CS 391L **Machine Learning WB** **Spring 2023**

Project Suggestions

1 Sparsity in neural networks

Description: Deep neural networks are computationally expensive and require a large amount of memory and computational resources. One approach to address this challenge is to introduce sparsity in neural networks. Sparsity refers to the phenomenon where a small number of neurons in a network are activated for a given input. Sparsity can reduce the computational cost and memory requirements of neural networks without sacrificing performance. In this project you can explore sparsity in neural networks and study its impact on the network's performance. Start with an overview of different techniques for inducing sparsity in neural networks, such as pruning, sparsity regularization, and quantization. Implement some of these techniques and compare their performance with traditional dense networks. You can experiment on popular datasets such as MNIST, CIFAR-10, and ImageNet. Compare the performance of these sparse networks with traditional dense networks in terms of accuracy, memory requirements, and inference time. You can also analyze the sparsity patterns of the networks and visualize the learned features to gain insights into how sparsity affects the network's ability to learn. Examine the interpretability of sparse networks and how they can provide insights into the underlying data.

References:

1. `https://arxiv.org/pdf/2102.00554.pdf`
2. `https://icml.cc/virtual/2021/tutorial/10845`
3. `https://pytorch.org/tutorials/intermediate/pruning_tutorial.html`

2 Exploring best practices for long-tailed classification

Description: Long-tail datasets are common in real-world scenarios where the majority of the data belongs to a few classes, while the remaining classes have relatively few samples. The presence of imbalanced label data distribution poses significant challenges to machine learning algorithms as they tend to overfit on the majority class, leading to poor performance on the minority classes. In this project, you can explore various techniques for addressing the class imbalance problem, such as oversampling, undersampling, generating synthetic data, changing loss formulation and ensemble methods for improving classification performance. You can experiment on long-tailed classification datasets such as EURLex-4K, ImageNet-LT, and LF-AmazonTitles-131K.

References:

1. `https://arxiv.org/pdf/2110.04596.pdf`
2. `https://www.youtube.com/watch?v=I8g0LlEER9g&ab_channel=YunchaoWei`
3. `https://github.com/Vanint/Awesome-LongTailed-Learning`

3 Applications of sequence-to-sequence models

Description: Transformer-based sequence-to-sequence models are the underlying engines behind many interesting applications emerging from deep learning (for e.g. ChatGPT). In this project, you can try to come up with innovative ways of applying these powerful models. One example could be to generate a chess engine using a sequence-to-sequence model since we can consider chess move generation as next token generation for a given input sequence. Another example, can be song/music generation using such sequence to sequence models.

References:

1. `http://jalammar.github.io/illustrated-transformer/`
2. `https://huggingface.co/blog/encoder-decoder`
3. `https://www.youtube.com/watch?v=kCc8FmEb1nY&ab_channel=AndrejKarpathy`
4. `https://medium.com/@noufalsamsudin/generating-music-with-seq2seq-models-627b2506265a`

4 Learning with multi-modal input data

Description: Multi-modal learning refers to the process of learning from multiple sources of data, including visual, auditory, and textual inputs. The potential of multi-modal learning lies in its ability to extract more comprehensive and robust representations of the input data, leading to improved performance and accuracy. In this project, you can explore the best practices for learning with multi-modal input data and investigate how to leverage the information from different modalities to improve the learning process and ultimately enhance the performance of the model. The project will involve the development of a multi-modal learning model, training on datasets with diverse modalities and testing its performance. One potential benchmark that you can consider is the Caltech-UCSD Birds-200-2011 Dataset - this dataset contains 11,788 images of birds, along with text descriptions of the birds. It can be used to build a model that classifies birds based on both image and text inputs.

References:

1. `https://www.youtube.com/watch?v=helW1htty08&ab_channel=ArtificialIntelligence`
2. `https://arxiv.org/pdf/1705.09406.pdf`
3. `https://github.com/pliang279/awesome-multimodal-ml`

5 Exploring double descent behavior in neural network scaling

Description: One of the key challenges in deep learning is to understand the behavior of neural networks as they scale in size and complexity. A recent phenomenon in this area is the double descent behavior, where the test error of a neural network first increases and then decreases as the number of parameters in the network increases. In this project, you can explore the double descent behavior in neural network scaling. Specifically, you can study the theoretical foundations of the double descent phenomenon and its relation to overfitting and generalization, and implement and train neural networks of varying sizes on standard image classification datasets such as CIFAR-10 and ImageNet. Analyze the behavior of the test error as a function of the number of parameters in the network and investigate the existence of the double descent phenomenon. Evaluate the impact of regularization techniques such as dropout and weight decay on the double descent behavior. Compare and contrast the results with existing literature in the area and discuss the implications of the findings.

References:

1. `https://openai.com/research/deep-double-descent`
2. `https://mltheory.org/deep.pdf`
3. `https://www.lesswrong.com/posts/FRv7ryoqtvSuqBxuT/understanding-deep-double-descent`

6 Exploring tree structure for hierarchical search in large output spaces

Description: In many real-world scenarios, the output space (number of classes in the dataset) can become very large. A common approach in such cases is to organize the labels in a tree-based hierarchy and apply a hierarchical search procedure (such as beam search) to efficiently search the label space. The quality of the hierarchical search procedure relies heavily on the quality of the tree structure. In this project, you can explore different ways of constructing the label tree and compare their performances and tradeoffs. Most of the existing methods define their label tree structure by performing hierarchical k-means on the labels. One potential direction that you can try is exploring graph-based partitioning methods (for e.g. METIS graph partitioning algorithm) of label space instead of k-means based partitioning. You can experiment on extreme classification datasets such as EURLex-4K and Amazon-670K.

References:

1. `https://www.amazon.science/publications/pecos-prediction-for-enormous-and-correlated-output-spa`
2. `https://github.com/KarypisLab/METIS`
3. `https://arxiv.org/pdf/2210.08410.pdf`

7 Gradient descent at the edge of stability

Description: Recent observations on training dynamics of Neural-Networks revealed an interesting phenomenon – Edge of Stability. Neural Network training can be formulated as an optimization problem as follows:

$$\min_w f(w) = \min_w \sum_{i=1}^{N} f_i(w)$$

where f_i is loss function corresponding to each datapoint and $w \in \mathbb{R}^d$ denotes the set of parameters flattened into a vector. Full-batch gradient descent minimizes the overall loss by using the following update rule for parameters w:

$$w_{t+1} := w_t - \frac{\eta}{N} \sum_{i=1}^{N} \nabla f_i(w_t)$$

Gradient descent for functions with *maximum sharpness* - $L = \max_w \|\nabla^2 f(w)\|_2$ typically converge when $2/\eta \geq L$ and loss diverges when $2/\eta < L$. But, conducting this update on a neural network showed that even if $2/\eta < L$, the loss still shows an overall downwards trend and the *sharpness* of the iterates- $\|\nabla^2 f(w_t)\|_2$ oscillates near $2/\eta$, at the edge of stability. In this project you can reproduce these findings with the full-batch gradient descent algorithm in simple neural networks.

References:

1. `https://arxiv.org/pdf/2205.09745.pdf`
2. `https://arxiv.org/pdf/2103.00065.pdf`
3. `https://arxiv.org/pdf/2207.14484.pdf`

8 Comparison of first-order vs second-order optimization methods

Description: Optimization is an essential component of machine learning algorithms. In order to optimize a cost function, various optimization techniques can be used, including first-order and second-order optimization methods. First-order optimization methods, such as stochastic gradient descent (SGD), are widely used in machine learning algorithms due to their simplicity and low computational cost. These methods compute the gradient of the cost function with respect to the model parameters and update the parameters based on this gradient. Second-order optimization methods, such as Newton's method, are more computationally expensive but can converge faster and more accurately than first-order methods. These methods compute the second derivative of the cost function and use this information to update the model parameters. More specifically, second-order optimization involves using a preconditioner matrix H_t to transform the gradient before making the update to the parameters as follows:

$$w_{t+1} \leftarrow w_t - \eta H_t^{-1} g_t,$$

where $g_t = \nabla f(w_t)$ denotes the gradient of the objective function f to be minimized. A carefully chosen matrix H_t can speed up the convergence compared to first-order methods such as SGD or Adam.

In this project, you can compare the performance of first-order (SGD, Adam) and second-order (Shampoo, Adafactor) optimization methods on various small-scale machine learning algorithms, including logistic regression and neural networks. Evaluate the optimization methods based on their convergence rate, computational cost, and accuracy. Additionally, explore the effect of hyperparameters such as learning rate, batch size, and regularization on the performance of both optimization methods.

References:

1. `https://robinschmidt.netlify.app/post/second-order/`
2. `https://arxiv.org/pdf/1802.09568.pdf`
3. `https://arxiv.org/pdf/1804.04235.pdf`
4. `https://github.com/google-research/google-research/tree/master/scalable_shampoo/pytorch`