CPSC 501: Advanced Programming Techniques

Assignment 2: 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 that you hand-in that are not your original work. You can put the citation into comments in your program. For example, if you find and use code found on a web site, 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. Citing sources avoids accusations of plagiarism and penalties for academic misconduct. **However, you may still get a low grade if you submit code that is not primarily developed by yourself. Cited material should never be used to complete core assignment specifications. You can and should verify and code you are concerned with your instructor/TA before submission.**
3. Discuss and share ideas with other programmers as much as you like, but make sure that when you write your code that 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, then this code is not yours.
4. **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 code itself, or modelling code after another student's algorithm. **You can not use (even with citation) another student's code.**
5. Making your code available, even passively, for others to copy, or potentially copy, is also plagiarism.
6. We will be looking 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/).
7. Remember, if you are having trouble with an assignment, it is always better to go to your TA and/or instructor to get help than it is to plagiarize. **The most common penalty is an F on a plagiarized assignment.**
8. For assignments **limited use of generative AI in writing assistance is acceptable. For example, grammar suggestion, or code suggestion tools for programming**. **Programming or text that is largely generative AI produced is not allowed**. Learners are ultimately accountable for the work they submit. **Use of AI tools must be documented in an appendix for the assignment. The documentation should include what tool(s) were used, how they were used, and how the results from the AI were incorporated into the submitted work. Failure to cite the use of AI generated content in an assignment will be considered a breach of academic integrity and subject to Academic Misconduct procedures.**

Late Penalty

Assignment Late Policy: Students may choose to submit one or more assignments late, for no more than a total of five (5) days over the entire semester for the combined 4 assignments.

Each 24-hour period late after an assignment deadline counts as one full day regardless of how many hours the assignment was late within that period. For example, deadlines are generally on a Friday, 11:59pm local time. That means an assignment submitted any time Saturday before 11:59pm local time will be considered as 1 day late and count against the total number of late days.

Use these days at your own discretion and without explanation during the course for assignment extensions. For example, you could submit your second assignment 3 days late and your final assignment 2 days late, or just your final assignment 5 days late.

If a student still has days left to use, then their assignment will be graded without penalty. If a student has no more days left, or their submission exceeds the days they have remaining, then they will receive a 0 grade for a late assignment.

Late day usage will be tracked in a D2L grading column.

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, csgit.ucalgary.ca

Description

The goal of this assignment is to explore **four** machine learning problems.

1. There will be **one** **Main** machine learning problem that runs the length of the assignment, and then
2. **three** others that are inter-leaved through-out the longer one. These 3 will operate more as a for credit tutorial in learning the basics that help you do the first part.

Each of these 4 parts of assignment should be stored in its own folder in a Gitlab project (**Main**/Part1/Part2/Part3). Grading will be roughly 50/50 between these two.

The **Main** machine learning problem is to find a dataset, load this dataset into **Python**, view this dataset using **MatPlotLib**/**Seaborn**, process this dataset using **NumPy**/**Pandas**, and then using **TensorFlow** in **Python** train a model to use a portion of this dataset's columns as input variables

and in attempt to predict another column as an output variable. Finally, you will view the performance of this trained model using **MatPlotLib**/**Seaborn** again. Feasibility of the dataset size, and time to train a model is important here. You should avoid large image datasets, or video, and limit yourself to tables of data that could be stored in .csv files.

The inter-leaved smaller three machine learning problems (Part1/Part2/Part3) are the following: Part 1 is to improve the logistic regression accuracy of a **MNIST** digit image recognition model. Part 2 is to replace the **MNIST** image data with a harder set of letter representation data and train a good model for it. Part 3 is to create and develop a model for data loaded from a **CSV** file about Coronary Heart Disease and associated risk factors. These three smaller problems are designed to give you a chance to explore different aspects of TensorFlow and then apply learned lessons to the **Main** larger problem.

TensorFlow 2.18

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 2.18 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.

Main Project Stage 1: Data Science and Machine Learning

Source a dataset for your project. This dataset should have at least 7 columns and 500 rows of data and have both numerical features and non-numerical (text-based) features. You should have at least 6 input variable columns of which 5 are numerical (the other should be text like a category label). You should have at least 1 output variable column that is numerical. If you want to use a dataset without these, then you'll have to request approval from your TA/instructor. The goal of this dataset is to use some of the column data (input variables) to predict at least one of the other columns of data (output variables). Again, these are minimums for input data

and more expansive input variable quantities, or the prediction of more than one output variable, is allowed.

For the first part of the project your requirement is to find a dataset and determine that you have the rights to use this dataset. Common sources of dataset are **kaggle.com** but can also include resources like the City of Calgary, Province of Alberta, or other sources that provide open datasets. Your dataset does not have to be a single file, but this will be common to find. You may find that your preferred dataset will require you to combine information from more than one source. This is allowed but you may want to consult your TA or in the instructor on advice if this is going to cause any concerns for your project in terms of scope.

1. Once you have this dataset (data-input.csv), load it into **Python** using **NumPy**/**Pandas**. At this point if you have more than one input file (data-input1.csv, data-input2.csv, etc.), then you will have to load and combine them into one dataset. Your **Python** code file (main1.py/main1.ipynb) should output a csv file of your complete dataset at this point (data.csv) with proper simple readable column headers (as would be produced by **Pandas** csv output). *[data.csv could end up looking like your data-input.csv if the data is already a single table in named column form]*
2. Once you have your full dataset you will need to clean that dataset. Cleaning a dataset involves exploring if there is missing data, inconsistent data, or other parts of dataset that would make it dangerous to use in machine learning. **Pandas** is a data science data processing library well-suited to do this. *Remember, garbage in -> garbage out for how machine learning works.* Your **Python** code file should output a csv file of your complete **cleaned** dataset (data-cleaned.csv) at this point with proper readable column headers (as would be produced by **Pandas** csv output).
3. You should use **Seaborn**/**MatPlotLib** to make **three** visualizations of your current dataset to help others understand some features of your dataset. For example, what is the distribution for a column of data (ex. histogram), or what is the relationship between two variables of your data (ex. scatter plot, line chart). These visualizations (plot1.jpg, plot2.jpg, plot3.jpg) should be output by your **Python** program.
4. Start a report (report.md) in the **base gitlab project** project folder that reports for **Stage 1:** where the data came from (paragraph), any specific challenges you had in combining/cleaning data or why it is clear cleaning did not need to be done (paragraph), and then a description of what your three visualizations show (paragraph each visualization).
5. At this point you'll stop with this part of project and do Part 1 of the smaller problems to explore **TensorFlow** machine learning.

Your Gitlab project should have in the **base gitlab project** folder:

- report.md (add a Main Stage 1 sub-section)

Your Gitlab project should have in the **Main sub-**folder:

- main1.py (or main1.ipynb)
- data-input.csv (or data-input1.csv, data-input2.csv, etc.)
- data.csv
- data-cleaned.csv
- plot1.jpg, plot2.jpg, plot3.jpg

Part 1: MNIST Logistic Regression

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 95%+ on the test data. Your model must be built in **TensorFlow 2.18** 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.

In a **report**, describe your hyper-parameters (paragraph), model (paragraph), results (paragraph) justifying the changes you made, including relevant code excerpts. In your report, it should be clear what changes you made, as well as how/why they should have improved the neural network's performance to reach your final accuracy of 95%+.

Your Gitlab project should have in the **base gitlab project** folder:

- report.md (add a Part 1 sub-section)

Your Gitlab project should have in the **Part 1 sub-**folder

- MNIST-Complete.py (MNIST-Complete.ipynb) *[based on MNIST-Starter]*

- MNIST.keras *[saved model from above with 95%+ on test data]*

Main Project Stage 2: Data Science and Machine Learning

Return to your overall project. At this point we want to create a simple neural network **TensorFlow** model to do a prediction on your data. The minimal requirement is that 5 numerical columns of your data must be involved as input variables and at least 1 numerical column predicted as an output variable. For now, use only numerical input/output variables. We'll introduce how to utilize your non-numerical variables like text/category data later.

1. Divide your rows of data into two groups for training/test data. You will need to justify your choice for the size of each group of data in your report later so consider why you are making the choice you make.
2. Split off the input variables and output variables into separate parts for each of the **training** input/output variables and **test** input/output variables.
3. Train a simple model like the model trained in Part 1 for MNIST on your **training** data. Then verify this model by running it against you **test** data. Make similar hyper-parameter and model adjustments as you tried in Part 1 to improve this model as best you can (note we are not done with the model yet and will improve it more later). *[You should note the model from Part 1 was a classifier and had 10 labels {0,1,…,9} which is why the last layer had 10 neurons with softmax activation and used sparse categorical cross-entropy, while if you choose something binary {0,1} here you will have a last layer of 1 neuron with sigmoid activation and will have binary cross-entropy.]*

Continue your previous report. You must discuss your choice: of what columns you chose for this simple model as input and output variables and why (paragraph), what choices you made when splitting data into train/test (paragraph), what choices you made in your simple model design (paragraph), what hyper-parameters your chose (paragraph), and discuss the current accuracy of your model on the training and test data (paragraph). (There is no required accuracy here for training or test data.)

Your Gitlab project should have in the **base gitlab project** folder:

- report.md (add a Main Stage 2 sub-section)

Your Gitlab project should add/change in the **Main** folder:

- main2.py (or main2.ipynb) *[modified version of main1]*

Part 2: 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**.

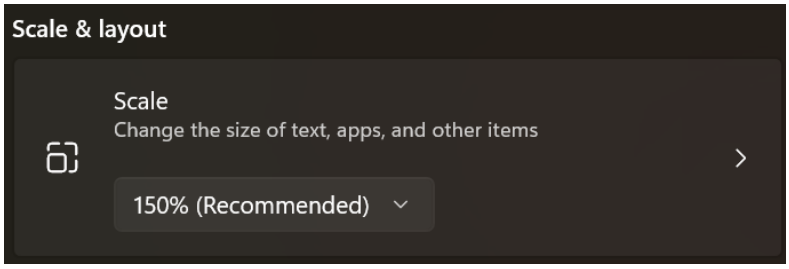
The starter code **notMNIST-Starter.py (notMNIST-Starter.ipynb)** has the new data loading included in place of the **keras MNIST** loading. This loading is done using the **numpy** library.

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



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 93% 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).

Your Gitlab project should have in the **base gitlab project** folder:

- report.md (add a Part 2 sub-section)

Your Gitlab project should have in the **Part 2 sub-**folder

- 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 93%+ on test data]
- whichever of predict_test, predict, interactive you modified

Main Project Stage 3: Data Science and Machine Learning

Return to your overall project. At this point we want to examine our model to determine what it appears to have not been good at. Choose at least one column of input and visualize how it relates to the predictions of your model in main3 (main3.ipynb). Add improvements learned in Part 2 to increase your accuracy of your model.

1. Modify your report and add your visualization (paragraph), and report what additional changes were made to your original model after Part 2 (paragraph). (There is no required accuracy here on either part of your data.)

Your Gitlab project should have in the **base gitlab project** folder:

- report.md (add a Main Stage 3 sub-section)

Your Gitlab project should add/change in the **Main** folder:

- main3.py (or main3.ipynb) *[modified version of main2]*

Part 3: Build a logistic regression model to predict coronary heart disease

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

You should divide this file into **heart_train.csv** and **heart_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 observed variables. There are 10 variables:

1. sbp: Systolic blood pressure
2. tobacco: Cumulative tobacco consumption, in kg
3. ldl: Low-density lipoprotein cholesterol
4. adiposity: Adipose tissue concentration
5. famhist: Family history of heart disease (1=Present, 0=Absent)
6. typea: Score on test designed to measure type-A behavior
7. obesity: Obesity
8. alcohol: Current consumption of alcohol
9. age: Age of subject
10. chd: Coronary heart disease at baseline; 1=Yes 0=No

Each following row contains the information of one patient.

There are 462 samples in total. *Source of the data:*
http://statweb.stanford.edu/~tibs/ElemStatLearn/

We will be using the first 9 variables to predict the last variable. That is, your input will be 1-d tensor of 9 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.

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).

Your Gitlab project should have in the **base gitlab project** folder:

- report.md (add a Part 3 sub-section)

Your Gitlab project should have in the **Part 3 sub-**folder

- CHDModel.py (or CHDModel.ipynb)
- CHDModel.keras
- heart_train.csv
- heart_test.csv

Main Project Stage 4: Data Science and Machine Learning

Return to your overall project and complete it. At this point we want to use lessons from Part 3 to integrate your non-numeric columns, and to consider over-fitting/under-fitting. Add improvements learned in Part 3 to accomplish both goals.

2. Modify your report discuss what additional changes were made to your original model after Part 3 for integrating the non-numeric data (paragraph), what you did for over-fitting/under-fitting concerns (paragraph), and it there were any other model decisions you made such as batch processing, normalization, one-hot-encoding, etc. (paragraph). (There is no required accuracy here on either part of your data.)

Your Gitlab project should have in the **base gitlab project** folder:

- report.md (add a Main Stage 4 sub-section)

Your Gitlab project should add/change in the **Main** folder:

- main4.py (or main4.ipynb) *[modified version of main3]*

Submit the following to the D2L Assignment 2 Dropbox:

1. An electronic copy (zip file) of your exported source code from your final Gitlab project
2. Directions/access to the **csgit.ucalgary.ca** project so that your TA can access your project and read your report. TAs will need to be added as at least a 'Developer' role to your code.

Version control: As in the previous assignments, you will be using GitLab to maintain version control and to share your final project with the TAs. Your assignment should be kept in a repository titled CPSC501W25A2. As you develop your code, make sure to use proper version control practices, making regular commits with descriptive messages. This includes keeping a local copy of your assignment (including the git repository) on your own computer, just in case.

Your readme.md for your project should include the expected base identifying information about the assignment, but also the source/rights to the dataset used in the **Main** part of assignment. If you are having issues completing Main/Part1/Part2/Part3 portions of the report.md then you can complete these in a report.pdf instead, but you should still have the base data source information requested in the readme.md

Your D2L dropbox should also include directions for the TA to access your project (a csgit link). This is how they will be able to access your code and commit history, so double-check this works correctly before submitting. Make sure to indicate in your dropbox submission whether you decided to implement the bonus part of the assignment. You may also include any information (known bugs, etc.) that you think will be useful to the TAs when grading.

Bonus

- Part 1: 98% or higher, Part 2: 95% or higher, Part 3: 75% or higher (on saved model)
- Complete a Dockerfile that someone could use with your repository so that they didn't have to setup a virtual environment like we did to get your code running on their computer. They'd just need to download setup Docker, download your Dockerfile instead, and run your code that is already inside of it. (Put this Dockerfile in the base of your Gitlab directory with your readme.md) Include readme.md information on the command line commands to execute each of your python files like main.py within the Dockerfile.

Grading

Main Report 1	Where the data came from (paragraph), any specific challenges you had in combining/cleaning data or why it is clear cleaning did not need to be done (paragraph), and then a description of what your three visualizations show (paragraph each visualization).	5	
Main Report 2	What columns you chose for this simple model as input and output variables and why (paragraph), what choices you made when splitting data into train/test (paragraph), what choices you made in your simple model design (paragraph), what hyper-parameters your chose (paragraph), and discuss the current accuracy of your model on the training and test data (paragraph).	5	
Main Report 3	Visualization of accuracy (paragraph), and report what additional changes were made to your original model after Part 2 (paragraph).	2	
Main Report 4	Additional changes were made to your original model after Part 3 for integrating the non-numeric data (paragraph), what you did for over-fitting/under-fitting concerns, and it there were any other model decisions you made such as batch processing, normalization, one-hot-encoding, etc.	3	
Main Code	Load data, clean data, save data, visualizations, split data, make model, fit model, predict data, non-numeric, over-fit/under-fit	10	
Part 1 Code	Source code for model	2	
Part 1 Report	Report on model/hyper-parameters	3	
Part 2 Code	Source code for model	2	
Part 2 Predict	Source getting load, save, predict to work	1	
Part 2 Improved	Image exploration and discussion	2	
Part 2 Report	Report on model/hyper-parameters	3	
Part 3 Data	Getting data ready	2	
Part 3 Model	Source code for model	2	

Part 3 Overfit	Examining over-fitting	2	
Part 3 Report	Report on model/hyper-parameters	3	
Gitlab usage	Commits, Structure, Readme.md, dataset source/rights	3	
Total		**50**	
Bonus	Part 1 98%+, Part 2 95%+, Part 3 75%+, Dockerfile	5	

Letter	**A+**	**A**	**A-**	**B+**	**B**	**B-**	**C+**	**C**	**C-**	**D+**	**D**	**F**
Points	**Above 50**	**47.5**	**45**	**42.5**	**40**	**37.5**	**35**	**32.5**	**30**	**27.5**	**25**	**Below 25**