(Re)visiting Seq2Seq: Application Beyond Machine Translation

Pete Tunkis

Data Scientist @ Arcurve, Inc.

May 26, 2021

ARCURVE

Agenda

- Intro: why the topic?
- Why Seq2Seq?
- Seq2Seq primer
- Caveats, use cases

A little bit of background…

Why the topic? Where'd this come from?

Background

- It all began in the run-up to YYCDataCon...
- Many potential extensions to m
  physical systems in a graph
- What about addressing the data collection piece?

ARCURVE

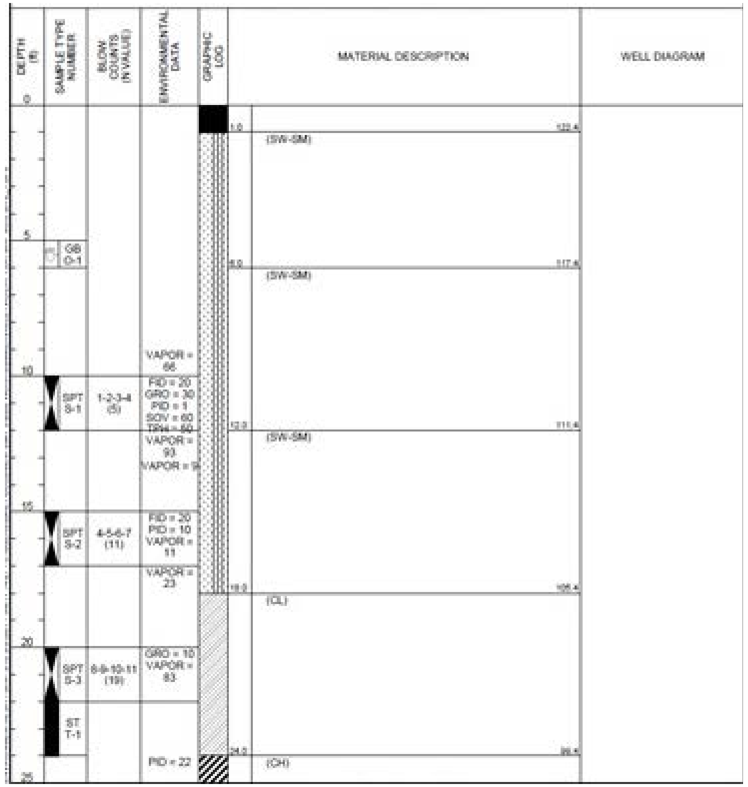
Background

- Toward a Proof of Concept
  - Can we give a helping hand with data collection/digitization?

- *"If you have a good understanding of the regional geology, you could most likely predict the next stratigraphic layer"*

  – A geologist buddy in Ft. Mac

- Potential usefulness across project lifespans?
  - Data (or lack thereof) have a habit of coming back to haunt…



ARCURVE

Background

- Borehole samples: lithology "sentences"
  - Implied patterns or order

- Work is out there on classifying sequences, what about predicting them?
  - E.g., Rokach & Maimon (2007) and more

- Why not research and apply an NLP technique that deals with aligning and translating explicitly order-dependent sequences
  - NMT, chat-bots

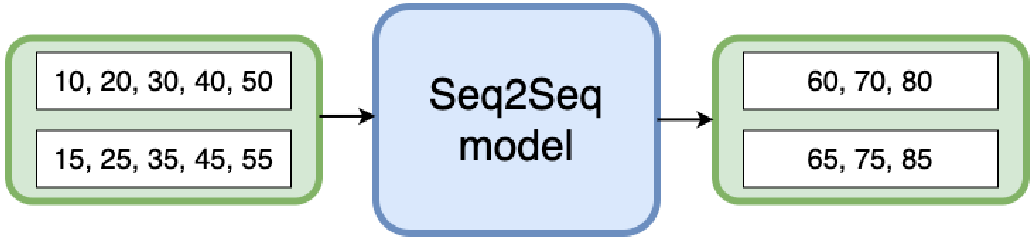
Enter Seq2Seq

Why this approach?

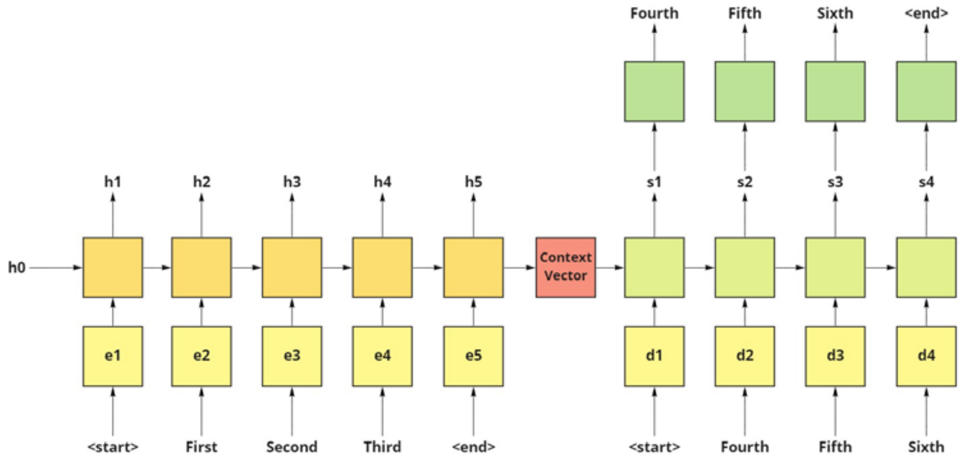
ARCURVE

Why Seq2Seq?

- Seq2Seq = “Sequence to Sequence”
  - A.k.a. Encoder-Decoder network model



- Quick summary:
  - Two Recurrent Neural Networks (RNN)
  - Two are better than one



- Emphasis on sequences and their order

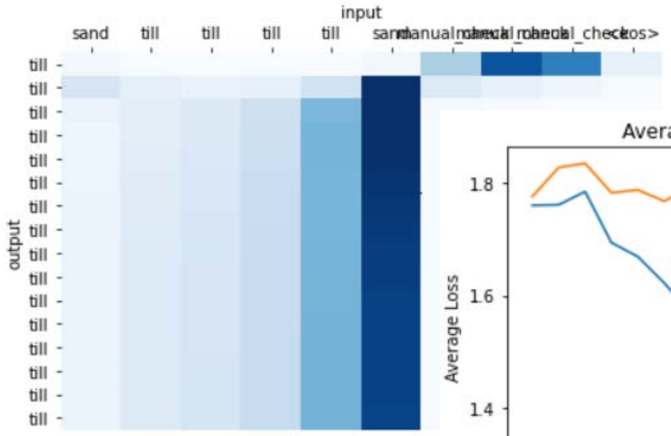
- More than just a translator/chat-bot tool?

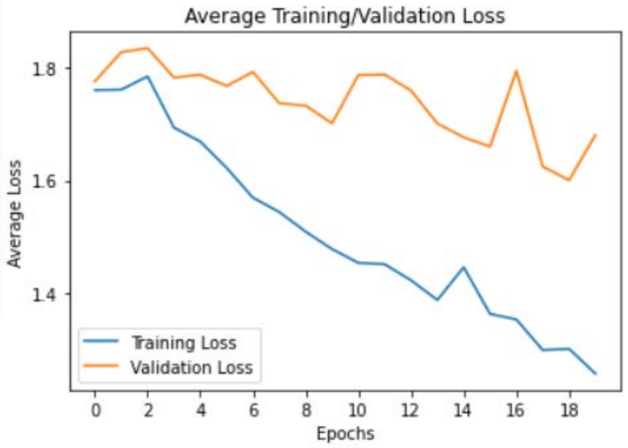
Why Seq2Seq?

- First attempt @ YYC DataCon
  - Thanks for the feedback! It’s a work-in-progress

- Used PyTorch
  - Seems more manipulable *prima facie*
  - API has a lot of built-in functions
  - Lots of easy-to-follow tutorials and follow-ups out there

- First crack wasn’t *super* successful, but technically it worked
  - Currently working on improvements, incorporating new/better data
  - …more reference to this later





Seq2Seq: A Primer

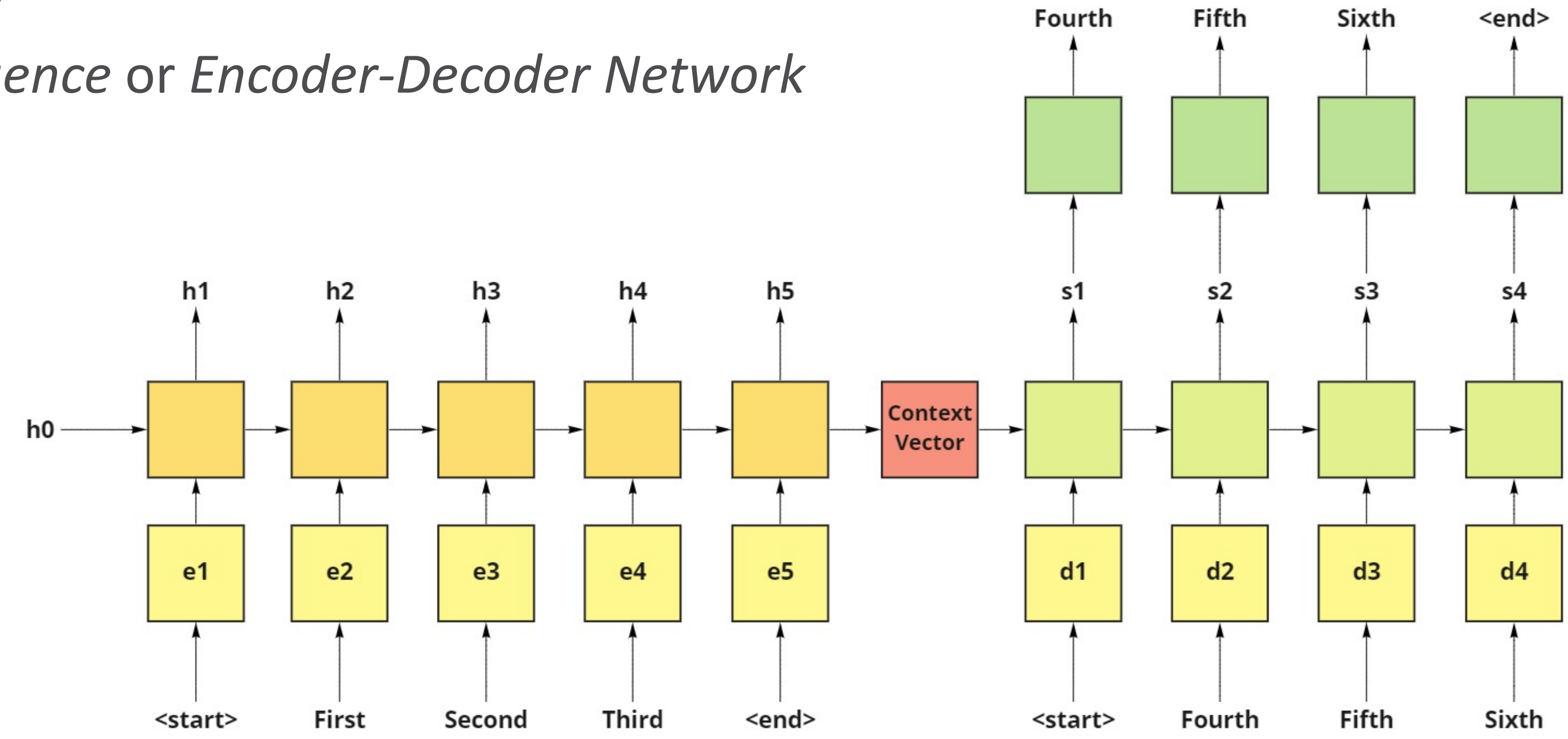
An overview (or refresher) on how it works, plus some other useful details

ARCURVE

Seq2Seq: A Primer

- What is Seq2Seq?
  - *Sequence to Sequence* or *Encoder-Decoder Network*
  - Two RNNs

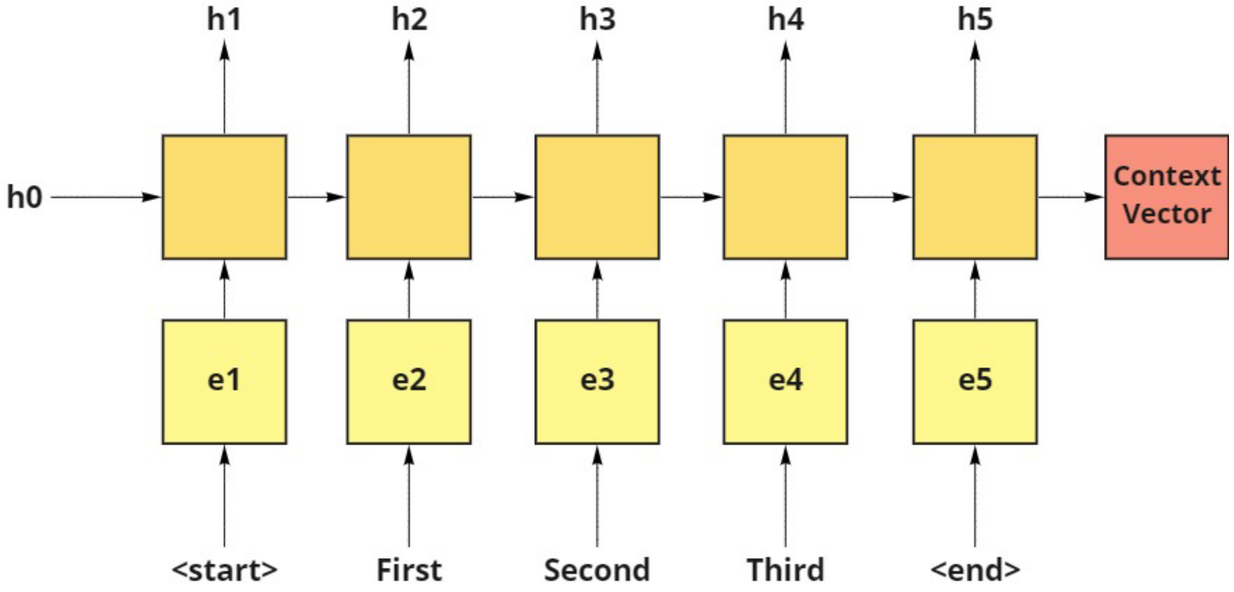
- How does it work?
  - Embed
  - Encode
  - Decode
  - Translate



ARCURVE

Seq2Seq: A Primer

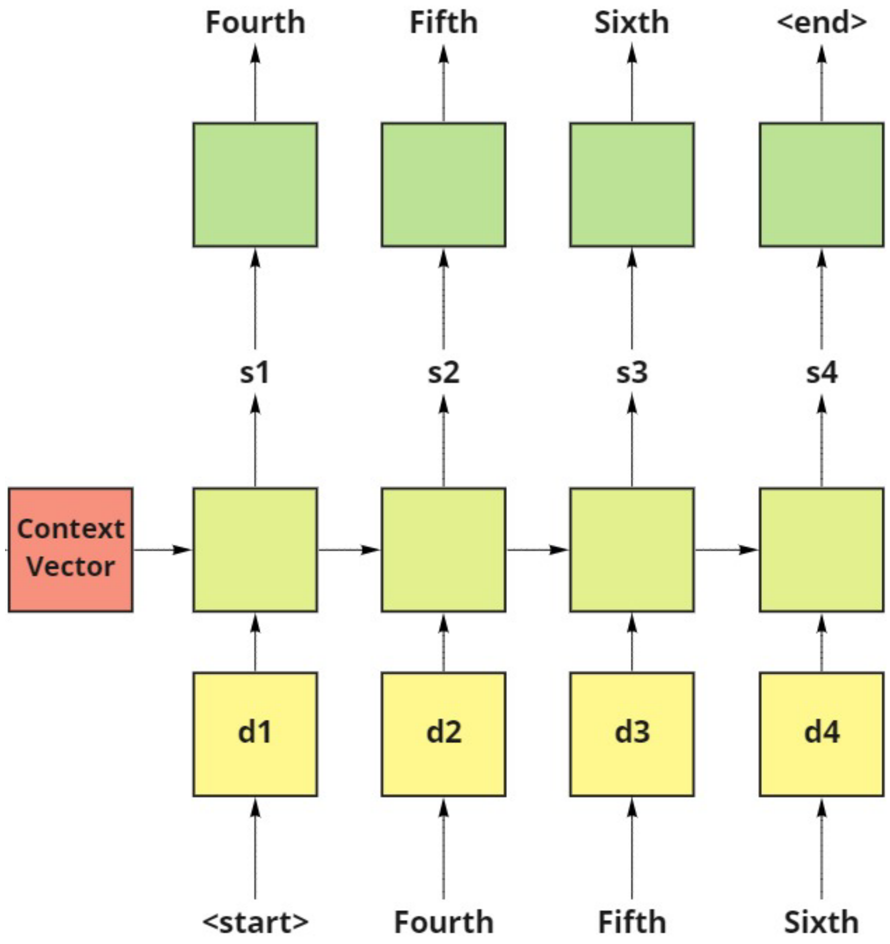
- Input
  - Append tags for functionality
  - Embed → Encode
    - Encoding using Long Short-Term Memory or Gated Recurrent Unit
    - …more on this later
  - Inputs are embeddings (e) and hidden states (h)
  - Hidden states updated
    - Cumulative
  - Context vector = “final” input hidden state



ARCURVE

Seq2Seq: A Primer

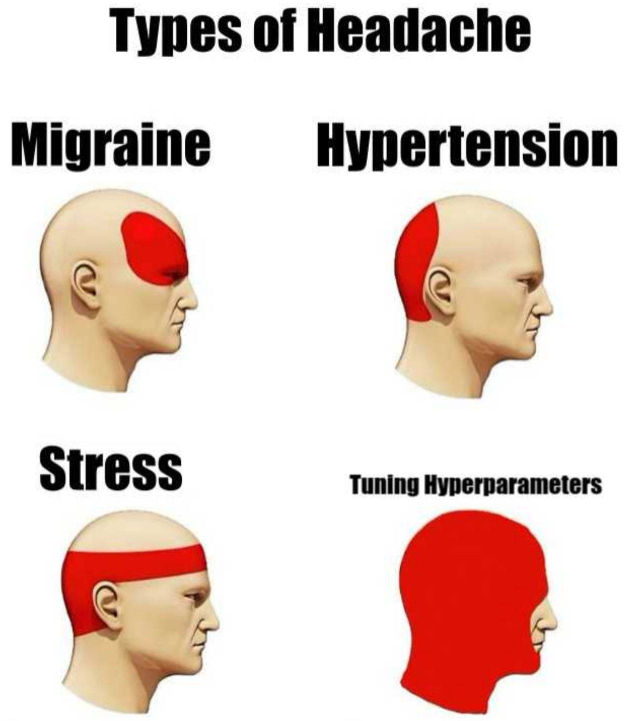
- Output
  - Embed (d) and decode with output hidden states (s)
    - NB: embedding layers for input/output are distinct!
  - Context vector = “initial” hidden state
  - Outputs generated sequentially
    - In training: teacher forcing?
  - Stop at <end>
    - In training: evaluate predicted vs known output, calculate loss, update model parameters

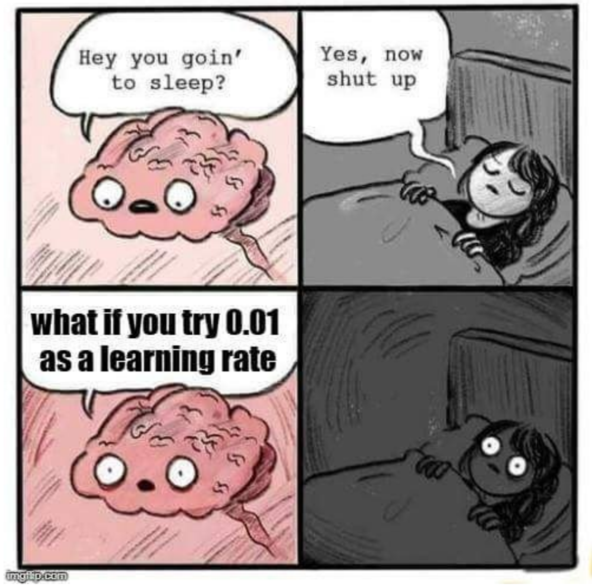


ARCURVE

Seq2Seq: A Primer

- Plenty of Options
  - Hyperparameters
    - Teacher-forcing ratio/on-off
    - Number of iterations/epochs
    - Learning rate
    - Hidden size/number of nodes
    - Number of RNN layers
  - Processes
    - Network weight optimizer
    - GRU vs LSTM for your RNNs
    - Attention strategy for decoders





ARCURVE

Seq2Seq: A Primer

- GRU vs LSTM for your RNNs
  - Address the vanishing gradient problem

- GRU: Gated Recurrent Unit
  - Less complex
  - Better with shorter sequences, smaller data
  - More computationally efficient
  - Can be fairly easy to modify
  - Hidden content is exposed (not hidden in memory unit)

- LSTM: Long Short-Term Memory
  - More accurate esp. with longer, context-rich sequences
  - Can require larger data
  - More complex, computationally expensive
  - Memory unit and hidden content not exposed

```python
class EncoderRNN(nn.Module):
    def __init__(self, itos:list, hidden_size:int, n_layers:int=1, dropout:float=0.1, emb_vecs:dict=None):
        super(EncoderRNN, self).__init__()

        self.input_size = len(itos)
        self.hidden_size = hidden_size
        self.n_layers = n_layers
        self.dropout = dropout
        self.embedding = self._init_embedding(itos, self.hidden_size, emb_vecs)
        self.gru = nn.GRU(self.hidden_size, self.hidden_size, self.n_layers, dropout=self.dropout, bidirectional=True)

    def _init_embedding(self, itos:list, em_sz:int, emb_vecs:dict=None):
        emb = nn.Embedding(len(itos), em_sz, padding_idx=1)
        if emb_vecs is not None:
            wgts = emb.weight.data
            miss = []
            for i, w in enumerate(itos):
                try:
                    wgts[i] = torch.from_numpy(list(emb_vecs[w]))
                except:
                    miss.append(w)
            print(f"Encoder embedding vector didn't have {len(miss)} tokens, example: {miss[5:10]}")
        return emb

    def forward(self, input_seqs:torch.tensor, input_lengths:torch.tensor, hidden:torch.tensor=None):
        # Note: we run this all at once (over multiple batches of multiple sequences)
        embedded = self.embedding(input_seqs)
        packed = torch.nn.utils.rnn.pack_padded_sequence(embedded, input_lengths)
        outputs, hidden = self.gru(packed, hidden)
        outputs, output_lengths = torch.nn.utils.rnn.pad_packed_sequence(outputs)  # unpack (back to padded)
        outputs = outputs[:, :, :self.hidden_size] + outputs[:, :, self.hidden_size:]  # Sum bidirectional outputs
        return outputs, hidden
```

Seq2Seq: A Primer

- Attention decoders?
  - Align
  - Translate

- Types/characteristics
  - Global
  - Local
  - Hard
  - Soft

- Big names
  - Bahdanau et al. (2014)
  - Luong et al. (2015)

```
class LuongAttnDecoderRNN(nn.Module):
    def __init__(self, attn_model:str, itos:list, hidden_size:int, n_layers:int=1, dropout:float=0.1, emb_vecs:dict=None):
        super(LuongAttnDecoderRNN, self).__init__()

        # Keep for reference
        self.attn_model = attn_model
        self.hidden_size = hidden_size
        self.output_size = len(itos)
        self.n_layers = n_layers
        self.dropout = dropout
        # Define layers
        self.embedding = self._init_embedding(itos, self.hidden_size, emb_vecs)
        self.embedding_dropout = nn.Dropout(dropout)
        self.gru = nn.GRU(self.hidden_size, self.hidden_size, n_layers, dropout=dropout)
        self.concat = nn.Linear(self.hidden_size * 2, self.hidden_size)
        self.out = nn.Linear(self.hidden_size, self.output_size)

        # Choose attention model
        if attn_model != 'none':
            self.attn = Attn(attn_model, self.hidden_size)


    def forward(self, input_seq:torch.tensor, last_hidden:torch.tensor, encoder_outputs:torch.tensor):
        # Note: we run this one step at a time

        # Get the embedding of the current input word (last output word)
        batch_size = input_seq.size(0)
        embedded = self.embedding(input_seq)
        embedded = self.embedding_dropout(embedded)
        embedded = embedded.view(1, batch_size, self.hidden_size)  # S=1 x B x N

        # Get current hidden state from input word and last hidden state
        rnn_output, hidden = self.gru(embedded, last_hidden)

        # Calculate attention from current RNN state and all encoder outputs;
        # apply to encoder outputs to get weighted average
        attn_weights = self.attn(rnn_output, encoder_outputs)
        context = attn_weights.bmm(encoder_outputs.transpose(0, 1))  # B x S=1 x N

        # Attentional vector using the RNN hidden state and context vector
        # concatenated together (Luong eq. 5)
        rnn_output = rnn_output.squeeze(0)  # S=1 x B x N -> B x N
        context = context.squeeze(1)  # B x S=1 x N -> B x N
        concat_input = torch.cat((rnn_output, context), 1)
        concat_output = torch.tanh(self.concat(concat_input))

        # Finally predict next token (Luong eq. 6, without softmax)
        output = self.out(concat_output)
```

ARCURVE

Using Seq2Seq

Use cases and caveats

ARCURVE

Challenges to Keep in Mind

- Requires "a lot of" data
  - Can use pre-trained vectors to speed training along if you have them
- Depending on architecture choices, can be computationally expensive
- PyTorch made trial/error tests quick thanks to how easy it is to switch between cuda and cpu
  - …but not everyone has a good graphics card
  - …or access to super/cloud-compute power

OK, OK—Seq2Seq Use Cases

- RNNs are useful in a variety of contexts!
  - When data are temporal
  - Input lengths vary
  - Context of inputs matters
  - Values can be ordered but are not continuous
  - Data are interpretable and analyzed in discrete steps

- RNNs can work with different input/output contexts!
  - Many-to-one
  - Many-to-many
  - Generative tasks

Thanks!

Questions?

Let's Connect!

LinkedIn – https://www.linkedin.com/in/petertunkis
Email – peter.tunkis@arcurve.com

References and Readings

- Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.
- Bengio, S., Vinyals, O., Jaitly, N., & Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. *arXiv preprint arXiv:1506.03099*.
- Bengio, Y., Goodfellow, I., & Courville, A. (2017). *Deep learning* (Vol. 1). Massachusetts, USA: MIT press.
- Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.
- Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015, June). An empirical exploration of recurrent network architectures. In *International conference on machine learning* (pp. 2342-2350). PMLR.
- Kaiser, Ł., & Sutskever, I. (2015). Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*.
- Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Lamb, A., Goyal, A., Zhang, Y., Zhang, S., Courville, A., & Bengio, Y. (2016). Professor forcing: A new algorithm for training recurrent networks. *arXiv preprint arXiv:1610.09038*.
- Luong, M. T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.
- Rokach, L., & Maimon, O. Z. (2014). *Data mining with decision trees: theory and applications* (Vol. 81). World scientific.
- Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*.
- Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., ... & Bengio, Y. (2015, June). Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning* (pp. 2048-2057). PMLR.