Transfer Learning with Network Traffic Data

Byron Barkhuizen
Télécom SudParis
for Submission
at
https://github.com/byronbark/IOTProject

ABSTRACT

UPDATED—28 August 2020. Transfer learning is a developing research field in artificial intelligence to assist the development of new models, typically in the case of limited data or label availabilities. The concept describes learning between a source and target domain or task in the same way that a human learns. It relies on the idea that there exists similarities or knowledge that can be transferred to perform a task in a target domain at higher level of accuracy. There are various ways of performing transfer learning and they are dependent on the type of data available and the differences in the source and target domain or tasks where learning is desired.

Keywords

Transfer Learning, Representation Learning, Domain Adaptation, Python, Network Traffic.

INTRODUCTION

Transfer learning is the establishing of a relationship between a source domain and a target domain, along with source tasks and target tasks. DARPA (Defense Advanced Research Projects Agency) defines the mission of transfer learning as 'the ability of a system to recognize and apply knowledge and skills learned in a previous task to novel tasks. The existence or assumption that there is some similarity between the source and target is the theoretical basis for the application of transfer learning (1). The features within a source or target domain have some probability distribution. When these features distributions are the same then the new dataset can essentially just be classified in the same way that the original dataset was. If the source and target dataset do not conform to the same probability distribution, then this cannot be done accurately. However, this does not mean that a model needs to be completely re-trained and re-classified on the new dataset, there theoretically can exist some knowledge that can be transferred between the two datasets.

The final goal for a transfer learning application is to reduce the need for large quantities of data collection to be necessary, to boost the speed at which a new model in a target domain can be trained, and the accuracy achieved through these models in novel environments.

STATE OF THE ART

Currently the simplest application of transfer learning involves re-using all or some parts of a pre-existing model. These models can be trained on almost anything, however a transfer learning application with this approach will perform better if there exist some similarities between the source and target domain. In this case the source domain would be the dataset that the initial model was trained on. A popular pre-trained model that is used is Google's ResNet which is trained on an enormous number of images for the task of classification. This pre-trained convolutional neural network exists of many layers, and the initial layers closest to the input learn some low-level features such as edges or gradients. These features are application agnostic, they exist on all images that we may use. For this reason, it is possible to 'cut' this model so we keep only this distribution of features, while getting rid of higher-level features that may not exist in the target domain. These features become more specific to the classes that the initial model was trained on and may be entirely unnecessary. This application of transfer learning is very easy to implement, however often it will not achieve a very high level of accuracy until it is retrained on the new classification task with some newly labeled data in the target domain.

EXPERIMENTS

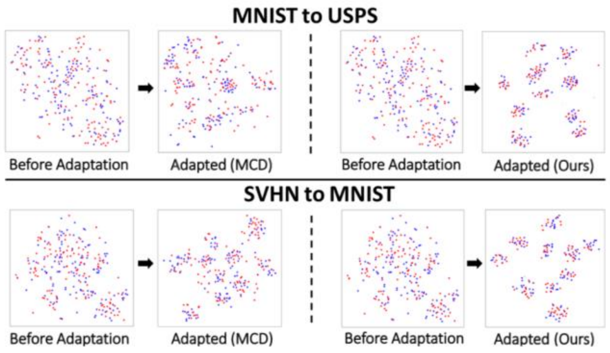
Data Pre-Processing

The data that is available in our experiments are PCAP (Packet Capture) files collected over various time spans. The PCAP is separated with the SplitCap tool, as well as the CICFlowMeter tool. We are left with raw PCAP files that are separated by flows and MAC address. We can use some scripts and Python programs to process these PCAP files and create binary files along with CSV files to represent the activity of individual devices. The data is normalized after they are split into test sets to not influence each other. At the end of the preprocessing we are left with 28x28 pixel images, and CSV files that represent 84 calculated time series features.

Domain Adaptation

Domain adaptation refers to bridging the gap between the differences in the source and target domains. Essentially, we want to minimize the differences in their representations. When they can be similarly represented (their feature distribution is the same) then learning has occurred. The new representation can then be used to train an initial model where we have rich data (source domain), and this model can be used on the target domain to achieve accurate results. More specifically, one method of performing domain adaptation is discriminative feature alignment.

An application of this is explored in the GitHub repository. It uses 3 different image sets, the popular MNIST image dataset along with SVHN and USPS. In the following images we can see a reduced dimensionality representation of what domain adaptation attempts to achieve. The individual points represent different images in latent space. Once a new representation is learned (after adaptation) we are again able to represent them in the latent space and there are clear clusters that are formed. In the case of the image dataset they are clusters that will each represent a different digit. The grouping before the adaptation does not exist and therefore any attempt to classify in the target domain would achieve extremely poor results.

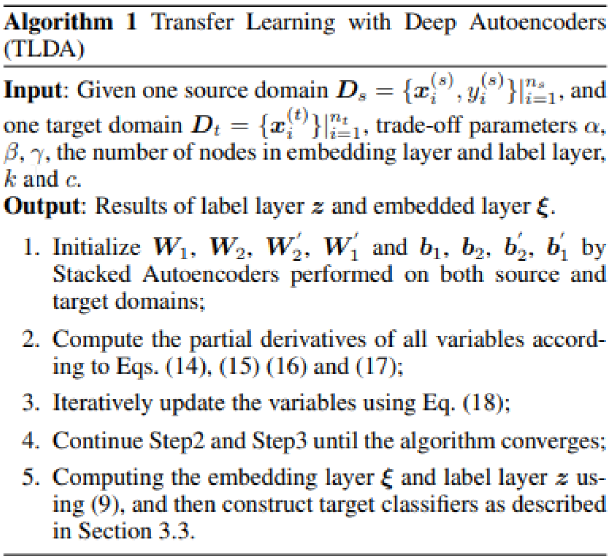


The application of this to our dataset as desired would be the following.

- Represent both images and CSV files as a NumPy array (pre-processing on CSV data) so that their initial representations are the same
- Learn best feature representation and modify both source and target data accordingly (such that on our latent space representation we will have one clear class).
- Train source model, test on target data on the new representation
- Testing can be done with any one class classifier

Autoencoder

TLDA (Deep Autoencoder)



Algorithm 1 Transfer Learning with Deep Autoencoders (TLDA)

Input: Given one source domain $\boldsymbol{D}_s = \{\boldsymbol{x}_i^{(s)}, y_i^{(s)}\}|_{i=1}^{n_s}$, and one target domain $\boldsymbol{D}_t = \{\boldsymbol{x}_i^{(t)}\}|_{i=1}^{n_t}$, trade-off parameters α, β, γ, the number of nodes in embedding layer and label layer, k and c.

Output: Results of label layer $\boldsymbol{z}$ and embedded layer $\boldsymbol{\xi}$.

1. Initialize $\boldsymbol{W}_1$, $\boldsymbol{W}_2$, $\boldsymbol{W}_2'$, $\boldsymbol{W}_1'$ and $\boldsymbol{b}_1$, $\boldsymbol{b}_2$, $\boldsymbol{b}_2'$, $\boldsymbol{b}_1'$ by Stacked Autoencoders performed on both source and target domains;
2. Compute the partial derivatives of all variables according to Eqs. (14), (15) (16) and (17);
3. Iteratively update the variables using Eq. (18);
4. Continue Step2 and Step3 until the algorithm converges;
5. Computing the embedding layer $\boldsymbol{\xi}$ and label layer $\boldsymbol{z}$ using (9), and then construct target classifiers as described in Section 3.3.

REFERENCES

**To be added on completion, all information presented within are from the collection of papers hosted on the GitHub repository