# Transliteration with Neural Sequence to Sequence Networks

Karl-Heinz Krachenfels
CIS, LMU-Munich, Germany

July 26, 20217

## Abstract

In this work we investigate the performance of different sequence to sequence architectures (seq2seq) for transliteration. We start our investigation with simple transliteration tasks based on artificially generated transliterations which easily demonstrate some of the key characteristics of the different seq2seq models. Specifically we investigate the seq2seq models based on LSTMs and Bi-LSTMs and show how they compare to encoder-decoder architectures as described in Cho et al. (2014). We then show the effect of adding an attention mechanism as described in Bahdanau et al. (2014). In the last part of the paper we visualize the attention matrix for some examples that illustrate different aspects of the attention mechanism.

## 1 Introduction

### 1.1 Motivation

Recurrent neural networks (RNNs) are well suited for NLP problems where we need to map a source sequence of characters, words or phonemes to a corresponding target sequence. Examples of such problems are machine translation, speech recognition, speech synthesis, part of speech tagging and named entity recognition to name a few. One of the main reasons for the success of RNNs is their ability in keeping context over long distances. RNNs can thus be seen as neural networks with a sort of memory. RNN based neural networks are therefore of great interest for NLP related tasks because many traditional models don't look at word order at all as e.g bag of words models or only look at local contexts as e.g. n-gram models. RNNs are also a promising approach for language models where structure is relevant. Examples are Dyer et al. (2015) who develop a new control structure for a stack LSTM to implement a dependency parser and Bowman et al. (2016) who develop a tree-structured neural network that is also based on a combination of a LSTM with a stack.

One last point to mention, is that in contrast to other neural models like convolutional neural networks (CNNs), RNNs are ideally suited for variable length input due to their recurrent nature.

### 1.2 Recent History of RNNs

In the ninties there was a raise in the interest of recursive neural networks (RNNs). One of the neural architectures developed at this time was the Elman network with a sort of feedback

loop for each hidden neuron, see Elman (1990). Unfortunately these networks were not able to model long distance dependencies. The problem was formally studied by a student of Jürgen Schmidhuber who came up with the theory for the so called vanishing/expolding gradient problem. As a neural architecture to overcome this problem the LSTM was invented, see Hochreiter and Schmidhuber (1995), Hochreiter and Schmidhuber (1997).

# 2 Models

## 2.1 seq2seq with LSTM

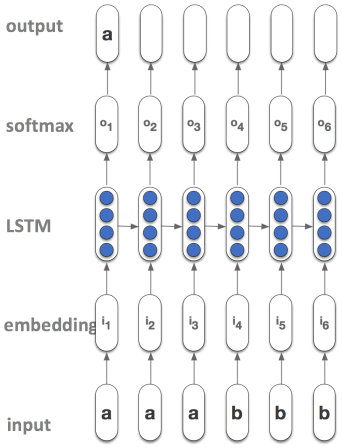


Figure 1: Illustration of a sequence to sequence architecture based on a LSTM

seq2seq architectures read a input sequence of varying length and map it to an output sequence. If we apply such models to language tasks then a simple network architecture consists of 3 layers:

- An embedding layer that maps a one hot encoding of an input symbol, which in our experiments will be a mapping of a character into a fixed size dense vector.
- A recurrent layer which takes as input the dense vector from the embedding layer, it computes an output and feeds back its hidden state to the recurrent layer for the next timestep. For our experiments we use a LSTM layer, but a GRU layer, see Chung et al. (2014),would be possible as well.
- A softmax layer takes the output and maps it to a target symbol, which is a character in our experiments.

## 2.2 Bidirectional LSTM

Assume a transliteration model where the sequence *aaa* is transliterated to *a* and *bbb* to *b*. In figure 1 you can see that after mapping the first *a* of the input sequence to a *a* of the output sequence the LSTM has no way to determine the next output symbol because it has no way to look into the future. The model would have to take into consideration the inputs at positions 2,3,4 to determine the right output. This problem can be solved by a bidirectional LSTM (Bi-LSTM) as shown in figure 2 where a LSTM that works in reverse order determines the right context for the given position. The idea of combining two recurrent networks to a bidirectional recurrent network (BRNN) goes back to Schuster and Paliwal (1997). BRNNs unfold their full potential when combined with the LSTM concept, see Graves and Schmidhuber (2005).

## 2.3 Encoder-Decoder Architectures

Another architectural variant for seq2seq mapping is the encoder-decoder architecture. In the simplest variant of this architecture the input is

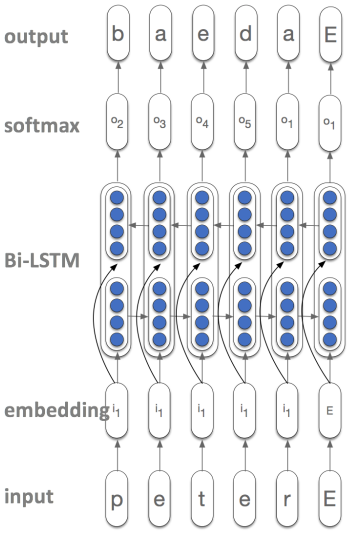


Figure 2: Illustration of a sequence to sequence with Bi-LSTM

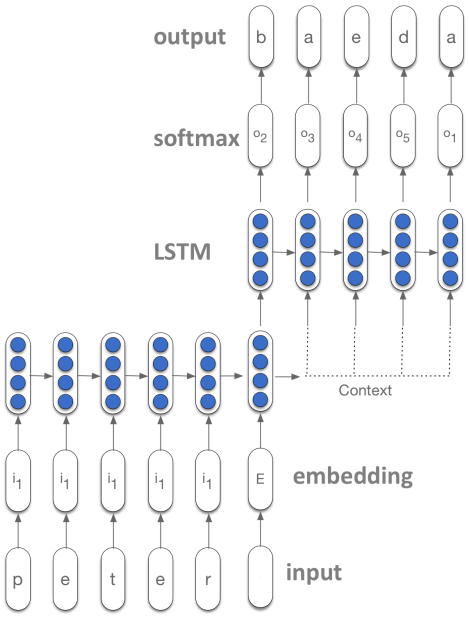


Figure 3: Illustration of an encoder-decoder architecture conditioned on the last encoder state

padded with an $E$ symbol and fed into a encoder RNN. The end state of the Encoder state is then fed into a decoder RNN that generates the target sequence. Sutskever et al. (2014) successfully applied this architecture for word based machine translation and achieved state of the art results. A natural intuition is that the encoder models the source language while the decoder models the target language.

Cho et al. (2014) extend the seq2seq architecture by conditioning each step of the decoder on a context. As context they take the last hidden state of the encoder. For our experiments we use a variant of this model where the decoder output state at time $t-1$ is not fed back into the decoder at time step $t$. Our model is shown in figure 3. With this simplification we get a end to end differentiable model which is comparable to models with greedy search as opposed to those decoders that use beam search.

### 2.4 Encoder-Decoder Architecture with Attention

Instead of conditioning on one context vector from the encoder Bahdanau et al. (2014) introduced the encoder-decoder architecture with attention. The idea of attention is to condition each time step of the decoder on the best matching time step of the encoder. This is done by learning an alignment function a

$$e_{ij} = a(s_i, h_j)$$

that calculates the energy between an encoder time step $j$ and the decoder step $i$. For the decoder state at time step $i$ we now can compute the activation score $\alpha_{ij}$ by applying a softmax function

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

Based on the activation scores $\alpha_{ij}$ and the hidden states of the decoder we now can compute a context vector that is an additional input signal to the decoder at the next timestep $s_t$, see figure 4.

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

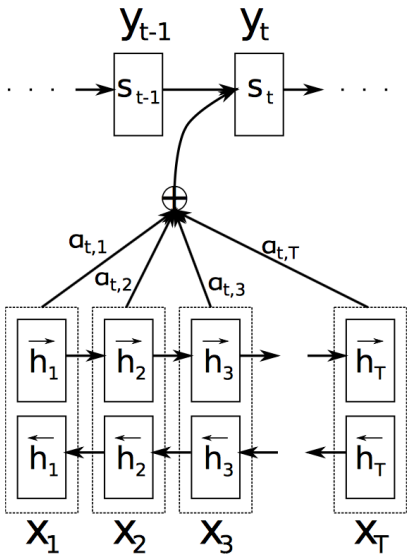


Figure 4: Illustration of an encoder-decoder architecture with attention, from Bahdanau et al. (2014)

### 2.5 Calculation of the Attention Score

The attention can be an arbitrary function that takes an encoder hidden vector $h_j$ at time step $j$ and a decoder hidden vector $s_i$ at time step $i$ and scores the correlation of the two. We can calculate the attention energy $e_{ij}$ then by either (see Neubig (2017))

- a dot product, $e_{i,j} = h_j^T s_i$,
- a bilinear function, $e_{i,j} = h_j^T W_\alpha s_i$,
- a neural network with a hidden layer (MLP).

For our experiments we use an attention function consisting of a MLP with the same hidden layer size as the size of the encoder layer and the decoder layers.

## 3 The Toy Problem

### 3.1 Description

The artificial problem that we use to illustrate some of the core characteristics of seq2seq neural networks uses transliterations generated with the following list of transliteration pairs:

```
[('k','g'),('t','d'),('p','b'),
('erl','la'),('e','ae'),
('er','a'),('lade','laad')
('a','a'),('b','b'),('c','c'),
...,('z','z')]
```

In order to generate one word pair we sample n random elements from the transliteration table, where n is a random integer number drawn from a uniform distribution between *minlength* and *maxlength*. A random sample for $n = 3$ could e.g. look like:

```
('u','u')
('k','g')
('erl','la')
```

and the corresponding transliteration pair would be

```
('ukerl', 'ugla')
```

# 4 Experiments

## 4.1 Generating Data for the Experiments

For our experiments we generate 300000 word pairs with a random length between 1 and 15. We train the network with learning rate $\eta = 0.1$. Furthermore we generate a test set of size 100 word pairs and measure the loss against the validation set 100 times during the training.

## 4.2 Model description and parameters

We used the following parameters for our model:

```
embedding_size=10
encoder_size=128
decoder_size=128
attention_hidden_layer_size=128
```

For the LSTM variants we use 128 as size for the LSTM layer(s). We generated the output sequence up to the point when the end symbol was generated which was also part of the target sequence during the training or up to a maximum length of two times the input length. The loss function that we used for training was the negative log loss for the prediction of each character in the softmax layer.

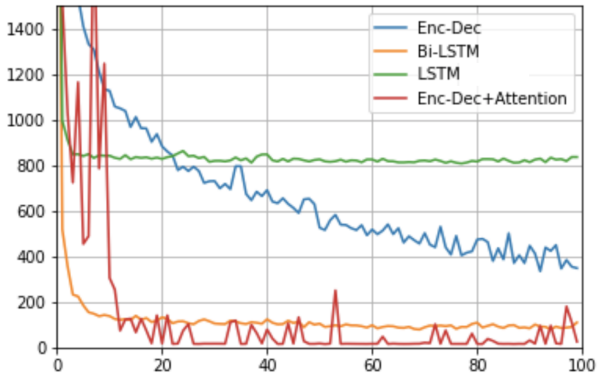


Figure 5: Loss on toy problem task

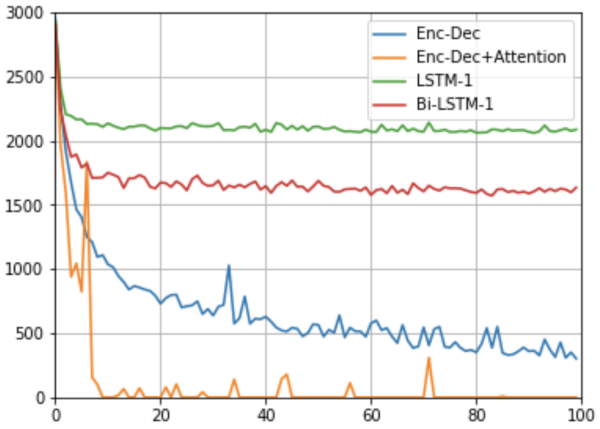


Figure 6: Loss on task to revert the character order

# 5 Results

## 5.1 Learning Curves

We conducted the experiments on the toy problem and on a problem to just revert strings. The results are shown in figure 5 and figure 6. We can see that the toy problem where the challenge is to do alignment over small distances is a less hard problem. We believe that the missing ability to handle right context is the reason

why the results for the LSTM are quite bad while the Bi-LSTM delivers good results. Nevertheless the Bi-LSTM cannot handle some transliterations probably when the distance between corresponding alignment positions becomes too large. For the string reversion problem both LSTM and Bi-LSTM network fail. In this case it seems that the encoder-decoder might come up with a good solution after very long training, while the encoder-decoder with attention converges quick to a solution with zero loss. It seems that we should adapt some of the hyper parameters concerning learning rate, optimizer, etc. to get rid of the peaks in the learning curve of the encoder-decoder with attention.

### 5.2 Qualitative Results

The table shows the transliteration for a longer problem. It seems that the Bi-LSTM performs on a high level. The Bi-LSTM fails in the revert string problem, see figure 6

| | petersfränkischemarmelade |
|---|---|
| LSTM | baedlsfreamarmaelaad |
| Bi-LSTM | baedasfrengischaemarmaelaad |
| Enc-Dec | baedasfreeeiiclaaaaaa |
| Enc-Dec+Att | baedasfrengischaemarmlaad |

## 6 Analysis of attention weights

### 6.1 Practical Alignment Problem

The matrix in figure 7 for the transliteration for *peter* $\rightarrow$ *baeda* shows that the attention mediates the alignment between input sequence and output sequence. You can nicely see how the attention matrix shows the position where a transliteration replaces one character on the source with two on the target side ($e \rightarrow ae$) and also where two characters on the source side are replaced with one character on the target side ($er \rightarrow a$).

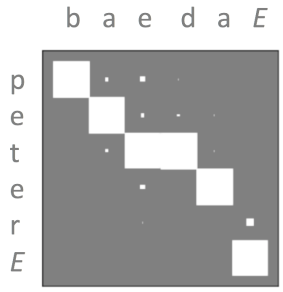


Figure 7: *peter* $\rightarrow$ *baeda*

### 6.2 Synthetic Alignments

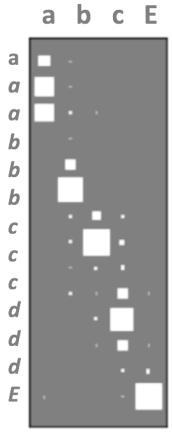


Figure 8: Transliteration of $aaa \rightarrow a, bbb \rightarrow b, ...$

In figure 8 and figure 9 you can see how a encoder-decoder architecture with attention handles the cases where the transliterations replace multiple characters on the left side with one character on the right side and vice versa.

We trained a network with an additional transliteration pair *abcd* $\rightarrow$ *dcba*. We tested the

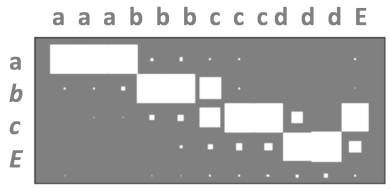


Figure 9: Transliteration of $a \rightarrow aaa, b \rightarrow bbb, ...$

encoder-decoder network with attention with the input sequence *abcdabcdabcdabcd*. You can see that the attention matrix focuses on the end of each segment.

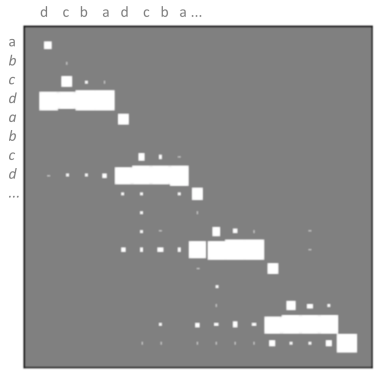


Figure 10: We added the rule $abcd- > dcba$ and tested on *abcdabcdabcdabcd*

### 6.3 Long Distance Example

We extended our transliteration table and added 30 additional copies of the pair $en \rightarrow a$ and additionally added a rule that was applied at the end of the generator algorithm that added a $g$ at the beginning of each target word when the source word ended with $en$. An example word pair could be $laufen \rightarrow glaufa$.

If we look into the weight matrix (figure 11) we can see that the attention weights put some

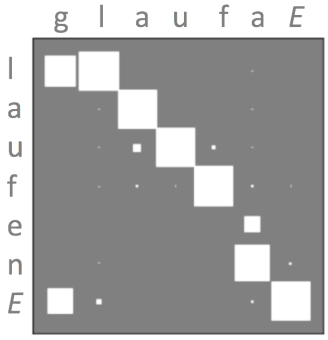


Figure 11: Long distance transliteration by adding a $g$ at the beginning depending on the end of the input sequence. Here the word pair is $laufen \rightarrow glaufa$

focus on the end state of the encoder, which can be interpreted as taking into account the end of the source sequence. In the case that there is a $en$ the decoder decides to put a $g$ at the beginning of the output sequence. We tested this behavior also with longer sequences and saw that it worked quite well while the Bi-LSTM failed on the longer sequences in this case.

### 6.4 Premature Network Example

In this experiment we wanted to find out the effect of attention in a not yet perfectly trained network. To do this trained our original model on a smaller set of only 100000 word pairs. We then tested with the word $petersfränkischemarmelade$. We found that the pattern of the attention weights clearly shows the place where the output gets fuzzy and incorrect.

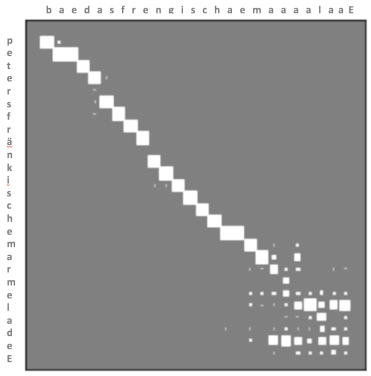


Figure 12: Premature transliteration of $petersfränkischemarmelade \rightarrow baedasfrengischaemaaaalaa$

### 6.5 Revert String Examples

Figure 13 shows a simple example matrix for a string reversion. Figure 14 shows an example of a longer sequence where the neural network confuses the start of the result sequence and where the result sequence is shorter. Figure 15 shows a sequence where a middle part is omitted.

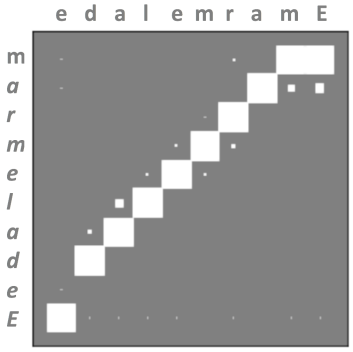


Figure 13: Reversion of $marmelade \rightarrow edalemram$

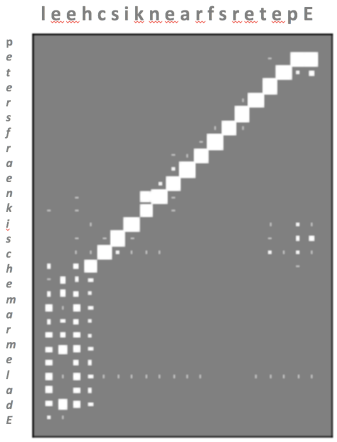


Figure 14: Reversion of long string where the beginning of the result string is confused $petersfraenkischemarmelade \rightarrow leehcsiknearfsretep$

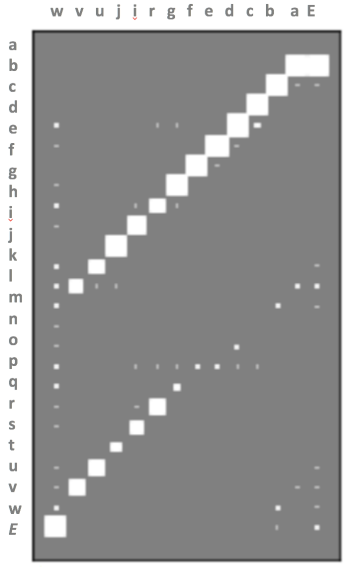


Figure 15: Reversion of sequence where the decoder forgets a part in the middle

# 7 Conclusion and Future Work

## 7.1 Conclusion

We believe that the invention of attention is one of the big innovations in NMT and also other deep learning fields during the last years. We found it interesting to inspect the attention weights and we believe that the weight matrix makes the mechanisms in the Neural Net more interpretable. We can directly see some internal mechanisms like long distance dependencies in the matrix which work well with attention and are not solved by any of the other architectures over long distances. We can also see that as soon as the weights loose focus the quality of the output sequence decreases and the network starts to stumble. The framework that we used i.e. dynet, was easy to use and removed much of the burden you normally cope with in typical seq2seq implementations like varying length of the sequences, extra dimension to support mini-batches and easy to use APIs. The framework also offers some tweaks under the hood, e.g. a modification to the LSTM architecture that results in some performance gains, see Neubig (2017).

## 7.2 Future Work

While our experiments mostly focus on simple transliteration tasks with simple alignments problems we believe that encoder-decoder networks are suited for harder tasks like generating morphological forms. An architecture for generating morphological forms could be achieved by conditioning the decoder on a signal that determines the desired form or by concatenating a potentially embedded form signal to the input vectors.

Other directions of research could go into feeding in context from different modalities, example tasks are multi-modal translation and image captioning based on attention on the features of an image, see Xu et al. (2015).

As we have seen in the examples, one of the main deficiencies of the encoder-decoder models is coverage, i.e. the problem that the decoder neither should omit parts nor add additional parts or repeat itself. One research direction to address this are Pointer Networks, see Vinyals et al. (2015). Tu et al. (2016) introduce a coverage vector to keep track of the parts that are already covered and See et al. (2017) use a combination of pointer network and coverage vector to generate summarizations.

Another interesting research direction is to use the attention mechanism not only to align but also to model structural properties. An example is shown in Allamanis et al. (2016), who use a convolutional attention mechanism to detect short and long range attentual features on a task to summarize source code.

Last but not least we find it interesting to research how the different character based approaches for tranliteration, morphology, translation can be integrated into one system that jointly solves the translation task.

# References

M. Allamanis, H. Peng, and C. Sutton. A convolutional attention network for extreme summarization of source code. In *International Conference on Machine Learning*, pages 2091–2100, 2016.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learn-

ing to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

S. R. Bowman, J. Gauthier, A. Rastogi, R. Gupta, C. D. Manning, and C. Potts. A fast unified model for parsing and sentence understanding. *arXiv preprint arXiv:1603.06021*, 2016.

K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

C. Dyer, M. Ballesteros, W. Ling, A. Matthews, and N. A. Smith. Transition-based dependency parsing with stack long short-term memory. *arXiv preprint arXiv:1505.08075*, 2015.

J. L. Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610, 2005.

S. Hochreiter and J. Schmidhuber. Long short-term memory, 1995.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

G. Neubig. Neural machine translation and sequence-to-sequence models: A tutorial. *arXiv preprint arXiv:1703.01619*, 2017.

M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

A. See, P. J. Liu, and C. D. Manning. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*, 2017.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

Z. Tu, Z. Lu, Y. Liu, X. Liu, and H. Li. Modeling coverage for neural machine translation. *arXiv preprint arXiv:1601.04811*, 2016.

O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2(3):5, 2015.