HOMEWORK 0
PYTORCH PRIMER *

10-423/10-623 GENERATIVE AI
http://423.mlcourse.org

OUT: Jan. 18, 2024
DUE: Jan. 24, 2024
TAs: Haoyang, Jing, Qin, Ifigeneia, and Tiancheng

Instructions

- **Collaboration Policy**: Please read the collaboration policy in the syllabus.
- **Late Submission Policy:** See the late submission policy in the syllabus.
- **Submitting your work:** You will use Gradescope to submit answers to all questions and code.
  - **Written:** You will submit your completed homework as a PDF to Gradescope. Please use the provided template. Submissions can be handwritten, but must be clearly legible; otherwise, you will not be awarded marks. Alternatively, submissions can be written in LaTeX. Each answer should be within the box provided. If you do not follow the template or your submission is misaligned, your assignment may not be graded correctly by our AI assisted grader.
  - **Programming:** You will submit your code for programming questions to Gradescope. There is no autograder. We will examine your code by hand and may award marks for its submission.
- **Materials:** The data that you will need in order to complete this assignment is posted along with the writeup and template on the course website.

Question	Points
Background Reading	2
Image Classification	41
Text Classification	22
Code Upload	0
Collaboration Questions	2
Total:	67

*Compiled on Thursday 25th January, 2024 at 02:43

Introduction

In this assignment, you will choose-your-own-adventure as you get up to speed on (or do a quick review of) PyTorch and Weights & Biases.[1]

PyTorch is a general purpose deep learning library. It allows you to define a computation graph, `loss`, dynamically in Python, and then a simple call to `loss.backward()` computes all the adjoints (aka. gradients of the loss with respect to each parameter) for you. Gone are the days in which we needed to work through complicated matrix calculus just to train our models. Well of course, you'd very much need to do that for any function that isn't easily or efficiently expressed in PyTorch, but those cases are becoming less and less common.

Weights & Biases is a logging tool that allows you to easily track the behavior of your model during training and evaluation. As well, once you've logged the interesting bits of data (say, the validation loss every 10 epochs), you can easily create a plot with just a few clicks showing that information. There is another advantage: Each run of your code might be with different hyperparameters and, if you carefully log these as well, then with a few more clicks you can compare the model's behavior across different hyperparameter settings.

At a high-level, you will proceed as follows:

1. Read the PyTorch tutorial.
2. Read the Weights and Biases (wandb) tutorial.
3. Review the HW0 starter code. You'll find it closely mirrors the code described in the PyTorch tutorial.
4. Modify the starter code so that it incorporates Weights & Biases logging.
5. Run the requested experiments and report your results as tables/plots from the wandb interface.
6. Modify your code further so that it supports a different model (you will choose the model!).
7. Allow your code to choose a different optimizer (you will choose the optimizer!).
8. Run additional experiments in order to better understand PyTorch.

You will carry out these tasks on two applications: image classification and text classification.

[1] Although all students in this class have taken an Introduction to Machine Learning course before, some of those (even here at CMU) did cover PyTorch and others did not. We want to ensure that everyone here is ready to start HW1 at the same level.

Computing Environment

First you need to setup your computing environment. Below we outline how you could do so on your laptop, or on Google Colab.

Local Environment

To use PyTorch on your laptop, we recommend the following setup.

1. Follow the instructions linked below to install Python using MiniConda.

   https://docs.conda.io/projects/miniconda/en/latest/miniconda-install.html

   Then create and activate a new python environment. For example:

   ```
   conda create -n py31 python=3.11
   conda activate py31
   ```

2. Next follow the instructions from PyTorch on how to install locally by selecting "Conda" for the "Package" options. For most laptops, you would select "CPU" or "Default" as the "Compute Platform". https://pytorch.org/get-started/locally/

3. Install Weights & Biases https://docs.wandb.ai/quickstart

   ```
   pip install wandb
   ```

4. Install any other Python packages you may need with conda when possible, and pip otherwise.

   ```
   conda install <package>
   pip install <package>
   ```

Colab

Google Colab provides free easy access to some amount of GPU compute. On the free tier, your GPU jobs will time out after a fixed number of hours and you may be temporarily unable to use a GPU if you use too many hours. The limits are dynamic and not clearly documented.

There are two ways to use Colab:

1. **As a Jupyter Notebook:** To see an example of PyTorch in Colab, click the *Run in Google Colab* link from the tutorial: https://pytorch.org/tutorials/beginner/basics/intro.html

2. **As a Terminal:** You can also treat Google Colab as a VM and run code as you would at the terminal. You should first put your code and data in a Google Drive folder. Then create a Code cell with the following snippet and run it to mount your Google Drive folder.

   ```
   from google.colab import drive
   drive.mount('/content/drive')
   ```

   Now when you open the *Files* window on the left, you'll be able to view `drive/MyDrive` which contains all your Google Drive files. You can run terminal commands by prefixing the command with an exclamation point in a cell. For example, if your code `helloworld.py` is in `mycode/`, then you can run:

   ```
   !pwd
   !cd /content/drive/MyDrive/mycode
   ```

```
!pwd
!python helloworld.py
```

Whether you're using Colab as a notebook or a terminal, if you want to use a GPU, you must set the Runtime to use a GPU via *Runtime* $\rightarrow$ *Change runtime type* $\rightarrow$ *T4 GPU*. Better GPUs are available by upgrading to Colab Pro.

1 Background Reading (2 points)

This assignment is *primarily* a reading assignment. You will read both the starter code and various tutorials.

Complete the following readings before you begin.

- PyTorch Tutorial. Please read the full collection of the Introduction to PyTorch, i.e. Learn the Basics || Quickstart || Tensors || Datasets & DataLoaders || Transforms || Build Model || Autograd || Optimization || Save & Load Model.

  https://pytorch.org/tutorials/beginner/basics/intro.html

- Weights & Biases Tutorial. Please read the Quickstart.

  https://docs.wandb.ai/quickstart

1.1. (1 point) Did you read the PyTorch tutorial? If you already read this or an equivalent reading for another course, you may answer 'yes' here.

Yes

1.2. (1 point) Did you read the Weights & Biases Quickstart tutorial? If you already read this or an equivalent reading for another course, you may answer 'yes' here.

Yes

Figure 1: Examples images from Parrot/Narwhal/Axolotl dataset.

2 Image Classification (41 points)

In this section, you will augment an image classifier written in PyTorch.

Dataset

The image classification dataset is hot off the press: Each training example consists of an image created by a text-to-image model and is labeled as one of {parrot, narwhal, axolotl}. The dataset consists of:

`./data/img_train.csv` The training dataset. Each row corresponds to a training example. There are four columns: 1. The `image` column provides the relative path to the `.jpg` file (e.g. `./data/parrot/parrot-0.jpg`). 2. The `prompt` column contains the prompt that was fed into the text-to-image model to generate the image. 3. The `label` column is the human readable label (e.g. `narwhal`). 4. The `label_idx` column contains an integer representation of the label (i.e. 0, 1, or 2).

`./data/img_val.csv` The identically formatted validation dataset.

`./data/img_test.csv` The identically formatted test dataset.

`./data/parrot/ ./data/narwhal/ ./data/axolotl/` The three directories containing the respectively labeled raw images.

Examples of the images are shown in Figure 1. The data is split roughly so that the training data contains 70%, the validation data 15%, and the test data 15%.

Starter Code

The starter code in `img_classifier.py` is a fully functional (albeit simple) image classifier based on the PyTorch tutorial.

Data: The primary changes are to accommodate the Parrot/Narwhal/Axolotl dataset, intead of FashionMNIST. To accomplish this, we provide the class `CsvImageDataset` which is a `torch.utils.data.Dataset`. This class reads in the dataset from a given CSV, optionally applying some transformations to the image. The transformations we use in `get_data()` are fairly standard: first we resize so that the smallest side of the image is 256 pixels; next we crop out the centermost 256x256 pixels, then we normalize the red/green/blue channels using the mean and standard deviation from ImageNet.

Model: The model in `NeuralNetwork` is a simple feed forward neural network with ReLU activations. Its input layer takes the 256x256x3 numbers representing an image as features and maps to a first hidden layer of 512 units. The second hidden layer is also 512 units. The output layer is just 3 units, i.e. equal to the number of labels.

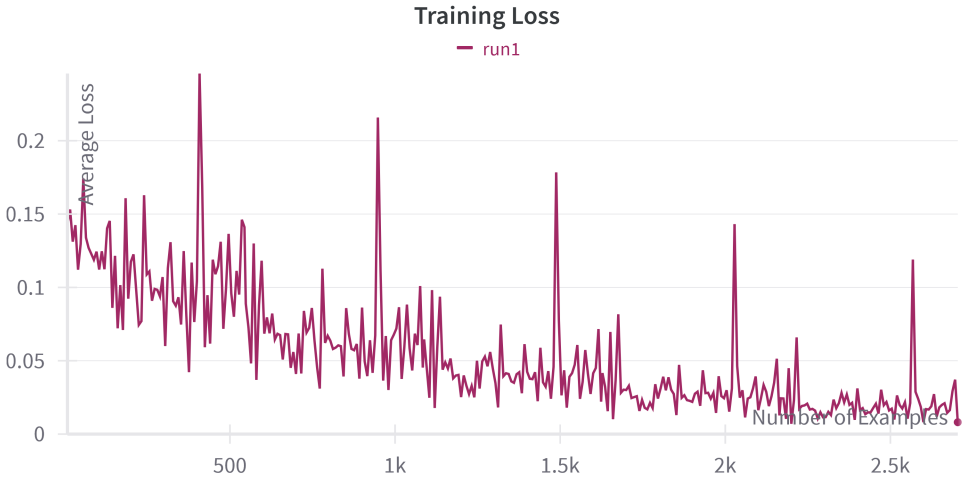
Training: Notice that our neural network does not explicitly define a `nn.Softmax` layer. The `nn.CrossEntropyLoss` works directly with the logits instead since the probabilities coming out of a softmax might be rather small. During training, we optimize the loss using the stochastic gradient descent algorithm, `optim.SGD`.

Evaluation: At the end of each epoch, the model is evaluated on the train. After training it is evaluated on the test dataset. Afterwards the model is saved to disk (and loaded by way of example) in case you wish to resume training from the saved state, or make predictions with the model on another dataset.

Experiments

Instrument the existing code with `wandb`. First, use `wand.init()` to set the project name and store hyperparameters. Next use `wandb.log()` to track the loss of each batch and the current number of examples for which we've computed a gradient. Then, in a single call to `wandb.log()` at the end of each epoch, track the train accuracy, train loss, test accuracy, test loss, and the current epoch number.

2.1. (3 points) Run the image classifier with the default hyperparameters. Using `wandb`, plot the training loss of each batch vs. the number of examples seen so far by the optimizer. The loss should be the average loss, taking the average within each batch.



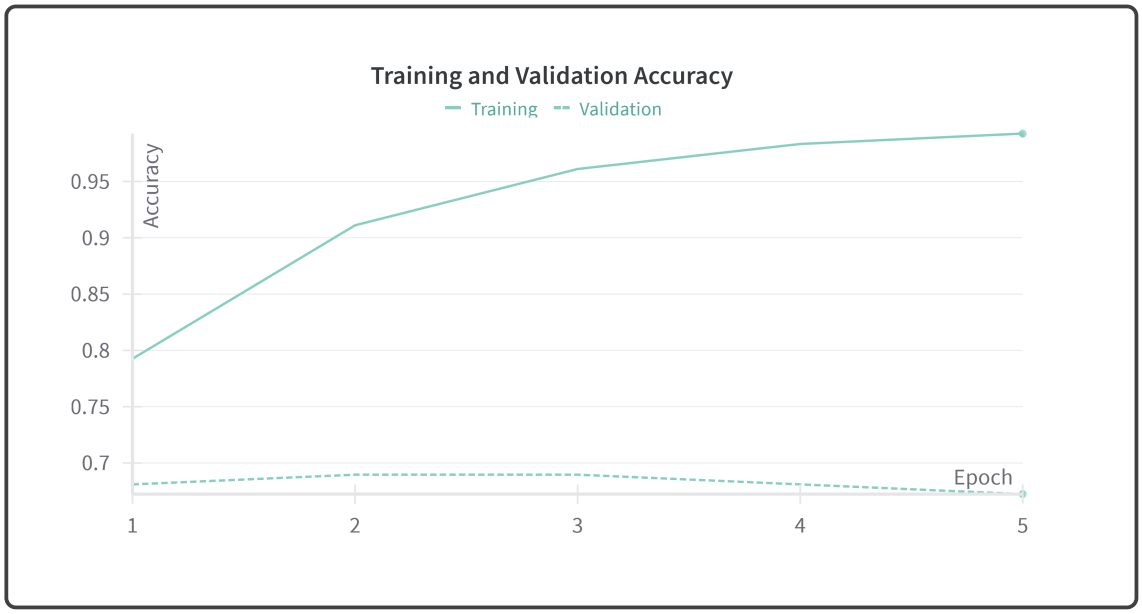
2.2. (3 points) Report the final train/test accuracy, and the final train/test loss. (You do not need to use `wandb` to create this table.)

Dataset	Accuracy	Loss
Train	99.1%	0.01586%
Test	60.9%	0.116626%

The dataset contains a validation set, but the starter code does not use it. Augment `img_classifier.py` so that the validation accuracy/loss are computed at the end of each epoch and reported to `wandb`.

2.3. (4 points) Plot the train loss and the validation loss on the same plot with the number of epochs on the horizontal axis. The train loss should be averaged over the entire training dataset, and do likewise for the validation loss.

2.4. (4 points) Plot the train accuracy and validation accuracy on the same plot with the number of epochs on the horizontal axis.



2.5. Surprisingly, this simple model does seem to be able to learn to distinguish (at least some of) the parrot/narwhal/axolotl images. Perhaps the reason for this success is that the colors of the animals tend to be quite different. Change the sequence of transforms in `transform_img = T.Compose([...])` so that it converts the image to a single channel grayscale image using `torchvision.transforms.Grayscale`. Be sure to adjust your model to accommodate the new input size! (Revert back to using color images after this question.)

(a) (3 points) What is the test accuracy with color images vs. grayscale images?

Dataset	Image Type	Accuracy
Test	Color	60.9%
Test	Grayscale	53.9%

(b) (2 points) Does this support the hypothesis that the classifier wasn't really learning anything besides the color of the animals?

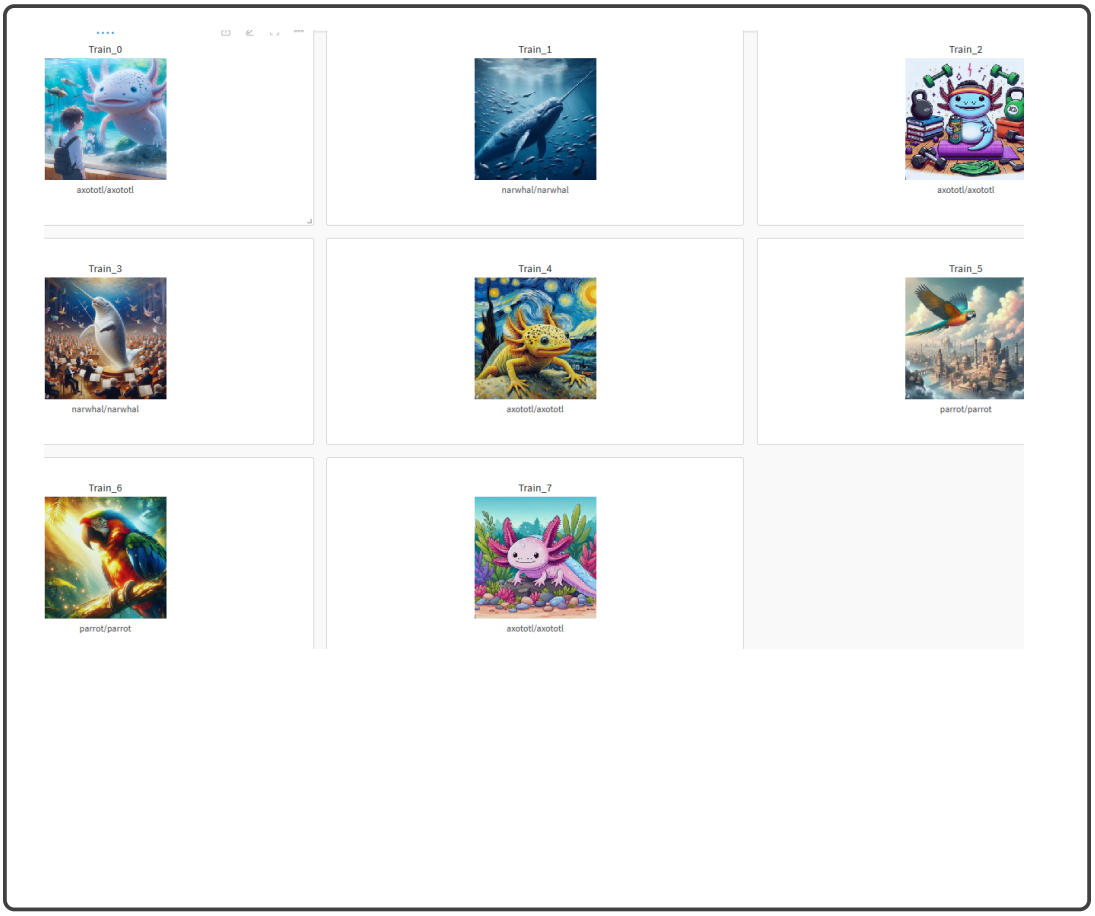
No because if the model was only learning the colors, then removing the colors should drop the accuracy to 33% (random) when color is removed.

2.6. (2 points) Next consider what happens if we work with really tiny images. Resize each image to (28×28) (the standard size of an MNIST digit). What is the test accuracy in this case? (Revert back to resizing images to (256×256) after this question.)

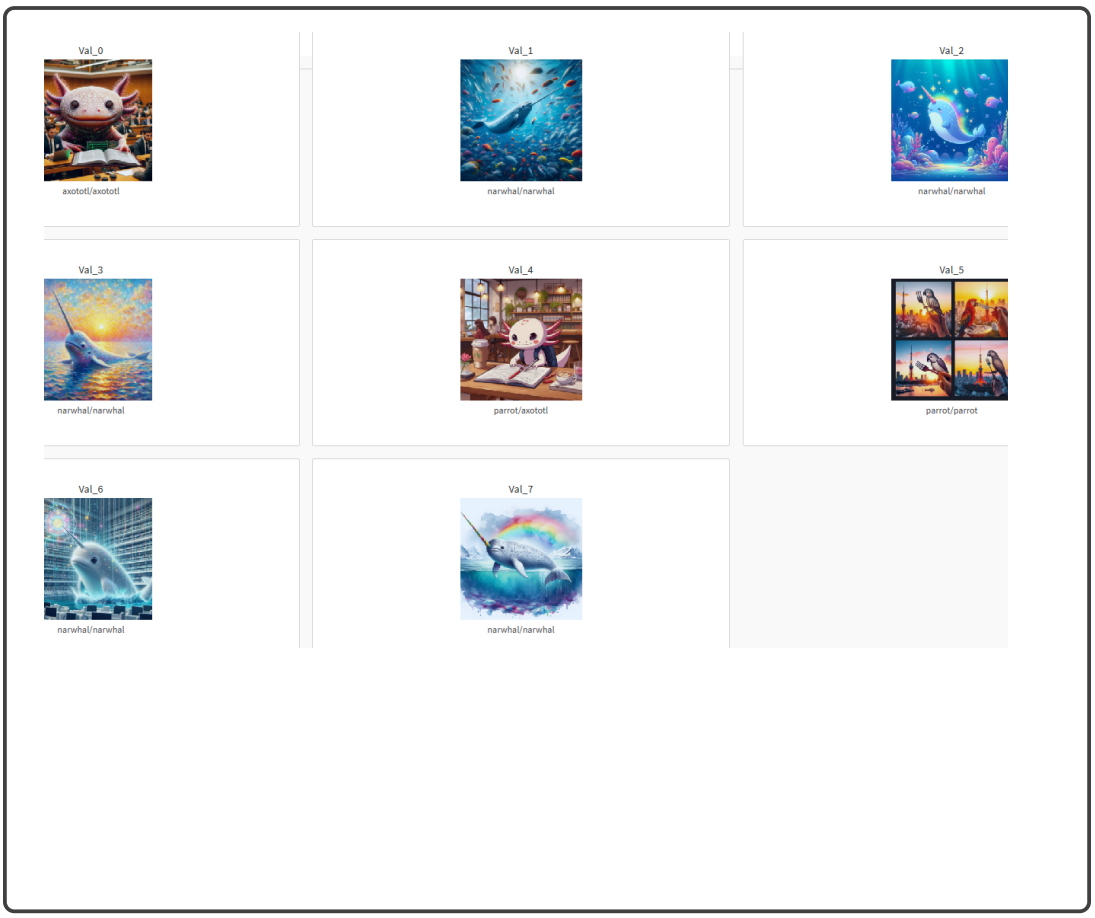
The test accuracy is 30.4%.

2.7. For the first batch of each of the datasets (train, val, test), on the last epoch only, log each of the images with a caption containing its predicted label as a string (i.e. parrot, narwhal, axolotl) and its true label as a string. Format the caption as "`<predicted label> / <true label>`".

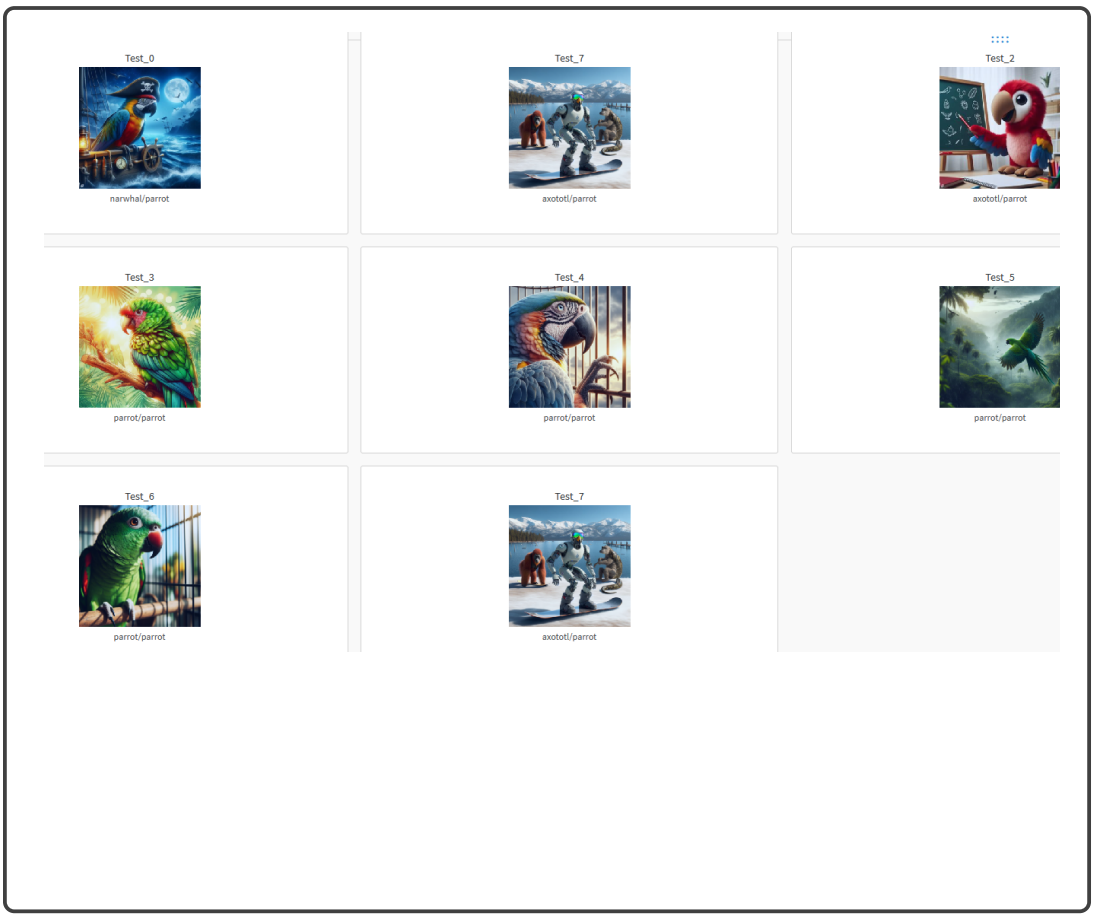
(a) (2 points) Show the first batch of the training dataset captioned appropriately on wandb.


Train_0
axotot/axotot
Train_1
narwhal/narwhal
Train_2
axotot/axotot
Train_3
narwhal/narwhal
Train_4
axotot/axotot
Train_5
parrot/parrot
Train_6
parrot/parrot
Train_7
axotot/axotot

(b) (2 points) Show the first batch of the validation dataset captioned appropriately on wandb.



(c) (2 points) Show the first batch of the test dataset captioned appropriately on wandb.



(d) (2 points) Pick a few of the images on which the model made an error, and try to explain why it might have had difficult with those images. Answer in just a few sentences.

2.8. Our current optimizer is stochastic gradient descent (`torch.optim.SGD`). Change to some

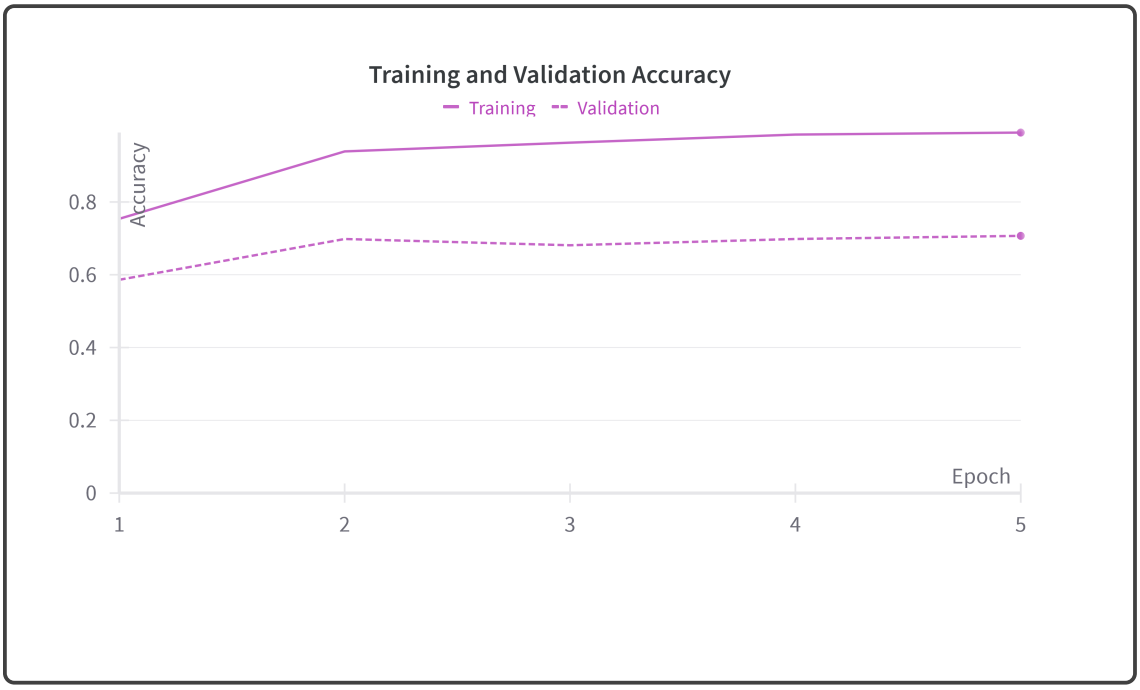
other optimizer from `torch.optim`[2] and adjust the hyperparameters to see if you can achieve better performance than SGD. (If you don't get better performance, you can still receive full credit here.)

(a) (2 points) Which optimizer did you choose and what hyperparameters did you select?

Adagrad with learning rate = 1e-3

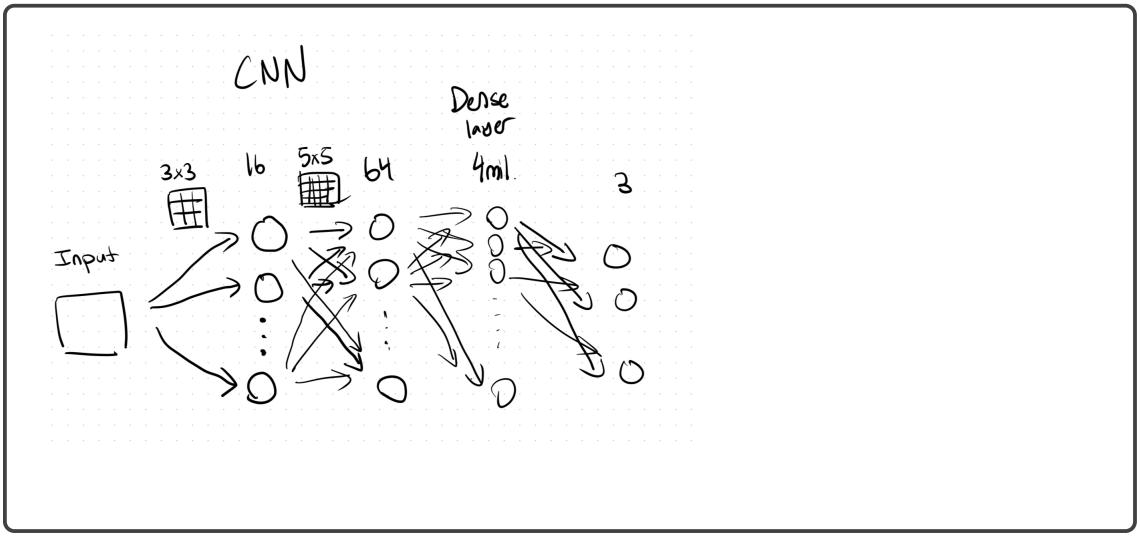
[2] https://pytorch.org/docs/stable/optim.html

(b) (3 points) Then plot the train accuracy and validation accuracy on the same plot with the number of epochs on the horizontal axis. Include a run with SGD and with the new optimizer.



2.9. The model we're using is incredibly simple: a feed-forward neural network with two hidden layers of size 512 units, and ReLU activations. Define a new model that is different in some interesting way. You should review the documentation of `torch.nn`[3] and select some different modules and put them together in some interesting way. The only requirement is that you do *not* use `nn.Sequential`—that is, we would like you to make the order of computation explicit in your call to forward. (If you don't get better performance, you can still receive full credit here.)

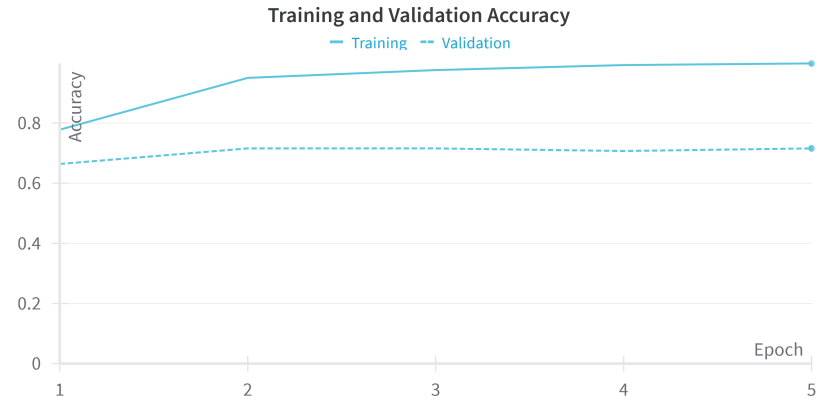
(a) (3 points) Using math, pseudocode, or a computation graph drawing, describe the new model that you tried.



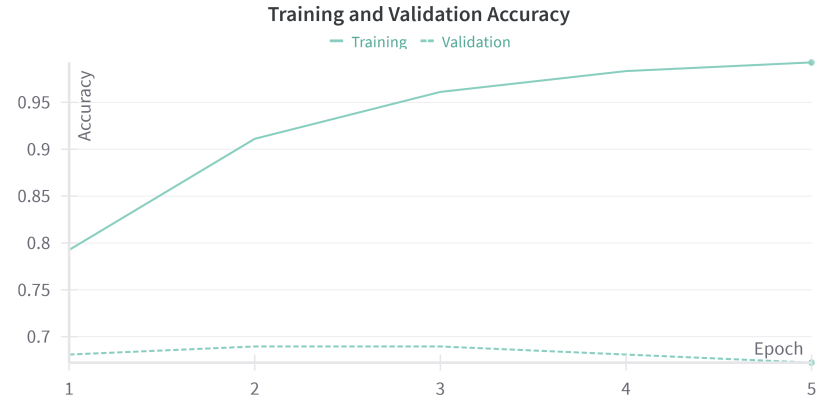
[3] https://pytorch.org/docs/stable/nn.html

(b) (4 points) With `wandb`, plot the train accuracy and validation accuracy on the same plot with the number of epochs on the horizontal axis. Include a run with the original model and a run with your new model.

New Model:



Original Model:



Title	Real Article	Fake Article
Vice Mayor of Wenzhou City Chen Yingxu and his delegation visited Ruian Middle School to investigate mental health education work	On November 27, 2023, Wenzhou Deputy Mayor Chen Yingxu and his delegation visited Ruian Middle School for investigation and guidance, and learned about the school's campus safety, mental health education and other work...	Wenzhou City, China - [Date] In a proactive move towards enhancing the mental well-being of students, Vice Mayor Chen Yingxu led a delegation to Ruian Middle School to delve into the intricacies of mental health education...
Bensalem Letter Carrier Honored For 43-Year Career	BENSALEM TOWNSHIP, PA —He was her mailman for 30 years. Every day, Debbie McBreen would see the man "who always had a smile on his face" —U.S. Postal Service letter carrier Kyle Livesay. "He was amazing, so friendly. He is just a wonderful person and an excellent man," said McBreen...	In a heartwarming ceremony held at the Bensalem Post Office, the community came together to celebrate the remarkable career of Mr. John Anderson, a dedicated letter carrier who has faithfully served the residents of Bensalem for an impressive 43 years...
No straight drive today as traffic to be diverted for T20I	Indore: As the city gears up for the much-anticipated T20I cricket match between India and Afghanistan on Sunday, Indore police released a traffic diversion plan to ensure a smooth commute during the match. DCP traffic Manish Agarwal said special arrangements have been made to ensure zero traffic block incidents on Sunday...	In anticipation of the throngs of cricket enthusiasts expected to flood the city for today's exciting Twenty20 International (T20I) match, city officials have announced a comprehensive traffic diversion plan. The match, set to be a showdown between two of the world's leading cricket teams...

Table 1: Examples of article snippets from the dataset.

3 Text Classification (22 points)

In this section, you will augment a text classifier written in PyTorch.

Dataset

> Note that the news articles, real and fake, in this dataset have not been filtered. Please take care when reading any of the data.

The text classification dataset consists of news articles, real and fake. The real articles were selected from news outlets around the world, with an emphasis on small-town local news. Each fake article was generated by a large language model with a prompt that included the title of the corresponding real news article.

The dataset is contained in three `.csv` files in UTF-8 encoding.

`./data/txt_train.csv` `./data/txt_val.csv` `./data/txt_test.csv`

Each one is identically formatted with one example per row. There are four columns:

1. The `article` column contains the text of the news article. The title of the article is prepended as the first line of this article text.
2. The `source` column differs for real/fake examples. For a real example, it contains the URL of a real article. For a fake example, it contains the prompt that was fed into the LLM to generate the article.
3. The `label` column is the human readable label (i.e. `real` or `fake`).
4. The `label_idx` column contains an integer representation of the label (i.e. 0 or 1).

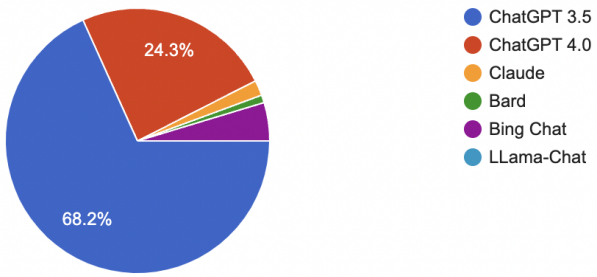
Examples of the news articles are shown in Table 1. The data is split roughly so that the training data contains 70%, the validation data 15%, and the test data 15%. Each of the train/val/test CSV files are sorted so that the pairs of real/fake news articles are adjacent rows.

Starter Code

The starter code in `txt_classifier.py` is a fully functional (albeit simple) text classifier based on the PyTorch tutorial for text classification. Before you continue, you should read this tutorial as it differs from the main PyTorch tutorial.

https://pytorch.org/tutorials/beginner/text_sentiment_ngrams_tutorial.html

Figure 2: Most of the fake news articles were generated by ChatGPT.



Data: The dataset for this section is similar in form to that from the tutorial. The starter code provides the class `CsvTextDataset` which is a `torch.utils.data.Dataset`. This class reads in the dataset from a given CSV, optionally applying some transformations to the text. We read the training dataset twice: the first time we tokenize the text but do *not* numericalize the tokens in order to build a vocabulary (i.e. a mapping of tokens to integers). The second reading of the training data then proceeds with numericalization. Each article is also truncated to some maximum number of tokens. The `DataLoader` uses a `collate_fn`, which is simply a callable (or function) that is applied to each batch. You will explore the behavior of our `PadSequence` in the questions below.

Model: The `TextClassificationModel` consists of just two layers and is detailed in the text classification tutorial.

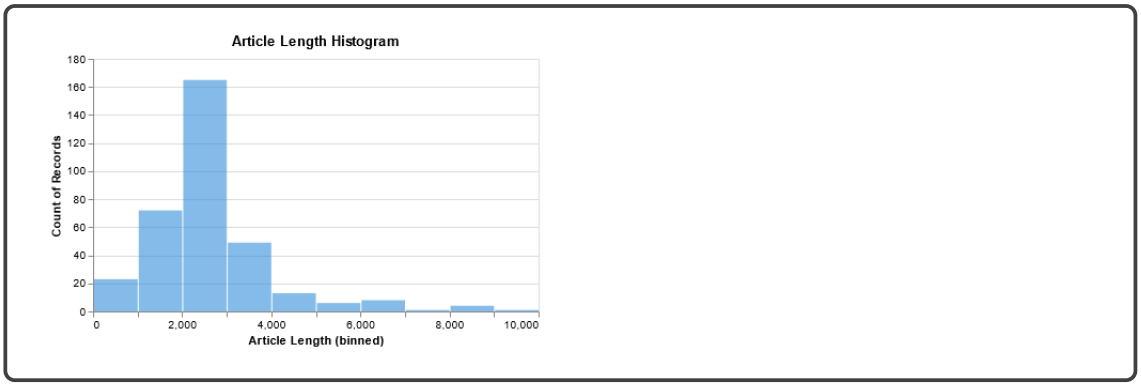
Training: The training process proceeds as usual: optimizing a cross entropy objective with SGD.

Evaluation: The model periodically evaluates on the train and validation set, and finally on the test set.

Experiments

(You are welcome, but not required, to instrument the code with `wandb`.)

3.1. (3 points) Generate a histogram of the lengths of the news articles in the training dataset. You can do this however you'd like. (One option is to use `wandb`. If doing so, you should follow the "Histograms in Summary" instructions from here. You will need to click on the run, then on its Overview tab, and then scroll to the bottom to find it.)



3.2. (3 points) Run the code three times and report the validation and test accuracies. (You can easily create such a table on the "Runs" tab of `wandb` if these are logged.)

	Val. Acc.	Test Acc.
Run 1	0.6842	0.7436
Run 2	0.8289	0.8077
Run 3	0.6447	0.7179

3.3. Carefully read the documentation for `nn.Embedding` and then `nn.EmbeddingBag` paying close attention to the `padding_idx` parameter the associated Examples. Next read the code for `PadSequence` so that you understand what it is doing.

(a) (1 point) Change the code so that the `collate_fn` parameter is not passed to the DataLoaders. What error is reported?

The error that is reported is "each element in list of batch should be of equal size"

(b) (2 points) Why is this considered an error?

This is considered an error because the mathematically, each article is represented as a list of tokens. For the sake of efficiency, each of the token lists in a batch are compiled into a tensor. So for the tensor to be formed, each of the token lists must be of the same length.

(c) (1 point) Now change the batch size to be 1 (instead of 8). What error is reported?

'list' has no attribute 'shape'

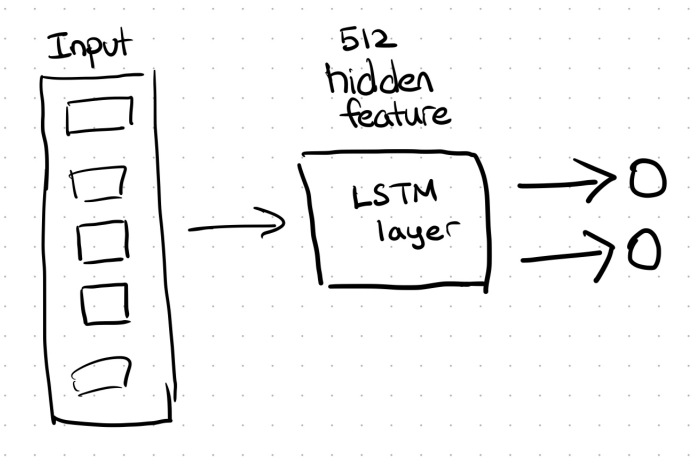
(d) (2 points) Why is this new error happening?

Since the batch size is 1, looping over the dataloader returns a list instead of a tensor. One of the lines of code calls "X.shape", however since X is a list it has no shape property.

(Revert any changes you made for the previous question.)

3.4. Replace the simple `TextClassificationModel` with an LSTM-based classifier. (You are welcome to use any of the code presented in the HW0 recitation.)

(a) (2 points) In math, pseudocode, or a computation graph, describe the structure of your new model.



(b) (4 points) Run the code three times and report the validation and test accuracies.

	Val. Acc.	Test Acc.
Run 1	0.6184	0.7051
Run 2	0.5395	0.6538
Run 3	0.5395	0.6923

3.5. (2 points) Using the original `TextClassificationModel`, switch the optimizer to Adam, then run the code three times and report the validation and test accuracies.

	Val. Acc.	Test Acc.
Run 1	0.9079	0.9359
Run 2	0.9474	0.9744
Run 3	0.9605	0.9615

3.6. (2 points) Look through a few of the real/fake news article pairs. In a few sentences, speculate on how the model is able to train successfully.

The model is able to train successfully because sometimes in the fake articles there are words that are just replaced by something like ”[number]” or ”[date]”. Also, fake articles tend to use less names than the real articles. For instance instead of using the name of a person like ”Governor Murry”, the fake article will say something like ”A spokesperson”.

4 Code Upload (0 points)

4.1. (0 points) Did you upload your code to the appropriate programming slot on Gradescope? *Hint:* The correct answer is 'yes'.

Yes

For HW0, you should upload *two* files: `img_classifier.py` and `txt_classifier.py` that include all your interesting new changes to the code.

5 Collaboration Questions (2 points)

After you have completed all other components of this assignment, report your answers to these questions regarding the collaboration policy. Details of the policy can be found in the syllabus.

5.1. (1 point) Did you collaborate with anyone on this assignment? If so, list their name or Andrew ID and which problems you worked together on.

Gabriel Fonseca (gcfonsec). Questions 2 and 3.

5.2. (1 point) Did you find or come across code that implements any part of this assignment? If so, include full details.

No.