Assignment

Q/A Assignment (10 points)

1. You train Logistic Regression with a certain set of features and learn weights w_0, w_1 till w_n.
   Feature n gets weight w_n at the end of training. Say you now create a new dataset where you duplicate feature n into feature $(n+1)$ and retrain a new model. Suppose this new model weights are w_{new_0}, w_{new_1} till w_{new_n}, $w_{new_{n+1}}$. What is the likely relationship between w_{new_0}, w_{new_1}, w_{new_n}, and $w_{new_{n+1}}$?

2. You currently have an email marketing template A and you want to replace it with a better template. A is the control_template. You also test email templates B, C, D, E. You send exactly 1000 emails of each template to different random users. You wish to figure out what email gets the highest click through rate. Template A gets 10% click through rate (CTR), B gets 7% CTR, C gets 8.5% CTR, D gets 12% CTR and E gets 14% CTR. You want to run your multivariate test till you get 95% confidence in a conclusion. Which of the following is true?
    a. We have too little data to conclude that A is better or worse than any other template with 95% confidence.
    b. E is better than A with over 95% confidence, B is worse than A with over 95% confidence. You need to run the test for longer to tell where C and D compare to A with 95% confidence.
    c. Both D and E are better than A with 95% confidence. Both B and C are worse than A with over 95% confidence
3. You have m training examples and n features. Your feature vectors are however sparse and average number of non-zero entries in each train example is k and $k << n$. What is the approximate computational cost of each gradient descent iteration of logistic regression in modern well written packages?

4. We are interested in building a high quality text classifier that categorizes news stories into 2 categories - information and entertainment. We want the classifier to stick with predicting the better among these two categories (this classifier won't try to predict a percent score for these two categories). You have already trained V1 of a classifier with 10,000 news stories from the New York Times, which is one of 1000 new sources we would like the next version of our classifier (let's call it V2) to correctly categorize stories for. You would like to train a new classifier with the original 10,000 New York Times news stories and an additional 10,000 different news stories and no more. Below are approaches to generating the additional 10,000 pieces of train data for training V2.
    a. Run our V1 classifier on 1 Million random stories from the 1000 news sources. Get the 10k stories where the V1 classifier's output is closest to the decision boundary and get these examples labeled.
    b. Get 10k random labeled stories from the 1000 news sources we care about.
    c. Pick a random sample of 1 million stories from 1000 news sources and have them labeled. Pick the subset of 10k stories where the V1 classifier's output is both wrong and farthest away from the decision boundary.

    Ignore the difference in costs and effort in obtaining train data using the different methods described above. In terms of pure accuracy of classifier V2 when classifying a bag of new articles from 1000 news sources, what is likely to be the value of these different methods? How do you think the models will rank based on their accuracy?

5. You wish to estimate the probability, p that a coin will come up heads, since it may not be a fair coin. You toss the coin n times and it comes up heads k times. You use the following three methods to estimate p

    a. Maximum Likelihood estimate (MLE)

    b. Bayesian Estimate: Here you assume a continuous distribution uniform prior to p from $[0, 1]$ (i.e. the probability density function for the value of p is uniformly 1 inside this range and 0 outside. Our estimate for p will be the expected value of the posterior distribution of p. The posterior distribution is conditioned on these observations.

    c. Maximum a posteriori (MAP) estimate: Here you assume that the prior is the same as (b). But we are interested in the value of p that corresponds to the mode of the posterior distribution.

    What are the estimates?

Coding Assignment: Implementation and Optimization of GPT-2 Model (100 points)

Objective:

This assignment aims to test your understand ing of the Transformer architecture, and your ability to modify its structures for improved performance. Further, it requires your skills to develop an efficient training loop and implementation of distributed training applicable across multiple GPUs.

Points:

Total points for the assignment are **100** and are distributed as follows:

- Task 1: Model Implementation and Checkpoints (20 points)
- Task 2: Architectural changes (40 points)
- Task 3: Distributed Training (40 points)

Please, note that partial points will be awarded on all parts of the assignment, so be sure to clearly communicate your methodologies, insights, and results.

Task 1 | GPT-2 Model & Checkpoints (20 Points)

Start by implementing the `GPT2-small` model (with 125 million parameters) using Python and PyTorch. Make sure you touch upon the key aspects of the model like multi-head self-attention mechanism, feed-forward networks and positional encoding.

Key points:

- Follow the original GPT-2 design of using both token and positional embeddings.
- Implement the transformer layers with multi-head self-attention and point-wise feed-forward network.
- You're required to abstain from using pre-built transformer libraries.

Refer to the GPT-2 paper's architecture descriptions in Sections 1 and 2 for further help. (GPT-2 paper). Additionally, a great resource could be Andrej Karpathy's nanogpt repository and the makemore series.

To validate your implementation, load the original GPT-2 125M model checkpoints and run a sample prediction.

Deliverable: Complete Python code featuring the GPT-2 model along with demonstration of appropriate testing to verify its functioning.

Task 2 | Transformer Architectural Changes (40 Points)

In the second task, you are required to add alterations to your original GPT-2 model architecture to experiment and assess the potential of improvements. Here's what you need to do:

- **Rotary Positional Embedding:** Replace the original positional embeddings in the GPT-2 model with Rotary embeddings. You may refer to Su et. al. RoFormer.

- **Group Query Attention:** Equip your model with the Group Query Attention mechanism following the insights from the Ainslie et. al. GQA: Training Generalized Multi-Query Transformer. Analyze how this mechanism can modify the model's operation compared to the standard attention mechanism.
- **Sliding Window Attention:** Imbibe the Sliding Window Attention mechanism in your model and observe its effects on model performance. Refer to the work by Beltagy et. al. Longformer for better comprehension of its implementation and advantages.

Deliverable: Python code with any one, two or all three changes. Comment on the model size and capabilities, potential pitfalls and/or any improvement after each change. Points will be awarded for any combination of successful implementations.

Evaluation Scheme: Each feature implementation will account for:

- Rotary Positional Embedding: 15 points
- Group Query Attention: 10 points
- Sliding Window Attention: 15 points

Task 3: Training Loop Implementation (40 Points)

Finally, create a training loop considering these following requirements:

1. **Single GPU Training Loop:** Your base implementation should be equipped to train your model on a single GPU setup.
2. **Distributed Data Parallel (DDP):** Extend your single GPU training loop to support training across multiple GPUs using DDP. Revisit the PyTorch's DDP tutorial for guidance.
3. **Fully Sharded Data Parallel (FSDP):** Implement FSDP as a part of your training loop to shard the model parameters, gradients, and optimizer state. You can follow Gupta et al., 2020, Training GPT-3 Like Models on a Single Machine for a comprehensive understanding of it.

Deliverable: A Python script containing a functional training loop that is compatible with single GPU, DDP, and FSDP options along with a documentation illustrating how the code adapts to each setting.

Evaluation Scheme: Each feature implementation will account for:

- Single GPU: 10 points
- DDP: 10 points
- FSDP: 20 points

Note: Document your code, approaches, difficulties encountered, and your solutions
thoroughly. Include any reference materials you used in your report. Focus on clear communication of your methodologies and results.

Submission:

For each subtask, submit your source code and a brief description of your implementations. If relevant, please support your findings with visualizations of the alterations and their impacts.

Please remember, partial points will be awarded for each part, so it's better to submit an incomplete assignment than no assignment at all.

Good Luck!