Instructor: Alina Vereshchaka

Assignment 2

Autoencoder and Transformer Architectures

Checkpoint: March 27, Thu, 11:59 pm

Due date: April 10, Thu 11:59pm

Welcome to Assignment 2! This assignment focuses on developing both theoretical and practical skills related to autoencoders and transformer networks. It consists of four parts with both theoretical and hands-on tasks where you will derive and analyze network setups, work with various datasets, and implement, train, and adjust different NN architectures. Libraries usage:

- You can use in-built functions or define some functions from scratch.
- All libraries are allowed, except those with predefined models. Those submissions will not be considered for evaluations unless explicitly mentioned.
- The assignment is expected to be fully completed using PyTorch (recommended) or Tensorflow. Mixing is not allowed.
- Download the datasets directly from the links provided. Pre-processed dataset downloads (eg., from PyTorch) aren't allowed unless explicitly mentioned.

Part I: Theoretical Part [20 points]

Part I.I: Autoencoders for Anomaly Detection [10 points]

In manufacturing industries, ensuring the quality of products is important. Detecting anomalies or defects in the production process can be challenging, especially in large-scale operations. Traditional methods often rely on manual inspection or rule-based systems, which can be time-consuming and prone to human error.

Scenario: A manufacturing company "ElectroGuard Innovations" produces electronic components, and they want to improve their quality control process by implementing an automated anomaly detection system. The company collects sensor data from the production line, which includes various parameters such as temperature, pressure, and voltage readings. Anomalies in these readings could indicate potential defects in the components.

Objective: Develop a concept for using autoencoders for anomaly detection in ElectroGuard's manufacturing process.

Autoencoder Architecture:

- Input Layer: 1000 neurons
- Hidden Layer 1: FC, 512 neurons, ReLU
- Bottleneck Layer: FC, 32 neurons

- Hidden Layer 2: FC, 512 neurons, ReLU
- Output Layer: FC, 1000 neurons, Sigmoid
- Loss Function: MSE
- Regularization: L2 Regularization on all weight matrices with a hyperparameter λ

TASKS:

1. Calculate the total number of trainable parameters in the autoencoder, including weights and biases. Show your detailed steps for each layer.
2. Explain how the L2 regularization term will be incorporated into the training process with the MSE loss. Describe its impact on the weight updates during backpropagation.
3. Draw a computational graph (examples: 1, 2, 3) for the autoencoder. Automated computational graph generators are not allowed.
4. Discuss potential challenges and limitations (at least 4) of using autoencoders for anomaly detection in manufacturing. Consider data, anomaly definition, model assumptions, and deployment.
5. Propose potential improvements or extensions (at least 4) to the system to enhance its effectiveness in detecting anomalies. Consider architecture, loss, preprocessing, integration.

Part I.II: Transformers and Self-Attention [10 points]

At the heart of the Transformer architecture lies the self-attention mechanism. It allows models to weigh the importance of different parts of the input sequence when processing each element. In this part, we will check the mathematical operations of the Transformer self-attention mechanism.

TASKS:

1. Explain the mathematical operations in self-attention, processing an input sequence $x = (x_1, x_2, \ldots, x_N)$ through these stages:

    a. **Linear transformations.** Describe how three linear transformations project the input sequence into:

    - **Query (q):** this vector plays the role of "asking questions" about the sequence elements.
    - **Key (k):** this vector helps determine which elements in the sequence are relevant for answering the queries.
    - **Value (v):** this vector contains the actual information from each element in the sequence.
    - Briefly explain how these transformations are typically performed using weight matrices (denoted by W_q, W_k, and W_v) and bias vectors.

b. **Scaled dot-product attention.** Explain how the model calculates attention scores using the query (q_i) and key (k_j) vectors. Include the formula for this step and discuss the role of the square root of the key vector dimension (d_k) in the normalization process.

c. **Weighted sum.** Describe how the calculated attention scores are used to weight the value vectors (v_j). Essentially, this step focuses on the relevant elements in the sequence based on the attention scores. Explain the mathematical formula for this weighted sum.

d. **Optional output transformation:** Explain the purpose of the optional final linear transformation (W_o) and bias term (b_o) in generating the latent representation z. How does this step potentially impact the final representation?

2. Draw the computational graph that depicts the flow of data through the self-attention mechanism. Include all the transformations mentioned in Question 1.

Submission of Part I (Theoretical part):

You may provide the answer using handwritten notes, typed in the MS Word or Latex. Your final pdf for Part I should contain clear, step-by-step derivations for all the requested calculations. Justify each step and clearly mark your final answers.

For the final submission, combine all your answers into a single PDF named:

a2_part_I_TEAMMATE1_TEAMMATE2.pdf

e.g. a2_part_I_avereshc_mjadhav.pdf

Part II: Autoencoders for Anomaly Detection [30 pts]

Implement autoencoder and explore its application for a real-world problem related to anomaly detection. The final model should achieve a test accuracy of greater than 80%.

DATASETS

Below is a list of datasets. Select ONE dataset based on your preference.

- Yahoo S5 Dataset: real server logs from a popular web service, containing various anomalies such as spikes, dips, and shifts in user traffic.
- Hard Drive Test Data: daily snapshot of each operational hard drive
- Numenta Anomaly Benchmark: dataset for evaluating anomaly detection algorithms, with a variety of time-series data from different domains, including financial market data. You can pick one of the data files.

Step 1: Data preparation

1. Select and load one dataset from the list above.

2. Analyze the dataset and provide the following statistics:
   - Number of samples (time points)
   - Number of features
   - Mean, standard deviation, minimum, and maximum values for each feature (or relevant descriptive statistics depending on the data type)
   - Provide a brief description (2-3 sentences) of the dataset: What does it represent? Where does it come from (provide a link)? What are the key variables?
3. Create at least three different visualizations to explore the dataset. Provide a short description explaining what each visualization shows.
4. Identify any missing values (e.g. using pandas.isnull().sum()). Handle any missing values (imputation or removal). Common imputation methods include: forward/backward fill, mean/median imputation, linear interpolation).
5. Preprocess the dataset.
   a. Normalize the data.
   b. [If needed] Address class imbalance in the target column. Possible solutions: oversampling; undersampling; data augmentation techniques for the minority class; assign higher weights to the minority class and lower weights to the majority class, etc.
   c. [If needed] Convert target variable needs to numerical format. You can use one-hot encoding. However, if you use torch.nn.CrossEntropyLoss for your network, it expects class indices (0, 1, 2) directly, not one-hot encoded vectors. Therefore, ensure your labels are integer tensors (e.g., torch.LongTensor).
6. Split the dataset into training, testing and validation sets. You can use train_test_split from scikit-learn.

Step 2: Autoencoder model building

1. Choose and implement one of the following: Autoencoder, Variational Autoencoder (VAE) or other version of Autoencoder for anomaly detection.
2. Experiment with different architectures. Build and train 3 different autoencoder architectures for anomaly detection. Consider experimenting with:
   - Different layer types (Dense, LSTM for time series, Conv1D for sequential data)
   - Number of hidden layers and units
   - Activation functions (ReLU, sigmoid)

   - Print model summary using torchinfo.summary
3. Model training and hyperparameter tuning:
   a. Choose an appropriate loss function and optimizer (e.g., Adam).
   b. Train your model and monitor its performance on the training and validation sets.
   c. Tune hyperparameters (learning rate, batch size, number of epochs, hidden

units, dropout rate) using the validation set.

d. Plot the training and validation loss and accuracy curves over epochs. Analyze the plots for signs of overfitting or underfitting. Adjust your model or hyperparameters if needed.

4. Save the weights of the trained neural network that provides the best results. Check saving and loading models (PyTorch)

Step 3: Evaluation and analysis

1. Evaluate your best model on the test set. Report the following metrics:
   - Training accuracy/loss
   - Validation accuracy/loss
   - Testing accuracy/loss
2. Depending on your chosen dataset, report relevant metrics like:
   - Regression: Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), R-squared (coefficient of determination). Use sklearn.metrics.
   - Classification: Precision, Recall, F1-score.
3. Provide the following plots:
   - Plot training and validation accuracy/loss curves over epochs.
   - Plot the distribution of the reconstruction errors (i.e., differences between input and output data points).
     Calculate the reconstruction errors for the test set. E.g. passing the test data through your encoder and decoder and calculating the loss (e.g., MSE) between the input and output for each test sample. Plot a histogram or kernel density estimate (KDE) of the reconstruction errors. Analyze this distribution. Based on this distribution, you can experiment with setting a threshold on the reconstruction error. Data points with reconstruction errors above the threshold can be classified as anomalies. Explain how you choose your threshold (e.g., based on percentiles of the error distribution, or by visually inspecting the distribution).
   - Use tensorboard to log training and validation metrics (optional)
4. Discuss and analyze:
   - Describe your final RNN/LSTM architecture in detail (number of layers, types of layers, hidden size, dropout rate, etc.).
   - Discuss your results, referencing the metrics and visualizations. Did your model achieve the expected accuracy? What were the challenges? How did hyperparameter tuning affect performance? Are there any patterns in the errors (e.g., consistent under- or over-prediction)?
   - Discuss the strengths and limitations of using autoencoders for anomaly detection.
5. **References**. Include details on all the resources used to complete this part, e.g. links to datasets, research papers or articles, code examples or tutorials you referred.

6. If you are working in a team, provide a contribution summary. We expect equal contribution for the assignment. If the contribution is highly skewed, then the scores of the team members may be scaled w.r.t the contribution.

Team Member	Step#	Contribution (%)

Part III: Build Transformer with PyTorch [25 points]

In this part, we will implement a Transformer model from scratch using the PyTorch framework. We will train the model on a provided dataset, exploring various optimization techniques and hyperparameter tuning to achieve optimal performance. Every component of the Transformer as defined in the paper is expected to be implemented.

The final model should achieve a test accuracy of greater than 80%.

DATASETS

List of Datasets. Select ONE dataset based on your preference.

STEPS:

Step 1: Data Exploration and Preprocessing

1. Select one dataset from the list above.
2. Data exploration:
   - Read, preprocess, and print the main statistics about the dataset
   - Use libraries like matplotlib, seaborn, or plotly to create at least 3 informative visualizations that reveal insights about the data and potential anomalies (e.g., polarity distribution, word count distribution, vocabulary size etc).
3. Text preprocessing:
   - Text cleaning. Remove punctuation, stop words, and unnecessary characters.
   - Text lowercasing. Ensure all text is lowercase for consistent representation.
   - Tokenization. Break down the text into individual words (or tokens). Explore libraries like `nltk` or spaCy for tokenization functionalities. You can also use keras tokenizer or Pytorch tokenizer.
   - Vocabulary building. Create a vocabulary containing all unique tokens encountered in the dataset.

- Numerical representation. Convert tokens into numerical representations using techniques like word embedding (e.g., Word2Vec, GloVe).

Step 2: Model Construction

1. Embeddings and positional encoding: Define an embedding layer to map tokens into numerical vectors. If using pre-trained embeddings, ensure they are compatible with your model's input dimension.
2. Implement the core Transformer architecture:
   - Encoder: Utilize *nn.TransformerEncoder* with multiple *nn.TransformerEncoderLayer* instances. Each layer typically comprises a multi-head self-attention mechanism, a feed-forward layer, and layer normalization.
   - Decoder: Employ *nn.TransformerDecoder* with multiple *nn.TransformerDecoderLayer* instances. These layers incorporate masked self-attention, multi-head attention over the encoder outputs, and a feed-forward layer with layer normalization.
3. Depending on your task (e.g., classification, sequence generation), define an appropriate output layer. For classification tasks, you might use a linear layer with a softmax activation function. Additionally, for classification and regression related tasks, the decoder can be removed completely. Auto-regressive sequence generation requires a decoder.
4. Print model summary using torchinfo.summary
5. Briefly describe the Transformer architecture you have defined.

Step 3: Training the Transformer

1. Preparing for training:
   - Divide the preprocessed data into training, validation, and testing sets using a common split ratio (e.g., 70:15:15 or 80:10:10).
   - Choose an appropriate loss function (e.g., cross-entropy loss for classification) and an optimizer (e.g., Adam) to update model parameters during training.
   - Choose the appropriate attention mask for the task.
1. Define a training loop. E.g. forward pass, calculate loss, backward pass, update parameters. Train the model.

Step 4: Evaluation and Optimization

1. Check your model's performance on the validation set. Monitor metrics like accuracy or loss to track progress. Explore at least 3 optimization techniques to improve the performance of your Transformer model. E.g. regularization (L1/L2), dropout, early stopping, learning rate tuning.
2. Discuss how the optimization techniques helped to improve the performance of the model.

3. Save the weights of the model that provides the best results. Check the saving and loading of models (Pytorch).
4. Discuss the results and provide the following graphs:
    a. Training accuracy, training loss, validation accuracy, validation loss, testing accuracy, and testing loss. Provide your short analysis.
    b. Plot the training and validation accuracy over time (epochs).
    c. Plot the training and validation loss over time (epochs).
    d. Calculate and report other evaluation metrics such as Precision, recall and F1 score. You can use sklearn.metrics.precision_recall_fscore_support.
    e. Plot the ROC curve.
    f. [Optional] Use TensorBoard (or a similar tool, e.g. Wandb) to log the training and validation loss and accuracy over epochs, generate the charts, and attach the SVG images of the charts.
5. **References**. Include details on all the resources used to complete this part, e.g. links to datasets, research papers or articles, code examples or tutorials you referred.
6. If you are working in a team, provide a contribution summary. We expect equal contribution for the assignment. If the contribution is highly skewed, then the scores of the team members may be scaled w.r.t the contribution.

Team Member	Step#	Contribution (%)

Part IV: Summarization using LLMs [25 points]

In this part, we will use a pre-trained large language model (LLM) to perform abstractive summarization. You will fine-tune a pre-trained LLM on these datasets and evaluate its performance using standard summarization metrics such as ROUGE, BLEU, and BERTScore. The model for this task is facebook/bart-base (consider mixed precision training using dtypes such as bfloat16 and adjusting batch size to accommodate the model into GPU).

The following scores are expected over the test sets.

- Billsum expected scores: {Rouge-1: >40, Rouge-2: >18, Rouge-L: >28, BLEU: >12, BERTScore: >75}
- Multinews expected scores: {Rouge-1: >35, Rouge-2: >5, Rouge-L, >13, BLEU: >3.5, BERTScore: >75}

DATASETS

Billsum – summarization of US Congressional and California state bills

Multi-News – news articles and human-written summaries of these articles

Step 1: Dataset preparation and preprocessing

1. Select and load one dataset from the list above. You can download these datasets using Datasets package.
2. Analyze the dataset and provide the main statistics
3. Preprocessing:
    - Tokenize the documents and their summaries using BartTokenizer from https://huggingface.co/facebook/bart-base. You can experiment with another tokenizers.
    - Set appropriate maximum input lengths (e.g., 1024 tokens) and target lengths (e.g., 256 tokens).
    - Additional pre-processing is optional.
4. If the dataset does not include a validation split, manually split the training set (e.g., 90% training, 10% validation).
5. Save the tokenized dataset locally to avoid reprocessing.
6. Briefly describe your preprocessing methodology.

Step 2: Model Fine-Tuning

1. Use the pre-trained model facebook/bart-base from Hugging Face.
2. Training:
    - Fine-tune the model on the tokenized training set.
    - Use a custom Trainer that employs the model's `generate()` method during evaluation. We override the Trainer class from 🤗 with a custom trainer that inherits from this Trainer.
    - Monitor training and validation loss over epochs.
3. Experiment with learning rate, batch size, number of epochs, etc. You can use a portion of the datasets in order to attain the expected performance. Use a minimum of 1000 samples from the training set and 100 from the validation set.
4. Briefly describe your training methodology.

Step 3: Evaluation and analysis

1. Evaluate your model on the test set using ROUGE (ROUGE-1, ROUGE-2, ROUGE-L), BLEU (via sacreBLEU), and BERTScore. Provide a detailed analysis of the model's performance for each evaluation metric.

- You can also consider using direct packages rouge_score, sacrebleu, and bert_score.

2. Include charts of training/validation loss and sample metric scores over the validation data.
3. Discuss any challenges faced (e.g., handling long documents, variability in summary quality, etc.).
4. Propose potential modifications or extensions to enhance summarization quality.
5. **References**. Include details on all the resources used to complete this part, e.g. links to datasets, research papers or articles, code examples or tutorials you referred.
6. If you are working in a team, provide a contribution summary. We expect equal contribution for the assignment. If the contribution is highly skewed, then the scores of the team members may be scaled w.r.t the contribution.

Team Member	Step#	Contribution (%)

Bonus points [max 15 points]

Vision Transformer (ViT) for Image Classification [5 points]

Use a vision transformer to solve Cats and Dogs Dataset. You can use pre-defined ViT model or implement from scratch. Deploy the model. Record a short video (~5 mins) on how it works.

STEPS:

1. Load and preprocess the dataset. This may include resizing images, normalizing pixel values, and splitting the dataset into training, validation, and testing sets.
2. Choose to use a pre-defined ViT model or implement it from scratch. You can use an in-buit predefined models for this part.
3. Train and evaluate your ViT model. Discuss your results.
4. Deploy your trained ViT model. This could be a simple script or application that takes an image as input and predicts whether it's a cat or a dog.
5. Record a short video (~5 mins) demonstrating how your deployed ViT model works. The video should showcase the model taking image inputs and providing predictions. Explain the key aspects of your implementation and deployment process in the video.
    a. Upload the video to UBbox and create a shared link
    b. Add the link at the end of your ipynb file.

6. References. Include details on all the resources used to complete this part.

SUBMISSION:

Submit your ViT implementation as a Jupyter Notebook file named:

a2_bonus_vit_TEAMMATE1_TEAMMATE2.ipynb

Ensure your notebook includes clear comments explaining your code and analysis.

Summarization using LLMs [5 points]

Choose a new dataset from Part IV (one you haven't used before). Repeat the steps from Part IV, and deploy the resulting model.

STEPS:

1. Follow all the steps as in Part IV for a new dataset. Provide brief analysis of the results. You are welcome to reuse your code.
2. Deploy your trained model. This could be a simple app that takes a text as input and returns a summary.
3. Record a short video (~5 mins) demonstrating how it works. Explain the key aspects of your implementation and deployment process in the video.
    a. Upload the video to UBbox and create a shared link
    b. Add the link at the end of your ipynb file.
4. References. Include details on all the resources used to complete this part.

SUBMISSION:

Submit your summarization task as a Jupyter Notebook file named:

a2_bonus_summary_TEAMMATE1_TEAMMATE2.ipynb

Ensure your notebook includes clear comments explaining your code and analysis.

Classification using pre-trained LLMs [5 points]

Use encoder LLMs to complete a classification task. We explore the richness in features derived from pretrained LLMs and hence, we don't finetune the models but rather use the embeddings and train a classifier head (could be non-linear) on top for spam classification.
Dataset: Enron spam

STEPS:

1. Load the enron spam dataset. Use the train/val/test splits and tokenize the text using your pre-trained LLM's tokenizer. Use your best judgement for the relevant input fields.

2. Model Setup – Probing:
    a. Load a pre-trained LLM (e.g., DistilBERT) for sequence classification. Bart is advised as it is an encoder-decoder model and encoder LLMs are suited for classification tasks. Choose a lightweight model that is amenable to your GPU size. Consider using DistilBERT, TinyBERT, MobileBERT, AlBERT or others.
    b. Freeze all base model weights and attach a lightweight MLP (the classification head) that maps the model's representations to binary labels. You may want to create a separate model class that defines these components and a forward function or use out of the box 🤗 classification wrappers.
    c. Use the [CLS] token if available or mean-pooled final hidden states.
3. Configure your training parameters (learning rate, batch size, epochs) and train the model using only the classifier head while the LLM remains frozen.
4. Evaluation and Analysis:
    a. Evaluate the model on the test set using accuracy, precision, recall and F1.
    b. Select two encoder LLMs and compare and discuss any performance trends.
    c. The best model is expected to attain {accuracy: >85%, F1: >85%, Precision: >85%, Recall: >82%}
5. References. Include details on all the resources used to complete this part.
6. Include all details/explanations and references directly in the notebook.

SUBMISSION:

Submit your linear probing implementation as a Jupyter Notebook file named:

a2_bonus_llm_TEAMMATE1_TEAMMATE2.ipynb

Ensure your notebook includes clear comments explaining your code and analysis.

ASSIGNMENT STEPS

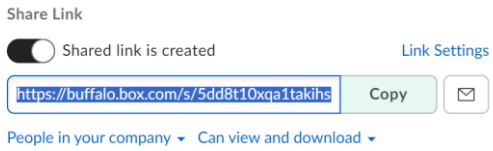
1. Submit checkpoint: Part I.I and Part II (March 27)

- Complete Part I.I: Autoencoders for Anomaly Detection and Part II the assignment.
- Include all the references at the end of each part.
- Your Jupyter notebook should be saved with the outputs.
- Saved weights of the trained networks:
    - Go to UBbox > New > Folder > CSE 676-B Assignment 2 by TEAMMATE 1 & TEAMMATE 2
    - Upload your saved weights into this folder. Include the model weights that

generate the best results for your model, named as a2_part#_TEAMMATE1_ TEAMMATE2.pt

e.g. a2_part2_avereshc_ manasira.pt

- Copy a shared link to the UBbox folder, so it can be viewed by people in your company & it can be viewed and downloaded.



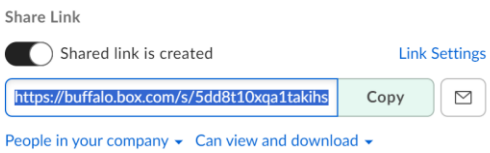
- Add the link to the txt file, named as a2_weights_TEAMMATE1_ TEAMMATE2.txt
- Submit this txt file as part of your submission on UBlearns

- Combine all files in a single zip folder that will be submitted on UBlearns, named as assignment_2_checkpoint_YOUR_UBIT.zip
- Submit to UBLearns > Assignments
- Suggested file structure:

assignment_2_checkpoint_TEAMMATE1_ TEAMMATE2.zip
  - a2_part_1_ TEAMMATE1_ TEAMMATE2.pdf
  - a2_part_2_ TEAMMATE1_ TEAMMATE2.ipynb
  - a2_weights_TEAMMATE1_ TEAMMATE2.txt
  - a2_datasets_TEAMMATE1_ TEAMMATE2.txt

2. Submit final results (April 10)

- Fully complete all parts of the assignment.
- Add all your assignment files in a zip folder including ipynb files for Part II, Part III, Part IV & Bonus part (optional) and txt file with a link to UBbox with links to weights and datasets.
- Datasets:
  - Go to UBbox > CSE 676-B Assignment 2 by TEAMMATE 1 & TEAMMATE 2
  - Upload your chosen datasets into this folder that you used for this assignment.
  - Copy a shared link to the UBbox folder, so it can be viewed by people in your company & it can be viewed and downloaded.



  - Add the link to the txt file, named as

a2_datasets_TEAMMATE1_ TEAMMATE2.txt

- Submit this txt file as part of your submission on UBlearns

- Name zip folder with all the files as assignment_2_final_TEAMMATE1_TEAMMATE2.zip, e.g. assignment_2_final_avereshc_manasira.zip
- Submit to UBLearns > Assignments
- Your Jupyter notebook should be saved with the outputs.
- Include all the references at the end of each part that have been used to complete that part.
- You can make unlimited number of submissions and only the latest will be evaluated
- If you are working in a team, we expect equal contribution for the assignment. Each team member is expected to make a code-related contribution. Provide a contribution summary by each team member in the form of a table below. If the contribution is highly skewed, then the scores of the team members may be scaled w.r.t the contribution.

Team Member	Assignment Part	Contribution (%)

- Suggested file structure (bonus part is optional):

assignment_2_final_TEAMMATE1_ TEAMMATE1.zip
- a2_part_1_ TEAMMATE1_ TEAMMATE2.pdf
- a2_part_2_ TEAMMATE1_ TEAMMATE2.ipynb
- a2_part_3_ TEAMMATE1_ TEAMMATE2.ipynb
- a2_part_4_ TEAMMATE1_ TEAMMATE2.ipynb
- a2_weights_TEAMMATE1_ TEAMMATE2.txt
- a2_datasets_TEAMMATE1_ TEAMMATE2.txt
- a2_bonus_ vit_TEAMMATE1_ TEAMMATE2.ipynb
- a2_bonus_summary_TEAMMATE1_ TEAMMATE2.ipynb
- a2_bonus_llm_TEAMMATE1_ TEAMMATE2.ipynb

Notes:

- Ensure your code is well-organized and includes comments explaining key functions and attributes. Also ensure that all the section headings for your solutions corresponding to this assignment are displayed. After running the Jupyter Notebooks, all results and plots used in your report should be generated and clearly displayed.

- You can submit multiple files, but they all need to be labeled with a clear name.
- Recheck the submitted files, e.g. download and open them, once submitted and verify that they open correctly
- Large files: any individual file, except .ipynb file, that is larger then 10MB (e.g. your model weights or datasets) should be uploaded to UBbox and you have to provide a link to them. A penalty of -10pts will be applied towards the assignment if there is a file submitted that is bigger than 10MB.
- It's ok if your final combined zip folder is larger then 10MB.
- The zip folder should include all relevant files, clearly named as specified.
- Only files uploaded on UBlearns and a submitted link to UBbox for saved weights are considered for evaluation.

Academic Integrity

The standing policy of the Department is that all students involved in any academic integrity violation (e.g., plagiarism in any way, shape, or form) will receive an F grade for the course. The catalog describes plagiarism as "Copying or receiving material from any source and submitting that material as one's own, without acknowledging and citing the particular debts to the source, or in any other manner representing the work of another as one's own.". Refer to the Office of Academic Integrity for more details.

Important Information

This assignment can be done individually or in a team of up to two students. The grading rubrics are the same for a team of any size.

- No collaboration, cheating, and plagiarism is allowed in assignments, quizzes, the midterms or final project.
- All the submissions will be checked using MOSS as well as other tools. MOSS is based on the submitted works for the past semesters as well the current submissions. We can see all the sources, so you don't need to worry if there is a high similarity with your Checkpoint submission.
- The submissions should include all the references. Kindly note that referencing the source does not mean you can copy/paste it fully and submit as your original work. Updating the hyperparameters or modifying the existing code is a subject to plagiarism. Your work has to be original. If you have any doubts, send a private post on piazza to confirm.
- All parties involved in any suspicious cases will be officially reported using the Academic Dishonesty Report form. Please refer to the Academic Integrity Policy for more details.

Late Days Policy

You can use up to 5 late days throughout the course that can be applied to any assignment-related due dates. You do not have to inform the instructor, as the late submission will be tracked in UBlearns.

Important Dates

March 27, Thu, 11:59 pm - Checkpoint Submission is Due

April 10, Thu, 11:59 pm - Final Submission is Due