CPSC 433: Aritificial Intelligence

Assignment: TensorFlow Neural Network Machine Learning

Collaboration

Discussing the assignment requirements with others is a reasonable thing to do and an excellent way to learn. However, the work you hand in must ultimately be your work. This is essential for you to benefit from the learning experience and for the instructors and TAs to grade you fairly. Handing in work that is not your original work but is represented as such is plagiarism and academic misconduct. Penalties for academic misconduct are outlined in the university calendar.

Here are some tips to avoid plagiarism in your programming assignments.

1. Cite all sources of code you hand in that are not your original work. You can put the citations into comments in your program. For example, if you find and use code found on a website, include a comment that says, for example:

   # The following code is from https://www.quackit.com/python/tutorial/python_hello_world.cfm.

   Use the complete URL so that the marker can check the source.

2. A tool like chat-GPT can be used to improve small code blocks. For example, three lines of code. If you get help from code assistance like Chat-GPT, you should comment above the block of code you requested assistance on debugging or improving and cite the tool used to get that suggestion. Using a tool like chat-GPT to write the majority of your assignment requirements will be treated as plagiarism if found without citation, and with citation, it will be treated as 0 for the component the student did not complete. Code improvement of short length will get credit if commented/cited properly.
3. Citing sources avoids accusations of plagiarism and penalties for academic misconduct. **However, you may still get a low grade if you submit code not primarily developed by yourself. Cited material should never be used to complete core assignment specifications. Before submitting, you can and should verify any code you are concerned about with your instructor/TA.**
4. Discuss and share ideas with other programmers as much as you like, but make sure that when you write your code, it is your own. A good rule of thumb is to wait 20 minutes after talking with somebody before writing your code. If you exchange code with another student, write code while discussing it with a fellow student, or copy code from another person's screen, this code is not yours.
5. **Collaborative coding is strictly prohibited**. **Your assignment submission must be strictly your code**. Discussing anything beyond assignment requirements and ideas is a strictly forbidden form of collaboration. This includes sharing code, discussing the code itself, or modelling code after another student's algorithm. **You can not use (even with citation) another student's code.**
6. Making your code available, even passively, for others to copy or potentially copy is also plagiarism.
7. We will look for plagiarism in all code submissions, possibly using automated software designed for the task. For example, see Measures of Software Similarity (MOSS - https://theory.stanford.edu/~aiken/moss/).
8. Remember, if you are having trouble with an assignment, it is always better to go to your TA and/or instructor for help rather than plagiarizing. A common penalty is an F on a plagiarized assignment.

Late Penalty

The individual assignment submitted within 24 hours of the initial deadline will receive 10% off, within 48 hours 20% off. After that the assignment will be an F.

Goal

Use Google TensorFlow to learn neural network-based machine learning in Python.

Technology

Python, TensorFlow, IPython, Jupyter Notebooks, Numpy, Pandas, MatplotLib, Seaborn

Specifics

Python 3, TensorFlow 2

Description

The goal of this assignment is to explore **two** machine learning problems:

- Part 1 is to replace the **MNIST** image data that you explored in your tutorials with a harder set of letter representation data and train a good model for it.
- Part 2 is to create and develop a model for data loaded from a **CSV** file about Canadian Football League draft/combine data.

These problems are designed to give you a chance to explore different aspects of TensorFlow.

TensorFlow 2.10

You can get more familiar with the **TensorFlow** documentation that you can find at

https://www.tensorflow.org/api_docs/python/ or by looking at the tutorials at https://www.tensorflow.org/tutorials . Note, we will be using **TensorFlow 2 for Python 3** for the Assignment. *You are expected to use Python 3.12.X* which is what is installed on lab machines.

To install **Python** packages, such as **TensorFlow**, on a university user account you will have to use a **virtual environment**. This should not be too challenging to setup and work in. There is a file called **setup.txt** with the assignment materials and tutorials will address it.

Instead of a local **Jupyter** install or online **Jupyter** notebook environment such as **Google Collab**. https://colab.research.google.com/ An online **Google Colab Jupyter** notebook environment lets us use both **Python 3** and **TensorFlow 2** from any location we desire that has internet access. Tutorials will also show how to use and code in this environment.

For the parts of this assignment starter files will be provided for **Jupyter** work. Certain material such as interactive GUI programs to make input images will **only work** in a **local** desktop environment.

Tutorial: MNIST Logistic Regression (if you didn't go)

The home page of the MNIST database is http://yann.lecun.com/exdb/mnist/. The dataset is divided into a set of 60,000 training examples and a set of 10,000 test examples, each consisting of separate files for the images and their labels. Details on the file format can be found at the bottom of the MNIST web page but we'll make use of TensorFlow to load this data so you don't have to concern yourself with investigating that website.

It is relatively simple to get accuracy of ~88% on the **MNIST** dataset with the provided starter code. However, for deployed machine learning this performance is generally considered unacceptable. The **MNIST** digit dataset is basically solved, and state of the art models reach accuracies above 99%. Your tutorial goal is to improve the performance of the starting model provided using hyper-parameters and simple changes to the structure of the neural network model.

Experiment by adjusting the hyperparameters (loss functions, optimizers, epochs, etc.) and/or structure (layers, activation functions, etc.) of the provided starter model to achieve an accuracy of 98%+ on the test data. Your model must be built in **TensorFlow 2.10** and **keras**. You will be provided the code of a starter model **MNIST-Starter.py (MNIST-Starter.ipynb)** that you are expected to modify to create **MNIST-Complete**. You should include lines of code that allow you to save your model as **MNIST.keras**.
https://www.tensorflow.org/tutorials/keras/save_and_load

The input/output layer of the **MNIST.keras** model should be such that you could import that model into the **MNIST-Starter** code in place of the compiled/fit model and run the evaluation query at the end on it skipping any creation and training steps.

Part 1: Logistic regression on a replacement for MNIST dataset

Machine learning community is a bit sick of seeing **MNIST** digits pop up everywhere, so they created a similar dataset and literally named it **notMNIST**. Created by ***Yaroslav Bulatov***, a research engineer previously at **Google** and then at **OpenAI**. **notMNIST** is designed to look like the classic **MNIST** dataset, but less 'clean' and extremely 'cute'. The images are still 28x28 and there are also 10 labels, representing letters 'A' to 'J'. The homepage for the dataset is http://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html .

I have done the work of reducing this dataset down into something that is in the same format as the **MNIST** dataset. This file is provided as **notMNIST.npz**.

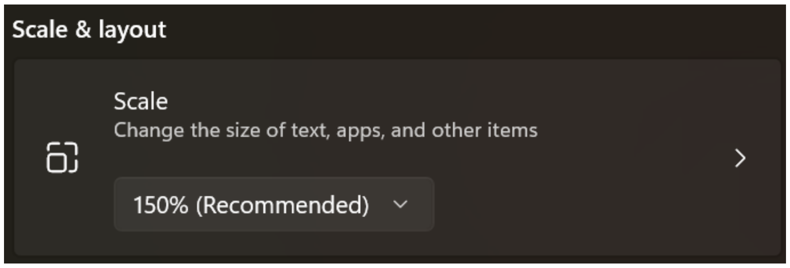
The starter code **notMNIST-Starter.py (notMNIST-Starter.ipynb)** has the new data loading included in place of the **keras MNIST** loading. This loading is done using the **numpy** library.

Build a model for this **notMNIST** data like you did for the tutorial and the **MNIST** data. This will be harder given the challenge of the letter pictures being much more diverse. To start re-use your model design and hyper-parameters to create **notMNIST-Partial**. One of my 98%+ MNIST model designs drops to 93% for notMNIST here.

Add a line of code to save your model to a file called **notMNIST-Partial.keras** like you did for the tutorial. Using this model, you can now use the following files:

predict_test.py (predict_test.ipynb) -> Provide index of test image from downloaded data and see prediction made by your model

grabimage.py -> Desktop GUI program to capture **image.png** images from mouse input, if you have issues the screen capture you may need to disable scaling/hdpi modes in your OS (below is an example in windows that causes issues at 150%, needs to be changed to 100%), you could also just use GIMP or similar tool to draw a b/w 28by28 pixel input image, a captured **image.png** has been provided you can edit


Scale & layout

Scale

Change the size of text, apps, and other items

150% (Recommended)

predict.py (predict.ipynb) -> Non-desktop program to take **image.png** image and predict using your model

interactive.py -> Desktop program combining **grabimage.py** and **predict.py** into one program for ease of use

In the **predict.py/predict_test.py/interactive.py** files you will have to change three lines to get them to work.

1. First a line to load your model. Reference the save/load tutorial again!
2. Second, a line to get an array of percent confidence in each class for your image. The beginner clothing prediction tutorial.
3. Third, a line to decide the index of the highest prediction in the previous array. The beginner clothing prediction tutorial.

Report what lines you added to get these programs to work in your report.

Use the above four programs to find **three** images in the test data (and/or create images using the tools provided) that you (or the test output label) would classify as one of the classes, but that your model gets clearly wrong on its prediction label. Make changes to your model that lead to these three images getting identified accurately. Store the image you made. Record what your model's accuracy was when this image was inaccurately predicted by **MNIST-Partial,** what changes you made, then what your model predicted after the changes. Your final model **notMNIST-Complete.keras** should reach 95% and be created by a file called **notMNIST-Complete**.

In a **report**,

1. describe your **Partial** model (paragraph),
2. the three images and their original **Partial** model label (paragraph),
3. then describe your changes to the model and how/why they improve the net's performance (paragraph at least),
4. and then show the three images and their new **Complete** model label (paragraph).

You should have the following to submit

- notMNIST-Partial.py (or notMNIST-Partial.ipynb) *[based on notMNIST-Starter]*
- notMNIST-Complete.py (or notMNIST-Complete.ipynb) *[based on notMNIST-Partial]*
- notMNIST-Partial.keras [saved model from notMNIST-Partial]
- notMNIST-Complete.keras [saved model from above with 95%+ on test data]
- whichever of predict_test, predict, interactive you modified
- report.pdf

Part 2: Build a logistic regression model to predict CFL draft likelihood

You have been given a **draft.csv** file of data in csv format.

You should divide this file into **draft_train.csv** and **draft_test.csv** data. Generally, test data is only a portion of the size of the training data. For example, in the **MNIST** data sets the test data is 1/6 of the size of the training data or (1/7 of the total data). For example, 60,000 training examples and 10,000 test examples.

For the file, the first row is the name of the columns variables. If an entry is missing data 0 is used.

1. position: What position a player had, text
2. longsnapper: Did a player identify as a long snapper, (1=Yes, 0=No)
3. height: Inches, numerical
4. weight: Pounds, numerical
5. draftage: Years, numerical
6. bench: Count of 225lb bench repetitions, numerical

7. forty: 40 yard dash hand timed seconds, numerical

8. fortyelectric: 40 yard dash electrical timed seconds, numerical

9. verticaljump: Vertical jump inches, numerical

10. broadjump: Horizontal jump inches, numerical

11. threecone: Three cone run seconds, numerical

12. shuttle: Shuttle weave run seconds, numerical

13. drafted: Was player drafted in CFL, (1=Yes 0=No)

Each following row contains the information of one player.

There are 671 samples in total.

We will be using the first 12 variables to predict the last variable. That is, your input will be 1-d tensor of 12 elements, and your label is binary. You should write the function to read in data yourself, and you should take care of dividing your data into train set and test set.

In terms of loading and processing csv data you will find the tutorial at https://www.tensorflow.org/tutorials/load_data/pandas_dataframe very helpful. You have 10 numeric pieces of data, 1 binary, and 1 category.

Start with the most basic of models.

```
tf.keras.Sequential([
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dense(1)
])
```

One problem you will notice when building this model is overfitting due to the small amount of data samples for training. A model like this at sufficient epochs could reach 100% on the training data! But still be ~60% for test data.

Examine the differences in your initial model attempts between higher accuracies of training data vs much lower accuracies in test data. Then make changes to your model to deal with overfitting and report the impact of your changes before and after. https://www.tensorflow.org/tutorials/keras/overfit_and_underfit will be of good use for this part.

In a report,

1. explain your data-splitting decisions (paragraph),
2. how you decided to handle the non-numerical input data (paragraph),

3. how you dealt with overfitting/underfitting (paragraph),
4. and report your hyper-parameters/results (paragraph).

You should have the following to submit

- CFLModel.py (or CFLModel.ipynb)
- draft_train.csv
- draft_test.csv
- report.pdf (should be same report from part 1 but updated with part 2 section)

Submit the following to the D2L Assignment Dropbox:

1. An electronic copy (zip file) of your source code files and models requested
2. Your report.pdf

Grading

Part 1 Code	Source code for model	2	
Part 1 Predict	Source getting load, save, predict to work	1	
Part 1 Improved	Image exploration and discussion	2	
Part 1 Report	Report on model/hyper-parameters	3	
Part 2 Data	Getting data ready	2	
Part 2 Model	Source code for model	2	
Part 2 Overfit	Examining over-fitting	2	
Part 2 Report	Report on model/hyper-parameters	3	
Code quality	Name, class, semester, tutorial, UCID, comments, citations, etc.	3	
Total		**20**	

Bonus

Letter	**A+**	**A**	**A-**	**B+**	**B**	**B-**	**C+**	**C**	**C-**	**D+**	**D**	**F**
Points	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**<9**