Society for Artificial Intelligence and Deep Learning

www.saidl.in

Spring 2022 Induction Assignment

Introduction

This is the official assignment for SAiDL inductions. Please join our Slack Workspace here and fill this form in order to register.

Deadline is **Thursday 17/02/2021 11:59 PM IST**.

A Note from Us

Through this assignment we aim to introduce you to key ideas in Machine Learning, Deep Learning and Artificial Intelligence. We value your time and have designed each question such that the process of solving it should add to your skill set while also giving us an idea of your abilities. We have also provided a collection of resources to help you learn the required tools and concepts along the way.

Please go through the instructions before attempting the questions. The questions might seem intimidating at first glance. Don't panic! They are meant to be challenging and will take some time and effort to complete. Try looking around to understand the concepts involved in the questions and try digging into these topics. It is expected that around 10% - 40% of time (depending on your prior familiarity) on each question will be spent on learning new concepts. The submissions will be judged primarily on the basis of the approach and the thought that went into attempting the answer, so be sure to document everything. Also make sure to submit your attempt irrespective of whether or not you have completed the assignment as a whole.

If you have any doubts, try to search for solutions on the internet first before asking doubts on the slack. We will also be assigning everyone mentors so keep them updated with your progress.

All the best!

Logistics

Submission Guidelines

The submission deadline is **Thursday 17/02/2021 11:59 PM IST**
Submission is to be made via: https://forms.gle/bs6MrbcoLV2egG446

The submission should be primarily in the form of a Git repo titled *SAiDL-Spring-Assignment-2022*. The repo should be **kept private up until the submission deadline**. Make sure you make the repo public while submitting the assignment.

You are required to document your approach and results for each question you attempt and analysis wherever required, in detail. The documentation should be included in the github repository along with the code and can be made in Markdown, LaTeX etc… Make sure to make your final report for each task as detailed and clear as possible.

Any form of plagiarisation or collaboration with others is strictly prohibited and will be penalised.

You'll be allotted a mentor for the assignment and you are required to inform them whenever you complete a question (submit the google form only after you have completed the assignment or after the deadline). All of this will be done through our Slack Workspace so make sure you join it for receiving all the assignments and for asking queries that you might have. You can also contact those mentioned below for queries.

SAiDL holds the rights to the questions asked in the assignment, if you choose to use this outside the assignment, you need to cite/acknowledge SAiDL.

Contact Details

In case of any doubts, clarifications, or guidance, you can contact one of us. We request that you stick to slack as the medium of communication for all the questions that you have. However, you can reach out to us through other means in case we fail to respond on slack.

- Vedant Shah (RL) - 7359313678
- Atharv Sonwane (Core ML) - 8237441175
- Omatharv (Core ML) - 9664944835
- Shrey Pandit (NLP) - 9820320640
- Vishwa Shah (NLP) - 8369270547
- Shreyas Bhat (CV) - 9082080984
- Hrithik Nambar (CV) - 8301874897
- S I Harini (CV) - 7021247073
- Hardik (CV) - 9427925103

Resources

The following is a list of resources to tools and topics you will need to be familiar with in order to solve the assignment questions. This section is more a selection of resources for things you will need along the way. It is not mandatory to complete, you should treat it more as a roadmap of things to learn and as a reference to get back to.

Python

Python is the Lingua Franca of AI (at the moment). Check out this guide by Google to start or the tutorial series by Sentdex. These books (1, 2) and the official documentation are good references.

Coursera Course on Machine Learning

This course is the launchpad for most AI enthusiasts and gives you your first hands-on approach to Machine Learning along with the maths behind it. The course requires implementation of ML algorithms in Octave/MATLAB. However, it is recommended that you attempt the Python versions of these exercises which can be found here.

Linux Terminal

This is one of the fundamental requirements for any computer scientist. The following will help you get started (1, 2).

Numpy

Crudely speaking this is MATLAB for Python. We suggest you do this after the Andrew NG course. You can go to its official tutorial or learn it with hands-on deep learning experience via deeplearning.ai's first course.

Pandas

This is one of the most crucial and powerful libraries in data science. To begin with, you end up learning how to read and write CSV and JSON files as well as how to manipulate Data Frame rows, columns, and contents. Again Sentdex to the rescue.

Matplotlib

Learn how to plot basic graphs. The pyplot submodule should be enough for the beginning. Sentdex is your saviour again. The docs are useful too!

PyTorch

PyTorch is a Machine Learning framework built to provide a toolkit for writing Deep Learning research and application code. We would suggest that you get hands-on experience with PyTorch by following the official tutorials.

Git

An integral part of most software projects, Git is a tool for version control and managing changes to code. This is a cool guide for a quick overview, for a more detailed intro check out this course. Make sure to sign up for the Student Developer Pack on GitHub.

Conda

Managing different packages when you are working on multiple projects can be a pain (especially in Python). Conda (and various other similar tools) is a highly featured package manager which makes life much easier. Check it out at the docs and learn how to use it here or here.

Resources for ML and AI

For Core Machine Learning, we recommend Bishop's *Pattern Recognition and Machine Learning* Textbook [Link] and for Core AI, Russel and Norvig's *Artificial Intelligence: A Modern Approach* [Link]. Both of these are very comprehensive textbooks that cover the fundamentals of both areas with easy to understand explanations. You do not need to complete them cover to cover in one go, but can use them more as reference books, learning about specific topics at a time.

For those who are new to Deep Learning, we recommend you start with the Stanford CS231 (Computer Vision): [YouTube Link] [Course Link] course which covers everything from loss functions and basics of Neural Networks to Computer Vision. Go through the video lectures and attempt the accompanying assignments to get a good idea of various fundamental concepts. The Deep Learning Book [Link] is also a great resource for detailed introductions to a wide range of concepts.

For those interested in getting into specific areas -
Natural Language Processing:

- Stanford CS224 [YouTube Link] [Course Link]

Reinforcement Learning:

- Berkeley CS285: [YouTube Link] [Course Link]
- NPTEL [YouTube Link]
- Sutton and Barto's Classic Introductory Text [Book Link]

Assignment

The assignment is made up of the following sections

1. Literature Review (Compulsory)
2. Core ML: Bayesian NNs via MCMC
3. Domain Specific Questions
    a. Natural Language Processing
    b. Reinforcement Learning
    c. Computer Vision
4. Paper Implementation

Section 1 is compulsory. Additionally, you should attempt **EITHER** of **BOTH** Section 2 and one task out of the three domain specific tasks in Section 3 **OR** attempt the Paper Implementation Task which makes up Section 4. Please note that the difficulty of the questions varies, and you will be judged primarily on the effort you have put in.

Along with how well you perform the tasks, you will also be evaluated on how clearly and thoroughly you document your approach and results. So make sure to make your final README or report for each task as detailed and clear as possible.

1. Literature Review (*Compulsory*)

Prepare an article between 1000 and 1500 words critiquing one of the following papers

- A ConvNet for the 2020s
- Lifelong Planning A*
- Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks
- Induction of Decision Trees
- Efficient Visual Pretraining with Contrastive Detection
- If you are attempting the Paper Implementation (Section 4) then you can use the same paper for this question.

Your critique should answer the following questions as clearly and concisely as possible

1. What is the main problem the paper is trying to tackle?
2. How have others tried it in the past?
3. What is different about this paper's approach?
4. What experiments did the paper perform to prove their central claim?
5. Where do you feel the paper falls short?
6. How would you build upon the paper's approach?

Your essay will be evaluated on (1) your understanding of the paper and more importantly (2) how you evaluate its strengths, weaknesses and areas of improvement.

2. Core ML: Bayesian NNs via MCMC

Consider your regular neural networks, they are made up of a bunch of parameters and given a dataset, you can iteratively modify those parameters until you reach a specific set that gives you good performance. You then use these trained parameters on your test data to generate predictions. The Bayesian approach to this process is to work with *distributions* for each parameter rather than considering it a single number. We start with an initial distribution over all parameters called the *prior*. Using Bayes theorem and the training dataset, we update this distribution according to how well it explains the dataset to obtain the *posterior* distribution.

The key point here being that we have distributions over our model parameters which are functions that give us the probability of the model parameter being a certain number. The question then becomes - how do we obtain parameters from this distribution? How do we sample from a distribution when we only have the probability scoring function? This is what MCMC (Markov-chain monte-carlo) does. In very broad terms, we sample parameters from a proposal distribution that is conditioned on the previous parameters sampled and accept this sample if its relative probability is higher. In doing so for a large number of parameters, we can approximate the underlying distribution.

This might sound like a lot of fancy words, especially if you have not come across Bayes Theorem before. Fear not! There are many explainers and blogs out there going over this. We especially recommend going through [1] which goes over the Bayesian way to do regression and how it is different from your usual approach and [2, 3] for a visual demonstration of MCMC. This should give you a high level idea of what we are going on about. Next, to get a more indepth look at how Bayesian inference and MCMC in particular work in practice, you should check out either one of [4, 5] for more thorough explanations and examples. Finally, if you have specific doubts, feel free to contact the relevant members of the SAiDL team.

Most importantly, don't let your unfamiliarity with Bayes deter you from attempting this question! Bayesian inference is a critical tool in Machine Learning and you should take this opportunity to learn about and gain experience with it.

Your task is to construct a Bayesian Neural Network and obtain its parameters using MCMC for the noisy XOR problem. A regular 2-input XOR function outputs 1 if exactly one of its inputs is 1. In the noisy XOR case, the inputs can be any real numbers between 0 and 1 and we output 1 if exactly one of the inputs is greater than 0.5. We want to model this function using a Bayesian MLP and obtain samples for its parameters with MCMC. These samples can then be used to evaluate performance on the test set. Since we have multiple samples of the parameters, we can get multiple predictions (each from a different MLP) which then make up an approximate distribution over the output. You can check out [4] for an example of Bayesian MLPs and [6] for an explanation of the noisy-XOR problem in this context.

Task in steps -

1. Prepare a dataset for the noisy XOR problem and split it into train and test.
2. Implement a prior, likelihood and posterior function for MLP parameters
3. Implement MCMC procedure (of your choice)
4. Sample MLP weights that have posterior probability for given training dataset using the above procedure. Hint: warm starting
5. Evaluate the sampled sets of weights on the test set. Hint: ensemble prediction.

The output for this task should be your implementation of the MCMC procedure and the Bayesian MLP along with a short README file describing the details of your methodology (hyperparameters) and results. You can use any MCMC sampler (Hamiltonian Monte Carlo, Metropolis-Hastings, etc) according to your choice. Along with this task, also construct a MLP model with a random sampler and compare your results with the MLP model from the MCMC sampler. Clearly specify any assumptions you have made for representing the model and making the code.

Bonus: Complete this question in Julia. It combines the ease of use of dynamic languages like Python with the performance of languages like C++ because of which it is increasingly being used for Machine Learning and AI..

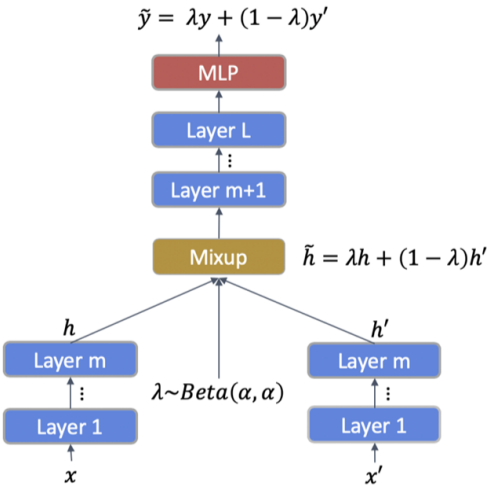
Resources -

1. https://towardsdatascience.com/introduction-to-bayesian-linear-regression-e66e60791ea7
2. https://www.youtube.com/watch?v=XV4yj4T4PBQ
3. https://chi-feng.github.io/mcmc-demo/
4. https://towardsdatascience.com/from-scratch-bayesian-inference-markov-chain-monte-carlo-and-metropolis-hastings-in-python-ef21a29e25a?gi=cefb8321cdb9
5. https://arxiv.org/pdf/2007.06823.pdf
6. https://arxiv.org/pdf/1910.06539.pdf

3. Domain Specific Questions

Natural Language Processing

The recent increase in the popularity of data augmentation in NLP has led to much research on this topic. It is known that data augmentation techniques help in preventing overfitting. Mixup is an interpolative data augmentation technique that has shown promising results in both computer vision and NLP. Paper - Link. Mixup can occur across any different hidden layer, as shown below -



$$\begin{aligned} \hat{x} &= \lambda x_i + (1-\lambda)x_j, \\ \hat{y} &= \lambda y_i + (1-\lambda)y_j, \end{aligned}$$

where $\lambda \in [0, 1]$ is a random number

Two samples would pass through the network until a layer (say l) then at layer l, both of these hidden representations would be mixed in a ratio λ (sampled randomly from beta distribution), and the mixed embedding representation would propagate through the rest layers identically to any normal sentence. Instead of mixing samples, storing them, and adding them along with training data, we suggest using just the mixed representations on the go while training, as this would include the prior case also.

Task: Implement a basic mixup strategy on the TREC dataset Link. Compare the performance gain achieved over original classification and a few discrete augmentations. Thoroughly document your approach (model architecture, hyperparameters etc…) as well as your results in a detailed README or report.

1. Train a baseline model for the TREC dataset (any variant, coarse or fine) and note the performance achieved.
2. Train another model with the mixup strategy involved during training and report the performance increase. Note - The Lambda (mixing ratio) for mixup should be chosen randomly and not a fixed value.
3. On a separate baseline model, augment sentences using discrete augmentation techniques like synonym replacement, word dropping, sentence flipping, etc. (pick any number of augmentation depending upon time availability)
4. Compare the performance of all 3 tasks and justify the results obtained.

Bonus: Pick any number of bonus parts depending upon time availability -

1) **Practical** - If we just mix samples from the same label amongst itself, eg. for Binary classification, mixing amongst sentences only where labels are 1 for both sentences or 0 for both sentences, modify the above code and empirically justify the reason for the observed increase/decrease in performance.
2) **Theoretical** - instead of randomly picking sentences for mixing can there be some logical reasoning behind sampling sentences and would it increase/decrease performance, if yes, propose some idea. If not then justify. (Hint - read the papers on Mixup and the reasoning why we want to do mixup)
3) **Practical** - Can the mixing ratio be initialised to some value initially (eg. 0.5) then make it trainable throughout the training regime, does this approach increase the performance, Yes/No. Justify results empirically.

Reinforcement Learning

Advances in Deep Reinforcement Learning have led to the development of various algorithms that enable an artificial agent to learn from complex image observations as compared to vector representations, for eg. like the ones available in classical gym environments like Cartpole of MountainCar. One of the earliest examples of this can be seen in the landmark paper "Playing Atari with Deep Reinforcement Learning" by DeepMind. The paper demonstrates that Deep Q-Learning can be used to solve a range of environments based on the Atari 2600 games using raw image observations which are encoded using a Convolutional Neural Network.

Deep Reinforcement Learning approaches are notorious for their low sample efficiency, i.e. they require large amounts of agent - environment interaction data leading to long training times which is one of the factors which makes them infeasible to use in practical applications. One reason for the low sample efficiency is that capturing essential information from high level sensory data (for eg. image observations) and encoding that information into representations is a challenging task and often essential for DRL to be sample efficient. Traditionally, and as done in [1], the representations are trained solely using the reward signal. However, with advances in representation learning approaches, modern day DRL approaches often use auxiliary tasks to introduce an additional training signal to train better representations of the image observations. The more efficient the representation, the easier it is for the agent policy to decide what action to take in the next-time step. This helps in improving the sample efficiency. The auxiliary tasks used most often is reconstruction loss but is being rapidly replaced by self-supervised learning approaches.

In reconstruction loss based approaches, the image observation obtained from the environment is encoded using CNN based autoencoder (as compared to simply encoding the observation using a CNN). The encoding is used by the agent policy to decide the actions and the agent (encoding) is trained jointly using the loss obtained from the agent (i.e. the reward signal) and the reconstruction loss obtained from the autoencoder. In self-supervised learning based approaches, the autoencoder can be replaced with any of the SSL / contrastive loss based approaches.

Task: Compare the performance of a Deep Q Network on Atari environments with the following modifications. Thoroughly document your approach (model architecture, hyperparameters etc…) as well as your results in a detailed README or report.

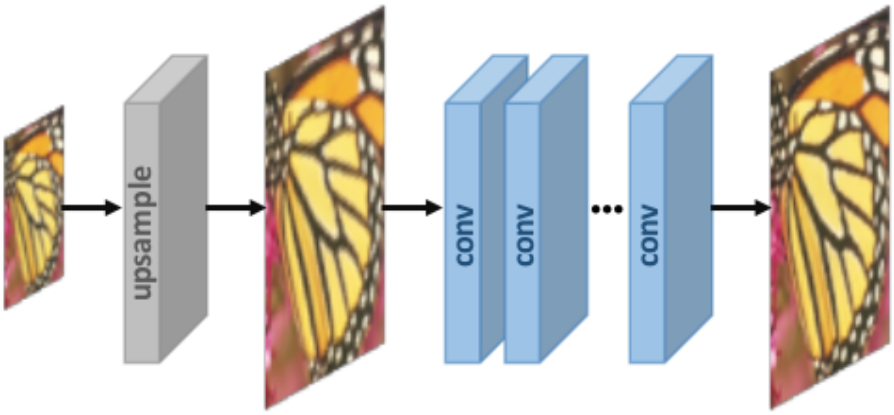
1. **Vanilla DQN**: Train and test a Deep Q-Network that can solve some of the Atari environments. Some environments you can use are: Atari Pong ('PongNoFrameskip-v4' in gym), Atari-Breakout ('BreakoutNoFrameskip-v4' in gym). You can either train your own DQN or use one of the existing implementations. (For eg. https://github.com/KaleabTessera/DQN-Atari - takes ~4 hrs to train on a colab GPU) (Using already existing implementations will not affect the evaluation)
2. **DQN + Reconstruction Loss**: Replace the CNN based encoder in the above by a CNN based autoencoder and use the reconstruction loss in addition to the agent loss to train the agent.
3. **DQN + SSL**: Replace the autoencoder with one of the following self supervised learning methods (Listed in order of increasing complexity):
    a. SimCLR [1]
    b. MOCO [2]
    c. Barlow Twins [3]
    d. BYOL[4]

Note that this is intended to be an experimental and analytical question. So you can use existing implementations for DQN (verified implementation), SimCLR, MOCO, Barlow Twins and BYOL if any. Be careful however and make sure that you understand the implementation and that it is correct (some metrics to judge this can be the number of stars on the repo, author's implementations, etc.)

Bonus: Try out as many methods mentioned in 3. above as possible. Benchmark the performances of 1., 2., and 3. on multiple Atari environments.

Computer Vision

Image super-resolution (SR) is the process of retrieving high-resolution (HR) images from the corresponding low-resolution (LR) images. It is an important task in image processing and deep learning has played a key role in taking super-resolution to the next level in the past decade. You can learn more about super resolution through [1] [2][3] [paper].



In this task, you will be implementing and evaluating variations of approaches to super-resolution on the Oxford-IIIT Pet Dataset [Official Website]. For this you will need to construct low resolution images from the original dataset to train and test your implementation. One way to do this is resizing the original data to a lower dimension.

Task: Implement code to perform the following experiments and provide thorough documentation of your approach (model architecture, hyperparameters etc…) as well as the results in the form of a detailed README or report.

1) Implement and evaluate the following approaches on the Oxford-IIIT Pet Dataset
    a) Super-resolution CNNs (SRCNN)
    b) Super-resolution ResNet (SRResNet)

2) Compare the performance of different types of interpolation (Bicubic, Bilinear, Nearest Neighbour). Interpolations are different techniques to upscale the lower resolution images which was classically used as an image super-resolution and image scaling method before the deep learning era.

3) Compare different types of upsampling methods in the above models. For example - Convolutions, transposed convolutions, Sub-pixel convolutions to name a few.

4) For the above evaluations, use different metrics like Peak Signal to Noise Ratio (PSNR) which is commonly used as a metric to account for loss of quality in different image enhancement and compression techniques, MSE/L2 loss etc. (use google to find more[4]). What are the evaluation metrics and loss functions used in the recent literature? How do they help in generating the textures and help in performance of the models?

Bonus: Traverse a (3x3) filter over the image in a way that the filter blackens out the pixels. Report how the new output compares to the original output from Question 1.

a) Infer from this experiment and explain the results. (If blackening out one region brings about a noticeable change, try and explain what is unique about that region)
b) Repeat this experiment with different dimensions of the filter - 5x5, 7x7… and report at what point does the output become noisy and redundant
c) Repeat this experiment with filters of different colours - (Red, Blue, Green) and with different types of filters - Gaussian Filter, Bilateral Filter, different low-pass and high-pass filters and report your results and observations.

From these results, explain the success of GANs in super-resolution and how it preserves textures in the final higher resolution image. Can you think of the cases when super-resolution would fail or break down?

4. Paper Implementation

Your task is to examine the central claim of a paper by implementing it and performing additional empirical experiments.

1. Select any one of the papers published in an A or A* conference (NeurIPS, ICML, ICLR, ICCV, AAAI, ACL, etc). First, read the abstract and give a rough reading of the paper you have chosen and identify its central claims. At this point inform us that you are attempting to implement the paper and tell us about the paper chosen.
2. Once you get approval from us, you can go ahead and give a thorough reading of the paper and implement the paper using a framework of your choice (we highly recommend PyTorch).
3. Formulate and perform additional experiments and ablations (such as trying on a different dataset, with different experimental parameters etc...) to test the central claims of the paper. You can discuss these with your mentor.
4. Document your code with a README and a short review for the paper (preferably in LaTeX). The documentation should contain the results of your evaluation and explain how your additional experiments are evaluating the claims of the paper.

You will be evaluated primarily on your understanding of the paper and the additional evaluations you have performed on its central claims. We are not expecting very clean code, however it should be able to replicate the results described in the paper and be well documented. While this task has not been designed for beginners, anyone is welcome to give it a go.