EDAN20
Final Examination

Pierre Nugues

October 25, 2022

The examination is worth 200 points: 90 points for the first part and 110 for the second one. The distribution of points is indicated with the questions. You need 50% to have a mark of 4 and 75% to have a 5.
Author and responsible for the examination: Pierre Nugues, telephone number: 0730 433 169

1 Closed Book Part: Questions

In this part, no document is allowed. It is worth 90 points.

Introduction. Cite three applications that use natural language processing to some extent. 3 points

Introduction. Annotate each word of the sentence below with its part of speech:

Most people on antidepressants do n't need them[1]

You will use parts of speech you learned at school: common noun, verb, auxiliary verb, negation adverb, pronoun, determiner, or preposition. 3 points

Introduction. Annotate each group (chunk) in the sentence:

Most people on antidepressants do n't need them

with its type, either verb group (VP), noun group (NP), or prepositional group (PP).

To annotate a group, draw a box around it and mark its type with either VP, NP, or PP.

There are one verb group, three noun groups, and one prepositional group. A prepositional group consists of one single word: The preposition. 4 points

Introduction. Represent the sentence:

Most people on antidepressants do n't need them

with a predicate–argument structure. You should identify one predicate, consisting of a verb, and up to three arguments:

[1]Retrieved on October 21, 2022 from `www.economist.com/leaders`

```
predicate(arg0, arg1, arg2)
```

where the arguments will correspond to a *needeer* (`arg0`), a *thing needed*, (`arg1`), and an adjunct, also called modifier, *AM-X* (`arg2`). As value for *X*, you will choose between locative (LOC), manner (MNR), temporal (TMP), and negation (NEG). 4 points

You will replace `predicate` and `arg...` with their values in the sentence.

Introduction. The sentence:

> Most people on antidepressants do n't need them

includes a coreference. Draw a graph representing it in the sentence and give the value of the antecedent and the anaphor.

Regular expressions. Describe what a concordance is and give all the **case-insensitive** concordances of the string *vibr* with one word before and one word after in the text below, including the title: 3 points

> **En av tre: Jag har skador på grund av vibrationer**
>
> Har du koll på hur mycket dina verktyg vibrerar? De flesta har noll koll, visar en ny undersökning bland 300 svenska hantverkare.
>
> ...
>
> – Jag har varit på byggen där man fortfarande använder sig av luftdrivna mutterdragare som 30-40 år gamla. De är så välbyggda att de aldrig går sönder men de vibrerar som tusan, kommenterar Jakob Riddar, forskningsingenjör vid Arbets- och miljömedicin i Lund, undersökningen.
>
> ...
>
> Två av tre brukar inte följa de säkerhetsföreskrifter som gäller för olika verktyg, till exempel att räkna ut användningspoäng för varje arbetspass.
>
> Undersökningen visar att unga personer som arbetar i mindre företag är mest okunniga. Äldre hantverkare på stora företag med över 200 anställda visade sig vara mest medvetna om säkerheten när de använder maskiner och verktyg.

Source: `www.byggnadsarbetaren.se`, retrieved October 21, 2022. Author Nina Christensen

Regular expressions. Identify what the regular expressions in the list below match in the text above (identify all the matches and just write one word before and after, or write no match if there is no match). The text include the title, but not the ellipses (...): 15 points

List of **case-insensitive** regular expressions:

1. `vibr`
2. `vibr.`
3. `vibr\p{L}+`
4. `\p{L}+erar`

5. `\p{Lu}\p{L}+,`
6. `bygg\p{L}+\s`
7. `\p{N}+`
8. `\p{N}{1,2}`
9. `\p{N}{4,}`
10. `(.)\1\.`

Regular expressions. What will be the output of the command:

```
tr -cs '0-9' '\n' <text
```

when applied to the complete text above (*En av tre: Jag har skador på grund av vibrationer...*). 3 points

Regular expressions. Describe the class of characters of matched by the following Unicode regular expressions:

1. `\P{Lu}`
2. `\p{Ll}`
3. `\p{Greek}`
4. `\P{Greek}`
5. `\p{Z}`
6. `\P{N}`
7. `\p{Cyrillic}`
8. `\P{Han}`

Note the case difference between `\p{}` and `\P{}`. 4 points

Regular expressions. In optical character recognition, some letters are confused with numbers and vice-versa as in: *183I* or *0rigin*. Note the letter *I* at the end of the number (*1831*) and number 0 at the beginning of the word (*Origin*). You will write and OCR postprocessor to detect them in a text. It will consist of a regular expression and a program to print them.

1. In a string mixing letters and numbers, you have at least one sequence of a letter followed by a number or the reverse. Write the regular expression matching such sequences. Use the Unicode regexes; 2 points
2. Before and after your sequence, you have a string of either letters or numbers, possibly none. Write the regular expression matching such sequences. Use the Unicode regexes; 2 points
3. Write the complete regular expression. Do not forget to insert properly the parentheses. 2 points
4. Write the Python code that will:
   (a) Read a file in a string 1 point
   (b) Using `re.finditer()` and a loop, print all the strings your regex matches. 2 points

Machine learning. Given a set of symbols, what is the one-hot encoding of a symbol? 2 points

Machine learning. Supposing we have a set of four symbols $\{a, b, c, d\}$, represent each of them as one-hot encoded vectors. 2 points

Machine learning. Let us suppose we want to create a language classifier. The classifier input will be a text from which it will extract the relative frequency of five characters $\{\alpha, \beta, \gamma, \delta, \epsilon\}$. As output, the classifier will predict the language of this text among three possible ones: Syldavian, Bordurian, and Nuevo Rican.

Represent graphically a fully connected neural network linking the five inputs to the three outputs. There is no hidden layer. Do not write all the weights, but a few of them. 3 points

Machine learning. What is the mathematical object used to store the weights? 1 point

Machine learning. Tell the name of the class to implement this fully connected layer either in Keras or PyTorch. Choose one of them and give the value of their parameters, if needed. 3 points

Machine learning. What is the mathematical operation inside this layer. 1 point

Machine learning. Give the name of the activation function that produces the classification in the end (remember you have three classes). 1 point

Machine learning. Give the name of the loss function you will try to minimize (remember you have three classes). 1 point

Machine learning. Give the name of the algorithm that will enable you to find the optimal weights. 1 point

Machine learning. Give the formula of the logistic function and draw its curve. 2 points

Machine learning. Give the formula of the softmax function for this vector:

$$(1, 2, 3).$$

The softmax value is a vector of dimension 3. Do not compute it. Just write the formula with three fractions. 2 points

Machine learning. Why is the softmax function often used as the last step of a classifier? 1 point

Machine learning. Should we decide to replace one-hot encoded symbols with embeddings, what would be the difference? 2 points

Machine learning. You decide to use the BERT transformer in Fig. 1 to classify the language. The tokens in the figure will correspond to the input characters.

1. For each input token, for instance Tok_1, what is the nature of the output, for instance T_1? 2 points

2. Give a brief description of the content of the blue box in BERT. 2 points
3. How would you carry out the language categorization with BERT? *i.e.* what is the mechanism or device behind the upper red arrow and *Class Label*? 4 points

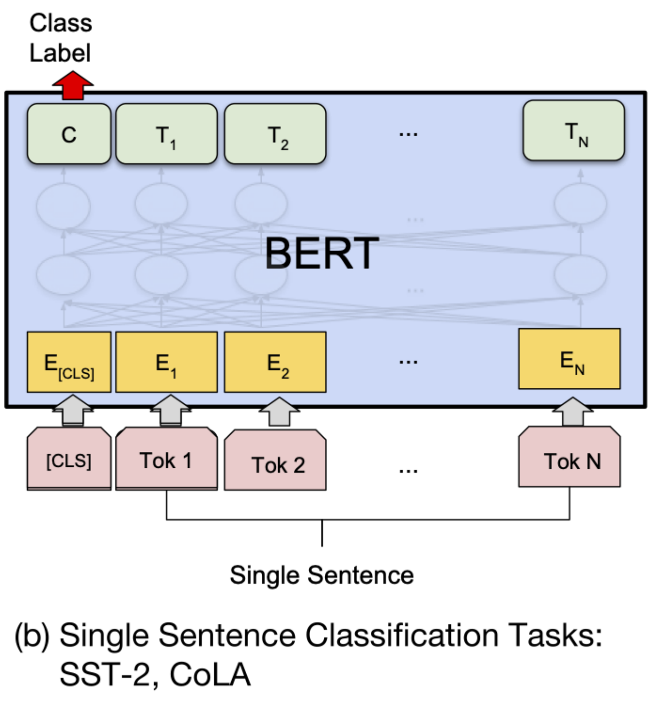

Figure 1: The BERT transformer, after Devlin et al., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019

Language Models. We will evaluate the probability of this sentence:

Jag har skador på grund av vibrationer

with a unigram and bigram language models. You will ignore the start-of-sentence token (`<s>`).

1. Give the probability formula of this sentence using a unigram language model. You will use the notation $P(\text{word})$ to denote the probability of a word; 2 points
2. Do the same with a bigram language model. 2 points
3. Compute these probabilities from the frequencies in Table 1. Use fractions to represent the terms in the product, *i.e.* you will write $\frac{1}{3}$, for instance. The total number of words is N. Do not try to compute the real values. This is not possible as you do not know N. 5 points

Tokenization. What is a byte-pair encoding (BPE). Describe in a few sentences the algorithm to derive a vocabulary from a corpus using BPE. 3 points

Words	Freq.	Bigrams	Freq.
Jag	171,000,000		
har	265,000,000	Jag har	31,100,000
skador	11,900,000	har skador	34,800
på	287,000,000	skador på	2,460,000
grund	41,400,000	på grund	23,700,000
av	289,000,000	grund av	24,500,000
vibrationer	1,010,000	av vibrationer	48,200

Table 1: Unigram and bigram frequencies. Counts retrieved on October 23, 2022 from Google with the prefix `site:se`

2 Programming Problem

In this part, documents are allowed. It is worth 110 points.

In this programming problem, you will implement a part of the transformer described in the paper *Attention is all you need* by Vaswani et al. (2017).

More precisely, you will program the forward pass of the decoder in the inference mode; see Fig. 1 of the paper, page 3, the vertical stack to the right. To make it possible, you will assume other people already trained a model on a large dataset and that this model is ready to use in the form of matrices. You will ignore the positional encoding described in Sect. 3.5 of the paper.

As programming language, you will use Python with the NumPy module. The accent is not on knowing precisely the Python functions. If, at a point of the examination, you know a function exists, but you do not remember it precisely, invent a name and describe the arguments you are using.

2.1 The Application

Your will apply your transformer to translate sentences. You will start from a bilingual corpus of pairs of translated phrases or sentences, such as in English and French:

English	**French**
final exam	*examen final*
proctor	*surveillant*
invigilator	*surveillant*
good luck!	*bonne chance !*

In the examination, to ease understanding, you will use pairs of English and Swedish sentences:

English	**Swedish**
final exam	*sluttenta*
proctor	*tentavakt*
invigilator	*tentavakt*
good luck!	*lycka till!*

Your parallel corpus will be available in the form of two lists of equal length containing the source and target pairs:

```
src_texts = ['final exam', 'proctor', ...]
trgt_texts = ['sluttenta', 'tentavakt', ...]
```

where, for any `i`, `src_texts[i]` and `trgt_texts[i]` form a pair.

2.2 Understanding the Paper

The article, *Attention is all you need*, should be familiar to you as you already read it for an assignment.

1. Summarize the transformer architecture in 5 to 10 lines. 5 points

2. Read Sect. 5.1 of the paper. The authors use a byte-pair encoded (BPE) vocabulary for the English-German dataset, shared between the source and the target languages. Justify the advantages of sharing a vocabulary. 3 points

3. In this examination, you will use characters instead of BPE subwords. What could be the *pros* and *cons* (*pro* and *contra*, arguments *for* and *against*) of using characters in translation. Motivate your answer with a description of the *pros* and then the *cons.* 3 points

2.3 Representing the Input

Given your lists `src_texts` and `trgt_texts`, write the code to:

1. Extract the set of all the characters from the source and target texts. Assign it to the variable `char_set`. 3 points

2. Create the list of the characters in this set and sort it. Assign it to `char_list`. 1.5 point

3. Create an index of the characters. Use a dictionary and call it `idx2char`. Start at index 4. For instance: 1.5 point

   ```
   {4: ' ',  5: '!',  6: '"',  7: '$', ...
   ```

4. Add these symbols: 1.5 point

   ```
   special_idx = {0: '<pad>', 1: '<unk>', 2: '<s>', 3: '</s>'}
   ```

   to your `idx2char` index, where `<pad>` is the padding symbol, `<unk>` is the symbol for an unknown character, `<s>` is the start-of-sentence symbol (here the start of a character sequence), and `</s>` is the end of a sequence. Note that you will not use the `<pad>` and `<unk>` symbols in the examination;

5. Create an inverted index of `idx2char` and call it `char2idx`: 1.5 point

   ```
   {' ': 4, '!': 5, '"': 6, '$': 7, ...
   ```

2.4 From Characters to Embeddings

You will assume you have a matrix **E** of trained embeddings, where each row represents one character. The row index of a character is the same as in `idx2char`. For instance row 5 of **E** contains the embedding of the exclamation mark: `!`.

The dimension of an embedding vector is `d_model` (d_{model}). It is the same as `d_k` (d_k) here as there is only one head.

Given a source and a target strings, you can think of *final exam* and *sluttenta*, you will now create two respective embedding matrices. This corresponds to the rose blocks at the bottom of Fig. 1 of the article.

Calling the two lists of characters `src_char` and `trgt_char`, write the code to create:

1. A list of their indices, `src_idx` and `trgt_idx`. 1.5 point

2. A matrix of their embeddings. You will call these matrices `X_enc` ($\mathbf{X}_{\text{enc}}$) for the source and `X_dec` ($\mathbf{X}_{\text{dec}}$) for the target.

   Note that: 4 points

   (a) For each matrix, you will consider only one character sequence and no batch. The matrix size will then be: length of the sequence $\times d_{\text{model}}$.

   (b) For the *final exam* and *sluttenta* sequences, this would yield:

$$\mathbf{X}_{\text{enc}} = \begin{bmatrix} -\text{embedding}(f)- \\ -\text{embedding}(i)- \\ -\text{embedding}(n)- \\ -\text{embedding}(a)- \\ -\text{embedding}(l)- \\ -\text{embedding}(\sqcup)- \\ -\text{embedding}(e)- \\ -\text{embedding}(x)- \\ -\text{embedding}(a)- \\ -\text{embedding}(m)- \end{bmatrix}; \mathbf{X}_{\text{dec}} = \begin{bmatrix} -\text{embedding}(s)- \\ -\text{embedding}(l)- \\ -\text{embedding}(u)- \\ -\text{embedding}(t)- \\ -\text{embedding}(t)- \\ -\text{embedding}(e)- \\ -\text{embedding}(n)- \\ -\text{embedding}(t)- \\ -\text{embedding}(a)- \end{bmatrix},$$

   where embedding(x) is a vector of dimension d_{model} representing x.

   (c) This part is just a lookup in $\mathbf{E}$ and there is a NumPy indexing shortcut. If you don't know it, write a function to stack the rows of character embeddings with the `np.vstack((X, x))` function, starting from the first row.

2.5 Decoding One Character

To make decoding more concrete, we review here how it predicts one character: We apply the encoder to a list of characters corresponding to the *final exam* source sequence:

```
['f', 'i', 'n', 'a', 'l', ' ', 'e', 'x', 'a', 'm']
```

Ideally, the decoder would output *sluttenta* as target sequence:

```
['s', 'l', 'u', 't', 't', 'e', 'n', 't', 'a']
```

Imagine we are in this ideal setting and at a point where the decoder has already predicted the first three characters:

```
['s', 'l', 'u']
```

Appending the last character (`'u'`) to the end of the previous input, (`['<s>', 's', 'l', 'u']`), what would ideally be the fourth (decoder) output? 1.5 point

2.6 Decoding a Sequence: An Overview

As we said, you will assume that the model has been trained and you will only consider the inference part: Given an input sentence in English, you run the transformer to output its translation in Swedish.

We will start with an overview of whole pipeline, see Fig. 1 in the article, page 3.

2.6.1 Defining a Notation

We will first model the transformer with two macro-functions `encode(source)` and `decode(target)`. The goal of this exercise is to write the top-level loop in `decode(target)`.

Encoder, left part of Fig. 1 in the article. Let us call $\mathbf{X}_{\text{enc}}$ the embedding matrix representing the *final exam* input. Let us call $\mathbf{Y}_{\text{enc}}$ the output of the encoding part of the transformer. We have:

$$\mathbf{Y}_{\text{enc}} = \mathit{encoder}(\mathbf{X}_{\text{enc}}).$$

What is the type of $\mathbf{Y}_{\text{enc}}$ with respect to $\mathbf{X}_{\text{enc}}$ (What kind of mathematical or numerical object it is?) And what is its dimensions? 1.5 point

Decoder, right part in Fig. 1 of the article. Let us call $\mathbf{X}_{\text{dec}}$ the embedding matrix representing the initial input and $\mathbf{Y}_{\text{dec}}$, the initial output

1. What are the types (kind of mathematical or numerical objects) of $\mathbf{X}_{\text{dec}}$ and $\mathbf{Y}_{\text{dec}}$? 1.5 point
2. The decoding function has two main arguments:

   $$\mathbf{Y}_{\text{dec}} = \mathit{decoder}(\mathbf{X}_{\text{dec}}, ?),$$

   where $\mathbf{X}_{\text{dec}}$ is one of them. Give the second one (look at Fig. 1). 1.5 point

2.6.2 Top-level Loop of the Decoder

You will now get a view of the decoding process from the top. This process is very similar to that of the last laboratory in the course.

Let us assume that we have run the encoder on the input representation of the *final exam* source sequence, $\mathbf{Y}_{\text{enc}} = \mathit{encoder}(\mathbf{X}_{\text{enc}})$, and we are starting the decoder.

1. Decoding the target characters is autoregressive. What does this mean? 3 points
2. The decoder needs a first character to start the first iteration. Give its symbol; 1.5 points
3. Given the encoded source, $\mathbf{Y}_{\text{enc}}$, and the first character from the previous question, the decoder predicts one character. Ideally what is this character? (Use the *final exam/sluttenta* pair) 1.5 point
4. For the second iteration, the new input consists of two characters. What are they? 1.5 point
5. What would be the ideal output of the second iteration? 1.5 point
6. When do you stop? Think in terms of predicted symbols. 3 points

2.6.3 Programming the Top-level Loop of the Decoder

In the rest of the examination, you will write your code for any sequence. You will now program in Python pseudo-code the top-level loop of the decoder in the inference mode. 6 points

```
def decode_sequence():
    ...
```

You will assume you have:

- A `decode()` function implementing the process above that, given an input sequence, outputs one character. You will define the appropriate arguments.
- A `vectorize()` function that vectorizes a list of characters into an embedding matrix (corresponding to the rose blocks in Fig. 1 of the article). This is the same as what you did in Sect. 2.4 earlier in this examination.

2.7 The Attention Module

The attention module is the core component of a transformer. It corresponds to the left part of Fig. 2 and to Eq. 1, page 4, in the article. You will now implement it. You will first ignore the mask. You will have one head only, meaning that $n = 1$ in Fig. 2, right part, and there is no concatenation.

2.7.1 The Linear Layer

You will now program the linear layer in Fig. 2, right part. You will assume that you have three already trained matrices: $\mathbf{W}_1^Q$, $\mathbf{W}_1^K$, and $\mathbf{W}_1^V$ corresponding to the three linear blocks in light blue. In this step, on the figure, we have $Q = K = V = \mathbf{X}_{\text{dec}}$.

1. Write the code to implement the linear layer. This is just a matrix product and the multiplication operator is `@` in numpy. You will denote Q_1, K_1, and V_1, the respective outputs of these three linear blocks: 3 points

$$\begin{aligned} Q_1 &= ... \\ K_1 &= ... \\ V_1 &= ... \end{aligned}$$

2.7.2 The Attention Layer

Write now the code to compute the attention layer: 6 points

$$\text{Attention}(Q_1, K_1, V_1)$$

following Eq. 1, page 4 of the article. You can use the builtin softmax function:

```
scipy.special.softmax(x, axis=None)
```

You will specify the axis.

The attention module is followed by a last matrix multiplication by W^O that we ignore in this examination.

2.7.3 Add and Norm: Residual Connection and Normalization

In this section, implement the add and norm block:

Add: The residual connection adds the input and the output of the attention function:

$$X + \text{Attention}(Q_1, K_1, V_1).$$

Norm: The layer normalization computes the mean of each row and, for each element in a row, subtracts the corresponding mean. It also computes the standard deviation of each row and divides each element in the row by this deviation.

Write a `layer_norm()` function that normalizes a matrix: 6 points

```
def layer_norm(Z):
    ...
```

You can either write a loop to compute the mean and standard deviation of each row or use the NumPy builtin functions:

```
numpy.mean(a, axis=None)
numpy.std(a, axis=None)
```

You will specify the axis.

2.7.4 The First Sublayer

Now assemble the code of the first sublayer using the functions you wrote before. Your input will be $\mathbf{X}_{\text{dec}}$, `X_dec`, and you will call the output $\mathbf{X}_{\text{dec\_sublayer1}}$, `X_dec_sublayer1`. 4 points

2.8 Second Sublayer

Write the code to compute the second sublayer, including the layer normalization. Your inputs will be $\mathbf{X}_{\text{dec\_sublayer1}}$, and $\mathbf{X}_{\text{enc}}$. Your will call the output $\mathbf{X}_{\text{dec\_sublayer2}}$, `X_dec_sublayer2`. 8 points

Note that in Fig. 1, page 3, of the article, the output of the first sublayer, represented by the third arrow to the right, corresponds to the Q parameter of the attention function. (This can be misleading given the order of the arguments in the attention function)

2.9 Third Sublayer

Write the code to compute the third sublayer, called feed forward, including the normalization. Your will call the output $\mathbf{X}_{\text{dec\_sublayer3}}$, `X_dec_sublayer3`.

Use the description of Sect 3.3, page 5, of the paper, but ignore the biases, b_1 and b_2. Rename the two already trained matrices in this section: `W1_dec` ($W_{1\text{dec}}$) and `W2_dec` ($W_{2_{\text{dec}}}$).

Note that NumPy has the `np.maximum(0, x)` function, equivalent to relu. 10 points

2.10 Classification Head

Read Sect. 3.4, page 5, of the paper.

1. What is the matrix of the last linear layer? (You have already used it in the beginning of this programming problem); 4 points
2. Apply the last linear layer to $\mathbf{X}_{\text{dec\_sublayer3}}$ using the matrix from the previous item. This is just a matrix multiplication. Call the result $\mathbf{Y}_{\text{dec}}$; 2 points
3. Apply softmax to $\mathbf{Y}_{\text{dec}}$ and extract the maximal arguments of the sequence using the appropriate NumPy function; 4 points
4. Give the predicted character: The last of the sequence that you will convert from an index to a character. 1.5 point

2.11 Masking

Masking in only used in the training step[2]. Justify why the decoder does not need masking in the inference step? 2 points

We will nonetheless implement it to have a complete decoder. See Fig. 2, left, of the article.

1. Create a `mask_scores(Z)` function that replaces all the elements in the upper triangular portion of matrix $\mathbf{Z}$, excluding the diagonal, with a value of -10^9. 6 points

   ```
   def mask_scores(Z):
       ...
   ```

   This function is just a double loop. See Fig. 2, where you replace the gray cells with -10^9 and keep the original values of the red cells. (Red cells may show in black on the printed document).

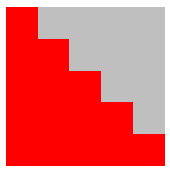

Figure 2: Mask

2. Create a `masked_attention()` function that includes masking. Just need to modify the function in Sect. 2.7.2 of this examination. 4 points

   ```
   def masked_attention(Q, K, V):
       ...
   ```

[2]Note that this is not the case for all kinds of transformers

2.12 A Last Word

If you found this exercise interesting, you can try to implement a complete decoder after the examination. If you decide to do it, simply download a parallel corpus and create a notebook, where you will initialize all the matrices with random values. Then follow the steps of the exam.