**Midterm Exam**

**Winter 2023**

**Conversational AI**

| **Student Name** |  |
| --- | --- |
| **Student ID** |  |

(please write your name and ID clearly)

**Instructions:**

* The midterm is closed book with no cheat sheet.
* Discussing with the other students is not allowed. Ask the instructor if a question is not clear to you.
* A scientific calculator is required. Laptops/tablets cannot be used.
* Bags, coats, and cell phones must be placed at the edge of the room.
* Cell phones must be placed on silent, before being stowed away in a bag/coat.
* You have **2 hours** (120 minutes) to reply to the 25 questions.
* A set of potentially useful equations is attached at the end of the exam text.
* If you need more space than that allocated in the boxes, please use the white pages attached at the end of this document. Add a note both on the box and on the white paper to make clear the connection (e.g. write “Continue at the end” in the box of P.2 and “Continuation of P.2” in the white paper).

**Marks:**

| **P1** | Conversational AI | / 4 |
| --- | --- | --- |
| **P2** | Deep Learning | / 6 |
| **P3** | Convolutional Neural Networks | / 4 |
| **P4** | Recurrent Neural Networks | / 6 |
| **P5** | Transformers | / 6 |
| **P6** | Speech Processing | / 4 |
| **Total** |  | / 30 |

| **P1** | **Conversational AI Basics** |
| --- | --- |

**P1.1 [2 points]** Briefly explain the purpose of keyword spotting in a conversational AI pipeline. Mention common approaches and challenges. Finally, specify if it is a supervised or unsupervised task.

| **Keyword Spotting** aims to detect one or more keywords in a speech recording.  Common Approaches:   * Small **neural networks** (CNNs, RNNs) trained on standard speech features (e.g, spectrograms, FBANKs, MFCCS).   Challenges:   * Being robust against **noise** and **reverberation**. Keywords are often pronounced far from the recording device. * Being robust against similar **words**. * Minimize False Positives and False Negatives * Achieve good performance with a **small-footprint** model working in **real-time**   See Slides Week1, “Keyword Spotting”. | |
| --- | --- |

**P1.2 [2 points]** Which of the following statements on Analog-to-digital (A/D) conversion are true (one or more than one can be true)? Check the right answer(s).

1. The quantization inevitably causes a loss of information with respect to the analog signal.
2. To avoid losing information, the sampling frequency must be greater or equal to the maximum frequency in the analog signal.
3. Aliasing happens when the speech signal is oversampled.
4. Telephone speech signals are sampled at 16 KHz.
5. When sampling at 16 KHz, we get 16000 samples for each second of audio.

| **P2** | **Deep Learning** |
| --- | --- |

**P2.1 [2 points]** The code available in appendix (A.1) attempts to implement an MLP for handwritten digit classification using the MNIST dataset. When you run it, however, the model does not converge and you see the following output:

Epoch: 1 train\_loss=2.3043, test\_loss=2.3036, test\_err=86.93%

Epoch: 2 train\_loss=2.3043, test\_loss=2.3036, test\_err=86.93%

Epoch: 3 train\_loss=2.3043, test\_loss=2.3036, test\_err=86.93%

Epoch: 4 train\_loss=2.3043, test\_loss=2.3036, test\_err=86.93%

Epoch: 5 train\_loss=2.3043, test\_loss=2.3036, test\_err=86.93%

Epoch: 6 train\_loss=2.3043, test\_loss=2.3036, test\_err=86.93%

Epoch: 7 train\_loss=2.3043, test\_loss=2.3036, test\_err=86.93%

Epoch: 8 train\_loss=2.3043, test\_loss=2.3036, test\_err=86.93%

Epoch: 9 train\_loss=2.3043, test\_loss=2.3036, test\_err=86.93%

Epoch: 10 train\_loss=2.3043, test\_loss=2.3036, test\_err=86.93%

Epoch: 11 train\_loss=2.3043, test\_loss=2.3036, test\_err=86.93%

Epoch: 12 train\_loss=2.3043, test\_loss=2.3036, test\_err=86.93%

Epoch: 13 train\_loss=2.3043, test\_loss=2.3036, test\_err=86.93%

Epoch: 14 train\_loss=2.3043, test\_loss=2.3036, test\_err=86.93%

Epoch: 15 train\_loss=2.3043, test\_loss=2.3036, test\_err=86.93%

Epoch: 16 train\_loss=2.3043, test\_loss=2.3036, test\_err=86.93%

Epoch: 17 train\_loss=2.3043, test\_loss=2.3036, test\_err=86.93%

Epoch: 18 train\_loss=2.3043, test\_loss=2.3036, test\_err=86.93%

Epoch: 19 train\_loss=2.3042, test\_loss=2.3036, test\_err=86.93%

Epoch: 20 train\_loss=2.3043, test\_loss=2.3036, test\_err=86.93%

Identify the bug and explain why it causes the model to not converge. Write one line of code to fix it.

| **Bug Description:** The step function of the optimizer is not called. This way,  The parameters are not updated and stay randomly initialized.  **I can fix it in this way:**  We can fix that by calling optimizer.step() right after l.backward(). | |
| --- | --- |

**P2.2 [2 points]** Briefly explain the role of activation functions in neural networks. Why do we need it? Mention some examples of functions commonly used and explain why some activations are better to mitigate vanishing gradient issues.

| Neural networks such as MLPs typically perform a linear transformation of their inputs followed  by a non-linear activation function: f(wx)  The role of the activation function is thus to add non-linearities to the neural network, which will  otherwise behaves as a simple linear model.  Examples of activation functions are sigmoid, tanh, ReLU, and Leaky ReLU.  Some activation functions such as sigmoid and tanh have two saturation points where the  gradient Is close to zero. This might cause vanishing gradient issues more often than ReLU  or leaky ReLU where the gradient is zero only on one side (i.e., for inputs below 0).  See Slides Week2, “Activation Functions”.  See Slides Week 3, “Saturating Activation Functions”. | |
| --- | --- |

**P2.3 [2 points]** Which of the following statements on Multilayer Perceptrons (MLPs) are true (one or more than one can be true)? Consider an MLP with a single layer. Check the right answer(s).

1. MLPs cannot learn complex input-output mappings.
2. All neurons of the hidden layer are connected to all the inputs.
3. They cannot be used for classification as they can learn linear mapping only.
4. For regression purposes, the cross-entropy loss is normally used.
5. If the activation function is linear, the optimization space is convex.

| **P3** | **Convolutional Neural Networks** |
| --- | --- |

**P3.1 [2 points]** Compute the output **o** of a 1D convolutional network. The input (2 channels) is the following:

\mathbf{X} = \begin{bmatrix} 
-1 & -2 & -3 &-4 & -5 & -6 \\
0 & 1 & 2 &3 & 4 & 5 \\
\end{bmatrix}

The filter is the following matrix:

\mathbf{W} = \begin{bmatrix} 
-1 & -1 & -1  \\
-1 & -1 & -1  \\
\end{bmatrix}

The number of output channels is 1, stride factor=1, dilation factor=1. Remember to add a proper amount of zero-padded elements at the edges of the input. For simplicity (as done in the lecture) do not add the bias term after applying the convolution.

| **o** = [2, 3, 3, 3, 3, 2]^T | |
| --- | --- |

**P3.2 [2 points]** Which of the following statements on Convolutional Neural Networks (CNNs) are true (one or more than one can be true)? Check the right answer(s).

1. The receptive field increases if the kernel size increases.
2. Batch normalization cannot be used with CNNs due to the multiple output channels.
3. Every output channel is computed with a different kernel.
4. CNNs are designed to easily learn long-term dependencies.
5. The stride factor can be used to downsample the input signal.

| **P4** | **Recurrent Neural Networks** |
| --- | --- |

**P4.1 [2 points]** Briefly describe a Vanilla RNN. Write an equation for it and explain all the terms involved. Mention what prevents them from easily learning long-term dependencies.

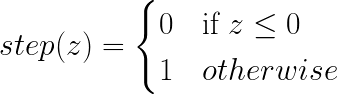
| The simplest RNN is called **Vanilla RNN** (or Elman RNN) is based on the following equation:  \mathbf{h}_t = tanh(\mathbf{W}^{(in)T} \mathbf{x}_t + \mathbf{W}^{(hh)T} \mathbf{h}_{t-1})  We perform a **linear transformation** of both the current input and the previous state. We sum them up and apply a **non-linearity** (Tanh or ReLU).  \mathbf{W}^{(in)} = \begin{bmatrix} w_{0,1}^{(in)} & w_{0,2}^{(in)} & ... & w_{0,M}^{(in)}\\ w_{1,1}^{(in)} & w_{1,2}^{(in)} & ... & w_{1,M}^{(in)}\\ ... & ... & ... & ...\\ w_{D,1}^{(in)} & w_{D,2}^{(in)} & ... & w_{D,M}^{(in)}\\ \end{bmatrix}\mathbf{W}^{(hh)} = \begin{bmatrix} w_{1,1}^{(hh)} & w_{1,2}^{(hh)} & ... & w_{1,M}^{(hh)}\\ w_{2,1}^{(hh)} & w_{2,2}^{(hh)} & ... & w_{2,M}^{(hh)}\\ ... & ... & ... & ...\\ w_{M,1}^{(hh)} & w_{M,2}^{(hh)} & ... & w_{M,M}^{(hh)}\\ \end{bmatrix}  \mathbf{x}_t = [1, x_{t,1}, x_{t,2}, ...,  x_{t,D}]^T\mathbf{h}_t = [h_{t,1}, h_{t,2}, ...,  h_{t,M}]^T  Despite the presence of the recurrent term ht-1, Vanilla RNNs often fail to learn long-term  dependencies. The tanh function applied over long chains of computations indeed causes the  gradient to vanish quickly. Moreover, we don’t have any multiplicative gate that can help  store the hidden state unchanged for an arbitrarily long number of time steps.  See slides Week 4, “Vanilla RNN”, and “Vanishing Gradient”. | |
| --- | --- |

**P4.2 [2 points]** Let’s consider the following recurrent neural network:





Where the step function is defined as



We want this RNN to store unchanged the previous hidden state when the input x is greater or equal than 2. Otherwise, the new hidden state should be equal to the current input. Consider h0=0. Consider only integer inputs.

For instance,

If x is [0, 1, 2, 3, 4, 0, -1, 5, 6, 7]

h should be [0, 1, 1, 1, 1, 0, -1,-1,-1,-1]

What are the values of w0 and w1 that implement the desired functionality?

| **w0 = -**1  **w1 =** 1  **Your computations here:**  Let’s process all the inputs and compute all the h with w0=1 and w1=-1  x=0, z=step(1\*0-1)=0 h=0\*0 + 1\*0=0  x=1, z=step(1\*1-1)=0 h=0\*0 + 1\*1= 1  x=2, z=step(1\*2-1)=1 h=1\*1 + 0\*2= 1  x=3, z=step(1\*3-1)=1 h=1\*1 + 0\*3= 1  x=4, z=step(1\*4-1)=1 h=1\*1 + 0\*4= 1  x=0, z=step(1\*0-1)=0 h=0\*1 + 0\*0= 0  x=-1, z=step(1\*(-1)-1)=0 h=0\*0 + 1\*-1= -1  x=5, z=step(1\*5-1)=1 h=1\*(-1) + 0\*5= -1  x=6, z=1, h=-1  x=7, z=1, h=-1 | |
| --- | --- |

**P4.3 [2 points]** Which of the following statements on Gated Recurrent Units (GRUs) are true (one or more than one can be true)? Consider an MLP with a single layer. Check the right answer(s).

1. GRUs employ an input, an output, and a forget gate.
2. Similarly to Vanilla RNNs, GRUs cannot learn long-term dependencies.
3. The multiplicative gates used in GRUs act as a regularization technique.
4. GRUs simplify the LSTM architecture by using fewer gates.
5. GRUs cannot be stacked on top of convolutional layers due to the multiple channels provided by CNNs.

| **P5** | **Transformers** |
| --- | --- |

**P5.1 [2 points]** Describe the positional embeddings used in Transformer. Explain why the model cannot converge without injecting positional embeddings.

| Transformers use positional embeddings derived from sine and cosine functions of different frequencies:    Where i is the index of an element in the positional embedding vector and pos is the position of the current input in the sequence.  The positional encoding is not a single real number, but a vector of dmodel elements that encode the input position using sinusoids of different frequencies. Given the positional embedding, we can retrieve exactly the position of the input element.  The positional embedding scheme uses sines at even positions and cosines at odd positions. This makes learning relative positioning between elements easier. It can be shown that the network can learn how to attend relative positions (e.g, current position + N) just by learning a linear transformation.  The positional embeddings are summed up with the input ones (no need for concatenation).  The sinusoidal positional embedding scheme used in Transformers can generalize to unseen positions much better than a simple counter. It can also attend relative positioning easier.  See Slide Week 5, “Positional Embeddings” | |
| --- | --- |

**P4.3 [2 points]** Which of the following statements on Scaled Dot Product Attention are true (one or more than one can be true)? Check the right answer(s).

1. It computes a dot product between the keys and the values.
2. It is used in the self-attention blocks of both the encoder and decoder.
3. Normally, multiple attention heads are used to capture different types of dependencies across the elements of the sequence.
4. A mask is applied to the encoded sequence due to the autoregressive nature of the encoder.
5. The dot product ensures the attention mechanism to output values between 0 and 1.

**P4.3 [2 points]** Which of the following statements on Transformers are true (one or more than one can be true)? Check the right answer(s).

1. Transformers are memory efficient even with long input sequences.
2. The output of the decoder can be computed in parallel with the output of the encoder.
3. A mask must be applied to the encoder to prevent the model from attending the first elements of the sequence.
4. At training time, teacher forcing is used in the decoder. This makes it possible to compute the decoder output in parallel for all the time steps of the output sequence.
5. The cross-attention is computed with the queries from the decoder and keys/values from the encoder.

| **P6** | **Speech Processing** |
| --- | --- |

**P6.1 [2 points]** Describe how the filter banks (FBANKs) features are computed. Briefly mention all the steps needed to compute them and depict a mel-filterbank.

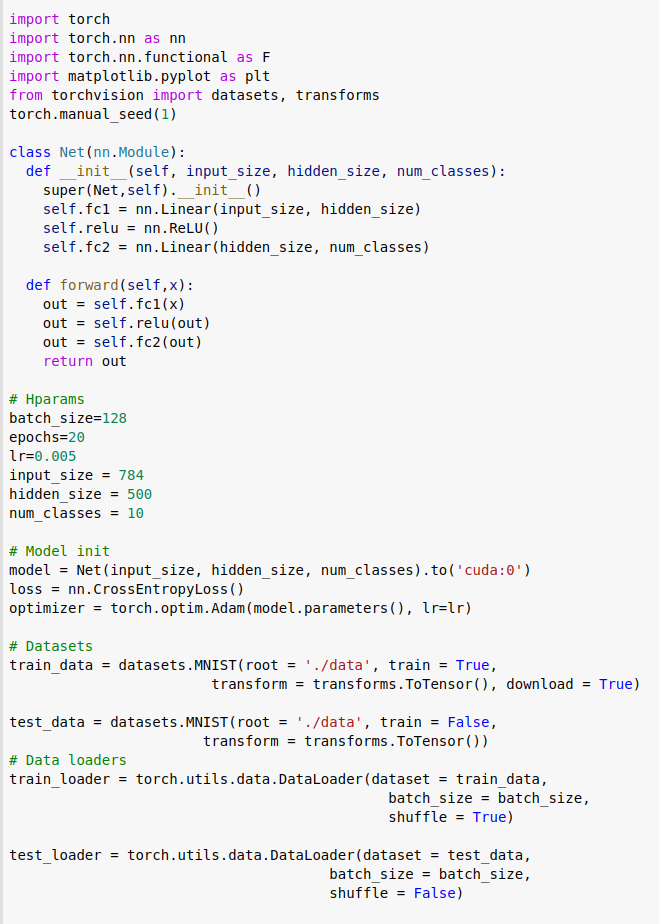
| Filterbanks are popular features used in many speech-processing tasks. To compute them, we  need to do the following steps:  1. Compute the spectrogram (typically with wlen=25ms and hop=10ms).  2. Integrate the spectrogram with a mel filter bank (normally with a triangular shape).  We employ more filters in the lower part of the spectrum and more in the higher part.  The bandwidth of the filters increases with the frequency. This is due to the fact that speech signals  Have more energy in the lower part of the spectrum.    See slides Week 6, “FBANKs” | |
| --- | --- |

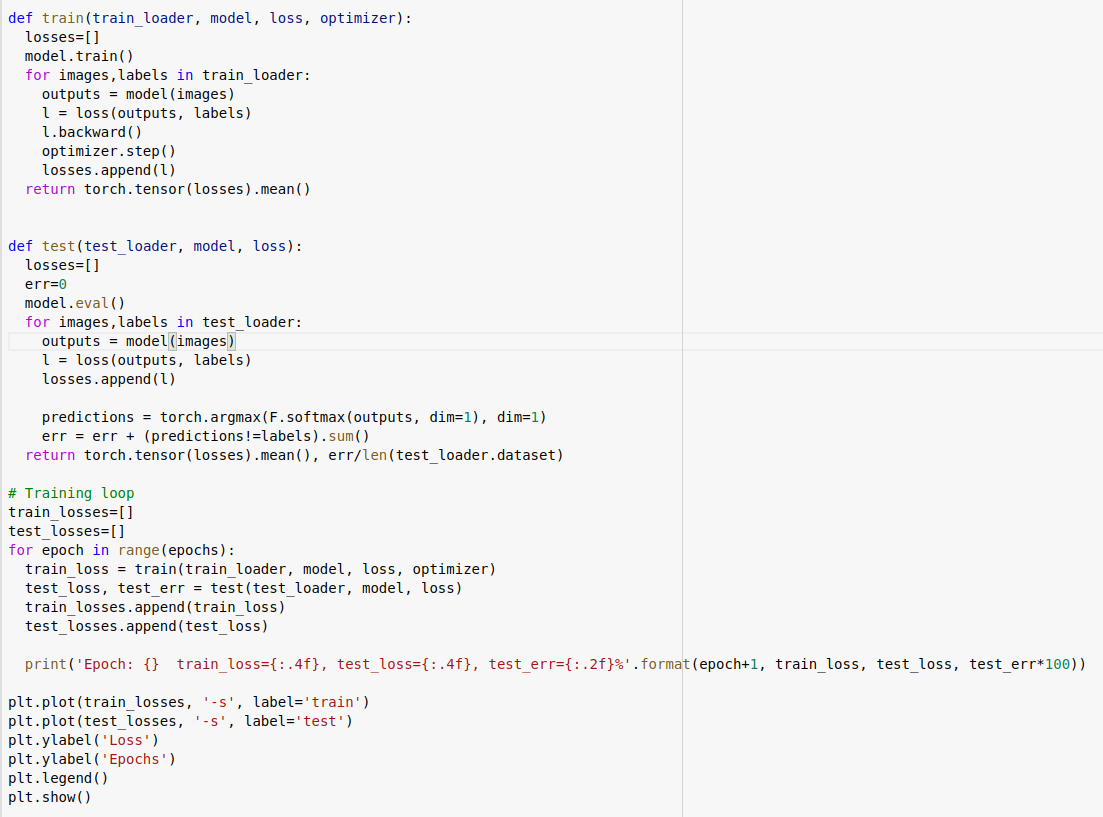
**P4.3 [2 points]** Which of the following statements on self-supervised models for speech feature extraction are true (one or more than one can be true)? Check the right answer(s).

1. These models can be fine-tuned when using them for a supervised classification task.
2. Computing features with these models is way less computationally demanding than computing FBANKs.
3. In self-supervised learning, the label is extracted from the signal itself.
4. The learned features are generally task-specific.
5. Self-supervised models are trained with a large collection of speech signals that are not manually annotated.

**Appendix**

**A.1: MLP for handwritten digit classification using the MNIST dataset**

****

****