Supervised models for an OpenAI driving game

General Machine Learning
Philippe FRAISSE - philipp2@stanford.edu

***Abstract*—Using the Open AI framework, we will use supervised learning with support of linear models and deep neural nets, in order to try to achieve honorable score at playing a driving game without human intervention.**

I. Introduction

When it comes to make an AI drive, there is an increasing need of data, because of the diversity of the task and the complexity of the model. At the same time the data is scarce and costly, as it can be expensive to pay an engineer to drive a modified car to gather data. For this kind of cases, the increasing capacity of computers in physics and graphics made simulation a better and cheaper approximation of reality. To peek into this area, this project will be about making an AI learn to play a simple driving game using the Open AI universe framework. The goal is to build a model that is able to achieve honorable game scores compared to scores of a non expert human player.

This project will have two main objectives. First, to build an end to end supervised learning model based only on randomly generated data. Second is to apply this deep learning model on another same objective driving game. The aim is to explore the transformations that could be made to the model's representation of the game, in order to exploit what was learned previously.

All along the project, several baselines will be used to benchmark results. The random action baseline : to verify the relevance of the results. The supervised learning baseline : which is a simple first attempt at the task to measure it's difficulty. The oracle will be the human level score in the game.

II. Related work

In this project we chose to see how far we could push supervised learning techniques to achieve high scores at a driving game. Chen and Yi [1] showed us the interest and possibility of using supervised learning in game context, to achieve high score without using reinforcement learning, with cheap computation and good generalization. They used very deep convolutional networks to achieve that, which have shown as like in the VGG implementation [2] their ability to process very well image related problems.

III. Environment, Data and Features

A. Environment

We will use OpenAI Universe, which is a convenient framework to manage input output of a game. It provides a large catalog of different types of games to build an AI on. Our choice will be a car game named "Dusk Drive" which is a timed race of approximately one minute, during which we need to take turns to stay on the road and avoid vehicles that we can encounter randomly. There is no possibility to "die" (ie infinite negative reward), all rewards are positive and going out of the road simply makes the car go slower. In this game the rewards are indexed on the speed.

To train our models we will use a Microsoft Azure Ubuntu 16 server with 6 cores, 56go of RAM and an Nvidia Tesla M60 for accelerated tensorflow computations.

B. Data generation

To generate data, we run random plays of the game to generate a 3-uple of (screen pixels, keyboard actions, reward) that will be used to learn how to play. We will then shift to data re-generation from the model, and compare results. That is, we would iteratively generate data with the current model, keep only the best games (one with highest final score) and retrain the model on it.

The dataset for the baseline model consists of 3706 frames and actions of random play. The dataset for the experiments consists of 87000 3-uples. The actions are uniformely distributed and the rewards are strongly right skewed since the actions are random, few of them are really rewarding.

C. Features

As we will focus on end to end models, we will try to keep the feature engineering at its minimum. The generated data is cropped from (768, 1024, 3) to the effective size of the game screen, that is (505, 795, 3). Then the frames are down sampled, to a drastic (5, 5, 3 averaged color channels) for the baseline model, and to (169, 265, 1 gray scale channel) for the other models. The screen captures are taken 3 times per second. The baseline training data is over 20 plays, the experiments training data is over 500 plays.

Examples of the described features can be seen in figures 1, 2 and 3

The frames are then arranged into states. Each state spreads across 5 time steps. It is composed of : 3 frames f(t-2), f(t-1), f(t), the current action a(t), and future rewards aggregated on r(t+1) + r(t+2). The set of 3 frames is meant to capture the motion and speed information. The aggregation of rewards is to capture the future rewards at an horizon of 2 time steps, which empirically has been a good compromise between the need to measure the immediate effectiveness of the action, and the expectation of futur rewards.

The frames are standardized column wise, that is each pixel is substracted by its mean value and divided by its standard deviation accross the training dataset.

Fig. 1. Original cropped screen of the game

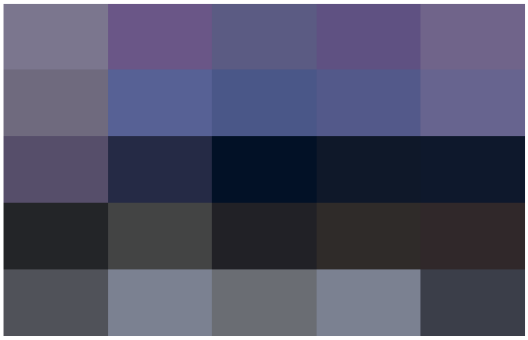

Fig. 2. Frame transformation used for the linear baseline model

Fig. 3. Frame transformation used for the experiments

The development set will be the results obtained by playing the game. We will process the data in the same way and standardize it with the statistics obtained on the training data.

IV. Supervised Learning methods

The objective of This part is to see what level of game scores can achieve a model that is only based on randomly generated actions. It can seem weird to not use reinforcement learning for this task, this is because this approach is studied in parallel in a sister project for cs221.

Excepted for the baseline, in this part we will focus only on end to end models. We will use neural networks to train our models, and try different architectures : fully connected neural network, convolutional neural network, recurrent neural network. We will also try to use a pretrained convolutional network and train the top layer. In order to compare the adds of each model, we will use the fully connected network as a top layer for the convolutional and the recurrent ones.

A. Baseline linear model

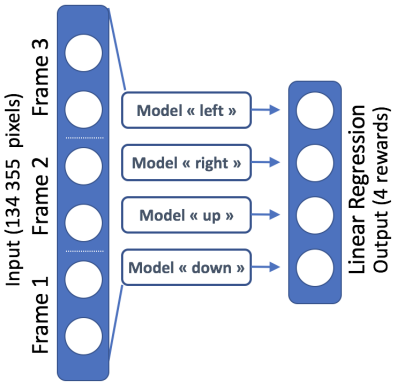

Fig. 4. Desciption of the baseline modeling

1) Method: In order to have a first estimate of the difficulty of the task, we built a model based on the very simple representation of the game states as we have shown. For this baseline, we have used one linear regression model per action (4 actions in total : up down left right) and train each on the states with corresponding action, as described in figure 4. We use a mean squared loss against the rewards in the partition of the dataset containing only one action type for a given model. We then evaluate the performance of our model by averaging the obtained scores over 20 plays. Each model outputs an expected future reward of the corresponding action, given a vector of screen pixels. During the test plays, each decision is based on the highest predicted action reward over the 4 models every 1/20 second.

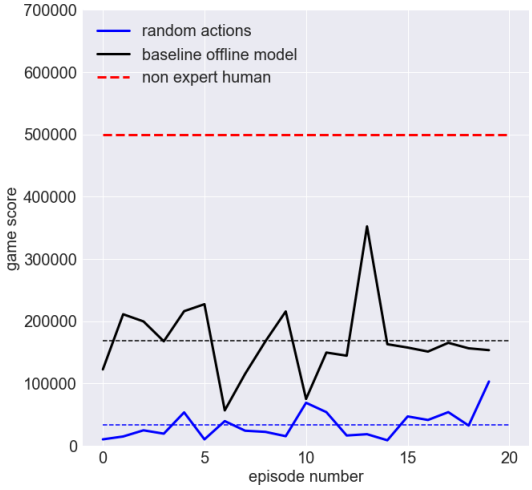

Fig. 5. Results of the baseline model on 20 plays

2) Results: In figure 5 we can see the performance of this supervised learning baseline model. This simple model applied to an overly simplified input of the problem is capable

of an average score of 170 000 versus 33 000 for a play with random moves. The model shows some capacity to stay on the road, make turns and avoid vehicles. Nevertheless, it sometimes rushes into barriers or gets stuck on the side of the road. This could be linked to the frequency of this cases in the random dataset which is essentially low reward situations where the car is out of the road. Also, to go back on the road, a sequence of actions has to be made, like pressing the same key multiple time, which makes it more difficult for the model to make good predictions, since it does not make this kind of sequence prediction. To address this we tried to train the baseline on higher filtered rewards of the random data, or on regenerated data from the model itself (data with a smarter play). Surprisingly this did not yield to better results. For the sake of conciseness we will not present the results of this investigation, but it seem this has to to with the exploration. As for reinforcement learning , in order for the model to learn, it needs to have an extensive view of the world states, and not just the best ones. As in our case, the model would not know what to do if it is out of the road, if this kind of data was not present (or more scarcely represented) in the training data.

B. Fully connected neural network

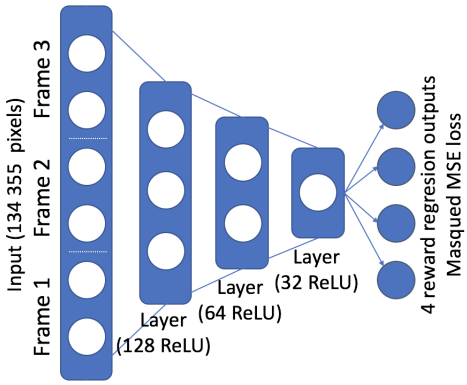

Fig. 6. Description of the fully connected neural net

1) Method: In this part we will use a fully connected neural network to predict the rewards of each action given the last 3 frames. As described in figure 6 the model is composed of 3 hidden layer, with a ReLU activation function. The loss function is a masked mean square error.

$$\text{Masked MSE} = \frac{1}{n}\sum_{i}^{n}\sum_{a \in actions}(\hat{Y}_{i,a} - Y_{i,a})^2 \mathbb{1}\{Y_{i,a} > 0\}$$

$$ReLU(x) = \max(0, x)$$

The term masked is used here to say that, since in each training example, the truth value of the output is a vector with 3 zeros and 1 reward (only one action at a time is considered per observation) we do not want the loss function to push the other rewards towards zero. For this reason we mask the output layer by multiplying it with zeros were the truth value is zero as well (for the non evaluated actions). This way the penalty that is computed is only function of the current action and its predicted value. The other values are thus ignored. We use an Adam optimizer to take care of the gradient norm and the decay of the learning rate. This model may seem a small neural network, but the size of the input makes it already a consequent number of 17 208 036 parameters. We made this choice to avoid a too flexible model, regarding the modest amount of data. The development scores (results of computed model in game play) are computed for each epoch over 3 plays of the game. This is used for early stopping of the training. This is repeated for different values of the learning rate. Here we had the best result on development scores for a learning rate of 0.001, at the epoch 10.

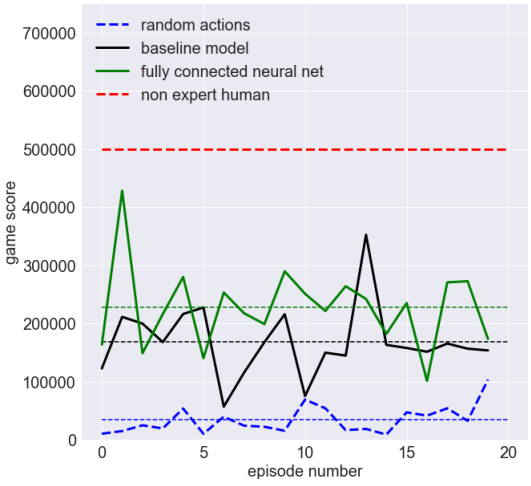

Fig. 7. Results of the fully connected neural net on 20 plays

2) Results: As we can see in figure 7, this model achieved an average score of 230 000 points, which is 7 times better than the random plays from which it is trained from, and 40% better than the baseline model.

C. Convolutional neural network

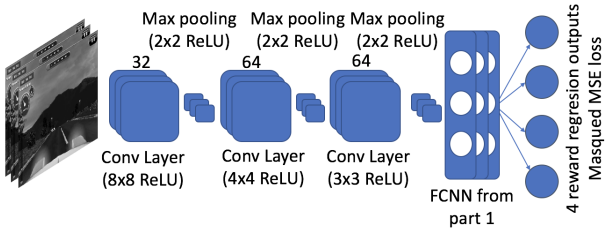

Fig. 8. Description of the convolutional neural net

1) Method: Convolutional network are known to achieve very strong performance at image recognition tasks. Here we will investigate how useful could be features computed by such networks, as inputs of the fully connected neural network we used in the preceding part. As we can see in figure 8, we will use a basic convolutional network with 3 successions of convolution and max pooling. Respectively (32, 8, 8), (64, 4, 4), (64, 3, 3) and (2, 2), (2, 2), (2, 2). We use ReLU activation on the max pooling layers. As before, we train the model on different values of learning rate and monitor the performance on real play

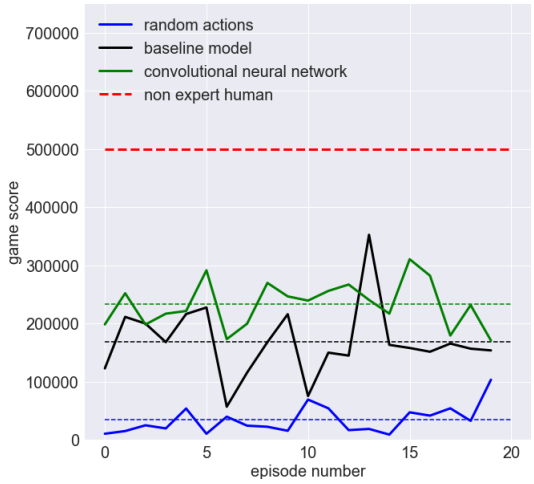

Fig. 9. Results of the convolutional neural net on 20 plays

2) Results: Here the use of convolutional layers as input to the fully connected one did not yield to a significantly increase of game scores versus the top layer alone. Performing on average 232 000 vs 230 000 for the FCNN alone. As we are new to the use of this method, we might have missed something in the implementation. Another hypothesis is the use of the channels for temporal frames, which is not the usual way to use these convolution layers, rather than for different color channels of the same image.

D. Pretrained Convolutional neural network

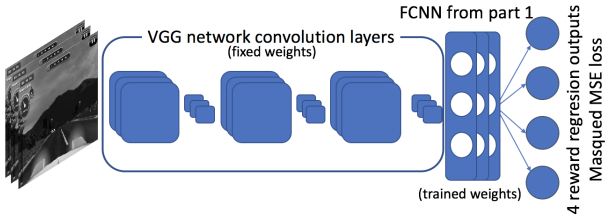

Fig. 10. Description of the use of a pre-trained VGG type convolutional neural net

1) Method: Here we will have a peek into transfer learning, since we did not get good results on our convolutional network, we will try a deep pretrained one, on top of which we will put our fully connected neural network from part 1. For this task we will use a pre trained VGG network. VGG is a very deep convolutional network for object recognition, created by the Visual Geometry Group of the University of Oxford, that is famous for performing well on the imagenet dataset. We will not retrain the parameters of the VGG network (stripped of its top layer). We will not do fine tuning on the top layer neither, these parameters will be initialized and trained like a new model. Here we will try to see if features extracted by a pretrained static network are useful for our top layer, and if the knowledge contained in the pre training, which has nothing to do with this problem, can be transferred to our driving problem.

2) Results: Actually the results were higher than human plays, but unfortunately it was not for a good reason. We don't know if it is an implementation problem or some kind of convergence, but it turns out that playing always 'up' and bouncing on walls and cars can yield to very high results. For honesty concerns, we each time discarded those types of models that even if they were achieving high results, were not really driving.

E. Recurrent neural network

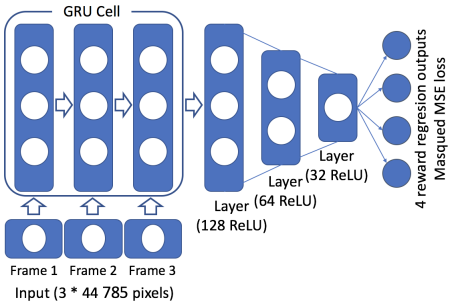

Fig. 11. Description of an RNN model with our fully connected network as a top layer

1) Method: Since we have a temporal structure in our states, because we use the last 3 sequence of frames to estimate which is the next action that will give the highest rewards, we want to exploit this through a recurrent neural network. As we can see in figure 11, this type of architecture takes sequentially the temporal data, and updates its transition parameters, which are used to propagate the information from one time step to the other. Thus at each time step, the internal layer of the RNN cell merges the information of the current time step, and all those before, according to a system of gates that lets pass some information, update, or delete other. The last RNN layer, ie the one of the last time step has encoded the sequence in its whole, and will back propagate the error, in a way that the gates of the RNN cell will, update, delete, transfer and merge the information of the time step more relevantly regarding to the output to predict. There are different types of RNN and ways to use them, here we will use a GRU cell, to limit the number of parameters, and speed up training. We will compute the internal representations only in forward direction since this is the direction of the motion, and will use an internal representation of dimension 500. After doing hyper parameter search according to the playing performance, we found that the tuning with the highest results was with a learning rate of 0.01 a dropout rate of 0.5 at epoch 10. It is to note that we used a stochastic gradient descent optimizer this time, since it seemed to perform better.

2) Results: We can see in figure 12 that the use of the GRU cell as feature for our FNCC brought useful information with an average of 280 000 points, this is a 20% increase versus the score of the top layer alone. This is 8.5 times the score of the random data it has been trained from.

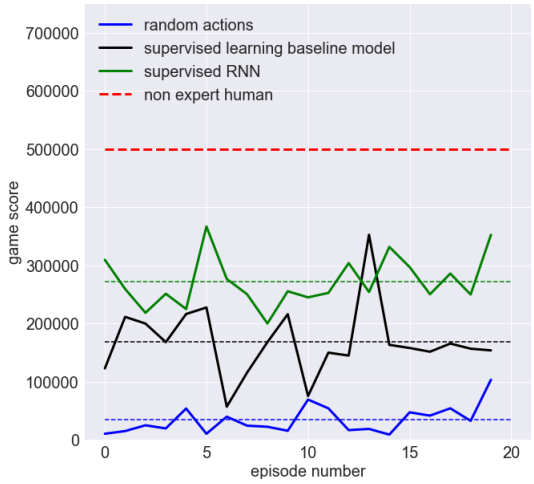

Fig. 12. Results of the recurrent neural net on 20 plays

Fig. 13. Processed frame of "neon racer" the game on which we want to apply transfer learning

V. Transfer learning experiments

In this part we will try to reuse the RNN model in the previous part on another driving game with the same objectives and actions but with different graphics. Here we would like to keep the model fixed, and explore transformations to the input data in order to adapt perception to the new objects. As for word vectors in NLP: the embeddings represents the semantics of the word, and can be re-trained on a specific corpus to have their sense specialized. We will explore ways to re-train / specialize the input representation to make it adapt to the new graphics of objects. We will compare game scores between our pre-trained model, the same model with no training.

A. Method

Here we will reuse the best model from the previous part, and try different fine tuning strategy on random plays of the new game. The fine tuning consists in adding training to some parts of the model, while keeping others fixed, and with a smaller learning rate in order to only bend the prediction, and not overwrite the knowledge the model already has. To do so we will consider fine tuning first only on the RNN part, then only the top layer part, and finally on both RNN and top layers. To assess performance, as before, we will compare a set of hyper parameter at different epoch and will chose the ones that yields to the highest results.

B. Results

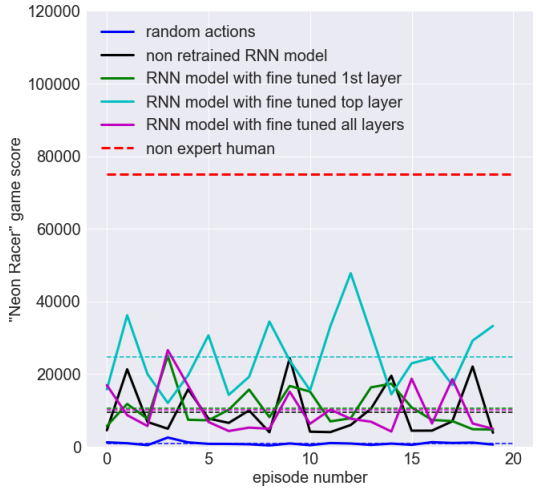

Fig. 14. Results of different strategies of fine tuning a pretrained model

As we can see in figure 14, retraining only the top layer yielded to an average of 25 000 points, that is 2.5 times better than without fine tuning, and 27 times better than the score of a random play. Retraining the RNN or both parts did not led to results that are significantly different than without fine tuning. This may be because the first part has much more parameters than the second, and thus need much more training to make a significant update. It is to note that this game is slightly harder than the first one, since going out of the road leads to a reset to the center loosing all speed, while in the first game, it was possible to drive out of the road as long as we wanted, while accumulating small rewards.

VI. Conclusion

It would have been fair to do an entire project on only one type of those model, since the range of transformations, optimization, architectures is wide. Here we tried to have our hands on neural network modeling, and see a broad range of possibilities, and potential problems tied to each type of architecture. We retain the relevance of the RNN architecture for temporal sequences which brought some tangible improvement to our base model. Concerning the convolutional networks, their complexity and difficulty to train makes us think that, like with embedding for NLP, it may be smarter for problems involving images to import a very deep pre trained network as AlexNet or VGG, and train our classifier or regressor on top of it.

The models we have trained could have been larger regarding to the size of the state space, and the complexity of our task. Some training loss indicator seemed to tell that we might had some bias in our model. This choice was made because we wanted to try different types of models, and not different architectures within a model. Moreover, Training

and tuning neural network is challenging, in order to provide the following results, we have trained, stored and evaluated more than 320 models. Morevover, since we are evaluated our models on games plays, it was more time consuming than it usually is for supervised learning.

As future work, we would like to pick the RNN model try to improve its architecture, or add some features like histogram of oriented gradients. Features were useful in the RL part of the sister project, where using a vanishing point estimation was of great help to know where to turn. To compare, our RL models reached around 370 000 points on "Dusk Drive" while supervised learning alone was 90 000 below, which is not too bad regarding the difference in input information. Many states have almost zero reward, since the random play often struggles around out of the road. We estimate less than 10% the number of states on the road, making rewarding decisions. Finally, we would like to use some object detection like YOLO as input features. We think also that for the transfer learning part, abstracting the graphics with features would help greatly to generalize.
Thank you for reading.

REFERENCES

[1] Zhao Chen, Darvin Yi. The Game Imitation: Deep Supervised Convolutional Networks for Quick Video Game AI.
[2] Karen Simonyan, Andrew Zisserman. Very deep convolutional networks for large scale image recognition.
[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller. Playing Atari with Deep Reinforcement Learning.
[4] Fabian Chan, Xueyuan, You Guan.Deep Q-Learning on Arcade Game Assault.