EDAN20
Final Examination

Pierre Nugues

October 28, 2024

The examination is worth 200 points: 100 points for the first part and 100 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 100 points.

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

2. **Regular expressions.** Describe what a concordance is and give all the **case-insensitive** concordances of the string *dödsolyck* with one word before and one word after in the text below including the title. In this definition, a concordance does not need to be a word (i.e. a string bounded by white spaces): 3 points

> **Northvolt riskerar mångmiljonböter efter dödsolycka.** På grund av en ny lag kan Northvolt få böter i mångmiljonklassen om företaget hålls ansvarigt för dödsolyckan som kostade en 25-åring livet i Skellefteå förra året.
>
> I december förra året avled en 25-årig man efter en explosionsolycka på Northvolt i Skellefteå.
>
> Hans död hade kunnat undvikas menade Arbetsmiljöverket som i januari i år åtalsanmälde Northvolt. Förra veckan delgav åklagare Christer Jarlås, vid Riksenheten för miljö och arbetsmiljö, bolaget om brottsmisstanke. Förhör kommer att inledas inom kort.
>
> Åklagaren har tidigare sagt till Dagens Arbete att 25-åringen sannolikt inte hade behövt dö om han haft rätt skyddsutrustning.
>
> En bot för en dödsolycka kan kosta företag mellan 1,5 och 5 miljoner kronor. Snittboten för dödsolyckor ligger på 1,9 miljoner kronor. Men sedan år 2020 finns en ny lag som gör det möjligt att höja företagsboten för riktigt stora företag.

Source: Johanna Edström, *Dagens Arbete*, 2024-10-02, `https://da.se/2024/10/northvolt-riskerar-mangmiljonboter/`, retrieved October 6, 2024

3. Regular expressions. Identify what the regular expressions in the list below match in the text above including the title (identify all the matches and just write one word before and after, or write no match if there is no match). Mark a matching white space with the symbol: ␣: 15 points

List of **case-sensitive** regular expressions[1]:

1. `olycka\.`
2. `olycka.`
3. `\p{L}olyckor`
4. `\p{L}+olyckor`
5. `\p{N}+-\p{L}+`
6. `^N\p{L}+`
7. `\s\p{Ll}å\s`
8. `\p{N}{3,}`
9. `\p{Ll}{17}`
10. `(\p{N}+)\1`

4. Encoding. Describe in two to three sentences for each item what is: 8 points

1. The Unicode character properties `\p{...}`
2. The Unicode character properties `\P{...}`
3. The set of characters matched by `\p{P}` or `\p{Punctuation}`
4. The set of characters matched by `\p{Hiragana}`
5. The difference between `\w` and `\p{L}`
6. What would be the visual result of the Python command:

   ```
   >>> '\N{LATIN CAPITAL LETTER A WITH RING ABOVE}'
   ```

7. What would be the visual result of the Python command:

   ```
   >>> '\N{LATIN CAPITAL LETTER A}\N{COMBINING RING ABOVE}'
   ```

8. Would the two output symbols have equal codes and why?

If you are not comfortable with Swedish characters, please replace items 6 and 7 with:

- What would be the visual result of the Python command:

  ```
  >>> '\N{LATIN CAPITAL LETTER E WITH ACUTE}'
  ```

- What would be the visual result of the Python command:

  ```
  >>> '\N{LATIN CAPITAL LETTER E}\N{COMBINING ACUTE ACCENT}'
  ```

5. Regular expressions. Programming with regular expressions:

[1] `Ll` corresponds to lowercase letters

Subword	Logprob	Subword	Logprob	Subword	Logprob
▁	-1.78	g	-3.62	ru	-6.17
r	-3.55	n	-3.68		
d	-3.60	u	-4.26		

Table 1: List of subwords and their logprobs

1. Outline the output of `re.search('abc', 'abcdeabcd')`
2. Give the output of `re.findall('abc', 'abcdeabcd')`
3. Outline approximately the output of

   `list(re.finditer('abc', 'abcdeabcd'))`
4. Outline the difference between `re.findall()` and `re.finditer()`
5. Give the output of

   `re.sub('(abc)(.+)', r'\2 \1', 'abcdef')`
6. Give the output of

   `re.sub('(abc)(.+)', '\2 \1', 'abcdef')`

   Note there is no `r` prefix in the second argument
7. Give the output of `re.split('b', 'abcdeabcd')`.

7 points

6. Tokenization. You will now tokenize the word ▁*grund* into subwords using a unigram model. Before this examination, your instructor trained a model on a novel by Selma Lagerlöf' and Table 1 shows an excerpt of the subwords he obtained with the logarithm of their probabilities (logprob).

1. Given the subwords in Table 1, write the two possible segmentations of ▁*grund*. 2 points
2. Give the probability the two segmentations using products of P(subword), where you will replace the subwords by their values. Give the formulas only. Do not use numerical values and do not compute the numerical values of the products. 1 point
3. Convert these formulas into sums of logprobs and give the two possible segmentations. 1 point
4. Compute manually the sums and give the resulting unigram segmentation. 1 point

7. Language models. You will now rewrite the likelihood of

$$P(\text{Northvolt riskerar mångmiljonböter efter dödsolycka})$$

using:

1. A unigram approximation; 1 point
2. A bigram approximation; 1 point
3. A trigram approximation; 1 point

Table 2: Statistics extracted from google.com on October 8, 2024 with the `site:se` prefix

Unigrams	Freqs.	Bigrams	Freqs.	Trigrams	Freqs.
Northvolt	3,570,000	Northvolt riskerar	7720	Northvolt riskerar mångmiljonböter	4
riskerar	13,200,000	riskerar mångmiljonböter	4410	riskerar mångmiljonböter efter	8
mångmiljonböter	8850	mångmiljonböter efter	2240	mångmiljonböter efter dödsolycka	4
efter	179,000,000	efter dödsolycka	53,800	–	–
dödsolycka	193,000	–	–	–	–

8. Language models. Compute the values of your three approximations with the statistics from Table 2 and where N is the total number of words in the corpus. Just give the fractions. Do not try to reduce them. 4 points

9. Machine learning. Let $\mathbf{u}$ and $\mathbf{v}$ be two vectors, where $\mathbf{u} = (u_1, u_2, ..., u_n)$ and $\mathbf{v} = (v_1, v_2, ..., v_n)$, write the formula of the dot product $\mathbf{u} \cdot \mathbf{v}$ of $\mathbf{u}$ and $\mathbf{v}$. 1 point

$$\mathbf{u} \cdot \mathbf{v} = ...$$

10. Machine learning. Let $\mathbf{u}$ and $\mathbf{v}$ be two vectors, where $\mathbf{u} = (u_1, u_2, ..., u_n)$ and $\mathbf{v} = (v_1, v_2, ..., v_n)$, write the formula of the cosine of $\mathbf{u}$ and $\mathbf{v}$. 2 points

$$\cos(\mathbf{u}, \mathbf{v}) = ...$$

11. Machine learning. PyTorch `Embedding` and `Linear` layers contain tabulated numbers. Your instructor created a 3×3 tensor:

```
values = torch.FloatTensor([[1, 2, 3],
                            [3, 2, 1],
                            [2, 1, 3]])
```

as well as an embedding and a linear layers:

```
E = nn.Embedding(3, 3)
M = nn.Linear(3, 3, bias=False)
```

He filled both `E` and `M` with `values` so that we have:

```
>>> E.state_dict()
OrderedDict([('weight',
              tensor([[1., 2., 3.],
                      [3., 2., 1.],
                      [2., 1., 3.]]))])

>>> M.state_dict()
OrderedDict([('weight',
              tensor([[1., 2., 3.],
                      [3., 2., 1.],
                      [2., 1., 3.]]))])
```

Although apparently identical, these layers have different purposes.

1. Describe the function of an embedding layer and given

   ```
   x = torch.tensor([1, 2, 2])
   ```

   as input, write the output of

   ```
   >>> E(x)
   ```

2. Describe the function of a linear layer and given the vector 4 points

   ```
   x = torch.tensor([1., 2., 2.])
   ```

   as input, compute the output of 4 points

   ```
   >>> M(x)
   ```

Remember that a matrix-vector product is the dot product of the matrix rows by the vector.

12. Machine learning. In this exercise, you will suppose that you have a multilingual corpus, where you want to detect the language of the texts, three in total. You decide to create a classifier with this model:

```
model = nn.Sequential(
    nn.Linear(input_dim, 5),
    nn.ReLU(),
    nn.Linear(5, nbr_classes)
)
```

You will denote $\mathbf{x}$ the input vector representing a text and $\hat{\mathbf{y}}$, the output.

1. Draw a network representing this architecture. You will suppose `input_dim` sufficiently small to be able to draw the figure; 2 points
2. Describe in a two or three sentences the mathematical operation carried out with `nn.Linear`; 2 points
3. Describe in two or three sentences the meaning of the arguments in `Linear(5, nbr_classes)`; 2 points
4. Describe what is a `ReLU()` function; 1 point
5. What is the size of $\hat{\mathbf{y}}$; 1 point
6. What do the coordinates of $\hat{\mathbf{y}}$ represent; 1 point

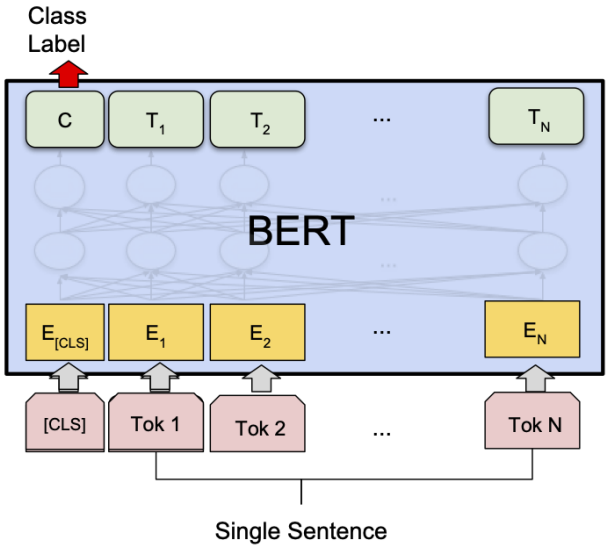
13. Machine learning. Give the values of the softmax function for the $(x_1, x_2, ..., x_n)$ vector: 2 points

$$\text{softmax}(x_1, x_2, x_n) = (..., ..., ...)$$

14. Machine learning. The formula of self-attention is:

$$\text{softmax}\left(\frac{XX^\intercal}{\sqrt{d_{\text{model}}}}\right)X,$$

1. Describe what X represents relatively to the input words; 2 points
2. Describe what $XX^\intercal$ represents relatively to the input words; 2 points
3. What is the purpose of the $\sqrt{d_{\text{model}}}$ division; 1 point



(b) Single Sentence Classification Tasks:
SST-2, CoLA

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

4. Why do we apply a softmax function? 2 points
5. Finally what is the result when we multiply softmax $\left(\frac{XX^\intercal}{\sqrt{d_{\text{model}}}}\right)$ and X. 2 points

15. Machine learning. Figure 1 shows how to use BERT as a classifier.

1. What is the purpose of the blue rectangle? 2 points
2. A BERT encoder consists of a stack of layers. Give the names of the two main components in each layer. 2 points
3. Outline how the weights in the blue rectangle are pre-trained. 3 points
4. Once pretrained, which output do we use to make the prediction in the case of a sentence or text classification? 2 points
5. Which PyTorch class do you need to add on top of the C box to classify an input sentence? 3 points

16. Machine learning. In this last exercise, you will outline the structure of the encoder-decoder architecture you saw in the last lab:

1. Draw a sketch of the architecture; 3 points
2. On this sketch represent an input sentence as well as the output sentence during the training step (the target sentence should show twice on the figure); 3 points
3. Once trained, describe how the system translates a sentence. 3 points

2 Programming Problem

In this part, documents are allowed. It is worth 100 points.

In this second part, you will program a sentence classifier using a encoder transformer stack. As architecture, you will follow the paper *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* by Devlin et al. (2019), where you will skip the pre-training step.

As programming language, you will use Python with the PyTorch module. The accent is not on knowing precisely the Python or PyTorch 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 Paper

Read the paper abstract, the introduction (Sect. 1), and the introduction of Sect. 3 until Sect. 3.1. Summarize them in 10 to 15 lines. 12 points

From now on, you will assume that you have pre-trained parameters and that you can load them in your model. You will fine-tune this model on a classification dataset.

2.2 The Dataset

1. The paper cites datasets for different purposes. What are the sentence classification datasets the authors used? 2 points

2. In this examination, you will use SST-2. Give the score the BERT models obtained on this dataset. For this task, the value is the accuracy. 2 points

The Stanford Sentiment Treebank[2], version 2, (SST-2) uses a set of 11,855 movie reviews as starting corpus. The authors extracted 215,154 unique phrases (subsentences) from these sentences and annotated them with a positive or negative sentiment. Table 3 shows an excerpt of it.

Table 3: Samples from the Stanford Sentiment Treebank (SST-2) dataset. Label 1 is positive, 0 is negative.

Idx	Sentence	Label
3000	buy is an accomplished actress , and this is a big , juicy role	1
3030	does a good job of establishing a time and place , and of telling a fascinating character 's story .	1
3060	every painful nuance , unexpected flashes of dark comedy	1
3090	how it washed out despite all of that	0
3120	a fantastically vital movie	1

[2]Source: `https://aclanthology.org/D13-1170.pdf`

2.3 Preprocessing the Dataset

You will assume the dataset is available in the form of three lists of dictionaries containing the training, validation, and test sets. These lists are called respectively `train_set`, `val_set`, and `test_set`.

As length, we have:

```
>>> len(train_set), len(val_set), len(test_set)
(67349, 872, 1821)
```

In each list, each dictionary contains three keys: `idx`, the index of the (sub)sentence in the dataset, `sentence`, the text of the (sub)sentence, and `label`, the sentiment. For instance:

```
>>>  train_set[3000]
 {'idx': 3000,
 'sentence': 'buy is an accomplished actress , and this is a
 big , juicy role ',
 'label': 1}
```

You will now tokenize the datasets and replace the tokens with indices.

1. Write a statement to tokenize a text, for instance `train_set[3000]['sentence']`, in a dictionary. This is just one statement as the tokens, including the punctuation, are already separated by white spaces. 3 points

2. BERT has a vocabulary size given by the set of subwords it uses. Find its value in the article and call it `NUM_WORDS`. 2 points

3. The sentences in BERT have also a maximal length. Find the values the authors used in the Appendix, in Sect. A.2, final paragraph. Use the lowest value and call it `MAX_LEN`. 2 points

4. We will not use a subword tokenizer in this examination. To limit the vocabulary to `NUM_WORDS`, we will count the words and map the least frequent words to the unknown token `UNK`. Write the code to gather all the words in the training set and count them. You can use the `Counter` class. Call the result `word_freqs`. As an output example, your instructor obtained: 6 points

   ```
   >>> word_freqs
   Counter({'the': 27205,
            ',': 25980,
            'a': 21609,
            'and': 19920,
            ...
   ```

5. Write the code to build a `token2idx` dictionary mapping the words or tokens to an index starting at index 3. You will start with the most frequent words and stop when you have reached `NUM_WORDS`.

   Note that `Counter` can order the words by frequency. Just use `word_freqs.most_common(n)` to output the list of the `n` most common elements. 3 points

6. In your `token2idx` dictionary, assign the special tokens `'[PAD]'`, `'[UNK]'`, and `'[CLS]'` to respectively index 0, 1, and 2. 1 point

7. Using `token2idx`, write the code to build a `idx2token` dictionary mapping each index to a token. 2 points

8. Write a function to tokenize the sentences in `train_set`, `val_set`, and `test_set` and convert them to tensors of indices. You will truncate the size of the tensors to `MAX_LEN`.

   ```
   def sent2idx(dataset, token2idx):
   ...
   ```

   You will store the results in the same dictionaries of your dataset.

   An input dictionary has the keys: `'idx'`, `'sentence'`, and `'label'`. You will just add the `word_idx` key to hold the indices. An output dictionary will then have the keys: `'idx'`, `'sentence'`, `'label'`, and `'word_idx'`.

   Your lists will start with the index of the `'[CLS]'` token and you will map all the unknown words to the `'[UNK]'` index. 6 points

9. Apply this function to `train_set`, `val_set`, and `test_set`. 1 point

   As an output example, your instructor obtained:

   ```
   >>> train_set[3000]
   {'idx': 3000,
    'sentence': 'buy is an accomplished actress , and this is a
                big ,  juicy role ',
    'label': 1,
    'word_idx': tensor([   2, 1835,   11,   17, 1247,  753,    4,
                           6,   22,   11,    5,  173,
                           4, 3239,  529])}
   ```

2.4 Encoder Transformer

We are now ready to program our BERT classifier. We will replicate a simplified version of architecture in the paper and we will first use random embeddings setting aside possible position embeddings. We will first review how to create all the objects we need and then integrate them in a model.

2.4.1 Embeddings

BERT embeds the words as vectors. You will create an embedding layer. The input will be a tensor of indices representing the words of a sentence and the output will be the vectors associated with the words.

1. Read the paper and tell what are the embedding dimensions of the base and large model of BERT? 1 point

2. Here you will use an embedding dimension, `D_MODEL=128`. Create an embedding object to store the words embeddings. You will need the PyTorch class:

```
nn.Embedding(num_embeddings, embedding_dim, padding_idx)
```

PyTorch will initialize the embedding object with random values, which is what we want. Note that number of words maybe less than `NUM_WORDS` for a small corpus. Use the exact number of words in your `idx2token` dictionary to create the object. 3 points

2.4.2 Encoder

We will now create an encoding stack. The input will be the vectors from the embedding layer and the output will be the same amount of autoencoded vectors.

1. In the paper, what are the number of heads and number of layers in the base and large models of BERT? 1 point

2. Create the stack with an embedding dimension of 128, four heads, and two layers. You will use the PyTorch classes to build the stack. You create an encoder layer with `TransformerEncoderLayer` and then you stack the layers with `TransformerEncoder` as with: 3 points

```
encoder_layer = nn.TransformerEncoderLayer(d_model, nhead)
transformer_encoder = nn.TransformerEncoder(encoder_layer, num_layers)
```

2.4.3 Extraction of the Encoded `'[CLS]'`

BERT carries out the classification through the encoded vector of the first token called `'[CLS]'`. See Fig. 1. We will here simplify the experiment and use a batch size of 1.

Given an `X` output representing the encoded vectors of a sentence, extract the first autoencoded vector from `X`. Note that this is just a slice. 3 points

2.4.4 Head

The autoencoded output is passed a classification head, here limited to one layer.

1. Give the name of the class you will use to implement this layer in PyTorch. 2 points

2. This layer takes two arguments: The input dimension and the number of classes. Give their values in your application case. Note that even if the problem is binary, you will give the exact number of output classes. 2 points

2.4.5 Creating the Model

In this section, you will create an `EncoderClassifier` class with the `__init__()` and `forward()` methods. You will insert a ReLU function between the transformer and the head.

```
class EncoderClassifier(nn.Module):
    def __init__(self, vocab_size, embed_dim, max_len,
                 num_heads, num_layers)
        super().__init__()
        ...

     def forward(self, x):
        ...
        return x
```

1. Write the `__init__()` method, where you create an `Embedding` layer, the transformer, the ReLU function, and the classification head. 10 points
2. Write the `forward()` method, where you extract the embeddings of the tensor containing the indices, you apply your encoder stack, you pass it to the dropout payer, extract the `'[CLS]'` vectors, and pass it to the classification head. 10 points
3. Create an `bert` object from the `EncoderClassifier` class. Apply it to a sentence. 1 point

Applying `bert` to one sentence of `train_set`:

```
>>> train_set[3000]['word_idx']
tensor([   2, 1835,   11,   17, 1247,  753,    4,    6,   22,
          11,    5,  173,    4, 3239,  529])
```

results in the output:

```
>>> bert(train_set[3000]['word_idx'])
tensor([0.0470, 0.1248], grad_fn=<ViewBackward0>)
```

2.5 Training the Transformer

You can now train a model with the SST-2 dataset in Table 3.

1. Define a loss and an optimizer; 5 points
2. Write the basic loop to train the model. You will read one sentence at a time from `train_set`. 5 points

2.6 Adding the Position Embeddings

BERT's authors used position embeddings in addition to the token embeddings, see Fig. 2. These position embeddings are just learnable vectors of the same dimension as the token embeddings associated to the positions, for instance to position 0, position 1, etc.

In Fig. 2, you have segments embeddings to encode two possible segments. As you have one sentence only, you will ignore these segment embeddings.

1. Create an embedding matrix to hold the position embeddings. You will replicate the instructions in Sect. 2.4.1 of the examination with the proper size. Remember that the maximal sentence length is `MAX_LEN`. 3 points

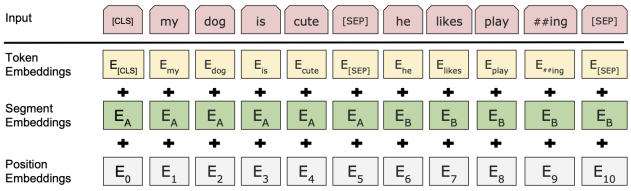


Figure 2: The BERT embeddings, after Devlin et al., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019

2. In the `EncoderClassifier()` class, add one line in the `def __init__()` method to create the position embeddings; 2 points

3. Given a tensor of token indices, `x`, generate the token positions using `torch.arange()` up to the last token. As an example, you have: 2 points

```
>>> torch.arange(5)
tensor([0, 1, 2, 3, 4])
```

4. Modify the `def forward(self, x)` function so that you add the position embeddings to the token embeddings to form the input emebddings as in Fig. 2. 5 points

2.7 A Last Word

If you found this part interesting, you can try to implement the complete model after the examination. If you decide to do it, simply download SST-2 (`https://huggingface.co/datasets/stanfordnlp/sst2`) and create a notebook. Then follow the steps of the exam. There are a few details I set aside, but they are easy to add.