Transfer Learning on EEG Dataset

518021910750 Futao Wei

Abstract

In this course project, we try to solve an emotion classification problem based on electroencephalogram (EEG) data. First, we train a C-SVM classifier as our baseline, which achieves an accuracy of approximately 58%. Then, we implement two classic transfer learning algorithms (UDAB [2] and ADDA [3]) for this task and test their performance.

1 Introduction

Brain-computer interface (BCI) makes it possible for computers to detect and understand human emotions in a real-time manner. After collecting EEG data using an EEG headset, effective algorithms are needed to interpret the signals.

EEG-based Emotion classification is a typical pattern classification problem in this area. Unlike traditional visual and audio signals, EEG signals are non-stationary with low signal-to-noise ratio and high variance among individuals, making their classification a challenging task. Meanwhile, transfer learning is a powerful machine learning technique which aims at transfer knowledge of one domain to another, and thus offers excellent potential for eliminating the inter-subject discrepancy (domain shift) among EEG data of different participants.

In this course project, our work can be summarized as three main parts:

1. We train a C-SVM classifier as our baseline.
2. We implement UDAB [2] algorithm for EEG-based emotion classification problem.
3. We implement ADDA [3] algorithm for EEG-based emotion classification problem.

2 Dataset Description

SJTU Emotion EEG Dataset(SEED) [1, 4], is a collection of EEG dataset provided by the BCMI laboratory which is led by Prof. Bao-Liang Lu. In this course project, a subset of SEED is used, which includes the EEG data of 15 participants. During the experiments, each participant watched 15 film clips about 3 min each of 3 emotion categories (happy, neutral, sad) for emotion arousal, while their EEG signals were collected. The EEG data were sampled once per second for 3394 seconds, which is of dimensionality 310.

3 SVM Classification

SVM is a simple yet effective machine learning technique, which learns a hyperplane for classification.

3.1 Implementation

C-SVM is used as our baseline. We employ the LinearSVC class of the scikit-learn library of Python to test SVM's performance on our EEG dataset.

Firstly, the 310-dimension features are normalized by removing the mean and scaling to unit variance using StandardScaler in the scikit-learn library, so that features of relatively larger magnitude do not dominate the objective function.

Then we do 15 tests with each of the 15 participants' data for testing and the rest for training and calculate the average accuracy.

3.2 Results

It's found that the mean accuracy peaks at 57.83% when $C = 0.25$. However, we note that the variance of accuracy is unacceptably high (93.90% at best, 39.16% at worst). In other words, the classification performance of SVM fundamentally depends on whether the testing EEG signals are distributed similarly to the training signals.

4 UDAB

4.1 Main Idea

Unsupervised Domain Adaptation by Backpropagation (UDAB) algorithm [2] uses a feature extractor network to project features in source and target domain to a common domain to reduce the domain shift.

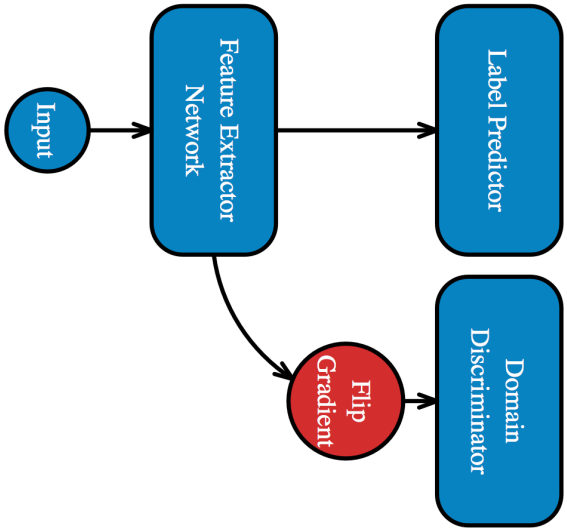

Figure 1: UDAB architecture[1]

As is shown in Figure 1, the UDAB network is composed of three parts – a feature extractor network, a label predictor, and a domain discriminator. Two losses are defined for network optimization – a classification loss and a domain discrimination loss. The classification loss is calculated with data in the source domain and is used to train the label predictor and feature extractor, i.e., encouraging the label predictor to classify more accurately and the feature extractor to extract features of better discriminativeness; the domain loss is calculated with data in both the source domain and the target domain and is used to train the domain discriminator and the feature extractor, i.e., encouraging the discriminator to distinguish between data in different domains and the feature extractor to project original features to a common domain (Note that a flip gradient layer is used for adversariality).

All the three networks are trained simultaneously. Moreover, a weight λ is introduced in the backward process of the flip gradient layer, which gradually increases from 0 to 1 during the training. The main purpose is to suppress the noisy signal from the domain discriminator at the early stages of training [2].

4.2 Implementation[2]

We use Pytorch library to implement our model. As described in the previous section, the model has three sub-networks – a feature extractor, a label predictor, and a domain discriminator; the layer-wise details can be found in `UDAB/models.py`. Besides, we choose cross entropy loss and Adam optimizer for model optimization as suggested in [2].

Like the experiments with SVM, we carry out a similar 15-fold cross-validation, i.e., taking one participant's data as target data without label and the rest as labeled source data.

What's more, we also conduct an ablation experiment which directly trains the predictor and feature extractor with labeled data from the source domain without domain adaptation, so that we can determine the contribution of domain adaptation.

4.3 Results

As is shown in Table 1, UDAB with domain adaptation achieves a roughly 35% increase in average classification accuracy compared with SVM (from 57.83% to 78.21%).

However, the improvement in mean accuracy is very little (from 76.15% to 78.21%) when contrasted with UDAB without domain adaptation.

[1] This figure is borrowed from the slides of this course.

[2] We referred to a GitHub repository during this implementation.

-	SVM	UDAB(DA)	UDAB(no DA)	ADDA(DA)	ADDA(no DA)
mean accuracy	57.83%	78.21%	76.15%	69.88%	72.60%
variance	0.019	0.008	0.011	0.014	0.004

Table 1: Performance Comparison

Meanwhile, we find that domain adaptation decreases variance of accuracy effectively (about 27% from 0.011 to 0.008).

5 ADDA

5.1 Main Idea

Adversarial Discriminative Domain Adaptation (ADDA) algorithm [3] improves on UDAB algorithm.

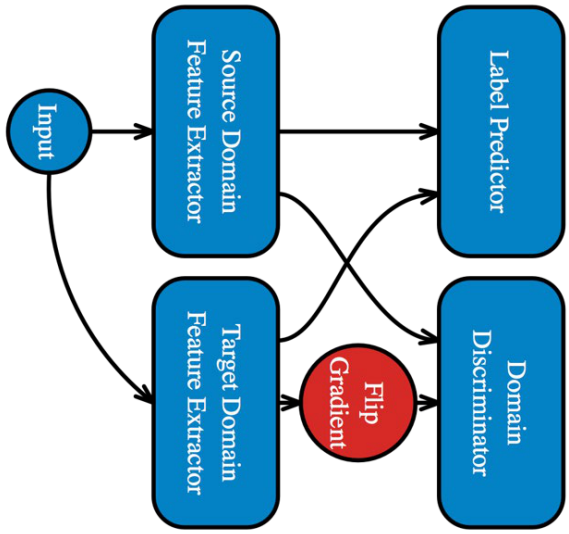

Figure 2: ADDA architecture[3]

As is shown in Figure 2, the ADDA network is composed of four parts – a feature extractor for source domain, a feature extractor for target domain, a label predictor, and a domain discriminator.

Despite the similar structure with UDAB, the training of ADDA network is vastly different. Firstly, the source domain feature extractor and label predictor are trained using only source domain data, i.e., encouraging the predictor to classify more accurately and the feature extractor to obtain more discriminative features. Secondly, the parameters of source domain feature extractor are copied to the target domain feature extractor. Thirdly, the domain discriminator and target domain feature extractor are trained using both source and target domain data, i.e., encouraging the discriminator to distinguish between features from source domain and from target domain and the target domain feature extractor to project the original features to a distribution similar to the output of source domain feature extractor (Note that the source domain feature extractor is fixed during the third stage of training).

To predict labels of data in target domain, source domain feature extractor and domain discriminator are no longer needed. The input data should be passed to target domain feature extractor and then to the label predictor.

5.2 Implementation[4]

We use Pytorch library to implement our model. The layer-wise details can be found in `ADDA/models.py`. Besides, we choose cross entropy loss and Adam optimizer for model optimization as suggested in [3].

Like the experiments with SVM, we carry out a similar 15-fold cross-validation, i.e., taking one participant's data as target data without label and the rest as labeled source data.

What's more, we also conduct an ablation experiment which directly trains the predictor and target domain feature extractor with labeled data from the source domain without domain adaptation, so that we can determine the contribution of domain adaptation.

5.3 Results

As is shown in Table 1, ADDA with domain adaptation achieves a roughly 21% increase in average

[3]This figure is borrowed from the slides of this course.

[4]We referred to a GitHub repository during this implementation.

classification accuracy compared with SVM (from 57.83% to 69.88%). However, in the ablation experiment, ADDA without domain adaptation performs even better in terms of both accuracy and variance. In other words, "negative transfer" happens.

6 Discussion

After testing UDAB and ADDA, we find that naively applying these two algorithms to EEG data cannot achieve a substantial improvement of performance like they did on image classification datasets like MNIST in [2, 3], although UDAB with domain adaptation obtains the highest accuracy with low variance.

We think the cause may be two-fold:

1. Directly applying transfer learning algorithms fails to take into account the special characteristics of EEG data, e.g., temporal co-relation. Therefore, moderate modifications of network architecture have to be made to take advantage of these properties.
2. The parameter tuning wasn't carried out in an exhaustive way due to limit on time, so that our models are likely to be sub-optimal.

7 Conclusion

In this course project, we implement two classic transfer learning algorithms – Unsupervised Domain Adaptation by Backpropagation (UDAB) and Adversarial Discriminative Domain Adaptation (ADDA), and compare their performance on SEED dataset with SVM as a baseline. We find that, when directly applying the algorithms to EEG data, the performance gain is very limited (UDAB) and even negative (ADDA) after removing the contribution of the neural network without domain adaptation.

References

[1] Ruo-Nan Duan, Jia-Yi Zhu, and Bao-Liang Lu. "Differential entropy feature for EEG-based emotion classification". In: *6th International IEEE/EMBS Conference on Neural Engineering (NER)*. IEEE. 2013, pp. 81–84.

[2] Yaroslav Ganin and Victor Lempitsky. "Unsupervised Domain Adaptation by Backpropagation". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1180–1189. URL: http://proceedings.mlr.press/v37/ganin15.html.

[3] Eric Tzeng et al. "Adversarial Discriminative Domain Adaptation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[4] Wei-Long Zheng and Bao-Liang Lu. "Investigating Critical Frequency Bands and Channels for EEG-based Emotion Recognition with Deep Neural Networks". In: *IEEE Transactions on Autonomous Mental Development* 7.3 (2015), pp. 162–175. DOI: 10.1109/TAMD.2015.2431497.

Appendices

A Code Folder Description

The code for this project is inside `project_code` folder.

- In `project_code/datatools` folder are scripts for loading the original .mat data files and transforming the data to ready-for-use forms, e.g., torch.utils.data.Dataset. In the three .pickle files are processed data for loading.
- In `project_code/svm` folder is our implementation of SVM as a baseline.
- In `project_code/UDAB` folder is our implementation of UDAB algorithms. Run `main.py` file in this folder to get the accuracies.
- In `project_code/ADDA` folder is our implementation of ADDA algorithms. Run `main.py` file in this folder to get the accuracies.