ARI 510 Lab 3: Representation Learning and Multimodal Fusion

University of Michigan-Flint

Fall 2024

(See course Canvas page for due date)

1 Overview

In this lab, you will work with the **Hateful Memes Dataset** from Facebook AI to develop and compare models for text-only, image-only, and multimodal (text + image) classification tasks. The dataset includes memes consisting of both images and embedded text, and the goal is to classify them as either *hateful* or *non-hateful*. You will gain experience training machine learning classifiers that operate on text inputs, image inputs, and a combination (fusion) of both. The lab only requires you to apply very simple approaches to doing this (while still following best practices) and you aren't expected to find the best possible multimodal classification model for the task for this assignment.

Important: The dataset we are working with is fairly large (especially since it contains images). If you are working in Google Colab, keep in mind that you will need to re-download the data any time the session ends due to inactivity. One strategy would be to start with a tiny subset of the data while you are developing your code and only run everything on the full dataset when you are convinced that everything is working correctly. There are other alternatives you might turn to for computational resources listed in section 6.2.

Disclaimer: There are code snippets throughout to help you get an idea about how the various steps work. These are not intended to solve the entire assignment for you and are provided "as is". There are many other ways to complete each step and you *do not need to use the provided code snippets.*

2 Getting Started

To download the dataset, you can use the following commands. This dataset uses Git Large File Storage (LFS), so ensure you have LFS installed.

```
git lfs install
git clone https://huggingface.co/datasets/neuralcatcher/
    hateful_memes
```

The dataset is organized into `train`, `dev`, and `test` splits. You can ignore the `unseen` split for this assignment. Now let's begin with the steps for building and training the models.

3 Part 1: Text-Only Classification

The first task is to build a text-only model that uses the provided meme text to classify whether a meme is hateful or not.

3.1 Step 1: Generating Text Embeddings

First, let's use a pretrained language model (BERT) to generate a single representation of our text inputs. We will tokenize the text, run it through BERT, and extract the sentence embeddings using the hidden state of the [CLS] token. Here's an example of how that works, from loading the pretrained BERT model to passing through some sentences.

```python
import torch
from transformers import AutoTokenizer, AutoModel

# Load BERT tokenizer and model from huggingface
model_name = 'bert-base-uncased'
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModel.from_pretrained(model_name)

# Set the model to evaluation mode since we won't train it
model.eval()

# Example text
texts = ["This is the first sentence.", "This is the second
    sentence."]

# Tokenize and encode the text inputs
inputs = tokenizer(texts, padding=True, truncation=True, max_length
    =128, return_tensors='pt')

# Pass the inputs through the BERT model
with torch.no_grad():
    outputs = model(**inputs)

# Extract the sentence embeddings (here we use the embedding of the
     special [CLS] token)
sentence_embeddings = outputs.last_hidden_state[:, 0, :]  # shape:
    [batch_size, hidden_state_size]
```

At this point, you might want to save the outputs so that you don't need to repeat the BERT embedding generation step during future runs. Then you can just load them during the next step. You might also want to check the shape of your output to make sure it's what you expected. For your reference, BERT uses a hidden state size of 768.

3.2 Step 2: Building the Classifier

Now that you have the text embeddings, let's build a simple feedforward neural network using pytorch that takes these embeddings as input and predicts the target class (hateful or not hateful).

```
import torch.nn as nn

# Define a simple fully connected neural network
class TextClassifier(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(TextClassifier, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        # this is our activation fuction between the layers
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_dim, output_dim)

    # show what a forward pass through the network should look like
    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

# Initialize the model
input_dim = TODO  # input embedding size (should match the size of
    your embeddings)
hidden_dim = TODO  #  hidden layer size
output_dim = 1  # Binary classification (hateful or not)

# create the untrained model based on this architecture
model = TextClassifier(input_dim, hidden_dim, output_dim)
```

3.3 Step 3: Training and Evaluation

Next, train the classifier using the binary cross-entropy loss function and evaluate the performance using accuracy, precision, recall, F1-score, and (let's add a new one) AUC-ROC. Use the development set only for now.

```
# Define the loss function and optimizer
criterion = nn.BCEWithLogitsLoss()  # Binary Cross-Entropy Loss
    with logits
# Adam is a widely used optimizer, often works better than plain
    old SGD
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

# Example training loop
num_epochs = TODO  # Define the number of epochs (passes through
    training set)
labels = torch.tensor([0, 1], dtype=torch.float32).unsqueeze(1)  #
    Example binary labels (0 or 1) using one-hot encoding,
    unsqueeze(1) just to rotate it for this example

# set to training mode (track the parameter updates needed)
model.train()
for epoch in range(num_epochs):
    # NOTE: this assumes the entire dataset fits into memory
    # in practice, you will need to split into batches
    # (look into pytorch dataloaders, these will help you do this)
    # essentially, load small batches of data at a time and update
    the weights

    optimizer.zero_grad()  # Reset gradients
    outputs = model(sentence_embeddings)  # Forward pass
    loss = criterion(outputs, labels)  # Compute the loss
    loss.backward()  # Backward pass
    optimizer.step()  # Update the weights

    # print the loss each time (can add more fine-grained tracking
    during the training process if you like, too)
    print(f"Epoch {epoch + 1}/{num_epochs}, Loss: {loss.item()}")

# Evaluation on the dev set (may want to do this after each epoch)
from sklearn.metrics import accuracy_score, precision_score,
    recall_score, f1_score, roc_auc_score

# make sure to stop tracking gradients for these inputs
model.eval()
with torch.no_grad():
    dev_outputs = model(sentence_embeddings)  # Example dev set
    outputs
    dev_preds = torch.sigmoid(dev_outputs).round()  # Apply sigmoid
     to convert to probabilities and round

    # Example evaluation metrics
    accuracy = accuracy_score(labels, dev_preds)
    precision = precision_score(labels, dev_preds)
    recall = recall_score(labels, dev_preds)
    f1 = f1_score(labels, dev_preds)
    auc_roc = roc_auc_score(labels, torch.sigmoid(dev_outputs))

    print(f"Accuracy: {accuracy}")
    print(f"Precision: {precision}")
    print(f"Recall: {recall}")
    print(f"F1-Score: {f1}")
    print(f"AUC-ROC: {auc_roc}")
```

By the end of this step, you should have a trained text-only classifier that can predict whether a meme is hateful or not based solely on its text content. Note that you can save a trained model if you want to be able to load it later without needed to go through the training process again (this might be helpful since you will need to use this model on the test set later).

4 Part 2: Image-Only Classification

Now you will build a model that uses only the images from the memes to perform classification.

4.1 Step 1: Generating image embeddings

You can use `torchvision` to load and preprocess images (e.g., resizing, normalization).

```
from torchvision import transforms
from PIL import Image

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    # these are the normalization values for resnet training
    dataset
    # when using resnet it's standard practice to normalize this
    way
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
    0.224, 0.225])
])

image = Image.open('path_to_image.jpg')
image = transform(image)
```

Then, let's use a pretrained ResNet to extract image features.

```
import torchvision.models as models

# Add a batch dimension (as ResNet expects a batch of images)
image = image.unsqueeze(0)  # shape: [1, 3, 224, 224]

# Load pretrained ResNet
model = models.resnet18(pretrained=True)

# Remove the final classification layer to get the raw feature
    vector (embedding)
resnet_feature_extractor = nn.Sequential(*list(model.children())
    [:-1])  # Remove last layer

# get the image representation we could use as features for our
    classifier later
with torch.no_grad():
    image_representation = resnet_feature_extractor(image)

# Reshape the output from [1, 512, 1, 1] to [1, 512]
image_representation = image_representation.view(
    image_representation.size(0), -1)
```

Again, you may want to save your image vectors for the dataset separately so you can load them later.

4.2 Step 2: Building the Classifier (again)

You should be able to follow the steps from section 3.1, but this time your input size will be based on the ResNet outputs instead of BERT's.

4.3 Step 3: Training and Evaluation

Train the model using images and evaluate it in the same way you did with the text data. The main difference here is the input representation that you are using for reach data point (image embedding instead of text embedding).

5 Part 3: Multimodal Classification

In this part, you will combine both text and image information to build a multimodal model! We will explore two fusion strategies: **early fusion** and **late fusion**.

5.1 Early Fusion (Feature-Level Fusion)

5.1.1 Step 1: Extract Text and Image Features

First, use the BERT model from Part 1 to extract text features and the ResNet model from Part 2 to extract image features. If you saved the features during those steps, you can just load them here.

5.1.2 Step 2: Concatenate Features and Train

Concatenate the image and text features, and pass them through a new fully-connected neural network to classify the meme.

```
# Concatenate text and image features
combined_features = torch.cat((text_features, image_features), dim
    =1)
```

You only need to train the fully-connected network part. The input size should now but the sum of the two input sizes from your previous NNs.

5.2 Late Fusion (Decision-Level Fusion)

In late fusion, you will first make predictions using the image and text models separately, then combine the predictions.

```
# Get predictions from text model and image model
text_logits = text_model(text_input)
image_logits = image_model(image_input)

# Combine predictions (e.g., averaging)
combined_logits = (text_logits + image_logits) / 2
```

Then use the combined logits to get your predicted class labe instead of the individual ones. Then, think about how you can wrap this entire process in a training loop so you can update the parameters of the text and image models based on the loss when making predictions using the combined logits. Train this combined network and compute the evaluation metrics one last time.

6 Part 4: Comparison and Results

Finally, compare the performance of all four approaches on the test set:

- Text-only model.
- Image-only model.
- Multimodal model with early fusion.
- Multimodal model with late fusion.

You should report the metrics like accuracy, precision, recall, F1-score, and AUC-ROC for each approach and present them in a table.

6.1 Deliverables

You should submit:

1. your written report (PDF)
2. the code you used for your experiments

6.2 Requirements

You will be assessed based on the completion of these requirements.

1. Written Report contains the following sections:
   (a) **Overview:** the main steps you completed during this assignment and your general approach to completing the steps.
   (b) **Results**: present your results (using a table) from your various models on the development set, as well as your final results on the test set
   (c) **Discussion**: what can we learn about the importance of text and image based features for the hateful memes dataset? What are the main things you learned about multimodal machine learning from doing this assignment? What challenges did you face during this assignment and how did you overcome them?
   (d) [if you used chatgpt, copilot, etc.] **Generative AI Statement**: short statement about how you used generative AI and your observations about how effective it was.
2. Code:
   (a) make sure your code is readable and documented
   (b) you may either submit a file (.py or ipynb file) directly or link to a github repository containing your code

7 Computational Resources

We have reached the point of the class where your own laptop or the free tier of Google Colab might become more difficult to work with. Therefore there are a couple of other options you may want to consider: Google Cloud and the Umich Great Lakes Cluster. We should have around $50 free GCP credits per student for this course that Prof. Wilson will share when that is confirmed. Further, each student has around $60 of credits for this course on the Great Lakes Cluster under the allocation named `ar510f24-class`. Keep in mind that the limits listed here will apply. You will need to create an HPC account and follow the instructions here to submit jobs to the cluster.

These resources may be useful for your project other other assignments later in the course, so try to budget your usage with that in mind.