Tackling Black Box Learning using Neural Networks

Titas Nandi
Supervisor: Dr. Arijit Mondal

1 Abstract of the project

The project aims at developing a neural network architecture to tackle the problem of **representation learning**. We train a classifier to predict labels on a dataset that is **not** human readable, without the knowledge of what the data consists of. We create a deep learning model that can learn from **both** labeled and unlabeled data, to take advantage of the large amount of **unannotated** data available. The system will be tested on 10,000 instances from a data distribution whose source is again **unknown**. The idea is inspired from the **Black Box Challenge** on Kaggle as part of *ICML 2013 Challenges in Representation Learning* and the training and test data is obtained from there.
Some of the interesting approaches used for solving this problem include using a sparse filtering technique [1] and an autoencoder architecture using ensemble learning [2].

2 Introduction

2.1 Literature survey

We looked into the major approaches used by the top performing teams in the challenge to get an idea about possible methodologies and benchmark results. The major issue was to envisage methods to **extract** information from the unsupervised data to support supervised classification. Lee [3] used an interesting method of assigning **pseudo labels** to the unsupervised data and train and update the pseudo labels every weight update of the neural network. Lukasz [1] used a method in which the unsupervised data was not directly used in training, but used in a **pre-training step** for **feature selection**. They used a method of unsupervised feature selection called sparse filtering, and used the reduced set of features for supervised classification. Xie [2] stacked horizontal and vertical **voting** along with deep learning to attain good results on the task.

3 Resources

We used the data provided by the organizers of this challenge. It consists of:

Train	**Test**	**Unannotated**	**Classes**
1000	10000	135735	9

Table 1: **Data ICML Black Box Challenge**

For sparse filtering, we referred the famous paper written by Ngiam [4] and the code available on github [1].This is a very generic version which we had to modify for our problem. For sparse filtering method, we also took help from code written by the best performing team in the contest [2]. For training neural nets, we used Keras [3] and we used both Matlab and Python for programming.

[1] https://github.com/jngiam/sparseFiltering

[2] https://bitbucket.org/dthal/blackbox-challenge-code

[3] https://keras.io/

3.1 Work done

We focused on both the methodologies for using the unsupervised data to our help. We used sparse filtering for **dimensionality reduction**, and then trained a **Multi Layer Perceptron** model on the supervised data using the selected features. In a separate approach, we also trained a neural network on both supervised and unsupervised data together, using the concept of pseudo labels. We present all our experimentations and architectural details in the following sections, along with baseline and benchmark results.

3.1.1 Baselines

The following baselines have been proposed by the organizers of the task:

- **Random Baseline** - 11.1%
- **Logistic Regression** - 21.1%
- **ZCA + 1 layer net** - 41%
- **ZCA + 3 layer net** - 51.5%

It is evident that the **last** baseline is quite a strong one and tough to beat. This was reflected in the competition where only one-third of the participating teams could beat this baseline.

3.1.2 Benchmark Results

- **Sparse Filtering + Feature Selection + SVM with linear kernel - 70.22 %**
- **Pseudo Labels + Denoising Autoencoder + Dropout - 69.58 %** [3]
- **Horizontal and Vertical Ensemble for Classification - 69.14 %**

3.1.3 Sparse Filtering

Sparse filtering is a method for **unsupervised feature learning**. The beauty of the algorithm lies in its simplicity - it works as a method for producing an alternate (reduced dimensionality) representation of the features by optimizing a simple cost function - the **sparsity of L2-normalized features**. This is in contrast to other popular unsupervised methods like sparse RBMs and autoencoders, which essentially try to approximate the distribution of the data and thus require a lot of **hyperparameter tuning**. Sparse filtering works on optimization of:

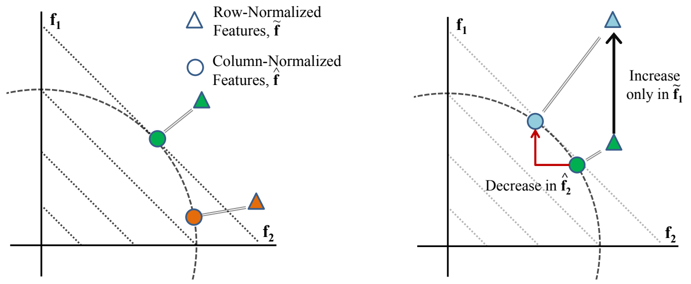
- **Population sparsity** - Only a few non-zero features are essential for a given training example
- **Lifetime sparsity** - Only a few training examples are essential for a given feature
- **High dispersal** - The distribution of features for any training example must be similar to that of any other example, this ensures uniform activation of features

Sparse filtering introduces competition among features by reducing them to their L2 norms, and retains only the **non-redundant** ones.

3.1.4 Sparse Filtering + Supervised Training

The approach towards solving this challenge using this method is detailed below:

- Break the large unsupervised data into **5000** example chunks for faster training
- Train a **feedforward Sparse Filter** on these chunks, where each chunk will be pulled in for training in data batches of given count, and produce 10 feature sets having revised weights



$\mathbf{f_1}$
Row-Normalized Features, $\tilde{\mathbf{f}}$
Column-Normalized Features, $\hat{\mathbf{f}}$
$\mathbf{f_2}$
$\mathbf{f_1}$
Increase only in $\tilde{\mathbf{f}}_1$
Decrease in $\hat{\mathbf{f}}_2$
$\mathbf{f_2}$

Figure 1: Left: Sparse filtering showing two features (f_1, f_2) and two examples (red and green). Each example is first projected onto the ℓ_2-ball and then optimized for sparseness. The ℓ_2-ball is shown together with level sets of the ℓ_1-norm. Notice that the sparseness of the features (in the ℓ_1 sense) is maximized when the examples are on the axes. Right: Competition between features due to normalization. We show one example where only f_1 is increased. Notice that even though only f_1 is increased, the normalized value of the second feature, $\hat{f}_2$ decreases.

- We picked out the top performing **120 features** out of 1875 initially
- Find the **revised representation** for the training and test (or development) data from these learned weights
- Train a **feedforward** Neural Network on the supervised data using these revised weights
- Experiment with the Neural Network architecture to achieve the best possible results on the public dataset

Table 2 illustrates the various experiments in neural network architecture that we did. Our best performing architecture achieves an accuracy of **64.74%** on the public set, at par with the best performing systems.

Submission	**Neurons**	**Layers**	**Activation**	**Dropout**	**Optimizer**	**Epoch**	**Batch Size**	**Accuracy**
Best	**1500**	**2**	**sigmoid**	**0.4**	**adam**	**200**	**128**	**64.74**
1	1000	1	relu	0.4	adam	20	128	60.12
2	200	2	sigmoid	0.4	adam	20	128	51.22
3	1000	2	sigmoid	0.4	adam	100	128	64.02
4	1000	3	sigmoid	0.4	adam	100	128	63.86
5	1000	2	sigmoid	0.4	adam	1000	128	63.80
6	1500	2	sigmoid	0.5	adam	200	128	64.50
7	2000	2	sigmoid	0.4	adam	200	128	64.66
8	1500	2	sigmoid	0.3	adam	200	128	64.66
9	1500	2	sigmoid	0.4	adam	200	256	64.42
10	1500	2	sigmoid	0.4	sgd	200	128	39.50
11	1500	2	relu	0.4	adam	200	128	61.72

Table 2: Neural Network Experiments on sparsed features

3.1.5 Plots

Since we could not track the test labels for the competition, thus, here we present the plots of **epochs vs accuracy** and **loss vs accuracy** on the validation set obtained by splitting the 1000 training examples in 1:4 ratio. The results are obtained on our highest performing neural network model.

We can see there are significant improvements in accuracy due to sparse filtering.

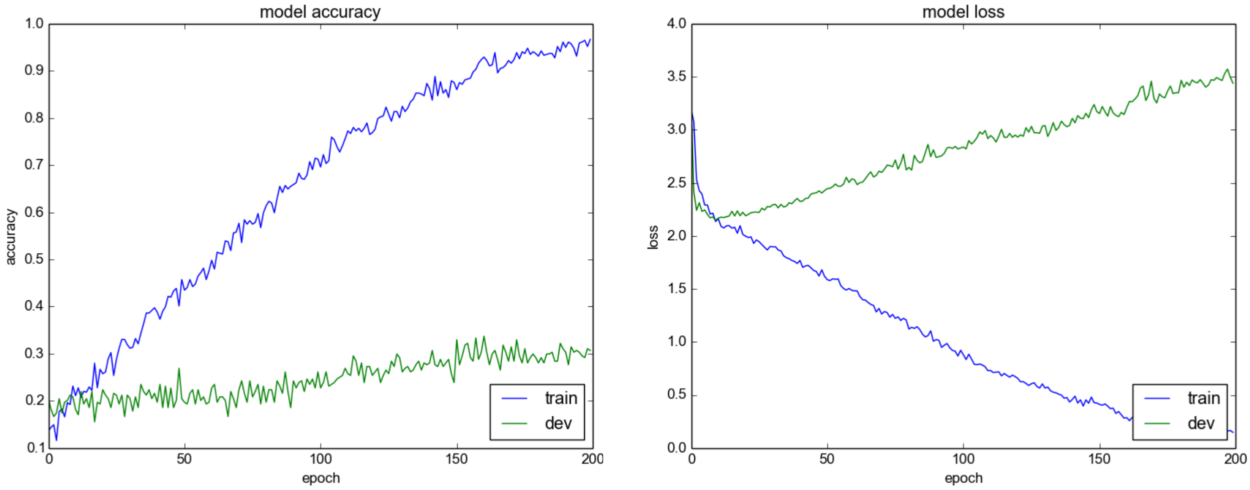


Figure 1: Validation plots for original data - 1875 dimensions

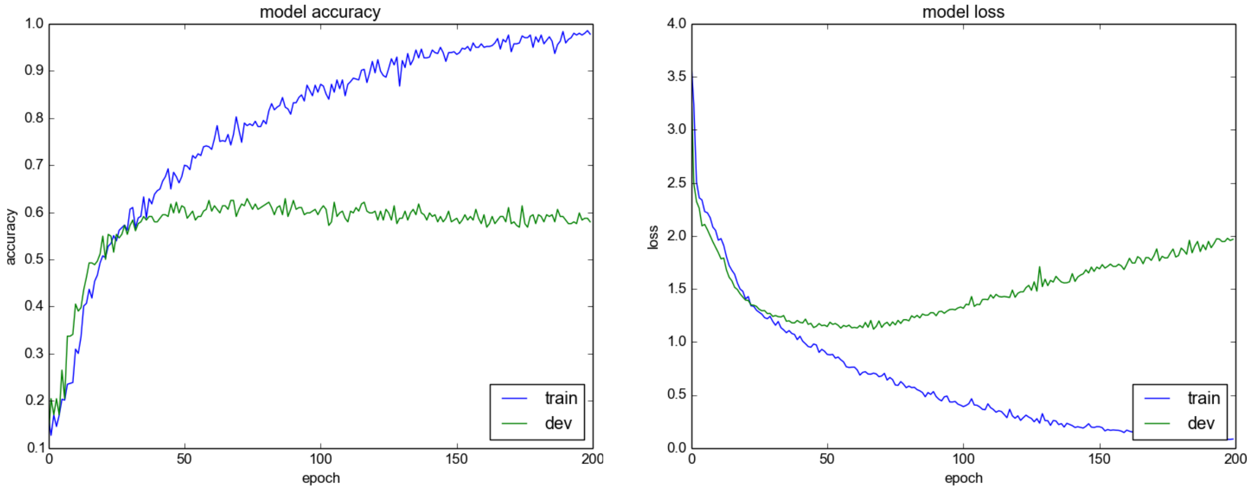
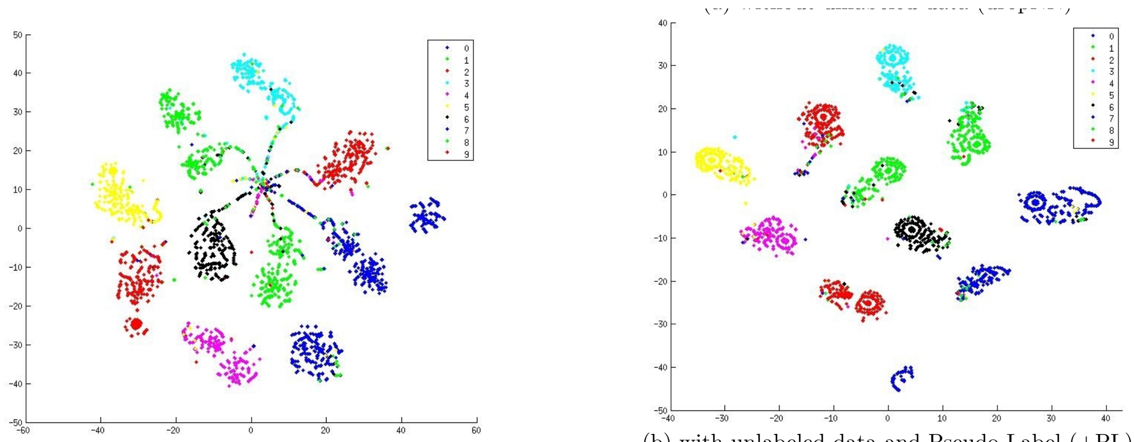


Figure 2: Validation plots for sparse filtered and ensembled data - 1200 dimensions



(a) without unlabeled data (dropNN)

(b) with unlabeled data and Pseudo-Label (+PL)

Figure 3: t-SNE 2-D embedding of the network output of MNIST test data

3.1.6 Pseudo Labels

This is a very interesting approach adopted by one of the teams, wherein we also use the unsupervised data in a supervised learning framework by generating probable labels, or **pseudo labels**, for it. In effect, this method works because it induces **entropy regularization** [5], that prefers **low density** separation among classes. We train a neural network using both sets of data to improve the **generalization performance**, the pseudo labels are expected to gradually improve (called **self-training**) and finally converge towards their correct values after several runs of training. The approach is detailed below:

- We train a feedforward neural net on the supervised examples
- We find probable labels of the unsupervised data
- Retrain the neural network with the **combined** data, i.e, supervised data with true labels and the unsupervised data with pseudo labels
- At this point, the network might not have learnt the pseudo labels properly or might be **overfitted**
- Retrain the network until **convergence** (till there are no significant changes in predicted labels)

Table 3 illustrates the predicted accuracies after multiple runs of the pseudo labels for two different neural network architectures

Iterations	**1 hidden layer with 1000 neurons**	**2 hidden layers with 1500 neurons each**
1	56.04	47.86
3	55.48	47.98
6	55.26	48.16
10	55.00	48.10
17	56.08	48.74

Table 3: Pseudo Labels training after certain iterations of the algorithm

We find **irregular** patterns in the pseudo labels training, which might be due to the **same** weightage given to supervised and unsupervised data in the error function. The error function must be **dominated** by the supervised training examples (since their labels are definitely correct), and we need to monitor the weight coefficients given to the unsupervised data in a **time dependent manner** as presented in the paper [3]. But, still the labels must improve with growing number of iterations and we are looking into the reason of this irregular behaviour and resultant low accuracy.

The code for our implementations or supporting codes used from other repositories or sources is available here [4].

3.2 Future work

Semi-supervised learning is a highly interesting and important part of Machine Learning. **Annotation** is expensive and more time taking than generating crude representations of data. Solving this problem efficiently indicates that we can solve many real life problems with a **small** amount of supervised data.
Future work includes further delving into the pseudo labels approach and understanding its intricacies to solve the issues in our present implementation. The sparse filtering method works well and we will experiment with neural network architectures to improve the accuracy.
A careful **hybrid** of both these methods can incorporate the strengths of both models, and challenge benchmark results. We also plan on applying this approach on data from **cQA sites** (where also we have a lot of unannotated data) to solve the problem of good answer selection and ranking, which is a project I have been working on for a long time.

References

[1] L. Romaszko, "A deep learning approach with an ensemble-based neural network classifier for black box icml 2013 contest," in *Workshop on Challenges in Representation Learning, ICML*, pp. 1–3, 2013.

[2] J. Xie, B. Xu, and Z. Chuang, "Horizontal and vertical ensemble with deep representation for classification," *arXiv preprint arXiv:1306.2759*, 2013.

[3] D.-H. Lee, "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks," in *Workshop on Challenges in Representation Learning, ICML*, vol. 3, p. 2, 2013.

[4] J. Ngiam, Z. Chen, S. A. Bhaskar, P. W. Koh, and A. Y. Ng, "Sparse filtering," in *Advances in neural information processing systems*, pp. 1125–1133, 2011.

[5] Y. Grandvalet, Y. Bengio, *et al.*, "Semi-supervised learning by entropy minimization.," in *NIPS*, vol. 17, pp. 529–536, 2004.

[4] `https://github.com/TitasNandi/ICML-BlackBox-Challenge`