ARDL: Attention based Random Deep Learning for Classification
ECE 247 Project

Siddhant Patil
siddhantpatil@g.ucla.edu

Ayushi Agarwal
ayushi15@g.ucla.edu

Dipti Ranjan Sahu
diptisahu11@g.ucla.edu

Rachel Menezes
rachelmenezes@g.ucla.edu

University of California, Los Angeles (UCLA)

Link to code repository: https://github.com/NNDLProject/ARDL

Abstract

The exponential growth in the number of complex datasets every year requires more enhancement in machine learning methods to provide robust and accurate data classification. Lately, deep learning approaches have achieved surpassing results in comparison to previous machine learning algorithms across many domains. In this paper, we introduce Attention based Random Deep Learning (ARDL): a new ensemble, deep learning approach for classification. ARDL trains multiple models of Deep Neural Network (DNN), Convolutional Neural Network (CNN) and Long Short-term Memory (LSTM) in parallel and combines their results using attention to produce better result of any of those models individually. This paper describes ARDL and shows test results for 20newsgroup text data. These experimental results demonstrate that ARDL achieves a minimum of 5% improvement in accuracy as compared to individual baseline models.

1. Introduction

In the data science community, classification and regression have been central tasks since ages. We limit the scope of the project to classification tasks. In the industry, classification is required in variety of applications. A few examples are document categorization, image classification and sentiment analysis. In the project, we specifically focus on classifying the text data. The traditional solutions to these problems are to train the Random Forest Classifier [5] and the XGBoost Classifier [2]. As these models, perform bagging and boosting respectively, they are supposed to generalize well on datasets. Over time datasets became complex and the recent solutions in the literature suggest that we train Deep Neural Networks to solve the problem [8], [6], [3]. The drawback with using just the Deep Neural Networks is that the network is specific to the problem that we target to solve and the choice of appropriate number of layers and hidden units is critical. We try to build a more general solution that will work with different data types on the classification problem without major changes in the architecture.

In the project, we target solving a text classification task using a mixture [7] of the Deep Neural Network (DNN), Convolutional Neural Network (CNN) and LSTM. Our architecture comprises of multiple CNNs, DNNs and LSTMs of random number of layers and nodes. We perform attention [4, 9] on the outputs of these networks. Intuitively, this can be thought of as model ensembling. Training networks with random number of layers and nodes helps in model generalization. The main contributions of our project are as follows:

- Use of a random combination of DNN, CNN and LSTM for a classification task in order to make the model more robust.
- Use of attention for combining the results of the different models as against the usual approach of majority voting.
- Use of batch normalization after each layer for quicker convergence and regularization.
- A set of tuned models for the 20 newsgroups dataset [1] (Link to dataset: https://archive.ics.uci.edu/ml/datasets/Twenty+Newsgroups). The pipeline can easily be applied to other text datasets. We restrict to only this dataset due to a limitation on the computation resources. We targeted training multiple models for analysis within the given time. We have converted the 20 class classification problem into a 7 class classification problem by merging classes of same major category.

The report is organized as follows. Section 2 talks about data preprocessing, random model generation and training. Section 3 compares the results achieved using individual networks vs ARDL. Section 4 elaborates on why ARDL performs better as compared to other networks.

2. Methodology

2.1. Data Preprocessing

In this paper, we use 20newsgroup data for text classification. We pass the text data through preprocessing steps where we convert the text to lowercase and eliminate stopwords, punctuations, hyperlinks and irrelevant characters. We employ GloVe 50D and 300D to extract word embeddings and use these embeddings to train and evaluate our model. For the words in the GloVe corpus, we use the embedding as it is and for the words not in the corpus, we generate a random Gaussian distribution of the same hidden as other words, with a weight scale of 0.6. Furthermore, we use data belonging to 20 categories and map these 20 labels to 7 major categories. The statistics about the data are highlighted in Table 1.

	Number of Samples
Train	11,314
Test	7,532

Table 1. Split of the sampled data 20newsgroup dataset

2.2. Random Models Generator

The novelty of this work is in using multi random deep learning models including DNN, LSTM, and CNN techniques for text classification. After extracting the embeddings for the text data, our next step is to generate multiple random models of Deep Neural Networks, CNN and LSTM in parallel. The number of layers and nodes for all of these deep learning multi models are generated randomly. The final model contains 1-2 random models of each architecture. Table 2 shows different parameters used across each architecture. We describe each individual model separately in the architecture document.

Model	**Hidden Layers**	**Hidden Nodes**
DNN	(1, 7)	(128, 600)
CNN	(3, 5)	(128, 256)
LSTM	(3, 8)	(32, 128)

Table 2. Random parameters across different architecture

2.3. Random Models Training

We create 2 random models for each architecture. We pass the text embeddings to different models as input. Every model is trained with different learning rates and upto different iterations. We train each model till the validation loss starts to saturate. For DNN, we varied the hyperparameters like dropout rate, regularization, batchsize and learning rate to reach a maximum accuracy of 69.3% on the test set. Similarly for CNN, we got a maximum accuracy of 75.3% and for LSTM, the accuracy went upto 78.1%. We build 2 final models - one consisting of best of 3 architectures and then fed it to the attention network. The other final model consisted of 2 best models of 3 architectures and then their outputs are connected to the attention network.a

2.4. Attention Based Learning

Our plan included to ensemble different methods for training our classification task and see how accuracy is affected. Instead of traditional methods of ensembling, that is to use the most occurring results out of the different models to predict the final output we planned on using Attention Based Ensembling.

Our aim basically was, to use attention layers to learn how to give different weights to each network for a particular data point. The assumption we made was that for different data points different types of models work better. Through our results we aim to corroborate this assumption.

Once we had our random models generated and trained, we used these models to learn the attention weights. Instead of training the entire model again, we freezed the weights to just train the attention layer using the outputs of each model. The architecture for the attention is given in the Architecture document.

3. Results

The section shows a comparison between the performance of each of the individual DNN, CNN and LSTM models with the 2 attention models trained using an ensemble of DNN, CNN and LSTM. All the models were trained using Adam optimizer and Crossentropy loss function.

The results comprises of 4 parts:

1. Training random DNN's: We created 2 random DNN architectures using our random generator. We kept the number of hidden layers between 1 and 7, and the number of nodes in each layer between 128 to 600. The networks were trained with learning rate = 1e-5, number of epochs 70 and batch size 512. The results of the DNN are summarized in Figure 1. The accuracy results on test dataset were 65.1% and 69.3% respectively.

2. Training random CNN's: The generic random generator is used to create 2 CNN architectures. The number of hidden layers is between 3 and 5, and the number of nodes in each layer between 128 to 256. The first network was trained with learning rate = 1e-5, number of epochs 110, and batch size 512. The second network was trained with a was trained with learning rate = 0.01, number of epochs 16, and batch size 512. The results of the CNN are summarized in Figure 2. The accuracy results on test dataset were 75.3% and 67.97% respectively.

3. Training random LSTM's: We created 2 random LSTM architectures using our random generator. We kept the number of hidden layers between 3 and 8, and the number of nodes in each layer between 32 to 128. The networks were trained with learning rate = 0.001, number of epochs 15 and batch size 512. The results of the LSTM are summarized in Figure 3. The accuracy results on test dataset were 72.2% and 78.1% respectively.

4. Training the attention layer: As discussed in the architecture the weights of all the networks are freezed before training the attentions to make sure on the weights given to the models are trained. We trained the attention layers using 2 architectures. One involving 1 CNN, 1 DNN, 1 LSTM and the other involving 2 CNNs, 2 DNNs and 2 LSTMs The attentions were trained for 60 and 70 epochs respectively with learning rate starting from 0.0001 and dropped to 1e-8 using annealing. The results of training the attentions are summarized in Figure 4. The accuracy results on test dataset were 82.8% and 83.4% respectively.

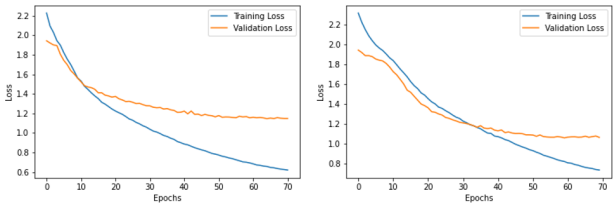

Figure 1. Result: DNN Individual Networks

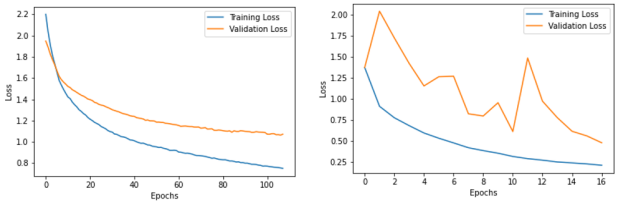

Figure 2. Result: CNN Individual Networks

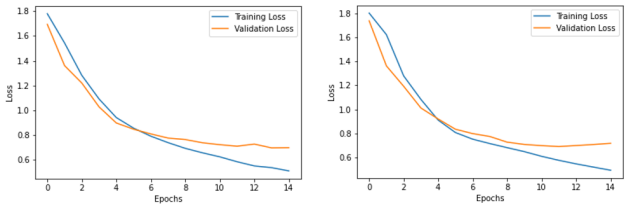

Figure 3. Result: LSTM Individual Networks

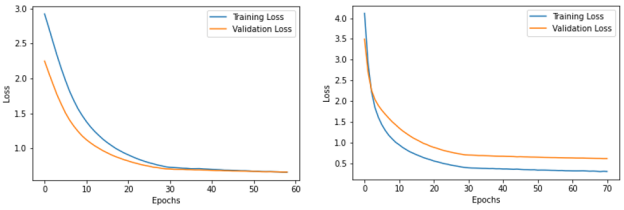

Figure 4. Result: Attention Individual Networks

4. Discussion

The classification task is an important problem to address in machine learning, given the growing number and size of datasets that need sophisticated classification. In this section, we discuss the experimental results including evaluation of method, experimental setup, and datasets.

Firstly, we can see both the DNN models perform very similar, resulting in a gradual decrease of validation loss as we train them for more epochs. Thus, the accuracies of both the DNN are very close i.e. 69.3% and 65.1%. The difference in the accuracies is due to different parameters used in the model architecture. We observe the same trend with both the LSTM models as well. Both the LSTM models fetch similar accuracies 78.1% and 72.2%. For CNN, we see a huge difference in the graphs. We used 2 contrasting learning rate to distinguish the model performance. As we can see, the first graph has a gradual decrease in validation loss with a lower learning rate but it takes 110 epochs. On the other hand, the second plot has a zig-zag validation curve with a higher learning rate but it attains a global minimum validation loss in lesser(16) epochs.

Secondly, we can see that LSTM outperforms CNN and DNN models. The reason behind this is that LSTM captures the dependency across different embeddings in the input vector. The same effect may be accomplished with DNN or CNN but that would require collecting the input vector across time and then feeding it to a large layer, resulting in a larger set of parameters to train compared to LSTM. So, we see empirically that LSTM attains better performance with lesser epochs.

Finally, we can see that as we train Attentions after freezing the weights of all the layers using our architecture the accuracy increased to a great extend. Thus, corroborating the assumption that for different data points different types of models work better. Since, all the attention layer is doing is giving the weights to different models. If attention were to pick the best model then the achieved accuracy should have been the best of the accuracies of all the models. However, this is not the case as per seen by the results.

In future we would like to expand our approaches to different datasets for classification tasks including images.

References

[1] 20 newsgroups dataset, 1999-09-09. 1

[2] T. Chen and C. Guestrin. Xgboost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug 2016. 1

[3] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014. 1

[4] A. Galassi, M. Lippi, and P. Torroni. Attention in natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32(10):4291–4308, 2021. 1

[5] T. K. Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1, 1995. 1

[6] K. Kowsari, D. E. Brown, M. Heidarysafa, K. Jafari Meimandi, M. S. Gerber, and L. E. Barnes. Hdltex: Hierarchical deep learning for text classification. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 364–371, 2017. 1

[7] K. Kowsari, M. Heidarysafa, D. E. Brown, K. J. Meimandi, and L. E. Barnes. Rmdl: Random multimodel deep learning for classification. In *Proceedings of the 2nd International Conference on Information System and Data Mining*, ICISDM '18, page 19–28, New York, NY, USA, 2018. Association for Computing Machinery. 1

[8] H. Lee, R. Grosse, R. Ranganath, and A. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. page 77, 01 20a09. 1

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1