Building an end-to-end speech recognizer

Moritz Wolter

01.02.2017

Outline

1. Introduction
   - Listen Attend and Spell
2. Methodology
3. Implementation
4. Results
   - Custom attention
5. Questions

The problem

- What is being said?
- What is interesting?
- What model architecture?
- Which hyper-parameters?
- How to ensure generalization?

Long Short Term Memory

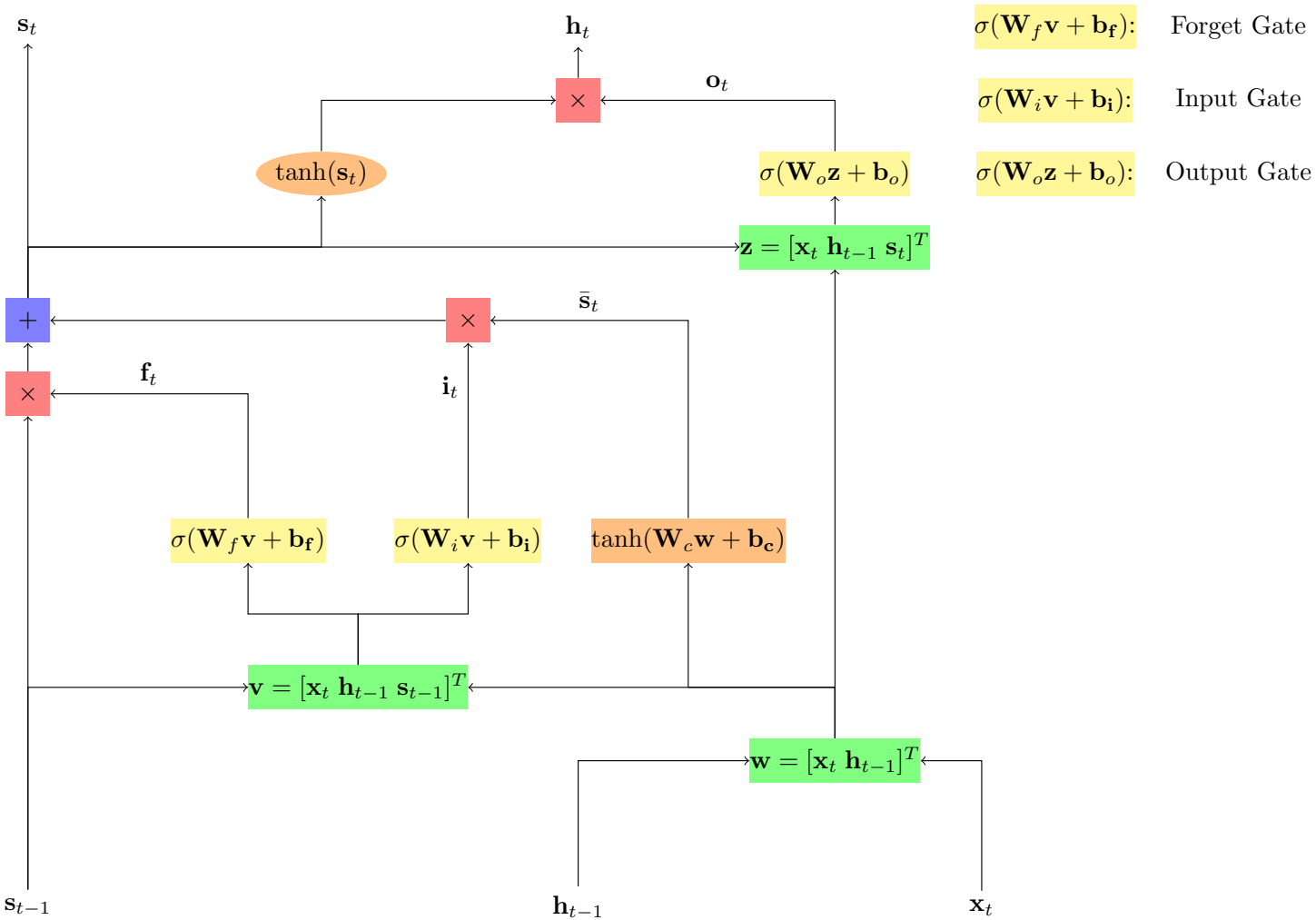

Figure 1 : LSTM cell architecture visualization.

The LAS-Architecture

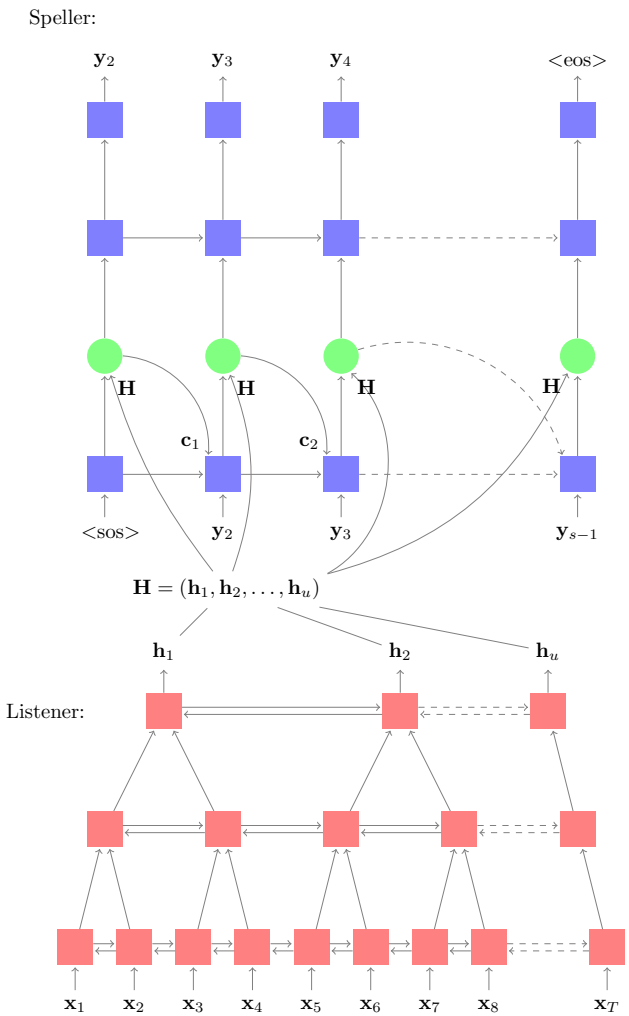

Figure 2 : Listener and speller.

The LAS-Equations

Attend and spell cell computations:

$$\mathbf{s}_i = \text{RNN}(\mathbf{s}_{i-1}, \mathbf{y}_{i-1}, \mathbf{c}_{i-1}), \tag{1}$$

$$\mathbf{c}_i = \text{AttentionContext}(\mathbf{s}_i, \mathbf{H}), \tag{2}$$

$$P(\mathbf{y}_i|\mathbf{x}, \mathbf{y}_{<i}) = \text{CharacterDistribution}(\mathbf{s}_i, \mathbf{c}_i). \tag{3}$$

AttentionContext computations:

$$e_{i,u} = \phi(\mathbf{s}_i)^T \psi(\mathbf{h_u}), \tag{4}$$

$$\alpha_{i,u} = \frac{\exp(e_{i,u})}{\sum_u \exp(e_{i,u})}, \tag{5}$$

$$\mathbf{c}_i = \sum_u \alpha_{i,u} \mathbf{h}_u. \tag{6}$$

Used methods

- Object oriented programming, python.
  - data encapsulation, partly replaced by `tf.variable_scope("...")`.
  - code structure and re-usability.
- Shape invariants, tensorflow.
  - Programmer must explicitly state changing dimensions in loop.
  - Makes code easier to read, and prevents bugs.
- Code quality, pylint.
  - if the python foundation' style guide is being followed.
  - looks for errors in the code.
- Version control, git.

Attend and spell cell layout

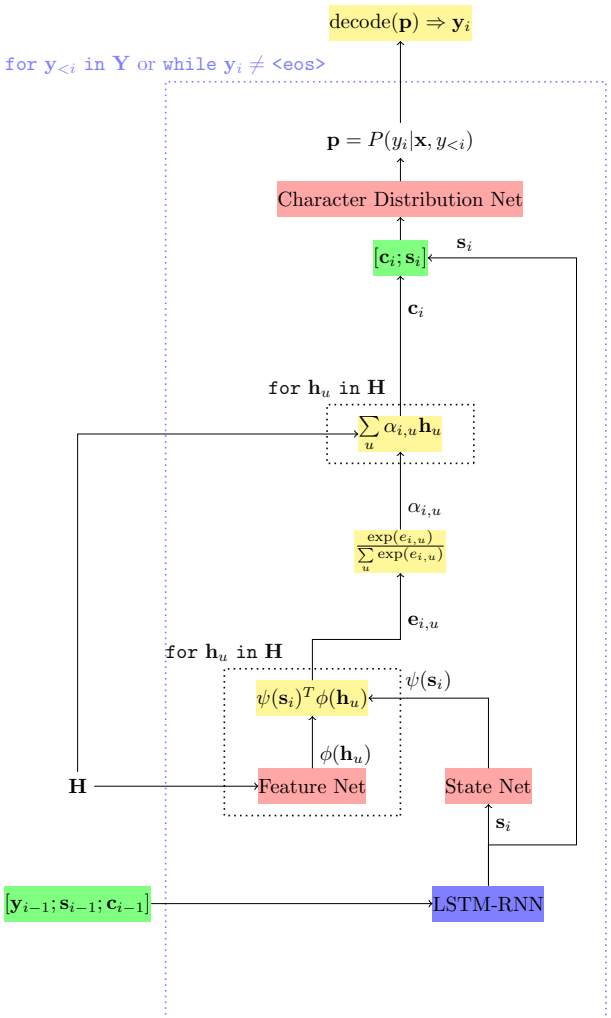

Figure 3 : Attend and spell cell flow chart.

Decoding loop logic

body loop logic

```
while keep_working:
  not_done_count = reduce_sum( logical_not( d ))
  done = equal(not_done_count, 0)
  stop = logical_or(done, greater(time, max_steps))
  keep_working = logical_not(stop)
```

setting the sequence length

```
  decoded = decode(logits)
  mask = tf.equal(decoded, <eos>)
  time_vec = ones(self.batch_size)*(time+1)
  sequence_lengths = select(d,
                            logits_sequence_length,
                            time_vec)
  d = logical_or(mask, d)
```

Dropout

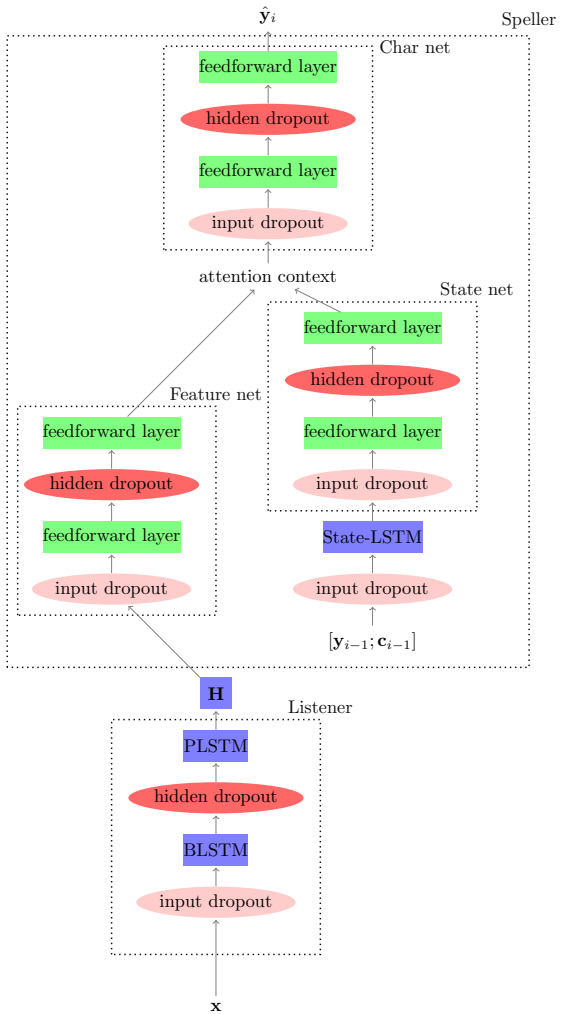

Figure 4 : Input dropout - light red. Hidden dropout - dark red.

The Listener with CTC

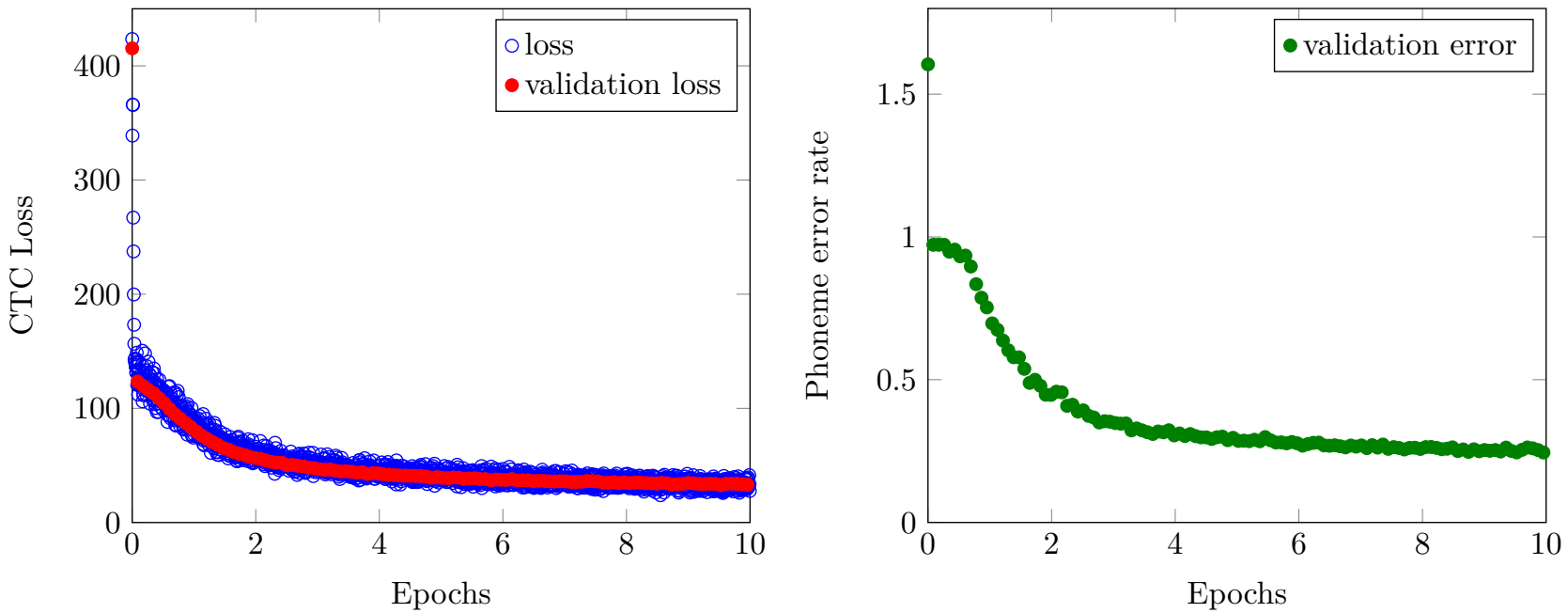

Figure 5 : Listener with attached CTC.

Greedy Decoding

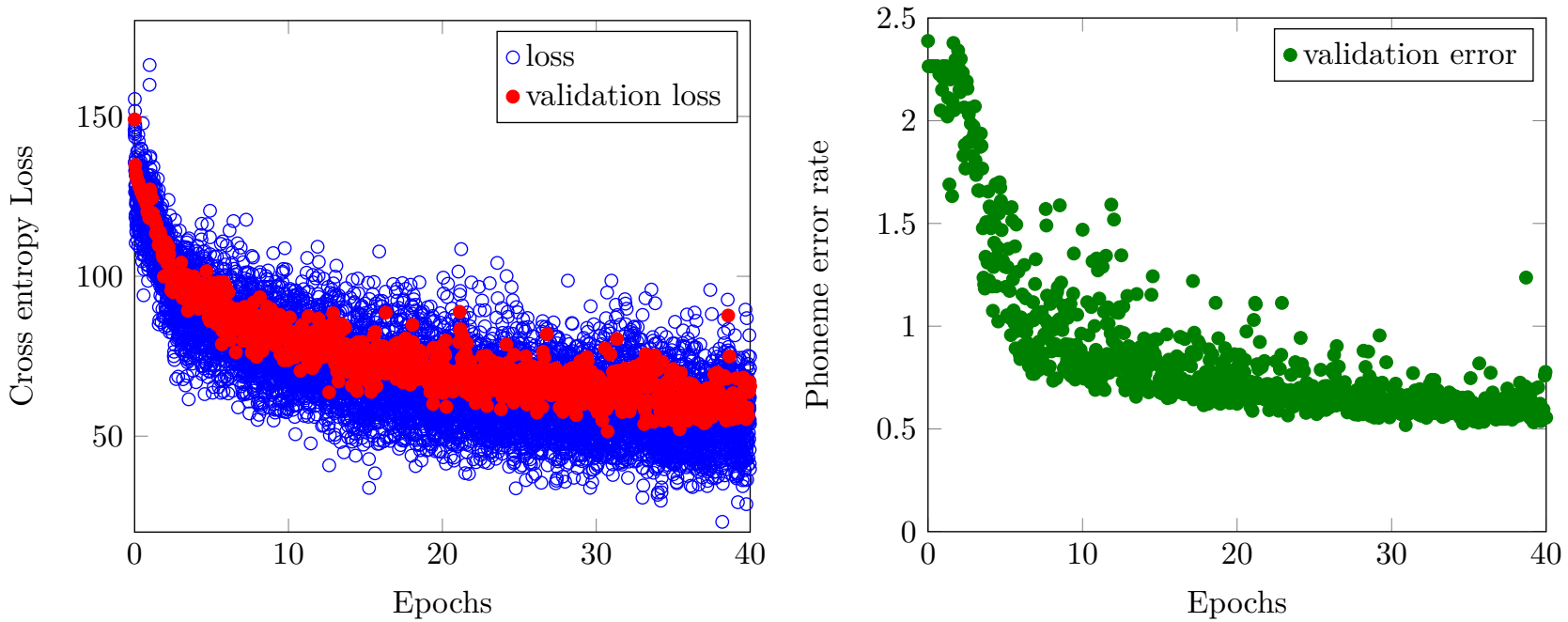

Figure 6 : The training progress shown for the full las architecture with greedy decoding, over 40 epochs, network output reuse probability 0.7 and input noise standard deviation 0.65 .

Alignment plots

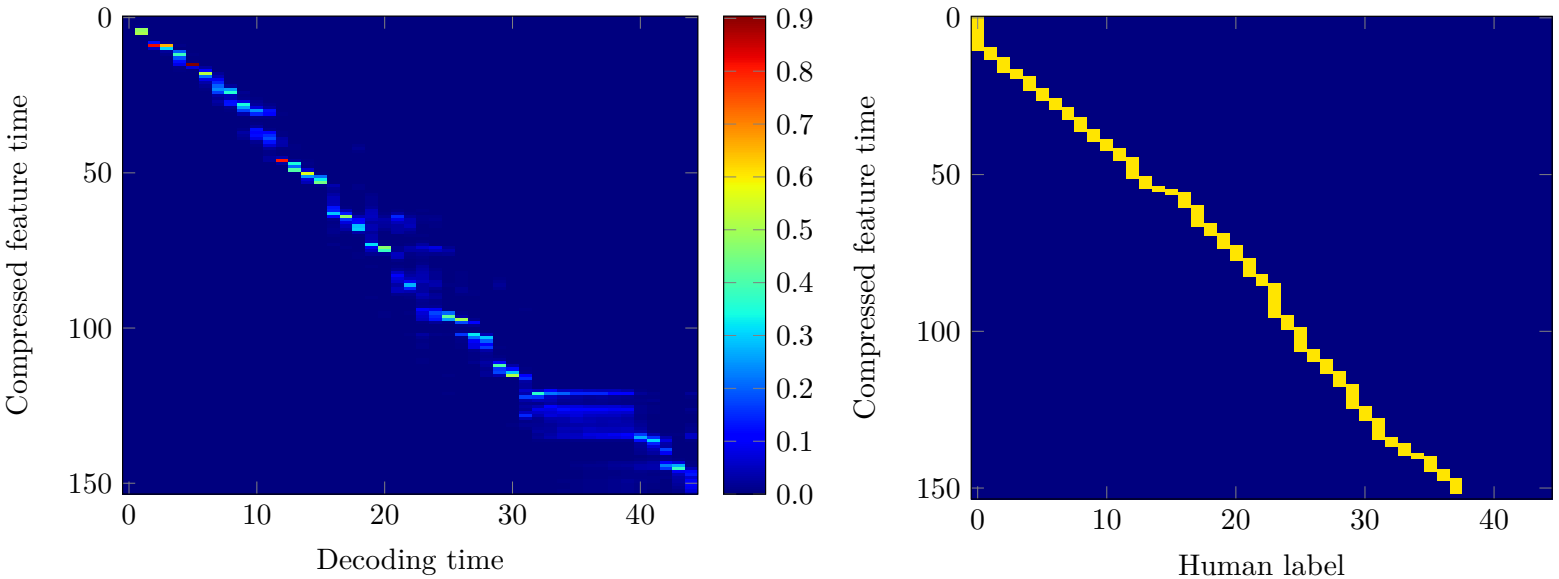

Figure 7 : Plot of the alignment vectors computed by the network for all 45 labels assigned to timit utterance `fmld0_sx295` (left), and alignments assigned by a human listener (right).

Alignment plots

Target labels

<sos> sil ih f sil k eh r l sil k ah m z sil t ah m aa r ah hh ae v er r ey n jh f er m iy dx iy ng ih sil t uw sil <eos>

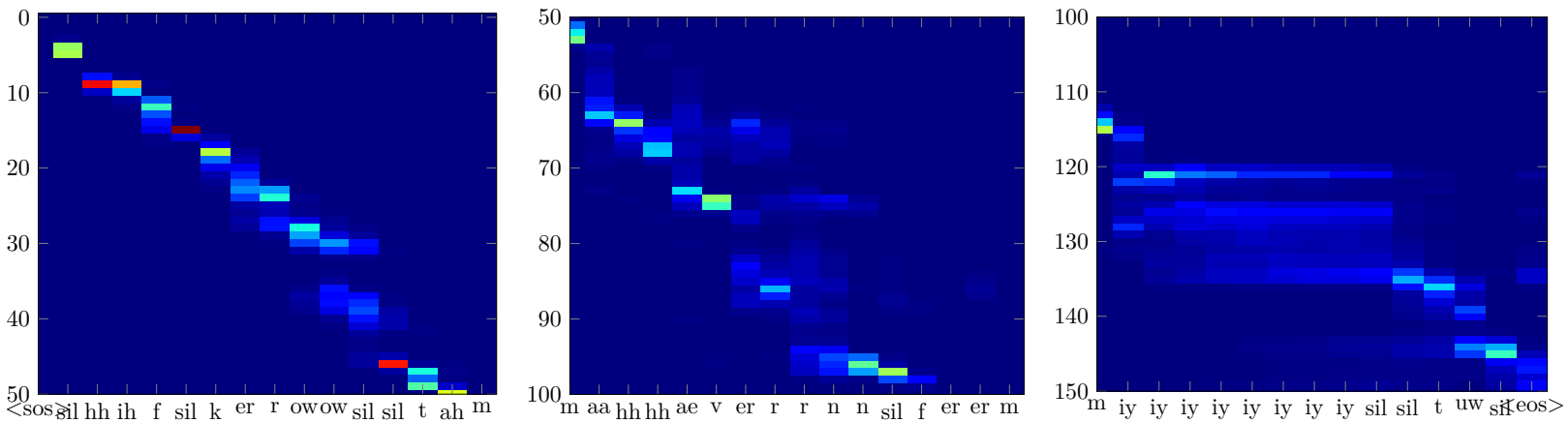

Figure 8 : Attention weights α and network output.

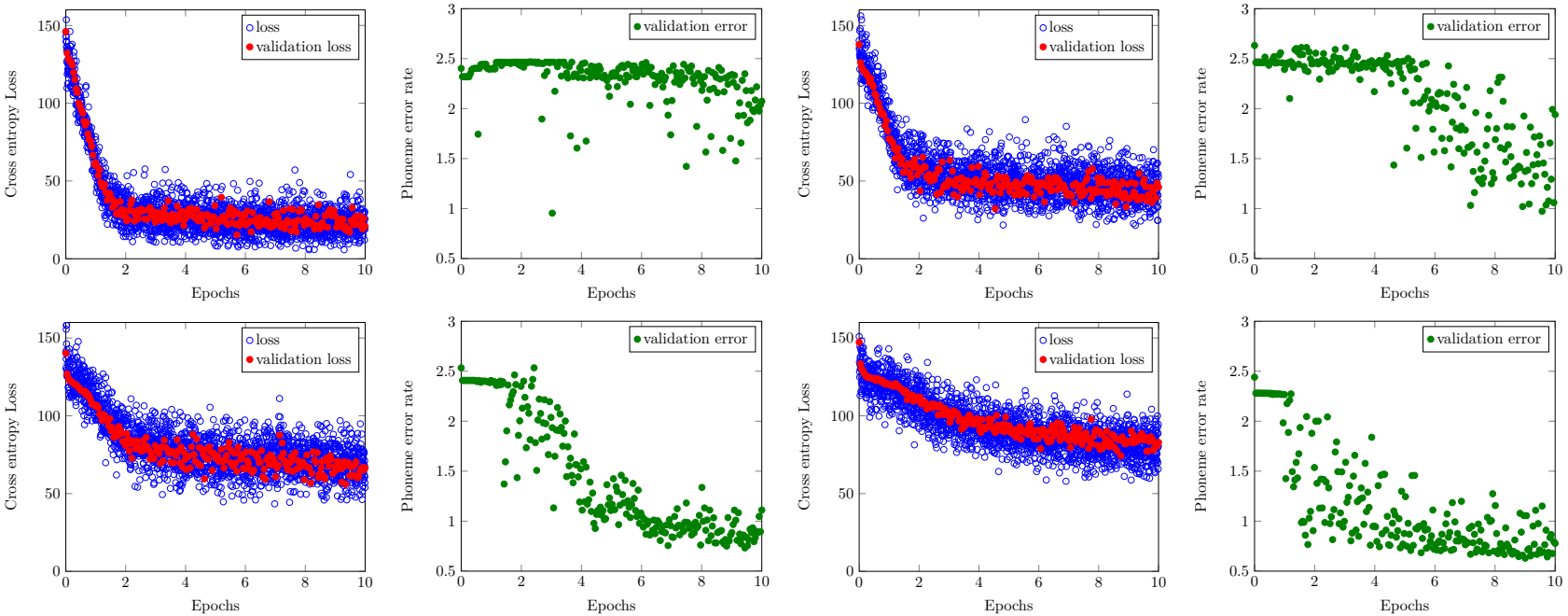

Figure 9 : Repetitions of the same experiment with increasing network output reuse probabilities $0.2, 0.4, 0.6, 0.8$, one experiment per row.

Dropout las with beam search

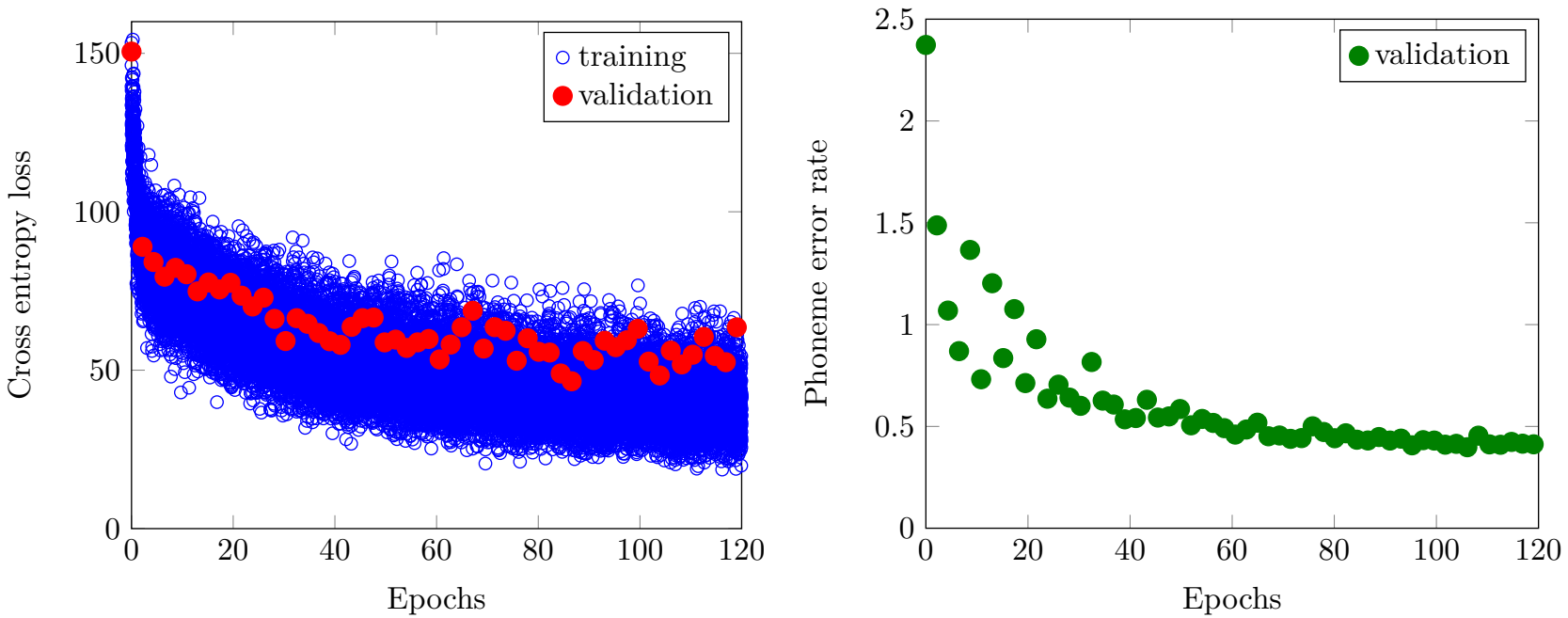

Figure 10 : Dropout las results.

Default attention

TODO

Conclusion

TODO

Questions

Thank's for your attention. Questions?
Now, or later `moritz@wolter.tech`.