Reinforcement Learning Applications in the Field of Progressive Image Compression

David Dunbier[1],

[1]Research School of Computer Science
Australian National University
Acton, ACT
Australia
{David.Dunbier@anu.edu.au}

Abstract. Classification is a task well suited to neural networks, as the many different factors that inform a classification decision are repeatedly judged on their importance and re-assessed in the training process. However, as classification tasks get larger this becomes a relatively expensive process, requiring significant amounts of energy and time as the input data grows larger.
There are likely features of the input data that can be ignored for the task of classification, as they may prove worth less to the final decision than other features. This paper sought to utilize a progressive compression method to input data and determine its effect on a classification task. Furthermore, this paper sought to implement a reinforcement learning method to evaluate the best ratio of compression to apply to the images, using the accuracy of the classification task both before and after compression to determine the value of further compression. While this paper was successful in implementing the compression and classification tasks required and could display the relative loss in accuracy that compression invokes, the reinforcement learning method was unable to be implemented.

Keywords: Neural Networks, Autoencoder, Compression, Classification, Accuracy, Reinforcement Learning

1 Introduction

Classification tasks performed by neural networks have come a long way since their inception, proving to be incredibly accurate when provided with enough data to train from. Additionally, neural networks are incredibly robust once trained, handling new inputs much better than a traditional system would. The system of processing large amounts of input data by considering all features of that data ensures accuracy, however it also drastically increases the processing power and storage space required to handle that data. With this massive demand for resources, classification tasks such can become prohibitive to run on lower-end machines when the tasks reach sufficient requirements.

The example of classification that this paper discusses was formulated upon the classification of test subjects by images of EEG signals, where each image consists of a large number of features. While the dataset provided by Yue Yao, Jo Plested and Tom Gedeon [1][2] is navigable by a machine with sufficient resources, over sufficient training iterations the slowdown becomes increasingly apparent. Each input of the input data set contains just under two hundred features, which assists classification greatly by giving a plethora of features to judge, however each training iteration takes longer to process because of it.

Reinforcement learning is another advanced technique in the field of neural networks, and methods that implement reinforcement learning are typically characterized by their sizable performance requirements and equally sizable potential to master tasks they are performing. The characteristic of value that reinforcement learning models utilize is key to how the decision-making policies are developed, with the properties of that value being determined by what the model should aim for. Reinforcement learning models applied to games often have this explicitly stated by the goals of the game itself and useful ways for players to measure them, such as in the ViZDoom competitions of 2016 and 2017 where the bots competed for the most kills on their peers.[7] Though a reinforcement learning model may seem to run counter to the overall goal of reducing computation resources required to successfully learn about this classification task, determining an optimal compression ratio using reinforcement learning is itself another useful function that could be used on data of varying resolution or in other contexts.

2 Task Selection

This paper will utilize an autoencoder neural network to compress the large inputs of the aforementioned dataset to a smaller intermediate form, and then reproduce that large form as the output with a reasonable quality of accuracy. This measure of accuracy will be that of the differences between the reproduced input and the original input, with the overall goal of minimizing the difference. This accuracy will be measured across multiple iterations of compression, where more and more features of the original input are progressively removed to further reduce output size. Another more-grounded measure of accuracy will also be taken by another neural network that has been trained in the classification task from which the dataset was taken. The accuracy to which that network can predict which class these newly-compressed images belong to makes for a more practical example of this technique at work.

If a suitable compression method can be found to reduce such large inputs to a neural network, it will increase the capacity for smaller devices to handle neural networks that can process this data, which would make neural networks easier to implement and their benefits more readily available in contexts that enforce limits on processing resources for devices. Making neural networks available to any type of computer smaller than a regular desktop computer (which has the capacity to handle bigger data inputs) has numerous useful applications, with significant research already being conducted in introducing neural network functionality to the medical field, as an example.[6]

Additionally, it would be beneficial for the system implemented to automatically learn about how compression affects the overall accuracy measured by the evaluating neural network. To do this, this paper will introduce a reinforcement learning approach to implementing compression. A reward signal will be introduced for choosing parameters where the accuracy loss between iterations of compression is minimal. Rewards will slowly be reduced as the compression ratio reduces the accuracy to below a suitable level, with the intended effect of allowing the compression network to determine how low the compression ratio can go before the data is compressed beyond usability. Doing so would enable the network to decide for itself an optimal compression ratio, which could then be useful in other contexts where data may have a different number of important variables to consider. This could then be used by more powerful computers to produce more consistent results even when working with data that varies in resolution or size.

3 Method

T.D. Gedeon and D. Harris' work on Progressive Image Compression provides a suitable method to balance an increase in performance of processing with a minimized loss of accuracy.[5] Using a neural network of three layers, the technique's first iteration begins with the input being passed through a hidden layer of less units than features of the input. In the hidden layer, the features of the input and weights of those features are collected and turned into vectors of the input space, being measured alongside the angle between them and other vectors. An attempt is then made to reproduce the output from the reduced form of the hidden layer and tested against the input to measure similarity.

If the measure of similarity is the same, or within acceptable limits, the network will prune the least significant unit(s) of the hidden layer and begins another iteration of training. These least significant units are those that create vectors with little magnitude, and groups of units with vectors that are either incredibly similar or equally opposing. [3][4] Such units when removed should have little effect on the capacity to reproduce the input from its compressed form, as their overall effect on the image is judged to be equally insignificant. [3][4]

Over sufficient iterations this technique will cause a decrease in the accuracy of reproducing the originals from the intermediate form, measured not only in similarity between output and input but also by the ability of another, separately trained neural network, to accurately predict the class each reproduced output should belong to.
A base classification task will be carried out by a neural network as specified by the experiments of Y. Yao et al., [1] with the recorded accuracy being used as a benchmark for further testing. After training the compression network, the inputs will be compressed and fed into the network performing the base classification task. The accuracy of the task performed when the inputs were compressed, compared with that of when the task has the uncompressed input, should prove the relative effects of compression and how it affects relative accuracy.
Though this does not give a fixed measure of overall accuracy, the measure in relative accuracy and comparison of the uncompressed and compressed inputs to the same task is the desired result. The judgement of which level of accuracy is acceptable or desired is dependent on the classification task itself, so this report assumes a wider range of allowed accuracy to report on the extended effects of compression.

With the desired goal of maintaining relative accuracy between classification tasks on uncompressed and compressed data being a measurable property, a reinforcement learning model will be constructed to take relative accuracy as a desired reward. Value will be assigned to minimizing accuracy loss between the two, and the policy will also account for overall loss of accuracy that has taken place during compression iterations, to determine when further compression would be too detrimental to consider. This property of 'how much compression is too much' will depend on the performance of the classification task used on the base dataset without any compression, however I believe a useful guiding point would be around a 30-50% drop in accuracy.

4 Results and Discussion

Unfortunately, I was unable to produce conclusive results to the overall task with my efforts. I was able to implement an autoencoder to compress and reconstruct the data in a similar vein to Y. Yao et al. [1], perform a classification task on the data before and after compression, and display the difference between the accuracy to the reader. However, the final stage of implementing the reinforcement learning method proved too difficult to correctly implement.

The dataset was reconstructed from the .mat file using modules from the pandas library for Python, as I am most familiar with using the data constructs in that library. The data was then split by using the train-test-split method from the sklearn python library, and preprocessed using a MinMaxScaler from the same library as I am again more familiar with using this library to perform these tasks. The preprocessing was used to normalize the values from the dataset, as there were numerous outliers that were exceedingly different from the majority of datapoints which would adversely affect all subsequent training and testing.

I began by implementing the base classification task, using a succession of three linear layers and applying both the pytorch standard ReLU function as an activation function and pytorch batch normalization to each layer prior to output. The first layer used 64 hidden units to significantly reduce the amount of features from the input, and the second layer used 16 hidden units to again significantly reduce features. The output layer produced one feature as an output and would be handled in a separate function, where the sigmoid activation function would be applied and the class would be predicted from that output.
The base classification task and its corresponding neural network proved to be quite accurate once correctly implemented, achieving an accuracy of around 85% when performed on the preprocessed dataset with sufficient training. In this task a training set of 70% of the preprocessed dataset was used to train, and around 30 epochs were required to achieve the most gains in accuracy. With this stage successfully implemented, I then moved on to creating the autoencoder for the task of compression.

Trained

	precision	recall	f1-score	support
0.0	0.92	0.64	0.75	1184
1.0	0.83	0.97	0.89	2134
accuracy			0.85	3318
macro avg	0.87	0.80	0.82	3318
weighted avg	0.86	0.85	0.84	3318

The autoencoder proved to be quite difficult to implement, largely due to the increased complexity I placed on it over development. Instead of producing a smaller encoder with only three layers, I instead opted to use a series of layers for both encoding and decoding. These linear layers would reduce the number of hidden units until reaching the target amount of features, decided by how much compression the reinforcement learning model would seek to apply. For the purposes of testing this autoencoder, I opted to reduce the features to 96 (half of the input features of the preprocessed data) as I believed this would show a significant change in accuracy.
Once compressed, the autoencoder then utilizes a mirrored series of linear layers to reconstruct the data from its compressed state, for the purposes of testing using the classification task. This would make the autoencoder more complicated to use and it took significantly longer than the classification task to train, however I believe this to be justified as in practice the trained model was able to quickly output encoded data from the preprocessed dataset once trained. The pytorch standard ReLU activation function was used across all layers except the final output, which utilized the pytorch TanH activation function instead.
The autoencoder was trained using a training set of 70% of the preprocessed dataset, across 10 epochs of training. Once trained, the autoencoder was tested using the entire dataset except the class column to produce the encoded data, which would then be used in the classification task.

In performing the classification task upon the preprocessed dataset and the same dataset once encoded, my classification network displayed the expected reduction in accuracy. However, I believe this to be the result of some discrepancy caused by encoder network, not by the expected increase in difficulty of classification due to the encoding reducing accuracy of the encoded data. When using the encoded dataset the classification task was unable to predict any of the samples as the class ‘non-alcoholic’.

Classification nn retrained on autoencoder data, testing

	precision	recall	f1-score	support
0.0	0.00	0.00	0.00	1184
1.0	0.64	1.00	0.78	2134
accuracy			0.64	3318
macro avg	0.32	0.50	0.39	3318
weighted avg	0.41	0.64	0.50	3318

It was in trying to undo this error in accuracy that I was unable to implement the reinforcement learning method I had hoped to achieve. As this incorrect behavior is only displayed by the classification task with the dataset post-encoding, I believe it must be due to a fault in the way I implemented the autoencoder and the resulting encoded data that it produces. I spent the remaining time I had on attempting to fix this behavior, which was ultimately unsuccessful.

As a result, the reinforcement learning method I had initially planned on using remains unimplemented, as the baseline measure from which it would draw the value of further compression is unsuitably inaccurate. If the accuracy between pre and post-encoding data could itself be measured, the remaining steps would be to implement another training loop and use a policy that considers the aforementioned accuracy measure to be the value the reinforcement learning model would seek. Further reductions in accuracy would be of a lower value, with the intended effect of allowing the reinforcement learning model to judge for itself when to cease further iterations.

4.1 Future Work

For future works, the main priority I would have would be in fixing the base task for the reinforcement learning method first. The reinforcement learning model I had hoped to implement is still an interesting prospect to me, and I would still aim to produce such a model with an accurate base of the correct classification task and a properly-functioning encoder.

References

[1] Yao, Y., Plested, J., Gedeon, T., "Deep feature learning and visualization for EEG recording using autoencoders, International Conference on Neural Information Processing", pp. 554--566. Springer (2018)

[2] Yao, Y., Plested, J., Gedeon, T., "Information-preserving feature filter for short-term EEG signals", Neurocomputing, vol. 408, pp. 91--99, Elsevier (2020)

[3] Gedeon, TD, Harris, D, "Network Reduction Techniques", Proc. Int. Conf. on Neural Networks Methodologies and Applications, AMSE, San Diego, vol. 2, pp. 25-34, 1991

[4] Gedeon., TD, Harris, D, "Network Reduction Techniques", Int. Conf. on Neural Networks Methodologies and Applications, AMSE, San Diego 1991.

[5] Gedeon, TD., Harris, D., "Progressive Image Compression", (1992), Accessed at http://users.cecs.anu.edu.au/~Tom.Gedeon/pdfs/Progressive%20Image%20Compression.pdf, on April 24th 2021

[6] Benjamens, S., Dhunnoo, P. & Meskó, B. "The state of artificial intelligence-based FDA-approved medical devices and algorithms: an online database." npj Digit. Med. 3, 118 (2020). Accessed at https://doi.org/10.1038/s41746-020-00324-0, on April 24th 2021

[7] Kempka, M., Wydmuch, M., Runc, G., Toczek, J., Jaskowski, W., "ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning", (2017), Accessed at https://arxiv.org/pdf/1605.02097.pdf, on 27th May 2021