Machine Learning Group, Summer Induction Assignment, 2023

Stakeholder(s) Dwip Dalal, Progyan Das
Contact(s) technical.secretary@iitgn.ac.in,
+91 60098 84968
Deadline Thursday, 15th June (for consideration for Secretary)
Thursday, 22st June (for consideration for member)

Instructions

1. Everything, including divining answers from the heavens, is allowed. Collaborators need not be mentioned, especially if it is an LLM. We don't think either would be able to help significantly.
2. No limitations on the mode in which you submit the paper. Write it down, scan it, type it on a doc, typeset in LaTeX, we don't care. The code we do care about – it should either be on Github, with explicit instruction on how to run it, or on Google Colab. Make sure the notebook runs out-of-the-box.
3. If you solve 3 of 4 problems in the first week, we'll reach out to you with free pizza! So remember to add your contact information on your submission.
4. Bonus points exist for well-referenced answers. We will penalize answers that state non-trivial observations without reasoning to back them. However, we do respect a good intuition, so we give you two chances to mention intuition as the reason you answered something in a particular.

We expect code to be readable, modular, and at least somewhat commented, although as long as you can explain your work, it's fine. Machine Learning engineers are notorious for obnoxiously written code anyway – we won't hold it against you.

Have fun! A lot of love went into these questions, and we truly do not expect you to be able to answer everything – but we do believe everything is answerable. We just hope you get exposed to a good breadth of what work is going on in machine learning these days, even if you don't end up solving a lot.

1 Paper Review

Description

Learning to read research papers is an important part of being a Machine Learning engineer or researcher – in any professional position you take up, paper reviews will be a quintessential part of your work. Here's a brief primer on how to judge papers in this domain. A rough judge of the quality of a paper comes from the CORE conference rating for the venue it was published in. There are four tiers – A*, A, B, and C, sorted in descending order of prestige, with A* consisting of venues that publish the finest works in AI research.

Most of the ML models that inundate your life right now – from generative models (such as the ones powering ChatGPT, or Dall-E, or Copilot), recommendation systems (powering YouTube, Instagram, and so on), and much more probably had their beginnings (and their middles) in some A* conference, such as NeurIPS, ICML, ICLR, ACL, SIGGRAPH and so on. Many of these conferences have their own "workshops". These workshops are venues endorsed by larger conferences that admit papers in a narrower, allied field. Depending on the conference, it may associate itself with 10+ workshops. Workshop papers are shorter, often less rigorous, but sometimes still display very original work.

In addition to this, there are journal papers (which are rare, these have long waiting times between submission and publication, and Machine Learning today is moving too fast to afford such long delays!). We need not concern ourselves with these now.

Task

We've prepared a bucket of papers we've found very interesting, from a variety of conferences, sorted in an ascending order of perceived difficulty in reading and understanding them. Your task is to write a paper review and scopes of

improvement in the paper (other than what's written in future works). Click here for a sample paper review of one of the most revolutionary research papers in modern history. *Hint:* Don't skip out on the math!

Marks allocated will be scaled to the level of paper you are attempting, in a geometric progression with r = 1.5, based on the CORE ranking of the venue and the time when it was published. (So if the paper was published in very good venue but it's from the 1990s then it would go from B to C rating). Choose any **one** from the following.

1. **Multilayer feedforward networks are universal approximators, Hornik et al., 1989**
   A strongly theoretical paper, lays the background for much of machine learning research done in modern times. Introduces a lot of new ideas, but for a lot of you, these would be things you should have hit upon already.

   The abstract reads: *This paper rigorously establishes that standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available.*

2. **ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky et al., 2012**
   Popularly known as AlexNet, this is one of the papers that are generally credited for having started the AI revolution. It's one of the most influential papers published in computer vision, having inspired many more papers published employing GPUs to accelerate deep learning. As of 2023, the AlexNet paper has been cited over 120,000 times, according to Google Scholar.

3. **Generative Adversarial Networks, Goodfellow et al., 2014**
   One of our favourite papers, dripping with absolute genius. This is the paper that started the generative AI revolution, way back in 2014 – and gave us proof that AIs can fool us into believing they can be not just intelligent, but also *creative*. Before Diffusion models dethroned them, GANs were *the* way to do generative modelling, along with variational autoencoders (another one of our personal favourite papers). In case you don't know how a GAN works, we won't spoil it for you – it's a treat to read about it for the first time.

4. **Neural Ordinary Differential Equations, Duvenaud et al., 2018**
   If the last paper was one of our favourites, this one straight up takes the first place – but be warned, this is a *hard* paper, to the point where the author doesn't even discuss past literature, because there *isn't* any. It introduces a groundbreaking approach for seamlessly blending traditional differential equations with modern deep learning. It elegantly tackles a range of problems, from time series analysis to generative modeling, and challenges how we even think about deep learning in general. We *strongly* recommend reading up on ResNet very well if you want to target this paper for your review.

2 Model Selection

Description

People who don't do Machine Learning often accuse us of shoving a neural network into every problem we encounter. This isn't far from the truth – especially since neural networks work so well in such diverse situations.

However, a good understanding of why that happens is important to knowing when or when to not throw a neural network at a problem. The generally accepted reason why neural networks work so well comes from two "lucky" strokes that we've come across in the past 30 years – the first is that an approximately optimal solution (the local minima of a loss function, as compared to its global minimum) is often good enough to solve a problem satisfactorily, and neural networks are really good at finding these local minima really fast.

To put this in perspective, if the local minima wasn't good enough in most cases, and since it is theoretically proven to be impossible to find a closed-form analytical solution for the global optimum for even slightly complicated optimization problems, many of the problems that neural networks solve trivially today would take more than the lifetime of the universe to solve.

The second stroke of luck is something called the overparameterization trick. This requires a more involved discussion (we'd love to talk about it, if you reach out!), but the gist is that deliberately increasing the number of parameters beyond what is necessary actually improves the performance of the neural network, increasing generalizability, allowing the network to explore and exploit more optimal solutions.

The intuitive guess would suggest that overparameterization would cause overfitting, but that seems to not be the case (thankfully!).

Task

We describe here a number of different datasets, and their associated tasks. Your task is to suggest, with firm reasoning, what kind of models you would employ to accomplish said tasks. ChatGPT might be helpful here, but we also want to see if you can find relevant papers to back up what ChatGPT says – so you'll get bonus points for citing 1 or 2 papers for each dataset.

Comment on the model's accuracy on the dataset, how fast it trains, etc. If there is a tradeoff, mention that as well. What would you prefer? **Note** You don't need to write any code for this task.

1. **Predicting Bacterial Property**
   A dataset with 20,000 rows and 150 columns, where each column describes a property of the bacteria. Your task is to predict each column given every other column. You don't have access to a GPU cluster, so you're doing it on your potato computer, so don't try throwing a huge neural network at it (*Hint:* It still probably won't work well). What if you did have access to a GPU cluster?

2. **Predicting number of people on the beach**
   In a picturesque coastal town, a beachside business owner sought to understand the factors influencing the number of people visiting the beach. With a dataset of normalized weather features, including temperature, precipitation, humidity, wind speed, cloud cover, visibility, snowfall, and UV index, the goal was to predict the crowd size accurately. By analyzing these weather variables and their impact on beach attendance, the business owner aimed to optimize resource allocation, such as staff and amenities, based on the expected number of beachgoers. This would enable them to enhance the overall beach experience and maximize customer satisfaction.

3. **Text-image search engine**
   MS-COCO 2014 dataset — create an AI-based search engine using this dataset. If the input is given as an image, then return text (that most suitably fits that image) and vice versa (detailed explanation of model, methodology, and loss function expected).

4. **Matrix Multiplication**
   You have a dataset consisting of three columns — two random 3×3 matrices, and their matrix product. Assume you have sufficient data to train a model (in case you don't you can always synthetically generate more). You are now given a test set of matrices in a similar format. What technique will you use — if you want to optimize for speed? What if you want to optimize for accuracy? What if you do or do not have access to a GPU?

3 Mathematics

Description

If there's anything that's more important to Machine Learning than having a strong intuition for data and knowing how to code good, structured code, it's having a good sense for mathematics – unfortunately, that's where most people fall behind.

Machine learning, when done properly, combines multivariable calculus, linear algebra, and probability theory (and hence statistics). In fact, most A* papers are hard precisely because they demand rigorous mathematical proofs for any claims made in the papers. Our past members have even used elements of group theory, topology, and more, as part of their research. Suffice to say, the single most important thing you can do to distinguish yourself from the vast majority of machine learning enthusiasts is to establish a strong background in these fields of mathematics.

Gaussian Processes Regression happens to be a great aid to this learning process – combining elements of probability, calculus, as well as linear algebra in the derivation of its closed form solution. That's what we'll focus on in this problem statement.

Remember, **a Gaussian process is a system of infinite Gaussian distributions, all finite subsets of which will always be in Gaussian distribution with some mean and covariance matrix.**

Task

For this question, we want you to consider a set of points X, a function F, and corresponding evaluations $F(X)$. Consider them as data points collected without any noise.

1. Find a surrogate function $f(x)$ using Gaussian Process regression that passes through all $(x, F(x))$ for all $x \in X$, such that it interpolates smoothly between them, and extrapolates "well" into the test data, X'. In other words, **given some data as the prior, derive the posterior predictive distribution of the corresponding Gaussian Process**.

   Clearly mention how you construct the kernel function, mention why it needs to be positive-semidefinite, understand what a Cholesky decomposition does to a matrix, and so on. This will go a long way in understanding how and why things work in the world of Machine Learning.

   Assume that the function is non-periodic, smooth, and differentiable everywhere. State your other assumptions clearly. Note that this is not an optimization problem, and that there does exist a closed form solution to this problem. Click here for a blog I wrote on the mathematics of Gaussian Processes a while back.

   Bonus: How would you define how "well" it extrapolates? Do you need to "optimize" it? How is it different from a regular neural network in its optimization step? Can you adjust this system to perform classification instead?

2. Now, implement Gaussian Process Regression, with the same math you derived, in `python`, using only basic operations and `numpy`, and check its performance on a toy dataset.

4 Implementation

Here's the last challenge – we have a bucket of models that we think are interesting, and rewarding to implement. You are allowed to use either `pytorch`, `jax`, or `tensorflow`, and all allied libraries (**not** `keras`**!**), but **no other libraries**.If you have a doubt as to what libraries you can use, reach out to us at the phone numbers or email ID listed at the top of the paper.

Attempt any **one** of these implementations. We have noted the rough difficulty of the implementation based on our own experience implementing them in the past.

1. **Vanilla Autoencoder** (*Easy*)
   Recommended libraries: `pytorch`
   An autoencoder is a neural network that is trained to reproduce its input at the output layer. This simple structure involves creating an encoder part which reduces input data into a lower-dimensional "bottleneck" layer, and then building a decoder part which attempts to reconstruct the original input from this bottleneck representation. The training process optimizes the weights in the network to minimize the difference (or "reconstruction error") between the input and output.

2. **U-Net** (*Easy-Medium*)
   Recommended libraries: `pytorch, tensorflow`
   U-Net is a convolutional neural network used primarily for semantic segmentation tasks. It follows an encoder-decoder structure, but with a twist: it has additional "skip connections" from the encoder to the decoder part that help to retain spatial information lost during pooling operations. Implementing a U-Net requires building an encoder part that uses convolutions and pooling to extract features and reduce spatial dimensions, and a decoder part that uses transposed convolutions (also known as deconvolutions) and skip connections to recover the spatial dimensions and make pixel-level predictions.

3. **Basic Transformer** (*Medium-Hard*)
   Recommended libraries: `pytorch`
   Transformers are a type of model that rely on self-attention mechanisms and have been very successful in NLP tasks. Implementing a transformer involves creating self-attention layers, which allow each part of the input to consider other parts when encoding its own information. Then, you must arrange these layers into encoder and decoder stacks, each containing several layers of self-attention and feed-forward neural networks. The encoder reads and processes the input data, and the decoder generates the output.

4. **Graph Convolutional Network** (*Easy-Medium*)
   Recommended libraries: `jax`
   Graph Convolutional Networks (GCNs) are used for tasks that involve graph-structured data. Implementing a GCN involves creating convolutional layers that can operate on graphs, meaning that they update the features of each node based on the features of its neighboring nodes. This requires defining how to aggregate and combine features from neighboring nodes, typically involving operations like summing, averaging, or taking the maximum.

5. **Generative Adversarial Network** (*Hard*)
   Recommended libraries: `pytorch`
   GANs consist of two parts, a generator and a discriminator, which are trained together. The generator creates fake data (trying to mimic the real data distribution), and the discriminator tries to tell the difference between real and fake data. The training process involves alternately optimizing the generator and discriminator: the discriminator is trained to maximize its ability to classify real vs. fake, and the generator is trained to maximize the discriminator's error on the fake data.