R&D Report

Language Modelling of Code-Switched Text

Vishwajeet Singh Bagdawat

150050046

Supervisor: *Prof. Preethi Jyothi*

2018

Contents

1 Introduction

Code-Switching refers to the use of more than one linguistic variety during conversation. Speakers tend to switch the language across the sentences or inside the sentences as well thus generating two types: Inter-Sentential switching and Intra-Sentential switching. Bilingual or Multilingual, speakers who are proficient in speaking more than one language, tend to code-switch and follow a new grammar and syntax which is different from either of the languages.

With the advent of web and globalization, a significant population is becoming bilingual or multilingual and thus producing more code-switched text on the social platforms like Facebook, Twitter etc. and also posing a challenge in Automatic Speech Recognition. Thus it becomes imperative to process the code-switched text for Natural Language Processing(NLP) tasks such as normalization, language identification, language modelling, POS tagging and more downstream tasks such as machine translation and Automatic Speech Recognition(ASR).

The Natural Language Processing and Automatic Speech Recognition tasks in recent times have seen rapid improvements and have reached human level parity but using the same techniques on code-switched text have resulted in degradation of quality. This motivates the need for specialized techniques and methodology from implementation as well theoretical standpoint. The code-switched text comes with it's own set of limitations and challenges which need to be addressed like the grammar and syntax followed closely depends on the languages used and the matrix language(dominant or more frequent), embedded language(less frequent) pair.

The report focuses on the language modelling aspect of the code-switched text and the experiments done to push the baseline.

2 Related Work

2.1 Factored Language Models

Factored Language Models(FLM)[1][2] have had a impressive run in modelling code-switched text. They allow for easy inclusion of morphological classes, stems, roots, and any other linguistic features that effects the word. The vanilla language model is given by the chain rule of probability as:

$$p(w_{1:T}) = \prod_t p(w_t|w_{1:t-1}) \tag{1}$$

The FLM assumes that the word is dependent on a number of features which are called factors and which runs in parallel to the actual sequence of words. The bundle for $w_t \equiv \{f_t^1, f_t^2, ..., f_t^K\}$ means that K factors determines w_t in some way. Therefore, the actual task is to estimate $p(f_{1:T}^{1:K})$. Now it is easy to see that there are a number of ways to factorize the $f_t^{1:K}$ term and thus represents huge number of statistical models.

$$p(f_t^{1:K}|f_{t-N+1:t-1}^{1:K}) = \prod_k p(f_t^k|f_t^{1:k-1}, f_{t-N+1:t-1}^{1:K}) \tag{2}$$

2.2 Functional Head and Inversion Constraints

The work done by Li and Fung to incorporate the syntactic and linguistic constraints found by linguists also point toward a interesting research direction[3][4].
In the inversion constraints, they use a parallel corpora to translate the sentence into the matrix language and then into the embedded language to find points where the code-switch can't happen. If the tokens aligned in matrix language sentence and embedded language sentence are crossing over then at these points the inversion or code-switch can't happen.
The Functional Head Constraint states that code-switch cannot occur between a functional head and its complements. They use a 2 pass strategy to first decode the speech and then use a lattice based parser to obtain the syntactic structure subject to the Functional Head Constraint. The language model is trained using parallel corpora.

3 Conditional RNNLM

The recent success of Recurrent Neural Networks for language modelling [5] makes RNNLM the most suitable architecture. The experiments described below will modify the basic RNNLM setup gradually to incorporate more side information.The models are coded up in Google's open source deep learning framework Tensorflow and trained on Nvidia's Titan GPU. There are a lot of implementation aspects specific to Tensorflow which will be explained at appropriate places.

3.1 The SEAME Corpus

The corpus used in the project is SEAME(South East Asia Mandarin-English)[6]. It is a conversational speech corpus recorded by speakers in Singapore and Malaysia. The corpus is 30 hour long code-switched speech along with the transcription consisting of spontaneous conversation between bilingual or multilingual speakers. For the task of language modelling the transcriptions are divided into training set, development set and testing set. The table below captures the higher level statistics about the data set.

	Train set	Dev set	Test set
# Sentences	54020	19976	19784
# Words	539185	195551	196462
% Zh words	55.24	48.22	72.18
% Eng words	44.75	51.77	27.81
# Unique tokens	24634	12375	12435
# Unique Zh tokens	14472	6422	7456
# Unique Eng tokens	10158	5951	4973

Table 1: SEAME corpus Statistics

The number of switch points is around 1.6 per sentence for train set, 1.59 for dev set and 1.63 for test set. We can clearly see that the statistics across the train and test set are not similar and this is also reflected in the mismatch of word tokens across the data meaning that quite a lot of frequent words in the test set are not present in the train set.

3.2 Recurrent Neural Networks(RNNs)

Recurrent Neural Networks are superior than their Feed Forward Neural Network counterparts because they are able to pass on information from previous word tokens. If we are trying to do language modelling then it makes perfect sense to have memory of previously seen tokens so that better estimates can be made.
The RNNs can be thought of as directed graphical models but with the edges representing deterministic transitions rather than modelling it in probabilistic sense. At the time of inference, exact inference is infeasible due to growing size of information that would be have to be stored. RNNs make a assumption that the transition is done using the same matrix at each time step which keeps it tractable and also helps generalize better to the data.

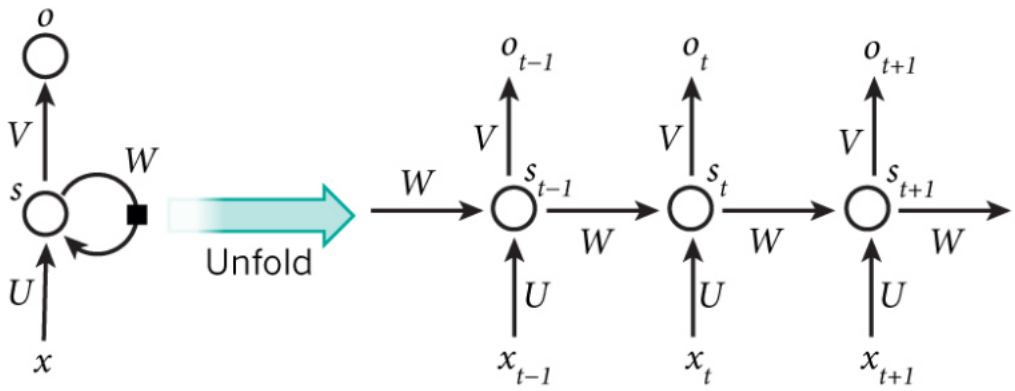

Figure 1: RNN unfolded across time. Source Nature

- x_t is the input data point at time t. It can be a one hot vector in the case of language model.
- s_t is the memory of the RNN which keeps the information about previously seen words in the sentence. The state at time t is calculated with the help of previous state and the current input.
  $s_t = f(Ux_t + Ws_{t-1})$ where $f()$ is some non-linear function such as $tanh$ or $sigmoid$.
- o_t is the output at time step t and if it has to be a probability distribution then it can be calculated as
  $o_t = softmax(Vs_t)$
- The parameters U, V, W are shared across time steps and are jointly learned during training. U is the embedding matrix.

3.3 RNN Language Models

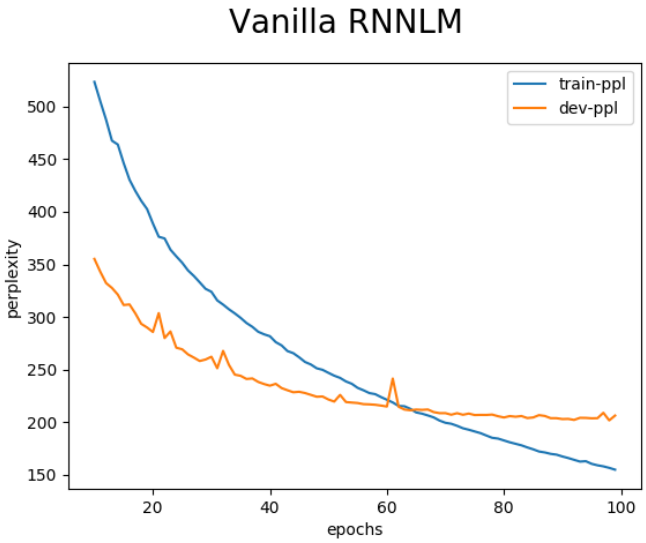

Figure 2: Perplexity plotted as the training progresses

3.3.1 Language Modelling Equation

The language modelling task involves prediction of probability distribution over the words given the context history. Let $w_1, w_2, ..., w_T$ be the words over the vocabulary. In the case of code-switching the vocabulary consists of word tokens from the matrix language as well as the embedded language. Therefore Equation(1) is the quantity of interest that we are modelling. At any given point of time t, only the context history $w_1, w_2, ..., w_{t-1}$ is visible.

3.3.2 Pre-Processing of Data

The pre-processing step is crucial for the success of any language modelling task. The word tokens in the wild have all sorts of differences in terms of the first letter being upper case and the same word having the first letter in small case at different places, acronyms having .in between. This necessitates **tokenization** of the words into a canonical form.
The next step is to append specialized symbols for start and end of sentence, $\langle \mathbf{s} \rangle$ and $\langle /\mathbf{s} \rangle$ respectively. These act as marker for the language model to stop if we are trying to generate sentences using the language model.

3.3.3 Handling Unknown Words at Inference

The statistics of the corpus shows that the model should be robust to unknown words at the test time. Therefore, a special symbol is also added to the vocabulary called unknown $\langle \mathbf{unk} \rangle$ token. In training set there will be no $\langle unk \rangle$ token and hence there will be no reliable estimates for it, to mitigate this problem, the training set is augmented in such a manner that the least frequent words are treated as $\langle \mathbf{unk} \rangle$ token. In the experiments the number of words changed were 1400 which is a tunable hyper parameter.

The parameters of the model are the matrices **U**, **V**, **W**. In the experiments
$\mathbf{U} \in \mathcal{R}^{vocabulary\_size \times 512}$
$\mathbf{W} \in \mathcal{R}^{512 \times 512}$
$\mathbf{V} \in \mathcal{R}^{512 \times vocabulary\_size}$ where vocabulary_size = 23236

3.3.4 Loss Function

The loss function the RNN is trying to minimize is the average negative log probability of the target words. The loss is chosen so that it serves as a proxy to the actual metric we trying to minimize which is **perplexity**.

$$Loss(y_{1:T}, o_{1:T}) = -\frac{1}{T} \sum_{t=1}^{T} log(o_t^{y_t}) \tag{3}$$

It should be noted that quantity we monitor instead is perplexity which is given as:

$$e^{-\frac{1}{T} \sum_{t=1}^{T} log(o_t^{y_t})} = e^{Loss} \tag{4}$$

3.3.5 Implementation Details

The model has 2 layers with dropout added for regularization and the cell used are **LSTM** cells. In Tensorflow, the graph is made using **dynamic_rnn** for faster building and the unrolling is pushed to run time. The model is trained for 100 epochs.

3.4 Oracle Conditional RNNLM

Inspired by[7], conditional RNNLM are explored with the additional information at each time step being whether the next word token is in the same language or not.

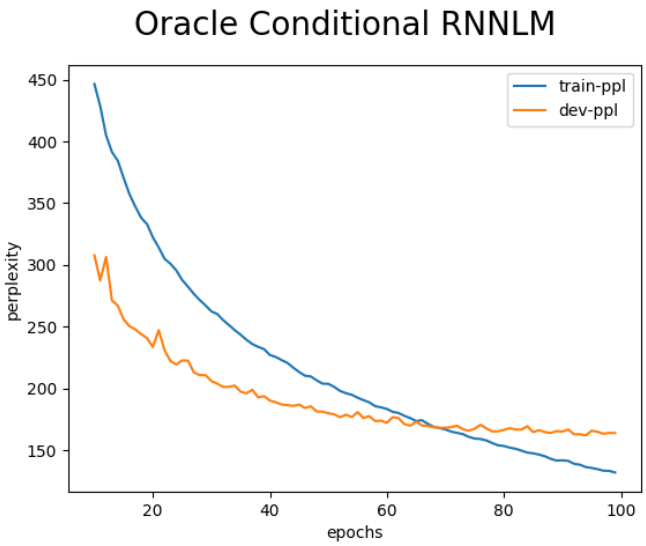

Figure 3

3.4.1 Generating the Segmentation Bits

The data set was used to first distinguish the English words from the Mandarin words by checking the UTF encoding of the characters. After that at each word token, the bit was set to 1 if the next word token is of different language and 0 otherwise. The special symbols are added afterwards, so at the time of loading of the data, segmentation bit is set to 0 for $\langle \mathbf{s} \rangle$ and $\langle /\mathbf{s} \rangle$.

3.4.2 Incorporating the Segmentation Bits

Let $w_1, w_2, ..., w_T$ be the word tokens and the corresponding segmentation bits be $p_1, p_2, ..., p_T$. After the embedding look-up of w_t, say e_t, the segmentation bit is appended to the embedding vector to form $x_t = \binom{e_t}{p_t}$. Now x_t is passed onto the RNNLM instead of just the embedding vector e_t.

3.4.3 Training v/s Inference

The experiment is just to check whether the additional information helps the model or not. We have still not discussed whether the method is Kosher. During training and the inference phase the segmentation bits are available to the RNNLM and are incorporated as described above.

3.5 Predicting the bit using RNN

The earlier method relies on segmentation bits for prediction at the inference time but as the data set is seen fully to generate the segmentation bits, the central constraint of language modelling is violated which is we can't see the word tokens ahead in time.
Therefore other methods have to be explored to predict the segmentation bits at inference time. Several possible solutions to this problem are discussed.

3.5.1 Modified Architecture and Loss Function

The architecture is modified in order to jointly learn the prediction of word probability and the prediction of the next bit. Therefore the loss described in Equation(3) is augmented to force the predicted bit to be correct as well. Let the new predictions be $l_t = \binom{o_t}{\hat{p}_t}$. The modified loss function is given by

$$Loss(y_{1:T}, l_{1:T}) = -\frac{1}{T}\{\sum_{t=1}^{T} log(o_t^{y_t}) + \lambda \sum_{t=1}^{T-1}(p_t log(\hat{p}_t) + (1-p_t)log(1-\hat{p}_t))\} \tag{5}$$

λ is another hyper-parameter which controls how much weight is to be given to making the prediction of segmentation bit correctly. The second term in the loss function is the basic cross entropy loss.

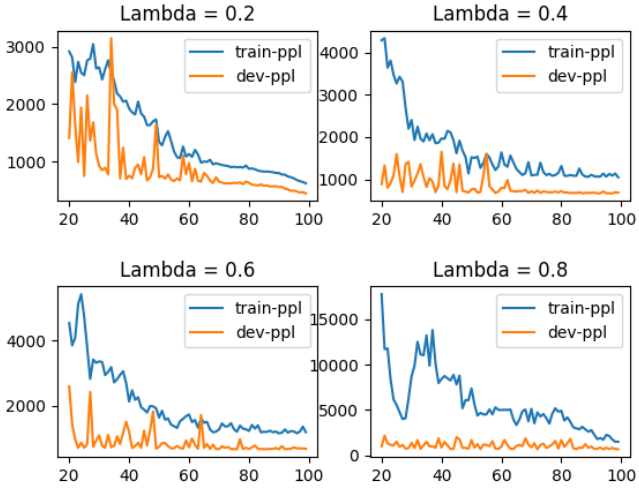

Figure 4: Effect of Varying λ on perplexity

The performance of the model drops significantly as lambda is increased but saturates after 0.6.

3.6 Greedy Decoding

Another technique used is to pass both the bits at inference time and calculate the entropy of the resultant distribution from both the bits. The bit with smaller entropy would be made the output. The intuition behind the technique is that if the entropy is smaller than the model is predicting with more confidence, and hence should be better as well.

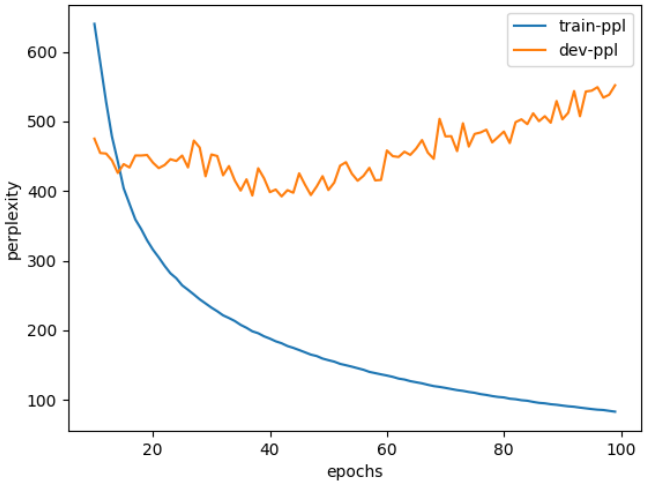

Figure 5: Greedy Decoding performance

3.6.1 Implementation details

Tensorflow's dynamic_rnn feature severely limits the ability to modify the model. Therefore, I had to fall back on to basic for-loop to unroll the architecture. The apprehension behind not doing this in the first place was the speed with which the model trained as the compile time would be significantly higher. But the training time was comparable to dynamic_rnn.

3.7 Lookahead Search

Let the words be $w_1, w_2, ..., w_T$. The sentence is padded with the starting symbol <s >and ending symbol </s >. The ground truth segmentation prediction bits be $p_1, p_2, ..., p_T$.
I'll consider the architecture of one layer RNN but the algorithm generalizes to multi-layered RNN as well.
The parameters of the model are listed below which are learned during training:
embedding matrix $\mathbf{U}(vocab\_size \times embedding\_size)$
state transition matrix $\mathbf{W}((embedding\_size + 1) \times rnn\_size)$
softmax output matrix $\mathbf{V}(vocab\_size \times rnn\_size)$
a bias vector $\mathbf{b}(vocab\_size \times 1)$.

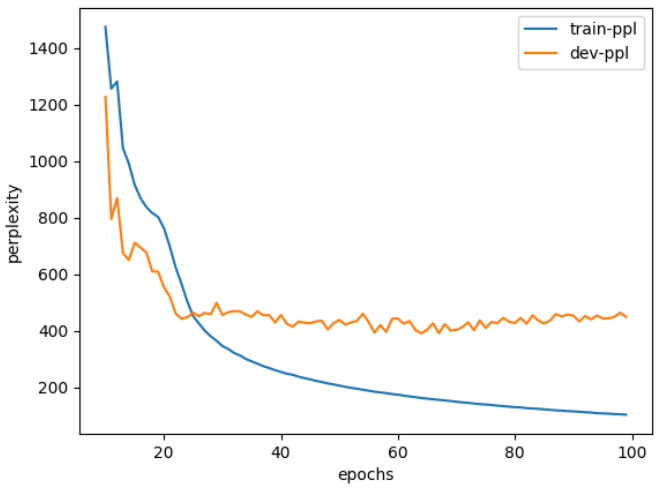

Figure 6: Lookahead Search performance

Algorithm 1 Training Phase

procedure TRAIN
    $state_{-1} \leftarrow$ zeros
    for $t \leftarrow 0...T$ **do**
        $e_t \leftarrow w_t \times \mathbf{U}$ ▷ embedding lookup of word w_t
        $state_t \leftarrow sigmoid([e_t\ p_t] + \mathbf{W} \times state_{t-1})$
        $Pr(w_{t+1} \mid [e_0\ p_0], [e_1\ p_1], ..., [e_t\ p_t]) \leftarrow softmax(\mathbf{V} \times state_t + \mathbf{b})$
    propogate_gradients()

Algorithm 2 Testing Phase

procedure TEST
  $state^0_{-1} \leftarrow$ zeros
  $state^1_{-1} \leftarrow$ zeros
  for $t \leftarrow 0...T$ **do**
    $e_t \leftarrow w_t \times \mathbf{U}$ ▷ embedding lookup of word w_t
    $state^{00}_t \leftarrow sigmoid([e_t\ 0] + \mathbf{W} \times state^0_{t-1})$
    $state^{01}_t \leftarrow sigmoid([e_t\ 1] + \mathbf{W} \times state^0_{t-1})$
    $state^{10}_t \leftarrow sigmoid([e_t\ 0] + \mathbf{W} \times state^1_{t-1})$
    $state^{11}_t \leftarrow sigmoid([e_t\ 1] + \mathbf{W} \times state^1_{t-1})$
    if beam_search = False **then**
      $l, m \leftarrow \underset{i,j \in \{0,1\}}{\operatorname{argmin}} \mathcal{H}(softmax(\mathbf{V} \times state^{ij}_t + \mathbf{b}))$
      $Pr(w_{t+1} \mid [e_0 p_0], [e_1 p_1], ..., [e_t p_t]) = softmax(\mathbf{V} \times state^{lm}_t + \mathbf{b})$
      $p \leftarrow \underset{i \in \{0,1\}}{\operatorname{argmin}} \mathcal{H}(softmax(\mathbf{V} \times state^{0i}_t + \mathbf{b}))$
      $q \leftarrow \underset{i \in \{0,1\}}{\operatorname{argmin}} \mathcal{H}(softmax(\mathbf{V} \times state^{1i}_t + \mathbf{b}))$
      $state^0_t \leftarrow state^{0p}_t$
      $state^1_t \leftarrow state^{1q}_t$
  compute_ppl()

4 Cross-Lingual Word Embedding

The matrix language and embedding language enjoy their separate space but for code-switched corpus getting good estimates of the vectors is a big challenge due to scarce data. The embedding language being smaller in percentage, the estimates for the word prediction suffer a lot. Whereas if the same word is replaced with the word in matrix language, then we can train the model on monolingual corpus which are present in adequate amount. This circumvents the problem of limited code-switched data and projects the problem to a monolingual setting. The idea is already worked upon by Cohn et.al.[8] to help in the low resource language modelling setting using bilingual dictionary. The essential goal therefore is to find good Cross-Lingual Word Embedding in the shared space.

4.1 MUSE

Facebook released MUSE[9], their library for multilingual word embedding with 2 methodology. In supervised setting and unsupervised setting. Unsupervised Learning uses Adversarial Training to find the matrix which projects the vector space of one language to another using just a linear transformation. The loss function in this setting is

$$Loss = \sum_{w^1 \in L1} \|Rw^1 - w^2\| \tag{6}$$

I used their code to find the word embedding for both the languages in a shared space. The dimension of the embeddings being 300.

4.2 Not freezing the weights

I initialized the word embedding matrix U with those found using MUSE. The words for which the embeddings were not present were initialized randomly. The gradients were propagated through the embeddings as well. The model converged to the same setting as of a Vanilla RNNLM. This means that the embeddings didn't had any affect on the performance.

4.3 Freezing the weights

The words for which the embeddings were found using MUSE were frozen i.e. the gradients were not allowed to flow through them. This was done using masking of the words at train time.
The model converged but the training perplexity itself was quite high. This suggests that the capacity of the model is severely limited by freezing the weights of the embedding matrix.

5 Analyzing the Shortcomings and Limitations

The entropy values were analyzed in the greedy decoding method to answer the significant degradation of the results. It was found that the hidden state of RNNLM carries a lot of information and when the bit is changed from 0 to 1, its affect is not felt straight away but in the next time step the entropy values drop by a good margin thus compensating the initial threshold.
To counter the above problem, look ahead search is proposed, though the results are better than just greedy decoding and it also converges well, the effect of wrong prediction at any time step has a cascading effect on the later predictions.
The cross lingual word embeddings are still a promising direction but the mismatch in the type of data we were testing on and the monolingual data the embeddings are trained on just washes away any improvement it might have.

6 Future Directions and Conclusion

The Vanilla RNNLM and greedy decoding can be used in parallel to predict the distribution. The time steps where the entropy values differ by a lot, we can be sure to use the greedy decoding otherwise we fallback on the Vanilla RNNLM for the prediction. The major challenge with this approach is the joint training algorithm to train both the networks simultaneously.
Another direction is to have a different Neural Network to predict the segmentation bits and use them to calculate the ditribution. The bottleneck would be to find a good predictor, as any wrong predictions have the cascading effect.

After doing the project I have realized the actual complexity of the problem and also has made me quite comfortable with Tensorflow :)

References

[1] K. Kirchhoff, J. Bilmes, K. Duh, K. Kirchhoff, J. Bilmes, and K. Duh, "Factored language models tutorial," 2008.

[2] H. Adel, N. T. Vu, K. Kirchhoff, D. Telaar, and T. Schultz, "Syntactic and semantic features for code-switching factored language models," *CoRR*, vol. abs/1710.01809, 2017.

[3] Y. Li and P. Fung, "Code-switch language model with inversion constraints for mixed language speech recognition," 2013.

[4] Y. Li and P. Fung, "Code switch language modeling with functional head constraint," 05 2014.

[5] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur, "Recurrent neural network based language model," 01 2010.

[6] D.-C. Lyu, T.-P. Tan, E. Chng, and H. Li, "An analysis of a mandarin-english code-switching speech corpus: Seame," 05 2018.

[7] C. D. V. Hoang, T. Cohn, and G. Haffari, "Incorporating side information into recurrent neural network language models," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1250–1255, Association for Computational Linguistics, 2016.

[8] T. Cohn, S. Bird, G. Neubig, O. Adams, and A. J. Makarucha, "Cross-lingual word embeddings for low-resource language modeling," in *EACL*, 2017.

[9] A. Conneau, G. Lample, M. Ranzato, L. Denoyer, and H. Jégou, "Word translation without parallel data," *arXiv preprint arXiv:1710.04087*, 2017.