQ). This can be found **here**.
- You are not allowed to show your assignments (text or code) to anyone. Only discuss the assignments with your fellow students. The student that show their assignment to anyone else could also be considered to cheat. Similarly, on zoom labs, only screen share when you are in a separate zoom room with teaching assistants.

- The computer labs are for asking all types of questions. Do not hesitate to ask! The purpose of the computer labs are to improve your learning. We will hence focus on more computer labs and less on assignment feedback. *Warning!* There might be bugs in the assignments! Hence, it is important to ask questions early on so you dont waste time of unintentional bugs.
- If you have any suggestions or improvements to the course material, please post in the course chat feedback channel, create an issue **here**, or submit a pull request to the public repository.

Contents

1 Variational Autoencoders

We are now going to implement a variational autoencoder in R using Tensorflow. You can find a lot of the code needed for this assignment in the file `variational_autoencoder.R`.

Note! Running Keras can be computationally heavy, and I suggest not to run the code in `markdown`. Instead, run the code in R and copy the results as output (see the assignment template for an example).

1. (1p) Start by loading the MNIST data in R. See the link above or previous assignments for details.
2. (1p) Now, implement a one-layer (encoder and decoder) feed-forward variational autoencoder with two latent dimensions. Both the encoder layer and the decoder layer should have 200 hidden units. You should end up with a variational autoencoder with roughly 310 000 - 320 000 parameters.
3. Print the model and include it in your report.
   (a) (1p) How many weights (parameters) are used to compute μ and σ^2 for the latent variables?
   (b) (1p) What layer represent the latent variables?
   (c) (1p) What does the lambda layer do in the model?
4. Now train your variational autoencoder on the MNIST data for 50 epochs. Visualize the latent state for the different numbers.
   (a) (1p) How do you interpret this latent state?
   (b) (1p) What numbers are better represented by the latent state?
   (c) (1p) What number is less well represented by the latent state?
5. (2p) Finally, encode all the 2:s in the MNIST test dataset to the latent state using your encoder. What is the mean of the digits "2" in the two latent dimensions? *Hint!* See `y_test` for the MNIST numbers. *Note!* Do not retrain your VAE. Just use the one you have already trained.
6. (1p) Visualize this value of the latent state as a 28 by 28-pixel image using your decoder.

2 Topic Models

We will now analyze the classical book Pride and Prejudice by Jane Austen using a probabilistic topic model. If you have not read the book, **here** you can read up on the story of this classical book.

For this part of the assignment, Griffiths and Steyvers (2004) is the primary reference. I would also recommend reading Blei (2012) before starting with this part of the assignment.

We will use a Gibbs sampler to estimate ten different topics occurring in Pride and Prejudice and study where they occur. A tokenized version of the book and a `data.frame` with stopwords can be loaded as follows:

```
library(uuml)
library(dplyr)
library(tidytext)
data("pride_and_prejudice")
data("stopwords")
```

1. (1p) As a first step, we will remove stopwords (common English words without much semantic information):

```
pap <- pride_and_prejudice
pap <- anti_join(pap, y = stopwords[stopwords$lexicon == "snowball",])


## Joining with `by = join_by(word)`
```

2. (1p) Then we will remove rare words. Here we remove words that occur less than five times.

```
word_freq <- table(pap$word)
rare_words <- data.frame(word = names(word_freq[word_freq <= 5]), stringsAsFactors = FALSE)
pap <- anti_join(pap, y = rare_words)


## Joining with `by = join_by(word)`
```

3. (1p) Now we have a corpus we can used to implement a probabilistic topic model. We do this by using the `topicmodels` R package. As a first step we will compute a document term matrix using the `tm` package, where we treat each paragraph as a document. How many documents and terms (word types) do you have?

```
library(tm)
crp <- aggregate(pap$word, by = list(pap$paragraph), FUN = paste0, collapse = " ")
names(crp) <- c("paragraph", "text")
s <- SimpleCorpus(VectorSource(crp$text))
m <- DocumentTermMatrix(s)
```

4. (1p) To compute a topic model with ten topics, we use a Gibbs sampling algorithm. Below is an example of how we can run a Gibbs sampler for 2000 iterations. Run your topic model for 2000 iterations.

```
library(topicmodels)
K <- 10
# Note: delta is beta in Griffith and Steyvers (2004) notation.
control <- list(keep = 1, delta = 0.1, alpha = 1, iter = 2000)
tm <- LDA(m, k = K, method = "Gibbs", control)
```

5. (1p) In the `uuml` R package you have three convenience functions to extract Θ, Φ and the log-likelihood values at each iteration. This is the parameter notation used in Griffiths and Steyvers (2004).

```
library(uuml)
lls <- extract_log_liks(tm)
theta <- extract_theta(tm)
phi <- extract_phi(tm)
```

6. (2p) As a first step, check that the model has converged by visualizing the log-likelihood over epochs/iterations. Does it seem like the model have converged?

7. (2p) Extract the 20 top words for each topic (i.e. the words with the highest probability in each topic). Choose two topics you find coherent/best (the top words seem to belong together). Interpret these two topics based on the storyline of the book. What have these two topics captured?

8. (2p) Visualize these two topics evolve over the paragraphs in the books by plotting the θ parameters for that topic over time (paragraphs) in the book. Think of this as the time-line of the book. On the y-axis, you should plot θ_i for your chosen topic i and the x-axis should be the paragraph number (first paragraph has number 1 and so forth).

9. (2p) How do these two chosen topics evolve over the course in the book? If you want, you can take a rolling mean of the theta parameters to more easily show the changes in the topic over the book. *Hint!* Here `zoo::rollmean()` might be a good function to use.

3 * Variational Autoencoders using Convolutional Neural Networks

This task is only necessary to get one point toward a *pass with distinction* (VG) grade. Hence, if you do not want to get a *pass with distinction* (VG) point, you can ignore this part of the assignment.

You can choose to either do this task or the task below to get VG on this assignment.

As we have seen previously, for images, we can get better performance using Convolutional Neural Networks. Hence we are going to repeat the exercise above using a convolutional neural network as encoder and decoder. You can find detailed code in the file `variational_autoencoder_deconv.R`.

1. (3p) Now implement a four-layer (encoder and decoder) convolutional neural network with two latent dimensions. There should be 50 filters in each convolutional layer. *Note!* A dense layer should be included as the last step in the encoder and the first step in the decoder. These layers should have 100 hidden units. You should end up with a variational autoencoder with roughly 2M parameters.

2. (1p) Print the model and include it in your report. How many weights (parameters) are used to compute μ and σ^2 for the latent variables? What layer represent the latent variables?

3. (1p) Now train your CNN variational autoencoder on the MNIST data for five epochs. Visualize the latent state for the different numbers. How do you interpret this result? Compare these results with the results from the feed-forward autoencoder.

4. (1p) Finally, encode all the 2:s in the MNIST test dataset to the latent state using your encoder (Hint!, see `y_test` for numbers). What is the mean of the digits "2" in the two latent dimensions?

5. (1p) Visualize the mean value of the digit 2 of the latent state as a 28 by 28-pixel image using your decoder.

4 * Prompt engineering with ChatGPT

You can choose to either do this task or the task above to get VG on this assignment.

This task is only necessary to get one point toward a *pass with distinction* (VG) grade. Hence, if you do not want to get a *pass with distinction* (VG) point, you can ignore this part of the assignment.

For this assignment, you need to have a user account at Open AI to generate an API token. See [this link] for details on how to get an API token to use.

Warning: Do not upload any files containing your API key to public repositories (Github, public google colab, etc.).

You also need to buy credits at Open AI. I think 1-2 dollars should be sufficient for this assignment.

As a second step you need to install the `ropenaiapi` R package. This can be done as follows:

```
library(remotes)
remotes::install_github('MansMeg/ropenaiapi')
```

In the next step we set the API key as an environment variable with:

```
library(ropenaiapi)
set_openai_api_key("[YOUR KEY GOES HERE]")
```

We can now call the API with the `openai_chat` function.

```
x <- openai_chat("Who is Gustav Vasa? Give a short answer.",
                 model = "gpt-3.5-turbo")
x
```

```
## assistant:
## Gustav Vasa was a Swedish king who led a successful rebellion
## against Danish rule and became the founder of modern Sweden in
## the 16th century.
```

See the `ropenaiapi` documentation with "'?openai_chat"' in R for further information on how to interact with the Open AI API.

Note! If you get "'Error: You exceeded your current quota, please check your plan and billing details."' that means that you don't have any credits to use for the calls. You then need to buy some credits from Open AI, 1-2 dollars should be sufficient for this task.

4.1 Make ChatGPT hallucinate

We are now going to understand how and when transformer-based decoder models 'hallucinate', i.e. when the models give factual incorrect information. See (Zhao et al., 2023,

, Section 7.1.2) for an introduction and Huang et al. (2023) for details on hallucinations in LLMs.

Now, create 3 different hallucinations with three different strategies (i.e. its not ok to use the same prompt for three incorrect generations) from the chatGPT models (you can choose which one you like). For each generated hallucination return:

1. The prompt and a the `seed` supplied to the API. You are free to both create hallucinations through a longer conversation and tuning parameters.
2. The response from the LLM.
3. Why the response is factually incorrect and weather it is an intrinsic or extrinsic hallucination.
4. Describe the strategy you use and the analysis why the hallucination happened.

As a final step, when you have succeded to get the LLM to hallucinate, change the model to GPT-4 (include exact which model you use) and run the same prompt that you could get to hallucinate.

1. Return the response from GPT4 for each prompt.
2. Is the response still factually incorrect? If not, see if you can get at least one hallucination for GPT-4. If you don't succeed, just state that.

4.2 In-Context Learning for few-shot classification

In-context learning is the idea to use large language models to complete tasks in a few-shot or zero-shot way. In this task we are going to try to classify political quotes in Swedish to weather they belong to the social democratic party manifesto or the conservative party (Moderaterna) manifesto.

Note! This data is included in `uuml` version 0.4.0 and later.

```
library(uuml)
data("pc_test")
data("pc_train")
```

Below is an example on how you can combine the data to a text string that can be combined to be used in a prompt. Note, you need to develop further on this.

```
example_string <- paste0(pc_train$quote,
                         ": ",
                         pc_train$party, collapse = "\n\n")
cat(example_string)
```

See (Zhao et al., 2023, , Section 8) for details on how to best engineer prompts to solve the classification task.

Now use the techniques (Zhao et al., 2023, , Section 8) to generate as good prompts as possible to classify weather a quote is from the social democratic or the conservative manifesto.

1. Start with zero-shot learning, ie only prompt the LLM without any demonstrations. Test to classify the quotes in the test set.

2. Test how far you can get by using better prompt descriptions to classify the quotes. Try to get responses as classes from the model.

3. Now add demonstrations by adding examples from the training set. Does that improve the accuracy on the testset?

4. Try at least three different strategies from (Zhao et al., 2023, , Section 8) to improve the quality. Discuss your conclusions.

References

Thomas L Griffiths and Mark Steyvers. Finding scientific topics. *Proceedings of the National academy of Sciences*, 101(suppl 1):5228–5235, 2004.

David M Blei. Probabilistic topic models. *Communications of the ACM*, 55(4):77–84, 2012.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models, 2023.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions, 2023.