PROJECT RESULTS

Hierarchical Multiscale Recurrent Neural Networks (HMRNN/HMLSTM)

Clemens Kriechbaumer

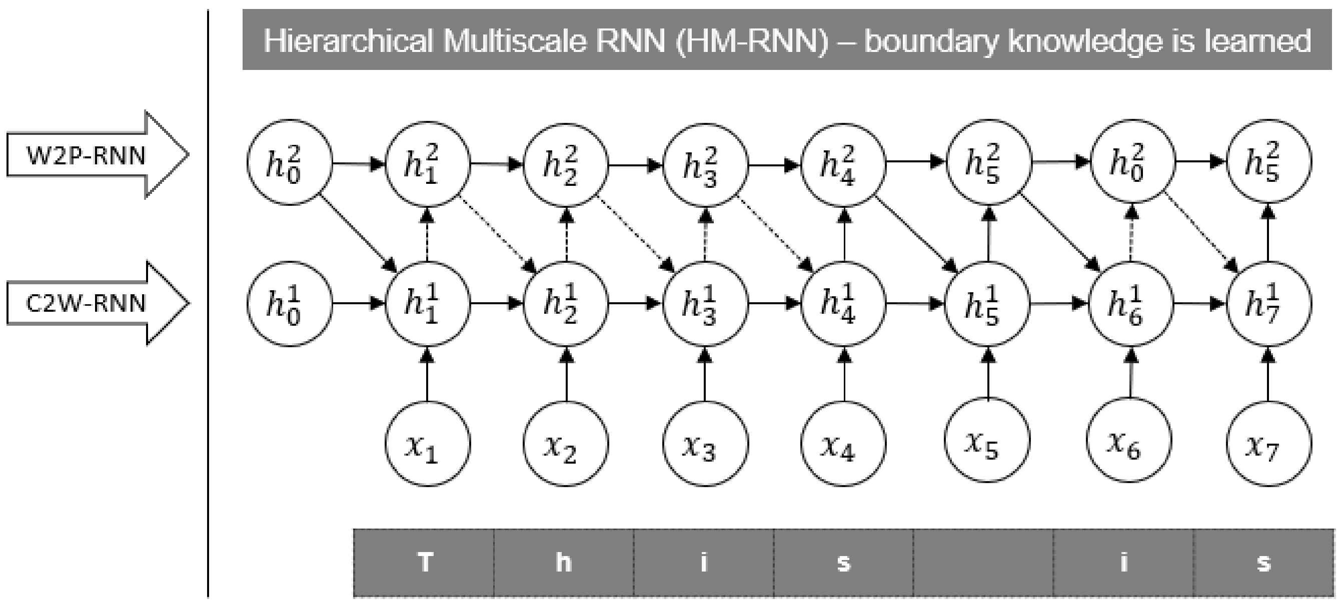
OVERVIEW

- RECAP HM-RNN ARCHITECTURE
- RESEARCH QUESTION
- METHOLOGY
- IMPLEMENTATION DETAILS
- RESULTS
- CONCLUSION

HM-RNN LEARNS BOUNDARY KNOWLEDGE

information flows

information is not flowing (this is learned, not necassary for the network)



HM-RNN KEY PRINCIPLES

- Parametrized boundary detector $z \in \{0, 1\}$ for each layer and timestamp
  - If $z_{t-1}^{l} = 1$, model consider this as end of segment (e.g. word or phrase)
  - Feeds the summarized representation (h … hidden state) to upper layer
  - Learns when a segment should end according to the target objective

- Boundary detector determines one of the following operation (based on the LSTM update rule)

UPDATE $(if\ z_{t-1}^{l} = 0\ and\ z_{t}^{l-1} = 1)$

$$c_t^l = f_t^l \odot c_{t-1}^l + i_t^l \odot g_t^l$$

$$h_t^l = o_t^l \odot tanh(c_t^l)$$

COPY $(if\ z_{t-1}^{l} = 0\ and\ z_{t}^{l-1} = 0)$

$$c_t^l = c_{t-1}^l$$

$$h_t^l = h_{t-1}^l$$

FLUSH $(if\ z_{t-1}^{l} = 1)$

$$c_t^l = i_t^l \odot g_t^l$$

$$h_t^l = o_t^l \odot tanh(c_t^l)$$

HM-RNN KEY PRINCIPLES

- Top-down connection

$$W_{l-1}^{l} \in \mathbb{R}^{\left(4\dim\left(h^{l}\right)+1\right)\times\dim\left(h^{l-1}\right)}$$

$$U_{i}^{j} \in \mathbb{R}^{\left(4\dim\left(h^{j}\right)+1\right)\times\dim\left(h^{i}\right)}$$

$$s_t^{recurrent(l)} = U_l^l\, h_{t-1}^l$$

$$s_t^{top-down(l)} = z_{t-1}^l U_{l+1}^l\, h_{t-1}^{l+1}$$

$$s_t^{bottom-up(l)} = z_t^{l-1} W_{l-1}^l\, h_t^{l-1}$$

$$\begin{pmatrix} f_t^l \\ i_t^l \\ o_t^l \\ g_t^l \\ \tilde{z}_t^l \end{pmatrix} = \begin{pmatrix} sigm \\ sigm \\ sigm \\ tanh \\ hard\ sigm \end{pmatrix} f_{slice}\left(s_t^{recurrent(l)} + s_t^{top-down(l)} + s_t^{bottom-up(l)} + b^{(l)}\right)$$

- Calculation of boundary detector (z)

$$hard\ sigm = max(0, min(1, \frac{ax+1}{2}))$$

$$z_t^l = f_{bound}(\tilde{z}_t^l)$$

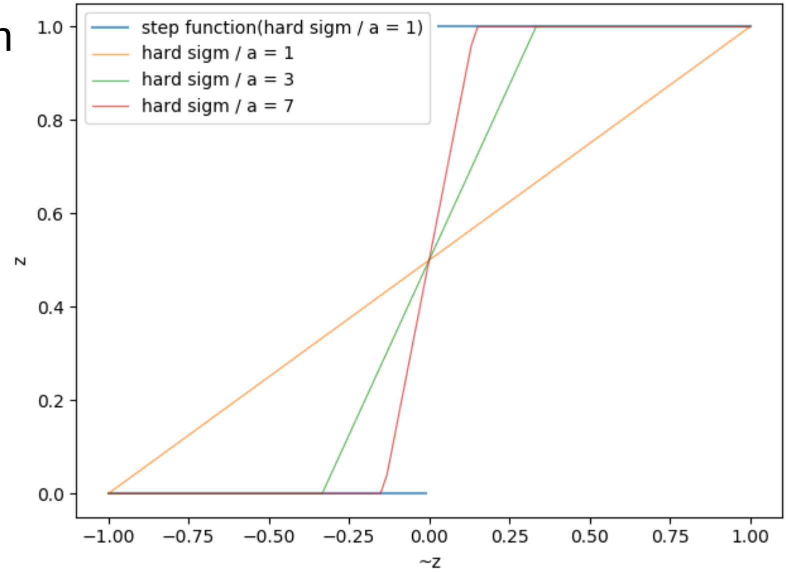
$$z_t^l = \begin{cases} 1\ if\ \tilde{z}_t^l > 0.5 \\ 0\ otherwise \end{cases}$$

JKU

HM-RNN KEY PRINCIPLES

- Boundary state is determined by a step function (non differentiable)
- Replace $f_{bound}\ (\sim z_t^l)$ with $f_{identity}\ (\sim z_t^l)$ during backpropagation
- Gradient of hardsigm w.r.t its input

$$\frac{\partial}{\partial x}\ max(0, min(1, \frac{ax+1}{2})) = \begin{cases} \frac{a}{2}\ if\ -1 < ax < 1 \\ 0\ otherwise \end{cases}$$



- Use slope annealing trick
  - Reduce discrepancy between the function by tuning hyperparameter a (slope)
  - In practice start with *a = 1* and slowly increase it per epoch *max(5, a)*

RESEARCH QUESTIONS / PROJECT WORK

- Implement HMLSTM architecture in Pytorch
- Compare basic predictive capability/performance with baseline LSTM
- Try measure boundary detection capability using empirical metrics
- Test/visualize boundary detection behavior on different layer architectures (pyramid/cone)

IMPLEMENTATION DETAILS

- Calculation of cell and hidden states (UPDATE, COPY, FLASH)

```
def forward(self, input: HMLSTMState):

    i, g, o, f, sz = self.calc_gates(input)

    c = torch.where(
        torch.eq(input.z, 1),
        i * g,  # flush
        torch.where(
            torch.eq(input.z_bottom, 0),
            input.c,  # copy
            input.c * f + i * g  # update
        )
    )

    h = torch.where(
        torch.eq(input.z, 0) & torch.eq(input.z_bottom, 0),
        input.h,  # copy
        torch.tanh(c) * o  # update / flush
    )

    z = self.calc_z(sz)

    return h, c, z
```

https://github.com/yongwu33/hmlstm

IMPLEMENTATION DETAILS

- Calculation of z

```python
class _CalcZ(nn.Module):
    def __init__(self, a: int = 1, th: float = 0.5):
        super(_CalcZ, self).__init__()

        self.round = Round(th)
        self.hardsigm = HardSigm(a)

    def forward(self, sz: torch.Tensor) -> torch.Tensor:
        z_tilde = self.hardsigm(sz)
        z = self.round(z_tilde)

        return z
```

- HardSigm and Round

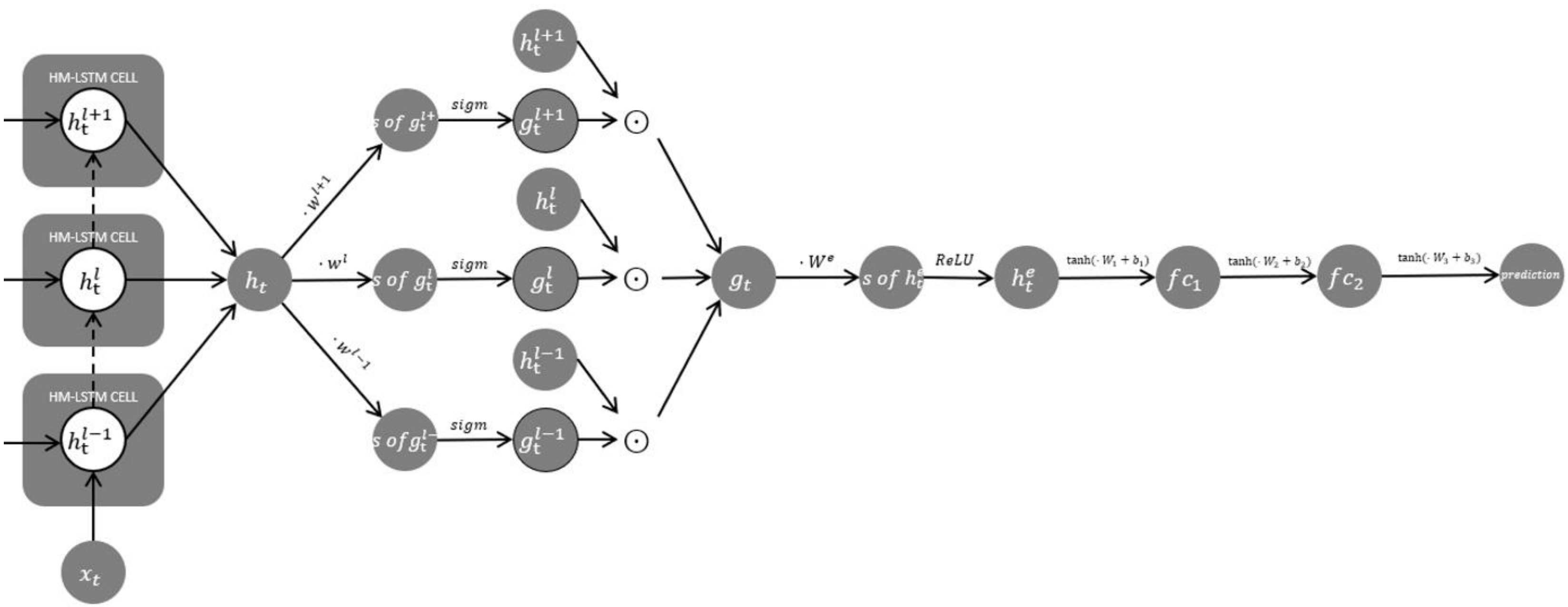
```python
class HardSigm2(nn.Module):
    """fastest version"""

    def __init__(self, a: int = 1):
        super(HardSigm2, self).__init__()

        self.a = Parameter(torch.scalar_tensor(a), requires_grad=False)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        t = 0.5 * (self.a * x + 1)

        return torch.clamp(t, min=0, max=1)
```

```python
class _Round(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x: torch.Tensor, th: float) -> torch.Tensor:
        # x[x >= th] = 1
        # x[x < th] = 0

        # x = torch.where(
        #     x >= th,
        #     torch.scalar_tensor(1, device=x.device),
        #     torch.scalar_tensor(0, device=x.device)
        # )

        # fastest version
        x = (x > th).float()

        return x

    @staticmethod
    def backward(ctx, grad_output: torch.Tensor):
        dth = None  # indeterminable
        dx = grad_output  # identity/pass through gradient

        return dx, dth
```

https://github.com/yongwu33/hmlstm

IMPLEMENTATION DETAILS

- Output Model



DATASET

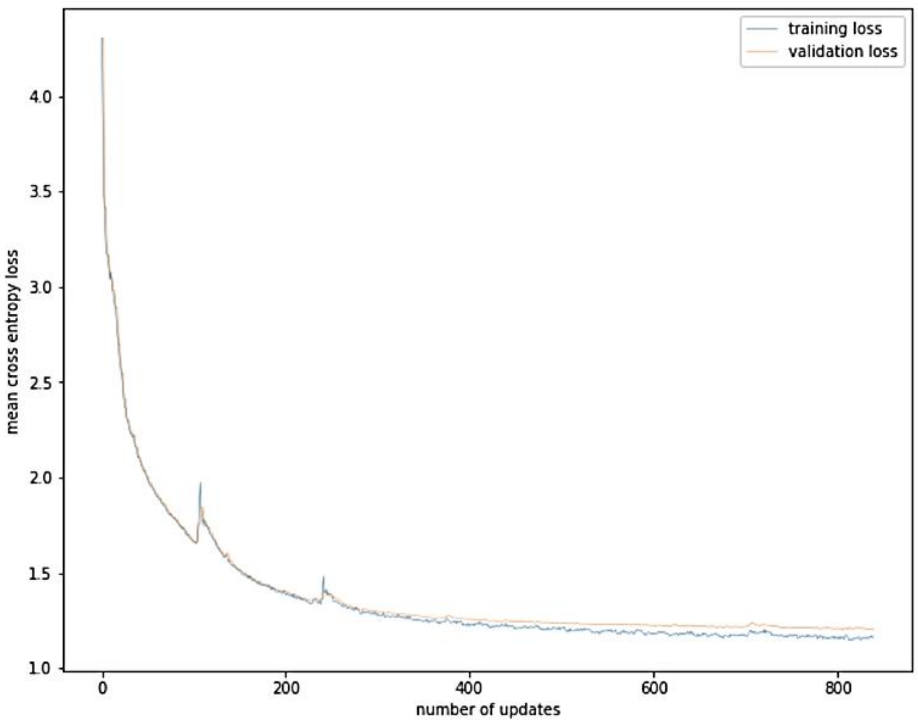
- Shakespeare Dataset (~10MB of text)
  - The following is a excerpt from “All’s Well That Ends Well”

***** ACT II *****
***** SCENE I. Paris. The KING's palace. *****
    Flourish of cornets. Enter the KING, attended with divers young Lords
    taking leave for the Florentine war; BERTRAM, and PAROLLES
KING
    Farewell, young lords; these warlike principles
    Do not throw from you: and you, my lords, farewell:
    Share the advice betwixt you; if both gain, all
    The gift doth stretch itself as 'tis received,
    And is enough for both.
First Lord
    'Tis our hope, sir,
    After well enter'd soldiers, to return
    And find your grace in health.

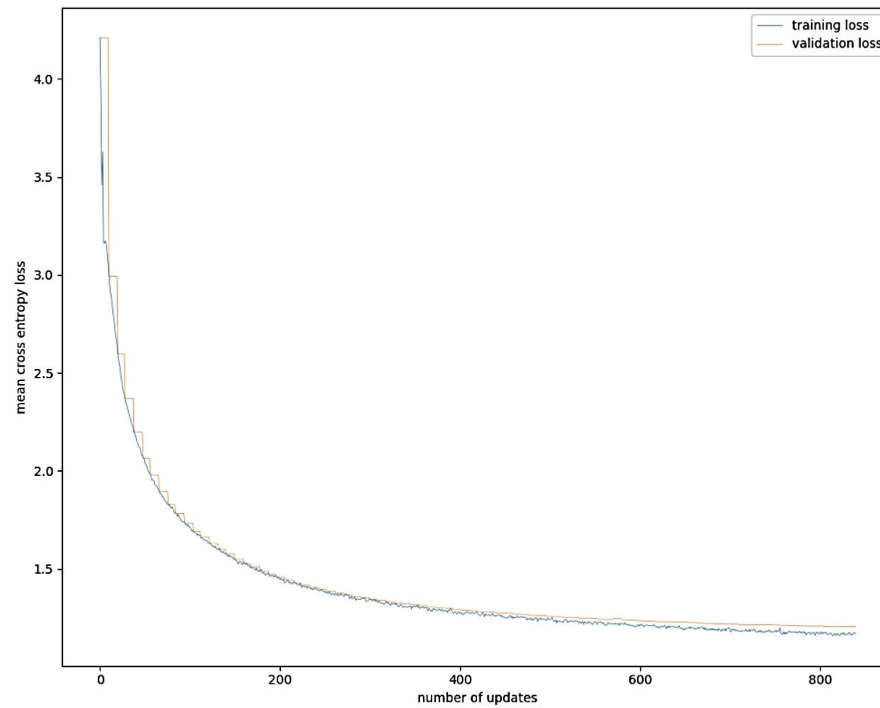
http://shakespeare.mit.edu/

COMPARISON TO BASELINE (LSTM)

- **HMLSTM** (best result)
  - Hidden layer sizes: 114, 204, 182
  - Number of trainable parameters: 1082868
  - Mean cross entropy error ~1.2



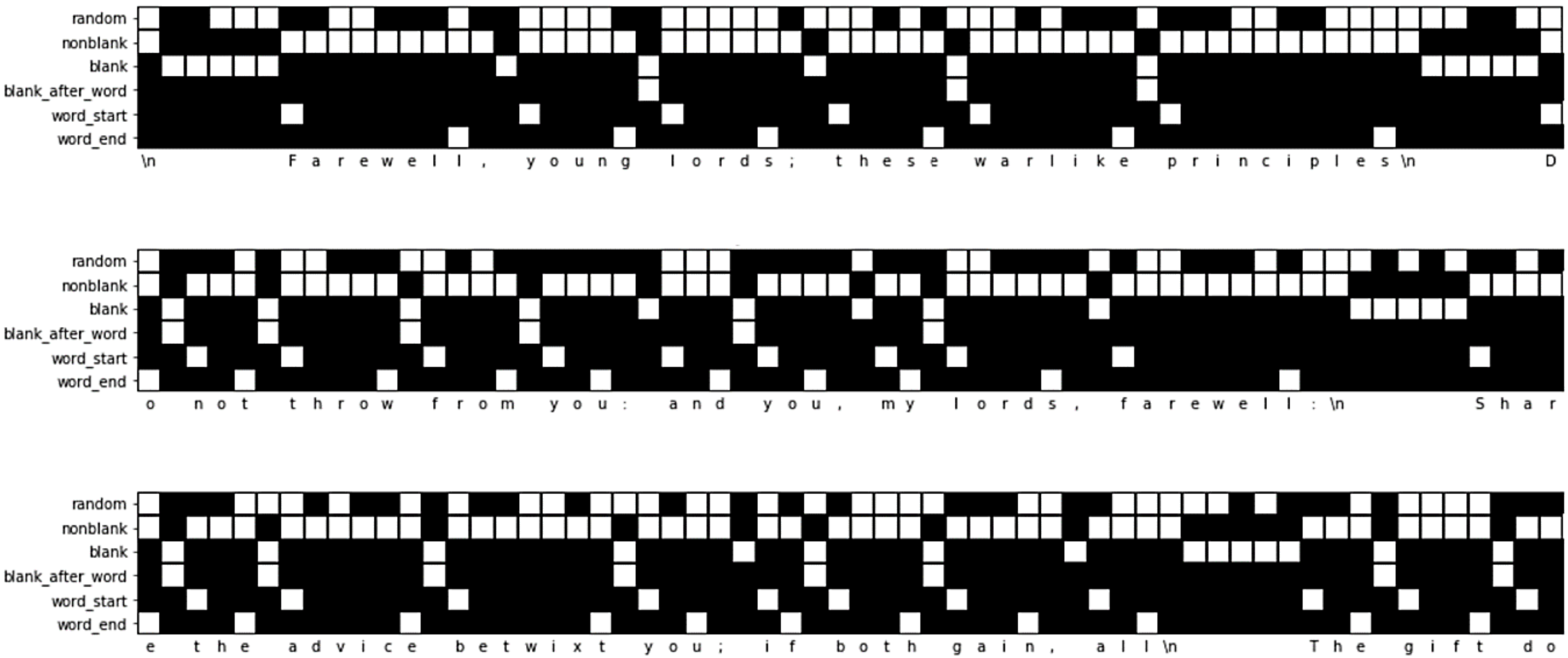
- **LSTM**
  - Hidden layer size: 450
  - Number of trainable parameters: 1083372
  - Mean cross entropy error ~1.2



JKU

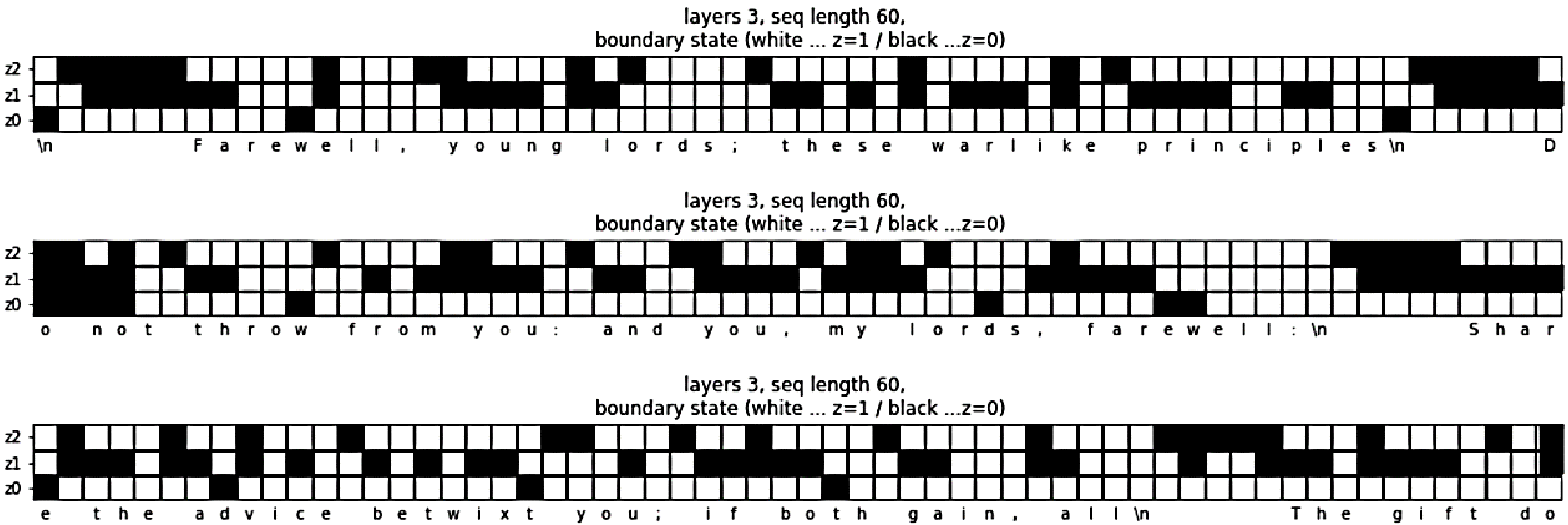
REFERENCE BOUNDARIES / METRICS

Farewell, young lords; these warlike principles
Do not throw from you: and you, my lords, farewell:
Share the advice betwixt you; if both gain, all
The gift do



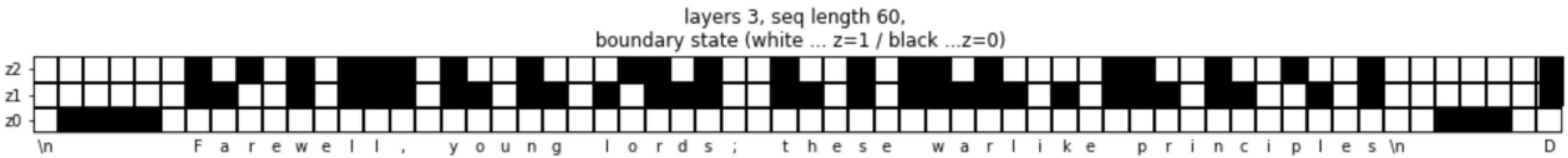
RESULTS - LEARNED BOUNDARIES

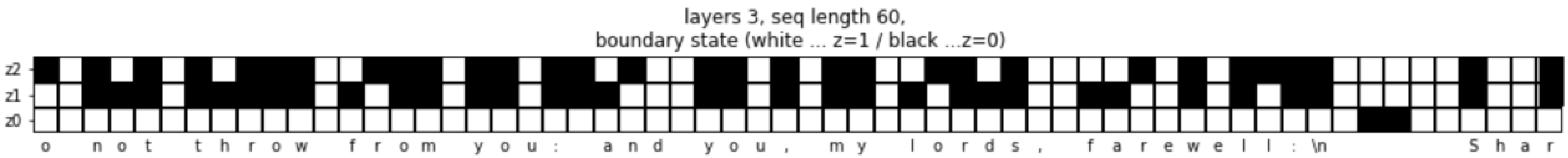
HMLSTM (114, 204, 182)

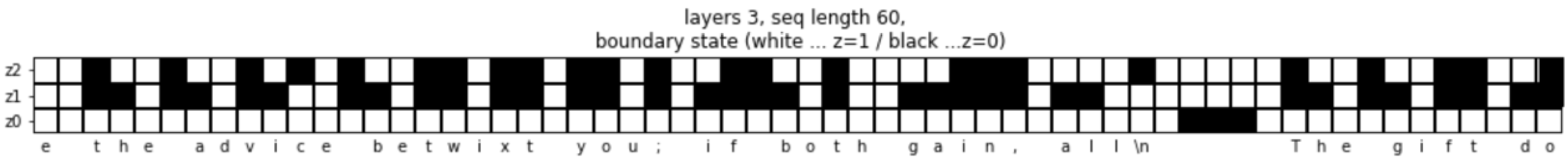


RESULTS - LEARNED BOUNDARIES

HMLSTM (256, 128, 64)

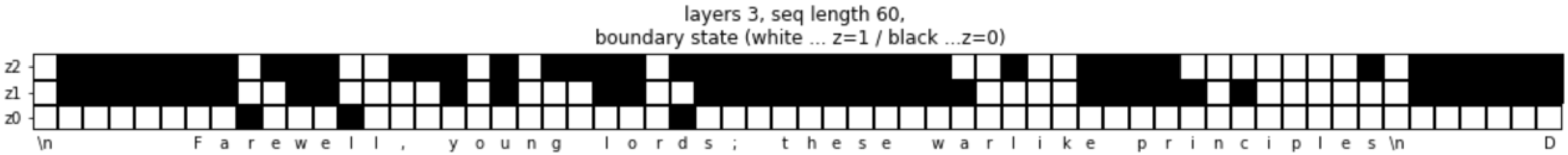


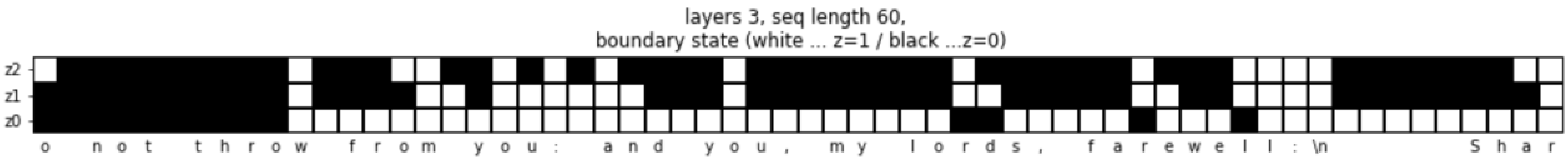


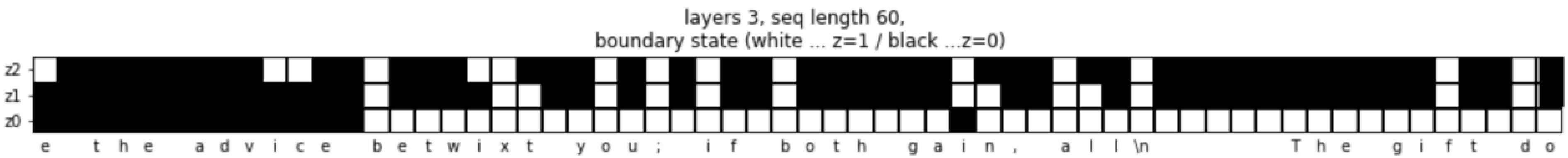


RESULTS - LEARNED BOUNDARIES

HMLSTM (256, 256, 256)

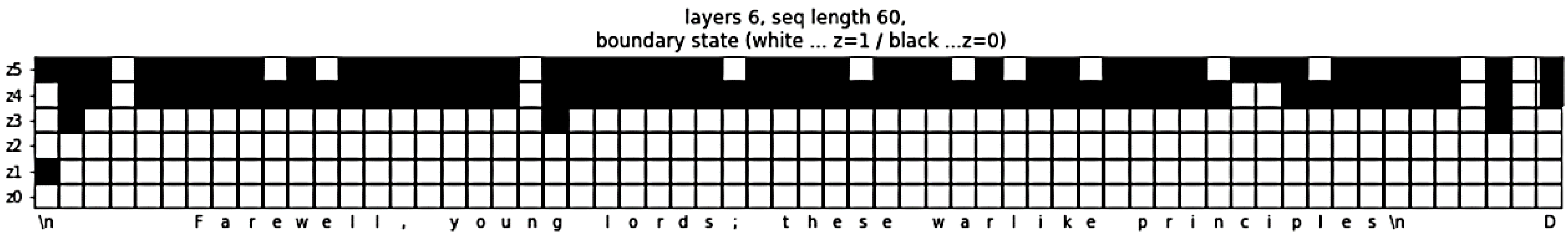


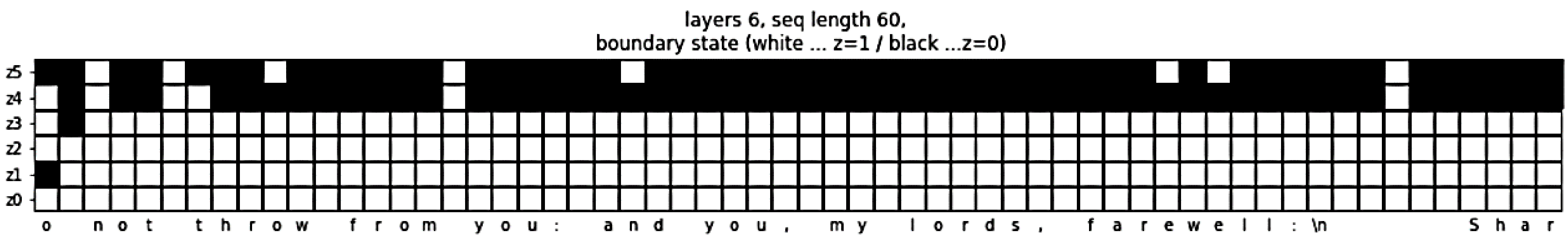


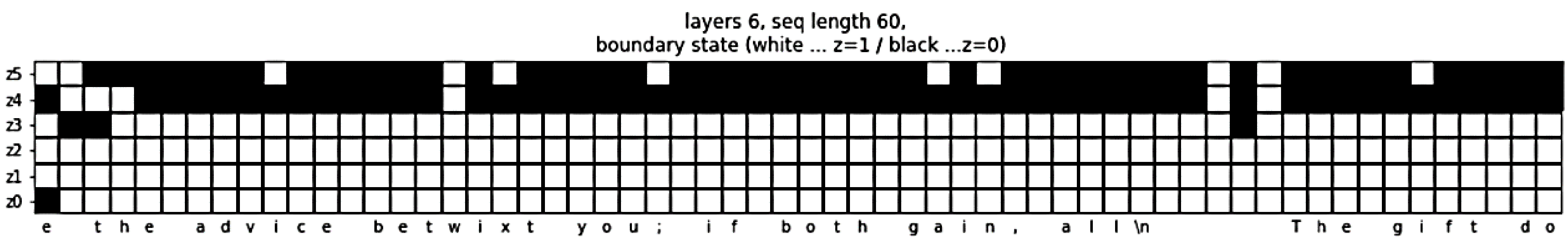


RESULTS - LEARNED BOUNDARIES

HMLSTM (160, 120, 100, 80, 60, 40)





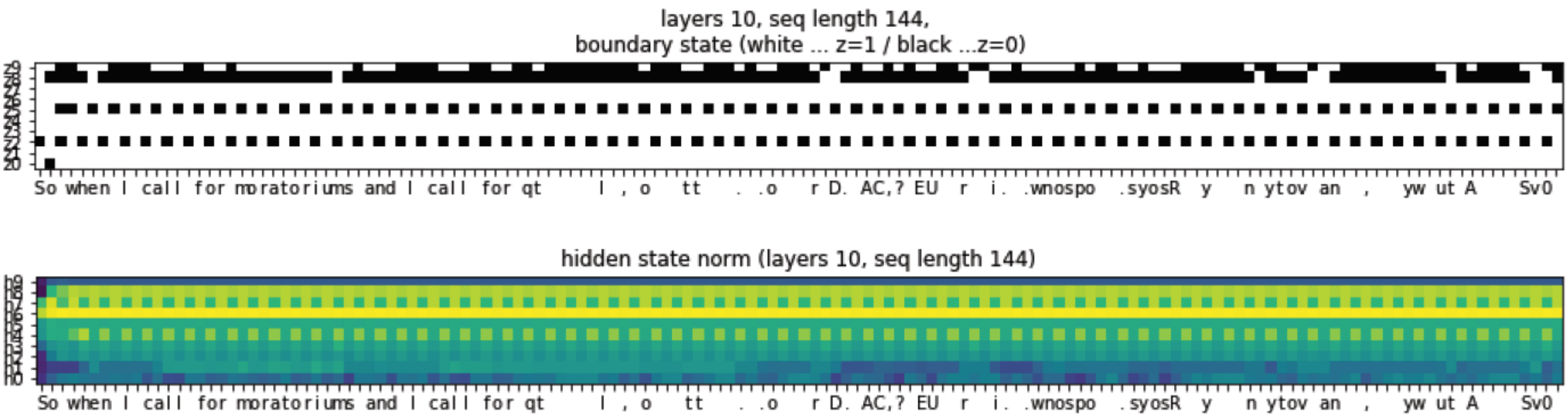


REFERENCE BOUNDARIES / METRICS

	word end boundary f1 score	word start boundary f1 score	blank boundary f1 score
Layers: 114, 204, 182			
z0	0.341895	0.277958	0.000796
z1	0.409628	0.085733	0.193027
z2	0.260545	0.274459	0.377622
Layers: 256, 128, 64			
z0	0.212748	0.167157	0.353526
z1	0.227303	0.025567	0.337433
z2	0.299610	0.287009	0.325131
Layers: 256, 256, 256			
z0	0.267679	0.058115	0.001736
z1	0.314568	0.073979	0.074297
z2	0.279764	0.262203	0.367048
Layers: 160, 120, 100, 80, 60, 40			
z0	0.289033	0.006389	0.100464
z1	0.136911	0.018611	0.094528
z2	0.286560	0.276549	0.371717
z3	0.287879	0.287856	0.391062
z4	0.288636	0.285700	0.387099
z5	0.286681	0.287159	0.391066

RESULTS – LAYER ARCHITECTURE

HMLSTM (10, 20, 30, 40, 50, 60, 70, 80, 90, 100)



CONCLUSION

- Architecture identifies boundaries – but not necessarily always human interpretable boundaries
- Therefore it is questionable if this architecture is useful at all
- Has similar performance (predictive) than a LSTM but is harder to train/more difficult to optimize for
- Gives a good insight in the difficulty to handle district variables within machine learning
- Interesting ability to visualize information flow within a stacked LSTM/RNN – maybe better insight/better interpretability

Thank you for your attention

Clemens Kriechbaumer