BLG 435E - Artificial Intelligence HW3

Student Name	Student ID
Uğur Ali Kaplan	150170042

Problem 1

Problem Definition:

$$Init(\neg Passed(Deadline))$$
$$Goal(At(Homework, Ninova))$$

$$Action(Solve(Homework),$$
$$PRECOND : \neg Passed(Deadline)$$
$$EFFECT : Finished(Homework))$$

$$Action(Upload(Homework),$$
$$PRECOND : WebsiteOn(\text{Ninova}) \wedge Finished(\text{Homework})$$
$$EFFECT : At(Homework, Ninova))$$

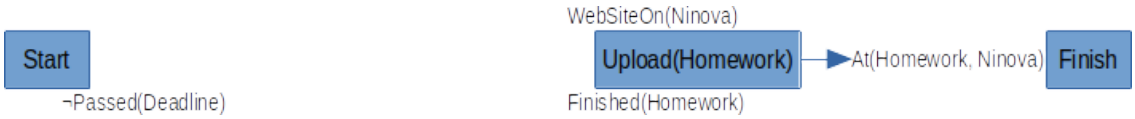
$$Action(StudyFrom(\text{Slides})),$$
$$PRECOND :$$
$$EFFECT : WebsiteOn(Ninova) \wedge Passed(Deadline) \wedge \neg Finished(Homework))$$

$$Action(Open(Ninova),$$
$$PRECOND :$$
$$EFFECT : WebsiteOn(Ninova))$$

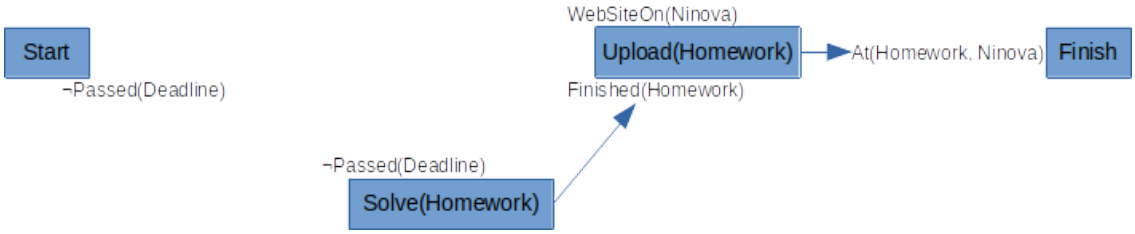
I will be solving this problem with partial order planning.

1. Create start and finish states.

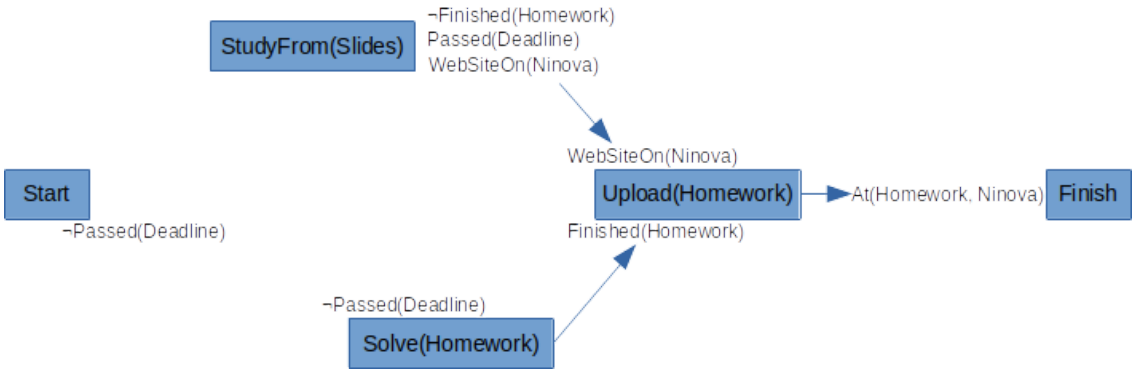
2. Pick an open precondition, namely $At(Homework, Ninova)$. Selected precondition is the effect of $Upload(Homework)$.



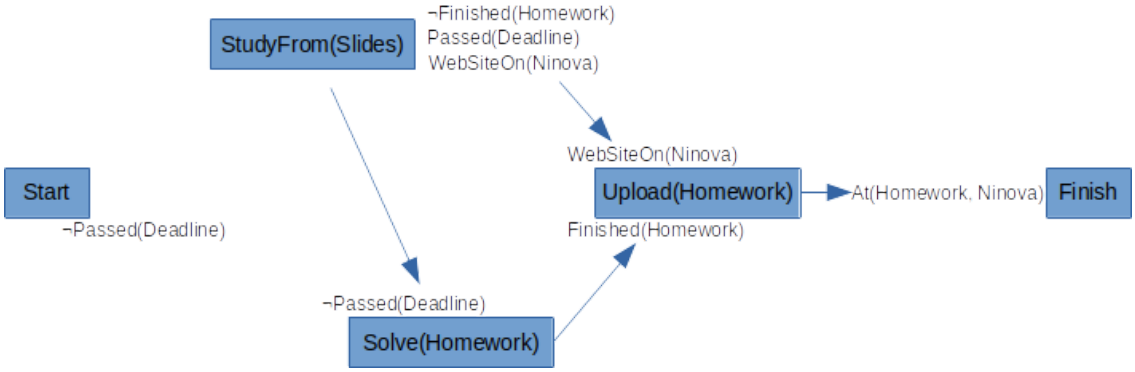
3. Pick $Finished(Homework)$. Selected precondition is the effect of $Solve(Homework)$.



4. Pick $WebSiteOn(Ninova)$. This is one of the effects of $StudyFrom(Slides)$.

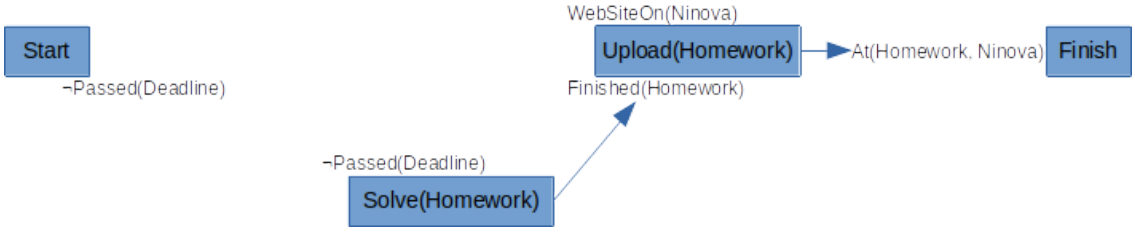


5. Since $StudyFrom(Slides)$ also have the effect of $\neg Finished(Homework)$, create a causal link with $Solve(Homework)$ to remove the conflict.

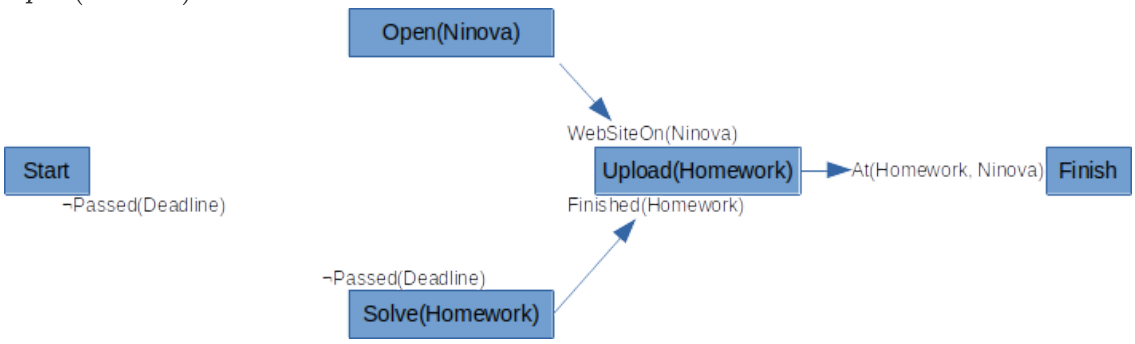


6. But since $Passed(Deadline)$ is also an effect of $StudyFrom(Slides)$, we cannot find another ordering of the selected actions. **Backtrack** and remove $Solve(Homework)$ as well as $StudyFrom(Slides)$.

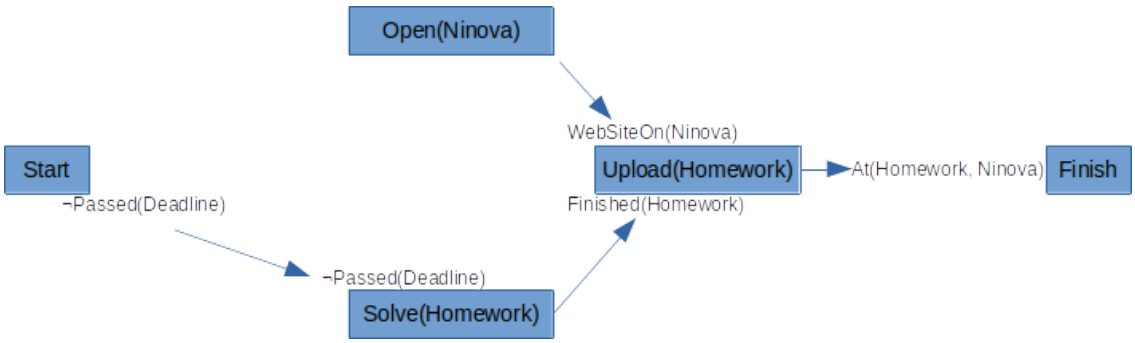
7. Again, add $Solve(Homework)$ for the $Finished(Homework)$ open precondition.



8. This time, pick another action for $WebSiteOn(Ninove)$ precondition. We select $Open(Ninova)$.



9. There is no conflicts. Connect $Start$ with the open precondition $\neg Passed(Deadline)$.



10. Planning problem solved.

Problem 2

1.
   - **Overfit**: When we train the model, our model may be overparameterized for model, and might learn everything in the data perfectly, even the noise. If a model learns its training data perfectly, we do not expect it to generalize well to the unseen data.
   - **Underfit**: If our model is too simple, or our assumptions about the data is wrong, our model might be unable to learn from the training data and its performance is not good whether we are measuring its performance on the training or the validation set.
2.
   - **Online Learning**: System is trained one by one, or in small batches, and is able to learn incrementally. An example is neural networks.
   - **Offline Learning**: System must be trained with the whole data, and cannot learn incrementally. An example is matrix defactorization models.
3. Our data should be representative of the real world but it can never fully represent the population. Therefore, by dividing the data into training and testing sets, we get two samples from the same distribution.

   If our model learns well in the training set, we assume it will also work well in the real world (real distribution of the data). To test our assumption, we use another sample which is also representative of the real world data distribution and our model's performance in the testing set will give us an idea of how it will perform in the real world with unseen data.

4.
   - **Supervised Model**
     - **Output**: Output will be bump (1) or not (0). Then, robot can select a direction to go to, and continue to move in that direction until it detects there will be a collision. So, robot will use the system as, should I go to left? If output is 0 (no bump), it should go to left. Input can be a random direction.
     - **Training Data**: I would create a software to send the robot in random directions for a randomly determined duration. So, the agent would go left for 5 seconds, then randomly go forward for 7 seconds and so on. In the end, after the robot moved around a bit, I can collect the sensor data from the bump and the laser scan sensors.

       At the end of this process, I will know which direction robot was going to, if there was a bump or not, and what is the environment situation as assessed by the robot at each timestep.
     - **Training Process**: I will supply the direction and the data of the laser scanner sensor to the model. It will try to guess the signals bump sensor is receiving.
     - **Algorithm**: We can use a convolutional neural network for training.
   - **Reinforcement Learning**
     - **Output**: Output will be the direction to go to.

- **Training Process**: State will be the data coming from the laser scan sensor. Actions will be to go to the left, right, forward or backwards. We will be trying to learn the optimal policy. Since there is no target, there is no need to define a score for movement. But there will be a negative reward if there is a bump and network will try to find the optimal set of actions and pairs.

  We can use a simulator for easier training.

  Instead of consecutive samples, we need to use random minibatches.

- **Algorithm**: We can use the Q-network given in the lecture slides for playing a game, but it will be using the data from the laser scanning sensor in our case.

Problem 3

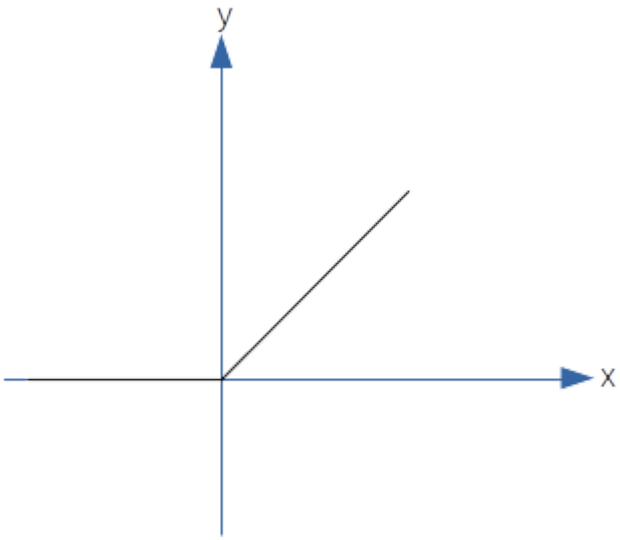
1. Loss function is used to determine the "loss" of the ML model, in other words, the numerical representation of the failure of the model. For example, if we want to measure how unsuccessful the model is in a classification task, we might use 0-1 loss.

   Let's say, we have 100 samples and our model correctly predicts 86 of them. Then, its 0-1 loss is

   $$\frac{1}{100}\sum_{i=1}^{14} 1 = 14\%$$

   This is how unsuccessful or model is, in terms of accuracy. Loss functions are also used to optimize the model during the training and for that purposes, 0-1 loss is not a good candidate because it is hard to optimize for. Therefore, we could pick other loss functions, such logloss, mean squared error and so on.

2. Neural networks are combination of perceptrons, which are linear models. Therefore, if you pipeline a number of perceptrons, you will be learning a linear function. To introduce non-linearity to the model, we use activation functions after getting the output of a perceptron. Activation functions are nonlinear. An example activation function is Rectified Linear Unit, shown below.

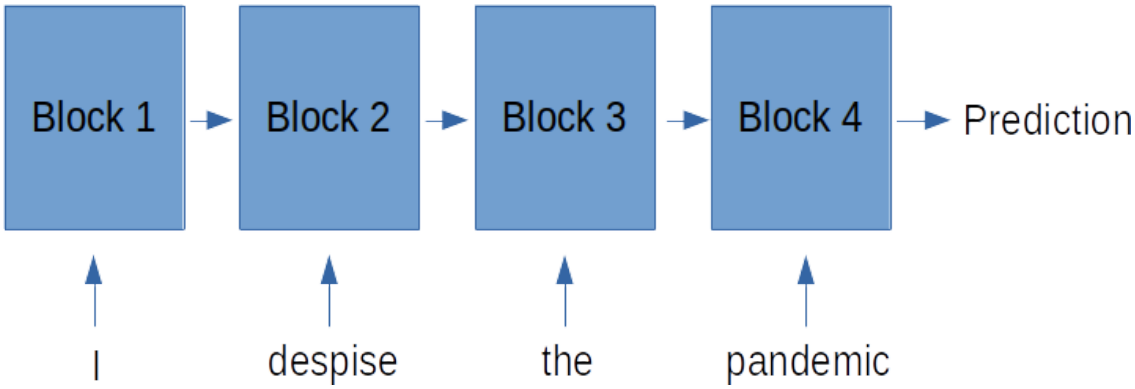


3. **Convolutional Neural Networks (CNNs)**: CNNs are mostly employed for computer vision tasks. This is because their main operation, namely convolution, is most suited to the tasks in which close features are related to each other. Since close pixels are related to each other, CNNs work well in computer vision tasks.

   Recurrent Neural Networks (RNNs): RNNs are mostly used in natural language processing, but they are suited to any type of sequential data assuming the data is stationary. The reason lies in the structure of the RNNs.

RNNs are built upon pipelined and jointly trained neural network blocks. Let's say we have an RNN consisting of 4 such blocks and our input is "I despise the pandemic and the increased workloads.".

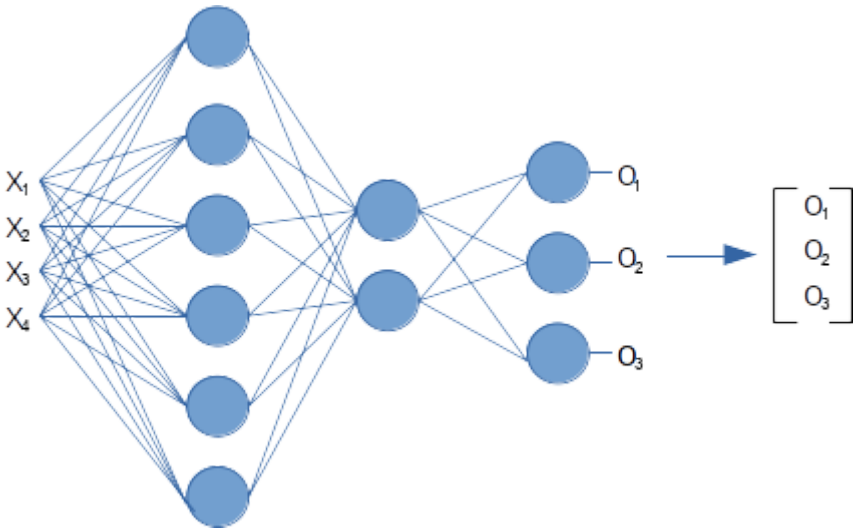
Then, for the first iteration of the training, our RNN model will have the the following structure:



Each block except the first one is trained with the output of the previous block in the sequence as well as the word in the corresponding order in the sentence. Therefore, prediction made by the block 4 accumulates information from the previous blocks and uses that to make a prediction. Thanks to this information accumulation from the sequence, RNNs work well in sequential data.

Note: The RNN figure shows the "unrolled version of RNNs", together they create one RNN block.

4. I have used an input layer of size 4, and a hidden layer of size 2 and finally, our output layer is of size 3.



As a loss function, I could use negative log likelihood. In order to do that, I should first apply softmax to the outputs.

Softmax:

$$f(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

where $k = \text{Number of classes}$ and $z_i = \text{Output i}$.

Negative log likelihood:

$$\sum_{i=1}^{K} log(1 + e^{-f(z)_i y_i})$$

In the end, when a new sample comes in, we get three outputs for each class. Whichever is the maximum of the three will be the answer of the network.

For example, if our outputs after are $[0.2, 0.7, 0.1]$, our answer will be 2 since 0.7 is the largest.