**Transferable Neural Networks and Automaton**

**Git-Repository:** <https://github.com/ovednagar/transfer-learning-and-automatons.git>

**Goal:**

In machine learning, the process of taking layers from a model that is trained to solve one problem and apply them to a different models in order to solve other problems is called Transferlearning, this process is not always successful. Some transfers end with improvement is accuracy and others end with no significant change or even with worse results. Our goal is to find out why it is working or not, and when should we use it.

We wish to focus on NLP problems. Because language is a complex structure its hard to find a connection between two different tasks, thus the question of will a transfer learning improve the accuracy? Is hard to answer for. So, instead of focusing on NLP tasks, we will answer these questions using automatons, and hopefully get some insights with respect to real world NLP tasks.

**Why automatons?**

Language is a sequential phenomenon, words occur in sequence over time, and the words that appeared so far constrain the interpretation of words that follow. Moreover, only certain sequences of words are considered to be "grammatical" in a language, i.e. in English the sequence "NLP is ambiguous" is a valid sentence, while "ambiguous NLP is" is not. Automatons are efficient and convenient tools for exploring linguistic phenomena, and the use of finite-state machines has already been shown to be successful in a wide range of linguistic tasks such as: lexical analysis, morphology and phonology, text-to-speech synthesis and speech recognition.

We wish to exploit the strong relation between finite-state machines and linguistics phenomena to get a better understanding of transfer learning application in the context of NLP. Unlike natural languages, languages that are defined by automatons are more controlled, therefor they are of great interest to us. They can allow to perform more accurate analysis of multi-tasking learning methods and to figure out when it useful or not, and why. Answering these questions in context of automatons can give us a more wide perspective of the same problems in the context of NLP.

**Demands:**

All code of the project will be uploaded to a Git repository

<https://github.com/ovednagar/transfer-learning-and-automatons.git>

Weekly updates of the project progress

**Work Plan**

* **A. build an FST module**the FST class should provide methods for:  
   - creating an FST module from the tuple - checking if a sequence belongs to the language of the FST  
   - creating random sequences according to the FST  
   - intersecting and union of FST's
* **B. Test randomly generated FST's with several Neural modules**- generate random FST's  
  - build binary LSTM model (in the language or not)  
  - build a Language model- test the models on the randomly generated FST's  
  - build a BI-LSTM for prediction of States within an FST
* **C. Research Transfer Learning**- Build a Multi Task model and find connection between the accuracy of the model and the different FST's we tried to learn with it

**Task 1.**

Build an FST module

**State class**

The first class that was created is the class that represents a single state. the class holds the following information about the state:

* Id: states name
* Type: is it accept/reject state
* Transition dictionary - The state is treated as weighted only if it has different weights for different symbols
* Method for generate random transition:   
  if it's weighted then the complexity is O(num-transitions)  
  if it's not weighted then it generate on using choice - assumed to be O(1)

It also implements the following methods:

* Method for Generate negative (not acceptable sequence) transition   
  **this function is not perfect yet**, it generates a transition to a target state  
  the target state is **not** utterly-accept state which means that there is a transition from the target state to a non-accept state. (its not perfect because the utterly-accept attribute is determined only by the transitions of the current state and not by the FST complete transition function, while for a single symbol a non-accept state is not reachable, there might be a sequence of symbols that will lead to a non-accept state, I'm using it that way because it's cheap (O(num-transitions) for initiation and O(1) for generating a transition), simple to implement and for this part of the project its seems sufficient enough)

The main class is the FST class

This class holds the following information about the automata

* The alphabet
* The states (including the division between accept states and the initial state)
* The transitions (as names to State objects, not the whole function)

It also implements the following methods

* \_\_str\_\_ for printing the FST using built in function
* A function go   
  if a sequence is given as an input then it checks if its accepted or not  
  otherwise it generates a random accepted sequence
* A function for generating negative samples, using the FST transition function.

Note: the FST adds states to the FST

* ART\_ACCEPT\_STATE: helps to terminate a random sequence generation
* REJECT\_STATE: helps to determine which transitions are legal for random generation

**Task 2.**

Union and intersection of a list of FST's  
Implemented using known method, complexity

**Task 3.**

Generate random FST

For a start we will try to generate a random automata with:



We will define randomly for each pair

The goal is to create an initially connected automata, so next will compute the probability for:

The probability for an edge in the graph of A is

,

I looked at the problem as:

Given a division of balls (k transitions) to cells (transitions targets), where each sequence of k cells is a block. What is the probability for at least one ball in a specific block?

We can calculate the probability for connectivity by "Random graph", E.N. Gilbert