Chapter 6

Refinements

The previous section gave a comprehensive description of the currently most commonly used basic neural translation model architecture. It performs fairly well out of the box for many language pairs. Since its conception, a number of refinements have been proposed. We will describe them in this section.

Some of the refinements are fairly general, some target particular use cases or data conditions. To given one example, the best performing system at the recent WMT 2017 evaluation campaign used ensemble decoding (Section 6.1), byte pair encoding to address large vocabularies (Section 6.2), added synthetic data derived from monolingual target side data (Section 6.3) , and used deeper models (Section 6.4).

6.1 Ensemble Decoding

A common technique in machine learning is to not just build one system for your problem, but multiple ones and then combine them. This is called an **ensemble** of systems. It is such a successful strategy that various methods have been proposed to systematically build alternative systems, for instance by using different features or different subsets of the data. For neural networks, one straightforward way is to use different initializations or stop at different points in the training process.

Why does it work? The intuitive argument is that each system makes different mistakes. When two systems agree, then they are more likely both right, rather than both make the same mistake. One can also see the general principle at play in human behavior, such as setting up committees to make decisions or the democratic voting in elections.

Applying ensemble methods to our case of neural machine translation, we have to address two sub-problems: (1) generating alternate systems, and (2) combining their output.

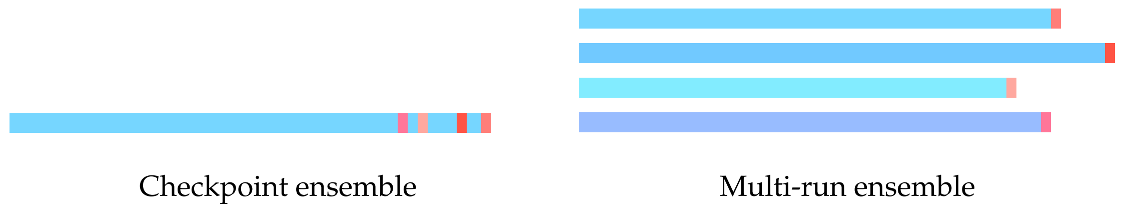


Figure 6.1: Two methods to generate alternative systems for ensembling: Checkpoint ensembling uses model dumps from various stages of the training process, while multi-run ensembling starts independent training runs with different initial weights and order of the training data.

6.1.1 Generating Alternative Systems

See Figure 6.1 for an illustration of two methods for the first sub-problem, generating alternative systems. When training a neural translation model, we iterate through the training data until some stopping criteria is met. This is typically a lack of improvements of the cost function applied to a validation set (measured in cross-entropy), or the translation performance on that validation set (measured in BLEU).

During training, we dump out the model at fixed intervals (say, every 10,000 iteration of batch processing). Once training is completed, we can look back at the performance of the model these different stages. We then pick the, say, 4 models with the best performance (typically translation quality measured in BLEU). This is called **checkpoint ensembling** since we select the models at different checkpoints in the training process.

Multi-run ensembling requires building systems in completely different training runs. As mentioned before, this can be accomplished by using different random initialization of weights, which leads training to seek out different local optima. We also randomly shuffle the training data, so using different random order will also lead to different training outcomes.

Multi-run ensembling usually works a good deal better, but it is also computationally much more expensive. Note that multi-run ensembling can also build on checkpoint ensembling. Instead of combining the end points of training, we first apply checkpoint ensembling to each run, and then combine those ensembles.

6.1.2 Combine System Output

Neural translation models allow the combination of several systems fairly deeply. Recall that the model first predicts a probability distribution over possible output words, and then commits to one of the words. This is where we combine the different trained models. Each model predicts a probability distribution and we then combine their predictions. The combination is done by simple averaging over the distributions. The averaged distribution is then the basis for selecting an output word.

See Figure 6.2 for an illustration. There may be some benefit to weighing the different systems differently, although in our way of generating them, they will all have very similar quality, so this is not typically done.

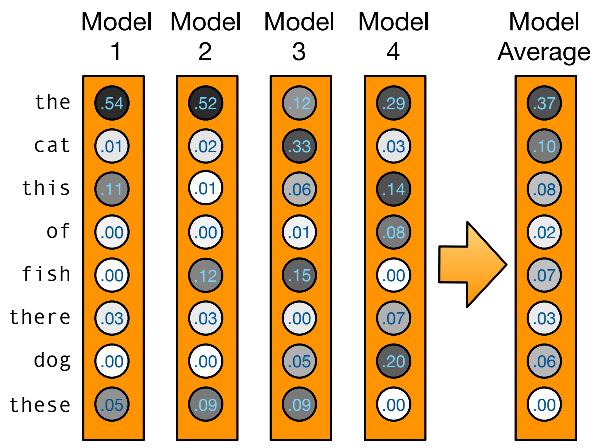


Figure 6.2: Combining predictions from a ensemble of models: Each model independently predicts a probability distribution over output words, which are averaged into a combined distribution.

6.1.3 Reranking with Right-to-Left Decoding

One more tweak on the idea of ensembling: Instead of building multiple systems with different random initialization, we can also build one set of system as before, and then a second set of system where we reverse the order of the output sentences. The second set of systems are called **right-to-left** systems, although arguably this is not a good name since it makes no sense for languages such as Arabic or Hebrew where the normal writing order is right to left.

The deep integration we described just above does not work anymore for the combination of left-to-right and right-to-left systems, since they produce output in different order. So, we have to resort to **reranking**. This involves several steps:

- Use an ensemble of left-to-right systems to generate an n-best list of candidate translations for each input sentence.
- Score each candidate translation with the individual left-to-right and right-to-left systems.
- Combine the scores (simple average) of the different models for each candidate, select the candidate with the best score for each input sentence.

Scoring a given candidate translation with a right-to-left system does require the require **forced decoding**, a special mode of running inference on an input sentence, but predicting a given output sentence. This mode is actually much closer to training (where also an output translation is given) the regular inference.

6.2 Large Vocabularies

Zipf's law tells us that words in a language are very unevenly distributed. So, there is always a large tail of rare words. New words come into the language all the time (e.g., *retweeting*, *website*, *woke*), and we also have to deal with a very large inventory of names, including company names (e.g., *eBay, Yahoo, Microsoft*).

Neural methods are not well equipped to deal with such large vocabularies. The ideal representations for neural networks are continuous space vectors. This is why we first convert discrete objects such as words into such word embeddings.

However, ultimately the discrete nature of words shows up. On the input side, we need to train an embedding matrix that maps each word into its embedding. On the output side we predict a probability distribution over all output words. The latter is generally the bigger concern, since the amount of computation involved is linear with the size of the vocabulary, making this a very large matrix operation.

Hence, neural translation models typically restrict the vocabulary to, say, 20,000 to 80,000 words. In initial work on neural machine translation, only the most frequent words were used, and all others represented by a *unknown* or *other* tag. The translation of these rare words was handled with a back-off dictionary.

The more common approach today is to break up rare words into **subword units**. This may seem a bit crude but is actually very similar to standard approaches in statistical machine translation to handle compounds (recall *website* $\rightarrow$ *web* + *site*) and morphology (*unfollow* $\rightarrow$ *un* + *follow*, *convolutions* $\rightarrow$ *convolution* + *s*). It is even a decent approach to the problem of transliteration of names which are traditionally handled by a sub-modular letter translation component.

A popular method to create an inventory of subword units and legitimate words is **byte pair encoding**. This method is trained on the parallel corpus. First, the words in the corpus are split into characters (marking original spaces with a special space character). Then, the most frequent pair of characters is merged (in English, this may be *t* and *h* into *th*). This step is repeated for a fixed given number of times. Each of these steps increases the vocabulary by one, beyond the original inventory of single characters.

The example mirrors quite well the behavior of the algorithm on real-world data sets. It starts with grouping together with frequent letter combinations (*e+r*, *t+h*, *c+h*) and then joins frequent words (*the*, *in*, *of*). At the end of this process, the most frequent words will emerge as single tokens, while rare words consist of still un-merged subwords. See Figure 6.3 for an example, where subword units are indicated with two "at" symbols (@@). After 49,500 byte pair encoding operations, the vast majority of words are intact, while rarer words are broken up (e.g., *critic@@ ises, destabil@@ ising*). Sometimes, the split seem to be morphologically motivated (e.g., *im@@ pending*), but mostly they are not (e.g., *stra@@ ined*). Note also the decomposition of the relatively rare name *Net@@ any@@ ahu*.

Further Readings A significant limitation of neural machine translation models is the computational burden to support very large vocabularies. To avoid this, typically the vocabulary is reduced to a shortlist of, say, 20,000 words, and the remaining tokens are replaced with the unknown word token "UNK". To translate such an unknown word, Luong et al. (2015c); Jean et al. (2015a) resort to a separate dictionary. Arthur et al. (2016) argue that neural translation models are worse for rare words

Obama receives Net@@ any@@ ahu

the relationship between Obama and Net@@ any@@ ahu is not exactly friendly . the two wanted to talk about the implementation of the international agreement and about Teheran 's destabil@@ ising activities in the Middle East . the meeting was also planned to cover the conflict with the Palestinians and the disputed two state solution . relations between Obama and Net@@ any@@ ahu have been stra@@ ined for years . Washington critic@@ ises the continuous building of settlements in Israel and acc@@ uses Net@@ any@@ ahu of a lack of initiative in the peace process . the relationship between the two has further deteriorated because of the deal that Obama negotiated on Iran 's atomic programme . in March , at the invitation of the Republic@@ ans , Net@@ any@@ ahu made a controversial speech to the US Congress , which was partly seen as an aff@@ ront to Obama . the speech had not been agreed with Obama , who had rejected a meeting with reference to the election that was at that time im@@ pending in Israel .

Figure 6.3: Byte pair encoding (BPE) applied to English (model used 49,500 BPE operations). Word splits are indicated with @@. Note that the data is also tokenized and true-cased.

and interpolate a traditional probabilistic bilingual dictionary with the prediction of the neural machine translation model. They use the attention mechanism to link each target word to a distribution of source words and weigh the word translations accordingly.

Source words such as names and numbers may also be directly copied into the target. Gulcehre et al. (2016) use a so-called switching network to predict either a traditional translation operation or a copying operation aided by a softmax layer over the source sentence. They preprocess the training data to change some target words into word positions of copied source words. Similarly, Gu et al. (2016) augment the word prediction step of the neural translation model to either translate a word or copy a source word. They observe that the attention mechanism is mostly driven by semantics and the language model in the case of word translation, but by location in case of copying.

To speed up training, Mi et al. (2016) use traditional statistical machine translation word and phrase translation models to filter the target vocabulary for mini batches.

Sennrich et al. (2016d) split up all words to sub-word units, using character n-gram models and a segmentation based on the byte pair encoding compression algorithm.

6.3 Using Monolingual Data

A key feature of statistical machine translation system are language models, trained on very large monolingual data set. The larger the language models, the higher translation quality. Language models trained on up to a trillion words crawled from the general web have been used. So, it is a surprise that the basic neural translation model does not use any additional monolingual data, its language model aspect (the conditioning of the previous hidden decoder state and the previous output) is trained jointly with the translation model aspect (the conditioning on the input context).

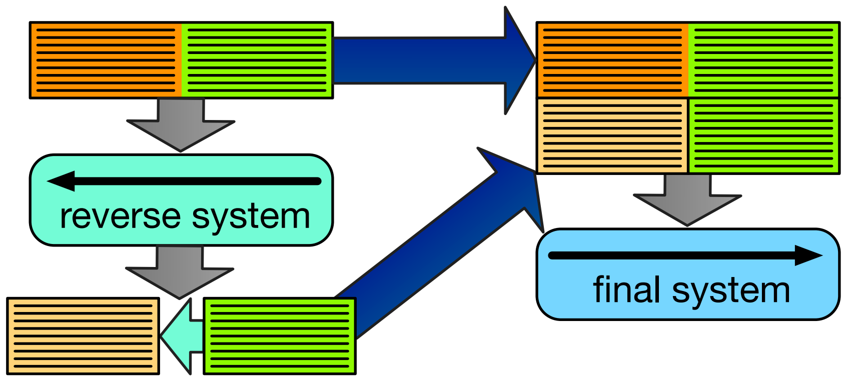


Figure 6.4: Creating synthetic parallel data from target-side monolingual data: (1) train a system in reverse order, (2) use it to translate target-side monolingual data into the source language, (3) combine the generated synthetic parallel data with the true parallel data in final system building.

Two main ideas have been proposed to improve neural translation models with monolingual data. One is to transform additional monolingual translation into parallel data by synthesizing the missing half of the data, and the other is to integrate a language model as a component into the neural network architecture.

6.3.1 Back Translation

Language models improve fluency of the output. Using larger amounts of monolingual data in the target language give the machine more evidence what are common sequences of words and what are not.

We cannot use monolingual target side data in our neural translation model training, since it is missing the source side. So, one idea is to just synthesize this data by **back translation**. See Figure 6.4 for an illustration of the steps involved.

- Train a reverse system that translates from the intended target language into the source language. We typically use the same neural machine translation setup for this as for our final system, just with source and target flipped. But we may use any system, even traditional phrase-based systems.

- Use the reverse system to translate target side monolingual data, creating a **synthetic parallel corpus**.

- Combine the generated synthetic parallel data with the true parallel data when building the final system.

There is an open question on how much synthetic parallel data should be used in relation to the amount of existing true parallel data. Typically, there are magnitudes more monolingual data available, but we also do not want to drown out the actual real data. Successful applications of this idea used equal amounts of synthetic and true data. We may also generate much more synthetic parallel data, but then ensure during training that we process equal amounts of each by over-sampling the true parallel data.

6.3.2 Adding a Language Model

The other idea is to train a language model as a separate component of the neural translation model. Gülçehre et al. (2015) first train the large language model as a recurrent neural network on all available data, including the target side of the parallel corpus. Then, they add this language model to the neural translation model. Since both language model and translation model predict output words, the natural point to connect the two models is joining them at that output prediction node in the network by concatenating their conditioning contexts.

We expand Equation 5.3 to add the hidden state of the neural language model s_i^{LM} to the hidden state of the neural translation model s_i^{TM}, the source context c_i and the previous English word e_{i-1}.

$$e_i = g(c_i, s_i^{\text{TM}}, s_i^{\text{LM}}, e_{i-1}) \tag{6.1}$$

When training the combined model, we leave the parameters of the large neural language model unchanged, and update only the parameters of the translation model and the combination layer. The concern is that otherwise the output side of the parallel corpus would overwrite the memory of the large monolingual corpus. In other words, the language model would overfit to the parallel training data and be less general.

One final question remains: How much weight should be given to the translation model and how much weight should be given to the language model? The above equation considers them in all instances the same way. But there may be output words for which the translation model is more relevant (e.g., the translation of content words with distinct meaning) and output words where the language model is more relevant (e.g., the introduction of relevant function words for fluency).

The balance of the translation model and the language model can be achieved with the type of gated units that we encountered in our discussion of the long short-term memory neural network architecture (Section 4.5). Such a gated unit may be predicted solely from the language model state s_i^{LM} and then used as a factor that is multiplied with that language model state before it is used in the prediction of Equation 6.1.

$$\begin{aligned}
\text{gate}_i^{\text{LM}} &= f(s_i^{\text{LM}}) \\
\bar{s}_i^{\text{LM}} &= \text{gate}_i^{\text{LM}} \times s_i^{\text{LM}} \\
e_i &= g(c_i, s_i^{\text{TM}}, \bar{s}_i^{\text{LM}}, e_{i-1})
\end{aligned} \tag{6.2}$$

6.3.3 Round Trip Training

Looking at the backtranslation idea from a strict machine learning perspective, we can see two learning objectives. There is the objective to learn the transformations given by the parallel data, as done traditionally. Then, there is the goal to learn how to convert an output lamguage

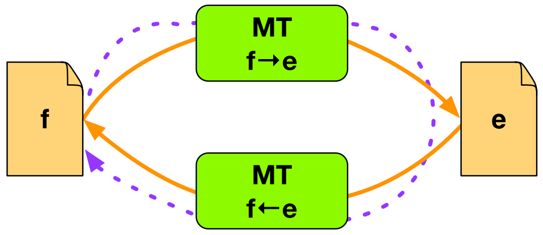


Figure 6.5: Round trip training: In addition to training two models $f \rightarrow e$ and $e \rightarrow f$, as done traditionally on parallel data, we also optimize both models to convert a sentence f into e and then restore it back into f, using on monolingual data in f. We may add a corresponding round trip starting with e.

sentence into the input language, and then back into the output language with the objective to match the traditional sentence. A good machine translation model should be able to preserve the meaning of the output language sentence when mapped into the input language and back.

See Figure 6.5 for an illustration. There are two machine translation models. One that translates sentences in the language direction $f \rightarrow e$, the other in the opposite direction $e \rightarrow f$. These two systems may be trained with traditional means, using a parallel corpus. We can also **round trip** a sentence f first through the $f \rightarrow e$ and then back through $e \rightarrow f$

In this scenario, there are two objectives for model training.

- The translation $\mathbf{e'}$ of the given monolingual sentence $\mathbf{f}$ should be a valid sentence in the language e, as measured with a language model $\text{LM}_e(\mathbf{e'})$.
- The reconstruction of the translation $\mathbf{e'}$ back into the original language f should be easy, as measured with the translation model $\text{MT}_{e \rightarrow f}(\mathbf{f}|\mathbf{e'})$

These two objectives can be used to update model parameters in both translation models $\text{MT}_{f \rightarrow e}$ and $\text{MT}_{e \rightarrow f}$.

Typical model update is driven by correct predictions of each word. In this round-trip scenario, the translation $\mathbf{e'}$ has to be computed first, before we can do the usual training of model $\text{MT}_{e \rightarrow f}$ with the given sentence pair ($\mathbf{e'}$,$\mathbf{f}$). To make better use of the training data, a n-best list of translations $\mathbf{e'}_1, ..., \mathbf{e'}_n$ is computed and model updates are computed for each of them.

We can also update the model $\text{MT}_{f \rightarrow e}$ with monolingual data in language f by scaling updates by the language model cost $\text{LM}_e(\mathbf{e'}_i)$ and the forward translation cost $\text{MT}_{f \rightarrow e}(\mathbf{e'}_i|\mathbf{f})$ for each of the translations $\mathbf{e'}_i$ in the n-best list.

To use monolingual data in language e, training is done in the reverse round trip direction. For details of this idea, refer to Xia et al. (2016).

Further Readings Sennrich et al. (2016c) back-translate the monolingual data into the input language and use the obtained synthetic parallel corpus as additional training data. Xia et al. (2016) use monolingual data in a dual learning setup. Machine translation engines are trained in both directions, and in addition to regular model training from parallel data, monolingual data is translated in a round trip (e to f to e) and evaluated with a language model for language f and reconstruction match back to e as cost function to drive gradient descent updates to the model.

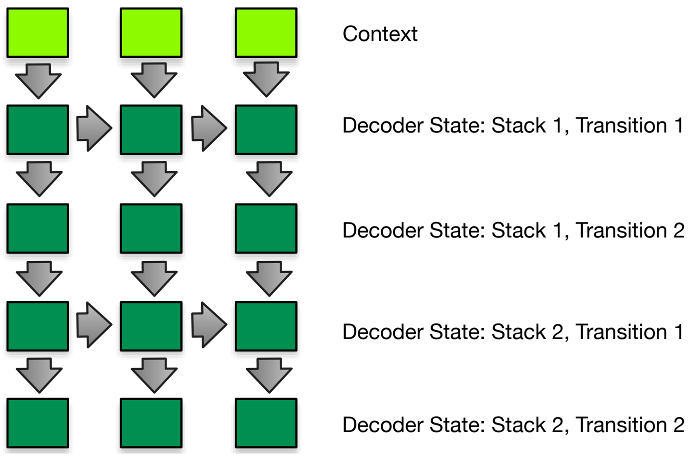

Context
Decoder State: Stack 1, Transition 1
Decoder State: Stack 1, Transition 2
Decoder State: Stack 2, Transition 1
Decoder State: Stack 2, Transition 2

Figure 6.6: Deep Decoder: Instead of a single recurrent neural network (RNN) layer for the decoder state, in a deep model, it consists of several layers. The illustrations shows a combination of a deep transition and stacked RNNs. It omits the word prediction, word selection and output word embedding steps which are identical to the original architecture, shown in Figure 5.3 on page 56.

6.4 Deep Models

Learning the lessons from other research fields such as vision or speech recognition, recent work in machine translation has also looked at deeper models. Simply put, this involves adding more intermediate layers into the baseline architecture.

The core components of neural machine translation are the encoder that takes input words and converts them into a sequence of contextualized representations and the decoder that generates a output sequence of words. Both are recurrent neural networks.

Recall that we already discussed how to build deeper recurrent neural networks for language modelling (refer back to Section 4.7 on page 50). We now extend these ideas to the recurrent neural networks in the encoder and the decoder.

What all these recurrent neural networks have in common is that they process an input sequence into an output sequence, and at each time step t information from a new input x_t is combined with the hidden state from the previous time step h_{t-1} to predict a new hidden state h_t. From that hidden state additional predictions may be made (output words y_t in the case of the decoder, the next word in the sequence in the case of language models), or the hidden state is used otherwise (via the attention mechanism in case of the encoder).

6.4.1 Decoder

See Figure 6.6 for part of the decoder in neural machine translation, using a particular deeper architecture. We see that instead of a single hidden state h_t for a given time step t, we now have a sequence of hidden states $h_{t,1}$, $h_{t,2}$, ..., $h_{t,I}$ for a given time step t.

There are various options how the hidden states may be connected. Previously, in Section 4.7 we presented two ideas. (1) In stacked recurrent neural networks where a hidden state

$h_{t,i}$ is conditioned on the hidden state from a previous layer $h_{t,i-1}$ and the hidden state at the same depth from a previous time step $h_{t-1,i}$. (2) In deep transition recurrent neural networks, the first hidden state $h_{t,1}$ is conditioned on the last hidden state from the previous time step $h_{t-1,I}$ and the input, while the other hidden layers $h_{t,I}$ $(i > 1)$ are just conditioned on the previous previous layer $h_{t,i-1}$.

Figure 6.6 combines these two ideas. some layers are both stacked (conditioned on the previous time step $h_{t-1,I}$ and previous layer $h_{t,i-1}$), while others are deep transitions (conditioned only on the previous layer $h_{t,i-1}$.

Mathematically, we can break this out into the stacked layers $h_{t,i}$:

$$\begin{aligned} h_{t,1} &= f_1(x_t, h_{t-1,1}) \\ h_{t,i} &= f_i(h_{t,i-1}, h_{t-1,i}) \quad \text{for } i > 1 \end{aligned} \tag{6.3}$$

and the deep transition layers $v_{i,i,j}$.

$$\begin{aligned} v_{t,i,1} &= g_{i,1}(\text{in}_{t,i}, h_{t-1,i}) \quad && \text{in}_{t,i} \text{ is either } x_t \text{ or } h_{t,i-1} \\ v_{t,i,j} &= g_{i,j}(v_{t,i,j-1}) && \text{for } j > 1 \\ h_{t,i} &= v_{t,i,J} \end{aligned} \tag{6.4}$$

The function $f_i(h_{t,i-1}, h_{t-1,i})$ is computed as a sequence of function calls $g_{i,j}$. Each of the functions $g_{i,j}$ may be implemented as feed-forward neural network layer (matrix multiplication plus activation function), long-short term memory cell (LSTM), or gated recurrent unit (GRU). On either case, each function $g_{i,j}$ has its own set of trainable model parameters.

6.4.2 Encoder

Deep recurrent neural networks for the encoder may draw in the same ideas as the decoder, with one addition: in the baseline neural translation model, we used bidirectional recurrent neural networks to condition on both left and right context. We want to do the same for any deep version of the encoder.

Figure 6.7 shows one idea how this could be done, called **alternating recurrent neural network**. It looks basically like a stacked recurrent neural network, with one twist: the hidden states at each layer $h_{t,i}$ are alternately conditioned on the hidden state from the previous time step $h_{t-1,i}$ or the next time step $h_{t+1,i}$.

Mathematically, we formulate this as even numbered hidden states $h_{t,2i}$ being conditioned on the left context $h_{t-1,2i}$ and odd numbered hidden states $h_{t,2i+1}$ conditioned on the right context $h_{t+1,2i}$.

$$\begin{aligned} h_{t,1} &= f(x_t, h_{t-1,1}) \\ h_{t,2i} &= f(h_{t,2i-1}, h_{t-1,2i}) \\ h_{t,2i+1} &= f(h_{t,2i}, h_{t+1,2i+1}) \end{aligned} \tag{6.5}$$

As before in the encoder, we can extend this idea by having deep transitions.

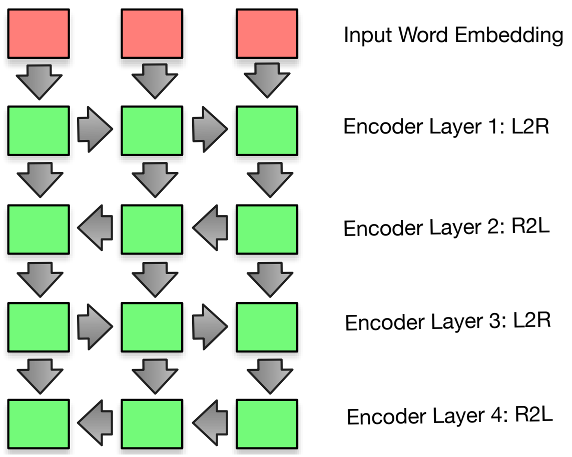


Figure 6.7: Deep alternating encoder: combination of the idea of a bidirectional recurrent neural network previously proposed for neural machine translation (recall Figure 5.2 on page 55) and the stacked recurrent neural network (recall Figure 5.2 on page 55). This architecture may be further extended with the idea of deep transitions, as shown for the decoder (previous Figure 6.6).

Note that deep models are typically augmented with direct connections from the input to the output. In the case of the encoder, this may mean a direct connection from the embedding to the final encoder layer, or connections at each layer that pass the input directly to the output. Such **residual connections** help with training. In early stages, the deep architecture can be skipped. Only when a basic functioning model has been acquired, the deep architecture can be exploited to enrich it. We typically see the benefits of residual connections in early training stages (faster initial reduction of model perplexity), and less so as improvement in the final converged model.

Further Readings Recent work has shown good results with 4 stacks and 2 deep transitions each for encoder and decoder, as well as alternating networks for the encoder (Miceli Barone et al., 2017). There are a large number of variations (including the use of skip connections, the choice of LSTM vs. GRU, number of layers of any type) that still need to be explored empirical for various data conditions.

6.5 Guided Alignment Training

The attention mechanism in neural machine translation models is motivated by the need to align output words to input words. Figure 6.8 shows an example of attention weights given to English input words for each German output word during the translation of a sentence.

The attention values typically match up pretty well with word alignment used in traditional statistical machine translation, obtained with tools such as GIZA++ or fast-align which implement variants of the IBM Models.

There are several good uses for word alignments beyond their intrinsic value of improving the quality of translations. For instance in the next section, we will look at using the attention

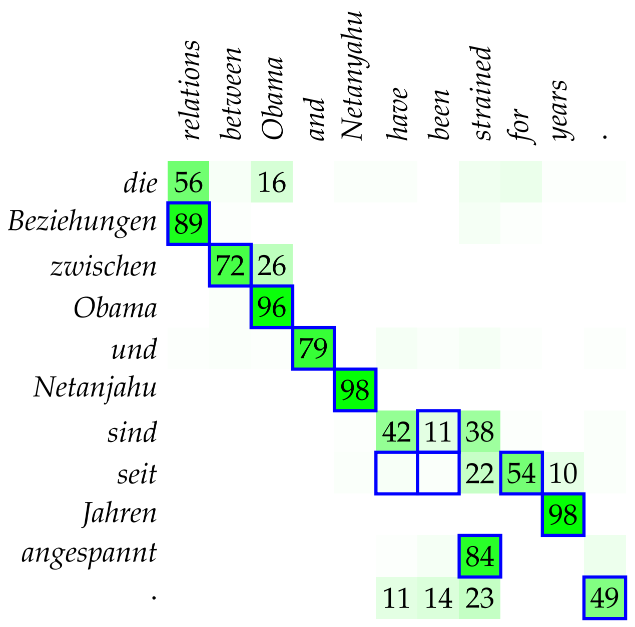


Figure 6.8: Alignment vs. Attention: In this example, alignment points from traditional word alignment methods are shown as squares, and attention states as shaded boxes depending on the alignment value (shown as percentage). They generally match up well, but note for instance that the prediction of the output auxiliary verb *sind* pays attention to the entire verb group *have been strained*.

mechanism to explicitly track coverage of the input. We may also want to override preferences of the neural machine translation model with pre-specified translations of certain terminology or expressions such as numbers, dates, or measurements that are better handled by rule-based components; this requires to know when the neural model is about to translate a specific source word. But also the end user may be interested in alignment information, such as translators using machine translation in a computer aided translation tool may want to check where an output word originates from.

Hence, instead of trusting the attention mechanism to implicitly acquire the role as word alignmer, we may enforce this role. The idea is to provide not just the parallel corpus as training data, but also pre-computed word alignments using traditional means. Such additional information may even benefit training of models to converge faster or overcome data sparsity under low resource conditions.

A straightforward way to add such given word alignment to the training process is to not change the model at all, but to just modify the training objective. Typically, the goal of training neural machine translation models is to generate the correct output words. We can add to this goal to also match the given word alignment.

Formally, we assume to have access to an alignment matrix A that specifies alignment points A_{ij} input words j and output words i in a way that $\sum_j A_{ij} = 1$, i.e., each output word's alignment scores add up to 1. The model estimates attention scores α_{ij} that also add up to 1 for each output word: $\sum_j \alpha_{ij} = 1$ (recall Equation 5.7 on page 58). The mismatch between given alignment scores A_{ij} and computed attention scores α_{ij} can be measured in several ways, such as cross entropy

$$\text{cost}_{\text{CE}} = -\frac{1}{I}\sum_{i=1}^{I}\sum_{j=1}^{J} A_{ij} \log \alpha_{ij} \tag{6.6}$$

or mean squared error

$$\text{cost}_{\text{MSE}} = -\frac{1}{I}\sum_{i=1}^{I}\sum_{j=1}^{J} (A_{ij} - \alpha_{ij})^2 \tag{6.7}$$

This cost is added to the training objective and may be weighted.

Further Readings Chen et al. (2016b); Liu et al. (2016) add supervised word alignment information (obtained with traditional statistical word alignment methods) to training. They augment the objective function to also optimize matching of the attention mechanism to the given alignments.

6.6 Modeling Coverage

One impressive aspect of neural machine translation models is how well they are able to translate the entire input sentence, even when a lot of reordering is involved. But this as aspect is not perfect, occasionally the model translates some input words multiple times, and sometimes it misses to translate them.

See Figure 6.9 for an example. The translation has two flaws related to mis-allocation of attention. The beginning of the phrase *"Social Housing" alliance* receives too much attention, resulting in a faulty translation with hallucinated words: *das Unternehmen der Gesellschaft für soziale Bildung*, or *the company of the society for social education*. At the end of the input sentence, the phrase *a fresh start* does not receive any attention and is hence untranslated in the output.

Hence, an obvious idea is to more strictly model **coverage**. Given the attention model, a reasonable way to define coverage is by adding up the attention states. In a complete sentence translation, we roughly expect that each input word receives a similar amount of attention. If some input words never receive attention or too much attention that signals a problem with the translation.

6.6.1 Enforcing Coverage during Inference

We may restrict the enforcing of proper coverage to the decoder. When considering multiple hypothesis in beam search, then we should discourage the ones that pay too much attention to some input words. And, once hypotheses are completed, we can penalize those that paid only little attention to some of the input. There are various ways to come up with scoring functions for over-generation and under-generation.

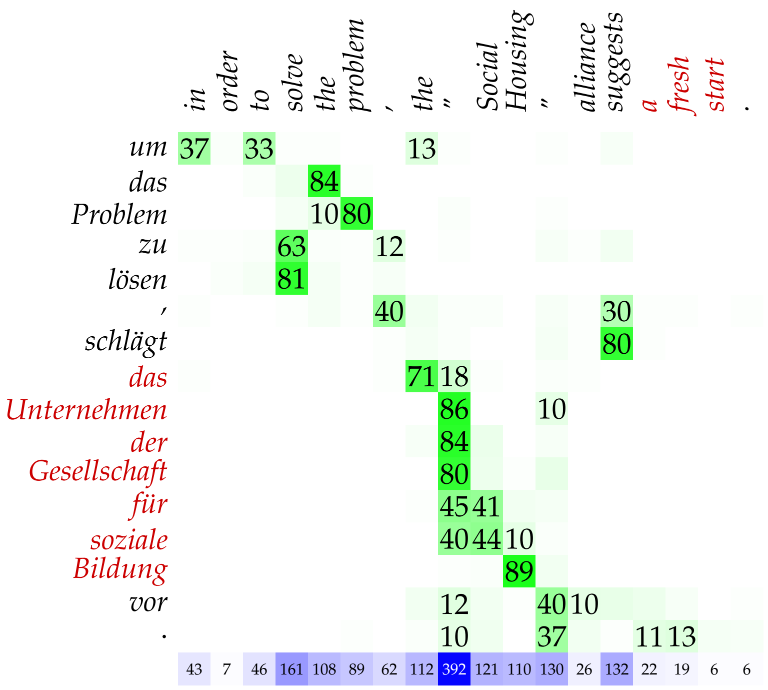


Figure 6.9: Example for over-generation and under-generation: the input tokens around *Social Housing* are attended too much, leading to hallucinated output words (*das Unternehmen*, English: *the company*), while the end of the sentence *a fresh start* is not attended and untranslated.

$$
\begin{aligned}
\text{coverage}(j) &= \sum_i \sum_k \alpha_{i,k} \\
\text{over-generation} &= \max\Big(0, \sum_j \text{coverage}(j) - 1\Big) \\
\text{under-generation} &= \min\Big(1, \sum_j \text{coverage}(j)\Big)
\end{aligned}
\tag{6.8}
$$

The use of multiple scoring functions in the decoder is common practice in traditional statistical machine translation. For now, it is not in neural machine translation. A challenge is to give proper weight to the different scoring functions. If there are only two or three weights, these can be optimized with grid search over possible values. For more weights, we may borrow methods such as MERT or MIRA from statistical machine translation.

6.6.2 Coverage Models

The vector that accumulates coverage of input words may be directly used to inform the attention model. Previously, the attention given to a specific input word j was conditioned on the previous state of the decoder s_{i-1} and the representation of the input word h_j. Now, we also add as conditioning context the accumulated attention given to the word (compare to Equation 5.6 on page 57).

$$a(s_{i-1}, h_j) = W^a s_{i-1} + U^a h_j + V^a \text{coverage}(j) + b^a \tag{6.9}$$

Coverage tracking may also integrated into the training objective. Taking a page from the guided alignment training (recall the previous Section 6.5), we augment the training objective function with a coverage penalty with some weight λ.

$$\log \sum_i P(y_i|x) + \lambda \sum_j (1 - \text{coverage}(j))^2 \tag{6.10}$$

Note that in general, it is problematic to add such additional functions to the learning objective, since it does distract from the main goal of producing good translations.

6.6.3 Fertility

So far, we described coverage as the need to cover all input words roughly evenly. However, even the earliest statistical machine translation models considered the **fertility** of words, i.e., the number of output words that are generated from each input word. Consider the English *do not* construction: most other language do not require an equivalent of *do* when negating a verb. Meanwhile, other words are translated into multiple output words. For instance, the German *natürlich* may be translated as *of course*, thus generating 2 output words.

We may augment models of coverage by adding a fertility components that predicts the number of output words for each input words. Here one example for a model that predicts the fertility Φ_j for each input word, and uses it to normalize the coverage statistics.

$$\begin{aligned} \Phi_j &= N\sigma(W_j h_j) \\ \text{coverage}(j) &= \frac{1}{\Phi_j} \sum_i \sum_k \alpha_{i,k} \end{aligned} \tag{6.11}$$

Fertility Φ_j is predicted with a neural network layer that is conditioned on the input word representation h_j and uses a sigmoid activation function (thus resulting in values from 0 to 1), which is scaled to a pre-defined maximum fertility of N.

6.6.4 Feature Engineering versus Machine Learning

The work on modeling coverage in neural machine translation models is a nice example to contrast between the engineering approach and the belief in generic machine learning techniques. From an engineering perspective, a good way to improve a system is to analyze its performance, find weak points and consider changes to overcome them. Here, we notice over-generation and under-generation with respect to the input, and add components to the model to overcome this problem. On the other hand, proper coverage is one of the features of a good translation that machine learning should be able to get from the training data. If it is not able to do that, it may need deeper models, more robust estimation techniques, ways to fight over-fitting or under-fitting, or other adjustments to give it just the right amount of power needed for the problem.

It is hard to carry out the analysis needed to make generic machine learning adjustments, given the complexity of a task like machine translation. Still, the argument for deep learning is that it does not require feature engineering, such as adding coverage models. It remains to be seen how neural machine translation evolves over the next years, and if it moves more into a engineering or machine learning direction.

Further Readings To better model coverage, Tu et al. (2016b) add coverage states for each input word by either (a) summing up attention values, scaled by a fertility value predicted from the input word in context, or (b) learning a coverage update function as a feed-forward neural network layer. This coverage state is added as additional conditioning context for the prediction of the attention state. Feng et al. (2016) condition the prediction of the attention state also on the previous context state and also introduce a coverage state (initialized with the sum of input source embeddings) that aims to subtract covered words at each step. Similarly, Meng et al. (2016) separate hidden states that keep track of source coverage and hidden states that keep track of produced output. Cohn et al. (2016) add a number of biases to model coverage, fertility, and alignment inspired by traditional statistical machine translation models. They condition the prediction of the attention state on absolute word positions, the attention state of the previous output word in a limited window, and coverage (added attention state values) over a limited window. They also add a fertility model and add coverage in the training objective.

6.7 Adaptation

Text may differ by style, topic, degree of formality, and so on. A common problem in the practical development of machine translation systems is that most of the available training data is different from the data relevant to a chosen use case. For instance, if your goal is to translate chat room dialogs, you will realize that there is very little translated chat room data available. There are massive quantities of official publications from international organizations, random translations crawled from the web, and maybe somewhat relevant movie subtitle translations.

This problem is generally framed as a problem of **domain adaptation**. In the simplest form, you have one set of data relevant to your use case — the **in-domain data** — and another set that is less relevant — the **out-of-domain data**.

In traditional statistical machine translation, a vast number of methods for domain adaptation have been proposed. Models may be interpolated, we may back-off from in-domain to out-of-domain models, we may over-sample in-domain data during training or sub-sample out-of-domain data, etc.

For neural machine translation, a fairly straightforward method is currently the most popular (see Figure 6.10). This method divides training up into two stages. First, we train the model on all available data until convergence. Then, we run a few more iterations of training on the in-domain data only and stop training when performance on the in-domain validate set

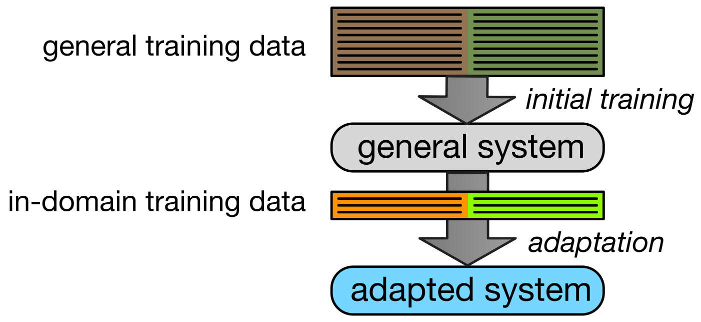


Figure 6.10: Online training of neural machine translation models allows a straightforward domain adaptation method: Having a general domain translation system trained on general-purpose data, a handful of additional training epochs on in-domain data allows for a domain-adapted system.

peaks. This way, the final model benefits from all the training data, but is still specialized to the in-domain data.

Practical experience with this method shows that the second in-domain training stage may converge very quickly. The amount if in-domain data is typically relatively small, and only a handful of training epochs are needed.

Another, less commonly used method draws on the idea of ensemble decoding (Section 6.1). If we train separate models on different sets of data, we may combine their predictions, just as we did for ensemble decoding. In this case, we do want to choose weights for each model, although how to choose these weights is not a trivial task. If there is just an in-domain and out-of-domain model, however, this may be simply done by line search over possible values.

Let us now look at a few special cases that arise in practical use.

6.7.1 Subsample in-domain data from large collections

A common problem is that the amount of available in-domain data is very small, so just training on this data, even in a secondary adaptation stage, risks overfitting — very good performance on the seen data but poor performance on everything else.

Large random collections of parallel text often contain data that closely matches the in-domain data. So, we may want to extract this in-domain data from the large collections of mainly out-of-domain data. The general idea behind a variety of methods is to build two detectors: one in-domain detector trained on in-domain data, and one out-of-domain detector trained on out-of-domain data. We then score each sentence pair in the out-of-domain data with both detectors and select sentence pairs that are preferred (or judged relatively relevant) by the in-domain detector.

The classic detectors are language models trained on the source and target side of the in-domain and out-of-domain data, resulting in a total of 4 language models: the source side in-domain model LM_f^{in}, the target side in-domain model LM_e^{in}, the source side out-of-domain model LM_f^{out}, and the target side out-of-domain model LM_e^{out}. Any given sentence pair from the out-of-domain data is then scored based on these models:

$$\text{relevance}_{e,f} = \Big(\text{LM}_e^{\text{in}}(e) - \text{LM}_e^{\text{out}}(e)\Big) + \Big(\text{LM}_f^{\text{in}}(f) - \text{LM}_f^{\text{out}}(f)\Big) \tag{6.12}$$

We may use traditional n-gram language models or neural recurrent language models. Some work suggests to replace open class words (nouns, verbs, adjectives, adverbs) with part-of-speech tags or word clusters. More sophisticated models not only consider domain-relevance but noisiness of the training data (e.g., misaligned or mistranslated data). We may even use in-domain and out-of-domain neural translation models to score sentence pairs instead of source and target side sentences in isolation.

The subsampled data may be used in several ways. We may only train on this data to build our domain-specific system. Or, we use it in a secondary adaptation stage as outlined above.

6.7.2 Only monolingual in-domain data

What if we have no parallel data in the domain of our use case? Two main ideas have been explored. Firstly, we may still use the monolingual data, may it be in the source or target language or both, for subsampling parallel data from a large pile of general data, as outline above.

Another idea is to use existing parallel data to train an out-of-domain model, then back-translate out-of-domain data (recall Section 6.3.1) to generate a synthetic in-domain corpus, and then use this data to adapt the initial model. In traditional statistical machine translation, much adaptation success has been achieved with just interpolating the language model, and this idea is the neural translation equivalent to that.

6.7.3 Multiple domains

Sometimes, we have multiple collections of data that are clearly identified by domain — typically categories such as information technology, medical, law, etc. We can use the techniques described above to build specialized translation models for each of these domains.

For a given test sentence, we then select the appropriate model. If we do not know the domain of the test sentence, we first have to build a classifier that allows us to automatically make this determination. The classifier may be based on the methods for domain detectors described above. Given the decision of the classifier, we then select the most appropriate model.

But we do not have to commit to a single domain. The classifier may instead provide a distribution of relevance of the specific domain models (say, 50% domain A, 30% domain B, 20% domain C) which are then used as weights in an ensemble of domain-specific models.

The domain classification may done based on a whole document instead of each individual sentence, which brings in more context to make a more robust decision.

As a final remark, it is hard to give conclusive advice on how to handle adaptation challenges, since it is such a broad topic. The style of the text may be more relevant than its content.

Data may differ narrowly (e.g., official publications from the United Nations vs. official announcements from the European Union) or dramatically (e.g., chat room dialogs vs. published laws). The amounts of in-domain and out-of-domain data differs. The data may be cleanly separated by domain or just come in a massive disorganized pile. Some of the data may be of higher translation quality than other, which may be polluted by noise such as mistranslations, misalignments, or even generated by some other machine translation system.

Further Readings There is often a domain mismatch between the bulk (or even all) of the training data for a translation and its test data during deployment. There is rich literature in traditional statistical machine translation on this topic. A common approach for neural models is to first train on all available training data, and then run a few iterations on in-domain data only (Luong and Manning, 2015), as already pioneered in neural language model adaption (Ter-Sarkisov et al., 2015). Servan et al. (2016) demonstrate the effectiveness of this adaptation method with small in-domain sets consisting of as little as 500 sentence pairs.

Chu et al. (2017) argue that given small amount of in-domain data leads to overfitting and suggest to mix in-domain and out-of-domain data during adaption. Freitag and Al-Onaizan (2016) identify the same problem and suggest to use an ensemble of baseline models and adapted models to avoid overfitting. Peris et al. (2017) consider alternative training methods for the adaptation phase but do not find consistently better results than the traditional gradient descent training. Inspired by domain adaptation work in statistical machine translation on sub-sampling and sentence weighting, Chen et al. (2017) build an in-domain vs. out-of-domain classifier for sentence pairs in the training data, and then use its prediction score to reduce the learning rate for sentence pairs that are out of domain.

Farajian et al. (2017) show that traditional statistical machine translation outperforms neural machine translation when training general-purpose machine translation systems on a collection data, and then tested on niche domains. The adaptation technique allows neural machine translation to catch up.

A multi-domain model may be trained and informed at run-time about the domain of the input sentence. Kobus et al. (2016) apply an idea initially proposed by Sennrich et al. (2016a) - to augment input sentences for register with a politeness feature token - to the domain adaptation problem. They add a domain token to each training and test sentence. Chen et al. (2016b) report better results over the token approach to adapt to topics by encoding the given topic membership of each sentence as an additional input vector to the conditioning context of word prediction layer.

6.8 Adding Linguistic Annotation

One of the big debates in machine translation research is the question if the key to progress is to develop better, relatively generic, machine learning methods that implicitly learn the important features of language, or to use linguistic insight to augment data and models.

Recent work in statistical machine translation has demonstrated the benefits of linguistically motivated models. The best statistical machine translation systems in major evaluation campaigns for language pairs such as Chinese–English and German–English are syntax-based. While they translate sentences, they also build up the syntactic structure of the output sentence. There have been serious efforts to move towards deeper semantics in machine translation.

Words	*the*	*girl*	*watched*	*attentively*	*the*	*beautiful*	*fireflies*
Part of speech	DET	NN	ADV	VFIN	DET	JJ	NNS
Lemma	*the*	*girl*	*watch*	*attentive*	*the*	*beautiful*	*firefly*
Morphology	-	SING.	PAST	-	-	PLURAL	
Noun phrase	BEGIN	CONT	OTHER	OTHER	BEGIN	CONT	CONT
Verb phrase	OTHER	OTHER	BEGIN	CONT	CONT	CONT	CONT
Synt. dependency	*girl*	*watched*	-	*watched*	*fireflies*	*fireflies*	*watched*
Depend. relation	DET	SUBJ	-	ADV	DET	ADJ	OBJ
Semantic role	-	ACTOR	-	MANNER	-	MOD	PATIENT
Semantic type	-	HUMAN	VIEW	-	-	-	ANIMATE

Figure 6.11: Linguistic annotation of a sentence, formatted as word-level factored representation

The turn towards neural machine translation was at first hard swing back towards better machine learning while ignoring much linguistic insights. Neural machine translation views translation as a generic sequence to sequence task, which just happens to involve sequences of words in different languages. Methods such as byte pair encoding or character-based translation models even put the value of the concept of a word as a basic unit into doubt.

However, recently there have been also attempts to add linguistic annotation into neural translation models, and steps towards more linguistically motivated models. We will take a look at successful efforts to integrate (1) linguistic annotation to the input sentence, (2) linguistic annotation to the output sentence, and (3) build linguistically structured models.

6.8.1 Linguistic annotation of the input

One of the great benefits of neural networks is their ability to cope with rich context. In the neural machine translation models we presented, each word prediction is conditioned on the entire input sentence and all previously generated output words. Even if, as it is typically the case, a specific input sequence and partially generated output sequence has never been observed before during training, the neural model is able to generalize the training data and draw from relevant knowledge. In traditional statistical models, this required carefully chosen independence assumptions and back-off schemes.

So, adding more information to the conditioning context in neural translation models can be accommodated rather straightforwardly. First, what information would be like to add? The typical linguistic treasure chest contains part-of-speech tags, lemmas, morphological properties of words, syntactic phrase structure, syntactic dependencies, and maybe even some semantic annotation.

All of these can be formatted as annotations to individual input words. Sometimes, this requires a bit more work, such as syntactic and semantic annotation that spans multiple words. See Figure 6.11 for an example. To just walk through the linguistic annotation of the word *girl* in the sentence:

- Part of speech is NN, a noun.
- Lemma is *girl*, the same as the surface form. The lemma differs for *watched* / *watch*.
- Morphology is singular.
- The word is the continuation (CONT) of the noun phrase that started with *the*.
- The word is not part of a verb phrase (OTHER).
- Its syntactic head is *watched*.
- The dependency relationship to the head is subject (SUBJ).
- Its semantic role is ACTOR.
- There are many schemes of semantic types. For instance *girl* could be classified as HUMAN.

Note how phrasal annotations are handled. The first noun phrase is *the girl*. It is common to use an annotation scheme that tags individual words in a phrasal annation as BEGIN and CONTINUATION (or INTERMEDIATE), while labelling words outside such phrases as OTHER.

How do we encode the word-level factored representation? Recall that words are initially represented as 1-hot vectors. We can encode each factor in the factored representation as a 1-hot vector. The concatenation of these vectors is then used as input to the word embedding. Note that mathematically this means, that each factor of the representation is mapped to a embedding vector, and the final word embedding is the sum of the factor embeddings.

Since the input to the neural machine translation system is still a sequence of word embeddings, we do not have to change anything in the architecture of the neural machine translation model. We just provide richer input representations and hope that the model is able to learn how to take advantage of it.

Coming back to the debate about linguistics versus machine learning. All the linguistic annotation proposed here can arguable be learned automatically as part of the word embeddings (or contextualized word embeddings in the hidden encoder states). This may or may not be true. But it does provide additional knowledge that comes from the tools that produce the annotation and that is particularly relevant if there is not enough training data to automatically induce it. Also, why make the job harder for the machine learning algorithm than needed? In other words, why force the machine learning to discover features that can be readily provided? Ultimately, these questions will be resolved empirically by demonstrating what actually works in specific data conditions.

6.8.2 Linguistic annotation of the output

What we have done for input words could be done also for output words. Instead of discussing the fine points about what adjustments need to made (e.g., separate softmax for each output factor), let us take a look at another annotation scheme for the output that has been successfully applied to neural machine translation.

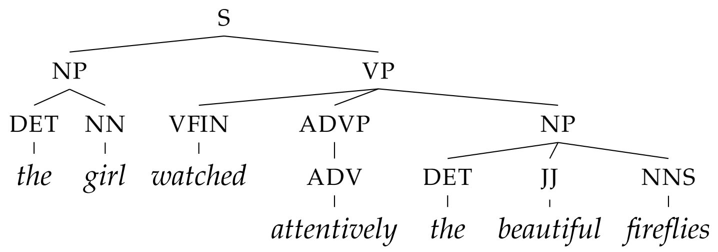
Sentence	*the girl watched attentively the beautiful fireflies*
Syntax tree	 S NP VP DET NN VFIN ADVP NP the girl watched ADV DET JJ NNS attentively the beautiful fireflies
Linearized	(S (NP (DET *the*) (NN *girl*)) (VP (VFIN *watched*) (ADVP (ADV *attentively*)) (NP (DET *the*) (JJ *beautiful*) (NNS *fireflies*))))

Figure 6.12: Linearization of phrase structure grammar tree into a sequence of words — e.g., *girl, watched* — and tags — e.g., (S, (NP,)

Most syntax-based statistical machine translation models have focused on adding syntax to the output side. Traditional n-gram language models are good at promoting fluency among neighboring words, they are not powerful enough to ensure overall grammaticality of each output sentence. By designing models that also produce and evaluate the syntactic parse structure for each output sentence, syntax-based models give the means to promote grammatically correct output.

The word-level annotation of phrase structure syntax suggested in Figure 6.11 is rather crude. The nature of language is recursive, and annotating nested phrases cannot be easily handled with a BEGIN/CONT/OTHER scheme. Instead, typically tree structures are used to represent syntax.

See Figure 6.12 for an example. It shows the phrase structure syntactic parse tree for our example sentence *The girl watched attentively the beautiful fireflies*. Generating a tree structures is generally a quite different process than generating a sequence. It is typically built recursively bottom-up with algorithms such as chart parsing.

However, we can **linearize** the parse tree into a sequence of words and structural tokens that indicate the beginning — e.g., "(NP" — and end — closing parenthesis ")" — of syntactic phrases. So, forcing syntactic parse tree annotations into our sequence-to-sequence neural machine translation model may be done by encoding the parse structure with additional output tokens. To be perfectly clear, the idea is to produce as the output of the neural translation system not just a sequence of words, but a sequence of a mix of output words and special tokens.

The hope is that forcing the neural machine translation model to produce syntactic structure (even in a linearized form) encourages it to produce syntactically well-formed output. There is some evidence to support this hope, despite the simplicity of the approach.

6.8.3 Linguistically structured models

The field of syntactic parsing has not been left untouched by the recent wave of neural networks. The previous section suggests that syntactic parsing may be done as simply as framing it as a sequence to sequence with additional output tokens.

However, the best-performing syntactic parsers use model structures that take the recursive nature of language to heart. They are either inspired by convolutional networks and build parse trees bottom-up, or are neural versions of left-to-right push-down automata that maintain a stack of opened phrases that any new word may extend or close, or be pushed down the stack to start a new phrase.

There is some early work on integrating syntactic parsing and machine translation into a unified framework but no consensus on best practices has emerged yet. At the time of writing, this is clearly still a challenge for future work.

Further Readings Wu et al. (2012) propose to use factored representations of words (using lemma, stem, and part of speech), with each factor encoded in a one-hot vector, in the input to a recurrent neural network language model. Sennrich and Haddow (2016) use such representations in the input and output of neural machine translation models, demonstrating better translation quality.

6.9 Multiple Language Pairs

There are more than two languages in the world. And we also have training data for many language pairs, sometimes it is highly overlapping (e.g., European Parliament proceedings in 24 languages), sometimes it is unique (e.g. Canadian Hansards in French and English). For some language pairs, a lot of training data is available (e.g., French–English). But for most language pairs, there is only very little, including commercially interesting language pairs such as Chinese–German or Japanese–Spanish.

There is a long history of moving beyond specific languages and encode meaning language-independent, sometimes called interlingua. In machine translation, the idea is to map the input language first into an interlingua, and then map the interlingua into the output language. In such a system, we have to build just one mapping step into and one step out of the interlingua for each language. Then we can translate between it and all the other languages for which we have done the same.

Researchers in deep learning often do not hesitate to claim that intermediate states in neural translation models encode semantics or meaning. So, can we train a neural machine translation system that accepts text in any language as input and translates it into any other language?

6.9.1 Multiple Input Languages

Let us say, we have two parallel corpora, one for German–English, and one for French–English. We can train a neural machine translation model on both corpora at the same time by simply concatenating them. The input vocabulary contains both German and French words. Any input sentence will be quickly recognized as being either German or French, due to the sentence context, disambiguating words such as *du* (*you* in German, *of* in French).

The combined model trained on both data sets has one advantage over two separate models. It is exposed to both English sides of the parallel corpora and hence can learn a better language model. There may be also be general benefits to having diversity in the data, leading to more robust models.

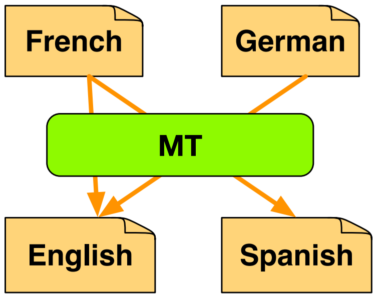


Figure 6.13: Multi-language machine translation system trained on one language pair at a time, rotating through many of them. After training on French–English, French–Spanish, and German–English, it is even able to translate from German to Spanish.

6.9.2 Multiple Output Languages

We can do the same trick for the output language, by concatenating, say, a French–English and a French–Spanish corpus. But given a French input sentence during inference, how would the system know which output language to generate? A crude but effective way to signal this to the model is by adding a tag like [SPANISH] as first token of the input sentence.

[ENGLISH] *N'y a-t-il pas ici deux poids, deux mesures?*
⇒ *Is this not a case of double standards?*

[SPANISH] *N'y a-t-il pas ici deux poids, deux mesures?*
⇒ *£No puede verse con toda claridad que estamos utilizando un doble rasero?*

If we train a system on the three corpora mentioned (German–English, French–English, and French–Spanish) we can also use it translate a sentence from German to Spanish — without having ever presented a sentence pair as training data to the system.

[SPANISH] *Messen wir hier nicht mit zweierlei Maß?*
⇒ *£No puede verse con toda claridad que estamos utilizando un doble rasero?*

For this to work, there has to be some representation of the meaning of the input sentence that is not tied to the input language and the output language. Surprisingly, experiments show that this actually does work, somewhat. To achieve good quality, however, some parallel data in the desired language pair is needed, but much less than for a standalone model (Johnson et al., 2016).

Figure 6.13 summarizes this idea. A single neural machine translation is trained on various parallel corpora in turn, resulting in a system that may translate between any seen input and output language. It is likely that increasingly deeper models (recall Section 6.4) may better serve as multi-language translators, since their deeper layers compute more abstract representations of language.

The idea of marking the output language with a token such as [SPANISH] has been explored more widely in the context of systems for a single language pair. Such tokens may represent the domain of the input sentence (Kobus et al., 2016), or the required level of politeness of the output sentence (Sennrich et al., 2016a).

6.9.3 Sharing Components

Instead of just throwing data at a generic neural machine translation model, we may want to more carefully consider which components may be shared among language-pair-specific models. The idea is to train one model per language pair, but some of the components are identical in these unique models.

- The encoder may be shared in models that have the same input language.
- The decoder may be shared in models that have the same output language.
- The attention mechanism may be shared in all models for all language pairs.

Sharing components means is that the same parameter values (weight matrices, etc.) are used in these separate models. Updates to them when training a model for one language pair then also changes them for in the model for the other language pairs. There is no need to mark the output language, since each model is trained for a specific language pair.

The idea of shared training of components can also be pushed further to exploit monolingual data. The encoder may be trained on monolingual input language data, but we will need to add a training objective (e.g., language model cross-entropy). Also, the decoder may be trained in isolation with monolingual language model data. However, since there are no context states available, these have to be blanked out, which may lead it to learn to ignore the input sentence and function only as a target side language model.

Further Readings Johnson et al. (2016) explore how well a single canonical neural translation model is able to learn from multiple to multiple languages, by simultaneously training on on parallel corpora for several language pairs. They show small benefits for several input languages with the same output languages, mixed results for translating into multiple output languages (indicated by an additional input language token). The most interesting result is the ability for such a model to translate in language directions for which no parallel corpus is provided, thus demonstrating that some interlingual meaning representation is learned, although less well than using traditional pivot methods.

Firat et al. (2016) support multi-language input and output by training language-specific encoders and decoders and a shared attention mechanism.