NLP exam HS24

HSLU Hochschule Luzern

January 16, 2025
Time: 90 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 **15 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.
- You can answer in German or English (and you can mix).
- Sign the declaration of academic integrity below.
- Good luck with the exam!

Question:	1	2	3	4	5	6	7	8	9	10	Total
Points:	5	10	9	10	4	18	10	14	4	6	90
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] Consider a tokenizer with BPE tokenization (e.g. the BertTokenizer). What happens to the tokens it outputs when the input text has spelling mistakes?

   Solution: The output tokens will be different from tokens without spelling mistakes. If the spelling mistake has not been seen often enough when the BPE tokens were decided, there will be no corresponding known token and they will be broken up into parts of known BPE tokens, down to single characters.
   (2 points for different tokens)

   (b) [3 points] We now run the input text with spelling mistakes through BERT. How will the spelling mistakes affect the quality of our model's outputs? What does it depend on?

   Solution: The quality of input text with spelling mistakes is generally worse, due to fewer training data with this exact spelling mistake. The severity of the quality degradation depends on how common the spelling mistake is, i.e. whether it has often been seen during pretraining/finetuning.
   (1 point for quality becomes worse, 2 for it depends if spelling mistake is common/has been seen in pretraining/finetuning, -1 point for general spelling mistakes in training but not this specific one)

2. **Embeddings**

(a) [4 points] Explain the TF and the IDF part of TF-IDF. For each, say what it is and why it's a useful metric of token importance.

Solution: TF measures the term frequency, i.e. how often a token appears in a document. The more often a token appears in a document, the more important it is for that document.
IDF is the inverse document frequency, i.e. 1/(number of documents the token appears in). If a term only appears in few documents, it is very specific to that document and receives high importance. Conversely, stopwords appear in almost every document, and are therefore not of high importance in any document.
(2 points each)

(b) [4 points] Name one advantage each of low-dimensional embeddings (e.g. word2vec) and high-dimensional embeddings (e.g. bag-of-words).

Solution: Low-dimensional: more efficient to compute with, trained embeddings store more information and can capture meaning
High-dimensional: dimensions are understandable, embeddings don't need to be trained
(2 points each, max 2 points per embedding type)

(c) [2 points] Given the following vectors for man, woman and king, estimate the vector for queen.

$$\text{man} = \begin{pmatrix} 4 \\ 1 \\ 0 \end{pmatrix}, \quad \text{woman} = \begin{pmatrix} 5 \\ -2 \\ 9 \end{pmatrix}, \quad \text{king} = \begin{pmatrix} 1 \\ 17 \\ -3 \end{pmatrix}$$

Solution: queen = king - man + woman

$$\text{queen} = \begin{pmatrix} 2 \\ 14 \\ 6 \end{pmatrix}$$

(2 points for correct solution, -0.5 for single mistake, otherwise 0 points)

3. **Recurrent Neural Networks**

Here is the formulation of an LSTM:

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f) \tag{1}$$
$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i) \tag{2}$$
$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o) \tag{3}$$
$$\tilde{c}_t = \tanh(W_c h_{t-1} + U_c x_t + b_c) \tag{4}$$
$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t \tag{5}$$
$$h_t = o_t * \tanh(c_t) \tag{6}$$

(a) [3 points] Explain the purpose of each of the 3 gates: $f_t, i_t,$ and o_t.

Solution: f_t: Forget gate, decides what part of the memory (cell state of the previous time step) to keep/remove.
i_t: Input gate, decides what part of the new memory (new cell memory $\tilde{c}_t$) to add to the memory (cell state).
o_t: Output gate, decides what part of the cell state should be output at the current time step t as the hidden state h_t.
(1 point each)

(b) You need to get rid of one gate.

i. [3 points] What do you do? Explain why.

Solution: Combine the forget gate and the input gate into a reset gate, as in the GRU architecture. Since the cell state is a linear combination of c_{t-1} and $\tilde{c}_t$, we can combine them with a single gate (f_t, i_t, or a new gate while removing the unused ones).
(3 points for a good explanation)

ii. [1 point] Which equation(s) above (1-6) do you remove?

Solution: 2 and 5. (or 1 and 5, or 1, 2, and 5).
(1 point for not forgetting any unused equations)

iii. [2 points] Which equation(s) do you add?

Solution:

$$c_t = f_t * c_{t-1} + (1 - f_t) * \tilde{c}_t$$

(2 points for correct equation, in combination with removed equations above)

4. **Advanced RNNs**

(a) [10 points] The code below shows the Python implementation of the `__init__` method of a 2-layer bidirectional sequence-to-sequence LSTM. Unfortunately, parts of the code have been lost. Fill in the gaps to reconstruct it.

```
class Seq2seqBiLSTM(nn.Module):
    """ Bidirectional 2-layer sequence-to-sequence LSTM. """

    def __init__(self, embedding_dim, hidden_dim):
        super().__init__()
        self.num_encoder_layers = 2
        self.num_decoder_layers = 2
        self.bidirectional = True
        self.num_directions = .....
        self.encoder_layer_1 = nn.LSTM(
            input_size=.....,
            hidden_size=.....,
            num_layers=1,
            bidirectional=.....,
        )
        self.encoder_layer_2 = nn.LSTM(
            input_size=.....,
            hidden_size=.....,
            num_layers=1,
            bidirectional=.....,
        )
        self.decoder = nn.LSTM(
            input_size=.....,
            hidden_size=.....,
            num_layers=self.num_decoder_layers,
            bidirectional=.....,
        )
```

Solution:

```python
class Seq2seqBiLSTM(nn.Module):
    """ Bidirectional 2-layer sequence-to-sequence LSTM. """

    def __init__(self, embedding_dim, hidden_dim):
        super().__init__()
        self.num_encoder_layers = 2
        self.num_decoder_layers = 2
        self.bidirectional = True
        self.num_directions = 2 if self.bidirectional else 1
        self.encoder_layer_1 = nn.LSTM(
            input_size=embedding_dim,
            hidden_size=hidden_dim,
            num_layers=1,
            bidirectional=self.bidirectional,
        )
        self.encoder_layer_2 = nn.LSTM(
            input_size=self.num_directions * hidden_dim,
            hidden_size=hidden_dim,
            num_layers=1,
            bidirectional=self.bidirectional,
        )
        self.decoder = nn.LSTM(
            input_size=embedding_dim,
            hidden_size=self.num_directions * hidden_dim,
            num_layers=self.num_decoder_layers,
            bidirectional=False,
        )
```

(1 point for each gap)

5. **Attention**

(a) [4 points] Explain the difference between self-attention and the type of attention used in sequence-to-sequence RNNs.

> **Solution:** In self-attention, the tokens pay attention to all the tokens of the *same* sequence, while in cross-attention, the decoder tokens pay attention to the encoder tokens.
> (2 points for each explanation)

6. **Transformer**

```python
def forward(self, x):
    output = x + self.pos_emb(x)  # add absolute position embeddings
    for layer in self.layers:  # iterate over layers
        # project to queries, keys and values
        q = self.project_q(output)
        k = self.project_k(output)
        v = self.project_v(output)
        # split to attention heads
        qs, ks, vs = self.split(q), self.split(k), self.split(v)
        # dot-product attention
        head_outputs = []
        for q_h, k_h, v_h in zip(qs, ks, vs):
            attention_scores = q_h @ k_h.tranpose(-1, -2)
            attention_scores /= math.sqrt(self.head_dim)
            attention_probs = F.softmax(attention_scores, dim=-1)
            head_outputs.append(attention_probs @ v_h)
        attn_out = torch.cat(head_outputs, dim=-1)  # concatenate
        attn_out = self.output_projection(attn_out)
        attn_out += layer_input  # add residual
        attn_out = self.layer_norm_1(attn_out)  # layer norm
        # FFN
        ffn_out = self.up_projection(attn_out)
        ffn_out = F.relu(ffn_out)
        ffn_out = self.down_projection(ffn_out)
        ffn_out += attn_out  # add residual
        output = self.layer_norm_2(ffn_out)  # layer norm
    return output  # final layer hidden states
```

(a) [12 points] Above you see an implementation of the forward pass of a Transformer encoder. What is required to change it for a Transformer decoder? For each change, list the line number at which you change (or after which you insert) something, then write the replaced/inserted line. You can also copy lines by giving the line numbers ("copy 42"). Correct Python syntax will not be graded.

Solution:

- 1: add `encoder_outputs` to function arguments (2 points)
  - def forward(self, x, encoder_outputs)
- 13 or 14: add causal mask to `attention_scores` (5 points)
  - causal_mask = torch.tril(torch.ones(batch_size, seq_len, seq_len))
  - scores_to_mask = 1 - causal_mask
  - attention_scores.masked_fill_(scores_to_mask, -1e8) or attention_scores -= scores_to_mask * 1e8
- 20: add cross-attention (5 points)
  - copy 5
  - 6: k = self.project_k(encoder_outputs)
  - 7: v = self.project_v(encoder_outputs)
  - copy 8-20

  (better but not necessary for full points: additional parameters for q, k, v projections and layer norm, e.g. self.project_q_2)

(-1 point per mistake for partial solutions)

(b) [4 points] In the Transformer architecture, why can't we add relative position embeddings just once to the input embeddings like we can for absolute position embeddings?

Solution: Absolute position embeddings are fixed for every input token, but relative position embeddings change based on which token is paying attention. For a sequence w_1, w_2, w_3, the relative position embedding of $w_1 \to w_2$ is $+1$, but for $w_2 \to w_2$ it is 0.

(c) [2 points] Where do we have to add it instead?

Solution: In self-attention.

7. **Pretraining**

(a) [6 points] Create BERT pretraining data from the sentence: *The quick brown fox jumps over the lazy dog.* Create a single sentence. Apply BERT's MLM pretraining objective and use each possible distortion of the input at least once. Label the distortions you applied.

Solution: The [MASK] blue fox jumps [MASK] the agile dog.

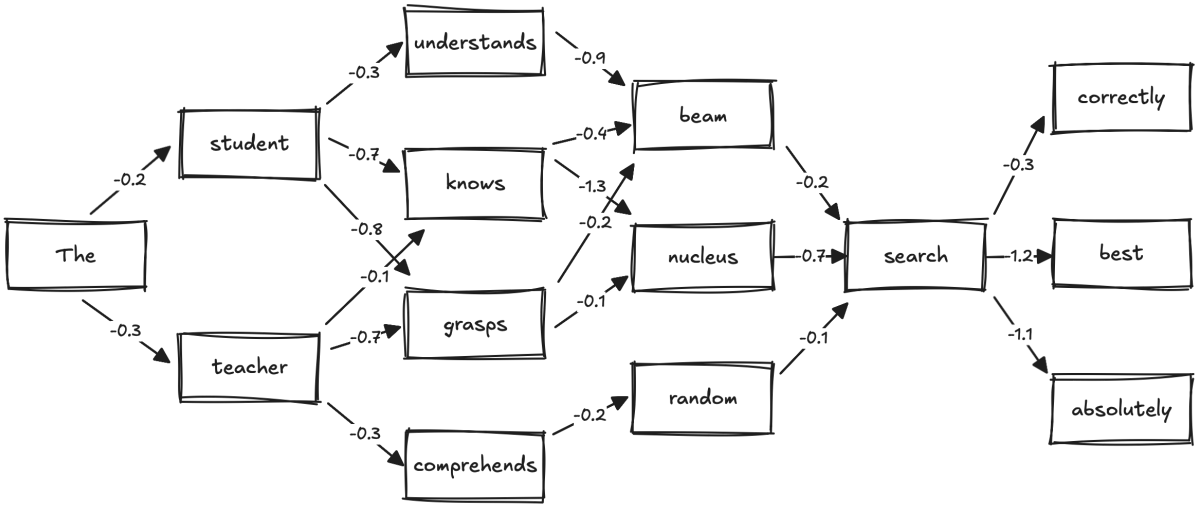
- quick → [MASK], over → [MASK]: replaced with a mask token
- brown → blue, lazy → agile: replaced with a different word
- All other words (e.g. The → The): kept the same word (and used in loss computation)

(2 points per correctly applied and labeled distortion, -2 points for creating 3 sentences, -1 point for wrong distortions)

(b) [4 points] Fact check the following statement s: *Joe Biden won the 2024 US presidential election.* You are only given a pure pretrained language model (no instruction tuning). Use PET-style verbalizers to fact check statement s.

Solution: Concatenate statement s with a verbalizer for *correct* and *incorrect* labels, then compare the language model's probability for both completions. Verbalizers should be the same length and use common words. A label *unveracious* itself would get low probability compared to a common label like *correct*.
correct = p("s This statement is correct.") > p("s This statement is wrong.")
(2 points for correct idea, 1 point for each verbalizer)

8. **Text Generation**



Beam search with transitions showing log probabilities.

(a) [10 points] Run beam search with length normalization on the example shown above. The starting token *The* is already given. Use the following settings: `num_beams` $= 2$, `min_new_tokens` $= 5$, `length_penalty` $= 1$. The length-normalized score with length penalty α is computed as:

$$\text{score}(h) = \frac{\log(p(h))}{\text{length}(h)^{\alpha}}$$

Write down the final hypotheses together with their logprobs.

Solution: Since we have to generate at least 5 tokens, we run beam search to the end. We sum the log probabilities and always keep the 2 hypotheses with the highest sum. We keep track of the sum of logprobs and the number of generation steps in brackets. The final number is the total length-normalized sum of logprobs.
Top hypotheses:

- The student (-0.2/1) understands (-0.5/2) beam (-1.4/3) search (-1.6/4) correctly (-1.9/5)
- The teacher (-0.3/1) knows (-0.4/2) beam (-0.8/3) search (-1/4) correctly (-1.3/5)

(1 point for each correctly selected word, -1 point for wrong choices or wrong sum of logprobs)

(b) [4 points] Now do it again, this time with `min_new_tokens` = 3. Write down the final hypotheses together with their logprobs.

Solution: Now the generation can terminate after 3, 4, or 5 steps. Top hypotheses:

- The student (-0.2/1) understands (-0.5/2) beam (-1.4/3) search (-1.6/4) correctly (-1.9/5)
- The teacher (-0.3/1) knows (-0.4/2) beam (-0.8/3) search (-1/4)

(2 points for each correct hypothesis and logprobs)

9. **Text Classification**

(a) [4 points] You train four models for text classification. For each of the following models, check the boxes of all the operations that **absolutely should not** be performed.

word2vec-GoogleNews-300

- ☒ **Lemmatize the input.**
- ☒ **Lowercase the input.**
- ☐ Stopword removal.
- ☐ Train your own classifier.

A randomly initialized Transformer.

- ☐ Lemmatize the input.
- ☐ Lowercase the input.
- ☐ Stopword removal.
- ☐ Train your own classifier.

bert-base-cased

- ☒ **Lemmatize the input.**
- ☒ **Lowercase the input.**
- ☒ **Stopword removal.**
- ☐ Train your own classifier.

distilbert-base-uncased-qa-boolq

- ☒ **Lemmatize the input.**
- ☐ Lowercase the input.
- ☒ **Stopword removal.**
- ☒ **Train your own classifier.**

Solution: (0.25 points per correct checkbox, 0 points for no checkbox marked at all)

10. **Guest Lectures**

(a) [2 points] In the NLP in Pharma guest lecture, we have seen that medical coding is a tough challenge, with roughly 80k labels (lowest level terms) in the Medical Dictionary for Regulatory Activities (MedDRA) to select from. Assume that ChatGPT has seen the (proprietary) dictionary in its pretraining. Write a prompt to make ChatGPT output the correct label for the symptom *stomach ache*.

Solution: What is the MedDRA lowest level term for the symptom "stomach ache"?
(1 point for mentioning MedDRA, 1 for mentioning stomach ache)

(b) [4 points] ChatGPT answers with: *The MedDRA label is "abdominal pain upper"*. How do you check this result with the help of an NLI model and a medical article describing stomach ache (which contains the correct MedDRA label)?

Solution: Run the NLI model with the medical article as the premise and *The MedDRA label for stomach ache is "abdominal pain upper"* as the hypothesis. We have to create a hypothesis that makes sense, just passing the label is too ambiguous. If the model classifies the pair as *entailment*, we can trust ChatGPT's answer.
(2 points for selecting the right premise and hypothesis, 2 points for correctly deducing the fact check result from the label)