19 Advanced Topics 2: Hybrid Neural-symbolic Models

In the previous chapters, we learned about symbolic and neural models as two disparate approaches. However, each of these approaches have their advantages.

19.1 Advantages of Neural vs. Symbolic Models

Before going into hybrid methods, it is worth talking about the relative advantages of neural vs. symbolic methods. While there are many exceptions to the items listed below depending on the particular model structure, they may be useful as rules-of-thumb when designing models.

First, the advantages of neural methods:

Better generalization Perhaps the largest advantage of neural methods is their ability to generalize by embedding various discrete phenomena in a low-dimensional space. By doing so they make it possible to generalize across similar examples. For example, if a word embedding is similar between two words, these words will be able to share information across training examples, but if we are representing them as discrete symbols this will not be the case.

Parameter efficiency Another advantage of neural models stemming from their dimension reduction and good generalization capacity is that they often can use many fewer parameters than the corresponding symbolic models. For example, a neural translation model may have an order of magnitude fewer parameters than the corresponding phrase-based model.

End-to-end training Finally, neural models can be trained in an end-to-end fashion. The symbolic models for sequence transduction are generally trained by first performing alignment, then rule extraction, then optimization of parameters, etc. As a result, errors may cascade along the pipeline, with, for example, an alignment error having an effect on all downstream processes.

In contrast, there are some advantages of symbolic methods:

Robust learning of low-frequency events One of the major problems of neural models is that while they tend to perform well on average, they often have trouble handling low-frequency events such as low-frequency words or phrases that occur only once or a few times in the training corpus, as the relevant parameters are only updated rarely during the SGD training process. In contrast, symbolic methods often are able to remember events from a single training example, as these events show up as a non-zero count in n-gram models or phrase tables. This is particularly important for the case when there is not much training data, and as a result, symbolic models often outperform neural models in situations where we do not have very much data.

Learning of multi-word chunks A corollary of the previous problem item is that symbolic models are often good at memorizing multi-word units, which are even rarer than words themselves. These show up as n-gram counts or phrase tables, and can be memorized from even a single training example with relatively high accuracy.

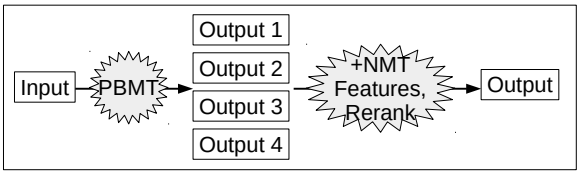

Figure 57: An example of reranking a symbolic PBMT system with neural features.

19.2 Neural Features for Symbolic Models

The first way of combining multiple methods together is to try to incorporate neural features into symbolic models to disambiguate hypotheses.

19.2.1 Neural Reranking of Symbolic Systems

The first method for doing so, **reranking**, is simple: we generate a large number of hypotheses with a symbolic system, use a neural system to assign a score to each of these hypotheses, and select the final hypothesis to output considering the scores of the neural system [14, 11]. An example of this is shown in Figure Figure 57.

Formally, this means that given an input F, we generate an n-best list $\hat{\boldsymbol{E}}$ from our symbolic system, and for each of the candidates, we calculate a new feature function:

$$\phi_{\text{NMT}}(F, \hat{E}) := \log P_{\text{NMT}}(\hat{E} \mid F), \tag{173}$$

where $P_{\text{NMT}}(\hat{E} \mid F)$ is the probability assigned to the output by the NMT system. This feature function can then be incorporated into the log-linear model described in Equation 136 as an additional feature function. In order to choose the weight λ_{NMT} that will tell us the relative importance of the feature function compared to the other features used in the symbolic system, we can perform parameter optimization using any of the algorithms detailed in Section 17.

One important distinction to make is the difference between *post-hoc reranking* as described above, and *incorporation of features* into the MT system itself. In post-hoc reranking, we first generate n-best candidates using some base model, calculate extra features, then re-rank the hypotheses in the n-best list using these features. This has the advantage that it is easy to test new feature functions without messing with the decoder, and has proven useful as a light-weight way to judge the improvements afforded by new methods [12]. On the other hand, the accuracy of the re-ranked results will be necessarily limited by the best result existing in the n-best list, and if the new features are very important for getting good results, reranking has its limits.[52]

19.2.2 Incorporating Neural Features in Symbolic Systems

In order to overcome the limits of n-best reranking, it is also possible to incorporate features directly into the search process for symbolic translation models. The important thing here is

[52] One way to explicitly measure the limits of reranking is by explicitly selecting the hypothesis in an n-best list that has the highest BLEU score. This best achievable hypothesis is often called an **oracle**, and gives an upper bound on the accuracy achievable by reranking. However, because measures like BLEU are overly sensitive to superficial differences in the surface form of the hypothesis (as noted in Section 10), for translation even the n-best oracle can be too optimistic, and not of much use as an upper bound.

that the features be in a form that makes it possible to perform search for symbolic translation models.

Fortunately, for phrase-based models and neural models, search proceeds in the same order; both models generate hypotheses from left to right. As a result, conceptually, it is relatively easy to incorporate neural models into phrase-based translation systems. Let's say a phrase-based system adds a new phrase to a hypothesis, extending the partial translation from $\hat{e}_1^{t_1}$ to $\hat{e}_1^{t_2}$, resulting in an additional $t_2 - t_1$ words in the target hypothesis. If this is the case, the neural machine translation system can calculate the probability of adding these additional words:

$$P(e_{t_1+1}^{t_2} \mid F, \hat{e}_1^{t_1}) = \prod_{t=t_1+1}^{t_2} P(\hat{e}_t \mid F, \hat{e}_1^{t-1}). \tag{174}$$

The log of this value would then be added to the NMT feature function:

$$\phi_{\text{NMT}}(F, e_1^{t_2}) := \phi_{\text{NMT}}(F, e_1^{t_1}) + \log P(e_{t_1+1}^{t_2} \mid F, \hat{e}_1^{t_1}). \tag{175}$$

This allows for search using these feature functions in WFST-based or phrase-based symbolic sequence-to-sequence models.[53]

19.3 Symbolic Features for Neural Models

It is also possible to create hybrid models in the opposite direction: incorporating symbolic information within neural sequence-to-sequence models. The general method for doing so is by using probabilities calculated according to a symbolic model as a seed in calculating the probabilities of a neural model.

Symbolic Biases: One example of this is incorporating discrete translation lexicons as a *bias* into neural translation systems [1]. One of the problems with neural MT systems is that they tend to translate rare words into other similar rare words (e.g. "Tunisia" may be translated into "Norway"), or drop rare words from their translations altogether (e.g. "I went to Tunisia" will be translated into "I went"). In contrast, discrete translation lexicons such as those induced by the IBM models tend to be relatively robust in their estimation of translation probabilities based on co-occurrence statistics, and will certainly never translate between words that never co-occur in the training corpus. If we have a lexicon-based translation probability

$$\boldsymbol{p}_{\text{lex}} = P_{\text{lex}}(e_t \mid F, e_1^{t-1}), \tag{176}$$

this can be used as a bias to a neural model P_{nn} as follows:

$$P_{\text{nn}}(e_t \mid F, e_1^{t-1}) = \text{softmax}(W\boldsymbol{h} + \boldsymbol{b} + \log(\boldsymbol{p}_{\text{lex},t} + \epsilon)). \tag{177}$$

This will seed the translation probabilities with the probabilities calculated using a lexicon. Here, ϵ is a smoothing parameter (like the ones that we used with neural language models), which prevents the model from assigning zero probability to any of the words in the output.

[53] Unfortunately, this method is not applicable to tree-based models using the CKY-like search algorithms described in Section 14. However, there are also methods for left-to-right generation in tree-based models, which impose some restrictions on the shape of the model but would make decoding with these feature functions tractable [18].

So how do we get this lexicon probability? Assume we have a translation probabilities $P_{\text{lex}}(e|f)$ for each word in the vocabulary, calculated by the IBM models or any other method. Given these probabilities, and a source sentence F, we can create a matrix where each column is a vector of lexicon probabilities for each word in the source sentence, and each row represents a word in the target vocabulary:

$$P_{\text{lex}} = \begin{bmatrix} P(e=1 \mid f_1) & P(e=1 \mid f_2) & \cdots & P(e=1 \mid f_{|F|}) \\ P(e=2 \mid f_1) & P(e=2 \mid f_2) & \cdots & P(e=2 \mid f_{|F|}) \\ \vdots & \vdots & \ddots & \vdots \\ P(e=V \mid f_1) & P(e=V \mid f_2) & \cdots & P(e=V \mid f_{|F|}) \end{bmatrix}. \tag{178}$$

Next, we can multiply this matrix by the attention vector at time step t, which allows us to consider the current attention values when choosing which of these lexicon probabilities to consider at the next time step:

$$\boldsymbol{p}_{\text{lex},t} = P_{\text{lex}} \boldsymbol{\alpha}_t. \tag{179}$$

This vector can then be incorporated into the full translation probability as shown in Equation 177, allowing the translation lexicon to bias the neural network's probabilities, which proves effective for preventing unreasonable translation mistakes.

Symbolic Alternatives: Another example of a way to incorporate symbolic probabilities into neural networks is as *alternatives* to using the standard neural probabilities. To give an example here, we could assume that the probability of the next word is calculated as an interpolation between the neural probabilities and the lexicon probabilities:

$$P_{\text{nn}}(e_t \mid F, e_1^{t-1}) = (1-\beta_t)\text{softmax}(W\boldsymbol{h} + \boldsymbol{b}) + \beta_t \boldsymbol{p}_{\text{lex},t}, \tag{180}$$

where β_t is an interpolation coefficient similar to that used in n-gram language models. β_t can be calculated based on the context, so if we are in a context where the model feels confident that its lexicon probability is correct it can set β_t very high to rely on the lexicon probability, and if we are in a context where the model feels confident that the neural model is more correct, it can set β_t to a low value close to zero.

This method of giving the model other alternatives to generate translations is flexible, and has been used in a number of contexts such as:

- Incorporation of translation lexicons [1] or phrase tables [16] into neural sequence-to-sequence models.
- Incorporation of "copying" mechanisms into neural models [5]. This makes it possible for the model to choose whether to output a word from the standard output vocabulary, or copy a word from the input sentence.
- Combination of neural and n-gram and neural language models [10]. This makes it possible to take advantage of the fact that neural language models have some advantages (e.g. better handling of long-distance dependencies and common phenomena), while n-gram language models have others (e.g. better memorization of low-frequency local word sequences).

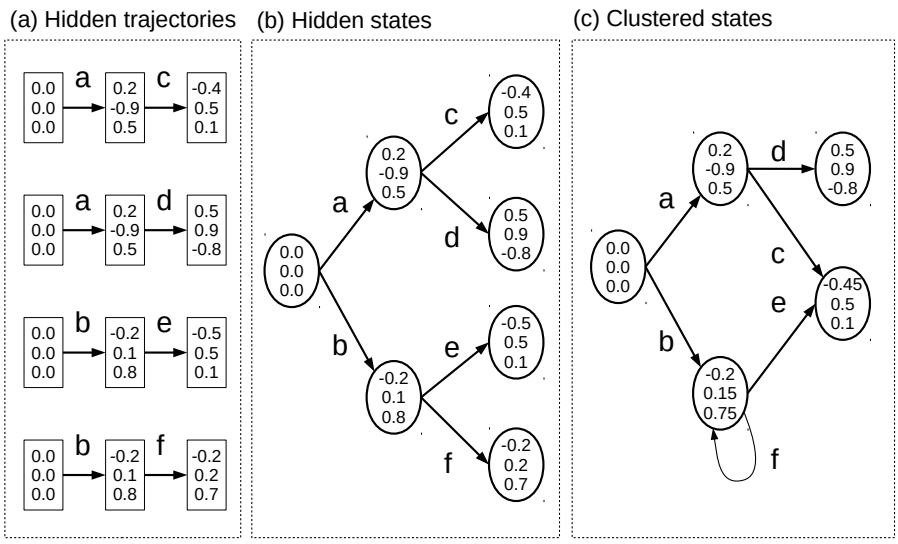

Figure 58: An example of extracing a finite state automaton from a recurrent neural network by (a) calculating RNN trajectories, (b) turning these trajectories into states, and (c) clustering similar states.

19.4 Extracting Symbolic Knowledge from Neural Networks

It is also possible to use neural networks to extract discrete, symbolic structure from text. This makes it possible to use the modeling power of neural networks, but also allow for modeling of discrete structures that tend to be more interpretable or conducive for use for other purposes.

One example of extracting knowledge from neural networks is the extraction of finite state automata from recurrent neural networks [4]. The basic idea behind these methods is that the hidden vector $\boldsymbol{h}_t$ of a neural network can be equated to a representation of its state in a finite state automaton. Let's assume that our RNN is a language model that calculates the probability of E given a state transition function

$$\boldsymbol{h}_t = \text{RNN}(\boldsymbol{h}_{t-1}, e_t), \tag{181}$$

and we would like to create an automaton approximating its state dynamics. This can be done in three steps:

Collecting evidence First, given a large corpus $\mathcal{E} = E_1, E_2, \ldots, E_{|\mathcal{E}|}$, we calculate the hidden state trajectories $\mathcal{H} = H_1, H_2, \ldots, H_{|\mathcal{E}|}$ for each sentence. For the ith sentence, hidden state trajectory $H_i = \boldsymbol{h}_{i,1}, \boldsymbol{h}_{i,2}, \ldots, \boldsymbol{h}_{i,|H_i|}$ is the sequence of hidden states that results from inputting E_i to the recurrent neural network.

Creating an expanded FSA Next, we can create an FSA where each unique vector $\boldsymbol{h}$ is assigned to its own state, and there is an edge labeled with word e_t traveling between every state $\boldsymbol{h}_{t-1}$ to $\boldsymbol{h}_t$.

Clustering states In actuality, because each hidden vector will be different depending on the prefix e_1^t, this will simply result in a tree that branches out until we have one path for each unique sentence in the corpus. This is not very interesting, of course, so to

create a more compact and interpretable FSA, we can cluster together similar hidden vectors into the same discrete state within the FSA. For example, [4] note that the hidden state in a standard RNN is between zero and one, quantize the state to a binary vector, and group states with the same binary vector together. There are also more sophisticated models, such as ones that train a hidden Markov model on top of the hidden state vectors [8, 17].

Another, indirect way in which neural models can contribute to symbolic systems is by improving the learning of alignments. These alignments can be used to improve the extraction of phrases or rules for symbolic sequence-to-sequence modeling methods. A variety of methods for doing so have been proposed, for example by using recurrent neural networks to predict the predictions of a standard symbolic unsupervised word alignment system [15], or by proposing specific model structures that allow for unsupervised learning of alignments [6].

19.5 Further Reading

There are a number of more advanced topics that can be considered when combining together neural and symbolic models.

Better search for symbolic models with neural features As noted in subsubsection 19.2.2, it is possible to perform search with neural features in symbolic models. However, there is still the problem that (unlike when using n-gram models), the context required to calculate these features during search is infinite, which makes it difficult to re-combine hypotheses, and efficiently search over a large number of hypotheses in non-exponential time. There have been a number of solutions to this problem, including shaping neural models so that the context they calculate does not significantly expand the hypothesis space [9, 13], group together hypotheses based on their n-gram context and keep only the best-scoring RNN states [2], or consider only limited contexts like those of n-gram models [3].

Models incorporating discrete variables in neural networks It is also possible to incorporate discrete variables in neural models. Some examples of this include the incorporation of hard attention, which uses a 0-1 discrete binary variable to indicate whether the model is attending to a word or not [7], or the use of discrete structures to model language [19]. Because it is not possible to perform back-propagation through discrete variables, it is often necessary to use more advanced optimization techniques, such as the reinforcement learning methods described in Section 17.

19.6 Exercise

As an exercise here, you could try to incorporate Model 1 translation probabilities that you calculated in the exercise of Section 11 into the attentional model that you implemented in the exercise for Section 8. This would involve calculating the matrix of Equation 178, and incorporating this into your attentional model, either as a bias (Equation 177), or through linear interpolation (Equation 180).

References

[1] Philip Arthur, Graham Neubig, and Satoshi Nakamura. Incorporating discrete translation lexicons into neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

[2] Michael Auli and Jianfeng Gao. Decoder integration and expected bleu training for recurrent neural network language models. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 136–142, 2014.

[3] Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1370–1380, 2014.

[4] C Lee Giles, Clifford B Miller, Dong Chen, Hsing-Hen Chen, Guo-Zheng Sun, and Yee-Chun Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405, 1992.

[5] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1631–1640, 2016.

[6] Joël Legrand, Michael Auli, and Ronan Collobert. Neural network-based word alignment through score aggregation. In *Proceedings of the First Conference on Machine Translation*, pages 66–73, Berlin, Germany, August 2016. Association for Computational Linguistics.

[7] Tao Lei, Regina Barzilay, and Tommi Jaakkola. Rationalizing neural predictions. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 107–117, 2016.

[8] Chu-Cheng Lin, Waleed Ammar, Chris Dyer, and Lori Levin. Unsupervised pos induction with word embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 1311–1316, 2015.

[9] Xunying Liu, Yongqiang Wang, Xie Chen, Mark JF Gales, and Philip C Woodland. Efficient lattice rescoring using recurrent neural network language models. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 4908–4912. IEEE, 2014.

[10] Graham Neubig and Chris Dyer. Generalizing and hybridizing count-based and neural language models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

[11] Graham Neubig, Makoto Morishita, and Satoshi Nakamura. Neural reranking improves subjective quality of machine translation: NAIST at WAT2015. 2016.

[12] Franz Josef Och, Daniel Gildea, Sanjeev Khudanpur, Anoop Sarkar, Kenji Yamada, Alex Fraser, Shankar Kumar, Libin Shen, David Smith, Katherine Eng, Viren Jain, Zhen Jin, and Dragomir Radev. A smorgasbord of features for statistical machine translation. In *Proceedings of the 2004 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, pages 161–168, 2004.

[13] Pushpendre Rastogi, Ryan Cotterell, and Jason Eisner. Weighting finite-state transductions with neural context. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 623–633, 2016.

[14] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *Proceedings of the 28th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 3104–3112, 2014.

[15] Akihiro Tamura, Taro Watanabe, and Eiichiro Sumita. Recurrent neural networks for word alignment model. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1470–1480, 2014.

[16] Yaohua Tang, Fandong Meng, Zhengdong Lu, Hang Li, and Philip L. H. Yu. Neural machine translation with external phrase memory. *CoRR*, abs/1606.01792, 2016.

[17] Ke M. Tran, Yonatan Bisk, Ashish Vaswani, Daniel Marcu, and Kevin Knight. Unsupervised neural hidden markov models. In *Proceedings of the Workshop on Structured Prediction for NLP*, pages 63–71, 2016.

[18] Taro Watanabe, Hajime Tsukada, and Hideki Isozaki. Left-to-right target generation for hierarchical phrase-based translation. In *Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 777–784, 2006.

[19] Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. Learning to compose words into sentences with reinforcement learning. *arXiv preprint arXiv:1611.09100*, 2016.