Society for Artificial Intelligence and Deep Learning

www.saidl.in

Summer 2021 Induction Assignment

Introduction

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

Deadline is **8th August 2021.**

A Note from Us

Through this assignment we aim to introduce you to some of the key areas of research in Machine Learning / Deep Learning and Artificial Intelligence in general. The assignment is split into separate sections, and has been designed keeping in mind the key areas in which these fields are focused and the latest developments. 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. All the files necessary for solving the assignment are present in this github repository**.**

Please keep in mind that we value your attempt more than the final results. The questions might seem intimidating at first glance. Don't panic! The questions 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. The submissions will be judged primarily on the basis of the approach and the thought that went into attempting the answer - hence, be sure to document everything. Make sure you submit the assignment irrespective of whether or not you have completed the assignment as a whole.

If you have any doubts, try to Google first - you'll find the answers to most of your questions on the internet itself. If still not clear, you can contact any of the SAiDL members.

All the best!

Logistics

Submission Guidelines

The submission deadline is **8th August 2021**
Submission is to be made via https://forms.gle/2xMM8b59jfHAUdpP9

The submission should be primarily in the form of a github repo titled *SAiDL-Summer-Assignment-2021*. The github 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 **Word / Docs / LaTeX etc.**

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

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.

Note: SAiDL holds the rights to the questions asked in the assignment, if you choose to use this outside the assignment, you will have 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 (RL) - 8237441175
- Somesh Singh (NLP) - 9140827410
- Yash Bhartia (NLP) - 9987242011

- Neelay Shah (CV) - 9270018699
- Shrey Pandit (CV)- 9820320640
- Hrithik Nambiar (CV) - 8301874897

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 3.8

Python is the Lingua Franca of AI. You will be learning Python 3.8. 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 savior 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.

Stanford's Courses for NLP, CV, RL

The following courses provide a great introduction to these key areas of ML. It is recommended to go through the video lectures and attempt the accompanying assignments to get a good idea of various fundamental concepts.

Natural Language Processing CS224: [Youtube Link] [Course Link]
Computer Vision CS231: [Youtube Link] [Course Link]
Reinforcement Learning CS234: [Youtube Link] [Course Link]

If you find yourself interested in the fascinating world of Reinforcement Learning, you can also go through UC Berkeley's CS285 course on Deep Reinforcement Learning - arguably the best resource to start exploring Deep Reinforcement Learning

Deep Reinforcement Learning CS285: [Youtube Link][Course Link]

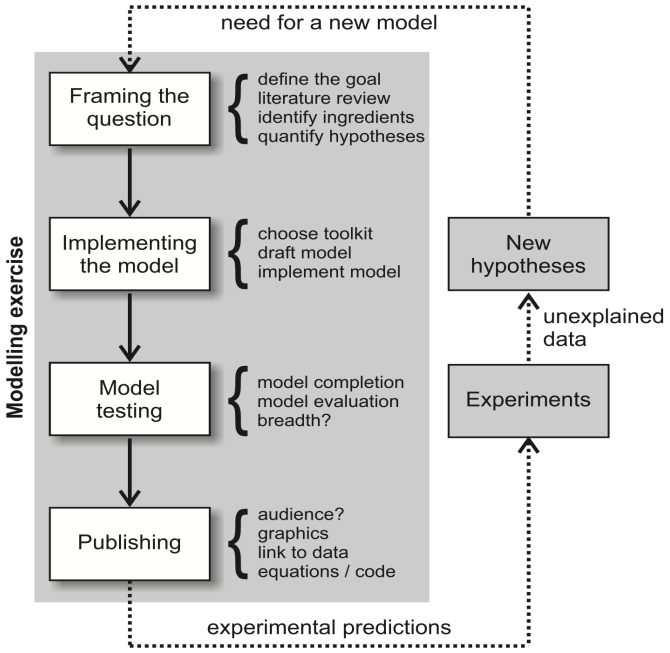
Assignment

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

1. Research Problem Formulation and Critical Analysis (*Compulsory*)

A recent paper by Gunnar Blohm, Konrad P. Kording, and Paul R. Schrater, titled "A How-to-Model Guide for Neuroscience" (Some notes for this paper), breaks down the process of doing research into 10 practical steps. Although this paper is for Neuroscience, it applies to other fields as well because the steps are not neuroscience specific.

Your task is to break down any new project idea/existing project into these steps (if needed, you can skip the last step, i.e., publishing). No code is required (if needed, you can just assume that the code was implemented for steps 7 and 8), only a plan for a project broken down into these steps; however, if you are writing about an existing project, you can provide a link to the code. The project can be anything as long as it is related to Artificial intelligence. Please assume no constraints on the dataset and computational power, focus on the questions and hypothesis, not on the datasets or feasibility. There is no word limit, but we would recommend keeping it short and to the point. We want to instill critical thinking. Hence the task is very subjective and has no right or wrong answers.



If you don't have an existing idea/project you can also take inspiration/build upon/use your knowledge from any one of the other questions in the assignment.

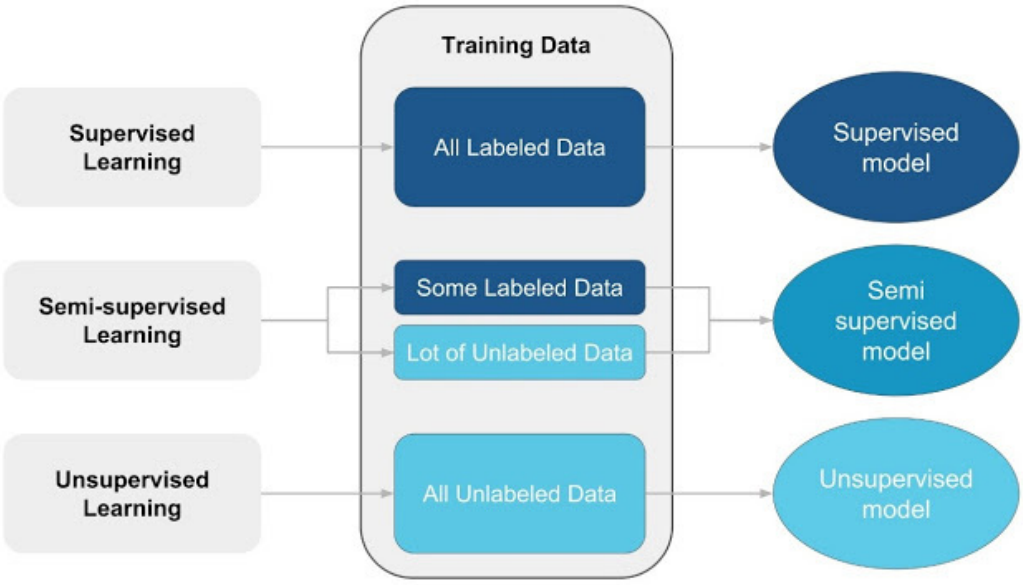
Please go through the paper for an example. Also, note that any new idea that you come up with belongs to you, we do not store/use/take any credit for it.

2. Domain Specific Tasks

A. Computer Vision

Deep supervised learning has been key to a lot of success in computer vision research over the past decade. However, its dependence on carefully labelled data, which is both expensive and time consuming to acquire, limits its potential for a lot of practical applications.

Semi-supervised learning describes a class of algorithms that seek to learn from both unlabeled and labeled samples, typically assumed to be sampled from the same or similar distributions. This article briefly introduces the semi-supervised learning setting. Unsupervised learning, on the other hand, aims to learn good representations of data using only unlabeled samples. Self-supervised learning is an unsupervised learning framework that relies on using auxiliary tasks that can be formulated without any labels. Speaking of computer vision specifically, self-supervised learning seeks to learn useful semantic representations of images. You can read more about self-supervised learning at 1, 2, and 3.



Your task is to compare supervised, semi-supervised and self-supervised learning approaches for achieving good performance for an image classification task when working with a partially labelled dataset (very few labelled instances in the dataset). This consists of the following steps (**required)** -

1. Use the STL-10 dataset for all subsequent tasks. When reporting evaluation metrics such as accuracy, use the entire test set they provide. Other than that, you are free to decide how you want to go about training.

2. Train an image classification model of your choice (eg. CNNs such as ResNets) on the labelled part of the dataset in a ***supervised*** way and report results on the test set. Use the same model architecture for the rest of the steps outlined.

3. Use pseudo-labelling for ***semi-supervised*** training. You are free to decide how exactly to implement this (and are encouraged to explore literature!). Bonus points for experimenting with other forms of semi-supervised training.

4. Use SimCLR (a ***self-supervised*** learning framework which uses contrastive learning, see 1 and 2) to learn good image representations from the unlabelled data. Train a linear classifier on top of the learned representations using the labelled data (here's an article which details a typical self-supervised learning workflow)

5. Document your final results (evaluation metrics such as test set accuracy) and specifications of your approaches (exact training strategy, model architectures, hyperparams, etc.). Compare how the three different approaches fared. In your report also write a paragraph about each of the following -
    - When you would expect semi-supervised learning approaches to fail. What can you do to avoid this?
    - Apart from achieving a high test set accuracy, what other metrics do you think are important while comparing and contrasting different learning approaches? (think sample efficiency, training time)

Apart from the steps outlined above, here are a couple of other things you might want to think about and explore **(optional)** -

- Clever data augmentation is at the heart of nearly all self-supervised learning approaches. SimCLR uses augmentations such as random cropping for contrastive learning of image representations. There has also been work on learning data augmentation strategies as well (see 1 and 2). How would such auto-augmentors fare with SimCLR? Can augmentation strategies learnt for a particular dataset (and possibly task) be used for other datasets? If yes, under what conditions? (1 and 2 are a couple of open-source auto augmentor implementations)

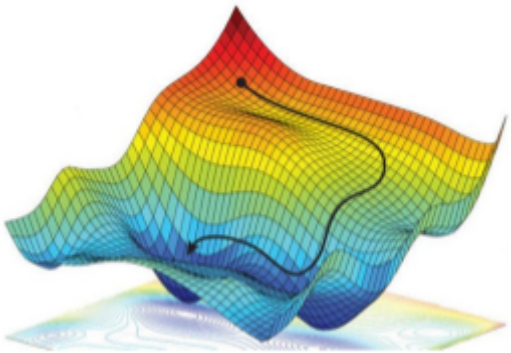
- Akin to how transfer learning is used in the supervised setting, can image representations learnt in a self-supervised manner be transferred as well?

- How could one go about *combining* semi and self supervised training to achieve good performance? (hint: S4L)

B. Reinforcement Learning

In RL we build models which can learn to behave in an environment by interacting with it in such a way as to maximise its reward. This simple framework has led to countless different RL algorithms and some pretty cool results (1, 2, 3 etc.). Introductions to the field can be found at 1, 2, 3, 4. For a more rigorous treatment see the classic textbook, or any of these lecture series (1, 2, 3)

Your task is to train an RL agent to perform convex optimisation over a given class of functions. That is, given a function $f : R^m \to R$, the agent needs to find its minima. For this task we will only consider quadratic functions with two inputs, but the idea is very general and can be used for optimisation in all sorts of settings as seen here. Think about it as climbing down a valley with limited perceptual information.



Example of continuous optimisation on a 2D function (non-convex)

This tasks consists of the following subtasks (all of which are **required** except for 4) -

1. Implement the environment

Environment in RL is used to refer to the world (often a simulator) that the agent is interacting with. The agent gives the environment the action it wants to take, the environment executes that action and returns the new state of the world along with the reward for that step. You can find examples of environments in OpenAI Gym and guides on how to build your own ones here and here.

Your task is to implement an environment in the OpenAI Gym format for convex optimisation tasks. The environment should be as general as possible but should at least be able to simulate two dimensional quadratic functions. For each episode, the

environment should sample a new set of coefficients for the quadratic function. An example of such a function might be $z = x^2 + 4y^2 + 0xy + 0x + 0y$ if (1, 4, 0, 0, 0) are the sampled coefficients.

The states returned by step() should contain the coordinates of the agent as well as the coefficients being used for the current episode. The simplest reward that can be used is the inverse of the distance of the minima from the current position. Although, you should formulate **at least two** additional reward functions and document how your solution performs on each of them and why that is so according to you.

2. Implementing and train a policy gradient algorithm

A policy is a function that takes the current state as input and outputs a distribution over the actions to take. Policy gradient is a class of RL algorithms that learn the policy function directly through iterative gradient updates based on the experience generated by the policy. Here are a few guides to policy gradient: 1, 2, 3.

Your task is to train a basic policy gradient algorithm on your convex optimisation environment. Your policy function can be a simple 2-3 layer MLP (Neural Network), which takes the state (consisting of the current coordinates and coefficients of the function for this episode) as input. Since the action being taken is continuous, the policy should output a distribution. This can be done by having the MLP output mean and variance and plugging it into a gaussian distribution. For example if the function is 2D then the MLP would output a 4 element vector containing mean and variance for distribution over steps for both dimensions. You can do this for any distribution of your choice (choose wisely!).

3. Evaluating your agent and documenting the results

Document the performance of your agent as well as training curves. Also, for this task to be even remotely useful, the RL agent must work for a wide range of coefficients of the quadratic function and not just the ones it was trained on. Prepare and execute an experiment to test to what extent the agent can generalise to functions it hasn't been trained on.

4. Bonus (optional)

In the above task, we are providing the coefficients being used in the current episode directly as part of the state, what would happen if this information wasn't given. Does

this mean our agent can't solve the problem? Think about how you would go down a valley if you were blind and had no knowledge of the surroundings. Can we apply the same principle here?

For bonus points, modify your environment implementation to only include current coordinates of the agent in the states, and train an RNN based policy (think about why?) instead of the MLP one described above. Compare the performance of the with and without coefficient setting and try to explain how you think the agent is working in the latter. For even more bonus points you can do further experiments on how the agent performs on different degree polynomials or even different classes of functions.

C. Natural Language Processing (NLP)

Earlier machine learning methods for NLP learned combinations of linguistically motivated features like **POS Tags,** and **Syntax Trees**. Though it was difficult at times to understand exactly how these combinations of features led to the decisions made by the models, practitioners at least understood the features themselves.

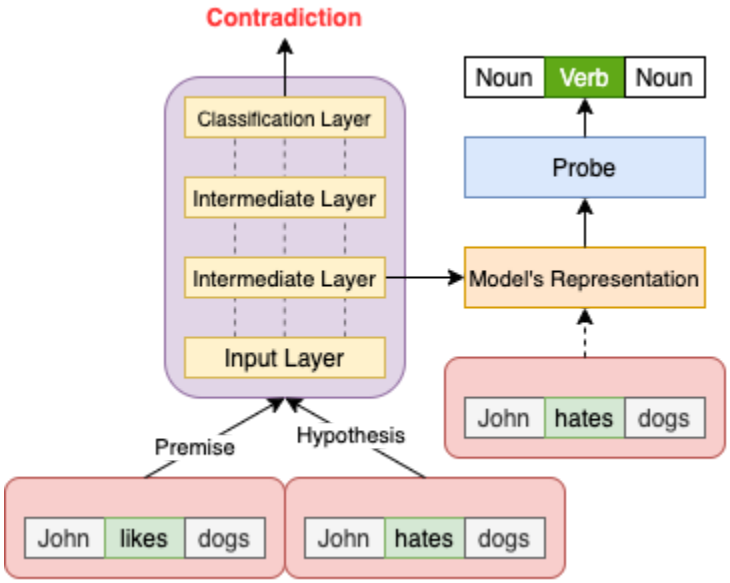
Now, NLP has come to a stage where large scale representation learning models form their own internal representation instead of features compared to the previous approaches (eg: **Word Embeddings**, **Transformers** etc). Despite the black box nature of these representations learning models, researchers intuit that the representations' properties may parallel linguistic formalisms.

There has been considerable work in interpreting these representations and what the model actually understands. One such way to interpret model learning is **probing** (You can read more about probing at 1, 2, 3, and 4). Once we have a model for the primary task (eg: Text Classification), we build a secondary classifier on top of the model's representation of original text, this is for the auxiliary task (eg: **pos tagging, TreeDepth**) and train the added probe classifier on your model representations for this task. A high evaluation accuracy (relative to baseline) in predicting the auxiliary property implies that the property was encoded to an extent in the representations and the probe found it.

For this assignment you have to use probing techniques to investigate the linguistic understanding in your model. Given a dataset containing short, ordered

sentence pairs your primary task is to determine the inference relationship namely, *entailment, contradiction, or neutral* (SNLI)[5], between the sentence pairs (Dataset).

Then use the unsupervised representation from your model to perform the auxiliary task. i.e. probe your model's layers for POS tagging (Available in SNLI, Brown Corpus) and you can use other auxiliary tasks for comparison (bonus) from Conneau et al. (2018) **[Section 2]**.



An example of an Architecture for probing!

Formally, The tasks in order are:

1. Determine the inference relationship between these sentence pairs in the SNLI dataset using some representation models (recommended techniques 1to use: Word Embeddings, RNNs / LSTMs etc).
2. Probe your model using the representations obtained from it for the auxiliary task(s)
3. If you are unable to solve Task2 completely, solve the auxiliary task(s) separately using models of your choice.

Task Specific Guidelines:

1. Task 1 is necessary, doing task 1 and 2 would be said to complete the question. However if you are not able to do part 2 you can attempt task 3.

2. A moderate baseline for Task1 will be 70%, however for Task2 there are no baselines because probing is an investigation technique and the analysis is more important than the performance.

3. Document your approach in detail and make sure you comment/explain your code. Give an in depth analysis of your results on the probe.
4. Do not use any external dataset and avoid plagiarism.
5. Note that we are not looking for extremely high performance, you will be evaluated on your understanding, efforts, discussion and analysis.

References:

1. Word embeddings
2. Long Short Term Memory Cells (LSTMs)
3. Stanford Natural language Inference Dataset
4. Part of Speech (POS) Tags
5. Probing: 1, 2, 3, and 4

---

3. Paper Implementation Task

If you are attempting this question your task is to implement an in-depth learning paper. Select any one of the papers published in top conferences (NeurIPS, ICML, ICLR, ACL, etc). First, read the abstract and give a rough reading of the paper you have chosen. At this point **inform us** that you are attempting to implement the paper and tell us about the paper chosen. Once you get approval from us, you can go ahead and give a thorough reading of the paper and try implementing the paper using a framework of your choice (we highly recommend PyTorch).

We are not expecting you to write a very clean implementation but will be testing you on your understanding of the paper. Your code should be able to replicate the results described in the paper. If possible, add hyperlinks to accepted papers and link from Papers with Code

Document your code with a README and a review for the paper (preferably in Latex). The paper review might contain limitations of the paper, could the authors have done some changes to get better results and some new insights that you might have for the paper. Remember this task has not been designed for beginners but anyone is welcome to give it a go.