MSBA – Optimization 2
Project 1 – Image Classification

Your job for this project is to become a full-stack developer. You will train a machine learning model, dockerize it, host it on a cloud back end, and develop a front-end webpage so that users can interact with it! This will all be centered around building a bot that plays Connect 4. You will train a neural network to play, and then host a webpage that accesses that model and allows visitors to play against your bot. Rules for Connect 4 can be found at: https://en.wikipedia.org/wiki/Connect_Four.

**Specifics:**

1. Build a big data set of board positions, and the best move for each position. Although Connect 4 is a solved game, I don't know the perfect strategy. To make up for this, we will use Monte Carlo Tree Search to find the 'best' move for each board position. Code for MCTS is included in this project. You can accomplish this by having MCTS play against itself. You will build a dataset of boards (X), and the moves recommended by MCTS (Y) for each corresponding board. It may help to build a more diverse dataset if sometimes you play a few random moves, instead of the move recommended by MCTS, to start the game. Don't add these random moves to your dataset – just the moves that come later recommended by MCTS.

   MCTS is a tunable algorithm that can increase or decrease its skill (and time to compute the move). You can learn more about it at https://mcts.netlify.app/ or https://www.harrycodes.com/blog/monte-carlo-tree-search or https://www.youtube.com/watch?v=UXW2yZndl7U. Be sure that your data set is large, and diverse. Also, be sure to end a game, and not add any more data to your training set for that game, once the game is won. Note that MCTS has randomness in it, so if you show the same board to MCTS twice, you might get two different recommendations for the 'best' move. The higher the skill, the less likely this is to happen. You're free to shrink your dataset if the same board shows up multiple times with different recommended moves. Maybe grab the most frequently recommended move?

   I encourage you to watch this documentary: https://www.youtube.com/watch?v=WXuK6gekU1Y. It is about a team of researchers at DeepMind (a subsidiary of Google) who trained a bot to play Go – a board game that is much harder than Chess. The first step of that bot was to train to play like human experts. We will replicate that first step for Connect 4, but instead of using a dataset of human expert moves, we will make a dataset of moves determined by MCTS. Later in the semester we will build upon this.

2. Using this dataset, train 2 neural networks for classification. One neural network will be a convolutional neural network and the other will be a transformer. We can think of the checkers on the board as pixels in an image, and then this is an image classification problem with 7 categories – one for each possible column to drop a checker. Your job is to find the best architecture you can for the CNN and the transformer and compare the 2 structures. You can test the quality of your trained models by having them play against MCTS and see how often they win (against a less sophisticated MCTS bot), or how many moves it takes to lose, or how accurately it predicts the best move on a second board/move dataset (validation_split). For the CNN, you can use multiple convolutional layers, different filter sizes, different number of filters, multiple max pool layers, several dense layers, regularizers, dropout layers, different batch sizes or number of epochs. For the transformer, you can use different hidden, key/query, and value dimensions, different number of heads for the MHSA layers, different number of layers. You may want to let the boxes overlap for the transformer, even though we did not do this in class. It's up to you how you build your networks, but we will eventually play against your bot, and if it loses too easily you will lose points on the assignment. You may be better off doing this part of the assignment on Colab rather than on your own machine. The network I trained that plays well took several hours to train on my laptop but took just a few minutes on Colab's GPU.

   There are two ways you can encode your board.
   a. A 6x7 grid of +1s, -1s, and 0s. +1 for every location where there is a +1 checker, -1 for every location where there is a -1 checker, and 0 for every empty spot.
   b. A 6x7x2 grid of +1s and 0s. (i,j,0) = 1 and (i,j,1) = 0 if the $i^{th}$ row, $j^{th}$ column is +1. (i,j,0) = 0 and (i,j,1) = 1 if the $i^{th}$ row, $j^{th}$ column is -1. And 0 everywhere else.

   Option a is easier to encode, but option b is probably a better representation for a neural network to learn. You can write a quick function that switches between the two if you want to.

   HINT: Connect 4 has 2 players – in our code it's plus and minus one. Your neural network only needs to be trained to play plus moves. Whenever it needs to play minus moves, or train using minus moves that were played, just multiply the board by -1 (for option a) or switch the $0^{th}$ and $1^{th}$ indices for the last dimension (for option b), and predict the best move for plus. This will become clearer when you look at the included code.

3. To present this information, you will build an interactive webpage. It is important to be able to write code that does machine learning that is accessible to more than just you! You will build your webpage using Anvil. Further instructions and help with Anvil are below. Your webpage should have two main components. One component should detail your training process – what went well, what went

poorly, what the best architecture was, why you think it worked well or not – a thorough description of training a neural network to play Connect 4. The second component should be interactive and allow the visitor to play against the bots you built. Think of this as an interactive blog post.

4. Anvil is not capable of hosting a tensorflow model, so you will host this on a cloud service – AWS. To do this you will need to build a docker image that is then hosted on the AWS machine and accessible to your Anvil page. Details on this are below too.

## Web Page – Anvil

The traditional tools to build webpages using python are django and flask. Both require knowledge of html. A new alternative has emerged recently called anvil: https://anvil.works . Anvil is an online service that allows you to easily build webpages by dragging and dropping components onto a page, like a wordpress webpage. But then the components can rely on python code.

Anvil is free to use but the free version is limited: you can't use numpy, pandas or other similar packages. There is a workaround for this that I will explain below.

The user community for Anvil is large and the forums are actively monitored by Anvil employees to answer questions. If you have a question of how to do something, you'll likely be able to find the answer on their page.

Anvil has many blog posts that teach how to use their tool. Here are a few that may be helpful.

1. https://anvil.works/learn/examples/meter-feeder
2. https://anvil.works/blog/plotting-in-matplotlib
3. https://anvil.works/docs/media
4. https://anvil.works/learn/tutorials/jupyter-notebook-to-web-app
5. https://anvil.works/forum/t/how-to-add-multiple-pages-within-an-app/74/3
6. https://anvil.works/docs/client/python/alerts-and-notifications
7. https://anvil.works/docs/deployment/quickstart
8. https://anvil.works/docs/users
9. https://anvil.works/learn/tutorials/google-colab-web-service

## Accessing Python Packages with Anvil

Many python packages are unavailable to Anvil, such as numpy, pandas, and tensorflow. To build a webpage that can use those packages there is a workaround.

Anvil can call python functions that live on other computers. For example, you can write a jupyter notebook on your computer that has some functions in it and tell anvil to access those functions. Then when someone uses your webpage, if that function needs to be called, it will be executed on your machine. This is the anvil uplink feature. Tutorial 4 above uses this feature.

The uplink feature requires your computer to be turned on and running your jupyter notebook when the page is called. This is burdensome. Another work around (which you will need to do for the project) is to host your python code on an AWS virtual machine. The simplest way to do this, that I know of, is through AWS Lightsail. There are MANY different types of AWS virtual machines. Lightsail simply aggregates a few AWS services (EC2, S3, ...) into a user-friendly virtual machine.

Go to aws.amazon.com and sign up for an account. You must sign up with a credit card, but you'll be able to launch a machine for free that won't charge your card. Once you create and login to your AWS account, in the top search bar search for lightsail and click the link. Under the instances tab, click the button to create an instance. Select Linux/Unix as the platform and select the Django app for the blueprint (this will just make sure python is installed on your instance). Then go down and pick whichever rate plan is currently free. As of me writing this, there are 2 plans that have 3 free months. Just remember to delete this instance when the semester is over to avoid being charged after the 3 months.

Once the instance is created (this may take 5-10 minutes of waiting for AWS to create the machine), go back to the lightsail page and on the instance tab there will be an icon of >_. Click that and it will open a new window that is the terminal for your machine. You can control your AWS machine through this terminal.

You AWS machine should NOT be used to train your neural networks. You should do that on your own machine or on Colab. Once you train your model, you'll save it and upload it to the AWS machine and allow your python code on that machine to reference the trained model.

You need to be able to transfer files (like your code and your tensorflow model) into your machine too; you do that using an ftp app called filezilla. Instructions can be found here: https://lightsail.aws.amazon.com/ls/docs/en_us/articles/amazon-lightsail-connecting-to-linux-unix-instance-using-sftp

You'll need to write your python code in a .py file, instead of a jupyter notebook. At the bottom of your .py file be sure to include anvil.server.wait_forever(). This will make sure your functions stay available to be called by your webpage. This code should just be sent the current board status and return a move by plugging the board into the neural network. It should not be used for training a model!

Once you're ready to host your back-end code you will need to dockerize it. Docker is a VERY popular tool in the data science and software development industries. It allows your code to be run on any machine anywhere, even if that machine is a different type of computer with different software/packages installed. Detailed instructions for Docker are attached to this assignment: README-docker.txt.

**Your anvil webpage should**:

1. Require a user to sign in with email and password
    a. Create an account for me with email: dan and password: Optimization1234
        i. I know dan is not an email address, but that's actually ok if you create the account for me!
    b. Do not allow users to create new accounts automatically.
    c. The rest of the page should only be accessible if a user is logged in.
2. Have a detailed description of how you built your NNs, some pictures of boards you can classify (find the best move) easily and some boards where you can't, and analysis of why some boards are classified poorly. Think of this as a medium article you would write about playing Connect 4 with your network.
3. Have a section/page that allows the user play against your bots. The user should be able to select if they will play against the CNN or the Transformer.
4. Have a button that allows the user to start a new game.
5. Publish the app at the following url: MSBA25optim2-{YOUR group number}.anvil.app
6. All your backend code that needs to use python packages should be hosted on AWS. Please submit a screengrab of your AWS machine running your code. We will not announce when we will grade your assignment, so it's important that you AWS server is up and running for a few weeks.
7. Make your site look nice, 50% of your grade will come from the appearance of your site.
8. Include a link to clone your anvil code in your submission.
9. The code you are running on your AWS backend should be submitted along with your links to the public site and your anvil clone.