Encoder-Decoder Models for Neural Machine Translation

Erweiterungsmodul: Machine Translation
Sommersemester 2020

Jindřich Libovický
libovicky@cis.lmu.de

LMU München, The Center for Information and Language Processing

May 27, 2020
1 Model Concept

2 Language Models and Decoders

3 Conditioning the Language Model & Attention

4 Inference

5 Final Remarks
Model Concept

1. Model Concept
2. Language Models and Decoders
3. Conditioning the Language Model & Attention
4. Inference
5. Final Remarks
Conceptual Scheme of the Model

I am the walrus.

Encoder

↓

intermediate representation

Decoder

↓

Ich bin der Walros.

Neural model with a sequence of discrete symbols as an input that generates another sequence of discrete symbols as an output.

- pre-process source sentence (tokenize, split into smaller units)
- convert input into vocabulary indices
- run the encoder to get an intermediate representation (vector/matrix)
- run the decoder
- postprocess the output (detokenize)
From Statistical to Neural MT

Statistical
- Meaning of a sentence can be decomposed into phrases.
- Adequacy and fluency can be modeled separately.

Neural
- Sentences are sequence of words (or smaller discrete units).
- (And maybe something about words having independent meaning and compositionality, but this is difficult.)
⚠ Disclaimer ⚠

- Don’t trust much the intuitions presented there: NNs can learn things different than we assume.
- All concepts used are from one half technical concepts, from the other half lose metaphors.
Language Models and Decoders

1. Model Concept
2. Language Models and Decoders
3. Conditioning the Language Model & Attention
4. Inference
5. Final Remarks
What is a Language Model

LM = an estimator of a sentence probability given a language

- From now on: sentence = sequence of words \( w_1, \ldots, w_n \)
- Factorize the probability by word
  i.e., no grammar, no hierarchical structure

\[
\begin{align*}
\Pr(w_1, \ldots, w_n) &= \Pr(w_1) \cdot \Pr(w_2|w_1) \cdot \Pr(w_3|w_2, w_1) \cdot \cdots \\
&= \prod_{i=1}^{n} \Pr(w_i|w_{i-1}, \ldots, w_1)
\end{align*}
\]
What is it good for?

- Substitute for grammar: tells what is a good sentence in a language
- Used in ASR, and statistical MT to select more probable outputs
- Being able to predict next word = knowing the language
  - language modeling is training objective for word2vec
  - BERT is a masked language model

- **Neural decoder is a conditional language model.**
**n-gram vs. neural LMs**

### n-gram
- Limited history = Markov assumption
- Transparent: estimated from n-gram counts in a corpus

\[
P(w_i|w_{i-1}, w_{i-2}, \ldots, w_{i-n}) \approx \sum_{j=0}^{n} \lambda_j \frac{c(w_i|w_{i-1}, \ldots, w_{i-j})}{c(w_i|w_{i-1}, \ldots, w_{i-j+1})}
\]

### Neural
- Conditioned on RNN state which gather potentially unlimited history
- Trained by back-propagation to maximize probability of the training data
- Opaque, but works better (as usual with deep learning)
Sequence Labeling

- Assign a label to each word in a sentence.
- Tasks formulated as sequence labeling:
  - Part-of-Speech Tagging
  - Named Entity Recognition
  - Filling missing punctuation

MLP = Multilayer perceptron
\[ n \times \text{layer: } \sigma (Wx + b) \]

Softmax for \( K \) classes with logits
\[
z = (z_1, \ldots, z_K):
\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}
\]
Language Model as Sequence Labeling

- **Input symbol**: one-hot vectors
- **Embedding lookup**
- **RNN cell (more layers)**
- **Classifier**
- **Normalization**
- **Distribution for the next symbol**

\[
P(w_1 | <s>) \quad P(w_1 | \ldots) \quad P(w_2 | \ldots)
\]
Sampling from a Language Model

<\text{s}>

embed

RNN

MLP

softmax

Pr(w_1|<\text{s}>)
sample

embed

RNN

MLP

softmax

Pr(w_1|...)
sample

embed

RNN

MLP

softmax

Pr(w_2|...)
sample

embed

RNN

MLP

softmax

Pr(w_3|...)
sample

...

J. Libovický (CIS LMU)
last_w = "<s>"
state = initial_state
while last_w != "</s>":
    last_w_embedding = target_embeddings[0
state = rnn(state, last_w_embedding
logits = output_projection(state)
last_w = vocabulary[np.random.multimial(1, logits)]
yield last_w
Training

Training objective: negative-log likelihood:

\[
\text{NLL} = - \sum_{i}^{n} \log \Pr (w_i | w_{i-1}, \ldots, w_1)
\]

I.e., maximize probability of the correct word.

- Cross-entropy between the predicted distribution and one-hot “true” distribution
- Error from word is backpropagated into the rest of network unrolled in time
- Prone to exposure bias: during training only well-behaved sequences, it can break when we sample something weird at inference time
Generating from a Language Model

Neural Machine Translation is

- a new technology developed by a team at the University
- a technology that uses neural networks and machine learning to
- a powerful tool for understanding the spoken language.

(Example from GPT-2, a Transformer based English language model, screenshot from https://transformer.huggingface.co/doc/gpt2-large)

Where is the source language?
Model Concept

Language Models and Decoders

Conditioning the Language Model & Attention

Inference

Final Remarks
Conditional Language Model

Formally it is simple, condition distribution of

- target sequence $y = (y_1, \ldots, y_{T_y})$ on
- source sequence $x = (x_1, \ldots, x_{T_x})$

$$
Pr (y_1, \ldots, y_n | x) = \prod_{i}^{n} Pr (y_i | y_{i-1}, \ldots, y_1, x)
$$

We need an encoder to get a representation of $x$!

What about just continuing an RNN...
The interface between encoder and decoder is a single vector regardless the sentence length.

state = np.zeros(rnn_size)
for w in input_words:
    input_embedding = source_embeddings[w]
    state = enc_cell(encoder_state, input_embedding)

last_w = "<s>",
while last_w != "</s>":
    last_w_embedding = target_embeddings[last_w]
    state = dec_cell(state, last_w_embedding)
    logits = output_projection(state)
    last_w = vocabulary[np.argmax(logits)]
yield last_w
Motivation: It would be nice to can use variable length input representation

- RNN returns one state per word ...
- ...what if we were able to get only information from words we need to generate a word.

**Attention** = probabilistic retrieval of encoder states for estimating probability of a target words.

**Query** = hidden states of the decoder
**Values** = encoder hidden states
Sequence-to-Sequence Model With Attention

- Encoder = bidirectional RNN
- Decoder step starts as usual
- Decoder state $s_0$ used to compute distribution the over encoder states
- Weighted average of encoder states = context vector
- Decoder state & context concatenated

Attention Model in Equations (1)

**Inputs:**
- decoder state $s_i$
- encoder states $h_j = [\overrightarrow{h_j}; \overrightarrow{h_j}] \quad \forall i = 1 \ldots T_x$

**Attention energies:**

$$e_{ij} = v_a^\top \tanh (W_a s_{i-1} + U_a h_j + b_a)$$

**Attention distribution:**

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

**Context vector:**

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$
Output projection:

\[ t_i = \text{MLP} \left( s_{i-1} \oplus v_{y_{i-1}} \oplus c_i \right) \]

...attention is mixed with the hidden state

Output distribution:

\[ p \left( y_i = k | s_i, y_{i-1}, c_i \right) \propto \exp \left( W_o t_i + b_k \right)_k \]

- Different version of attentive decoders exist
- Alternative: keep the context vector as input for the next step
- Multilayer RNNs: attention between/after layers
state = np.zeros(emb_size)
fw_states = []
for w in input_words:
    input_embedding = source_embeddings[w]
    state, _ = fw_enc_cell(encoder_state, input_embedding)
    fw_states.append(state)

bw_states = []
state = np.zeros(emb_size)
for w in reversed(input_words):
    input_embedding = source_embeddings[w]
    state, _ = bw_enc_cell(encoder_state, input_embedding)
    bw_states.append(state)

enc_states = [np.concatenate(fw, bw) for fw, bw in zip(fw_states, reversed(bw_states))]

last_w = "<s>"
while last_w != "</s>":
    last_w_embeding = target_embeddings[last_w]
    state = dec_cell(state, last_w_embeding)
    alphas = attention(state, enc_states)
    context = sum(a * state for a, state in zip(alphas, enc_states))
    logits = output_projection(np.concatenate(state, context, last_w_embeding))
    last_w = np.argmax(logits)
    yield last_w
The agreement on the European Economic Area was signed in August 1992.

It should be noted that the marine environment is the least known of environments.

"This will change my future with my family," the man said.

Destruction of the equipment means that Syria can no longer produce new chemical weapons.

One of the motivations behind the proposed approach was the use of a fixed-length context vector in the basic encoder–decoder approach. We conjectured that this limitation may make the basic encoder–decoder approach to underperform with long sentences. In Fig. 2, we see that the performance of RNNencdec dramatically drops as the length of the sentences increases. On the other hand, both RNNsearch-30 and RNNsearch-50 are more robust to the length of the sentences. RNNsearch-50, especially, shows no performance deterioration even with sentences of length 50 or more. This superiority of the proposed model over the basic encoder–decoder is further confirmed by the fact that the RNNsearch-30 even outperforms RNNencdec-50 (see Table 1).

### Attention vs. Alignment

Differences between attention model and word alignment used for phrase table generation:

<table>
<thead>
<tr>
<th>attention (NMT)</th>
<th>alignment (SMT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>probabilistic</td>
<td>discrete</td>
</tr>
<tr>
<td>declarative</td>
<td>imperative</td>
</tr>
<tr>
<td>LM generates</td>
<td>LM discriminates</td>
</tr>
</tbody>
</table>
Image Captioning

Attention over CNN for image classification:

A woman is throwing a **frisbee** in a park.

A dog is standing on a hardwood floor.

A **stop** sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of **people** sitting on a boat in the water.

A giraffe standing in a forest with **trees** in the background.

Further Architectural Tricks

Figure 1: Model architecture of RNMT+. On the left side, the encoder network has 6 bidirectional LSTM layers. At the end of each bidirectional layer, the outputs of the forward layer and the backward layer are concatenated. On the right side, the decoder network has 8 unidirectional LSTM layers, with the first layer used for obtaining the attention context vector through multi-head additive attention. The attention context vector is then fed directly into the rest of the decoder layers as well as the softmax layer.

We use a shared vocabulary of 32K sub-word units for each source-target language pair. No further manual or rule-based post processing of the output was performed beyond combining the sub-word units to generate the targets. We report all our results on newstest 2014, which serves as the test set. A combination of newstest 2012 and newstest 2013 is used for validation.

To evaluate the models, we compute the BLEU metric on tokenized, true-case output.

For each training run, we evaluate the model every 30 minutes on the dev set. Once the model converges, we determine the best window based on the average dev-set BLEU score over 21 consecutive evaluations. We report the mean test score and standard deviation over the selected window. This allows us to compare model architectures based on their mean performance after convergence rather than individual checkpoint evaluations, as the latter can be quite noisy for some models.

To enable a fair comparison of architectures, we use the same pre-processing and evaluation methodology for all our experiments. We refrain from using checkpoint averaging (exponential moving averages of parameters) (Junczys-Dowmunt et al., 2016) or checkpoint ensembles (Jean et al., 2015; Chen et al., 2017) to focus on evaluating the performance of individual models.

This procedure is used in the literature to which we compare (Gehring et al., 2017; Wu et al., 2016).

4 RNMT+

4.1 Model Architecture of RNMT+
The newly proposed RNMT+ model architecture is shown in Figure 1. Here we highlight the key architectural choices that are different between the RNMT+ model and the GNMT model. There are 6 bidirectional LSTM layers in the encoder instead of 1 bidirectional LSTM layer followed by 7 unidirectional layers as in GNMT. For each bidirectional layer, the outputs of the forward layer and the backward layer are concatenated before being fed into the next layer. The decoder network consists of 8 unidirectional LSTM layers similar to the GNMT model. Residual connections are added to the third layer and above for both the encoder and decoder. Inspired by the Transformer model, per-patch layer normalization (Ba et al., 2016) is applied within each LSTM cell. Our empirical results show that layer normalization greatly stabilizes training. No non-linearity is applied to the LSTM output. A projection layer is added to the encoder final output.

Multi-head additive attention is used instead of the single-head attention in the GNMT model. Similar to GNMT, we use the additional projection aims to reduce the dimensionality of the encoder output representations to match the decoder stack dimension.

Optimize negative log-likelihood of parallel data, backpropagation does the rest.

If you choose a right optimizer, learning rate, model hyper-parameters, prepare data, do back-translation, monolingual pre-training ...

During training confusion: decoder inputs vs. output

inputs $y[: -1]$  
\[
< s > \quad y_1 \quad y_2 \quad y_3 \quad y_4 \\
\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
\text{Decoder} \\
\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
\text{outputs } y[1:]  
\quad y_1 \quad y_2 \quad y_3 \quad y_4 \quad </ s >
Inference

1. Model Concept
2. Language Models and Decoders
3. Conditioning the Language Model & Attention
4. Inference
5. Final Remarks
Getting output

- Encoder-decoder is a conditional language model
- For a pair $x$ and $y$, we can compute:

$$\Pr(y|x) = \prod_{i=1}^{T_y} \Pr(y_i|y_{:i}, x)$$

- When decoding we want to get

$$y^* = \arg\max_{y'} \Pr(y'|x)$$

☠ Enumerating all $y$’s is computationally intractable ☠
Greedy Decoding

In each step, take the maximum probable word.

\[ y_i^* = \underset{y_i}{\operatorname{argmax}} \Pr(y_i|y_{i-1}^*, \ldots, <s>) \]

last_w = "<s>"
state = initial_state
while last_w != "</s>":
    last_w_embedding = target_embeddings[last_w]
    state = dec_cell(state, last_w_embedding)
    logits = output_projection(state)
    last_w = vocabulary[np.argmax(logits)]
yield last_w
⚠ Greedy decoding can easily miss the best option. △
Beam Search

Keep a small $k$ of hypothesis (typically 10).

1. Begin with a single empty hypothesis in the beam.
2. In each time step:
   1. Extend all hypotheses in the beam by all (or the most probable) from the output distribution (we call these candidate hypotheses)
   2. Score the candidate hypotheses and add them to the beam
3. Finish if all $k$-best hypotheses end with $</s>$
4. Sort the hypotheses by their score and output the best one.
Beam Search: Pseudocode

beam = [(["<s>"], initial_state, 1.0)]
while any(hyp[-1] != "</s>" for hyp, _, _ in beam):
    candidates = []
    for hyp, state, score in beam:
        distribution, new_state = decoder_step(hyp[-1], state, encoder_states)
        for i, prob in enumerate(distribution):
            candidates.append(hyp + [vocabulary[i]], new_state, score * prob)
    beam = take_best(k, candidates)
Implementation issues

- Multiplying of too many small numbers → float underflow
  need to compute in log domain and add logarithms

- Sentences can have different lengths
  
  This is a good long sentence.
  
  \[
  0.7 \times 0.6 \times 0.9 \times 0.1 \times 0.4 \times 0.4 \times 0.8 \times 0.9 = 0.004
  \]

  This
  
  \[
  0.7 \times 0.01 = 0.007
  \]

- Sorting candidates is expensive: \(k\)-best can be found in linear time
Final Remarks

1 Model Concept

2 Language Models and Decoders

3 Conditioning the Language Model & Attention

4 Inference

5 Final Remarks
A bit of history (1)

- **2013** First encoder-decoder model
  

- **2014** First really usable encoder-decoder model


- **2014/2015** Added attention (crucial innovation in NLP)


- **2016/2017** WMT winners used RNN-based neural systems

A bit of history (2)

- **2017** Transformers invented (outperformed RNN)
  

- **2018** RNMT+ architecture shows that RNNs can be on par with Transformers
  

The development still goes on…

Unsupervised models, document context, non-autoregressive models, multilingual models, …
Summary

- Encoder-decoder architecture = major paradigm in MT
- Encoder-decoder architecture = conditional language model
- Attention = way of conditioning the decoder on the encoder
- Attention = probabilistic vector retrieval
- We model probability, but need heuristics to get a good sentence from the model