Using Reinforcement Learning on Datasets with Progressive Image Compression to Show Most Important Features for Predicting Labels

Ammar Manazir
u6585552@anu.edu.au
Research School of Computer Science
Australian National University
Acton, ACT
Australia

Abstract

This paper aims to implement the technique of progressive image compression, extended by the use of reinforcement learning, on a dataset to train a neural network to identify which class an entry belongs to. The goal of this was to maintain as much accuracy as possible whilst compressing the dataset as much as possible. This was attempted by changing the number of hidden neurons per epoch, and comparing accuracy values, to see whether significant accuracy was gained or lost, or whether change was minimal, and thus identify which are the significant features. After this, a reward signal would be used based on the decreasing accuracy loss over the epochs. This would in, theory, allow improve accuracy of the compressed dataset. The results of this method are, unfortunately, quite poor, due to a poor implementation of the method, and do not provide any insight on their own, and instead, research was done into successful experiments with similar methods.

Introduction

I have selected the dataset ‘sars-cov-1’, which contains data about the blood pressure, temperature, nausea, and abdominal pain of the subjects. The reason I have selected this dataset is because it has a good number of features and a large number of entries, which gives me a lot of data to work with during my investigation. There are 1000 entries for each of the mentioned features (i.e., blood pressure contains 1000 different patient’s blood pressure information, temperature contains 1000 different patient’s temperature information, etc.). Each individual entry is a float value between 0 and 1, for example 0.2427.

Another positive is that the scale of the data is all similar, all being float values between 0 and 1, and thus saves me the step of normalising the values in pre-processing, as all of the data is of the same type and in the same range. I will add another column to the data sets, explained in the method, which will be an integer from 1 to 4.

Besides the technical reasons for selecting the dataset, I believed that this dataset was also a better cause to investigate and could help hospitals out in triaging patients if it can be accurately predicted whether a patient is one of the mentioned 4 classes. It can also prevent potential cross contamination between patients with different illnesses if the network can identify who has an illness and needs to be isolated.

The goal of this work was to classify each subject as one of four classes: normal, high blood pressure, pneumonia or Sars. I modelled this problem in my neural network by first combining the four data sets into one, with a new column to identify which data set each entry was originally from. After this, I split the data into testing and training, and used them in conjunction with NumPy and pytorch methods to come up with the different weights, and then used these with an autoencoder, which I will go into more detail on in the method. After this, the features are compressed and evaluated for their accuracy, after which I will vary the level of compression per epoch. Finally, a reward signal is set up for when accuracy loss decreases across multiple epochs.

I believe this task is important to tackle because it has the potential to help hospitals with triaging patients by training the neural network to identify whether a patient has a number of medical conditions, as mentioned before, and this could potentially reduce the burden on the staff. This network is being used on around 4000 entries, which is far more than what most hospitals can handle, and if the model can accurately make predictions, it will work significantly faster than staff can, and alert them to potential diseases spreading in patients faster.

Method

The technique I am using is known as Progressive Image Compression, which, broadly speaking, breaks down data into its most important features whilst maintaining accuracy to its source, such as when used with images, compresses an image down to where it still contains features that would most significantly define the image. So, if one were to view the compressed image, they could immediately tell that it is the same one as the original, or as close to it in accuracy as can be. In the context of this task, I will be attempting to maintain accuracy of the network whilst hiding more neurons (compressing the dataset).

The task I have set out to complete is to have the neural network identify which of the 4 classes a data entry belongs to. I will use an input layer that has every feature, same as the output layer, and implement the technique using the hidden layer by increasing the number of hidden units and seeing how much accuracy is maintained. If there is a larger drop in accuracy, we know the particular feature was more important, and if not, it can be ignored and thus the dataset compressed. For every epoch, I will vary the number of hidden units in order to have varying levels of compression.

Finally, I will implement reinforcement learning. Reinforcement learning is when an agent learns how to make the best possible decisions over time through the use of rewards and penalties. The feedback that the agent receives serves as the reward for actions taken in a particular state, with the goal being to maximise the cumulative reward. The initial action taken is random, given the agent could not have any rewards at the time and thus no incentive to pick a certain action. The long term return for the agent in a state is known as the Q value. This is all how reinforcement learning functions, but when it comes to the context of a neural network, this is slightly different, although as can be seen from Figure 1, the basics are the same. Normally, a table is used to get the Q values for an agent, however when used in a neural network, the network predicts the Q value in a given state instead, as it would be very inefficient to manage Q values in a table. This is because the network could have thousands of actions and states.

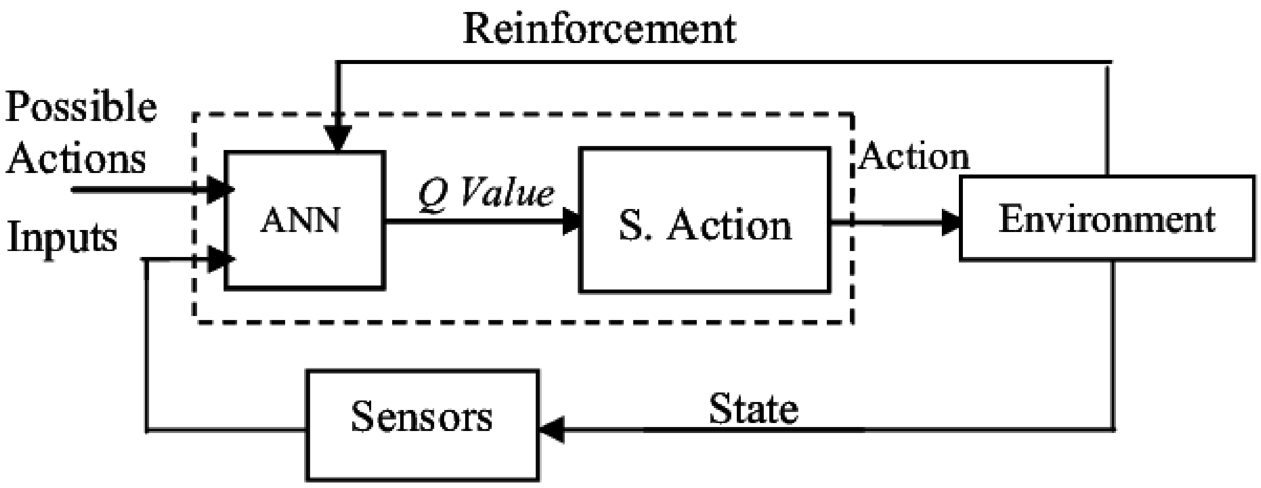

Figure 1: Reinforcement Learning in a Neural Network [3]

I went about this task by first by first combining the 4 datasets into one large dataset, with an added column to distinguish which dataset an entry originally belonged to, and used this as the prediction feature, as this is what the model would be attempting to predict. After separating the first 23 features, which contain the information of the dataset, and the last feature, which was the one we were predicting, I then use the sklearn package to generate the testing and training data splits.

I realised that due to there being 4 classes in the output, I could not use a sigmoid activation function as initially planned since it showed the probability of data belonging to a certain output class. Instead, I used a SoftMax activation function, as it worked well when the output classes were mutually exclusive, and no output could be part of multiple classes. It functioned as multiple sigmoid functions, with there being a probability given for data belonging to each of the output classes given in a list.

I decided to implement an autoencoder to control the number of epochs, batch size and learning rate, as I knew from prior research that this would help with this particular implementation. I had previously attempted the progressive image compression technique with no success, where I had found that the use of an autoencoder could have benefitted me greatly.

A paper by Aman Kawatra talks about how he used progressive image compression, in conjunction with an autoencoder, to achieve results; in his model, the accuracy did not drop much when the incoming and outgoing connections from the hidden layer units, that were found to be irrelevant, were removed from the network [4].

Unfortunately, I was unable to get my neural network, nor auto encoder to work sufficiently. In theory, the strategy should work as proven by Aman's work, however in practice I was unable to recreate the functionality. Since this was the case, it also meant that any attempts at utilising reinforcement learning was also unsuccessful, as the network itself did not work.

Results and Discussion

As I was unable to get any meaningful results from my implementation, I will instead talk about another's work, which was successful in implementing my proposed technique and method, with the exception in that he did not use reinforcement learning like I had attempted.

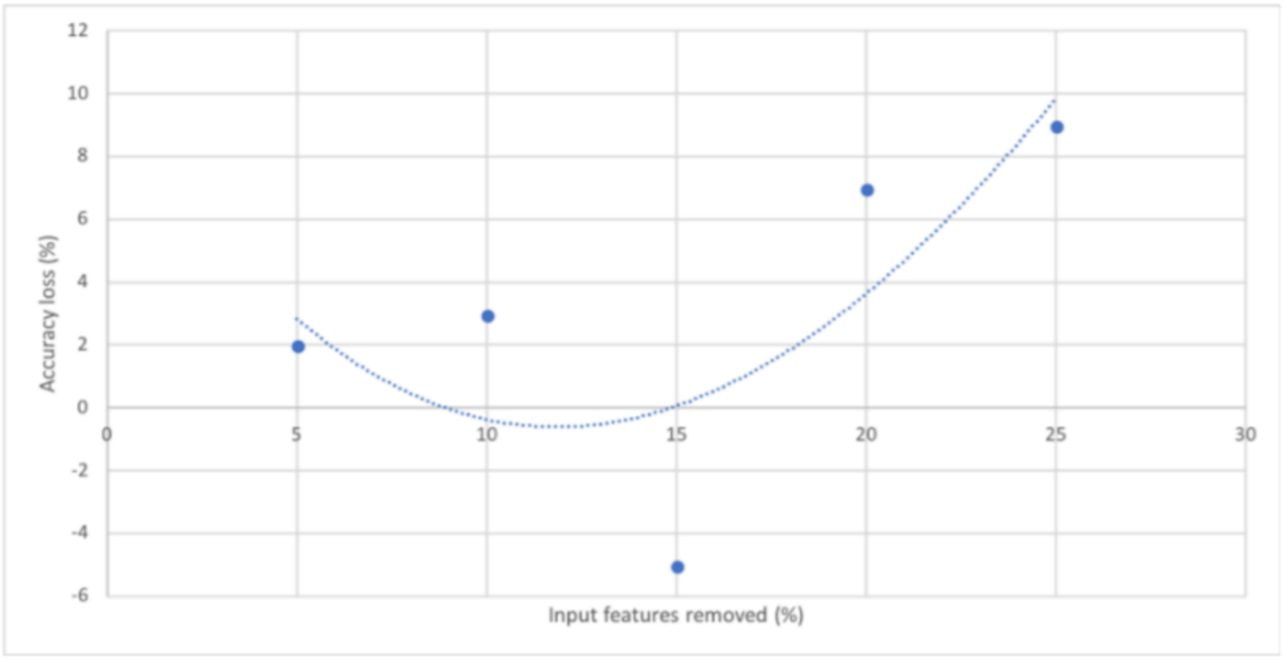

Figure 2: Accuracy Loss vs Input Features Removed [4]

Above, in Figure 2, are the results from Aman's method. His results show that, overall, as the number of input features removed increases, the accuracy loss increases, which makes sense given that when features are removed, it potentially makes it more difficult for the network to predict the correct results. However, these results also show the potential for progressive image compression, as at 15% input features removed, the accuracy loss actually decreases significantly, showing that if only the more important features are used, as is the goal of the technique, the accuracy also improves.

Conclusion and Future Work

In conclusion, whilst my method was not able to produce any meaningful results, others have used this technique in conjunction with a non-image dataset to extract the most relevant hidden information. This shows the potential for using progressive image compression, and that it can be used to solve the task I had set out to at the start of this paper. Unfortunately, I find it difficult to make any conclusions with regards to my attempted use of reinforcement learning beyond it working in theory.

In the future, I will get a better understanding of the technique I wished to use, as well as how much a neural network can be broken down and changed, so that I can experiment more and find more ways to affect the accuracy of the model. I will also extend the technique by using reinforcement learning to help the network identify the core features faster and more efficiently.

References

[1] Cornell University, 2016. *Designing Neural Network Architectures using Reinforcement Learning*. [online] Cornell University. Available at: <https://arxiv.org/abs/1611.02167> [Accessed 31 May 2021].

[2] Gedeon, T. and Harris, D., 1992. Progressive image compression. Baltimore, MD, USA: IEEE.

[3] Hatem, Mezaache & Abdessemed, Foudil. (2009). Simulation of the Navigation of a Mobile Robot by the QLearning using Artificial Neuron Networks.. 547.

[4] Kawatra, A., n.d. Neural Network pruning using hidden layers output’s unit vector angles and autoencoder for feature selection. [online] Available at: <http://users.cecs.anu.edu.au/~Tom.Gedeon/conf/ABCs2018/paper/ABCs2018_paper_95.pdf> [Accessed 27 April 2021].

[5] Mendis, B. S., Gedeon, T. D., & Koczy, L. T. (2005). Investigation of aggregation in fuzzy signatures, in Proceedings, 3rd International Conference on Computational Intelligence, Robotics and Autonomous Systems, Singapore. <http://users.cecs.anu.edu.au/~Tom.Gedeon/pdfs/Investigation%20of%20Aggregation%20in%20Fuzzy%20Signatures.pdf>

[6] Theis, L., Shi, W., Cunningham, A. and Huszár, F., 2017. .

[7] Toderici, G., Vincent, D., Johnston, N. and Jin Hwang, S., 2017. Full Resolution Image Compression With Recurrent Neural Networks.

[8] Yu, T., n.d. [online] Available at: <http://users.cecs.anu.edu.au/~Tom.Gedeon/conf/ABCs2019/paper/ABCs2019_paper_v2_82.pdf> [Accessed 27 April 2021].