Uppsala University

Introduction to Machine Learning, Big Data, and AI

Assignment 5

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.
- You should submit the report to **Studium**.
- To pass the assignments, *you should answer all questions not marked with **, although minor errors are ok.
- To get VG on the assignment, *all questions should be answered, including questions marked with a **, although minor errors are ok.
- 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**.
- Deadlines for all assignments are **Sunday 23.59**. See the course page for dates.
- 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 General Questions

You will be able to answer the following questions based on the reading assignments for this assignment. See the course plan for detailed reading **here**.

1. What is the difference between a recurrent neural network and an ordinary feed-forward neural network? (max 1 paragraph)
2. What is LSTM neural network, and why does it work better on long sequences than a naive RNN? (max 1 paragraph)
3. What is the main difference between a GRU and LSTM neural network? (max 1 paragraph)
4. Name at least one difference between the BERT model and the original transformer model? (max 1 paragraph)

2 Recurrent Neural Networks

As a first step, we will try using Recurrent Neural Networks on a text classification task. Here Chapter 6 - 6.2 in Chollet and Allaire (2018) will be the primary reference.

2.1 Text classification using Keras

Start by downloading the IMBD dataset and preprocess it as follows.

```
library(tensorflow)
library(keras)
imdb <- dataset_imdb(num_words = 10000)
c(c(input_train, y_train), c(input_test, y_test)) %<-% imdb
input_train <- pad_sequences(input_train, maxlen = maxlen)
input_test <- pad_sequences(input_test, maxlen = maxlen)
```

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. Create a simple ordinary neural network for text classification in Keras using an embedding layer of dimension 16 and a hidden layer with 32 hidden states (and an output layer with one unit). Use a validation split of 20% observations in the validation set and a batch size of 128. Train your network using rmsprop and use binary cross-entropy as the loss function. Print the model and return your validation accuracy after ten epochs. Most of these settings we will reuse later in the assignments.
2. Explain what the embedding layer is doing.
3. Now setup a simple Recurrent Neural Network with 32 hidden units based on the same embedding layer. Print the model and present your result.

4. The model you have implemented now. Which recurrent network does best describes the model you have now implemented out of Fig. 10.3 and 10.5 in Goodfellow et al. (2016)? Motivate why.

5. Explain the role of the parameters U, W in the recurrent net (see Goodfellow et al., 2017 Section 10.2). How large are these parameters in your model specified above, and where are they included in the Keras output?

6. Now, explain the role of the parameters V (see Goodfellow et al., 2017 Section 10.2). How many parameters are included in V, and where are they included in the Keras output?

7. We are now going to implement an LSTM network. Implement a standard LSTM network with 32 hidden units using Keras. Print your model and your validation accuracy.

8. Extend your network in ways you think fit. Report at least two other extensions/-modifications of your network and why you chose them. Report the Keras model output and validation accuracy.

9. Reason why we cannot get as good performance as with the MNIST dataset. Is the problem bias, variance, or the Bayes error? Why?

3 Transformers and Attention

In this assignment, we will implement a multi-head attention transformer layer. These layers are currently used within all transformer-based models, such as GPT and different BERT models. A good read for this assignment is the Illustrated transformer by Jay Alammar, which you can find [here](http://jalammar.github.io/illustrated-transformer/).

Start by loading the parameters that we are going to use for the implementation as follows.

```
library(uuml)
data("transformer_example")
```

1. Start out by computing the query, key and value matrices for one attention head by implementing it in a function you call `qkv()`. The function should work like this.

```
# Pick out the matrix for the first attention head
Wq <- transformer_example$Wq[,,1]
Wk <- transformer_example$Wk[,,1]
Wv <- transformer_example$Wv[,,1]

# Pick out the first three words and their embeddings
X <- transformer_example$embeddings[1:3,]

qkv(X, Wq, Wk, Wv)
```

```
## $Q
##             [,1]        [,2]       [,3]
## the    0.4722259  0.04995783 -0.5350845
## quick -0.3662435  0.12144160  0.3454785
## brown -0.1029677 -0.12728414  0.1817097
##
## $K
##                [,1]          [,2]       [,3]
## the    0.094360579 -0.203807092 -0.1851229
## quick -0.033313240  0.279012100  0.2530560
## brown -0.004457052  0.001013468  0.0133802
##
## $V
##                [,1]        [,2]        [,3]
## the    0.317318525 -0.35023010  0.13284078
## quick  0.009929565  0.04208206 -0.15412097
## brown -0.316413241  0.27717408  0.02725089
```

2. Now, based on your query, key and value, compute the attention of that given attention head for the three chosen tokens.

```
attention(res$Q, res$K, res$V)
## $Z
##                [,1]           [,2]         [,3]
## the    0.012395453 -0.0212420459  0.009404870
## quick -0.003759269 -0.0008360029 -0.005108890
## brown  0.002412222 -0.0088974612  0.001147999
##
## $attention
##              the     quick     brown
## the   0.3601932 0.3080896 0.3317172
## quick 0.3088780 0.3582373 0.3328847
## brown 0.3300375 0.3360583 0.3339042
```

3. Interpret the attention values. What does the second row mean?

4. Now we have everything in place for implementing it all as a multi-head attention layer. The layer will take in embeddings and then return a 3-dimensional embedding per word. Run your code on all words in the included example.

```
multi_head_self_attention(X,
                          transformer_example$Wq,
                          transformer_example$Wk,
                          transformer_example$Wv,
                          transformer_example$WO)
##                [,1]           [,2]         [,3]
## the   -0.014189613 -0.0040299008 -0.006756286
## quick -0.009963516 -0.0010724342 -0.001996524
## brown -0.006394562 -0.0006626115 -0.002219108
```

4 * Implementing a simple RNN

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.

We are going to implement a one-layer recurrent neural network based on the `rnn_example` data. We are going to implement a simple layer described in Section 10.2.0 in Goodfellow et al. (2016). See this section for details.

We will set the dimensionality of the hidden state to 4 and the output dimension to 3. In this task, we will set h_0, the initial starting state, to a zero-vector. The input will be the embeddings in the `rnn_example` data.

```
library(uuml)
data("rnn_example")
```

1. We start out by implementing a simple RNN linear unit that takes the parameters `W`, `U`, `b`, the previous hidden state `h_t_minus_one` and an input embedding `x_t` and return the activation input a_t. It should work as follows.

```
X <- rnn_example$embeddings[1,,drop=FALSE]
h_t_minus_one <- matrix(0, nrow = hidden_dim, ncol = 1)
a_t <- rnn_unit(h_t_minus_one, X,
                W = rnn_example$W,
                U = rnn_example$U,
                b = rnn_example$b)
a_t
##
##              the
## [1,]  0.5819145
## [2,] -2.2686535
## [3,] -0.6410312
## [4,]  1.4891931
##
```

2. Now implement the `tanh()` activation function.

```
h_t <- activation(a_t)
h_t
##
##              the
## [1,]  0.5240555
## [2,] -0.9788223
## [3,] -0.5656013
## [4,]  0.9031762
##
```

3. As the next step, implement the output function and the softmax function. These functions should work in the following way.

```
yhat_t <- softmax(output_rnn(h_t, rnn_example$V, rnn_example$c))
yhat_t
##
##             the
## [1,] 0.3063613
## [2,] 0.2930885
## [3,] 0.4005502
##
```

4. Now we are ready to implement the full recurrent layer. It should take an input matrix X and the neural network parameters and return the hidden states and the softmax output as follows.

```
X <- rnn_example$embeddings[1:3,,drop=FALSE]
rnn_layer(X,
          W = rnn_example$W,
          V = rnn_example$V,
          U = rnn_example$U,
          rnn_example$b,
          rnn_example$c)
##
## $ht
##               [,1]        [,2]       [,3]       [,4]
## the    0.52405551 -0.97882227 -0.5656013  0.9031762
## quick -0.05951368  0.03988226  0.8241800 -0.6562744
## brown -0.08984008  0.92822217 -0.1563247 -0.6657626
##
## $yhat
##             [,1]      [,2]      [,3]
## the   0.3063613 0.2930885 0.4005502
## quick 0.2838013 0.3490452 0.3671536
## brown 0.2878002 0.3666877 0.3455121
```

5. Now, what is the value of the hidden state `h_t` for the token `dog`?

5 Taking BERT out for a spin (Optional assignment)

Unfortunately, there is no simple way to run BERT directly from R yet. There are two ways you can try to use BERT. There are two ways you can try BERT using Keras.

1. Using reticulate in R. Here is a tutorial.
2. Run BERT using Python in Colab here.

The second approach is probably more straightforward, and there you could try out running BERT using a GPU. In both cases, it is built upon Keras, and you will understand most parts.

The (optional) assignment is to run through the tutorial and summarize your preferred method's experience and thoughts on using BERT.

References

François Chollet and Joseph J Allaire. *Deep Learning with R*. Manning, 2018.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.