Using A Set Of Specialized Shallow Autoencoder Networks For Dynamic Incremental Supervised Classification

Rafi Qumsieh
rafiqumsieh@gmail.com

June 6, 2020

Abstract

State-of-the-art deep learning methods perform very well on classification tasks, but they do not show flexibility with modifying the output classes and with incremental learning. We propose a system with an architecture that allows for adding new classes dynamically and learning incrementally by using a specialized shallow autoencoder network for each prediction class. The system was trained on the MNIST handwritten digits dataset and produced accuracies comparable to a convolutional neural network in larger datasets and higher accuracies in few-shot learning settings.

1 Introduction

State-of-the-art deep neural networks excel at supervised classification tasks, but a major problem that arises from using these architectures is the difficulty of dynamically adding new output classes to the network. Traditionally, the output classes have to be known and pre-defined prior to training, or even data collection. This can cause issues if other classes need to be added to the model in the future as the network needs to be trained to incorporate the new class's data while remembering the previously-used data. This can be a very time and computationally consuming process. The network's architecture also assumes some form of a priori relationship between classes since the last layer in such architectures includes neurons for each of these classes, and it can be argued that the representations learned in latter layers encode inputs in a way to distinguish between those classes, but there not need to be a prior relationship between the prediction classes. Some of the methods that are used to address this problem are:

1. Including the new class's data in the training data, adjusting the net-

work's output layer, then re-training the model from scratch to incorporate the new class. As can be seen, this needs to be repeated every time a new class needs to be added. The process can be extremely time and energy consuming, impractical, and biologically implausible.

2. Another more practical solution is to use some form of transfer learning. The idea is to use the previously-trained network and trim the latter layers, then replace the last output layer with an output layer that contains neurons for the new class. After that, the new network is trained on the new class's data. While this might be more energy-efficient than the previously mentioned method, it is still prone to what is known as "catastrophic forgetting" : The effect of forgetting the previously learned data to adjust for the new class's data since the same set of weights are used to encode the information[1 5]. Ian Goodfellow et al. discussed the issue of the catastrophic forgetting and recommend using dropout to address it[2] .

3. This topic is still an active research area. For example, Kirkpatrick et al. developed a method called Elastic Weight Consolidation which involves constraining the learned parameters to remain close to the parameters that optimized the previously learned tasks by adding quadratic penalties in the loss function[3].

While the first two mentioned methods can perform well in some situations, their approaches lack the flexibility and efficiency needed for incremental learning. In this paper, an alternative architecture will be discussed that allows for both easily modifying the output classes and learning incrementally as the system receives new training data.

The system attempts to solve the problem of needing to re-train the model everytime a new class is added by using a modular architecture that is able to expand by creating a new, independent module for each incoming class. In other words, creating a neural network that is specialized in identifying that particular class only. One can loosely argue that such system already exists in nature: The neocortex. The neocortex is a thin sheet that it is about 6 layers deep with a very small thickness compared to its surface area[4]. When scientists rewired visual inputs to the auditory cortex in some trials, that part of the neocortex was able to process the visual information [3]. One possible implication of this is that the neocortex consists of similarly-structured general-purpose units. This approach can also be compared to using horizontal scaling versus vertical scaling to increase the computing power of systems. The approach discussed here generally requires more neurons than traditional deep neural networks, but it will also require significantly shallower architectures which makes the tradeoff computationally bearable.

2 The System

The system consists of N autoencoder neural networks corresponding to N prediction classes. Each network specializes in recognizing one particular class and is not concerned with the other classes. During training, each network is trained on its corresponding class's dataset in a supervised manner. During prediction, an incoming input signal gets passed to each network independently, and the network with the lowest reconstruction loss (expressed as the L1 or L2 norm difference between its output and the input) is chosen as the predicted class. The figure below depicts the architecture of the system:

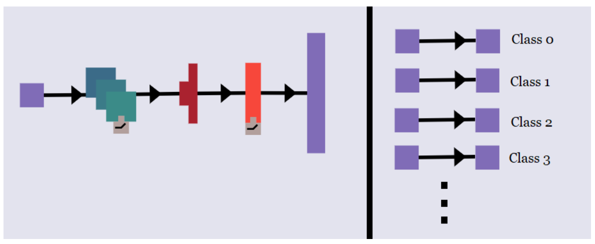

Figure 1: On the left, a traditional convolutional neural network. On the right, a set of specialized shallow autoencoder networks, an autoencoder network for each output class.

Each specialized network is a shallow autoencoder which consists of 1 input layer and 1 output layer with no hidden layers, and uses SELU as its activation function, and the RMSProp for its optimizer. The initial weights and biases of the whole network are set to zero. The loss function can be a mean-squared error function or a mean absolute error function. SELU activation functions are defined as the following:

$$SELU(x) = \lambda \left\{ \begin{array}{ll} x, & \text{for } x>0 \\ \alpha e^x - \alpha, & \text{for } x \leq 0 \end{array} \right\}$$

Where α and λ are non-learnable constants that are not subject to hyperparameter tuning.

When a new class needs to be added to the system, a new autoencoder neural network is created specifically for that new class and is trained on the class's training data. This makes it possible to add as many classes as needed without needing to modify the previously-learned connections. Another benefit of using this architecture is the ability to incrementally learn individual data points or batches of data points to any class at any order and at any time in an on-line setting.

Below is a pseudocode of the algorithm:

Training

```
Initialize an empty array of networks
For each class:
Append a new instance of a network to networks:
 For each (x, y) in training_data for class:
  Train the new network on (x, y) for 1 epoch
```

Testing

```
Initialize Accuracy to zero
Initialize Counter to zero
For each (x, y) in testing_data :
 Initialize Distances as empty array
 For each network in networks:
  Evaluate input x using network as output o
  Calculate distance using L1 or L2 norm between input x and output o
  Append to distance to distances array
 Take softmax of distances
 Choose arg. of min. distance as predicted class
 If prediction equals y:
  Increment Counter
Calculate Accuracy
```

3 Experiment

3.1 Testing the addition of new classes

In this part of the experiment, the first 6 digit classes of the MNIST digits dataset were chosen. 6 independent but identical networks were created for each digit class and were trained on their corresponding digit's training data. After that, the system was tested using the test data and accuracies were recorded. Then, the 7th digit was picked, and a new network was created to handle that new class. After training the new network on the new class's data, a test was run for the whole system and the accuracy for all the classes was recorded.

To test the ability of the system to learn incrementally, when we introduced the 7th digit, we initially trained the system on half of the new digit's training data. Then, we ran the testing data through the system to see the effect of adding only half of the digit's data. The accuracies were recorded.

3.2 Comparing results to traditional methods

In this part of the experiment, we wanted to see how this architecture's performance compares to traditional state-of-the-art deep neural networks. We ran the system against a deep convolutional neural network with optimized hyperparameters for both networks while varying the number of training images per digit/class.

4 Results

4.1 Testing the addition of new classes

Below is the table that shows the effect of adding new classes on accuraries:

IPD	Prior	Half	Full Half	Full	Post
10	0.816	0.750	0.808	0.800	0.807
20	0.840	0.650	0.850	0.800	0.836
30	0.900	0.600	0.911	0.950	0.900

Table 1: First column is the number of training images per digit/class. Second: The accuracy of the system against test data prior to introducing the new class. Third: The accuracy of the system after training on half of the new classes data and testing on the new class's own data only. Fourth: The accuracy of the system after training on half of the training data for the new class and testing on all test data for all classes. Fifth: The accuracy of the system after training on all of the training data for the new class and testing on the new class's own testing data. Sixth: The accuracy of the system against all test data after learning the new class.

4.2 Comparing results to traditional methods

Below is a graph that shows the accuracy versus the number of images per digit for the specialized networks architecture and a convolutional neural network:

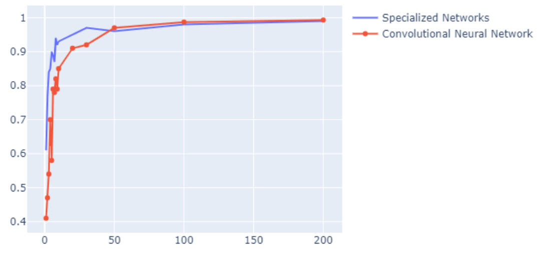

Figure 2: The accuracy versus the number of training images per digit for the specialized networks and a convolutional neural network.

5 Discussion

In this paper, we attempt to solve the problem of the inflexibility of traditional deep learning networks to dynamically and incrementally learn by using an alternative modular architecture that allows for adding a new autoencoder network module for each new incoming class. This approach allows the networks to maintain their previously-learned parameters while allowing for new information to be learned separately.

Initially, for each specialized network, we tried a feedforward binary classifier configuration with a binary crossentropy loss function. That configuration did not yield good accuracies. The suspected reason behind the low accuracies is that we were trying to encode all the possible information about the class in a single logit which makes it virtually impossible for each network to distinguish between classes. What the network might end up concluding from that architecture is if there is any input at all, the output neuron should fire.

For that reason, an autoencoder seemed suitable as it encodes as much information as it can about the input without needing to present other classes for comparison. After several trials, the best configuration that was found is a shallow autoencoder with no hidden layers that maps the input of size 28×28 to itself. The weights and biases were initalized to zero to give the network no initial bias towards a certain configuration. The optimal activation function at the output was found to be the SELU activation function. Of course, to implement a dense set of connections in the autoencoder using Tensorflow, we had to flatten the input first.

The results seem to indicate that the system works as intended, at least in the case of the MNIST digits dataset. It is important to recognize that the MNIST dataset is a fairly easy dataset to learn. The system needs to be tested against datasets with more complex distributions. This paper is intended as a pilot to spur interest in further investigation of this type of architecture for such tasks. It is worth noting that perhaps using some form of encoding as input to the system might help it perform well for complex data distributions.

A major problem that this architecture can face is that each incoming signal needs to pass through each autoencoder to determine the predicted class. This can be a very costly process. The problem exists in the absence of parallelization of computing. One possible solution is to have an indexing mechanism that precedes the system and that directs the incoming signals to the appropriate subset of networks for processing.

This architecture can significantly require more neurons than traditional neural networks. One possible solution to overcome this is by performing some sort of pooling on the input prior to processing it through the system or by including a relatively smaller hidden layer for each specialized network to reduce the number of connections needed.

Future research will involve investigating the efficiency of such architectures along with investigating the potenial application of this type of architectures in self-supervised learning by automatically learning new classes.

References

[1] Z. Chen and B. Liu. Lifelong machine learning. https://www.cs.uic.edu/~liub/lifelong-learning/continual-learning.pdf.

[2] Ian J. Goodfellow et al. An empirical investigation of catastrophic forgetting in gradient-based neural networks. https://arxiv.org/abs/1312.6211.

[3] James KirkPatrick et al. Overcoming catastrophic forgetting in neural networks. https://arxiv.org/pdf/1612.00796.pdf.

[4] Ronan L et al. Differential tangential expansion as a mechanism for cortical gyrification. https://pubmed.ncbi.nlm.nih.gov/23542881/.

[5] Robert M French. Catastrophic forgetting in connectionist networks. https://www.sciencedirect.com/science/article/abs/pii/S1364661399012942.

[6] MIT. Mit researcher finds that part of brain used for hearing can learn to 'see'. http://news.mit.edu/2000/brain.