Introduction to Machine Learning

Final project: Report

Author: Irek Nazmiev, SE-2

Link to Colab: click

Outline

Explanation of the model

Formalities

The theory and principles of work are taken from Unsupervised Domain Adaptation by Backpropagation. Further text is also based on this paper.

The code samples are taken from wogong/pytorch-dann GitHub repository.

This model has nothing common with the one from the previous submission as its idea was based on paper rather than our labs, and implemented using another sources of code.

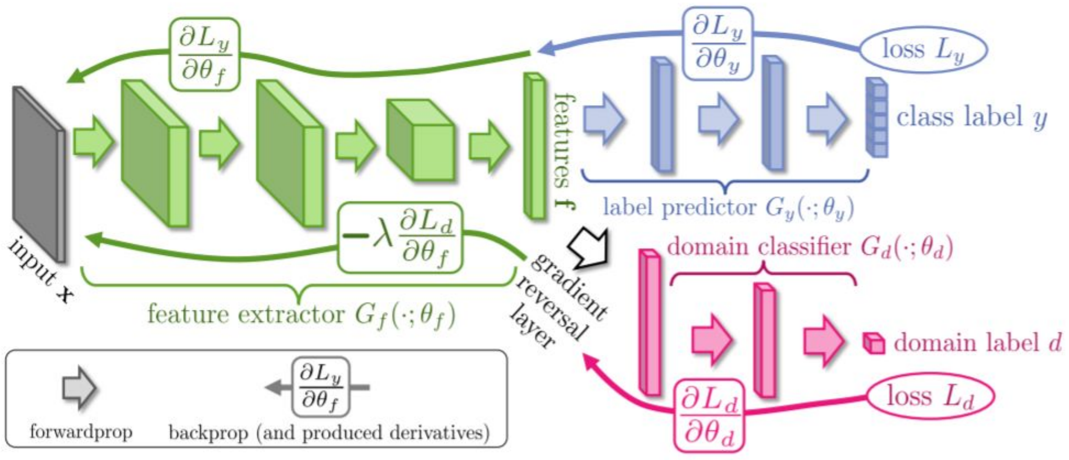
Basic concepts

The model provides *unsupervised domain adaptation of deep feed-forward architectures, which allows large-scale training based on large amount of annotated data in the source domain and large amount of unannotated data in the target domain* (Ganin Y, Lempitsky V. Unsupervised domain adaptation by backpropagation. arXiv preprint arXiv:1409.7495. 2014 Sep 26.).

The main approach is to combine domain adaptation and deep feature learning within one training process - *deep domain adaptation*. The goal is to embed domain adaptation into the process of learning, so that the classification is being processed considering features that are both *discriminative* and *invariant* to the change of domains. Thus, the feed-forward network can be applied to the target domain - because of decreasing the gap between source and target domains by using domain adaptation.

To achieve discriminativeness and domain-invariance of learning features, they're jointly optimized using two discriminative classifiers operating on them: the *label predictor* that predicts class labels, and the *domain classifier* that discriminates between the source and the target domains during training.

All three training processes are embedded into an appropriately composed *deep feedforward network* that uses standard layers and loss functions, trained using standard backpropagation algorithms based on stochastic gradient descent with momentum (initially, but further SGD was changed to Adam optimizer due to task requirements). Also the *gradient reversal layer* was used, which leaves the input unchanged during forward propagation and reverses the gradient by multiplying it by a negative scalar during the backpropagation.



$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_y}{\partial \theta_y}$
loss L_y
class label y
features $\mathbf{f}$
label predictor $G_y(\cdot; \theta_y)$
$-\lambda \frac{\partial L_d}{\partial \theta_f}$
input $\mathbf{x}$
feature extractor $G_f(\cdot; \theta_f)$
gradient reversal layer
domain classifier $G_d(\cdot; \theta_d)$
domain label d
loss L_d
$\frac{\partial L_d}{\partial \theta_d}$
forwardprop
backprop (and produced derivatives)

Picture 1: The image is taken from here

The proposed architecture includes a deep feature extractor (**green**) and a deep label predictor (**blue**), which together form a standard feed-forward architecture.

> **Unsupervised domain adaptation** is achieved by adding a *domain classifier* (**red**) connected to the feature extractor via a *gradient reversal layer*.

Gradient reversal multiplies the gradient by a certain negative constant during the backpropagation-based training, and ensures that the feature distributions over the two domains are made similar (as indistinguishable as possible for the domain classifier), thus resulting in the *domain-invariant* features.

Implementation details

The code is based on wogong/pytorch-dann which is supposed to be an implementation of paper chosen by me as primary.

The code GitHub repository was cleaned from excess features, reformated, rebuilt. Also, it was properly documented.

In addition, Xavier initialization and Adam optimizer were added in order to satisfy project requirements. Anyway, they didn't particularly affect the results much.

The code and results of program work are available on Google Colab: link.

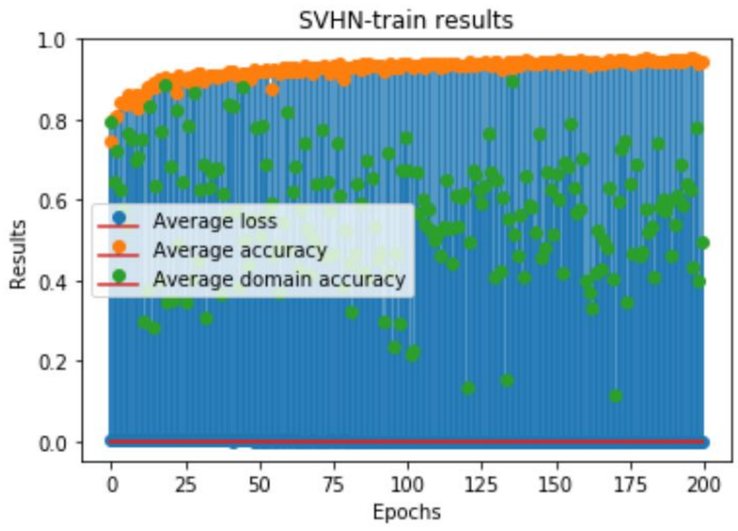
The list of important hyperparameters

num_epochs = 200, log_step = 50, eval_step = 1, manual_seed = None, alpha = 0, lr = 0.01, momentum = 0.9, weight_decay = 1e-6

Accuracy

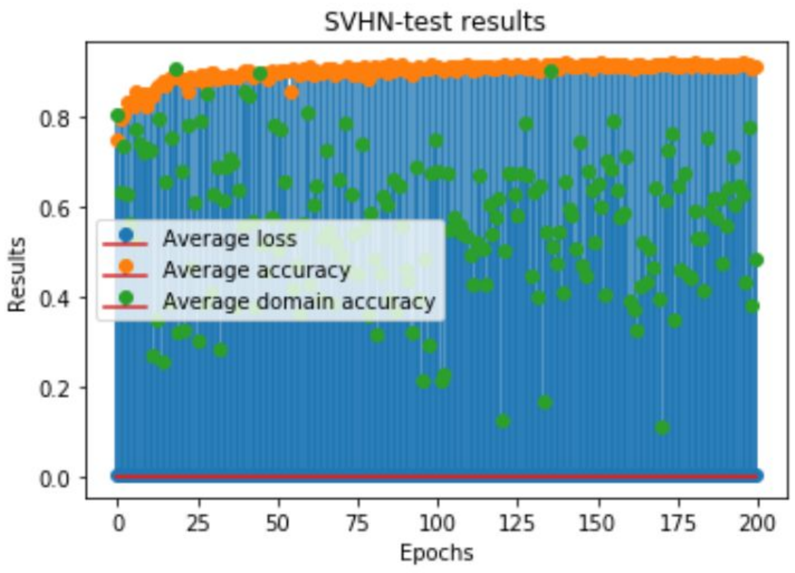
SVHN-train

The *average loss* was always insignificantly **small**.
The *maximum average accuracy* of SVHN-train dataset is **95.26%**.
The *maximum average domain accuracy* of SVHN-train dataset is **88.447880%**.



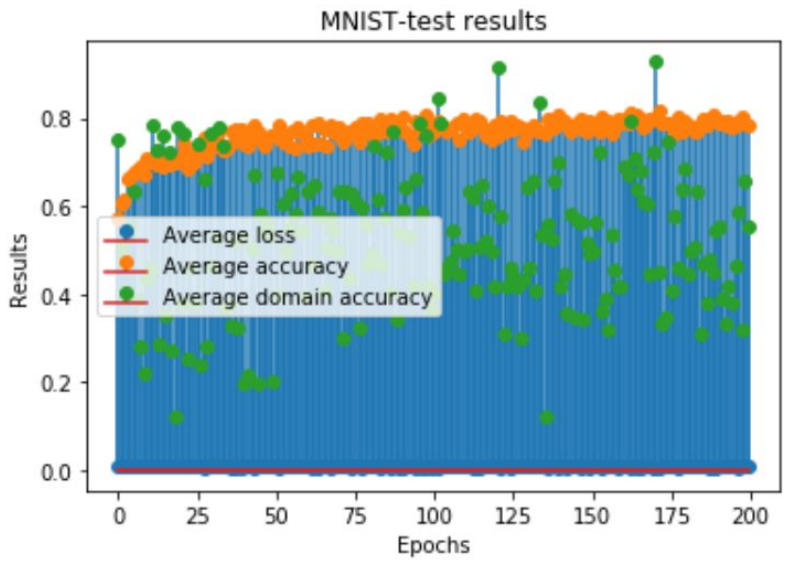
SVHN-test

The *average loss* was always insignificantly **small**.
The *maximum average accuracy* of SVHN-test dataset is **92.20%**.
The *maximum average domain accuracy* of SVHN-test dataset is **90.821275%**.



MNIST-test

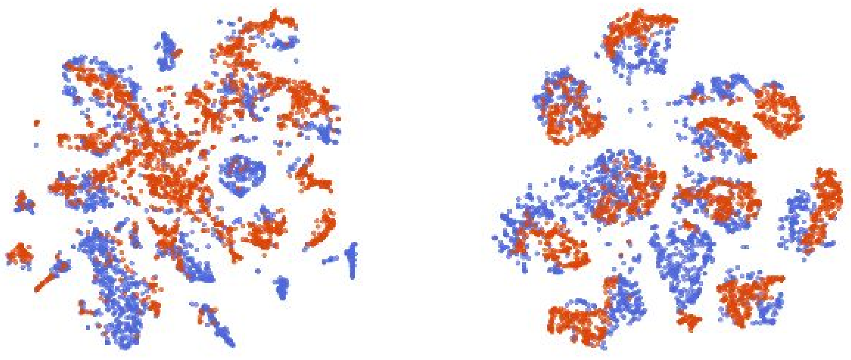
The *average loss* was always insignificantly **small**.
The *maximum average accuracy* of MNIST-test dataset is **81.79%**.
The *maximum average domain accuracy* of MNIST-test dataset is **92.858574%**.



Latent space

The implementation of Latent space visualization using t-SNE from sklearn was added, but not completely integrated into the project. The functions and code parts taken from here can be found in the project being commented or just not used at all. That’s happened because of me not being able to figure it out in time.

But here is an example of how it could look loke from here:



Conclusion

The results show that used technique of Unsupervised Domain Adaptation by Backpropagation solves the problem, but the results could be better. The idea is simple and can be integrated in just common feed-forward model with backpropagation, and the simplicity of the model clearly explains the results of model's work.

The domain gap was solved successfully by separating the program into 3 parts: deep feature extractor, deep label predictor and domain classifier which is used exactly for these purposes.

The simplicity of the approach leads to the not very high results of accuracy. While other approaches can give up to 99%, current approach gives only around 81% of average accuracy (the maximum result in my implementation, which is improved version of paper's one).

The model can be improved further by using better augmentation techniques (in current model the augmentation is awful), tuning better hyperparameters and randomizing better seed.

Initially I wanted to get high accuracy by using complicated augmentation techniques on base model, that could transform samples from SVHN set to the MNIST ones as close as possible. But the idea was thrown away. Now I understand that it could be successfully integrated into the project.