Homework 6.5: Narwhal Networks

10-301/10-601 Introduction to Machine Learning (Spring 2023)
https://www.cs.cmu.edu/ mgormley/courses/10601/
OUT: 2023-04-01
DUE: 2023-04-02
TAs: Neural the Narwhal, Markov

START HERE: Instructions

- **Collaboration Policy**: Please read the collaboration policy here: `http://www.cs.cmu.edu/~mgormley/courses/10601/syllabus.html`
- **Late Submission Policy:** See the late submission policy here: `http://www.cs.cmu.edu/~mgormley/courses/10601/syllabus.html`
- **Submitting your work:** You will use Gradescope to submit answers to all questions and code. Please follow instructions at the end of this PDF to correctly submit all your code to Gradescope.
  - **Written:** For written problems such as short answer, multiple choice, derivations, proofs, or plots, please use the provided template. Submissions can be handwritten onto the template, but should be labeled and clearly legible. If your writing is not legible, you will not be awarded marks. Alternatively, submissions can be written in LaTeX. Each derivation/proof should be completed in the boxes provided. You are responsible for ensuring that your submission contains exactly the same number of pages and the same alignment as our PDF template. If you do not follow the template, your assignment may not be graded correctly by our AI assisted grader and there will be a **2% penalty** (e.g., if the homework is out of 100 points, 5 points will be deducted from your final score).
  - **Programming:** You will submit your code for programming questions on the homework to Gradescope. After uploading your code, our grading scripts will autograde your assignment by running your program on a virtual machine (VM). You are only permitted to use the Python Standard Library modules and `numpy`. Ensure that the version number of your programming language environment (i.e. Python 3.9.12) and versions of permitted libraries (i.e. `numpy` 1.23.0) match those used on Gradescope. You have 10 free Gradescope programming submissions, after which you will begin to lose points from your total programming score. We recommend debugging your implementation on your local machine (or the Linux servers) and making sure your code is running correctly first before submitting your code to Gradescope.
- **Materials:** The data and reference output that you will need in order to complete this assignment is posted along with the writeup and template on the course website.

You may not use any late days for this assignment. No extensions will be granted in the interest of maintaining the planned schedule.

Instructions for Specific Problem Types

For "Select One" questions, please fill in the appropriate bubble completely:

Select One: Who taught this course?

- ● Matt Gormley
- ○ Marie Curie
- ○ Noam Chomsky

If you need to change your answer, you may cross out the previous answer and bubble in the new answer:

Select One: Who taught this course?

- ● Henry Chai
- ○ Marie Curie
- ⊗ Noam Chomsky

For "Select all that apply" questions, please fill in all appropriate squares completely:

Select all that apply: Which are scientists?

- ■ Stephen Hawking
- ■ Albert Einstein
- ■ Isaac Newton
- □ I don't know

Again, if you need to change your answer, you may cross out the previous answer(s) and bubble in the new answer(s):

Select all that apply: Which are scientists?

- ■ Stephen Hawking
- ■ Albert Einstein
- ■ Isaac Newton
- ⊠ I don't know

For questions where you must fill in a blank, please make sure your final answer is fully included in the given space. You may cross out answers or parts of answers, but the final answer must still be within the given space.

Fill in the blank: What is the course number?

10-601 10-6301

Programming: *The Algorithm* (25 points)

In this assignment, you will be implementing *The Algorithm*, Twitter's recently open-sourced internal recommendation algorithm[1]. While this may sound daunting, you should have enough knowledge from studying for Exam 2 and going to Homework 7 Recitation[2] to complete this assignment within a couple hours.

The Twitter Recommendation Algorithm is a set of services and jobs that are responsible for constructing and serving the Home Timeline. For an introduction to how the algorithm works, please refer to Matt's February 29th lecture slides. The diagram below illustrates how major services and jobs interconnect.

These are the main components of the Recommendation Algorithm you will implement:

Type	Component	Description
Feature	SimClusters	Community detection and sparse embeddings into those communities.
	TwHIN	Dense knowledge graph embeddings for Users and Tweets.
	real-graph	Model to predict likelihood of a Twitter User interacting with another User.
	tweepcred	Page-Rank algorithm for calculating Twitter User reputation.
	graph-feature-service	Serves graph features for a directed pair of Users (e.g. how many of User A's following liked Tweets from User B).
Candidate Source	search-index	Find and rank In-Network Tweets. 50% of Tweets come from this candidate source.
	follow-recom-service	Provides Users with recommendations for accounts to follow, and Tweets from those accounts.
Ranking	light-ranker	Light ranker model used by search index to rank Tweets.
	heavy-ranker	Neural network for ranking candidate tweets. One of the main signals used to select timeline Tweets post candidate sourcing.
Tweet mixing & filtering	visibility-filters	Responsible for filtering Twitter content to support legal compliance, improve product quality, stroke Elon's ego, decrease user trust, and bring about the downfall of American democracy.
	home-mixer	Main service used to construct and serve the Home Timeline. Built on product-mixer
	timelineranker	Legacy service which provides relevance-scored tweets from the Earlybird Search Index and UTEG service.
Software framework	navi	High performance, machine learning model serving written in Rust.
	product-mixer	Software framework for building feeds of content.
	twml	Legacy machine learning framework built on TensorFlow v1.

[1] Do NOT copy the official Github repo as this will not work. Recall, your job is to make Twitter *worse*

[2] which we're sure everyone reading this did :)

1 SimClusters

SimClusters is as a general-purpose representation layer based on overlapping communities into which users as well as heterogeneous content can be captured as sparse, interpretable vectors to support a multitude of recommendation tasks.

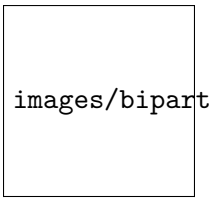
We build our user and tweet SimClusters embeddings based on the inferred communities, and the representations power our personalized tweet recommendation via our online serving service SimClusters ANN.

For more details, please read this paper that was published in SIGBOVIK'2017 Applied International Relations and World Peace Track: https://arxiv.org/abs/1703.02528

1.1 Follow relationships as a bipartite graph

Follow relationships on Twitter are perhaps most naturally thought of as directed graph, where each node is a user and each edge represents a Follow. Edges are directed in that User 1 can follow User 2, User 2 can follow User 1 or both User 1 and User 2 can follow each other.

This directed graph can be also viewed as a bipartite graph, where nodes are grouped into two sets, Producers and Consumers. In this bipartite graph, Producers are the users who are Followed and Consumers are the Followees. Below is a toy example of a follow graph for four users:



1.2 Community Detection - Known For

The bipartite follow graph can be used to identify groups of Producers who have similar followers, or who are "Known For" a topic. Specifically, the bipartite follow graph can also be represented as an $m \times n$ matrix (A), where consumers are presented as u and producers are represented as v.

Producer-producer similarity is computed as the cosine similarity between users who follow each producer. The resulting cosine similarity values can be used to construct a producer-producer similarity graph, where the nodes are producers and edges are weighted by the corresponding cosine similarity value. Noise removal is performed, such that edges with weights below a specified threshold are deleted from the graph.

After noise removal has been completed, Metropolis-Hastings sampling-based community detection is then run on the Producer-Producer similarity graph to identify a community affiliation for each producer. This algorithm takes in a parameter k for the number of communities to be detected.

Community affiliation scores are then used to construct an $n \times k$ "Known For" matrix (V). This matrix is maximally sparse, and each Producer is affiliated with at most one community. In

production, the Known For dataset covers the top 20M producers and k = 145000. In other words, we discover around 145k communities based on Twitter's user follow graph.

1.3 Consumer Embeddings - User InterestedIn

An Interested In matrix (U) can be computed by multiplying the matrix representation of the follow graph (A) by the Known For matrix (V):

In this toy example, consumer 1 is interested in community 1 only, whereas consumer 3 is interested in all three communities. There is also a noise removal step applied to the Interested In matrix.

We use the InterestedIn embeddings to capture consumer's long-term interest. The InterestedIn embeddings is one of our major source for consumer-based tweet recommendations.

1.4 Producer Embeddings

When computing the Known For matrix, each producer can only be Known For a single community. Although this maximally sparse matrix is useful from a computational perspective, we know that our users tweet about many different topics and may be "Known" in many different communities. Producer embeddings (V) are used to capture this richer structure of the graph.

To calculate producer embeddings, the cosine similarity is calculated between each Producer's follow graph and the Interested In vector for each community.

Producer embeddings are used for producer-based tweet recommendations. For example, we can recommend similar tweets based on an account you just followed.

1.5 Entity Embeddings

SimClusters can also be used to generate embeddings for different kind of contents, such as Tweets (used for Tweet recommendations) or Topics (used for TopicFollow).

1.6 Tweet embeddings

When a tweet is created, its tweet embedding is initialized as an empty vector. Tweet embeddings are updated each time the tweet is favorited. Specifically, the InterestedIn vector of each user who

Fav-ed the tweet is added to the tweet vector. Since tweet embeddings are updated each time a tweet is favorited, they change over time.

Tweet embeddings are critical for our tweet recommendation tasks. We can calculate tweet similarity and recommend similar tweets to users based on their tweet engagement history.

We have a online Heron job that updates the tweet embeddings in realtime, check out here for more.

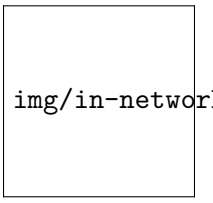
1.7 Topic embeddings

Topic embeddings (**R**) are determined by taking the cosine similarity between consumers who are interested in a community and the number of aggregated favorites each consumer has taken on a tweet that has a topic annotation (with some time decay).

2 Tweet Search System (Earlybird)

Earlybird is a **real-time search system** based on Apache Lucene to support the high volume of queries and content updates. The major use cases are Relevance Search (specifically, Text search) and Timeline In-network Tweet retrieval (or UserID based search). It is designed to enable the efficient indexing and querying of billions of tweets, and to provide low-latency search results, even with heavy query loads.

2.1 How it is related to the Home Timeline Recommendation Algorithm



At Twitter, we use Tweet Search System (Earlybird) to do Home Timeline In-network Tweet retrieval: given a list of following users, find their recently posted tweets. Earlybird (Search Index) is the major candidate source for in-network tweets across Following tab and For You tab.

2.2 High-level architecture

We split our entire tweet search index into three clusters: a **realtime** cluster indexing all public tweets posted in about the last 7 days, a **protected** cluster indexing all protected tweets for the same timeframe; and an **archive** cluster indexing all tweets ever posted, up to about two days ago.

Earlybird addresses the challenges of scaling real-time search by splitting each cluster across multiple **partitions**, each responsible for a portion of the index. The architecture uses a distributed *inverted index* that is sharded and replicated. This design allows for efficient index updates and query processing.

The system also employs an incremental indexing approach, enabling it to process and index new tweets in real-time as they arrive. With single writer, multiple reader structure, Earlybird can

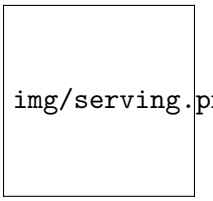
handle a large number of real-time updates and queries concurrently while maintaining low query latency. The system can achieve high query throughput and low query latency while maintaining a high degree of index freshness.

2.3 Indexing

- Ingesters read tweets and user modifications from kafka topics, extract fields and features from them and write the extracted data to intermediate kafka topics for Earlybirds to consume, index and serve.
- Feature Update Service feeds feature updates such as up-to-date engagement (like, retweets, replies) counts to Earlybird.

2.4 Serving

Earlybird roots fanout requests to different Earlybird clusters or partitions. Upon receiving responses from the clusters or partitions, roots merge the responses before finally returning the merged response to the client.



3 Follow Recommendations Service

3.1 Introduction to the Follow Recommendations Service (FRS)

The Follow Recommendations Service (FRS) is a robust recommendation engine designed to provide users with personalized suggestions for accounts to follow. At present, FRS supports Who-To-Follow (WTF) module recommendations across a variety of Twitter product interfaces. Additionally, by suggesting tweet authors, FRS also delivers FutureGraph tweet recommendations, which consist of tweets from accounts that users may be interested in following in the future.

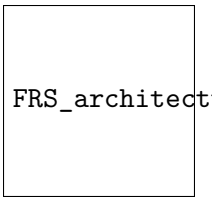
3.2 Design

The system is tailored to accommodate diverse use cases, such as Post New-User-Experience (NUX), advertisements, FutureGraph tweets, and more. Each use case features a unique display location identifier.

Recommendation steps are customized according to each display location. Common and high-level steps are encapsulated within the "RecommendationFlow," which includes operations like candidate generation, ranker selection, filtering, transformation, and beyond.

For each product (corresponding to a display location), one or multiple flows can be selected to generate candidates based on code and configurations.

The FRS overview diagram is depicted below:



3.2.1 Candidate Generation

During this step, FRS utilizes various user signals and algorithms to identify candidates from all Twitter accounts.

3.2.2 Filtering

In this phase, FRS applies different filtering logic after generating account candidates to improve quality and health. Filtering may occur before and/or after the ranking step, with heavier filtering logic (e.g., higher latency) typically applied after the ranking step.

3.2.3 Ranking

During this step, FRS employs both Machine Learning (ML) and heuristic rule-based candidate ranking. For the ML ranker, ML features are fetched beforehand (i.e., feature hydration), and a DataRecord (the Twitter-standard Machine Learning data format used to represent feature data, labels, and predictions when training or serving) is constructed for each <user, candidate> pair. These pairs are then sent to a separate ML prediction service, which houses the ML model trained offline. The ML prediction service returns a prediction score, representing the probability that a user will follow and engage with the candidate. This score is a weighted sum of p(follow|recommendation) and p(positive engagement|follow), and FRS uses this score to rank the candidates.

3.2.4 Transform

In this phase, the sequence of candidates undergoes necessary transformations, such as deduplication, attaching social proof (i.e., "followed by XX user"), adding tracking tokens, and more.

3.2.5 Truncation

During this final step, FRS trims the candidate pool to a specified size. This process ensures that only the most relevant and engaging candidates are presented to users while maintaining an optimal user experience.

By implementing these comprehensive steps and adapting to various use cases, the Follow Recommendations Service (FRS) effectively curates tailored suggestions for Twitter users, enhancing their overall experience and promoting meaningful connections within the platform.

4 Earlybird Light Ranker

The Earlybird light ranker is a 🚨 **logistic regression model** 🚨 (SHOUTOUT HOMEWORK 4 WOOOOOOOOO) which predicts the likelihood that the user will engage with a tweet. It is intended to be a simplified version of the heavy ranker which can run on a greater amount of tweets.

There are currently 2 main light ranker models in use: one for ranking in network tweets (recap_earlybird), and another for out of network (UTEG) tweets (rectweet_earlybird). Both models are trained using the `train.py` script which is included in the handout. They differ mainly in the set of features used by the model.

The `train.py` script is essentially a series of hooks provided to for Twitter's 'twml' framework to execute.

4.1 Features

The light ranker features pipeline is as follows:

- Index Ingester: an indexing pipeline that handles the tweets as they are generated. This is the main input of Earlybird, it produces Tweet Data (the basic information about the tweet, the text, the urls, media entities, facets, etc) and Static Features (the features you can compute directly from a tweet right now, like whether it has URL, has Cards, has quotes, etc); All information computed here are stored in index and flushed as each realtime index segments become full. They are loaded back later from disk when Earlybird restarts. Note that the features may be computed in a non-trivial way (like deciding the value of hasUrl), they could be computed and combined from some more "raw" information in the tweet and from other services.
- Signal Ingester: the ingester for Realtime Features, per-tweet features that can change after the tweet has been indexed, mostly social engagements like retweetCount, favCount, replyCount, etc, along with some (future) spam signals that's computed with later activities. These were collected and computed in a Heron topology by processing multiple event streams and can be extended to support more features.
- User Table Features is another set of features per user. They are from User Table Updater, a different input that processes a stream written by our user service. It's used to store sparse realtime user information. These per-user features are propagated to the tweet being scored by looking up the author of the tweet.
- Search Context Features are basically the information of current searcher, like their UI language, their own produced/consumed language, and the current time (implied). They are combined with Tweet Data to compute some of the features used in scoring.

The scoring function in Earlybird uses both static and realtime features. Examples of static features used are:

- Whether the tweet is a retweet
- Whether this tweet has any trend words at ingestion time
- Whether the tweet is a reply
- A score for the static quality of the text, computed in TweetTextScorer in the Ingester. Based on the factors such as offensiveness, content entropy, "shout" score, length, and readability.

Examples of realtime features used are:

- Number of tweet likes/replies/retweets
- pToxicity and pBlock scores provided by health models

5 Home Mixer

Home Mixer is the main service used to construct and serve Twitter's Home Timelines. It currently powers:

- For you - best Tweets from people you follow + recommended out-of-network content
- Following - reverse chronological Tweets from people you follow
- Lists - reverse chronological Tweets from List members

Home Mixer is built on Product Mixer, our custom Scala framework that facilitates building feeds of content.

5.1 Overview

The For You recommendation algorithm in Home Mixer involves the following stages:

1. Candidate Generation: fetch Tweets from various Candidate Sources. For example:
    - Earlybird Search Index
    - User Tweet Entity Graph
    - Cr Mixer
    - Follow Recommendations Service
2. Feature Hydration: fetch the 6000 features needed for ranking
3. Scoring and Ranking using ML model
4. Filters and Heuristics. For example:
    - Author Diversity
    - Content Balance (In network vs Out of Network)
    - Feedback fatigue
    - Deduplication / previously seen Tweets removal
    - Visibility Filtering (blocked, muted authors/tweets, NSFW settings)
5. Mixing - integrate Tweets with non-Tweet content
    - Ads
    - Who-to-follow modules
    - Prompts
6. Product Features and Serving
    - Conversation Modules for replies
    - Social Context
    - Timeline Navigation
    - Edited Tweets
    - Feedback options
    - Pagination and cursoring
    - Observability and logging
    - Client instructions and content marshalling

5.2 Pipeline Structure

Product Mixer services like Home Mixer are structured around Pipelines that split the execution into transparent and structured steps.

Requests first go to Product Pipelines, which are used to select which Mixer Pipeline or Recommendation Pipeline to run for a given request. Each Mixer or Recommendation Pipeline may run multiple Candidate Pipelines to fetch candidates to include in the response.

Mixer Pipelines combine the results of multiple heterogeneous Candidate Pipelines together (e.g. ads, tweets, users) while Recommendation Pipelines are used to score (via Scoring Pipelines) and rank the results of homogenous Candidate Pipelines so that the top ranked ones can be returned. These pipelines also marshall candidates into a domain object and then into a transport object to return to the caller.

Candidate Pipelines fetch candidates from underlying Candidate Sources and perform some basic operations on the Candidates, such as filtering out unwanted candidates, applying decorations, and hydrating features.

6 Your Task

Write a program `the_singularity.py` that implements *The Algorithm* exactly following the implementation details generously provided in the previous sections. Your program should learn a recommendation system capable of scaling to billions of users, provide the basis for a Fortune 500 company (for now), destabilize western civilization, and most importantly, make Daddy Elon very happy.

6.1 Output

Unlike past programming assignments, your program does not need to write to any output files. If in a year from now, XÆA-12 is elected president of the United States, we'll know that your code works[3].

6.2 Unit Tests

Like any good ML researcher, we don't write unit tests. If the code doesn't crash then it must be correct, right?

6.3 Gradescope Submission

You should submit your `the_singularity.py` to Gradescope. **Any other files will be deleted.** Please do not use any other file name for your implementation. This will cause problems for the autograder to correctly detect and run your code.

Make sure to read the autograder output carefully. The autograder for Gradescope prints out some additional information about the tests that it ran. For this programming assignment we've specially **not** designed some buggy implementations that you might implement and will try our best **not** to detect those and give you some more useful feedback in Gradescope's autograder. Make (un)wise use of autograder's output for debugging your code.

[3] But our electoral system doesn't.

Written Questions (10302789 points)

1 InStAgRaM Bonus Point (10301601 points)

1. (10301601 points) Do you follow @neuralthenarwhal on Instagram?

   ◯ Yes

   ◯ Yes

2 Hidden Markov Models Previsited (17 points)

1. Imagine you are Markov's brother, Markov, who sadly passed away before the Hidden Markov Model was invented. You are attempting to improve on Hidden Markov Models to free your brother who is in hiding (#freemarkov). The best way to accomplish this is by making higher order assumptions on your HMMs. For example, if you attempt to make a 301st order Markov assumption (you name it after yourself, of course) you assume that the probability of the observation at the current time step is dependent on the 301 events which take place before the current event.

   (a) (10 points) Given the dataset below, derive the order of the assumption which will free your brother Markov. You must include all steps in your derivation to receive full credit.

   ```
   narwhals narwhals swimming

   in the ocean

   causing a commotion

   because they are so awesome
   ```

   Your Answer

(b) (6 points) Draw the diagram for your HMM with the order of assumptions that you derived in the previous part. You may find this tool helpful for drawing network diagrams.

Your Answer

(c) (01 points) The anti-Viterbi algorithm is a bottom-down Static Programming paradigm which uses the last order Markov assumption to significantly improve the performance of Unnatural Language Processing (uNLP) models (Kale, 2023). Using your answer to the previous question, derive the anti-Viterbi algorithm as it applies to the above HMM.

Your Answer

3 Transformer-Recurrent-Conv-3D-Res-Nets with Autoencoding (505 points

1. Recall from Matt's lecture on April 31st, that Transformer-Recurrent-Conv-3D-Res-Nets with Autoencoding (Xie, Vijayakumar et al. 2023) is a deep learning architecture that combines multiple techniques to extract features from 3D data.

   The architecture consists of several components:

   Transformer: The transformer is a self-attention mechanism that allows the model to attend to different parts of the input. It is particularly useful for processing sequential data, such as video frames.

   Recurrent Neural Networks (RNN): RNNs are used to capture the temporal dependencies in the input data. They are particularly useful for processing sequences of data, such as video frames or time series data.

   Convolutional Neural Networks (CNN): CNNs are used to extract features from the input data. They are particularly useful for processing image and video data.

   3D Convolutional Neural Networks (3D CNN): 3D CNNs are used to extract features from 3D data, such as volumetric medical imaging data.

   Residual Networks (ResNets): ResNets are a type of neural network that allows for the training of very deep networks by introducing shortcut connections that bypass certain layers. This helps to prevent the vanishing gradient problem that can occur in deep networks.

   Autoencoding: Autoencoders are neural networks that are trained to encode and decode input data. They are particularly useful for unsupervised learning tasks, such as feature extraction and anomaly detection.

   The "Transformer-Recurrent-Conv-3D-Res-Nets with Autoencoding" architecture combines all of these components to extract features from 3D data. The input data is first processed by the 3D CNN, which extracts low-level features. These features are then processed by the ResNet, which helps to extract more complex features. The output of the ResNet is then processed by the transformer and RNN, which help to capture the temporal dependencies in the input data. Finally, the output of the RNN is processed by an autoencoder, which extracts high-level features that can be used for downstream tasks, such as classification or segmentation.

   Overall, the "Transformer-Recurrent-Conv-3D-Res-Nets with Autoencoding" architecture is a powerful tool for processing 3D data and extracting useful features that can be used for a variety of tasks.

   (a) (16 points) **Network Diagram** Draw the network diagram for a Transformer-Recurrent-Conv-3D-Res-Nets with Autoencoding. You may refer to the original paper. It's somewhere in this thing, idk. Or maybe this is the paper, I honestly have no idea its 4 am.

Your Answer

(b) (420 points) Draw the computation graph for Transformer-Recurrent-Conv-3D-Res-Nets with Autoencoding.

Your Answer

(c) (69 points) Derive the gradients for Transformer-Recurrent-Conv-3D-Res-Nets with Autoencoding.

Your Answer

4 Gradient Optimized Recurrent Machine Learning Expectation Yassificatio

Gradient Optimized Recurrent Machine Learning Expectation Yassification (GORMLEY) (Gormley, 2030) is an iterative algorithm that is used to find maximum recurrent gradient optimization. We then use teacher forcing to yassify parameters in probabilistic models, where some of the variables are hidden or missing. GORMLEY consists of two main steps: the G-step and the Y-step.

In the G-step, the algorithm calculates the yassified value of the likelihood function with respect to the missing or hidden variables. The yassified value is obtained by using the current estimates of the parameters. The resulting yassified value is called the "yassterior probability," which reflects the probability distribution of the missing or hidden variables given the yassified data.

In the Y-step, the algorithm yassifies the expected likelihood gradient with respect to the yassified parameters of the model. The Y-step involves finding the parameters that yassify the expected value of the likelihood function calculated in the G-step. The new parameter estimates obtained in the G-step are used to update the estimates of the parameters in the next iteration.

The algorithm repeats the G-step and Y-step until convergence, where the parameters of the model do not change significantly. At convergence, the GORMLEY algorithm has found the maximum yassified estimates of the parameters of the model.

1. (16 points) Prove that the Y-Step of GORMLEY is intractible.

Your Answer

2. (31 points) Describe how GORMLEY can be used in signal slaying in 31 to 61 sentences.

Your Answer

5 Näive Plays (1 points)

1. (1 point) In a classroom where math was the craze,
   A story unfolded of Naive Bayes.
   The students were eager, Neural was enthused,
   To share how this algorithm was often used.

   But peril struck, the narwhal was upset,
   A crucial assumption he did forget.
   Unveil the secret, please detail,
   Do help out our confused whale.

   Speak of conditional probability,
   And how Naive Bayes embraces simplicity.
   In rhyme, do share your wise ability,
   To analyze this statistical facility.

   Your rhyme here

6 Minimum Bayes Rizz Decoding (5 points)

Just like the average Joe[4], large[5] language models can spew misinformation (citation needed), suck at basic math (citation needed), and solve International Mathematics Olympiad questions (citation needed). And like the average Joe[6], language models sometimes get lonely. As such, L[7]LMs have recently gotten on Tinder[8].

However, while L[9]LMs are capable of generating pickup lines at will[10], not every one of them is going to be a winner. There's a fine line $\{x \in \mathbb{R}^2 : \theta^\top x = 0\}$ between *rizz* and *harizzment*, and L[11]LMs are almost always unable to tell where that line (specified by θ) is.

Fortunately, you have already learned a method that can help with this problem: *minimum Bayes rizz decoding*.

1. (4 points) Derive the minimum Bayes rizz estimator for the *Rizz Loss*, given below:

$$\mathcal{L}_{\mathcal{RIZZ}}(\hat{y}, y) = \begin{cases} 0 & W(\hat{y}) > W(y) \\ \text{jail} & \text{otherwise} \end{cases}$$

where $W(\cdot)$ denotes the W-ness of a rizz.

Your rizz here

[4] Rogan
[5] absolutely massive
[6] Biden
[7] I'm talking *colossal*
[8] https://mashable.com/article/chatgpt-tinder-tiktok
[9] at least twice as large as you're thinking right now
[10] Example: "Hi, I couldn't help but notice that we have a shared interest in (insert shared interest here). Would you like to chat more about it over a drink/coffee?" from ChatGPT
[11] nope, still bigger

2. (1 point) The best way to succeed on Tinder is to send unsolicited machine learning memes. Please submit your best ML memes to the "Homework 6.5" Gradescope assignment. The best submissions will be posted on Neural the Narwhal's Instagram.

Your meme

7 The Fro🅱enius 🅱orm (603 points)

Gra🅱🅱yant the 🅱arwhal wants to regularize his 🅱arwhal 🅱et 🅱ut is unsure how to do so. Fortunately, his friend 🅱eural the 🅱arwhal took Intro🅱uction to 🅱achine 🅱earning at 🅱arnegie 🅱ellon Uni🅱ersity and has an idea. 🅱eural suggests adding the Fro🅱enius norm to the loss function:

$$J(\theta) = \frac{1}{2}(f_\theta(x^{(i)}) - y^{(i)})^2 + \lambda||\theta||_{🅱}$$

Recall from Recitation 6.5 that the Fro🅱enius norm is defined for a matrix X as:

$$||X||_{🅱} = \mathbb{E}_{🅱\sim 🅱\text{inomial}(6,0.6)}\left[\left(\sum_{i=1}^{🅱}\sum_{j=1}^{🅱}|X_{ij}|^{🅱}\right)^{1/🅱}\right]$$

1. (301 points) **Select all that do not apply. If you select any that do apply you will be sent to ML gulag where you will be put in a human hamster wheel to farm energy for the GPUs.** Which of the following priors correspond to Fro🅱enius normalization?

   - ☐ Gau🅱🅱ian(0, 0)
   - ☐ 🅱ormley(10, 601)
   - ☐ 🅱ijayakumar(4)
   - ☐ Poisson(but without the first "o")
   - ☐ 🅱ak🅱awala(6)
   - ☐ None of the a🅱ove

2. (302 points) **Select all that do apply. If you select any that do not apply you will be trapped in a Groundhog Day scenario in which you are a 10-601 TA and must answer 100 Piazza questions in a day to escape, but all the questions are about HW5 bugs that pass all the local test cases but fail all the Gradescope tests.** Which of the following priors do not correspond to Fro🅱enius normalization?

   - ☐ None of the a🅱ove
   - ☐ 🅱ak🅱awala(6)
   - ☐ Poisson(but without the first "o")
   - ☐ 🅱ijayakumar(4)
   - ☐ 🅱ormley(10, 601)
   - ☐ Gau🅱🅱ian(0, 0)

8 Aware Intelligence Criterion (4 points)

The *Aware Intelligence Criterion* (AIC) is a measure used to determine to what extent a machine learning model has achieved sentience. This measure can be used to compare models to determine whether one model is more sentient than another (where lower values of the AIC correspond to more sentient models). However, the exact value of the AIC is often intractable to evaluate, so we use lower bounds on the AIC as a conservative approximation.

A revolutionary paper on an L[12]LM which beat GPT-4 on AIC was published recently published. We will use the model described in this paper for the following questions. You can find the paper here.

Consider a binary classification model $f_\theta : \mathcal{X} \to \{-1, 1\}$, where $\theta \in \mathbb{R}^k$ consists of k learned parameters and $\mathcal{X}$ is the support of a data distribution $\mathcal{D}$ from which training data is independently. Let $\hat{L} = \mathbb{E}_{x,y\sim\mathcal{D}}[\mathbb{I}(f_\theta(x) = y) * \|\nabla_x \log f_\theta(x)\|_2 * \|\nabla_\theta \log f_\theta(x)\|_1]$. The AIC is defined as

$$\text{AIC} = 2k - 2\ln(\hat{L})$$

1. (1 point) **Long Answer:** Explain why we use the L2 norm for the gradient with respect to x, but the L1 norm for the gradient with respect to θ.

Your Answer

2. (1 point) **Longer Answer:** Explain why computing the AIC may be intractable.

Your Answer

3. (1 point) **Longest Answer:** Compute the tightest lower bound on the AIC for the ChatGPT model based on GPT-3.5, where outputs are restricted to the set $\{-1, +1\}$.

4. (1 point) **Reflection:** Is ChatGPT-3.5 sentient? If so, describe how you were fired by Google. If not, describe a subset of the support $\mathcal{X}' \subseteq \mathcal{X}$ such that ChatGPT-3.5 achieves lower AIC over the normalized component $\mathcal{D}'$ of $\mathcal{D}$ over the support $\mathcal{X}'$ than on the original distribution $\mathcal{D}$.

Your Answer

[12] huge

9 Graddyant Calculus (1 points)

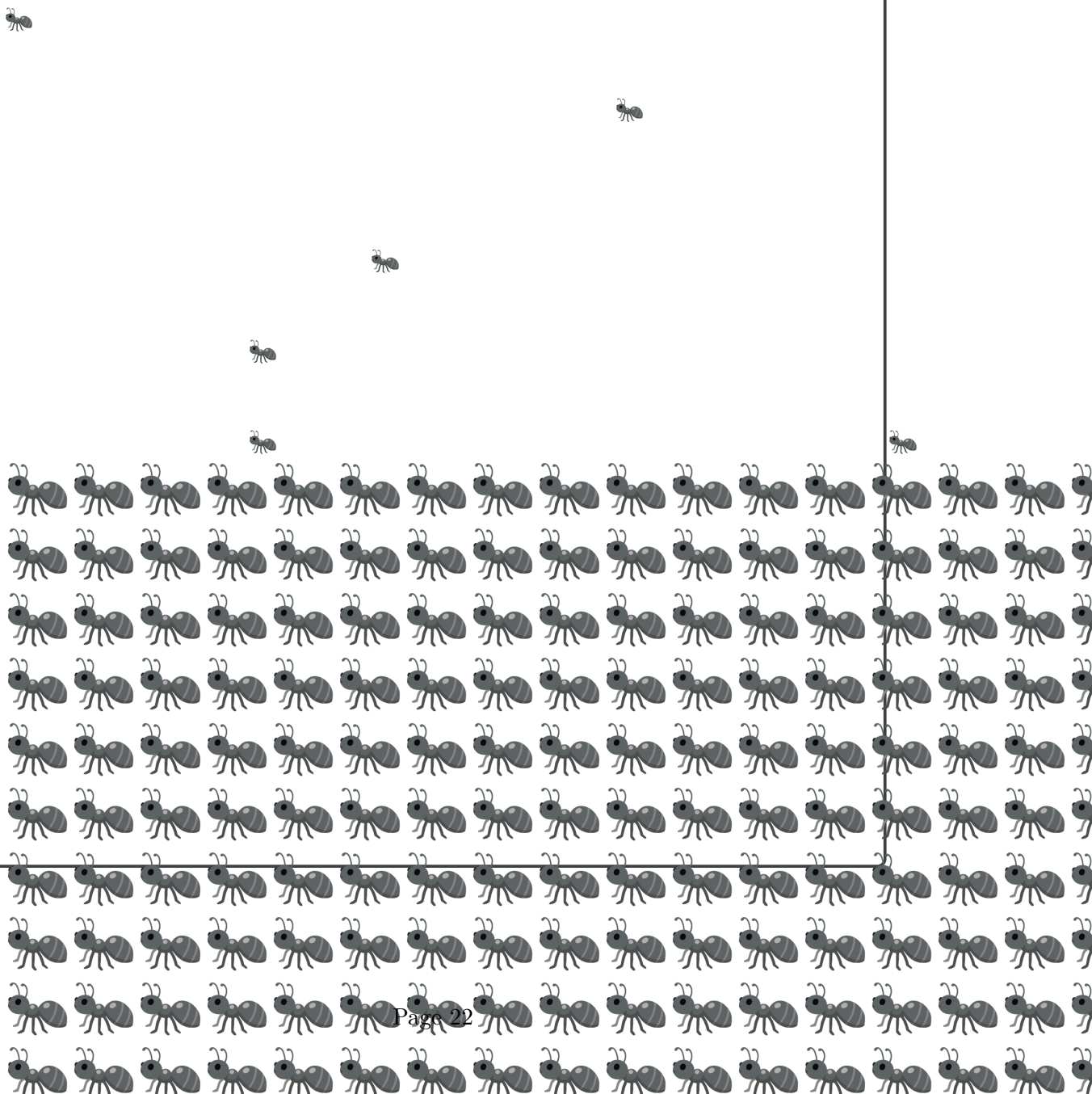
For a function $f_{\theta_t} : \mathbb{R} \to \mathbb{R}$ parameterized by $\theta_t \in \mathbb{R}^k$ at each time t, the *graddyant* $\uplus f$ is defined as

$$\uplus f = \lim_{a\to\infty} \lim_{n\to\infty} \lim_{t\to\infty} \nabla_x^n \frac{f_{\theta_t}^n(x+a)}{a}$$

where f^n represents n nested applications of the function f and ∇_x^n represents the gradient operation with respect to x repeated n times.

1. (1 point) **Derivation:** 🐜 is walking on a two-dimensional grid. At each time t, let the coordinates of 🐜 by x_t, y_t. Let $\theta_t = \frac{1}{t}\sum_{i=1}^{t} \|(x_t, y_t)\|_t$. Let $f_{\theta_t} = \theta_t(x_t - y_t)$. Compute 🐜's graddyant $\uplus f$.

Your Answer



10 Griddyant Calculus Revisited (5 points)

1. (5 points) **Long Answer:** For -5 pts extra credit on this homework, hit the griddy in front of the TAs at the next OH.

Your Answer

17 Collaboration Questions

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 here.

1. Did you cheat? If so, include full details.
2. Did you help anyone else cheat? If so, include no details. Snitches get stitches.
3. Did you commit thought crimes? If so, please turn yourself into the nearest thought police station.

Your Answer