In the beam search, you may get the end of sequence token </s> while decoding. That completes one generation. You move this generation into a final list for reranking and you divide its final score by its length (length normalization).

This does describe how to do attention with top k and all

<https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html#evaluation>

Attention Lecture. Starts around time 30:00

<https://bluejeans.com/playback/s/P2Vdy2Jopi0tASZisha6W4pMnwruiFRj5DJhSwNmhC3ZoK5J7lGP6ZHAQ5AAzxf6>

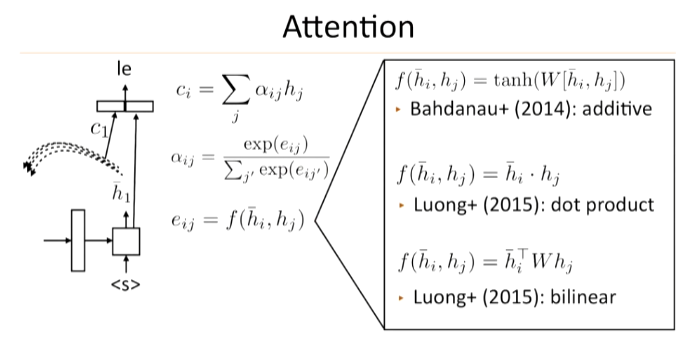
NOTE This is saying that tensor contraction or what you want is different than just straight matrix mult as it only has to work for 1 dim and you do it using einstein notation

<https://math.stackexchange.com/questions/63074/is-there-a-3-dimensional-matrix-by-matrix-product>

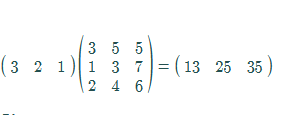
Should help with sizing

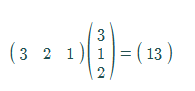
<https://pytorchnlp.readthedocs.io/en/latest/_modules/torchnlp/nn/attention.html>

<https://stackoverflow.com/questions/36030963/dot-product-along-third-axis>

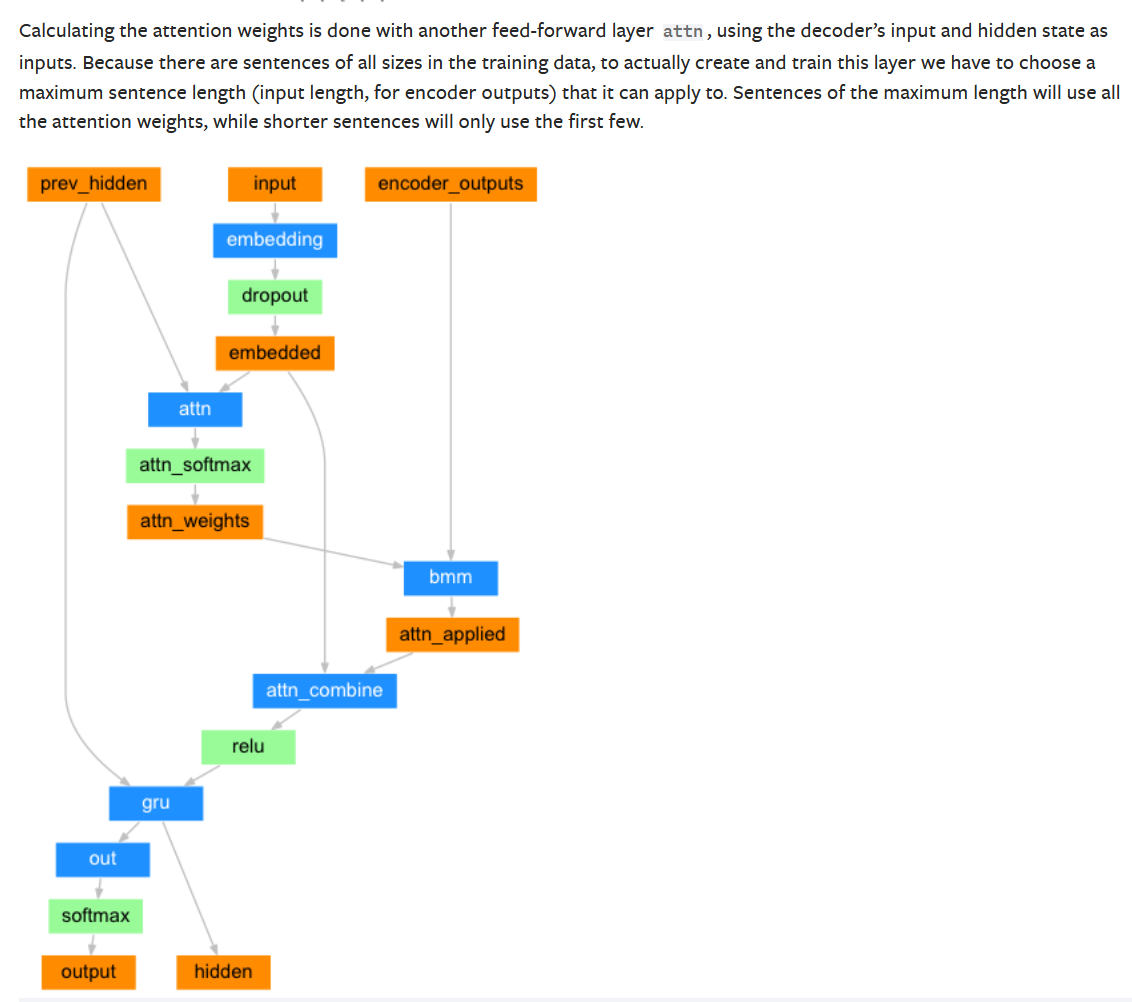


As long as your hidden j’s are as columns in a matrix H you can do that as h\_i + H





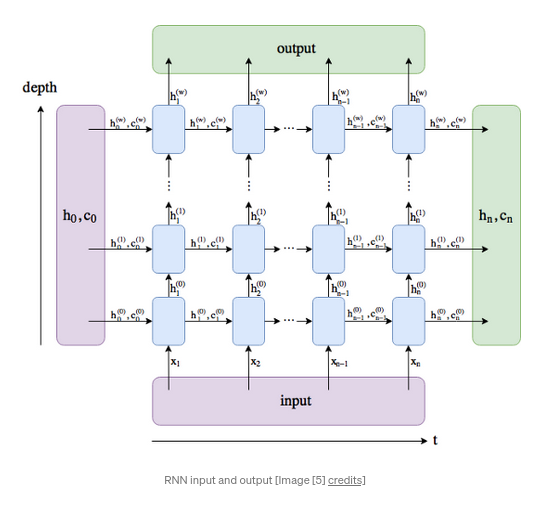
Confused by this diagram as it says that prev hidden and input are the only two things that go into attention



Look very carefully at this. OUT is JUST THE TOP LAYER OF THE RNN but for all input positions, HIDDEN is JUST THE LAST INPUT POSITION HIDDEN but for ALL LAYERS

out is the output of the RNN from all timesteps from the last RNN layer.

h\_n is the hidden value from the last time-step of all RNN layers.



file:///C:/Users/Collin/Downloads/lec11-seqseq2(3).pdf

Use torch top k for beam search

Torch.topk

Always pass logits through linear layer going to vocab size then sofmax for choice and argmax

OTHER SPECIAL TOKENS

pad\_word = "<pad>"

bos\_word = "<s>"

eos\_word = "</s>"

unk\_word = "<unk>"

pad\_id = 0

bos\_id = 1

eos\_id = 2

unk\_id = 3

Word id 1 is the start symbol

Word id 2 is the end symbol

Word id 5 is the period

Word id 7 is the question mark

Sentences dont have end and start symbol to start.

where ?

This does that

word\_tokens = vocab.tokenized\_sentence(sentence)

['<s>', 'where', '?', '</s>']

word\_ids = vocab.get\_ids\_from\_sentence(sentence)

NOTE this also adds the start and end tokens

[1, 6, 7, 2]

vocab.decode\_sentence\_from\_ids(word\_ids)

where ?

def collate\_fn(data):

"""Creates mini-batch tensors from the list of tuples (src\_seq, trg\_seq).

We should build a custom collate\_fn rather than using default collate\_fn,

because merging sequences (including padding) is not supported in default.

Seqeuences are padded to the maximum length of mini-batch sequences (dynamic padding).

<https://pytorch.org/docs/stable/generated/torch.nn.utils.rnn.pad_sequence.html>

For the seq2seq baseline are we supposed to implement the boolean masking for our loss? It said in the encode method that we didn't have to do this but I'm not sure how else to reduce my loss.

If I were to do this, how would I mask out the loss with this boolean mask? The crossentropyloss method doesn't have a parameter for a boolean mask. Should we implement the loss ourselves?

I used the ignore\_index parameter of the NLLLoss function so I did not need any mask

# You need to setup a target mask here for the target sequence so that we only add loss coressponing to the indicidies that are NOT padding

# NOTE THE TARGET SEQUENCE AND INPUT SEQUENCE HAVE ENTIRELY DIFFERENT MASK/PADDINGS/LENGTHS (question then response)

How to pack?

Notive the input to the call torch.nn.utils.rnn.pack\_padded\_sequence has a lengths argument that gives the true unpadded lengths of the original sequences

a = [torch.tensor([1,2,3]), torch.tensor([3,4])]

b = torch.nn.utils.rnn.pad\_sequence(a, batch\_first=True)

>>>>

tensor([[ 1, 2, 3],

[ 3, 4, 0]])

torch.nn.utils.rnn.pack\_padded\_sequence(b, batch\_first=True, lengths=[3,2])

>>>>PackedSequence(data=tensor([ 1, 3, 2, 4, 3]), batch\_sizes=tensor([ 2, 2, 1]))

Why do we pack in addition to padding?

So basically padding is how we let the pytorch models know what data is padding vs actual data and it can use this to internally optimize the model and avoid computing on padding which is just waste

<https://stackoverflow.com/questions/51030782/why-do-we-pack-the-sequences-in-pytorch>

When training RNN (LSTM or GRU or vanilla-RNN), it is difficult to batch the variable length sequences. For example: if the length of sequences in a size 8 batch is [4,6,8,5,4,3,7,8], you will pad all the sequences and that will result in 8 sequences of length 8. You would end up doing 64 computations (8x8), but you needed to do only 45 computations. Moreover, if you wanted to do something fancy like using a bidirectional-RNN, it would be harder to do batch computations just by padding and you might end up doing more computations than required.

Instead, PyTorch allows us to pack the sequence, internally packed sequence is a tuple of two lists. One contains the elements of sequences. Elements are interleaved by time steps (see example below) and other contains the size of each sequence the batch size at each step. This is helpful in recovering the actual sequences as well as telling RNN what is the batch size at each time step. This has been pointed by @Aerin. This can be passed to RNN and it will internally optimize the computations.

[torch.nn.utils.rnn.pad\_packed\_sequence — PyTorch 1.8.1 documentation](https://pytorch.org/docs/stable/generated/torch.nn.utils.rnn.pad_packed_sequence.html)

[How can I print the shape of a tensor inside the forward function? - PyTorch Forums](https://discuss.pytorch.org/t/how-can-i-print-the-shape-of-a-tensor-inside-the-forward-function/1143/5)

[GRU — PyTorch 1.8.1 documentation](https://pytorch.org/docs/stable/generated/torch.nn.GRU.html)

[Seq2Seq with Pytorch. Welcome! This is a continuation of our… | by Adam Wearne | Medium](https://medium.com/@adam.wearne/seq2seq-with-pytorch-46dc00ff5164)

[Clarification regarding the return of nn.GRU - nlp - PyTorch Forums](https://discuss.pytorch.org/t/clarification-regarding-the-return-of-nn-gru/47363)

This clearly shows how the input should be shaped and what is expected. If you in fact wanted to one-hot encode your data, you would need to use torch.nn.functional.one\_hot. To best replicate what the cross entropy loss is doing under the hood, you'd also need nn.functional.log\_softmax as the final output and you'd have to additionally write your own loss layer since none of the PyTorch layers use log softmax inputs and one-hot encoded targets. However, nn.CrossEntropyLoss combines both of these operations together and is preferred if your outputs are simply class labels so there is no need to do the conversion.

<https://stackoverflow.com/questions/62456558/is-one-hot-encoding-required-for-using-pytorchs-cross-entropy-loss-function>