NLP exam HS23

HSLU Hochschule Luzern

November 22, 2023
Time: 60 minutes

Full name: _____

Initials: _____

Instructions

- This is a *closed book* exam. No course material or other additional material is allowed.
- Fill in your full name as well as your initials in block letters at the top of this page.
- Write your initials on each page.
- This exam consists of **11 pages**. Make sure you have all pages.
- If you have questions, raise your hand to clarify any uncertainties.
- Use the designated space for your answers. You may use the back of the page as additional space. If you do, indicate that your answer continues on the back.
- Write clearly and legibly. Only readable answers give points.
- Sign the declaration of academic integrity below.
- Good luck with the exam!

Question:	1	2	3	4	5	6	7	8	Total
Points:	6	2	14	9	14	3	8	4	60
Score:									

Declaration of Academic Integrity

By signing below, I pledge that the answers of this exam are my own work without the assistance of others or the usage of unauthorized material or information.

Signature: .....

1. **Introduction to NLP**

   (a) [2 points] Name 2 reasons why we use libraries when writing code.

   Solution:

   - speed up development
   - less code = fewer chances for mistakes
   - community has tested the code
   - documentation already exists
   - library code is optimized for performance

   (b) [2 points] When using byte-pair encoding (BPE) on the entire Internet, check the words which would most likely get their own vocabulary entry:

   ☒ **the**

   ☐ Nunapitchuk (← this is a city in Alaska with 500 inhabitants)

   ☒ **Fahrrad**

   ☐ Bezirksratsvorsteheruniform

   (c) [2 points] For the previous question, explain why.

   Solution: The words that appear often in the training data will most likely get their own token.

2. **Embeddings**

   (a) [1 point] Are embeddings from topic models contextualized (i.e. they change with different contexts)?

   Solution: No, topic model embeddings are static.

   (b) [1 point] Explain what fastText adds over word2vec or GloVe.

   Solution: Expected answer: Character n-gram embeddings. I also gave the point for: Better performance.

3. **Recurrent Neural Networks**

(a) [10 points] Find the bugs (only mistakes, ignore performance improvements) in the following training loop that will lead to the program crashing or the model not training at all. Assume all imports are correct. Mark each mistake and write a short explanation of what is wrong with it. There are 5 mistakes.

```
def train(dataloader, embedding, lstm, classifier, optimizer, config):
  for i in tqdm.tqdm(range(config.num_epochs)):
    for batch in dataloader:
      assert len(batch) == 1, "Code only works with a batch size of 1"
      example = batch[0]
      emb = embedding(example)
      h0 = torch.zeros(config.num_layers, config.hidden_dim)
      c0 = torch.zeros(config.num_layers, config.hidden_dim)
      with torch.no_grad():
        outputs, (hn, cn) = lstm(emb, (h0, c0))
        # classify from each hidden state except the last
        predictions = classifier(outputs[:-1])
      # targets are shifted to the left,
      # predict each token except the last
      targets = example[:-1]
      loss = F.cross_entropy(input=targets, target=predictions)
      loss.backward()
      optimizer.zero_grad()
      optimizer.step()
      return loss
```

Solution:

- with torch.no_grad in training loop disables gradients
- predict each token except the first (i.e. `targets = example[1:]`)
- order of predictions and targets is wrong in cross-entropy calculation
- optimizer.zero_grad after loss.backward deletes all gradients
- Function returns after first training example

(b) [1 point] A neural network classifier usually computes scores for each of its output classes, called logits. In what range are logits (give min and max value)?

Solution: -infinity to +infinity

(c) [2 points] Why can't we use logits as probabilities? Give 2 reasons.

Solution: They are not in the interval [0, 1] and do not sum to 1. (0.5 points for "they are not normalized", need to specify to what value probabilities are normalized.)

(d) [1 point] How do we get probabilities from logits?

Solution: We use a normalizing function such as the softmax.

4. **Attention**

(a) [1 point] Before the introduction of attention, how was the information from the encoder passed to the decoder in a sequence-to-sequence RNN?

Solution: The last encoder hidden state was used to initialize the initial hidden state of the decoder.

(b) [4 points] Attention was first developed for sequence-to-sequence RNNs.

i. Describe the idea behind attention.

Solution: Each decoder time step decides how much information to incorporate from each encoder hidden state (1 point). It pays more attention to encoder hidden states that are more useful for its task (1), e.g. predicting the next token.

ii. Describe how it is done mathematically. What are the steps to compute attention? Don't give formulas.

Solution: Attention weights are computed that serve as weights for the encoder hidden states (0.5 points) when computing the context vector. The attention scores between decoder and encoder hidden states (0.5) are computed in the attention function (0.5), for which e.g. additive or multiplicative attention can be used. These scores are then converted into probabilities with a softmax (0.5).

(c) [2 points] Explain how ELMo includes the context into its word representations.

Solution: It runs a sentence through a 2-layer BiLSTM, then concatenates static word embeddings with contextual embeddings from layers 1 and 2.

(d) [2 points] For the following tasks, do you expect contextualized embeddings to improve performance by a lot?

i. Find synonyms.

☒ **Small improvement**

☐ Large improvement

ii. Word sense disambiguation: Determine the meaning of polysemous words in a sentence. Polysemy = a single word has multiple meanings.

☐ Small improvement

☒ **Large improvement**

5. **Transformer**

   (a) [8 points] Given a library that implements the following methods (in alphabetical order), create a forward pass of a Transformer decoder by picking the right methods and combining them in the correct order. Use the letter for each method. Don't fill in the method's arguments. Neither points nor the space in a line correspond to the number of methods you have to pick. If you write `ABC` on the first line, that means that first method A is applied to input_ids, then B to the output of A, then C.

   A. add_position_embedding
   B. add_residual
   C. apply_relu
   D. apply_softmax
   E. cross_attention
   F. get_embedding
   G. layer_norm
   H. project_to_higher_dim
   I. project_to_lower_dim
   J. project_to_same_dim
   K. select_hidden_state
   L. self_attention
   M. self_attention_with_causal_mask

```
def transformer_decoder_forward(target_sequence: str) -> torch.Tensor:
  input_ids = tokenizer.encode_plus(target_sequence, return_tensors='pt')


  ..................................................................
  for i in range(config.num_layers):


      ..............................................................


  next_word_probabilities = ..........................................
  return next_word_probabilities
```

Solution:

1. FA (2 points)
2. MBG EBG HCIBG (1 point for M, E, HCI each, 1 point for BG)
3. K(JC)HD (1 point for K, 1 point for HD)

(b) [6 points] We learned that the attention modules of the previous library had bugs, but the rest was fine. Fortunately, we found another library with the following methods. Implement cross-attention by filling in the missing methods (use the letters) and **fill in their arguments**.

A. multiply_queries_and_keys
B. multiply_with_values
C. output_projection
D. project_to_key
E. project_to_query
F. project_to_value
G. reshape_for_attention_heads
H. scale_with_sqrt_of_dim
I. softmax
J. split_to_heads

```
def cross_attention(encoder_outputs, decoder_inputs):
  q = ............(.....................................................)
  k = ............(.....................................................)
  v = ............(.....................................................)
  qhs, khs, vhs = split_to_heads(q, k, v)
  attention_outputs = []
  for qh, kh, vh in zip(qhs, khs, vhs):
    x = ............(.....................................................)
    x = ............(.....................................................)
    x = ............(.....................................................)
    x = ............(.....................................................)
    attention_outputs.append(x)
  output = torch.cat(attention_outputs, dim=-1)
  return output_projection(output)
```

Solution:

```
def cross_attention(encoder_outputs, decoder_inputs):
  q = project_to_query(decoder_inputs)     (E, 1 point)
  k = project_to_key(encoder_outputs)      (D, 0.5)
  v = project_to_value(encoder_outputs)    (F, 0.5)
  qhs, khs, vhs = split_to_heads(q, k, v)
  attention_outputs = []
  for qh, kh, vh in zip(qhs, khs, vhs):
    x = multiply_queries_and_keys(qh, kh)  (A, 1)
    x = scale_with_sqrt_of_dim(x)          (H, 1)
    x = softmax(x)                         (I, 1)
    x = multiply_with_values(x, vh)        (B, 1)
    attention_outputs.append(x)
  output = torch.cat(attention_outputs, dim=-1)
  return output_projection(output)
```

6. **Pretraining**

(a) [3 points] For each of the following tasks, select the suitable model(s).

i. Write a summary for this article.

☐ Encoder-only

☒ **Encoder-decoder**

☒ **Decoder-only**

ii. Label a text with named entities.

☒ **Encoder-only**

☐ Encoder-decoder

☐ Decoder-only

iii. Determine whether the premise entails the hypothesis.

☒ **Encoder-only**

☐ Encoder-decoder

☐ Decoder-only

iv. Answer the question by returning a text span from the given paragraph.

☒ **Encoder-only**

☐ Encoder-decoder

☐ Decoder-only

v. Answer the following question: Why is the sky blue?

☐ Encoder-only

☒ **Encoder-decoder**

☒ **Decoder-only**

vi. Translate the following sentence into French: This is easy.

☐ Encoder-only

☒ **Encoder-decoder**

☒ **Decoder-only**

7. **Text Generation**

   (a) [8 points] Your colleague AM has been working on a summarization class, and created a pull request (see next page). You were assigned to peer review his code. Check the code for correctness and efficiency, and suggest improvements. You only have to say what needs to be changed and why. You do not have to provide the implementation for the changes.

   Solution: (2 points for each improvement found, up to a maximum of 8.)

   - Initialize the `nn.Module` with `super().__init__()`.
   - Put model in eval state with `.eval()`.
   - Move model and data to GPU if one is available.
   - `run` needs an additional `self` argument, tokenizer and model should be called with `self.`, otherwise not visible in run method.
   - Process articles in batches for efficiency.
   - Default max_length for the generate function will be 20 tokens, which is usually too short. Should be increased.
   - Default decoding algorithm is greedy decoding. Change to beam search/top-k sampling/nucleus sampling for better outputs.
   - Skip special tokens when decoding for cleaner outputs, no postprocessing necessary.

8. **Text Classification**

   (a) [1 point] Why is negation a difficult problem for sentiment lexicons?

   Solution: Lexicons treat words individually, negation depends on context.

   (b) [3 points] How would you use an NLI model to automatically fact check a politician's arguments during a debate? Assume you have access to a strong online search that returns relevant websites for each argument.

   Solution: Search for relevant websites, for each website run the NLI model (1 point). Compute a score (1): +1 If the website entails the argument (1), -1 if the website contradicts the argument, 0 otherwise. Potentially weigh

```python
from torch import nn
from transformers import BartTokenizer, BartForConditionalGeneration


class Summarizer(nn.Module):
  """ Summarization with a BART model. """

  def __init__(self):
    """ Loads the tokenizer and model. Uses bart-large-cnn, which has
    ↪  been finetuned on the CNN/DailyMail summarization dataset. """
    self.tokenizer =
    ↪  BartTokenizer.from_pretrained("facebook/bart-large-cnn")
    self.model = BartForConditionalGeneration.from_pretrained(
      "facebook/bart-large-cnn"
    )

  def  run(article):
    """ Encodes the article, generates the summary, decodes the text.
    ↪  Returns the summary text. """
    inputs = tokenizer.encode_plus(article, return_tensors='pt')
    # AM: hmm, not sure about this one...
    summary_ids =  model.generate(**inputs)
    summary = tokenizer.decode(summary_ids[0])
    return summary


# We test our implementation by generating a summary for the below
↪  example article.
article = """LONDON, England (Reuters) -- Harry Potter star Daniel
↪  Radcliffe gains access to a reported £20 million ($41.1 million)
↪  fortune as he turns 18 on Monday, but he insists the money won't cast
↪  a spell on him. To the disappointment of gossip columnists around the
↪  world, the young actor says he has no plans to fritter his cash away
↪  on fast cars, drink and celebrity parties. "I don't plan to be one of
↪  those people who, as soon as they turn 18, suddenly buy themselves a
↪  massive sports car collection or something similar," he told an
↪  Australian interviewer earlier this month."""
summarizer = Summarizer()
summary = summarizer.run(article)
print(summary)
# Output: <s> Harry Potter star Daniel Radcliffe gets £20M fortune as he
↪  turns 18 Monday . Young actor says he has no plans to fritter his
↪  cash away . </s>
```

websites with their reputation.