Fast Gated Neural Domain Adaptation: Language Model as a Case Study

Jian Zhang, Xiaofeng Wu, Andy Way, Qun Liu
ADAPT Centre
School of Computing
Dublin City University, Ireland
jian.zhang,xiaofeng.wu,andy.way,qun.liu@adaptcentre.ie

Abstract

Neural network training has been shown to be advantageous in many natural language processing applications, such as language modelling or machine translation. In this paper, we describe in detail a novel domain adaptation mechanism in neural network training. Instead of learning and adapting the neural network on millions of training sentences – which can be very time-consuming or even infeasible in some cases – we design a domain adaptation gating mechanism which can be used in recurrent neural networks and quickly learn the out-of-domain knowledge directly from the word vector representations with little speed overhead. In our experiments, we use the recurrent neural network language model (LM) as a case study. We show that the neural LM perplexity can be reduced by 7.395 and 12.011 using the proposed domain adaptation mechanism on the Penn Treebank and News data, respectively. Furthermore, we show that using the domain-adapted neural LM to re-rank the statistical machine translation *n*-best list on the French-to-English language pair can significantly improve translation quality.

1 Introduction

One challenge which rises above others in natural language processing (NLP) is where application performance decreases when there are *dissimilarities* between the training and the testing environments. In research on domain adaptation, training data with the same style and topic (van der Wees et al., 2015) as the test data is often defined as in-domain (ID) data, with everything else called general-domain (GD) data. The ID data is often scarce and expensive to obtain, whereas the GD is plentiful and easy to access.

One approach to address the situation of scarce ID training data is through data selection. For example, using the perplexity difference between ID and GD can select data that is close to ID and away from GD (Moore and Lewis, 2010). The selected data can then be concatenated with ID data for training. However, the drawback of using data selection is a threshold needs to be set, which often requires many models to be trained and evaluated. The situation will worsen if neural networks are used since the computational cost is immense in neural network training, where models often require days to converge even with the help of GPU-accelerated computing. Thus, simply adapting previous domain adaptation techniques into the neural network framework may not be efficient or effective. Ideally, we want to have an adaptation approach which has the ability to learn knowledge from huge corpora at speed. The question that arises here is how to make use of large amounts of GD data but avoiding long training times.

In neural network training, words are represented as distributed representations, so-called "word vectors", which can be pre-trained or trained with a specific task in mind. Although a pre-trained word vector model is also learned with a neural network, the training can be very fast. Recent optimized work shows learning word vectors can process more than 100 billion tokens in one day on a single machine (Mikolov et al., 2013c). Another advantage of a pre-trained word vector model is its flexibility, as it can be used later for different task-specific models. Furthermore, the pre-trained and the task-specific word vector models have no functional difference. Accordingly, we think it is very natural to use them together. In this paper, we propose to perform domain adaptation from the large pre-trained word vector models instead of the raw text, i.e. adapting from large pre-trained word vector into the task-specific one. In this approach, we can make use of huge GD corpora with little speed overhead. We can also adapt the richer GD word representations into ID training.

2 Background

2.1 Word Vectors

Word vectors are important building blocks in neural networks. While much work has been proposed (Hinton et al., 1986; Mikolov et al., 2013b), we focus on the most popular approach of Mikolov et al. (2013b), which uses a neural network to produce distributed word representations. Unlike other training algorithms, labeled data is not required. It uses context words as features to make predictions. Word vectors can capture linguistic regularities and similarities (Mikolov et al., 2013d) in the training corpus. For example, using vector operations "king" - "man" + "woman" can result in a vector which is close to the word "queen".[1]

2.2 Neural Language Model

In a nutshell, the Recurrent Neural Network (RNN) language model (LM) uses the previous words to estimate the probability of the next word. The simplest RNN LM consists of a look-up layer, a hidden layer with recurrent connections and an output layer. The input words are firstly taken by the look-up layer and converted into word vector representations. Then, the hidden layers project the word vectors into a context vector with the states of input histories maintained. The output layer is a *Softmax* function. It decodes the context vector and distributes probabilities over all words in the vocabulary. The word with the highest probability is then chosen as the predicted output.

For notational convenience, the look-up layer, the hidden layer and the output layer of RNN LM can be represented as in Equations (1), (2) and (3), respectively:

$$x_t = \textit{look-up}(s) \tag{1}$$

$$h_t = \textit{RNN}(x_t, h_{t-1}) \tag{2}$$

$$y(t) = Softmax(f(h_t)) \tag{3}$$

where x_t is the word vector representation of s, s is the input at time t, and *look-up* is the look-up layer. The hidden layer *RNN* is then applied on x_t and the previous hidden state h_{t-1} to obtain the current hidden state h_t. f is a function that can map the hidden state into a vocabulary size vector. $y(t)$ is the prediction, which is the distribution over all words in the vocabulary.

Early work on neural LMs used simpler networks, such as the feed-forward neural network (Bengio et al., 2003). Later work (Mikolov et al., 2010) used simple RNNs for the input sentences, which showed large improvements over neural LMs. However, the simple RNN suffers from the vanishing gradient problem (Bengio et al., 1994). The Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) or the more recently introduced gated recurrent unit (GRU) (Chung et al., 2014) use gates to control the information flow from previous words. Thus, LSTM and GRU are better at capturing long-term dependencies than simple RNNs, and are often chosen in practice.

2.3 Gated Recurrent Unit

We use the GRU as a case study in this paper. The GRU consists of an update gate and a reset gate, as in Equation (4):

$$\begin{aligned} u_t &= \sigma(W_u x_t + U_u h_{t-1} + b_u) \\ r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\ \tilde{h}_t &= tanh(W x_t + U(r_t \odot h_{t-1}) + b) \\ h_t &= (1 - u_t) \odot h_{t-1} + u_t \odot \tilde{h}_t \end{aligned} \tag{4}$$

[1]The example is taken from Mikolov et al. (2013d)

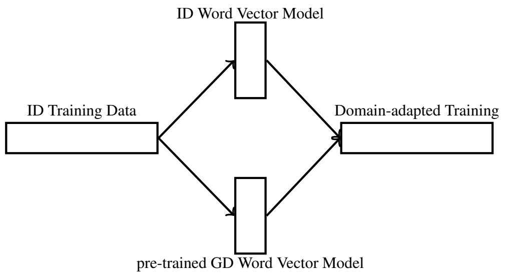


Figure 1: Adaptation Flow

where u_t is the update gate; r_t is the reset gate; $\tilde{h}_t$ is the candidate activation; $\odot$ is the element-wise multiplication operation, and h_t is the linear interpolated output between the previous hidden state h_{t-1} and the candidate activation. Intuitively, the update gate determines the interpolation weights between the previous hidden state h_{t-1} and the candidate activation, and the reset gate determines the information flow from previous hidden states. If the reset gate is set to be 1 and the update gate is set to be 0, the GRU is equivalent to the simple RNN. W_u, U_u, W_r, U_r, W and U are the weight parameters, and b_u, b_r and b are the bias values of the corresponding gates. σ is the sigmoid function and $tanh$ is the hyperbolic tangent function.

3 Adaptation Mechanisms

In this section, we describe several adaptation mechanisms proposed in this paper. In order to distinguish the input layers or hidden layers used in the network, we use Equations (1) and (2) to represent the ID training path. We use Equations (5) and (6) to notate the training path of GD:

$$x_t^* = \textit{look-up}^*_{pre-trained}(s) \tag{5}$$

$$h_t^* = \textit{RNN}^*(x_t^*, h_{t-1}^*) \tag{6}$$

where *look-up*$^*_{pre-trained}$ is a pre-trained word vector model and static.[2] x_t^* is the word vector representation of input s, h_t^* is the hidden state and h_{t-1}^* is the previous hidden state of word s. For example, given input s from the ID training data, we can obtain two word vector representations (x_t and x_t^*). In addition, the two representations can then be fed into the corresponding hidden layer (*RNN* and *RNN**[3]). It is also worth mentioning that the proposed LMs are trained from scratch on ID training data, but adapting knowledge from the GD word vector model. Thus, the hidden state in Equation (6) is not strictly the "GD hidden state"; it uses the word embeddings from the GD pre-trained word vector model, but the inputs are still ID data.[4] Figure 1 shows the adaptation flow proposed in this paper, where the domain-adapted training is presented in Section 3.1, 3.2 and 3.3.

3.1 Adaptation on Word Vectors

The look-up table in the neural network contains word vector representations for all words in the vocabulary. We first propose to integrate the word vectors of ID and GD. The word vector from the ID look-up table contains the input word meanings with adaptation, whereas the GD look-up table contains richer word representations trained on a very large data set. We propose the following two approaches to adapt from the GD look-up table:

1. Word Vector Concatenation (WVC): we concatenate the word vectors obtained from ID and GD lookup-tables, as in Equation (7):

$$x_t^{\textit{WVC}} = [x_t^*, x_t] \tag{7}$$

[2] By static, we mean the pre-trained word vector model is not updated during training.

[3] Depending on the adaptation methods that we will describe in this section, *RNN** is not always used. For example, when the adaptation is performed on word vectors (Section 3.1), *RNN** is not used.

[4] Note that Equation (1) and Equation (5) have the same input s.

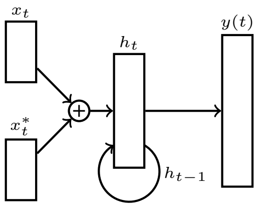


Figure 2: RNN LM with adaptation on word vectors, where $\oplus$ indicates the adaptation mechanisms described in Section 3.1.

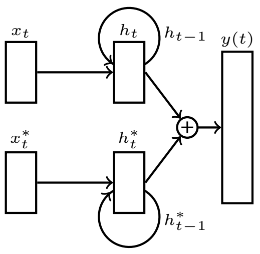


Figure 3: RNN LM with adaptation on context representations, where $\oplus$ indicates the adaptation mechanisms described in Section 3.2.

2. Word Vector Sum (WVS): we sum the two word vectors from ID and GD lookup-tables. In this approach, the two word vectors need to always have the same dimensionality, as in Equation (8):

$$x_t^{WVS} = x_t^* + x_t \tag{8}$$

When adapting, we replace the x_t in Equation (2) with x_t^{WVC} or x_t^{WVS}. Figure 2 is a graphical illustration of adaptation on word vectors.

3.2 Adaptation on Context Representations

Another approach is to delay the domain adaptation step until the context information is available. The RNN encapsulates the word vectors of the current words and previous context, and then produces a representation of the current context. Intuitively, if we maintain separate RNNs, where one uses the ID word representation and another one uses the GD word representation, there will be two pieces of context information available to us, namely contexts with the meaning of ID and GD. In this case, the following domain adaptation approaches can be taken:

1. Context Vector Concatenation (CVC): we can concatenate the two context vectors, as in Equation (9):

$$h_t^{CVC} = [h_t^*, h_t] \tag{9}$$

2. Weighted Context Vector Concatenation (WCVC): we can extend the CVC approach by applying a concatenation weight on h_t^* in Equation (6). Thus, the network can have simple control over the amount of the information flowing from GD, as in Equation (10):

$$h_t^{WCVC} = [Wh_t^*, h_t] \tag{10}$$

3. Context Vector Sum (CVS): we can also add the ID context vector and the GD context vector. We then have the compacted information from two domains to represent the context, as in Equation (11):

$$h_t^{CVS} = h_t^* + h_t \tag{11}$$

4. Weighted Context Vector Sum (WCVS): another approach is to apply a weight vector on the GD context vector. Thus the information from GD can be controlled before compacting, as in Equation (12):

$$h_t^{WCVS} = Wh_t^* + h_t \tag{12}$$

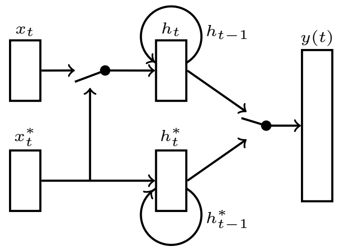


Figure 4: RNN LM with gated adaptation, where ● indicates the gates mechanisms described in Section 3.3.

Therefore, we replace the h_t in Equation (3) with h_t^{CVC}, h_t^{WCVC}, h_t^{CVS} or h_t^{WCVS} when adapting. Figure 3 is a graphical illustration of adaptation on context representations.

3.3 Gated Adaptation

However, directly applying adaptation on the context may not be efficient. The information adapted from GD is forced to be used by ID, i.e. the concatenation or sum operations. Furthermore, the adaptation operations are segmented. Ideally, we want to have an adaptation mechanism that can sequentially adapt the information from GD for the sequence inputs. Thus, we propose various gated domain adaptation mechanisms.

1. Word Vector Gating (WVG): in the WVG approach, we first design a gate to control the information flow from GD word vectors, as in Equation (13):

   $$u_t^{WVG} = \sigma(W_u x_t + U_u h_{t-1} + W_u^* x_t^* + U_u^* h_{t-1}^* + b_u) \tag{13}$$

   where u_t^{WVG} is the adapted update gate on word vectors. It is computed using the known knowledge, i.e. x_t and x_t^* are the word vectors of the current word from ID and GD, respectively, and h_{t-1} and h_{t-1}^* are the previous context vectors of ID and GD, respectively. We then use a linear sum to combine the word vector representations (ID and GD), as in Equation (14):

   $$x_t^{WVG} = u_t^{WVG} \odot x_t + (1 - u_t^{WVG}) \odot x_t^* \tag{14}$$

   where x_t^{WVG} is the domain-adapted word vector. Such an adaptation approach ensures that when the gate u_t^{WVG} tends to 1, we only use the ID word vector x_t, and when u_t^{WVG} tends to 0, the information from GD is fully cascaded to the current word vector.

   To use the domain-adapted word vector, we can simply replace the original x_t in Equation (2) with x_t^{WVG}.

2. Context Vector Gating (CVG): a similar gating mechanism can also be applied to the context vector, as in Equation (15):

   $$r_t^{CVG} = \sigma(W_r x_t + U_r h_{t-1} + W_r^* x_t^* + U_r^* h_{t-1}^* + b_r) \tag{15}$$

   where r_t^{CVG} is the adapted update gate on the context vectors. We can also use the linear sum operation to combine the context vectors of ID and GD, as in Equation (16):

   $$h_{t-1}^{CVG} = r_t^{CVG} \odot h_{t-1} + (1 - r_t^{CVG}) \odot h_{t-1}^* \tag{16}$$

   We then use the domain-adapted context vector h_{t-1}^{CVG} to replace the original context vector h_{t-1} in Equation (2).

3. Domain-Adapted GRU (DAGRU): the WVG and CVG mechanisms can also be used together. In this way, we can adapt both the word and context information of GD to the ID word and context vectors.

Figure 4 illustrates the gated adaptation mechanisms.

	Sentences	**Tokens**
Training	42,068	887,521
Validation	3,370	70,390
Test	3761	78,669

Table 1: Statistics of the Penn Treebank corpus.

3.4 Properties

It is also worth mentioning the following properties in our proposed adaptation mechanisms:
Property 1: One of the main differences between our proposed adaptation mechanisms and previous adaptation mechanisms is that we adapt word vector representations from a pre-trained word vector model trained using a large GD data set. However, previous work focused on adapting raw word(s) from GD, i.e. the data selection approach or the model combination approach. Our adaptation method is a more natural fit for neural network training. Furthermore, we think there is no contradiction between our approach and the previous data selection approach (Moore and Lewis, 2010), as data selection can always be performed before LM training. Note that our approach performs domain adaptation during LM training.
Property 2: Our approach also differs in the notation of GD data. Previous work defines a large corpus as the GD data, whereas our GD data here is the pre-trained GD word vector model. Thus, our adapted RNN LM model is still (only) trained using ID sentences. In practice, this is an efficient adaptation mechanism since there will be no extra training time brought by the additional training corpus.
Property 3: The pre-trained GD word vector model is static, which means it is not updated during training. This is because the pre-trained word vector model is obtained from a very large GD data set. The word vectors in such a model are not domain-specific. By keeping it static, we interpret it as a "knowledge database", and the knowledge should be consistent. Another practical reason for not updating the pre-trained GD word vector model is that fewer parameters need to be optimized in the network.

4 Experiments

4.1 Adaptation on Penn Treebank and News Corpus

The typical setting of domain adaptation is small amount of ID training data and large amount of DG training data. Accordingly, we choose to use the availability of widely known Penn Treebank (Marcus et al., 1993) portion of the Wall Street Journal corpus in our LM adaptation experiment.[5] The words outside the 10K vocabulary frequency list are mapped to the special *unk* token; sections 0-20 are used for training, and sections 21-22 are used for validation. We report the perplexity on data from sections 23-24. More detailed data statistics are summarized in Table 1. We use the pre-trained word vector Google *word2vec*[6] (Mikolov et al., 2013a) as the GD "look-up table". It is trained on about 100 billion words, and consists of 3 million words and phrases. The word vectors are 300-dimensional in the *word2vec* model.

In the experiments, our LM is trained with a single GRU hidden layer containing 600 hidden units. We uniformly initialize the weight parameters between [-0.1,0.1]. We set the maximum training iterations to 25. We set the initial learning rate to be 1, and then apply the learning rate with a decay factor of 0.5 after 13 iterations. The model is optimized using stochastic gradient descent with batch size of 20. We set the back-propagation through time to 40 time steps. The word embedding size in the loop-up layer is set to 600 for the baseline models. For a fair comparison, we use word embedding size 300 in the gated adaptation experiments since we are also using the pre-trained word vectors of 300.

Table 2 lists the perplexity results for the LM experiments. The baseline (KN5) model is a 5-gram LM trained using modified Kneser-Ney smoothing. It results in 148.007 and 141.186 perplexities on the validation and test data set, respectively. The baseline (*word2vec*) model is a neural LM, which uses the

[5]We download the data from `http://www.fit.vutbr.cz/~imikolov/rnnlm/simple-examples.tgz`
[6]`https://code.google.com/archive/p/word2vec/`

	Validation Set	Test Set
Baselines		
Baseline (KN5)	148.007	141.186
Baseline (*word2vec*)	121.871	117.730
Baseline (Standard)	92.983	89.295
Adaptation on Word Representations		
WVC	95.149	91.414
WVS	88.398	85.231
Adaptation on Context Representations		
CVC	90.337	86.168
WCVC	88.551	85.067
CVS	88.244	84.721
WCVS	90.293	86.679
Gated Adaptation		
WVG	90.937	87.853
CVG	90.301	86.832
DAGRU	**86.247**	**81.900**

Table 2: LM perplexity on Penn Treebank corpus.

	Sentences	Tokens
Training	181,108	4,691k
Validation	3,000	64k
Test	3,003	71k

Table 3: Statistics of the News corpus.

pre-trained *word2vec* model as the embedding layer. It can achieve 121.871 and 117.730 perplexities on the validation and test data set, respectively. The baseline (Standard) model is a neural LM, which is trained only on the ID data, it can achieve 92.983 and 89.295 perplexities on the validation and test data set, respectively. For the baseline (*word2vec*) model, we observe a sudden explosion in the evaluation perplexity. We experimentally set the learning rate with a decay factor of 0.5 after 4 iterations.

In the adaptation on word representations experiments, we found that summing up the word vectors in WVS can outperform the concatenation approach of WVC. Adding up the context representations in CVS is also more useful than concatenating the context representations in CVC. Thus, we can draw the conclusion that information from GD should be compressed (summed) into ID rather than using scattered (concatenated) representations. However, weighted vectors can be harmful to the sum approaches in WCVS. We think this is because the adapted representation is a newly computed vector after the sum operation, where the weight vectors are hidden behind the sum operation. Thus the model can be hard to optimize. In contrast, when applying weight vectors in the concatenation cases in WCVC, the adapted representation is still separable into domains. Thus the weights in the adapted model are easy to optimize. However, observing the experimental results, only a small positive impact can be observed when applying weighted vectors to the concatenation approaches. For example, approximately 1 perplexity point difference can be found between CVC and WCVC models. This indicates that the approach of using weight vectors for domain adaptation in neural network training is too simple.

We now move our focus to the News corpus. We test the DAGRU adaptation approach on the target side of the French-to-English News Commentary v10 corpus from the WMT2015 translation task. We use corpus newstest 2013 for evaluation and newstest 2014 for testing the trained LMs. More detailed data statistics are summarized in Table 3. Table 4 presents the LM perplexity differences between the baseline LM and the adapted LM. On the News training corpus, the adapted LM can also produce a better result than the baseline LM with a perplexity reduction of 12.

4.2 Statistical Machine Translation Re-ranking

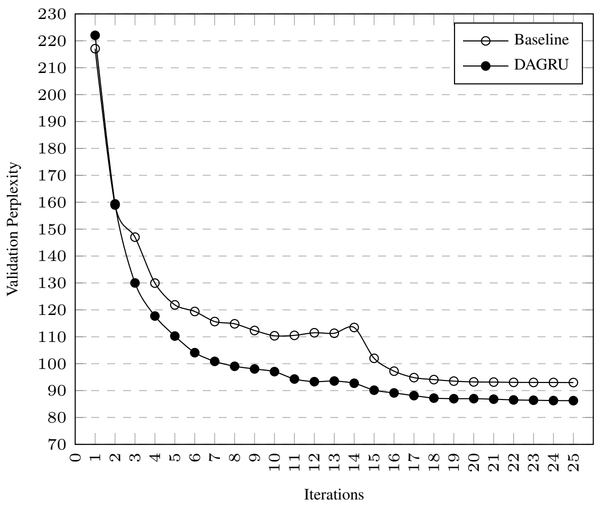
In this experiment, we apply the DAGRU-adapted LMs trained in Table 4 on the statistical machine translation (SMT) *n*-best re-ranking task. We use the French-to-English News Commentary v10 and Europarl v7 corpus from the WMT2015 translation task to train our baseline SMT system. The newstest 2013 and 2014 data sets are used for tuning and testing for the SMT, respectively. The SMT baseline

System	Validation Set	Test Set
Baseline LM	94.341	109.420
DAGRU LM	85.551	**97.409**

Table 4: LM perplexity on News dataset.

System	Tuning	Test
Baseline	26.18	27.14
Baseline + baseline LM	26.71	27.65
Baseline + DAGRU LM	27.17	**27.96**

Table 5: BLEU scores of baseline SMT and re-ranked SMT.



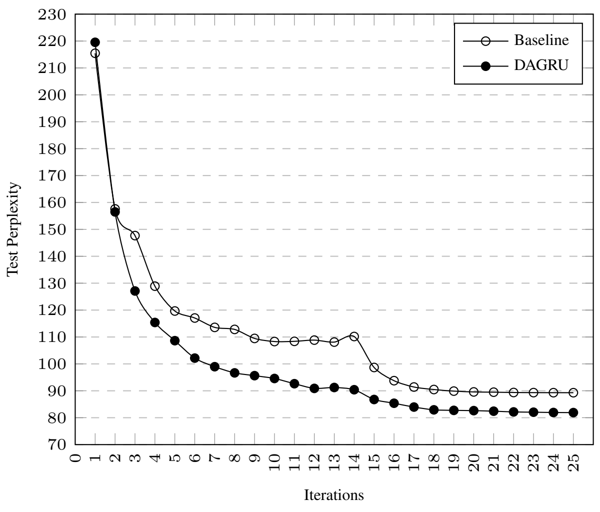


Figure 5: Language model coverage plot for baseline LM and DAGRU LM on the Penn Treebank data.

is trained using Moses (Koehn et al., 2007), with a lexicalized reordering model (Galley and Manning, 2008) and a 5-gram KenLM (Heafield, 2011) LM trained on the target side of the SMT training data.

The re-ranked SMT systems can both increase the baseline BLEU score significantly as seen in Table 5. Using the baseline LM for re-ranking, we can observe a 0.51 absolute improvement (1.8% relative). Using the DAGRU LM for re-ranking, we an obtain a 0.82 absolute (3% relative) improvement. Comparing the two re-ranked systems, we observe that using DAGRU LM for re-ranking can provide an additional increase of 0.31 (1.1% relatively) in BLEU score. The improvement is statistically significant (Koehn, 2004). The significance testing uses bootstrapping method at the level p = 0.05 level with 1,000 iterations.

	Validation Set	Test Set
Word Embedding = 50		
Baseline	104.465	101.285
SENNA (Collobert et al., 2011)	95.453	92.026
GloVe (glove_6b) (Pennington et al., 2014)	**95.292**	**91.251**
Word Embedding = 100		
Baseline	97.034	93.645
GloVe (glove_6b) (Pennington et al., 2014)	**89.521**	**85.330**
Word Embedding = 200		
Baseline	94.201	90.993
GloVe (glove_6b) (Pennington et al., 2014)	**86.638**	**82.762**
Word Embedding = 300		
Baseline	92.983	89.295
GloVe (glove_6b) (Pennington et al., 2014)	86.438	82.593
GloVe (glove_42b) (Pennington et al., 2014)	87.096	82.297
GloVe (glove_840b) (Pennington et al., 2014)	86.853	82.165
Google (word2vec)	**86.247**	**81.900**

Table 6: The DAGRU adaptation on different word vector models

4.3 Observations

There are many reasons to explain the efficiency of the DAGRU approach. First, from the perspective of adapted information, our method is not only adapting knowledge from GD, but also with very little speed overhead in the neural network training framework. It makes use of the internal data representation in neural network training. We adapt GD information directly from distributed word representations, which is a more natural way of learning in the neural network.

A further possible explanation is that the DAGRU approach is a better fit to the sequence learning tasks. We proposed several adaptation methods in Section 3, where adapting word vector representations ignores the previously adapted histories. Words are adapted individually from GD at each step. The approach of adapting context vector representations has the same issue. Although the adapted information is the context vectors which are generated by previous histories, the adaptation is still segmented. Taking the CVC approach as an example, the adapted context vector h_t^{CVC} is only used for predicting outputs in the output layer at step t, but not in the adaptation operation at step $t + 1$. In contrast, the DAGRU approach can achieve sequential adaptation. The gates (u_t^{WVG} and r_t^{CVG}) are computed by the previous adapted context vector h_{t-1} and the previous GD context vector $h_{t_1}^*$. Furthermore, the computation of the current adapted representation also involves h_{t-1} and $h_{t_1}^*$. Thus, the adaptation histories are managed by the model and sequentially traverse along the input string.

Furthermore, we also compare the learning curves between the baseline LM and the DAGRU LM on validation and test data of the Penn Treebank. As Figure 5 shows, the predictions become more certain and accurate after iterations for training both LMs. Already after training iteration 2, the DAGRU LM starts to outperform the baseline LM in terms of perplexity at every iteration. The plots flatten after 18 iterations, and the learning begins to converge for both the baseline LM and DAGRU LM.

To demonstrate the scalability of the DAGRU adaptation approach, we also train LMs adapting from other freely available word vector models. SENNA (Semantic/syntactic Extraction using a Neural Network Architecture) is the word vector model received after a LM training (Collobert et al., 2011). The training data is obtained from Wikipedia. GloVe (Pennington et al., 2014) – Global Vectors for Word Representation – provides several versions of word vector models. The glove_6b model is trained on Wikipedia data and the English Gigaword Fifth Edition corpus;[7] the glove_42b model is trained on the Common Crawl data; and the glove_840b model is trained on the the Common Crawl and additional web data.

Table 6 presents the experimental results of DAGRU adaptation using different word vector models as GD data. The baseline models are trained on the Penn Treebank only. The word embedding numbers in Table 6 indicate the word vector size of the adapting word vector model, e.g. the SENNA model has a word vector size of 50 under Word Embedding = 50 setting. For the baseline systems with embedding

[7]`https://catalog.ldc.upenn.edu/LDC2011T07`

size 50 and 100, we observe a sudden explosion in the evaluation perplexities with the decay factor setting described in Section 4.1. We experimentally set the learning rate to a decay factor of 0.5 after 8 iterations. In Table 6, the DAGRU approach can produce better perplexity results in all settings.

5 Related Work

Domain adaptation for *n*-gram LMs is a well-studied research field. In general, there are approaches to select data which are similar to ID from GD (Moore and Lewis, 2010; Axelrod et al., 2011; Duh et al., 2013; Toral, 2013). There are also model mixture approaches (Bellegarda, 2004; Hsu and Glass, 2006; Allauzen and Riley, 2011), which try to find a weight to combine the ID LM and GD LM.

In neural LM work, one approach to perform domain adaptation is to use an additional adaptation layer to combine the GD neural LM into the ID neural LM (Park et al., 2010; Ter-Sarkisov et al., 2014). However, an LM trained on all GD data is required. Curriculum learning (Bengio et al., 2009), which rearranges the training data in a particular order to improve generalization, is also applied on domain adaptation on a neural LM (Shi et al., 2013). In the work of Mikolov and Zweig (2012), word predictions are conditioned on the word topic representations. Thus, building multiple topic-specific language models is avoided.

6 Conclusions and Future Work

In this paper, we present and compare several domain adaptation approaches using neural LM training. Compared to previous domain adaptation methods, we propose the idea of learning GD knowledge directly from GD word vectors instead of from raw sentences. Using the proposed domain adaptation gating mechanism, we demonstrate LM perplexity improvements on the Penn Treebank and News domain data sets. We compare the adaptation performance on several publicly available word vector models. We also apply the adapted LM in the SMT re-ranking task. The experimental results suggest that the proposed domain adaptation gating approach can efficiently produce a better LM and significantly improve SMT translation performance.

Our domain adaptation gating mechanism is not only limited to LM training. We are interested in further exploring its performance on other NLP tasks, e.g. neural machine translation. In future work, we are also interested to find out the efficiency of applying it to other gated RNNs, such as LSTM.

Acknowledgments

We would like to thank the three anonymous reviewers for their valuable and constructive comments and the Irish Center for High-End Computing (`www.ichec.ie`) for providing computational infrastructures. The ADAPT Centre for Digital Content Technology (`www.adaptcentre.ie`) at Dublin City University is funded under the Science Foundation Ireland Research Centres Programme (Grant 13/RC/2106) and is co-funded under the European Regional Development Fund.

References

Cyril Allauzen and Michael Riley. 2011. Bayesian Language Model Interpolation for Mobile Speech Input. In *INTERSPEECH 2011, 12th Annual Conference of the International Speech Communication Association*, pages 1429–1432, Florence, Italy.

Amittai Axelrod, Xiaodong He, and Jianfeng Gao. 2011. Domain Adaptation via Pseudo In-domain Data Selection. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 355–362, Edinburgh, UK.

Jerome R. Bellegarda. 2004. Statistical Language Model Adaptation: Review and Perspectives. *Speech Communication*, 42(1):93–108.

Yoshua Bengio, Patrice Y. Simard, and Paolo Frasconi. 1994. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Trans. Neural Networks*, 5(2):157–166.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3:1137–1155.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, June 14-18, 2009*, volume 382 of *ACM International Conference Proceeding Series*, pages 41–48, Montreal, Quebec, Canada.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *Computing Research Repository (CoRR)*, abs/1412.3555.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. 2011. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research*, 12:2493–2537.

Kevin Duh, Graham Neubig, Katsuhito Sudoh, and Hajime Tsukada. 2013. Adaptation Data Selection using Neural Language Models: Experiments in Machine Translation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Volume 2: Short Papers*, pages 678–683, Sofia, Bulgaria.

Michel Galley and Christopher D. Manning. 2008. A Simple and Effective Hierarchical Phrase Reordering Model. In *2008 Conference on Empirical Methods in Natural Language Processing, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 848–856, Honolulu, Hawaii, USA.

Kenneth Heafield. 2011. KenLM: Faster and Smaller Language Model Queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland.

G. E. Hinton, J. L. McClelland, and D. E. Rumelhart. 1986. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1. In *Distributed Representations*, pages 77–109. MIT Press, Cambridge, MA, USA.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. LONG SHORT-TERM MEMORY. *Neural Computation*, 9(8):1735–1780.

Bo-June Paul Hsu and James R. Glass. 2006. Style & Topic Language Model Adaptation Using HMM-LDA. In *EMNLP 2007, Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing, 22-23 July 2006*, pages 373–381, Sydney, Australia.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open Source Toolkit for Statistical Machine Translation. In *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, Prague, Czech Republic.

Philipp Koehn. 2004. Statistical Significance Tests for Machine Translation Evaluation. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing , EMNLP 2004, A meeting of SIGDAT, a Special Interest Group of the ACL, held in conjunction with ACL 2004, 25-26 July 2004, Barcelona, Spain*, pages 388–395, Barcelona, Spain.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Tomas Mikolov and Geoffrey Zweig. 2012. Context Dependent Recurrent Neural Network Language Model. In *2012 IEEE Spoken Language Technology Workshop (SLT)*, pages 234–239, Miami, Florida ,USA.

Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. 2010. Recurrent Neural Network Based Language Model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association*, pages 1045–1048, Makuhari, Chiba, Japan.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient Estimation of Word Representations in Vector Space. *Computing Research Repository (CoRR)*, abs/1301.3781.

Tomas Mikolov, Quoc V. Le, and Ilya Sutskever. 2013b. Exploiting Similarities among Languages for Machine Translation. *Computing Research Repository (CoRR)*, abs/1309.4168.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013c. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013*, pages 3111–3119, Lake Tahoe, Nevada, United States.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013d. Linguistic Regularities in Continuous Space Word Representations. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics*, pages 746–751, Atlanta, Georgia, USA.

Robert C. Moore and William D. Lewis. 2010. Intelligent selection of language model training data. In *ACL 2010, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 220–224, Uppsala, Sweden.

Junho Park, Xunying Liu, Mark J. F. Gales, and Philip C. Woodland. 2010. Improved Neural Network Based Language Modelling and Adaptation. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, September 26-30, 2010*, pages 1041–1044, Makuhari, Chiba, Japan.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543, Doha, Qatar.

Yangyang Shi, Martha Larson, and Catholijn M. Jonker. 2013. K-component Recurrent Neural Network Language Models using Curriculum Learning. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding, Olomouc, December 8-12, 2013*, pages 1–6, Czech Republic.

Aram Ter-Sarkisov, Holger Schwenk, Fethi Bougares, and Loïc Barrault. 2014. Incremental Adaptation Strategies for Neural Network Language Models. *Computing Research Repository (CoRR)*, abs/1412.6650.

Antonio Toral. 2013. Hybrid Selection of Language Model Training Data Using Linguistic Information and Perplexity. In *Proceedings of the Second Workshop on Hybrid Approaches to Translation*, pages 8–12, Sofia, Bulgaria.

Marlies van der Wees, Arianna Bisazza, Wouter Weerkamp, and Christof Monz. 2015. What's in a Domain? Analyzing Genre and Topic Differences in Statistical Machine Translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing*, pages 560–566, Beijing, China.