Overview

last lecture
- how do we represent language in neural networks?
- how do we treat language probabilistically (with neural networks)?

today’s lecture
- how do we model translation with a neural network?
- how do we generate text from a probabilistic translation model?

Modelling Translation

Suppose that we have:
- a source sentence \( S \) of length \( m \) \((x_1, \ldots, x_m)\)
- a target sentence \( T \) of length \( n \) \((y_1, \ldots, y_n)\)

We can express translation as a probabilistic model

\[
T^* = \arg \max_T p(T|S)
\]

Expanding using the chain rule gives

\[
p(T|S) = p(y_1, \ldots, y_n|x_1, \ldots, x_m)
\]

\[=
\prod_{i=1}^{n} p(y_i|y_1, \ldots, y_{i-1}, x_1, \ldots, x_m)
\]

Differences Between Translation and Language Model

- Target-side language model:

\[
p(T) = \prod_{i=1}^{n} p(y_i|y_1, \ldots, y_{i-1})
\]

- Translation model:

\[
p(T|S) = \prod_{i=1}^{n} p(y_i|y_1, \ldots, y_{i-1}, x_1, \ldots, x_m)
\]

We could just treat sentence pair as one long sequence, but:
- We do not care about \( p(S) \)
- We may want different vocabulary, network architecture for source text
**Differences Between Translation and Language Model**

- Target-side language model:
  \[ p(T) = \prod_{i=1}^{n} p(y_i|y_1, \ldots, y_{i-1}) \]

- Translation model:
  \[ p(T|S) = \prod_{i=1}^{n} p(y_i|y_1, \ldots, y_{i-1}, x_1, \ldots, x_m) \]

- We could just treat sentence pair as one long sequence, but:
  - We do not care about \( p(S) \)
  - We may want different vocabulary, network architecture for source text
  → Use separate RNNs for source and target.

**Summary vector**

- Last encoder hidden-state “summarises” source sentence
- With multilingual training, we can potentially learn language-independent meaning representation

**Encoder-Decoder for Translation**

- Encoder: natürich, hat, john, spaß
- Decoder: of course, john has fun

![Diagram of Encoder-Decoder for Translation](image-url)
Summary vector as information bottleneck

Problem: Sentence Length
- Fixed sized representation degrades as sentence length increases
- Reversing source brings some improvement [Sutskever et al., 2014]

Solution: Attention
- Compute context vector as weighted average of source hidden states
- Weights computed by feed-forward network with softmax activation

Encoder-Decoder with Attention

of course john has fun

Decoder

 naturally hat john spaß

Encoder

h1 h2 h3 h4

x1 x2 x3 x4

0.7 0.1 0.1 0.1

0.6 0.2 0.1 0.1

of course john has fun

Decoder

 naturally hat john spaß

Encoder
Attentional encoder-decoder: Maths

- Simplifications of model by [Bahdanau et al., 2015] (for illustration)
- Plain RNN instead of GRU
- Simpler output layer
- We do not show bias terms
- Decoder follows **Look, Update, Generate** strategy [Sennrich et al., 2017]
- Details in [https://github.com/amunmt/amunmt/blob/master/contrib/notebooks/dl4mt.ipynb](https://github.com/amunmt/amunmt/blob/master/contrib/notebooks/dl4mt.ipynb)

**Notation**
- $W, U, E, C, V$ are weight matrices (of different dimensionality)
  - $E$ one-hot to embedding (e.g. $50000 \cdot 512$)
  - $W$ embedding to hidden (e.g. $512 \cdot 1024$)
  - $U$ hidden to hidden (e.g. $1024 \cdot 1024$)
  - $C$ context (2x hidden) to hidden (e.g. $2048 \cdot 1024$)
  - $V_o$ hidden to one-hot (e.g. $1024 \cdot 50000$)
- Separate weight matrices for encoder and decoder (e.g. $E_x$ and $E_y$)
- Input $X$ of length $T_x$; output $Y$ of length $T_y$
Attentional encoder-decoder: Maths

**encoder**

\[
\overrightarrow{h}_j = \begin{cases} 
0, & \text{if } j = 0 \\
\tanh(\overrightarrow{W}x_{xj} + \overrightarrow{U}xh_{j-1}), & \text{if } j > 0 
\end{cases}
\]

\[
\overleftarrow{h}_j = \begin{cases} 
0, & \text{if } j = T_x + 1 \\
\tanh(\overleftarrow{W}x_{xj} + \overleftarrow{U}xh_{j+1}), & \text{if } j \leq T_x 
\end{cases}
\]

\[
h_j = (\overrightarrow{h}_j, \overleftarrow{h}_j)
\]

**decoder**

\[
s_i = \begin{cases} 
\tanh(W_s\overrightarrow{h}_i), & \text{if } i = 0 \\
\tanh(W_yE_yy_{i-1} + U_yy_{i-1} + C_c), & \text{if } i > 0 
\end{cases}
\]

\[
t_i = \tanh(U_os_i + W_yE_yy_{i-1} + C_c)
\]

\[
y_i = \text{softmax}(V_o t_i)
\]

**attention model**

\[
e_{ij} = v_i^\top \tanh(W_ah_{j-1} + U_ah_j)
\]

\[
\alpha_{ij} = \text{softmax}(e_{ij})
\]

\[
c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j
\]

**Attention model**

- side effect: we obtain alignment between source and target sentence
- information can also flow along recurrent connections, so there is no guarantee that attention corresponds to alignment
- applications:
  - visualisation
  - replace unknown words with back-off dictionary [Jean et al., 2015]
  - ...

**Attention model also works with images:**
Decoding

**exact search**
- generate every possible sentence $T$ in target language
- compute score $p(T|S)$ for each
- pick best one

- intractable: $|\text{vocab}|^N$ translations for output length $N$  
  $\rightarrow$ we need approximative search strategy

**approximative search/1: greedy search**
- at each time step, compute probability distribution $P(y_i|S, y_{<i})$
- select $y_i$ according to some heuristic:
  - sampling: sample from $P(y_i|S, y_{<i})$
  - greedy search: pick $\arg\max_y p(y_i|S, y_{<i})$
- continue until we generate $<$eos$>$

- efficient, but suboptimal
Decoding

approximative search/2: beam search
- maintain list of $K$ hypotheses (beam)
- at each time step, expand each hypothesis $k$: $p(y_i^k|S, y_{<i}^k)$
- select $K$ hypotheses with highest total probability:
  \[
  \prod_i p(y_i^k|S, y_{<i}^k)
  \]

- relatively efficient . . . beam expansion parallelisable
- currently default search strategy in neural machine translation
- small beam ($K \approx 10$) offers good speed-quality trade-off

Ensembles

- combine decision of multiple classifiers by voting
- ensemble will reduce error if these conditions are met:
  - base classifiers are accurate
  - base classifiers are diverse (make different errors)

Ensembles in NMT

- vote at each time step to explore same search space (better than decoding with one, reranking n-best list with others)
- voting mechanism: typically average (log-)probability
  \[
  \log P(y_i|S, y_{<i}) = \frac{\sum_{m=1}^M \log P_m(y_i|S, y_{<i})}{M}
  \]
- requirements for voting at each time step:
  - same output vocabulary
  - same factorization of $Y$
  - but: internal network architecture may be different
- we still use reranking in some situations
  example: combine left-to-right decoding and right-to-left decoding

Further Reading

Required Reading
- Koehn, 13.5

Optional Reading
- Sequence to Sequence Learning with Neural Networks. (Sutskever, Vinyals, Le):
- Neural Machine Translation by Jointly Learning to Align and Translate. (Bahdanau, Cho, Bengio):


