Engineer's Thesis

Wojciech Pratkowiecki

December 2018

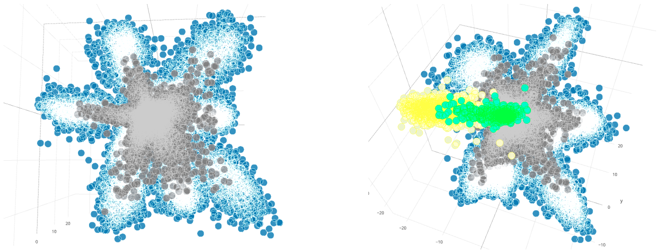
1 Visualization

To get some intuitions about how Gradient Reversal Layer and Conceptors work I tried to visualize the output of feature extractor. To do so I added an extra layer G_3 that takes from feature extractor G_f the output vector $f \in R^D$ and maps it to $f' \in R^3$, so the input vector x is mapped into 3-dimensional space as $G_3(G_f(x))$. Now we can treat it as a 3-dimensional point.

1.1 MNIST and MNIST-M

Training model with such a small layer inside is much more difficult task, as many information may be lost during mapping do R^3. Therefore at the beginning I tried to get a simple MNIST classifier with no Gradient Reversal Layer. Fortunately the model achieved 98% accuracy on the test set, but it performed poorly on MNIST-M - it managed to recognize just 31% of digits, while without 3D mapping model's accuracy is 61%. Now we can check how the input space X looks after mapping by feature extractor. Figure 1(a) shows the result of mapping for all samples from MNIST test set (blue) and MNIST-M test set (gray). Each point represents coordinates of vector obtained by passing it through feature extractor with 3D mapping, so the points cloud can be described as $F_i = \{G_3(G_f(x))|x \in X_{test_i}\}$, where $i \in \{\text{MNIST, MNIST-M}\}$. The shape of F_i resembles a multi-armed star. Moreover it seems like F_{MNIST} is just a stretched version of $F_{MNIST-M}$. We can expect that each arm contains mapped pictures of same digit. As we can see in Figure 1(b) zeros from both source and target distributions lie in the same direction and are represented by corresponding arm of stars.

Intuitively the domain adaptation may be seen as a task of finding a mapping for both source and target domain that yields the output which we can classify well, but we should be unable to tell the domain that the input sample comes from. In ours case we can assume that if we managed to extend our mapping with a mechanism that "stretches" the output for samples from target domain and leaves those from source distribution unchanged, we could get a classifier for both domains.



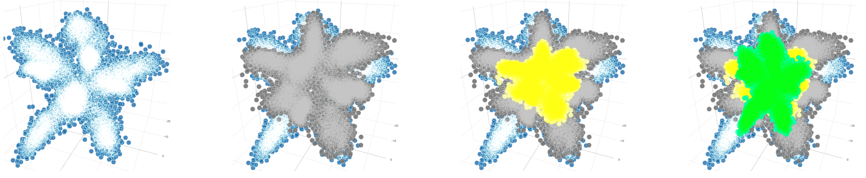
(a) MNIST and MNIST-M (b) Zeros of datasets

Figure 1: The visualization of samples transformed by feature extractor with output size equals 3. (a) presents two stars formed by samples from source domain (MNIST/blue) and a target domain (MNIST-M/gray) (b) shows how pictures of zeros are mapped - yellow points are zeros from MNIST and green ones from MNIST-M

1.1.1 Conceptors

Our first approach could be using of conceptor computed with samples from F_{MNIST}. As we assume, the conceptor defines an ellipsoid that approximates points cloud from certain distribution. Therefore multiplying some set of points by the concpetor matrix C_{MNIST} would project the set into space of F_{MNIST}. Unfortunately in 3-dimensional case, the C_{MNIST} is almost an identity matrix I_3. We should not be surprised, as our data points cloud is a mulit-armed star and limiting it with an ellipsoid requires it to be a sphere, we cannot approximate those points with any kind of flatten ellipsoid. Moreover, the singular value decomposition (SVD) of conceptor matrix C results in $C = U\Sigma U^T$, where U and U^T can be interpreted as rotation matrices and Σ is a diagonal matrix with singular values of C on diagonal. As all singular values of conceptor matrix are real numbers from $[0, 1]$, the conceptor matrix multiplying is in fact rotation, shortening (as non-zero values of Σ are ≤ 1) in certain directions and then re-rotating all the points (see Figure 2). Therefore we cannot transform $F_{MNIST-M}$ to resemble F_{MNIST} with conceptor matrix C_{MNIST}, as the multiamred star formed by points cloud would get even smaller, so it would increase the difference between source and target domains.

With this kind of knowledge one thing we can do is transforming points from F_{MNIST} with conceptor matrix $C_{MNIST-M}$ instead of transforming the $F_{MNIST-M}$ with C_{MNIST}. As mentioned before, the concpetor matrix is making points cloud tighter, so we could try to take the bigger points cloud (the one corresponding to MNIST) and fit it to the size and shape of the smaller one (see Figure 3). As we already have a well-trained model that performs well in MNIST recognition, we can extend it with a layer with no learnable parameters, that gets the input and multiplies it by conceptor matrix $C_{MNIST-M}$. Formally our model so far is a sequence of a feature extractor G_f (with parameters Θ_f), 3D mapping and its parameters Θ_3 and label classifier G_y with parameters Θ_y, so the result for input vector x is $G_y(G_3(G_f(x; \Theta_f); \Theta_3); \Theta_y)$. Now we want to



(a) F_M (b) $U^T \cdot F_M$ (c) $\Sigma \cdot U^T \cdot F_M$ (d) $U \cdot \Sigma \cdot U^T \cdot F_M$

Figure 2: The visualization of transformation points cloud by its conceptor based on SVD of concpetor matrix C. The representation F_{MNIST} of MNIST test set (blue points) has firstly been rotated (gray), then scaled (yellow) and finally re-rotated (green)

freeze the trained feature extractor and 3D mapping (let's call this part of our architecture as G_3) and multiply its result by $C_{MNIST-M}$, what gives us the new representation ($G_3^C = C_{MNIST-M} \cdot G_3$) of the input.

The mapping of points from target domain with G_3^C compared to transforming source domian with G_3 is illustrated in the second picture of Figure 3. If the model constructed with non learnable G_3^C and re-learned G_y learns to classify the digits from source domain we would expect the model containing G_3 and G_y perform well on target domain, as the G_3 transforms MNIST-M to grey points cloud from mentioned picture, so G_y should hardly see the difference between domains. After just few epochs the label classifier adjusted to new representation and managed to reach 98% accuracy on MNIST test set. Now we "unpack" the G_3^C to G_3 and check its performance on MNIST-M. The model got 36% of accuracy, so it improved by 16% from previous 31%. We can see the improvement, but we could expect something better, as we forced domains to get really similar.

The last picture of Figure 3 presents three points clouds - the blue one is the result of transforming source domain test set with G_3C. The yellow is target domain test set after transformation by G_3 (same as grey points from the second picture). The green points cloud is a mapping target domain by G_3^C. As conceptor matrix is making points cloud tighter this distribution was expected. It is worth mentioning that we can easily get a domain predictor G_{d_3} that is performing well on both G_3 and G_3C transformation - the accuracy of prediction whether the point belongs to target or source domain (equivalent to determinate if the point with given coordinates is blue or gray if we are considering distribution from first picture of Figure 3) reached 95%.

If we look closer at the plot of points we could see that many of target domain ones lie closely to the middle of coordinate system, so maybe the mapping works fine for just few examples which forms the multi-armed star, but most of samples are left with poor transformation, which is unclear for label classifier. Therefore after this test we can have few conclusions. Firstly we cannot use our conceptors intuitions from 3-dimensional space in d-dimensional one, as in 3D we are not able to approximate the multi-armed star obtained by samples with nothing but sphere. Nevertheless the improvement of predicting labels for target domain after including conceptor layer shows, that we should consider using conceptors during domain adaptation. The density of target domain points near the $(0,0,0)$ shows that those points clouds look similar at the first glance, but we cannot get a good classifier for target domain with this kind of representation. With this knowledge next things we can do to have a better understanding of our problem is trying to get a model with Gradient Reversal Layer in 3-dimensional space, and check the performance of conceptor layer in higher dimensions. Moreover we can check the variant with mapping feature extractor to 2-dimensions to verify all noticed behaviours.

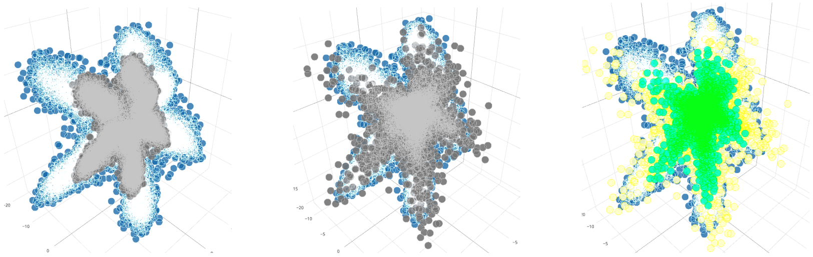


Figure 3: Transforming data with conceptor matrix $C_{MNIST-M}$ corresponding to target domain samples. First picture shows the 3D mapping of source domain (blue) and its transformation. The second one presents the transformation G_3C of samples form source domain and 3D mapping G_3 of target domain test set. Last picture shows distributions after training label classifier on G_3C mapping - blue points are again transformation of source domain, yellow ones are 3D mapping of target domain (equivalent to grey points from previous picture) and green points cloud is its G_3C transformation.

Before changing learning method we can see what happens when we change an activation function. All the plots shows that points cloud for both domains differ in size. One way to limit this magnitude is to use a *tanh* activation function, which maps the input into $(-1,1)$. Training model with 3-dimensional layer and *tanh* activations is even more difficult, I managed to get 78% accuracy on source domain test set, but it was performing better for target domain than model with *leakyReLU* (43%). As we can see in Figure 4 the mapped points formed something like a cube. The first plot presents the distribution of samples from source domain. Yellow points are locations of pictures of zero digit. We can see more those clusters on the edges of cube, so they should correspond to different digits. Second plot contains the distribution of target domain (gray). It is hard to find some pattern in them, they seem to be shuffled.

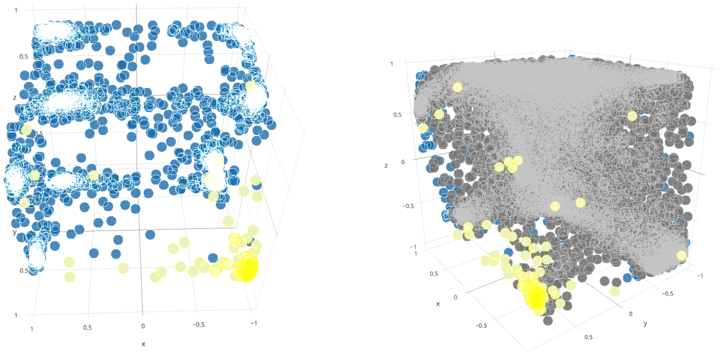


Figure 4: Changing activation function form *leakyReLU* to *tanh*

1.1.2 Gradient Reversal Layer

We already know that GRL is a helpful tool in domain adaptation. In higher dimensions models with GRL achieved even 80% accuracy. There are some approaches using Generative adversarial networks (GAN) that perform even better, so we can still improve our method. Therefore understanding of what really GRL does is necessary to move on. We can use a model with 3-dimensional feature extractor output then and visualize the results. Now our architecture is not only feature extractor with 3D mapping G_3 and label predictor G_y, but also domain classifier G_d that is using GRL during training so that representation yielded by G_3 does not contain information about domain. After a short training I got a model with 96% accuracy on source domain test set and 40% accuracy on target domain test set. The improvement is almost 30% in relation to model without GRL and domain predictor. We can expect now that if we plot the G_3 mapping of points clouds for source and target domain we should see again two multi-armed stars but more similar than in previous section.

However the result is really surprising. We can see in Figure 5 how samples from source and target domain were transformed. First picture presents the distribution of source domain. It looks similar to the one obtained with model without GRL. The second picture is something we didn't expect - samples from our target domain - MNIST-M formed a blob that is not similar to the source domain distribution at all. When we look at the last picture of the figure we can see the location of samples with label zero. Those from target domain are focused around zeros form source domain, but the matching seems much worse than in model without GRL. Therefore, the reason why GRL is such an improvement becomes even more incomprehensible.

I decided to take a closer look at the training. During the second run model reached even 52% accuracy on target domain's test set. This result was obtained after second epoch of training, after the last one the model was able to distinguish 47% of MNIST-M digits. Such a big difference in performance

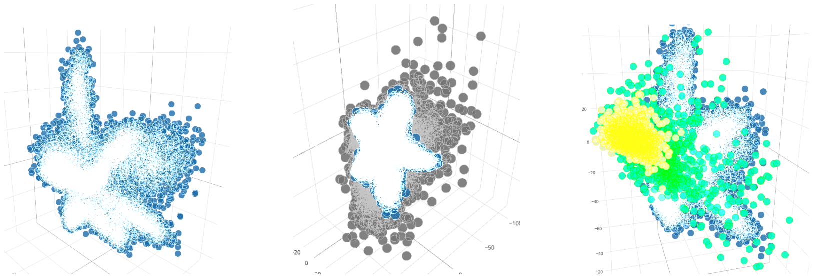


Figure 5: Output of feature extractor mapping when GRL is used during training. First picture presents how source domain samples are transformed. On the second one there are points clouds for both source (blue) and target (gray) domain. Last picture shows how points with 0 label are distributed - yellow come from source domain while green are from MNIST-M

between two runs of the same training for the same architecture shows that with 3-dimensional layer model becomes very unpredictable. I ran the training (for a new instance of architecture) few more times, during third run model reached 47% accuracy after first epoch but it ends with just 35%. During next training the loss became NaN so the model crashed. After that I got a model with 55% accuracy after 5-th epoch out of 10. Next runs also produced very diverse results. Result for source domain kept high during all of them. Figure 6 presents some results. First two pictures shows the distribution of points for model from second run - the one that reached 52% accuracy during training and ended with 47%. We can see that the best model from the training produced the representation closest to the distribution of source domain samples. After whole training the mapping is again very chaotic and noisy. The last picture is a plot from the training with 55% best and 52% final accuracy. Although the results are better the points clouds seem to be less suited to the source domain distribution.

Every training reached its best accuracy after one of the firsts epochs, that are characterized with lower λ coefficient. As λ changes nonlinearly from 0 to 1 I decided to use fixed value λ during the training. The coefficient was set to 0.3, 0.4 and 0.5. All the models did not managed to get a better performance than already obtained. The accuracy within a single training was also unstable, so fixing λ did not make an impact on the research. We are able to make over 50% improvement with GRL then, but we cannot tell clearly why it works. Figure 7 shows how particular digits were mapped by feature extractor from model with over 50% accuracy on MNIST-M test set. Each arm of star obtained from source domain samples corresponds to individual digit. In blob formed by target domain test set we can see a cluster for each digit, but it still looks like shuffled. Last picture presents the location of points for both domain in coordinate system. It seems that some of them overlap, what may cause better

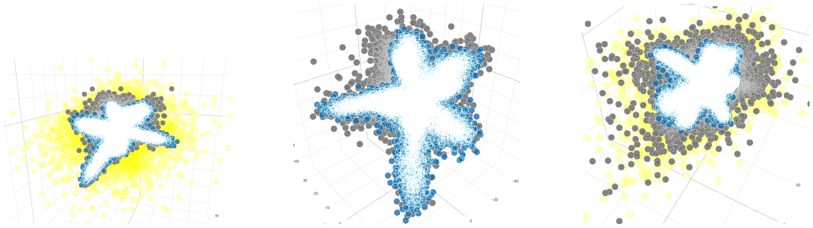
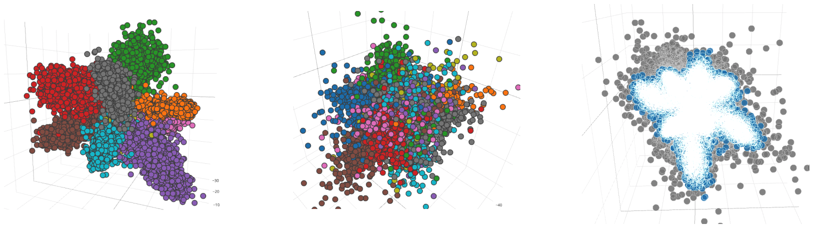


Figure 6: Model with GRL during other runs. First two pictures shows plots for model with 52% accuracy during training and 47% after last epoch. Yellow points are the final mapping of target domain, grey ones are obtained with feature extractor from model with best accuracy. Blue ones are transformed samples from source domain. Last picture is a corresponding plot for the best obtained model.



(a) MNIST mapped (b) MNIST-M mapped (c) Both points clouds

Figure 7: Mapping for particular digits. The points clouds are obtained with feature extractor of model that uses GRL during training. Last picture shows how both points cloud are located in 3-dimensional coordinate system.

performance of a classifier.

1.1.3 2-dimensional case

Visualization with digit division is a good way to understand what really happens during training. To make plots even more understandable we can change our feature extractor to map output into 2-dimensional space. Although finding right parameters gets more difficult, after few epochs of training we can get a model with 95% accuracy for source domain (MNIST) and 31% for samples from MNIST-M. These results are satisfying as we are asking our model to pack all the crucial information about 28x28 pixel photo into 2 real numbers. First two pictures of figure 8 presents the transformation of MNIST and MNIST-M test sets. Each point is a mapping of input image with digit corresponding to point's color. MNIST samples are formed into 7-armed star, each for different digit. Input images for 3 digits (3,5,8 in this case) are stacked in the middle, which does not interfere with good classifier performance. Samples from target distribution also formed similar 7-armed star but, like mentioned before, a large number of them was mapped near the middle of coordinates system. This characteristic is more clearly visible in 2-dimensional case. As input images for different digits are represented by similar numbers the classifier cannot be sure while predicting the label.

We can also apply a training with Gradient Reversal Layer for this architecture. Just like before the source domain samples are mapped into multi-armed star and visualization of points cloud obtained by mapping the target domain resluts with a plot of huge blob (last picture of figure 8). The performance of this model is better thanks to the use of a GRL, but I could not get any new useful knowledge from this research. Also trying to improve the model's performance with conceptor did not succeed. As in the 3-dimensional case the conceptor of our mapped set of points is an ellipse that is limiting the points cloud. Here we can see that it would be difficult to approximate our samples by something but a circle, hence the conceptor matrix is almost an identity matrix. Therefore filtering the point by the matrix hardly changes its coordinates. Once again I must point out that we cannot be sure taht this behaviour of conceptor would happen in higher dimension.

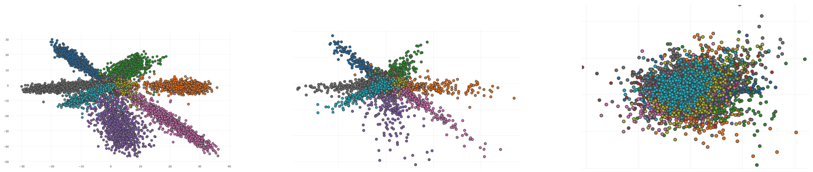
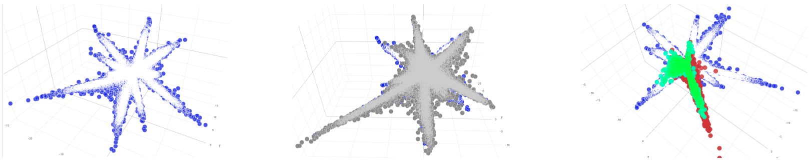


Figure 8: Mapping for particular digits with feature extractor yielding 2-dimensional vector. First picture shows the distribution for samples from source domain, second one for input from target domain and the last one is a mapping of target domain samples after training with GRL applied.

1.2 SVHN and MNIST

All previous observations were made for highly correlated source and target domain, as MNIST-M is obtained with MNIST dataset. To be sure that results of the research is not caused by the connection between MNIST and MNIST-M we should try to run the experiments for some other domain. Once again I used SVHN dataset as a source domain and tried to improve its performance in MNIST digit recognition.



(a) SVHN (b) SVHN and MNIST (c) Zeros of datasets

Figure 9: The visualization of samples transformed by feature extractor with output size equals 3. (a) presents the star formed with samples from source domain (SVHN), (b) shows how data from target domain is mapped and (c) where zeros for both domains lie in the space