Transfer Learning Across Algorithms, Across Domains: A Preliminary Analysis

Sharice Mayer, Cyrus Rice, Lane Barton, Shehab Ibrahim, David Djernaes, Marina Neumann

Table of Contents:

1. INTRODUCTION

The inspiration for this project was to take an exploratory step toward understanding transfer learning techniques for deep learning. The objective was to examine if we could train a workable model with a dataset from one domain and apply a separate dataset from a different domain and determine if a meaningful amount of improved performance could be detected cross-domain through our tests. Concomitant with our goal of testing the outcome of datasets from different domains, our project also involved developing a second model to test the outcomes when applied to a different learning algorithm.

The project was executed using two prominent algorithmic learning models; a multi-layer neural network (NN) and a hard-clustered K-means analysis. The datasets used in our evaluations were: MNIST(handwritten digits), NIST Special Database 19(handwritten characters), and Optdigits(binary optical representation of handwritten digits). The MNIST database is comprised of handwritten digit samples representing the values 0-9[1]. The dataset represents each image as a 28x28 pixel image, which has been converted to a 1 dimensional array of 784 integers, each representing a pixel within the image, ranging from 0-255 depending on the RGB value of the particular pixel. The NIST Handprinted Forms and Characters Database[2] provided data consisting of handwritten samples of the 26 capital letters of the alphabet. Although the structural representation of a single example is identical between the MNIST and NIST datasets, each dataset is composed of digits and characters respectively, and therefore differ both in domain and number of classes. The third data set - Optdigits - renders optical images of each digit 0-9 using a 32x32 grid of 1's and 0's to represent the data, where 0's represent blank space and 1's represent part of the handwritten digit.[3]

[1] MNIST Database of handwritten digits: http://yann.lecun.com/exdb/mnist/
[2] NIST Database of handwritten letters: https://www.kaggle.com/sachinpatel21/az-handwritten-alphabets-in-csv-format/home
[3] OptDigits Database: https://www.openml.org/d/28

2. THE PROCESS

2.1 Multi-Layered Neural Network Algorithm Design

For our implementation of a multi-layer neural network, we leveraged an existing implementation in the Scikit-Learn Python library (specifically the MLPClassifier[4] object) that would efficiently and consistently run a NN. Using this code, three different NN models were generated in the following forms: a 784>196>X and two separate 784>196>X>Y. Note, the format N_1>N_2>N_3…>N_K has N_1 for the input layer, N_2 though N_{K-1} are hidden layers, and N_K is the output layer. Additionally, X represents the initial trained model outputs and Y represents the post-transfer outputs. For example, let X = the 10 MNIST digit classes and Y = 26 NIST letter classes. The final model has the form of 784>196>10>26.

To do transfer learning, the 784>196>X NN was trained and then the weights after convergence were passed one of the 784>196>X>Y NNs. The other similarly sized NN was given random weights as if starting from scratch, allowing us to do a comparison between transfer and baseline implementations with exactly the same number of nodes and weights. Both of these MNNs were trained on the same data and the overall time for the model's weights to converge was recorded for comparison. For our experiment we did 20 total trials, 10 trials transferring weights trained on letters to a NN to be trained on digits and 10 trials in converse.

2.2 K-means Clustering Algorithm and Data Preparation

Each dataset was prepared in a similar way for processing with our K-means models. We wanted to be able to project the results into a readable graph output, necessitating dimensional reduction. We used the Python sklearn module's PCA[5] method to reduce the data from high-dimension to two-dimensions before training on our model. We were aware this would

[4] MLPClassifier reference: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
[5] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

likely influence our loss function and accuracy, but for the purpose of this project we wanted the data to match dimensionally for the transfer learning element of testing as well to have visual output for reference. For each training set, we maintained a constant number of times to training iterations on the model with a new set of initial means. We did this 10 times for every run of k-means, and picked the results with the lowest sum-squared error of the 10 options. We trained each data set alone once. We also used transfer learning on each data set, meaning we trained with k-means on a previous data set, and, using the final means produced from this training, initialized the means for this data set. Then we would train this data set with those initial means.

3. Multi-layered Neural Network Algorithm: Expected vs. Observed

For the multi-layer neural network, we tested the two different directions of information transfer - training a NN on letters, then passing that network to a NN to train on digits, and vice versa. In the case where letter training was transferred to digits training, we saw some improvement against a baseline of training an identically structured NN with random weights on the digits datasets. As can be seen in the following table, 8 out of 10 trials showed a reduction in time for training to converge ranging from a 1% to 50% reduction from the baseline. This was an average of ~20 second improvement or an 8% reduction from the baseline. However, we noted that this does not factor in the time to train a NN on letters prior to transferring that information.

Training On Digits With and Without Transferring Training on Letters												
		Trials										
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Avg**
W/ trans-fer	**Time (s)**	187.1	249.2	206.2	160.2	184.5	196.4	231.8	250.1	192.3	290.5	214.8
	Iters	57	74	74	59	64	66	80	85	72	72	70.3
W/o trans-fer	**Time (s)**	221.6	261.3	246.8	324.6	228.6	198.2	238.8	162.4	180.7	280.8	234.4
	Iters	59	71	84	101	79	68	78	63	56	65	72.4

During the inverse case training a NN on digits data, then transferring information to a NN training on the letters dataset, we found much more inconsistent results. As can be seen in the table, 6 of 10 trials saw improved times to converge from training with transfer learning as opposed to the baseline and an average improvement of about 30 seconds (roughly 6.7%). However, a closer look at the values showed wild fluctuations in outcomes with differences between baseline and transfer-learning NN ranging from less than 10% to 500% of the lower value. These results lead us to conclude that our setup provides inconsistent results.

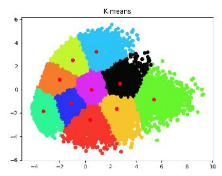
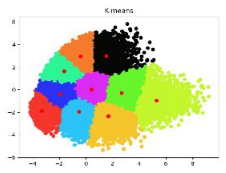
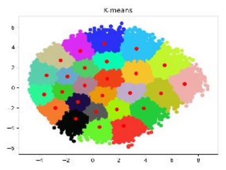
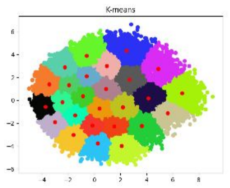
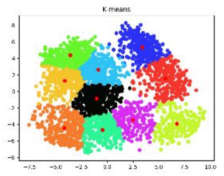
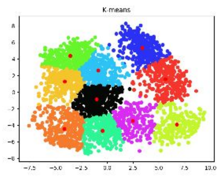
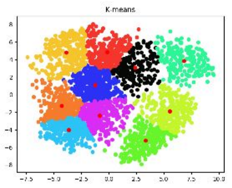
Training On Letters With and Without Transferring Training on Digits											
System	**Trials (values are time for training to converge, in seconds))**										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Avg**
W/ transfer	694.6	944.4	179.1	593.7	144.8	693	165.4	318.8	161.7	142.2	403.8
W/o transfer	155.7	650.2	1108.7	154.5	171.2	580	178.4	594.8	163.4	571.6	432.9

There are two possible practical explanations for our results. From a machine learning standpoint, some of the variability may simply be due to whether the randomized baseline was started with a better set of weights. Beyond that, the fact that the second scenario required a final hidden layer of 10 nodes to 26 output nodes meant that a single error in the value of the hidden layer affected 26 different outputs; meaning that a “bad” weight update had a bigger impact on the need to overcorrect, while a “good” adjustment satisfied a lot more situations.

From a practical standpoint, it makes sense that it’s easier for 26 letters to have applicable information to just 10 digits than vice versa. For example, if you are given something you think is a 1, should that correlate to an l, j, i, t, or maybe r? Conversely, with an ‘l’ you may choose between a 1 or 7, but with less variability to account for you can be more confident in your guess. Thus, our interpretation is that transfer learning for MNN applies decently from letters to digits, but more uncertainty from digits to letters.

4. K-means Clustering Algorithm: Expected vs. Observed

Results

Data set one	Data set two	Time elapsed	Sum squared error	Plot
mnist	--	1144	29466	
a - z	mnist	1311	29076	
a - z	--	1922	16440	
mnist	a - z	2197	16491	
optdigits	--	38	6950	
mnist	optdigits	36	6990	
a - z	optdigits	42	7043	

First we trained on mnist alone (row 1). From the graph we saw that k-means worked very well on clustering for the mnist data set. Next we trained on mnist using the final centroids from training on a-z data first (row 2). Since there are 26 letters, therefore 26 clusters, we used both the 10 final centroids plus 16 randomly selected to use as initial means for mnist. The fact that the two domains of letters and digits didn't seem correlated led us to predict that this run of k-means on mnist wouldn't have any improvement, and may even be worse due to the fact that the initial means may take longer to settle and have larger error, as they started in areas that aren't data points in mnist. Our results showed the algorithm took almost three extra minutes, but brought the error down by ~400. Due to a lack of trials and experience, we don't know whether this can be replicated, or if this is significant improvement vs a minor difference. The results still clustered well, with few minor differences in cluster shapes when training mnist solo.

Next, looking at the results of k-means on the a-z data set, training solo (row 3) clustered very well. Next we trained on mnist and used the 10 final centroids to initialize 10 random centroids when training the a-z data set (row 4). The remaining 16 means were initialized normally, to random points in the a-z data set. Similar to transfer learning on mnist, we had low confidence that the results here would be better than the a-z solo results. In row 4 you can see that the error went up a minor amount, and time went up by ~4 minutes. Like before, we would have to do multiple trials to see if this can be replicated.

Finally looking at the results of training on the optdigits data set, we only had ~3000 data samples in this set, causing sparsity in the plots. Despite this, k-means clustered the data very well. Training on mnist first (row 6), we initialized 10 centroids for training on optdigits with the 10 final centroids from training on mnist. We had higher confidence that results would be better here than when training optdigits solo because mnist and optdigits both have 10 clusters, and the data seemed more correlated, as they are both pictures of digits. We were wrong, as the results were very similar to the results for training on optdigits solo. This was surprising, as this was the transfer learning combination that seemed most correlated. As you can see, the plot here and with optdigits trained solo are nearly identical, perhaps because the original centroids chosen were from the final means of mnist, and thus didn't need to move much. Finally we trained on the a-z data set (row 7) and picked 10 random centroids from that result to initialize when training on optdigits. Our error and time were slightly worse than both above k-means runs, which we expected due to possible lack of correlation between a-z and optdigits data.

Overall we did not see improvement when training using transfer learning on k-means. However, there were many things we could have done differently. We could have conducted more trials, manipulated more variables, used different data sets, and tried different methods. Our knowledge of the topic and total time spent on it are not sufficient enough to conclude that transfer learning isn't effective for k-means unsupervised learning.

5. Conclusion

Transfer learning is the ability for a learned network on a certain dataset to transfer its "knowledge" to a different dataset entirely. This offers great potential in the advancement of machine learning, especially with natural language processing, and can potentially produce some of the best performance and most effective results with less overall training time. We based our NN character classification in work completed by Cohen(et al)[6] in the hopes to expand understanding within the application of transfer learning research pertaining to both supervised and unsupervised learning using similar and dissimilar domain data. In our experiments, we compared two approaches of supervised and unsupervised learning to transfer learning on the MNIST and NIST datasets with widely varying results. When experimenting with

[6] EMNIST Research - Cohen et al: https://arxiv.org/abs/1702.05373v1

supervised learning and building our neural network of letters to digits, we found that we had between 1% to 50% reduction in training time, which averaged about an 8% reduction from the baseline. Yet, when we used the trained network to transfer learning from digits to letters we found an average 6.7% improvement but significant variability in our results, which made it difficult to claim definitive improvement as it pertains to transfer learning.

Unfortunately, when training for our unsupervised learning approach, we weren't able to identify any significant improvements in our clustering of data in either transfer direction. While we assume with more time, knowledge, and varying experiments there could be notable optimization in transfer learning, we were not able to conclusively prove this in our k-means experiments, as can be seen in our nearly identical clustering graphs and increased error rate results. From our experiments between unsupervised and supervised learning, we can conclude that supervised learning worked better with transfer learning. While our project was very exploratory in nature, we found that transfer learning has great capacity yet deserves a more thorough understanding from us programmers to be applied to datasets and tuned for best optimization.

"The only true wisdom is in knowing you know nothing"

~Socrates