International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878, Volume-8 Issue-4, November 2019

An Efficient Neural Network Model by Weight Roll Algorithm

Siddhartha Dhar Choudhury, Kunal Mehrotra, Christhu Raj, Rajeev Sukumaran

Abstract*: *Deploying deep learning models require extraction of the model weights from the training environment and saving them to files that can be shipped to production. Often complex models have large model file size and it is difficult to transport those models, this paper aims to reduce the size of the model file while transferring the trained weights to production environment. Weight rolls is an algorithm that rolls down (reduces) the trained model weights to a smaller size, in some cases even reduced by a proportion of one thousand (1,000). On the production environment this is again unrolled to regain the original weights that were learned by the neural network during its training phase. Weight rolls uses a compressed pictorial representation of the weights array along with a pix-to-weight neural network to transport the learned weights which can be used on the other end for the unrolling process. The pix-to-weight network maps the pixels of the compressed weight image to the original floating point values which in the unrolling phase is used to transform the pixels into corresponding floating point values of trained weights.

Keywords* : *Loss function, regression, neural network, training time.

I.INTRODUCTION

This paper deals with an algorithm that can help in reducing the size of the model file while it is being transferred from development environment to production. Often the model is trained on a different machine than the one that is used to serve the model to potential customers, so after training the models they need to be saved into a file generally called the weights file and this file needs to be transported into the production server. Deep learning models are able to solve complex problems and sometimes require model with millions of parameters, resulting in a huge model file size which can be tough to transfer from the development machine.

In some cases, like transfer learning where pre-trained architectures are used to build customized models on top of them the weights file can be very huge. These models involve humongous numbers of weights which often pose a problem while serving on the production environment.

Weight rolls uses a pictorial representation along with a small neural network called the pix-to-weight neural network to transform the weights into small sized files and later they can be unrolled in the production server to get the original set of weights that were learned by the neural net. The entire process involves two steps:- rolling the weights in the development machine and unrolling the weights back to their original floating point representation in the production server.

Weight rolls first converts the entire set of weights from various layers into a huge vector and this vector is mapped into a 8-bit image file. Since, the 8-bit image file contains pixel values ranging from 0-255 unlike the weights which are generally floating point numbers (both positive and negative), a small neural network called the pix-to-weight neural network is used to map the normalized pixel values to the floating point representation of the weights, this process is called rolling and is done in the development environment.

In the production environment after transferring the pix-to-weight neural network along with the 8-bit pictorial representation of the trained model, the unrolling process is done. During this process the weights of the trained model are transformed back to their original floating point representation using the pix-to-weight network. After the transformation process the unrolled weights array can be restored back to their original shapes as specified in the various layers of the trained neural network. This completes the process of transferring weights file from development to production environment in a small amount of time even if the original model file was very large in size.

The process of rolling and unrolling into the weight rolls incurs a small amount of loss, but it is too insignificant to be wary of and thus can be overlooked considering the proportion by which weight rollers reduce the weights file size. As will be evident from the experimentation section weight rollers are capable of reducing file size by even a thousand fold in some cases (especially in transfer learning models).

The details of the percentage change in weights file size can be found out in the experimentation section which also provides details about the various models the algorithm was tested on and their roll values (value by which the size of the original weight file and the rolled weight file vary).

Though weight rolls help in reducing file size, it should be noted that this is only for the transportation of the file from development environment to production server and does not aim to reduce the file size while making predictions as the weights get unrolled using the transformation function

Revised Manuscript Received on November 15, 2019
***** Correspondence Author

Siddhartha dhar Choudry, Computer Science and Engineering, SRM Institute of Science and Technology, Chennai, India. siddhathadhar_soumen@srmuniv.edu.in
Kunal Mehrota, Christhu Raj, Computer Science and Engineering, SRM Institute of Science and Technology, Chennai, India. mrchristhu-raj@gmail.com
Rajeev Sukumaran, Teaching Learning Centre, Indian Institute of Technology Madras, Chennai, India. rajeev.s@wmail.iitm,ac.in

provided by the pix-to-weight network after the file is uploaded to the production environment. The following sections introduce the concepts of rolling and unrolling done in the weight rolls algorithm in detail.

II. RELATED WORK

A. Deep Compression

In their present structure, Deep Neural Networks require huge memory to subsidize their gigantic over-parameterization. Numerous endeavors have been made to lessen the document size of Neural Networks, for the most part depending on strategies, for example, Weight Pruning or Quantization, or SVD deterioration of Weight Matrices.

Deep Compression [1] consists of three different stages:

1. Pruning
2. Quantization
3. Huffman Encoding

A1. Pruning

Pruning [2] portrays the way toward covering out specific weights in a Deep Neural Network. This requires actualizing a cover over the Neural Network layers with the end goal that they emphasize distinctively through the y = Wx + b task. Weight pruning isn't equivalent to just setting certain weights to 0. It has many nuances associated with it, for instance, if a weight is lies between an interim of [-0.5, 0.5], it will be conceal out.

A2. Quantization

Quantization is a procedure to diminish the quantity of bits expected to store each weight in the Neural Network through weight sharing. Weights in a Deep Neural Network are often represented by 32-bit floats, like, '2.70381'. In Quantization, a k-Means calculation is applied to scan for groups that depict the weights in the system. In the event that the weights are to be represented with 3 bits, this would result in $2^3 = 8$ centroids used to group the weights. Each weight is then mapped to it's separate centroid. For instance, '2.70381' -> '2'. The 8 centroids along these lines structure the 'Codebook' used to outline unique weights to the comparing 3-bit weights. This Codebook is then tweaked during training.

The Codebook is adjusted by means of a comparative component as classic backprop/SGD. The partial derivative of each weight is registered and they are totaled for each discrete 3-bit weight. For instance, a progression of the '2' 3-bit weight may have the relating fractional subordinate updates of '0.2', '0.1', '0.2', and '0.3'. These subsidiaries are amassed and '2' is optimized to become ‘2.2'.

A3. Huffman Encoding

Huffman Encoding is a mainstream procedure in compression to exploit skewed/biased distribution of values. For instance, if 20 weights map to '2', 10 weights map to '3', and 3 weights map to '8', it would bode well to encode 2 as '00', 3 as '10, and 8 as something like '1110'. Huffman encoding is utilized for this situation to diminish the measure of bits expected to represent the weights in the Quantized Codebook.

III.UNDERSTANDING WEIGHT ROLLS

The following are the different components of weight rolls, these will help in better understanding the concept :-

A. Rolling

This process is carried out in the development environment and is used to compress the weight file into three different parts:-

1. Single Dimensional Image
2. Neural Network architecture
3. Pix-to-weight Neural Network

The weights and biases across multiple layers of the neural network are collected together into a single array and converted into one-dimensional array. This array is then saved as an 8-bit image. Since the weights and biases are floating point numbers unlike the pixel intensities in an image, there is a need for a function that can map those pixel values to the floating point numbers so that when the parameters are required on the production server then they can be calculated from the pixel intensities of the 8-bit image that is sent to the server. This process of converting the parameters into a single dimensional image is called "rolling" of a model. And in the production server these values are converted to their previous floating point numbers using the "pix-to-weight" neural network [4]. The architecture of the model needs to be extracted from the development server as this will help in building the model and loading weights into them during inference in the serving environment.

Thus the final model to be shipped consists of the above mentioned three components and ranges from hundreds of kilobytes to a few megabytes (in case of large models. In contrast to the huge weights file for deep neural networks with several layers this will hardly take any time in transporting the model between development and production environment.

A1. Single Dimensional Image

This is the 8-bit representation of the parameters of a neural network which is created during the rolling process. This contains the actual trained weights of the network and is to be shipped to production for serving the deep learning model to customers. This consists of black and white pixels and in most cases takes a few kilobytes of space.

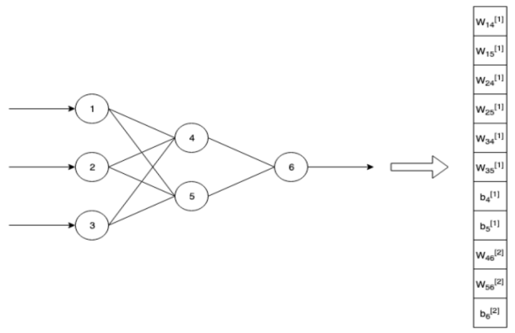

Figure 1. Neural Network Data for Single Dimension

Convert Neural Network to 1-D Image. Representation used: $W_{14}^{[1]}$ = Weight for the edge from 1 to 4 from a layer 1 node i.e. node 1.

A2. Neural Network Architecture

The neural network architecture is required during the unrolling phase done in the production server. This is required to build the model and also to know the specific details about number of nodes in a particular layer so that the one-dimensional image can be converted back to usable parameters. This is the exact configuration of the computational graph of the neural network and through this the data is passed to get the final inference.

A3. Pix-to-weight Network

As the one-dimensional image is in 8-bit representation so all the pixel values will range from 0-255, but the values of weights and biases are generally floating point numbers (positive or negative) and thus a function is required to convert back these pixel values into usable parameters i.e. to their floating point values, during the unrolling phase. Neural networks are also known as "universal function approximates" so using a small neural network these pixel values can be mapped to their corresponding floating point value.

B. Unrolling

This step is done in the production server. This involves the conversion of the pixel values from the shipped one-dimensional image to the floating point values using the pix-to-weight neural network. First the pixel values are converted into floating point values and then rearranged into multi-dimensional arrays according to the shapes in the neural network architecture exported from the development environment. Then the model is built and the weights and biases are loaded into this neural network, since the values are converted back to their original values and dimensions so this step is called "unrolling" of the neural network model.

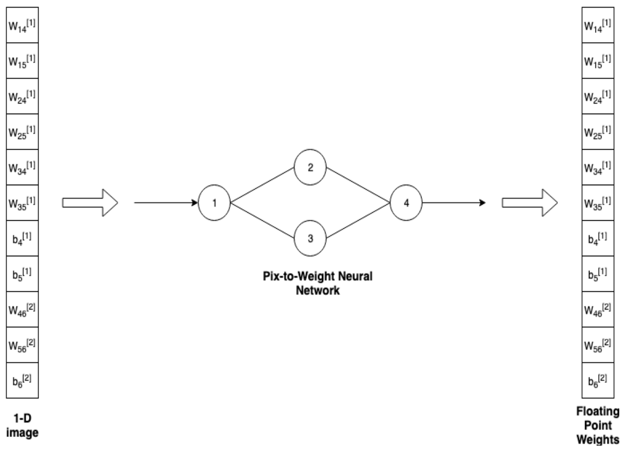

Figure 2. Unrolling Network

C. Roll Value

It is known that a neural network can never achieve 100% accuracy on new data on which it has not been trained, and the pix-to-weight is a small neural network so in converting the pixel values to floating point weights and biases incurs a slight loss. This is called the "roll value" of the rolled model. Since, the time required for transportation decreases significantly and the small pix-to-weight network has only a small loss it can be ignored.

The roll value depends on the size of the neural network model and hence the number of parameters as more number of parameters will give the pix-to-weight neural network a chance to learn better mappings. Roll value is thus inversely proportional to the number of parameters, more values will give a smaller inference loss.

$$rollvalue \propto \frac{1}{parameters}, rollvalue \propto \frac{1}{neuralnet}$$

IV.EXPERIMENTATION

The following experiments were conducted to verify the weight rolls algorithm:

A. Transfer Learning with Inception v3

Inception v3 is a pre-trained convolutional neural network on the Image Net database. As the network is trained on this huge database containing images belonging to thousand different classes it has knowledge of a wide array of representational details and using those base representations we can build a neural network to learn image classes that were not part of the image net database, this method is called transfer learning [3].

The Inception v3 model contains approximately 24 million parameters, the resulting weights file size after training with custom data reaches around 100 megabytes (depending on the custom layers added). By applying weight rolls this size gets substantially reduced and can be shipped much easily to the production environment.

In the experiments conducted, we created an image classification model that classifies between ten different species of birds. Two convolutional layers and one fully connected layer was used on top of Inception v3 for this purpose. The following are the details about the layers used in the network:

1. **Convolutional Layer 1:**
    1. Number of filters: 128
    2. Kernel size: 3
    3. Activation: Rectified Linear Unit (ReLU)
2. **Convolutional Layer 2:**
    1. Number of filters: 64
    2. Kernel size: 3
    3. Activation: Rectified Linear Unit (ReLU)
3. **Dense Layer 1:**
    1. Number of nodes: 10
    2. Activation: Softmax

The following are the results of the experiment:

1. **Size of original model file:** 100 MB
2. **Size of one dimensional image:** 8 KB
3. **Size of pix-to-weight neural network:** 7 KB
4. **Size of model architecture:** 147 KB
5. **Total size of rolled model:** 162 KB
6. **Amount of reduction in model file size** = (100 * 1024) / 162 = 632 times less
7. **Percentage decrease in file size** = (102400 -

162) / 102400 * 100 = 99.84%
8. **Roll Value** = 0.3

B. Transfer Learning with ResNet50

ResNet50 is a pre-trained convolutional neural network on the ImageNet database. It has fifty layers (including convolutional, fully connected, pooling among others) and approximately 25 million parameters. The weights file size for this network is about 105 megabytes (MB) in size (depending on the number of custom layers on top of ResNet50). By applying weight rolls technique this is reduced to kilobytes.

During experimentation we trained an image recognition model that detects the presence of apples and oranges in an image. A single convolutional layer was used followed by two fully connected layers. The following are the details about the layers used in the network:

1. **Convolutional Layer 1:**
    1. Number of filters: 256
    2. Kernel size: 3
    3. Activation: Rectified Linear Unit (ReLU)
2. **Convolutional Layer 2:**
    1. Number of filters: 128
    2. Kernel size: 3
    3. Activation: Rectified Linear Unit (ReLU)
3. **Dense Layer 1:**
    1. Number of nodes: 1
    2. Activation: Sigmoid

The following are the results of the experiment:

1. **Size of original model file:** 105 MB
2. **Size of one dimensional image:** 10 KB
3. **Size of pix-to-weight neural network:** 7 KB
4. **Size of model architecture:** 150 KB
5. **Total size of rolled model:** 167 KB
6. **Amount of reduction in model file size** = (105 * 1024) / 167 = 640 times less
7. **Percentage decrease in file size** = (107520 - 167) / 1075200 * 100 = 99.83%
8. **Roll Value** = 0.5

These experiments huge amount of difference weight rolls have on the size of the weights file of a trained neural network and helps in easy shipping of models to production.

V.CONCLUSION

From the experimentations we found that the weight rollers can reduced the size of the weight files by even a thousand fold in some cases. This small file size helps in transporting the model file to production environment easily where it is transformed from the image pixels to the original floating point weights using a small pix-to-weight neural network. These rolled models are useful when dealing with complex models (often pre-trained architectures used in transfer learning problems). The small size of the pix-to-weight network is much smaller as compared to the larger trained model, and together with the pictorial representation of weights can be transported to production environment.

ACKNOWLEDGMENT

We would like to thank Andrew Ng, Geoffrey Hinton, Sebastian Thrun for their deep learning courses in Coursera and Udacity from where we started our deep learning journey. We would like to acknowledge Shashank Pandey for stimulating our creativity and proof reading this paper to provide his valuable inputs. Rushil Gupta provided much-needed support with LATEX typesetting.

REFERENCES

1. Song Han, Huizi Mao, William J. Dally, Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding, 4th International Conference on Learning Representations, ICLR 2016.
2. Qing Yang, Wei Wen, Zuoguan Wang, Hai Li, Joint Pruning on Activations and Weights for Efficient Neural Networks, CoRR, June 2019.
3. Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, Chunfang Liu, A Survey on Deep Transfer Learning, CoRR, August 2018.
4. Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, Luc Van Gool, Practical Full Resolution Learned Lossless Image Compression, CoRR, November 2018.
5. Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, Efficient Processing of Deep Neural Networks: A Tutorial and Survey, CoRR, 2017.
6. Tim Salimans, Diederik P Kingma, Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks, NIPS, 2016.
7. Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Auerlio Ranzato, Nando de Freitas, Predicting Parameters in Deep Learning, NIPS, 2013.
8. Christian Herold, Yingbo Gao, Hermann Ney, Improving Neural Language Models with Weight Norm Initialization and Regularization, Proceedings of the Third Conference on Machine Translation: Research Papers, 2018.
9. Günter Klambauer, Thomas Unterthiner, Andreas Mayr, Sepp Hochreiter, Self-Normalizing Neural Networks, CoRR, 2017.
10. Kang-Ho Lee, JoonHyun Jeong, Sung-Ho Bae, An Inter-Layer Weight Prediction and Quantization for Deep Neural Networks based on a Smoothly Varying Weight Hypothesis, CoRR, 2019.
11. Yuchen Hou, Lawrence B Holder, Deep learning approach to link weight prediction, International Joint Conference on Neural Networks (IJCNN), 2017.
12. Yasunori Yamada, Tetsuro Morimura, Weight Features for Predicting Future Model Performance of Deep Neural Networks, International Joint Conference on Artificial Intelligence (IJCAI), 2016.