Sandwiching Transferred Models of Sparse Vectors in Deep Learning

Pranav A
Hong Kong University of Science and Technology
Hong Kong

ABSTRACT

The task assigned was to predict the trading action that the user is likely to read according to app contents about users by using transfer learning. While implementing deep learning architectures, we observed the non-convergence of common loss functions. Hence we proposed masked loss function which could mask the sparsity information in the loss function. Further, we proposed an autoencoder based model by sandwiching two distinctly trained networks together. We then introduced a novel architecture, bi-directional autoencoders which produces two outputs for a single input vectors. The backward pass of the network based on the alternating gradient optimization methods. Finally, through our experiments, bi-directional autoencoders outperforms the rest of the given models.

KEYWORDS

transfer learning, masked loss,autoencoders, backward propagation,sandwiching

ACM Reference Format:
Pranav A. 2018. Sandwiching Transferred Models of Sparse Vectors in Deep Learning. In *Proceedings of Research Project*. ACM, New York, NY, USA, 5 pages. https://doi.org/0.00/0000

1 INTRODUCTION

Transfer learning involves of learning of patterns from different domains. It is a branch in machine learning which focuses on using a large enough dataset to train a model and then apply the model to a new and relevant model instead of learning from zero. Deep learning deals with data abstraction based on very complex structure and non-linear conversion whose structure is analogous to the human neural system. In this project, we apply transfer learning with deep learning to predict the articles users may like to read by given some informations about users.

We encoded the vectors as App, L1 and L2. Each index of the App vector corresponds to fraction of the Apps of a particular category. Each index of the L1 and L2 vectors correspond to average weight of the particular article category. L1 consists of coarse level article categories and L2 comprises finer article categories. It may be noted that L2 is quite sparse compared to the L1 vector. Typically, App acts as the input vector while L1 and L2 vectors acts as the outputs.

Research Project, December 2017, HKUST, Hong Kong

ACM ISBN 123-4567-24-567/08/06.
https://doi.org/0.00/0000

Training using sparse vectors is quite challenging and numerically unstable. During the training of our networks, we found that commonly used loss functions were not converging at all. Thus, we came up with our masked loss function which masks the sparsity of the vectors and focuses on the non-zero indices. This has been discussed in detail in the section 2 of this report.

Section 3 discusses the basics and implementation design of the baseline transfer learning based models. Using the advantage of dimensionality differences of the vectors, we proposed first sandwiching technique which is inspired from the architecture of the autoencoders, explained in section 4.

Dealing with sparsity was still a struggle since we were unable to filter out relevant articles for the particular users. Hence, we proposed more robust version of club-sandwich, which gives two output vectors for a single input vectors. The output vectors work in an adversarial fashion which are optimized using alternating gradients. Section 5 lays down detailed architecture for this notion. To the best of our knowledge, we are the first to propose these architectures in the deep learning field. Further sections of this report explore the analysis and potential of these proposed models.

2 MASKED LOSS

As noted before, the L2 vector is quite sparse and difficult to learn in the deep learning based architectures. Through our experimentations we found that most of the commonly used loss functions were not converging at all. Thus, we tweaked the Mean Squared Error loss a bit by multiplying the squared distance with the sparse vector. If the predicted output is y and the actual output is y', then the masked loss is given by,

$$\sum_{i=1}^{n} y'_i(y_i - y'_i)^2$$

The backward pass would only have the difference of additional y' in their gradients. Typically, it would look something like:

$$\frac{\partial l}{\partial w} = 2y'(y - y')\sigma'(h)\frac{\partial h}{\partial w}$$

where $\sigma(\cdot)$ is the activation function, h is the presumed hypothesis (like the matrix product), l is the loss function and w is the weight to be updated.

The figure 1 shows convergence behaviors of MSE loss against the Masked Loss function. We see that Masked loss based backprop is able to converge but MSE loss gets stuck too early.

3 BASELINE

In this section, we explain how we would implement the baseline of transfer learning model. Most transfer learning with deep learning

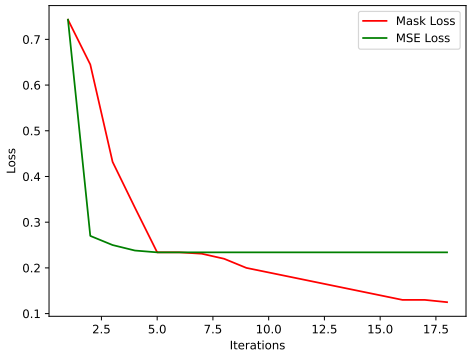

Figure 1: This figure shows the comparison between Mask Loss and MSE Loss. We can see that the green line which represents the MSE Loss doesn't go down all the time while the red line which represents Mask Loss is always reduce the loss.

often use pre-training. This pre-trained model is reused and a few modifications on layers are done to train on the targeted dataset.

For example, in image classification, practitioners often download pre-trained models from ImageNet datasets. After downloading the model, they may change the architecture of the model by introducing some layers or freezing some layers. This idea of pre-training was coined by Bengio et al [1]. In this project, we would implement a basic version of this transfer learning technique as the baseline.

We have the App, L1 and L2 vectors as described in the previous sections. First we would train mappings from App to L1 vectors profusely. After this network has been trained, we would add another layer on top of this network. This network learns mappings from the outputs of previous network (L1) to L2 vectors. Thus, training this network would correspond to the learning mappings from L1 and L2 and finetune the original weights. The figure illustrated below shows how the mappings from the input (App) vectors to output (L2) vectors using this elementary baseline transfer learning principle.

4 AUTOENCODERS

Our first proposed model is based on the notion of Autoencoders. In the layman terms, the autoencoders comprises an encoder and a decoder. Here, we have trained encoder and decoder modules independently. After an exhaustive training, we join them together by "sandwiching" supplementary weights between encoder and the decoder modules.

Firstly, we learn the mappings from App to L1 vectors as similar to the baseline model section. This is analogous to the encoder part, because the dimension of L1 vector is smaller than App vector, or L1 vector "encodes" the information about App vector.

Simultaneously, we also learn the mappings from L1 to L2 vectors independently. This is analogous to the decoder part, because the dimension of L2 vector is smaller than L1 vector, or L2 vector "decodes" the embeddings about L1 vector.

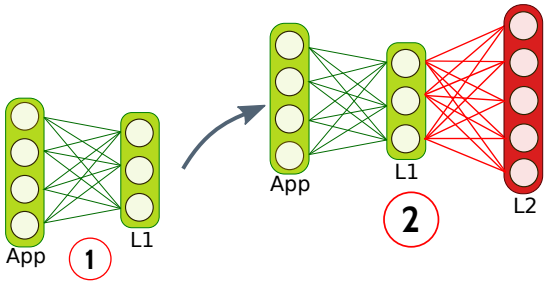

Figure 2: This figure illustrates the baseline transfer learning procedure. First we train the network from App to L1 vectors (illustrated by the green network). Then we add another layer (illustrated by red networks) on top of it and finetune weights on L2 dataset.

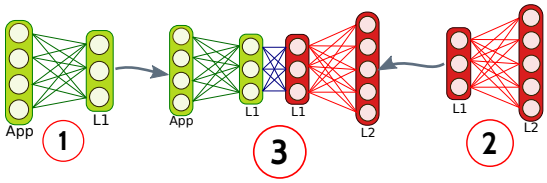

Figure 3: This figure illustrates the process of autoencoders in transfer learning. The green network corresponds to the encoder part which is training from App to L1 vectors. The red network corresponds to the decoder part which is training from L1 to L2 vectors.The third step illustrates sandwiching encoder and decoder parts together. The sandwich layers are illustrated by blue neurons. Finally we finetune these weights by training from App to L2 vectors.

Once we have the trained encoders and decoders, we will connect them together by sandwiching additional neurons between them. Finally we would finetune these weights by training from App to L2 dataset.

An advantage of using this model over baseline is that we learn robust embeddings of L2 in L1 through the decoding process.

5 BI-DIRECTIONAL AUTOENCODERS

In this section, we would explain about our second proposed model, which we name it as "Bi-Directional Autoencoders".

In the autoencoders, we used L1 as the hidden layer and L2 as the output layer for the finetuning part. However, L2 vector is too large and sparse to learn as compared to the L1 vector. Therefore, just relying on L2 vectors would be unreasonable. This model makes use of both L1 and L2 as the outputs with only App as the inputs.

Recalling from the previous section, we train encoders (App to L1) and decoders (L1 to L2) independently. However, learning mappings from L1 to L2 is just embedding information. Thus, we modify the decoder part here with learning mappings from App to L2, rather than L1 to L2.

The following figure illustrates the architecture of this model. Here the sandwiched layer is the dotted blue lines, which are loosely

connected which means the sandwiching layer would only operate in the backward pass, not in the forward pass.

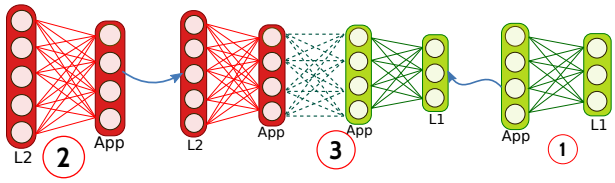

Figure 4: This figure illustrates the process of Bi-Directional Autoencoders. The red network corresponds to the mappings trained from App to L1 vectors. The green network exemplifies the training from App to L2 vectors. The third step represents combination of those two networks and the sandwich layers are illustrated by blue dotted line, which are loosely connected neurons.

Once this model is adequately defined, we would now explain how forward pass would work as follows. In the figure 5, one can see that we provide the inputs in the middle of the networks, which then produces the outputs of L1 and L2 vectors respectively. This is quite an unorthodox but effective model because using same input, we would acquire two independent vectors.

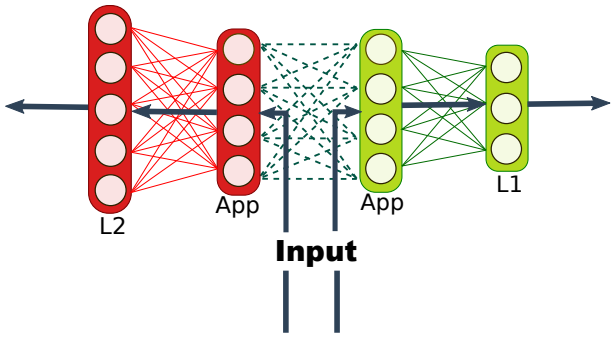

Figure 5: This figure illustrates the forward pass of Bi-Directional. we use the same App vectors as input for both green network which is encoder part and red networks which is decoder part. Then we gain L1 vectors and L2 vectors as different and independent outputs.

It could be seen that backward pass would work here in a different manner, somewhat analogous to backprop algorithm of Bi-RNNs. We use the notion of alternating gradients here to backpropagate one by one.

Firstly, we will backpropagate the gradients obtained in decoder part and update the weights of decoder module. Simultaneously, we will backpropagate the gradients obtained in encoder part and update the weights of encoder module. Once, encoder and decoder parts are updated, we would enable the connecting networks. The weights of the connecting networks would be updated by adding gradients obtained with respect to decoders, then moving to the encoders. Now weights in encoders would be updated with the gradients of decoders. Simultaneously, weights in decoders would be updated with the gradients of encoders. This backward pass is explicated more clearly explicated in the algorithm 1 and figure 6.

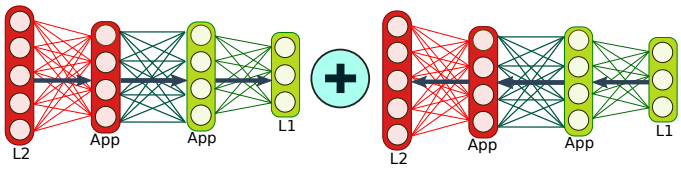

Figure 6: This figure illustrates the process of Bi-Directional back propagation which we use the gradient descent to update weights. Firstly, we back prop from L1 to App vectors(encoders) and L2 to App vectors(decoders) respectively to update the weights. Then we update the weights of sandwich layers using the gradients from both directions. Finally, we back prop from encoder part to decoder part to update the weights of decoder part, and in opposite, we back prop from decoder part to encoder part to update weights of encoders.

Algorithm 1 Alternating Gradient Method for Backpropagation

$w_e \in$ Neurons in Encoders
$w_d \in$ Neurons in Decoders
$w_c \in$ Neurons in Connectors
l_d = Loss function towards the decoder part
l_e = Loss function towards the encoder part
α_d = Learning rate towards the decoder part
α_e = Learning rate towards the encoder part
for every backward pass step **do**

$$w_e \leftarrow w_e - \alpha_e \frac{\partial l_e}{\partial w_e}$$
$$w_d \leftarrow w_d - \alpha_d \frac{\partial l_d}{\partial w_d}$$
$$w_c \leftarrow w_c - \alpha_d \frac{\partial l_d}{\partial w_e} - \alpha_e \frac{\partial l_e}{\partial w_c}$$
$$w_d \leftarrow w_d - \alpha_e \frac{\partial l_e}{\partial w_d}$$
$$w_e \leftarrow w_e - \alpha_d \frac{\partial l_d}{\partial w_e}$$

Here, we would use L1 vectors as the filter followed by the selection using L2 vectors. More information is given in the experimental setup section about this.

6 EXPERIMENTAL SETUP

6.1 Preprocessing

The obvious first step was to clean and pre-process the dataset. Among many kinds of users' information, we think that the most important features which contain the App using frequency, coarse categories (which is L1 categories in article) and finer categories (which is L2 categories) which are represented by App vectors, L1 vectors and L2 vectors respectively. After we remove the redundant data, we detect the dimensions of useful informations and find that App using frequency has 52 dimension, L1 article categories has 23 dimensions and L2 categories has 164 dimensions. Some users had more than one item but each item includes different informations. Thus, we need to combine them into one feature vector. As a result,

we create an App vector, l1 vector and l2 vector for each user and store them in corresponding files. After the data cleaning, we reduce the database from more than 170 thousand items to 7857 items. Here, 6000 were used for training, 1000 for validation and 857 for testing purposes.

6.2 Baseline

After preprocessing the data, we constructed the baseline model of transfer learning. Pytorch was used for implementing this framework. Firstly, we created a two-layer neural network to train App vector to L1 vector with 1024 neurons in each layer. Adam is used as optimization function and ReLU is the activation function. Once the network is trained, we transferred it with adding a new layer to predict L2 vector directly. Therefore, for this baseline model, the input is App vectors and output is L2 vectors.

Also, to avoid overfitting problem, we carry out a grid search on dropout values. We find that the best dropout values for each layers are 0.05, 0.1, 0.0. In addition, we augmented L2 regularization in case of overfitting to the given loss function.

6.3 Autoencoders

Our first proposed model is based on combining transfer learning with autoencoders in unsupervised learning and finetune the weights of this whole model. From App to L1, the model is similar to the baseline model which contains two hidden layers and each layer has 1024 neurons. We apply ReLU as activation function and our Masked loss function because it converges well. In addition, the dropout values are from the baseline grid search and optimization function used is Adam. But from L1 to L2, we use no hidden layers. Because through our experiments, we find the number of layers was making no difference in results. . Then we transfer the two learned networks and sandwich a layer between them and finetune the weights of the whole model. After that, we can combine them into one model whose input is App vector and output is just L2 vector.

6.4 Bi-directional autoencoders

For the bi-directional autoencoders, we trained the mappings from App to L1 vectors and App to L2 vectors. We used 2 hidden layers with 1024 neurons in each layers for App to L1 training. We used 4 hidden layers with 1024 neurons in each layers for App to L2 training. ReLU, Masked Loss, L2 regularization and Adam were used. Alternating gradient method was used to backpropagate the errors.

Once, the network was trained, we filtered top 2000 articles using the closest distance from the L1 vectors. After choosing the top 2000 articles for each user, then we used top 5 recommendations using the information gained from the L2 vectors.

7 RESULTS

The results are reported in the following table. Clearly Bi-directional Autoencoders outperforms better than rest of them due to the filtering technique. MSE Loss based functions were not able to perform well as they got stuck in optimum too soon. Underperformance of Autoencoders and Baseline is due to the fact that just L2 outputs are not enough for the prediction. We further analyze the learning curves of the Bi-directional Autoencoders model. We find no evidence of overfitting or underfitting here. Overfitting is not observed due to our several efforts to avoid it and also transfer learning models act as a regularizer.

Table 1: Test Results

	Category Accuracy	Blind Precision Results
without Transfer Learning + MSE Loss	18%	-
Baseline + MSE Loss	20%	-
Baseline + Masked Loss	22%	-
Autoencoders + Masked Loss	35%	-
Bi-directional Autoencoders + Masked Loss	**40%**	-

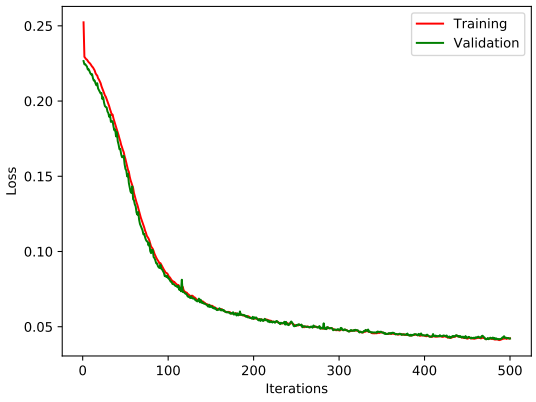

Figure 7: This figure illustrates the change of loss values along with iterations.The red line represents values of training while the green line represents that of validation. From this chart, we can see that two types of values converge very well as the iteration goes by.

8 RELATED WORK

The idea of pre-training and reusing it for the training in the targeted datasets was originally coined in the Bengio's work [1], whose approximation has been implemented in this paper as the baseline. The idea of the masked loss has been partly inspired the truncated gradient for the sparse vectors [2], which has been often found as numerically unstable within the usage of deep learning frameworks. Although, not implemented by the authors, but they believed that BiGANs could be used for the transfer learning [3]. At the starting of this project, we proposed that BiGANs could be used for this problem. However realizing that the idea of BiGANs has been implemented, and it was difficult to stabilize the alternating gradients for the BiGANs, we decided to discard the idea and use autoencoders instead. We believe the augmented Lagrangian techniques to optimize the gradient flows simultaneously could result in better optimization, as compared to the alternating backprop which was proposed in this report. ADMM (Alternating Directions of Multiple Multipliers) works as the gradient-free optimization which we believe could give better results [4].

9 FINAL REMARKS

In this project, we proposed masked loss function which is a tweak to avoid sparsity. We also proposed autoencoders based sandwich model where we transferred two pre-trained models and learn mappings between those two models by finetuning weights of this sandwich model. Finally, we proposed bi-directional autoencoders which were optimized using alternating gradients. One of our biggest missteps was to consider this project as a pure regression problem. But as a matter of fact, we should have preprocessed our data according to the cold-start recommendation problem. This resulted the lower precision of our projects despite introducing robust architectures. Nevertheless, we proposed a lot of novel algorithms which surprisingly behaved well within this tight time constraints. In the future, we would like to solve the joint backprop optimization using ADMM for our bi-directional autoencoders model.

REFERENCES

[1] Yoshua Bengio. 2012. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. 17–36.

[2] John Langford, Lihong Li, and Tong Zhang. 2009. Sparse online learning via truncated gradient. *Journal of Machine Learning Research* 10, Mar (2009), 777–801.

[3] T. Miyato, A. M. Dai, and I. Goodfellow. 2016. Adversarial Training Methods for Semi-Supervised Text Classification. *ArXiv e-prints* (May 2016). arXiv:stat.ML/1605.07725

[4] G. Taylor, R. Burmeister, Z. Xu, B. Singh, A. Patel, and T. Goldstein. 2016. Training Neural Networks Without Gradients: A Scalable ADMM Approach. *ArXiv e-prints* (May 2016). arXiv:cs.LG/1605.02026