Modular Deep Encoder-Decoders

Sebastian Arnold[1], Prashanth Gurunath Shivakumar[2], Cheng Qu[3] and Qiangui Huang[4]

arnolds@usc.edu[1]	pgurunat@usc.edu[2]	chengqu@usc.edu[3]	qianguih@usc.edu[4]
9013085897	2251924199	2385279985	9532576000

Abstract

In this short paper, we propose a Modular Deep Encoder-Decoder network (MDED) for text classification. The basic idea behind MDED is to extend a pre-trained model, which consists of a set of deep encoder-decoders, with an additional source of data modality by adding a new encoder to the network. MDED has been applied to text classification on Saudi Newspapers Arabic Corpus (SaudiNewsNet). A modular Long Short Term Memory (LSTM) network is built. Experimental results show that our algorithm can efficiently combine new source of data and pre-trained model. The training process can converge into a local optimum in very few epochs when a new data source is added. With a good initial encoder-decoder, the modular LSTM networks can improve classification accuracy by up to 0.51% when a new data source is added.

I Introduction

In the past decade, deep learning has gained a lot of momentum in various fields. It has been successfully applied to natural language processing (NLP). End-to-end trained deep learning algorithms have been proven to outperform traditional algorithms on various NLP tasks. The success of deep learning is accounted for trading off computationally complexity for performance. The high complexity demand for training deep learning algorithms is inefficient especially when there is new complementary data modality to be used to update a model and re-train it from scratch every time. To alleviate this problem, a novel Modular Deep Encoder-Decoder network (MDED) is proposed in this paper. The basic idea behind MDED is to extend a pre-trained model, which consists of a set of deep encoder-decoders, with an additional source of data modality by adding a new encoder to the network.

More formally, we want to jointly learn a set of *encoder* functions $\{E_i\}_0^N$ mapping samples $x \sim \chi_i$ from a set of data distributions $\{\chi_i\}_0^N$ to a fixed-sized vector embedding V.

$$\forall i \in [0; N] : E_i(x) : \chi_i \to V_i \in \Re^M$$

$$V = \sum_i^N V_i$$

The embedding V is then fed into a decoder function D which in the case of classification learns a mapping from the vector space of V to the label space L.

$$D(v) : \Re^M \rightarrow L \in \Re^D$$

Finally, we want to extend the set of encoders by learning a new encoder E_{N+1} which handles samples from a different dataset χ_{N+1}, without retraining our trained decoders and encoders.

$$E_{N+1}(x) : \chi_{N+1} \rightarrow V_{N+1} \in \Re^M$$

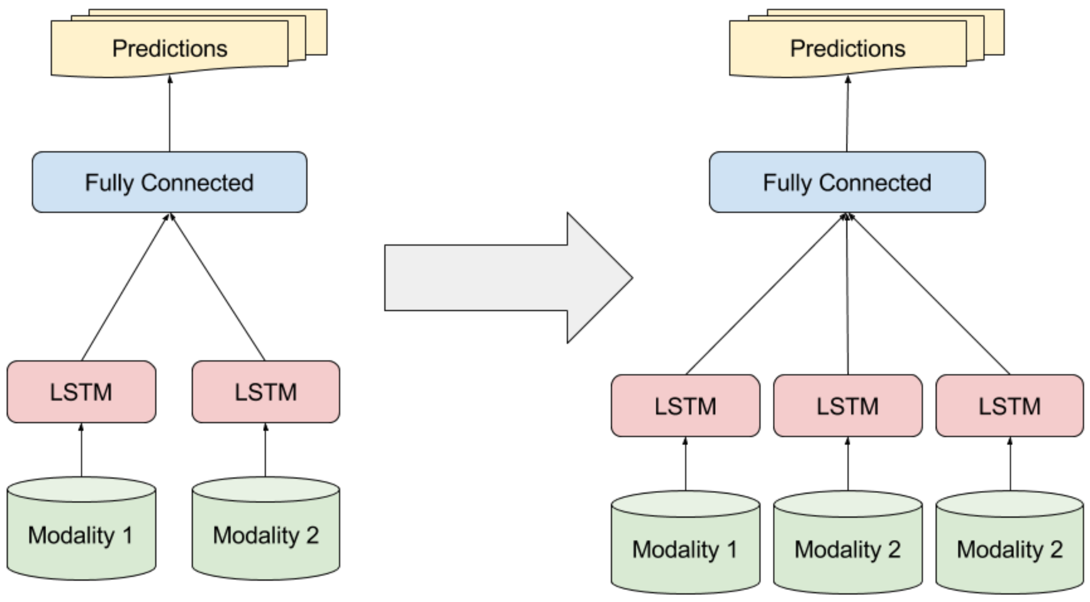


Figure 1: Multi-modal Transfer Learning

Although we could have used any kind of mapping, we chose to use deep learning algorithms, as they easily learn hierarchical representations and have been known to highly outperform other statistical techniques on natural language tasks. Learning embeddings has been previously done by [1], although our proposed contribution is slightly different. [1] used deep autoencoders to obtain a better initialization of the parameters of their model. Our approach instead is much closer in spirit to the work of Sutskever [2] and Vinyals [3]. They both use encoders on a sequence of data to generate a *thought-vector* which will then be decoded in the desired target sequence. In some sense, our proposal adds the transfer learning component to their contribution. This approach is also similar to the work of Karpathy & al [4] where they mapped images to their captions with embeddings.

In this paper, we evaluate the effectiveness of idea by applying the concept towards a simple NLP classification task. We adopt Saudi Newspapers Arabic Corpus [5] to classify text into different journals based on the article content, author and title.

II Method

Materials

Corpora

We use the freely available Saudi Newspapers Arabic Corpus [5]. The dataset contains a total of 31,030 Arabic newspaper articles, with a total number of 8,758,976 words.

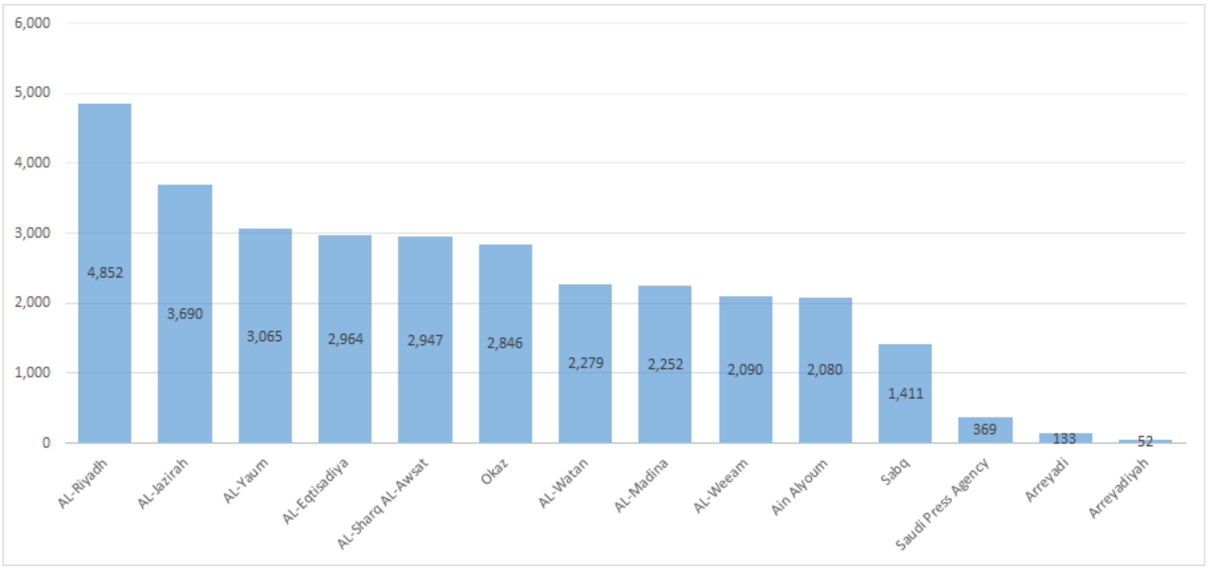


Figure 2: Figure shows article-newspaper distribution

Article objects are represented in json format, with UTF-8 encoding. We wrote a script to download the data from github repo, unzip each file, and read them in as json objects. Each json object contains the following fields:

- source: A string identifier of the newspaper from which the article was extracted.
- url: The full URL from which the article was extracted.
- date_extracted: The timestamp of the date on which the article was extracted. It has the format YYYY-MM-DD hh:mm:ss.
- title: The title of the article.
- author: The author of the article.
- content: The content of the article.

The dataset is partitioned into training, development and testing subsets. 80% of the data is assigned for training and 10% for development and testing. The development set is utilized for hyper-parameter tuning and the results are finally presented on the testing set. During partitioning, care is taken to ensure appropriate percentage of data is seen in all the classes. Figure 3 shows the distribution of data used for training and testing in each class. Figure 4 shows the distribution of sentence lengths of the content of the articles.

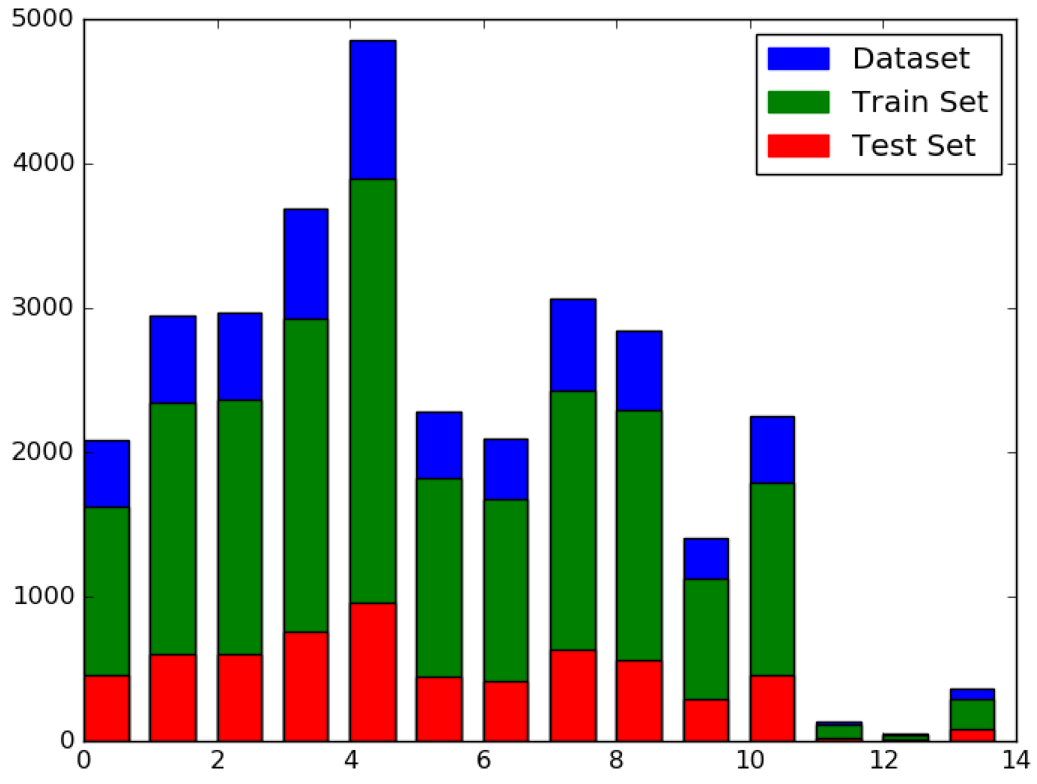


Figure 3: Distribution of data for training and testing

Tools

For training the neural networks, Neon toolkit was employed [6]. Neon is Nervana's python based deep-learning library. The toolkit was chosen because of its flexibility of distributed training on both CPU and GPU and its good support for neural network components.

Procedure

Data Pre-processing and Features

Firstly, the UTF-8 encoded data was checked for trailing spaces for removal. Next, the data was filtered by removing punctuation symbols, retaining only a subset consisting of {, . ! ( ) ?}. The data was then split into words which are used as tokens for building the vocabulary and finally obtaining a feature representation for training the neural network. A vocabulary is constructed by assigning unique integer word-ids to each unique word appearing in the corpora. The punctuations are treated as a separate token and contributes to the vocabulary. The url, date and title fields of the corpora were discarded since they were irrelevant for our purpose.

After building the vocabulary for the corpora, the original sentences of the corpora

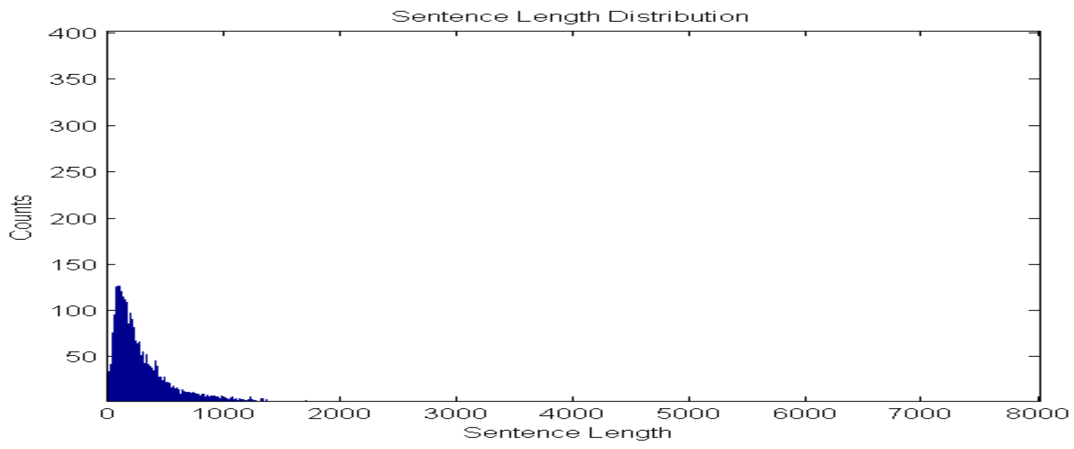


Figure 4: Distribution of data for training and testing

are mapped on a word-basis to obtain a series of integers representing the sentence equivalent which form the features to be input to the embedding layer of the neural network. Alternatively, the integer mapping could be replaced with one-hot sparse encoding of each word constituting the sentence. The features are shuffled and split for training and testing. For efficient data storage and retrieval the features are compressed to HDF5 file format.

Embedding Layer

An embedding layer is used to compute a word embedding $W : words \mapsto \mathbb{R}^n$ which is a function mapping words of a language to a high dimensional vectors which are continuous, distributed representation of the vocabulary words with good semantic properties. The hope of such a representation is to project similar words to similar regions in the hyperspace from a semantic point of view.

We tried two approaches when building the vector embedding from multiple modalities. The first one was to vertically stack the different embeddings from the different encoders and learn the actual embedding from a linear combination from this higher dimensional vector. This led to poor results and we instead decided to experiment with both the summing over every embedding, as well as taking their mean in each dimension.

Baseline System

We construct two baseline systems, one to classify the journals based on the content of the articles and the second on their authors. Identical architectures are used for both the baseline systems. The baseline system is a recurrent neural network with one hidden Long Short Term Memory (LSTM) layer succeeding the embedding layer. The dimension of both the embedding layer and the LSTM was fixed to 128. The output layer is a softmax layer with 14 outputs, one per class. The model minimize a cross-entropy loss using adaptive gradient descent. [7]

Proposed System

The training for evaluation is performed as follows. We begin by training a single encoder and the decoder on a single modality. (eg, the content of the articles) We measure the obtained accuracy on the test set and serialize the model's parameters on disk. In the second step, we recreate the previously trained encoder and decoder, and initialize them with their respective serialized parameters. We also instantiate a new encoder architecturally identical to the first one but with untrained parameters. Finally, we load both the previous and the new modality from the dataset and their corresponding labels, before proceeding to the training steps as described in the previous sections.

Hyper-parameter Tuning

Using the development set, the hyper-parameters were fine-tuned for better performance. The tuning of sentence length cut-off for training the embedding layer was found to be critical especially when training on the content of articles. Setting lower sentence length resulted in over-training of the system. The distribution of the sentence lengths were plotted as shown in Figure 4. Looking at the distribution, we found the value of 2048 to be optimal. Different values for learning rate, hidden-layer dimensionality and non-linearity were experimented. Through a series of experiments, we found the optimal setting for initial learning rate is 0.01, optimal hidden-layer dimensionality is 128, and tangent function is the optimal activation function in hidden layers.

Evaluation

The evaluation procedure for all experiments is based on the classification accuracy of each statistical model on a held-out test data. Meanwhile, we identify that the accuracy is not an ideal evaluation criteria considering the skewed nature of class distribution. To get a more meaningful representation of the performance F1-score or AUC metrics will be provided in the future work. However we stress the validity of the accuracies provided in the results section by comparing it to the random baseline. Taking into account the multi-class (14 class problem) nature of the classification and the prior for each class, we would like to point out that the average accuracy or chance accuracy of the data is 9.68% which forms the random baseline of the problem under investigation.

III Results

Baseline System

First, three single LSTM networks are trained for text classification, where title, author and content values are used for training data, respectively. The hyper-parameter setting is same as introduced in above section. The batchsize is 32 and training ends after 10 epochs.

The classification performance of these models are reported in Table.1. From Table.1 we can see that author model has the best classification performance (73.89%) among these three models. And title model has the worst performance (21.23%) although its

Table 1: Classification performance of single LSTM network

Training data	Title	Content	Author
Training accuracy (%)	92.93	96.31	85.41
Testing accuracy (%)	21.23	43.68	**73.89**

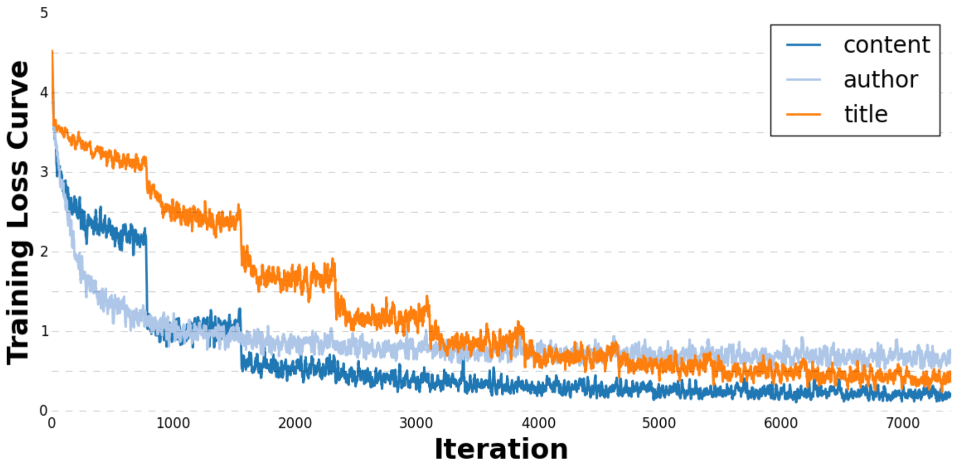


Figure 5: Training loss and testing loss of single LSTM networks.

training accuracy achieves 92.93%, which means training using only the title of articles will result in serious overfitting.

Moreover, the training behavior of these three models are presented in Fig.5.

Modular LSTM network

In order to further improve the classification performance, new sources of training data are added to the base single LSTM network. For each single LSTM network, one new data source is added to it, respectively. For example, for content LSTM network, author of the articles and title of the articles are added to it as an additional source of training data. Technically, a new encoder is added to the original LSTM network. The training parameters are same as above. Performance of these modular LSTM networks are reported in Table.2.

Experimental results show that our modular LSTM networks can effectively exploit new data source in most of the cases. For the author model, which is trained using only author of articles, the classification accuracy has been improved by 0.57%. One interesting finding is that after adding a new data source to content model, the classification is decreased by 0.34% and 1.31%. We can also find that the content-author model [1] has a much better performance than author-content model. That's because the single author model has a better performance than the single content model, which demonstrates that a good initial model is very crucial in modular LSTM networks.

[1]author is the based data source and content is the added data source

Table 2: Classification performance of modular LSTM networks

Base source	Title		Content		Author	
New source	Content	Author	Title	Author	Content	Title
Training accuracy (%)	95.59	95.51	97.12	97.13	86.48	85.79
Testing accuracy (%)	21.69	21.64	42.37	43.34	**74.46**	74.04
Improvement (%)	0.46	0.41	-1.31	-0.34	**0.57**	0.15

Moreover, the training behavior of them are presented in Fig.6. In Fig.6, the training loss of initial single LSTM and modular LSTm networks are compared with each other. From Fig.6 we can see that, in all three cases, after a new data source is added, the training loss can converge to a local optimal very fast and the loss values accordingly decrease rapidly. Once a new data source is added, the whole training process can finish in less than 1,000 iterations (near 2 epochs). This demonstrates the efficiency of our MDED network.

IV Discussion

The results of the baseline system for each modality by themselves show significant improvement over the random baseline. We asses that the author modality to be more informative for classification as expected due to low overlap of author names across journals.

In future, more informative evaluation metrics in terms of F1-score and AUC will be provided for better reflection of performance of each system. We plan to also analyze the per class performance. We expect classes with less samples to be less accurate during classification because of data sparsity, which could be solved by discarding the classes with low sample counts thereby increasing the overall accuracy of the system.

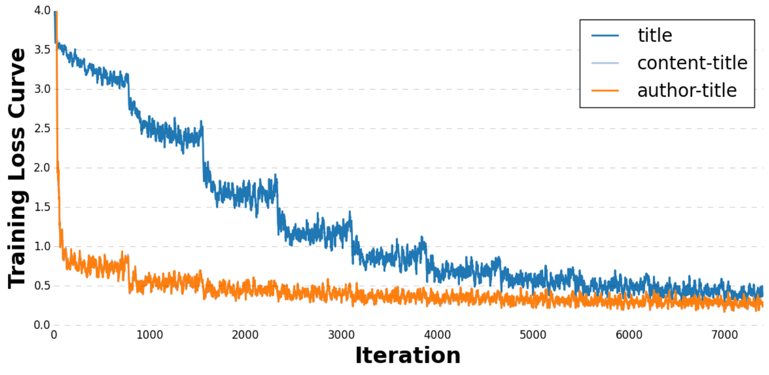
In future, we would like to apply the concept to other databases and pose the concept to solve inter-database learning. We could also prove the concept in other areas like speech, image and other signal processing domains.

References

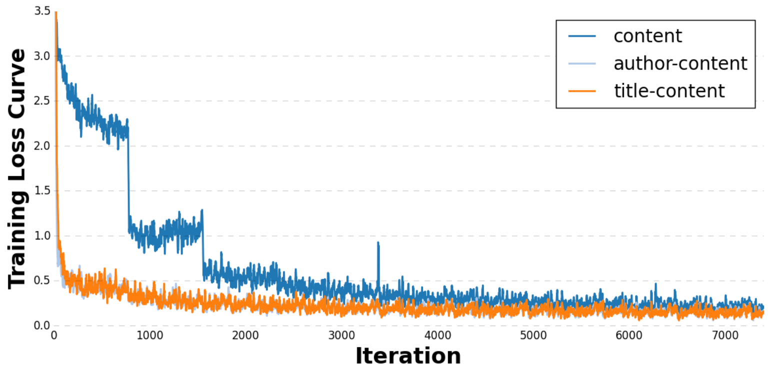
[1] F. Zhuang, X. Cheng, P. Luo, S. J. Pan, and Q. He, "Supervised representation learning: transfer learning with deep autoencoders," in *Proceedings of the 24th International Conference on Artificial Intelligence*, pp. 4119–4125, AAAI Press, 2015.

[2] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, pp. 3104–3112, 2014.

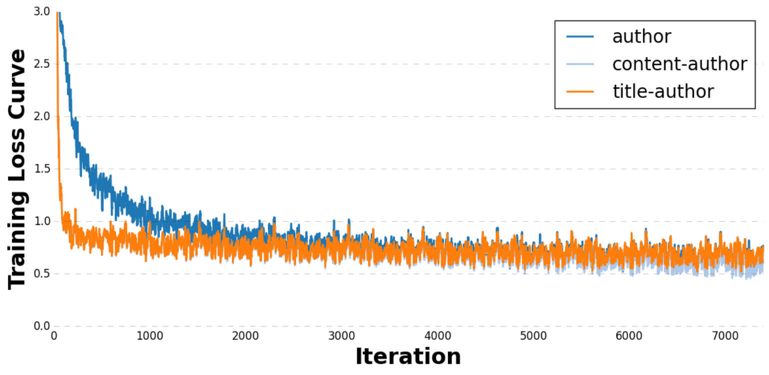
[3] O. Vinyals, Ł. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton, "Grammar as a foreign language," in *Advances in Neural Information Processing Systems*, pp. 2755–2763, 2015.



(a)



(b)



(c)

Figure 6: Training loss and testing loss of modular LSTM networks: (a) title model based modular LSTM; (b) content model based modular LSTM; (c) author based modular LSTM. In the legends content-title author-title, and etc., the previous one denotes base data source and latter one denotes added data source.

[4] A. Karpathy, A. Joulin, and F. F. F. Li, "Deep fragment embeddings for bidirectional image sentence mapping," in *Advances in neural information processing systems*, pp. 1889–1897, 2014.

[5] M. Alhagri, "Saudi newspapers arabic corpus (saudinewsnet)," 2015.

[6] "Neon by nervana systems." `https://github.com/NervanaSystems/neon`.

[7] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *The Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

V Labor Division

- **Cheng: Coding** - Data retrieval, unzipping, reading, unicode decoding; **Report Writing** - Corpora, Evaluation.
- **Quiangui: Coding** - Core model and augmented model training, Data partitioning, Word2vec model, Training log parsing, Hyper-parameter tuning, Model training, Makefiles. **Report Writing** - Abstract, Introduction, Hyper-parameter tuning, Results.
- **Prashanth: Coding** - Data pre-processing, Data partitioning, Feature representation for embedding layer, Baseline system; **Report Writing** - Data pre-processing and features, Embedding Layer, Baseline System, Hyper-parameter tuning, Evaluation, Discussion.
- **Sebastian: Coding** - Neural Network training, Makefiles; **Report Writing** - Abstract, Introduction, Proposed System, Discussion.

VI Word Count

- 109 Abstract
- 392 Introduction
- 254 Materials
- 593 Procedure
- 126 Evaluation
- 60 Results: Baseline System
- 92 Results: Modular LSTM network
- 146 Discussion
- 1772 Total