1. **2 advantages and 2 disadvantages of NMT(Neural Machine Translation) and Linguistic Based MT(LBMT)**

Statistical MT

- uses predictive algorithms to teach a computer how to translate text. These models are created, or learned, from parallel bilingual text corpora and used to create the most probable output, based on different bilingual examples.
- Using this already translated text, a statistical model guesses or predicts how to translate foreign language text. Statistical MT has different subgroups, including word-based, phrase-based, syntax-based and hierarchical phrase-based.
- *The benefit of Statistical MT is its automation. One drawback is that this system needs bilingual material to work from, and it can be hard to find content for obscure languages.* Statistical MT is a “rule-based” MT method, using the basis of corpora translations to create its own text segments.

Neural Networks

- NMT is the newest method of MT and said to create *much more accurate translations than Statistical MT.*
- NMT is based on the model of neural networks in the human brain, with information being sent to different “layers” to be processed before output.
- NMT uses deep learning techniques to teach itself to translate text based on existing statistical models. It *makes for faster translations than the statistical method and has the ability to create higher quality output.*
- NMT is able to use algorithms to learn linguistic rules on its own from statistical models. *The biggest benefit to NMT is its speed and quality.*
- NMT is said by many to be the way of the future, and the process will no doubt continue to advance in its capabilities.

<u>*NMT disadvantages:*</u>
First, NMT can only translate on a sentence by sentence basis. Second, NMT needs a lot of training data to become fluent.
A serious concern is also that neural systems are very hard to debug. While in phrase-based statistical machine translation systems there is still some hope to trace back why the system came up with a specific translation (and remediating that problem), there is little hope for that in neural systems. The system may just produce output words that look good in context but have little to do with the source sentence.

<u>*NMT Limitations:*</u>
Sparsity – Solved
World Similarity – Solved
Finite Context – Not
Computational Complexity - Softmax

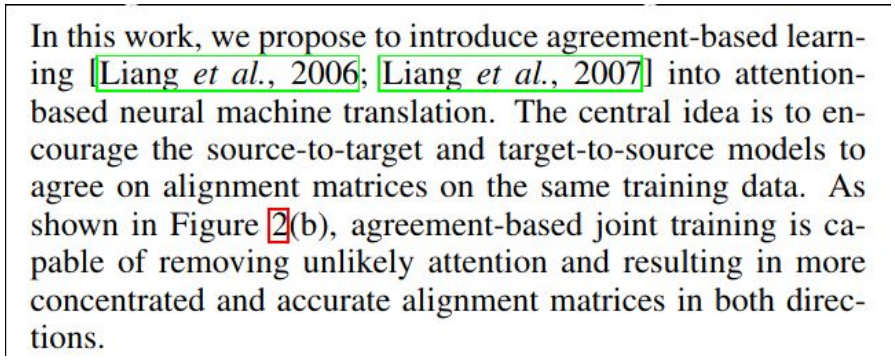
2. **Critical comparison between NMT and LBMT approaches.**

3. **In Attention based translation methods, we see a tendency to repeat words or phrases in the output. Why does this happen? How would you fix it?**

   Happens because each repeated sequence of words has a higher probability than the sequence without the next repetition.

   Can be solved by

   1. **Agreement**: Get a forward and backward LM to agree; symmetrization applied to LSTMs

      In this work, we propose to introduce agreement-based learning [Liang *et al.*, 2006; Liang *et al.*, 2007] into attention-based neural machine translation. The central idea is to encourage the source-to-target and target-to-source models to agree on alignment matrices on the same training data. As shown in Figure 2(b), agreement-based joint training is capable of removing unlikely attention and resulting in more concentrated and accurate alignment matrices in both directions.

   2. **Coverage**: Keep a variable C that tracks the words that have been translated from the source.

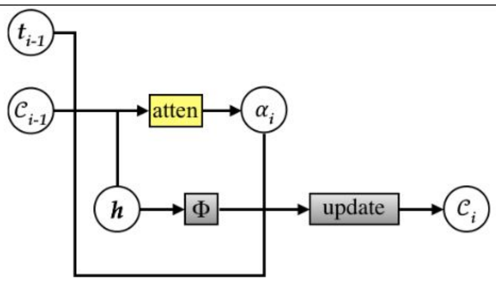
      

      Figure 3: Architecture of coverage-based attention model. A coverage vector $\mathcal{C}_{i-1}$ is maintained to keep track of which source words have been translated before time $i$. Alignment decisions $\alpha_i$ are made jointly taking into account past alignment information embedded in $\mathcal{C}_{i-1}$, which lets the attention model to consider more about untranslated source words.

4. **Explain luong attention mechanisms. Why is it better than Bahdanau Attention?**
   https://towardsdatascience.com/sequence-2-sequence-model-with-attention-mechanism-9e9ca2a613a

Luong attention takes into account of its attention calculation the previously translated word (from the output decoder): that is, $s_{t-1.}$ It is better than Bahdanau because it uses this additional context to better calculate attention weights to give better results.

5. **What mechanism would you add to the Attention based model to address out of vocabulary words in the source? Explain your choices.**
   - Smoothing in LM
   - Use fastText

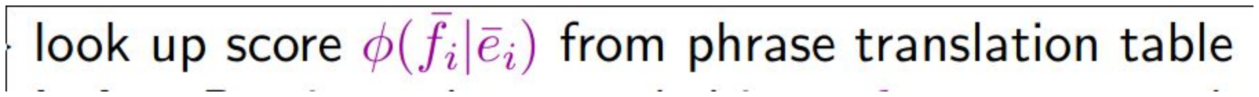
6. **Describe the various word based MT models? What are the failures of word based MT.**

   Word based models may not be the best candidate for the smallest unit of translation as they will often break down in frequent one-to-many mappings ( and vice versa). Translating a group of words instead of a single word helps in resolving ambiguities. Eg. bank has different meanings. River bank, money bank etc
   If we have large parallel corpora then we can learn more variation and more useful phrase translations.

7. **What is alignment? How is the phrase table (along with phrase translation probability) created for Phrase based MT?**

   For MT, we do two basic things. Translation of words: what do the words map to; could be to more than one word in either direction and movement of translated words to their correct positions in the target sentence; called alignment.

   Phrase table is a strength of PBSMT and stores entries like these:
   europas ||| in europe ||| 0.0251019 0.066211 0.0342506 0.0079563
   To get such table we first find word alignment between each sentence pair of the parallel corpus. And then extract phrase pairs that are consistent with this word alignment.
   Definition of consistency: We call a phrase pair (f,e) consistent with an alignment A, if all words in f that have alignment points in A have these with words in e and vice versa.
   Now since we have extracted phrase pairs we can count how many times particular phrase was aligned with some particular phrase.



look up score $\phi(\bar{f}_i|\bar{e}_i)$ from phrase translation table

8. What are the advantages of LSTMs over RNNs? Clearly explain the architecture of LSTMs and GRUs.

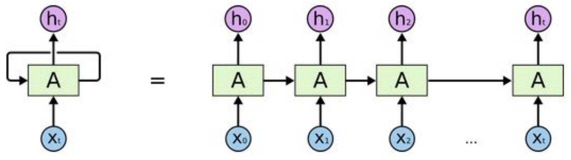
Ans). All RNNs have feedback loops in the recurrent layer. This lets them maintain information in 'memory' over time. But, it can be difficult to train standard RNNs to solve problems that require learning long-term temporal dependencies. This is because the gradient of the loss function decays exponentially with time (called the vanishing gradient problem). LSTM networks are a type of RNN that uses special units in addition to standard units. LSTM units include a 'memory cell' that can maintain information in memory for long periods of time. A set of gates is used to control when information enters the memory, when it's output, and when it's forgotten. This architecture lets them learn longer-term dependencies. GRUs are similar to LSTMs, but use a simplified structure. They also use a set of gates to control the flow of information, but they don't use separate memory cells, and they use fewer gates.
**LSTM** is well-suited to classify, process and predict time series given time lags of unknown duration. Relative insensitivity to gap length gives an **advantage** to **LSTM** over alternative RNNs
Advantages of Recurrent Neural Network

- The main advantage of RNN is that RNN can model sequence of data (i.e. time series) so that each sample can be assumed to be dependent on previous ones
- Recurrent neural network are even used with convolutional layers to extend the effective pixel neighborhood.

Disadvantages of Recurrent Neural Network

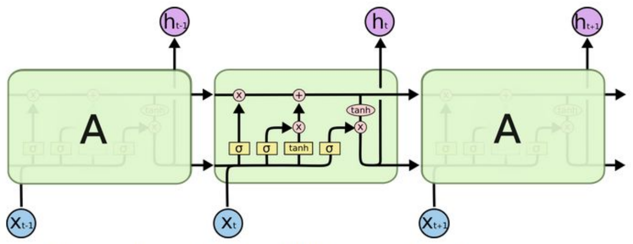
- Gradient vanishing and exploding problems.
- Training an RNN is a very difficult task.
- It cannot process very long sequences if using tanh or relu as an activation function.

RNN:

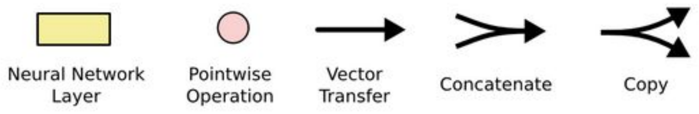


An unrolled recurrent neural network.

LSTM:



The repeating module in an LSTM contains four interacting layers.



In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at $h_{t-1}$ and $x_t$, and outputs a number between $0$ and $1$ for each number in the cell state $C_{t-1}$. A $1$ represents "completely keep this" while a $0$ represents "completely get rid of this."

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state. In the next step, we'll combine these two to create an update to the state.

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

It's now time to update the old cell state, $C_{t-1}$, into the new cell state $C_t$. The previous steps already decided what to do, we just need to actually do it. We multiply the old state by $f_t$, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

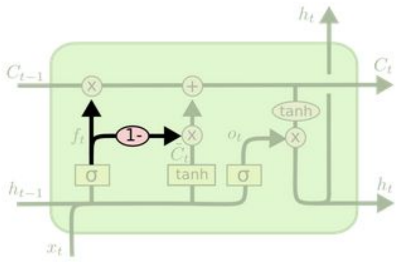
$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Variants:

One popular LSTM variant, introduced by Gers & Schmidhuber (2000), is adding "peephole connections." This means that we let the gate layers look at the cell state.

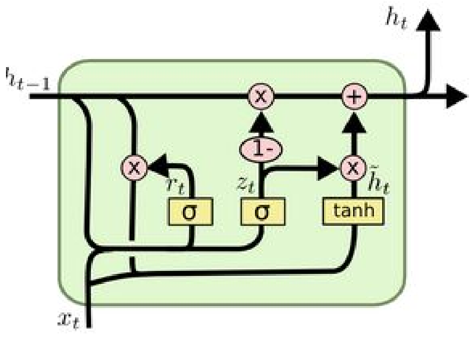
$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \ + \ b_o\right)$$

Another variation is to use coupled forget and input gates. Instead of separately deciding what to forget and what we should add new information to, we make those decisions together. We only forget when we're going to input something in its place. We only input new values to the state when we forget something older.



$$C_t = f_t * C_{t-1} + (\mathbf{1} - \boldsymbol{f_t}) * \tilde{C}_t$$

**Gated Recurrent Unit, or GRU.** It combines the forget and input gates into a single "update gate." It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

9. **Most Indo Aryan and Dravidian Languages exhibit rich morphology. This results in a large number of word forms for words belonging to certain categories (noun, verbs). Typically vector embedding methods take words as input. Can you design a vector embedding mechanism which can learn embeddings such that different morphological variations (eg. walk, walking, walks, walked) of the same root (walk) are closer in the vector space?**

    a. Give the weighted sum of the stem and the wordform
    b. The weighted sum of its morphemes, with the root morpheme, has the greatest weight
    c. Map the stem in the space and have vectors for all the possible morphemes. The embedding of the word = stem + morph1 + morph2 + ...

10. **Sequence to sequence models have been employed for machine translation with some success. But Attention-based models have overtaken these models. What value do Attention models bring in order to outperform bidirectional LSTMs based seq-seq models.**

    Seq2Seq has the downside of representing an entire sentence using a single fixed-length vector. So, when they try to translate long sentences, most of the sentence meaning ends up coming from the last few words, resulting in inaccurate translations. Even when bi-LSTMs are used, (you know how to elaborate by now, fill in bullshit).

In Attention-based models, we use the output of each iteration of the input encoder LSTMs in order to predict the subsequent decoder output. As such, each word of the encoder output takes into consideration the attention it should pay to each word of the input, by assigning weights (affinities) to their encoder outputs. …. Mention Luong and Bahdanau … talk about performance drop-off.

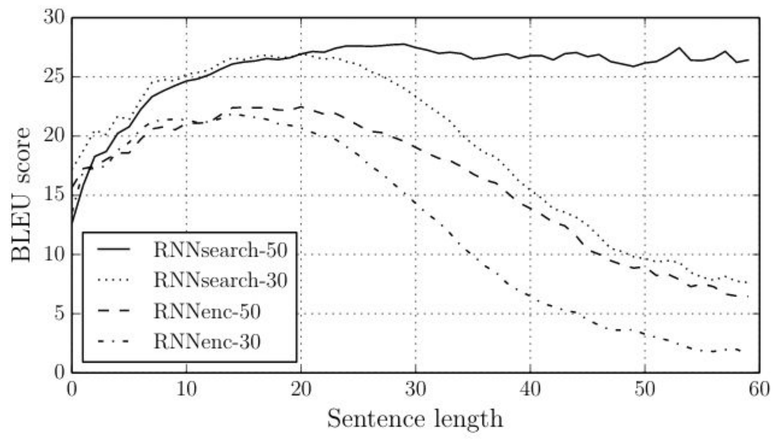


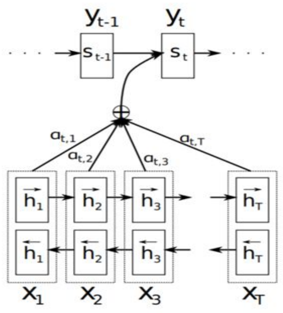
Figure 2: The BLEU scores of the generated translations on the test set with respect to the lengths of the sentences. The results are on the full test set which includes sentences having unknown words to the models.

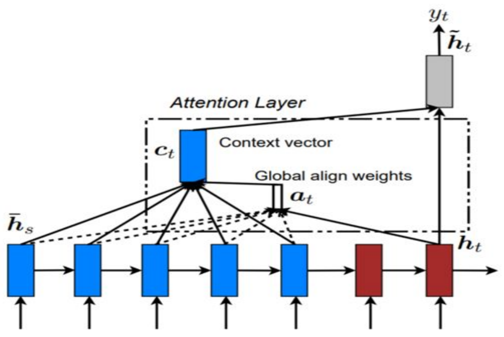
11. **Will Attention based models as applied to English MT work as well for Dravidian Languages as they do for english given all other aspects (training data etc) remain the same? What changes are required if any?**

    a. Will not work the same, as agglutinativeness does not allow for straightforward sequencing. Might have to character-level seq2seq
    b. Seq2seq on the stems and use morph analyser to for grammatically correct sentences
    c. Need Morph analyser anyway cuz it's cool.

Answers

4. Key differences between Bahdanau and Luong attention mechanism:


$y_{t-1}$
$y_t$
$s_{t-1}$
$s_t$
$a_{t,1}$
$a_{t,2}$
$a_{t,3}$
$a_{t,T}$
$\overrightarrow{h}_1$ $\overrightarrow{h}_2$ $\overrightarrow{h}_3$ $\overrightarrow{h}_T$
$\overleftarrow{h}_1$ $\overleftarrow{h}_2$ $\overleftarrow{h}_3$ $\overleftarrow{h}_T$
$x_1$ $x_2$ $x_3$ $x_T$
Bahdanau attention mechanism



Luong attention mechanism

Bahdanau concatenation of the forward and backward hidden states in the bi-directional encoder. Luong attention uses hidden state at the top layer in both encoder and decoder
Computation of attention in Bahdanau and Luong attention mechanisms: Bahdanau et al. uses the concatenation of the forward and backward hidden states in the bi-directional encoder and previous target's hidden states in their non-stacking unidirectional decoder. Loung et al. attention uses hidden states at the top LSTM layers in both the encoder and decoder
Luong attention mechanism uses the current decoder's hidden state to compute the alignment vector, whereas Bahdanau uses the output of the previous time step

Alignment functions: Bahdanau uses only concat score alignment model whereas Luong uses dot, general and concat alignment score models

$$\operatorname{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W_a} \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_a}[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$

**Luong's attention** is also referred to as Multiplicative attention. It reduces encoder states and decoder state into attention scores by simple matrix multiplications. Simple matrix multiplication makes it is faster and more space-efficient. Luong suggested two types of attention mechanism based on where the attention is placed in the source sequence:

1. Global attention where attention is placed on all source positions
2. Local attention where attention is placed only on a small subset of the source positions per target word

The commonality between Global and Local attention

- At each time step t, in the decoding phase, both approaches, global and local attention, first take the hidden state $h_t$ at the top layer of a stacking LSTM as an input.
- The goal of both approaches is to derive a context vector $c_t$ to capture relevant source-side information to help predict the current target word $y_t$

- Attentional vectors are fed as inputs to the next time steps to inform the model about past alignment decisions.

Global and local attention models differ in how the context vector $c_t$ is derived
Before we discuss the global and local attention, let’s understand the conventions used by Luong’s attention mechanism for any given time t

- $c_t$ : context vector, $a_t$ : alignment vector, $h_t$ : current target hidden state, $h_s$ : current source hidden state, $y_t$: predicted current target word, $\tilde{h}_t$ : Attentional vectors