EDAN20
Final Examination

Pierre Nugues

October 24, 2023

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.

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

Introduction. Annotate the words of the sentence below with their part of speech. Do not annotate the word *is* that is added to improve the understanding of the sentence:

First word [is] discovered in unopened Herculaneum scroll[1]

You will use parts of speech you learned at school among these ones: common noun, proper noun, verb, adjective, or preposition. 3 points

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

First word discovered in unopened Herculaneum scroll

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, two noun groups, and one prepositional group. A prepositional group consists of one single word: The preposition. 4 points

[1] Title of a blog post about reading the contents of papyrus scrolls using machine-learning techniques. These scrolls were carbonized by the Vesuvius eruption and were deemed illegible until earlier this month; see Fig. 1, October 12th, 2023. Retrieved October 17th, 2023 from `https://scrollprize.org/firstletters`

Figure 1: Carbonized Vesuvius scrolls. Credit: University of Kentucky

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

> **Staten måste ta ansvar för de arbetsskadade**
>
> Arbetsskadeförsäkringen håller på att tyna bort samtidigt som arbetsskador med dödlig utgång ökar.
>
> Sveriges första statliga arbetsskadeförsäkring infördes redan 1901 och är därmed vår äldsta socialförsäkring. Men de senaste åren har försäkringens betydelse i tysthet urholkats kraftigt och det utan att arbetsskadelagstiftningen ändrats. Antalet beslut som Försäkringskassan fattar om att godkänna en arbetsskada och bevilja livränta har minskat dramatiskt sedan 2005, från cirka 7 300 till 900.

Source: Abridged from Dan Holke, *Svenska Dagbladet*, 2023-09-14, `https://www.svd.se/a/APM6pz/dan-holke-staten-maste-ta-ansvar-for-de-arbetsskadade`, retrieved October 17, 2023

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 does not include the title: 15 points

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

1. `försäkring\s`
2. `försäkring\.`
3. `för\p{L}+kring`
4. `[Ff]örsäkring`
5. `\p{Lu}?örsäkring`
6. `[\p{L}\p{N}]+\.+$`

7. `\p{N}+`
8. `\p{N}{4,}`
9. `A\p{L}{2,12}`
10. `(.)(.)\2\1`

Encoding. Describe in three to four sentences for each item what is: 5 points

1. Unicode
2. The Unicode character properties
3. The advantages of Unicode with respect to other encoding systems
4. UTF-8
5. The normal form composition and decomposition (NFC and NFD)

Regular expressions. Describe the set of characters of matched by the following legacy or Unicode regular expressions:

1. `[A-Za-z]`
2. `[0-9]`
3. `\P{L}`
4. `\p{N}`
5. `\p{Z}`
6. `\P{Arabic}`
7. `\p{Devanagari}`
8. `\p{Whitespace}`

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

Regular expressions. In most text editors, the text you type is wrapped around so that it does not go beyond a certain width. The text is aligned on the left-hand side and is ragged on the right-hand side.

With certain text editors, to limit the width, you can apply a "hard wrap" and break lines by inserting a carriage return, `\n`, after a certain number of characters, for instance 72, as in:

```
Arbetsskadeförsäkringen håller på att tyna bort <- hard wrap
samtidigt som arbetsskador med dödlig utgång  <- hard wrap
ökar.                                          <- new paragraph

Sveriges första statliga arbetsskadeförsäkring <- hard wrap
infördes redan 1901 och är därmed vår äldsta <- hard wrap
socialförsäkring. Men de senaste åren har <- hard wrap
...
```

Note that the hard wraps do not split words.

You will write a regular expression to create these hard wraps with here 72 characters. The idea is to match up to 72 characters followed by a space or up to 72 characters if this sequence contains no space. Then, you will replace the trailing space with a carriage return.

1. Write a regular expression that matches sequences of at least one and at most 72 characters followed by one or more white spaces: `' '`; 3 points
2. Create a group to remember the matched characters of the first part of the sequence, but not the trailing white spaces; 1 point
3. To handle sequences of 72 characters with no space, create a disjunction that either matches your first expression or a sequence of 72 characters with no constraint on what follows. Remember this part too; 2 points
4. Write a Python statement that replaces your regex in a text with a carriage return; 4 points

Tokenization. Using a corpus limited to this text:

har minskat dramatiskt

you will apply the byte-pair encoding algorithm to find:

1. The initial vocabulary of this corpus. Give the list of its symbols; 1 point
2. The first merge pair; 1 point
3. Using the vocabulary and the pair, tokenize the word *skador*. In case a symbol is not in the initial vocabulary, you will fall back to this symbol. 2 points

Language models. You will now rewrite the likelihood of

$$P(\text{Staten måste ta ansvar för de arbetsskadade})$$

using:

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

Given that *Staten* starts with an uppercase letter you can assume that this is the start of a sentence. You will then use a `<s>` symbol in your bigram and trigram approximations.

Language models. Compute the values of

$$P(\text{ansvar})$$
$$P(\text{ansvar}|\text{ta})$$
$$P(\text{ansvar}|\text{måste ta})$$

with the statistics from Table 1 and where N is the total number of words in the corpus. Just give the fractions. Do not try to reduce them. 4 points

Machine learning. Given a vocabulary of 10,000 words, describe how a word would be represented using:

1. one-hot encoding; 2 points
2. embedding vectors of 100 dimensions. 2 points

Table 1: Statistics extracted from Google on October 18, 2023 with the `site:se` prefix

Unigrams	**Freqs.**	**Bigrams**	**Freqs.**	**Trigrams**	**Freqs.**
måste	111,000,000	måste ta	2,920,000	måste ta ansvar	159,000
ta	120,000,000	ta ansvar	1,480,000	–	–
ansvar	32,600,000	måste ansvar	2,100	–	–

Machine learning. Given two multihot encoded documents: $\mathbf{u} = (1, 0, 1, 0, 1)$ and $\mathbf{v} = (0, 1, 1, 0, 1)$, compute:

1. The dot product of the two vectors: $\mathbf{u} \cdot \mathbf{v}$; 1 point
2. The norms of $\mathbf{u}$ and $\mathbf{v}$: $||\mathbf{u}||$ and $||\mathbf{v}||$; 1 point
3. The cosine of $\mathbf{u}$ and $\mathbf{v}$ (do not calculate it exactly). 1 point

Machine learning. In this exercise, you will suppose that you have a multilingual corpus, where the texts are written in one of three possible languages: α, β, or γ. You use the class below to train a language detector.

```
class Model(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.fc1 = nn.Linear(input_dim, 3)

    def forward(self, x):
        return self.fc1(x)
```

1. Describe in a few sentences the purpose of the `__init__` and `forward()` methods overall; 2 points
2. Describe in two or three sentences what you do when you execute this statement: `nn.Linear(input_dim, 3)` 2 points
3. Describe in two or three sentences what you do when you execute this statement: `return self.fc1(x)` 2 points

Machine learning. Once you have created the class, you create a `model` object and fit its parameters with this double loop:

```
for epoch in range(100):
    for X_batch, y_batch in dataloader:
        y_batch_pred = model(X_batch)
        loss = loss_fn(y_pred, y_batch)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Answer each question in two or three sentences:

1. What is the purpose of these two loops? 2 points

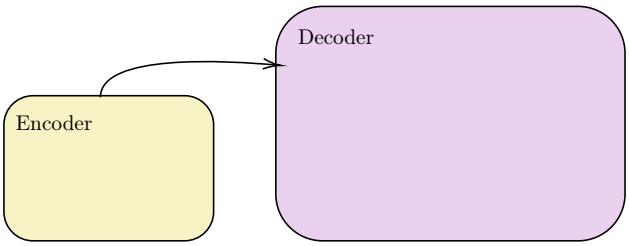


Figure 2: The encoder-decoder architecture

2. What is the content of `X_batch`, `y_batch`, and `y_batch_pred`; 2 points
3. Explain this statement `loss_fn(y_pred, y_batch)`; 2 points
4. Explain this statement `loss.backward()`; 2 points
5. Explain this statement `optimizer.step()`. 2 points

Machine learning. Once trained, your model predicts the language of five samples with the following probabilities after being normalized by the softmax function:

Language	Language index	Language one-hot code	Prediction
α	0	$(1, 0, 0)$	$(0.7, 0.2, 0.1)$
γ	2	$(0, 0, 1)$	$(0.1, 0.6, 0.3)$
β	1	$(0, 1, 0)$	$(0.3, 0.5, 0.2)$
α	0	$(1, 0, 0)$	$(0.4, 0.4, 0.2)$
γ	2	$(0, 0, 1)$	$(0.2, 0.3, 0.5)$

Compute the cross-entropy loss. You will not reduce the logarithms. Just write log and the values. 3 points

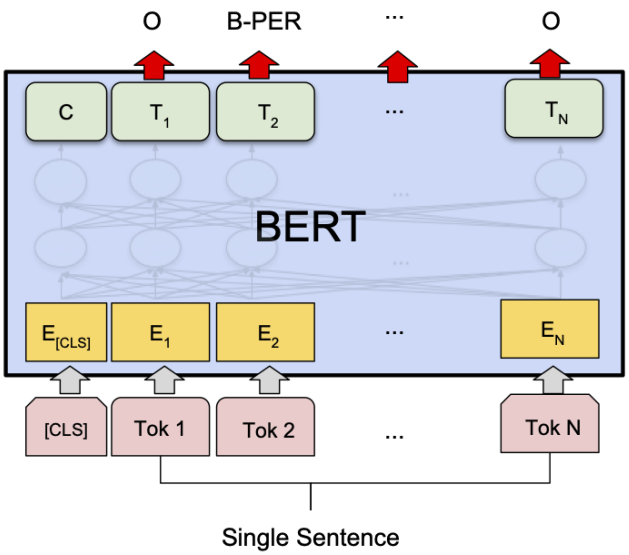
Machine learning. In machine translation, the encoder-decoder architecture has shown the best performance so far. Using characters and the English–French pair: (Hello, Bonjour), annotate the architecture in Fig. 1 with the source and target strings, possibly shifted. You will consider two steps:

1. Training step. You will reproduce (draw) the figure on your exam paper, complement it with the characters, and describe this step; 4 points
2. Inference step. Do the same as for the previous step for inference. 3 points

The encoder and decoder can use a RNN, LSTM, or transformer architecture. Your description should be independent of it.

Machine learning. Transformers

1. Describe qualitatively in a few sentences what is self-attention in a transformer architecture. You will refer to the embeddings in your description and compare them with those of GloVe. 3 points
2. Give the names of the two main components (stacks) of the original transformer architecture. 1 point
3. BERT uses the encoder part of transformer. How is it pre-trained? 2 points



(d) Single Sentence Tagging Tasks:
CoNLL-2003 NER

Figure 3: The BERT transformer for sentence tagging tasks, after Devlin et al., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019

4. You can use BERT for sequence tagging as in Fig. 3. Reproduce the figure and annotate it with the words and chunks from the 3rd question of this examination and tell how you would fine-tune it for this task. 3 points

2 Programming Problem

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

In this programming problem, you will analyze and implement a sentence embedder described in the paper *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks* by Reimers and Gurevych (2019). In the rest of the paper, we will call this system SBERT[2].

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 Application

Before looking at the paper, we will consider the comparison of pairs of sentences. This corresponds to the main application of SBERT and we will use BERT as starting point.

2.1.1 GLUE

GLUE is a popular set of benchmarks used to assess NLP tools. One of the GLUE benchmarks is to compare a pair of sentences and decide if they are similar or not. GLUE has six such tasks called semantic textual similarity (STS). MNLI, QQP, QNLI, STS-B, MRPC, and RTE:

- MNLI, and the earlier SNLI, label each pair with either *entailment*, *contradiction*, or *neutral*;
- QQP and MRPC whether two questions or sentences are equivalent or not;
- STS-B whether two sentences are semantically equivalent with a score ranging from 1 to 5;
- RTE whether the second sentence is an entailment of the first one;
- Finally, in the QNLI corpus, each pair consists of a question and a sentence. The pair is positive if the sentence contains the answer to the question and negative otherwise.

We examine here SNLI as SBERT used them to train its model. It consists of over 500,000 lines with the text of the pairs and their labels. The authors created the dataset by giving volunteers a sentence (the premise) and asking them to write a second sentence (the hypothesis) that is either definitely true (entailment), that might be true (neutral), or that is definitely false (contradiction). Table 2 shows five samples from the SNLI dataset. The last column is the class.

Describe succinctly how you could build a classifier for this corpus with a bag-of-word technique:

1. How you would vectorize the two sentences; 3 points

[2]Note that this paper is unfortunately not very well written, but this is often the case, even in high-level conferences

Table 2: Samples from the SNLI question pairs dataset. Source `https://nlp.stanford.edu/pubs/snli_paper.pdf`

Premise	Hypothesis	Label
A man inspects the uniform of a figure in some East Asian country.	The man is sleeping	contradiction
An older and younger man smiling	Two men are smiling and laughing at the cats playing on the floor.	neutral
A black race car starts up in front of a crowd of people	A man is driving down a lonely road	contradiction
A soccer game with multiple males playing	Some men are playing a sport	entailment
A smiling costumed woman is holding an umbrella	A happy woman in a fairy costume holds an umbrella	neutral

2. What kind of classification architecture you could use. 3 points

2.1.2 BERT

Figure 4 shows the settings BERT uses for classification tasks of sentence pairs, where it concatenates the two sentences and inserts a `[SEP]` symbol in between.

1. Reproduce Fig. 4 on your exam paper and annotate it with two pairs from Table 2 with the input and output (two drawings). Use only the five first words of each sentence; 2 points

2. Given that you have a pretrained BERT model, how would you train the class label? 2 points

2.2 Understanding of the Paper

Read the summary of the paper as well as Sections 1 and 3, *Introduction* and *Model*. In the *triplet objective function*, an anchor is a start sample, the positive sample is close to the anchor, while the negative one is different. Considering a language detector, think of a sentence in Swedish as the anchor. A positive sample would be another sentence in Swedish and a negative one could be a sentence in English.

1. Summarize SBERT in 10 to 15 lines. Note that a three-way softmax classifier is simply a logistic regression with three classes. 8 points

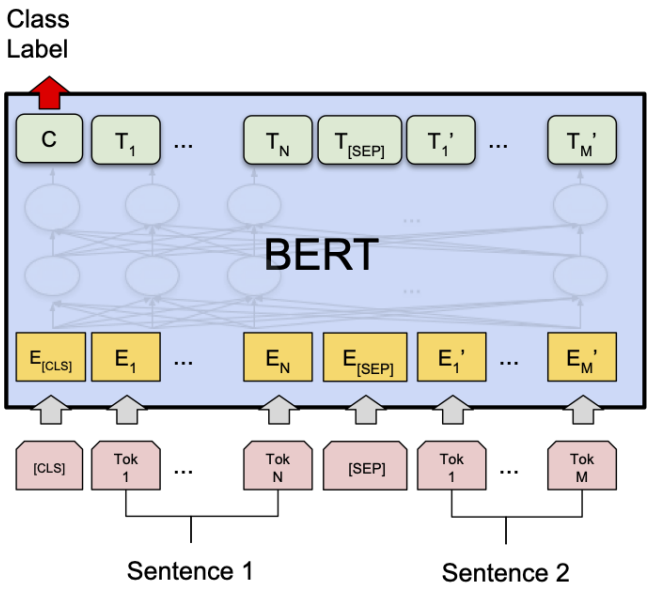
2. In Figure 2 of the paper, comment what is the output of a SBERT encoder (describe what are $\mathbf{u}$ and $\mathbf{v}$) and how the representations of two sentences are compared (after $\mathbf{u}$ and $\mathbf{v}$). 3 points

2.3 Baseline Technique: Overview

In Sect. 4.1, the authors propose a baseline technique for computing a semantic textual similarity between two sentences that uses GloVe embeddings.

1. Describe this technique in 5 to 10 lines. 5 points

2. Reading the results in Table 1, what do you think of the performance of the baseline in relation to its complexity? And relatively to the BERT technique in Fig. 4 . 3 points



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

Figure 4: The BERT transformer for sentence pair classification tasks, after Devlin et al., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019

2.4 Preprocessing

Before you implement the baseline, you will preprocess the corpus in a way that will also serve your subsequent programming of SBERT.

You skip reading the files and you assume that you have an initial dataset, `dataset_str`, that consists of a list of triples `(s1, s2, label)`, where `s1` is a string containing the first sentence, the premise, `s2`, the second one, the hypothesis, and `label` is either *contradiction*, *neutral*, or *entailment*. The first item of `dataset_str` is:

```
('A person on a horse jumps over a broken down airplane.',
 'A person is training his horse for a competition.',
 'neutral')
```

See Table 2 for more examples.

2.4.1 Tokenization

1. Write a `tokenize` function to tokenize a sentence.

   ```
   def tokenize(sentence):
   ```

```
...
return token_list
```

You will use a Unicode regular expression to identify the words, numbers, and punctuation. Note that the GloVe tokens include punctuation signs that you can extract with the `\p{P}` class. 5 points

2. Apply your tokenizer to `dataset_str` so that you tokenize `s1` and `s2` and replace them with tokens. Call the new dataset: `dataset_tokens`. 1 point

After tokenizing `dataset_str`, the first item of `dataset_tokens` is:

```
(['a', 'person', 'on', 'a', 'horse', 'jumps', 'over', 'a',
  'broken', 'down', 'airplane', '.'],
 ['a', 'person', 'is', 'training', 'his', 'horse', 'for', 'a',
  'competition', '.'],
  'neutral')
```

2.4.2 Indexing

To speed up this exercise, you will assume that the GloVe vectors are already stored in a PyTorch matrix called `glove` and the corresponding words in a list called `glove_words`. The three first words in this list are the symbols `'[PAD]'`, `'[UNK]'`, and `'[CLS]'`. The corresponding three first rows of `glove` are initialized with zero vectors. The rest, in the list and in the matrix, consists of the words initially contained in the GloVe file.

1. Build two dictionaries `token2idx` and `idx2token` mapping each word in `glove_words` to an index and vice-versa; 2 points

2. Build two dictionaries `label2idx` and `idx2label` mapping each label in `dataset_tokens` to an index and vice-versa (there are only three labels); 1 point

3. Apply `token2idx` and `label2idx` to `dataset_tokens` to replace the tokens and the labels with indices. If a token in a sentence is unknown, replace it with the `'[UNK]'` index (i.e. 1). Call the new dataset `dataset`. 3 points

After this preprocessing, the first item of `dataset` is:

```
([10, 902, 16, 10, 2870, 11073, 77, 10, 2324, 138, 7353, 5],
 [10, 902, 17, 791, 29, 2870, 13, 10, 994, 5],
 2)
```

2.5 Baseline Technique: Implementation

You will now implement the baseline in Sect. 4.1 of the paper for two sentences stored in two strings: `s1` and `s2`. To make it easier, we create a PyTorch `Embedding` object and assign it the values of `glove`:

```
glove_embs = nn.Embedding(glove.size()[0],
                          glove.size()[1],
                          padding_idx=0).from_pretrained(glove)
```

You will use `glove_embs` to create the baseline.

1. Write a function that takes an input string and the embedding object, `glove_embs`, and that returns the mean of the word embeddings: 5 points

   ```
   def mean_embs(input_str, glove_embs):
       ...
       return mean
   ```

   In this function, you will tokenize the string, look up the indices, and apply `glove_embs` to extract the embeddings. You can compute the mean with `torch.mean()`. Do not forget to give the dimension.

2. Write a cosine function to compute the cosine of two vectors. 2 points

   ```
   def compute_cosine(v1, v2):
       ...
       return cosine
   ```

3. Apply your functions to compute the cosine similarity of the two sentences: `s1` and `s2`. 2 points

2.6 Creating the SBERT Stacks

You will now create a SBERT class. As input, you will use one pair of sentences at a time (no batch or mini-batch) from `dataset`. As input embeddings, you will use the `glove` embeddings.

You will use the PyTorch classes to build an encoder stack. You create an encoder layer with `TransformerEncoderLayer` and then you stack the layers with `TransformerEncoder` as with:

```
encoder_layer = nn.TransformerEncoderLayer(d_model=100, nhead=4)
transformer_encoder = nn.TransformerEncoder(encoder_layer, num_layers=6)
```

Supposing you have a sequence of 10 tokens with an embedding dimension of 100 as in GloVe,

- You create random input embeddings this way:

  ```
  src = torch.rand(10, 1, 100)
  ```

  The number in the middle is the size of the batch, here one. In the rest of the examination, you will use a batch of one. You can ignore it in your programs though.

- Then you apply `transformer_encoder` to output the contextual embeddings with:

  ```
  out = transformer_encoder(src)
  ```

In this section, you will now follow Fig. 1 of the paper and proceed from bottom to top up to the **u** and **v** vectors.

2.6.1 Preliminaries

In Fig. 1 of the paper,

1. How many BERT encoding stacks do you have? What can you tell about them (or it). Read carefully the caption of Fig. 1 in the paper; 2 points
2. And what is the default pooling method? You will use this default in the rest of the examination. 2 points

2.6.2 The SBERT Stacks

In this section, you will create an SBERT class with the `__init__()` and `forward()` methods up to the **u** and **v** output (lower part of Fig. 1 before **u** and **v** are merged).

```
class SBERT(nn.Module):
    def __init__(self, glove)
        super().__init__()
        ...

    def forward(self, u, v):
        ...
        return u, v
```

1. Write the `__init__()` method, where you create an `Embedding` layer from `glove` and the BERT stack or stacks you need. To create a stack, follow the code example above and choose the arguments you want; 10 points
2. Write the `forward()` method, where you extract the GloVe embeddings of two tensors containing the indices of your two sentences and apply your BERT stack or stacks. You will then pool the output of each stack with the default method. 10 points
3. Create an `sbert` object from the `SBERT` class. Apply it to a sentence pair: the premise and hypothesis. 1 point

Applying `sbert` to the first item of `dataset` results in an output of two tensors:

```
> sbert(dataset[0][0], dataset[0][1])
(tensor([[ 1.1657, -0.2701, ...]]),
 tensor([[ 0.3433, -0.5665, ...]]))
```

where each tensor has 100 dimensions.

2.7 Adding a Logistic Regression Head

You will now create a head to classify the combined outputs **u** and **v** as described in the *classification objective function*. You will start from the `SBERT` class from the previous section and complement it. Just indicate what you need to modify.

1. In the `__init__()` method, add an object that implements the logistic regression head. You will pass the dimension of the vectors `d_model` (n in the paper) and the number of classes k as parameters to the `SBERT` class. You will need the `Linear` class. 5 points

2. In the `forward()` method,

   (a) Combine the **u** and **v** vectors to form a new vector. This is just a concatenation with **u** and **v**, and a third term (Follow Fig. 1 of the paper as well as Table 6). You will use the `torch.abs()` and `torch.cat(tensors, dim=-1)` functions; 3 points

   (b) Add the logistic regression function and return the class logits. 2 points

3. Create an `sbert` object from the new `SBERT` class. Apply it to a sentence pair: the premise and hypothesis. 2 points

For the first item of `dataset`, we have an output of three logits:

```
> sbert(dataset[0][0], dataset[0][1])
tensor([[ 0.2100, -0.0471,  0.3710]]
```

2.8 Training SBERT

You can now train a model with the natural language inference (NLI) corpus in Table 2.

1. Define a loss and an optimizer; 4 points

2. Write the loop to train the model. You will read one triple at a time from `dataset`. Note that a similar loop was given in Part I. 4 points

2.9 Fine-tuning SBERT

After you have pretrained SBERT, you can fine-tune it on STS data to output the similarity between two sentences as shown in Fig. 2 of the paper. While SNLI has three labels, the STS dataset labels the pairs with six grades ranging from 0 to 5, where 0 means that the two sentences are completely different and 5 that they are equivalent.

The paper does not define the training loss, but the SBERT site[3] provides a clarification of it: $||\text{input\_label} - \cos(\mathbf{u}, \mathbf{v})||_2$, which correspond to the Euclidian norm.

1. Modify the `forward` function in `SBERT` so that you have the output pictured in Fig. 2 of the paper; 3 points

2. Tell how you would modify the loss so that you can fine-tune `SBERT`. Note that PyTorch has a mean squared error loss called `nn.MSELoss`. 2 points

2.10 A Last Word

If you found this exercise interesting, you can try to implement the complete model after the examination. If you decide to do it, simply download SNLI and create a notebook. Then follow the steps of the exam. There are a few details I set aside like the conversion of the lists to long tensors, but they are easy to add.

[3] `https://www.sbert.net/docs/package_reference/losses.html#cosinesimilarityloss`