Disclaimer: This presentation is a work of fiction written from the perspective of a 2020 researcher traveling back in time to mid 2013 to share some 2020 xNN based application ideas; references to credit the actual inventors of the various ideas is provided at the end

Sequence-Sequence Mechanisms for Neural Machine Translation

Jaisurya Nanduri
jxn180029@utdallas.edu
May 1, 2013

1 - Motivations

What is the task we want to solve?

1.1 Machine Translation

Machine Translation

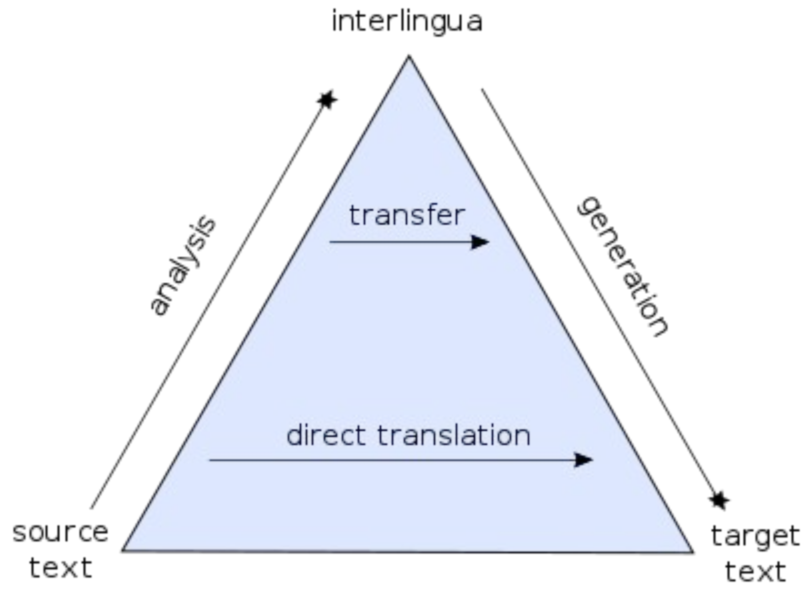
1. One of the earliest goals of computing has been to translate one language to another.

2. Classical approaches have been use of statistical knowledge about lexicons in the languages.

3. Usage of Neural Nets for Machine Translation has been a recent exploration.

Approaches and Work in MT

1. Rule Based
2. Statistical Based
3. Example Based
4. Hybrid MT
5. Neural MT



Neural MT

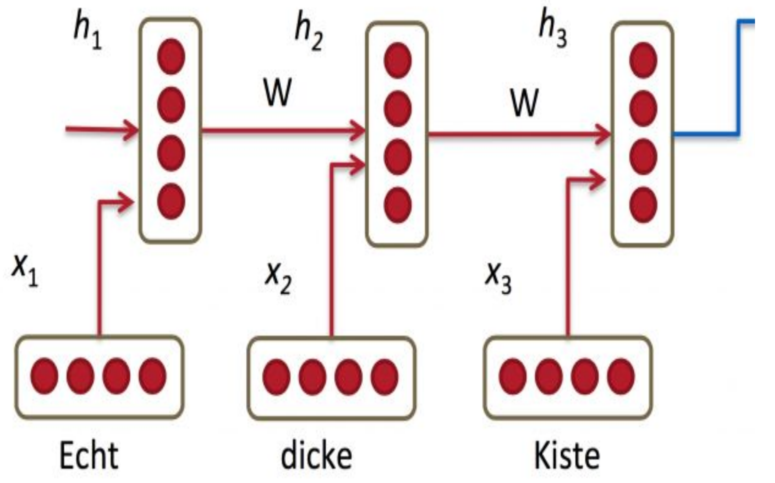
1. Approaching Machine Translation by usage of Neural Nets.
2. Sub-field has been in the beginning stages of development.
3. Key breakthrough occurred in 2012 with Schwenk et al., showing successful usage of a simple feedforward neural network in the framework of phrase based MT.

1.2 Existing Research - RNN

1. Simple feedforward networks too primitive for usage for a complex task such as MT.
2. Why?
   a. Traditional feed-forward neural networks take in a fixed amount of input data all at the same time and produce a fixed amount of output each time
   b. For any meaningful interpretation, we need lengthier inputs and outputs causing exponential issues to regular feed forward networks.
3. This is why, we traditionally use Recurrent Neural Networks for sequential information

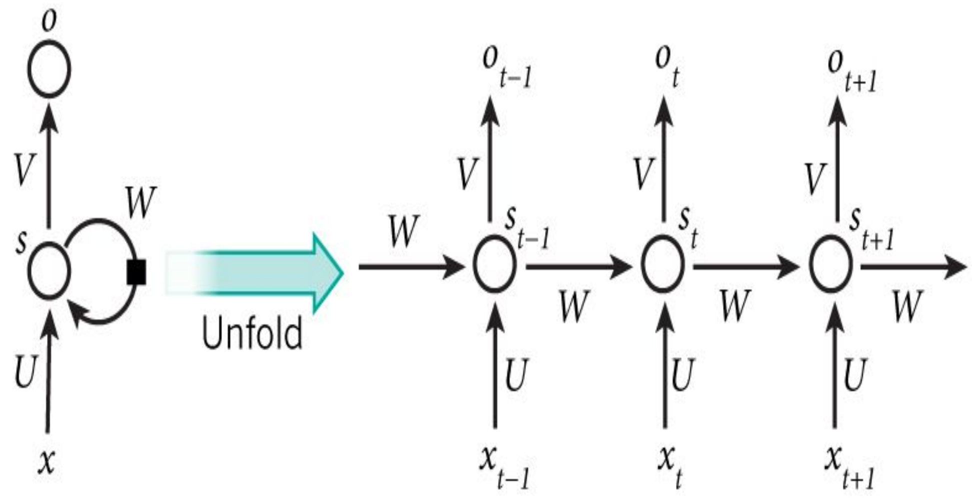
Recurrent Neural Networks (RNN)

1. RNNs do not consume all the input data at once. Instead, they take them in one at a time and in a sequence.
2. RNNs try to understand the data as a sequence in which the unit is dependent on its predecessors.
3. At each step, the RNN does a series of calculations before producing an output. The output, known as the hidden state, is then combined with the next input in the sequence to produce another output.



Recurrent Neural Networks (RNN)

1. Each RNN unit captures both input as well as some information passed on from a previous RNN unit (unlike other NNs)
2. This is key as it captures context, something necessary in sequential data such as text, speech and language but not present in image data.
3. This forms a chain of units (can be understood as a chain of information) where each unit influences its successors.



Power Behind RNNs

RNNs are very powerful, because they:

1. Distributed hidden state that allows them to store a lot of information about the past efficiently.

2. Nonlinear dynamics that allows them to update their hidden state in complicated ways.

3. No need to infer hidden state, pure deterministic.

Note that the weights are shared over time

Essentially, copies of the RNN cell are made over time (unrolling/unfolding), with different inputs at different time steps.

Drawbacks of Vanilla RNNs

1. Vanishing / Exploding gradients for lengthier inputs.
2. Overfitting - memorization
3. Handling on unseen characters
4. Representation of textual data

Drawbacks of Vanilla RNNs

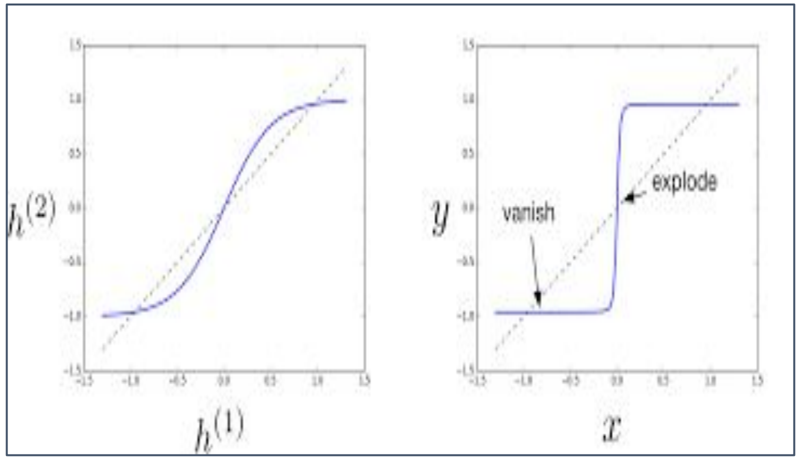
1. Issues such as Overfitting and Handling of Unseen data can be tackled with Hyperparameter tinkering.

2. These problems are not unique to any one architecture and are a recurring problem shared by all models.

3. RNNs however suffer uniquely from Exploding / Vanishing gradients due to deficiencies in Architecture.

Vanishing/Exploding Gradients

1. In deep networks or recurrent neural networks, error gradients can accumulate during an update and result in very large gradients.
2. These in turn result in large updates to the network weights, and in turn, an unstable network.
3. At an extreme, the values of weights can become so large as to overflow and result in NaN values or drop to 0.

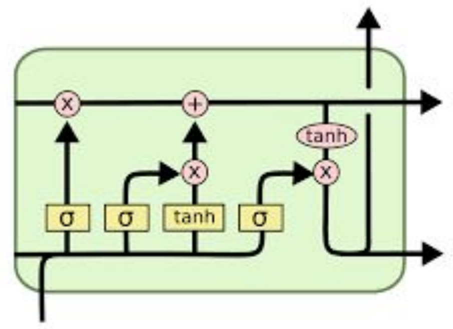


How to tackle vanishing/exploding gradients?

1. LSTM was proposed in 1997 by Sepp Hochreiter and Jürgen Schmidhuber.
2. By introducing Constant Error Carousel (CEC) units, LSTM deals with the vanishing gradient problem.
3. Use Gradient Clipping for Exploding gradient problem.

LSTM Networks

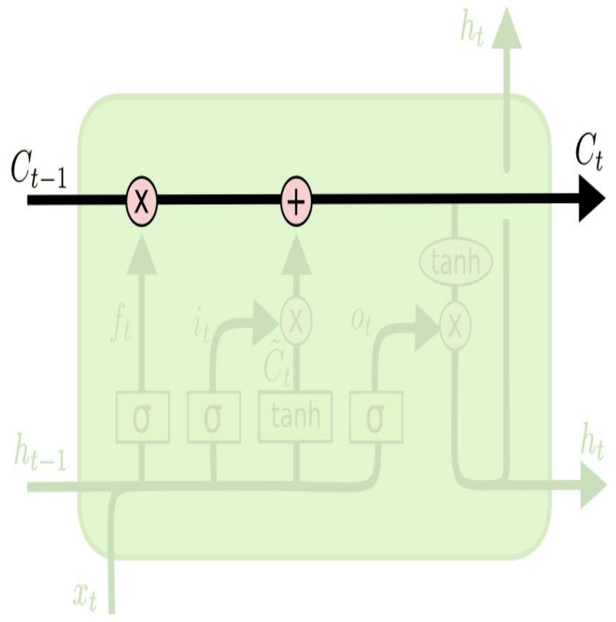
1. The key to the LSTM solution to the technical problems was the specific internal structure of the units used in the model.
2. Each memory cell's internal architecture guarantees constant error ow within its constant error carrousel (CEC).This represents the basis for bridging very long time lags.
3. Two gate units learn to open and close access to error ow within each memory cell's CEC. The multiplicative input gate affords protection of the CEC from perturbation by irrelevant inputs



LSTM Brief

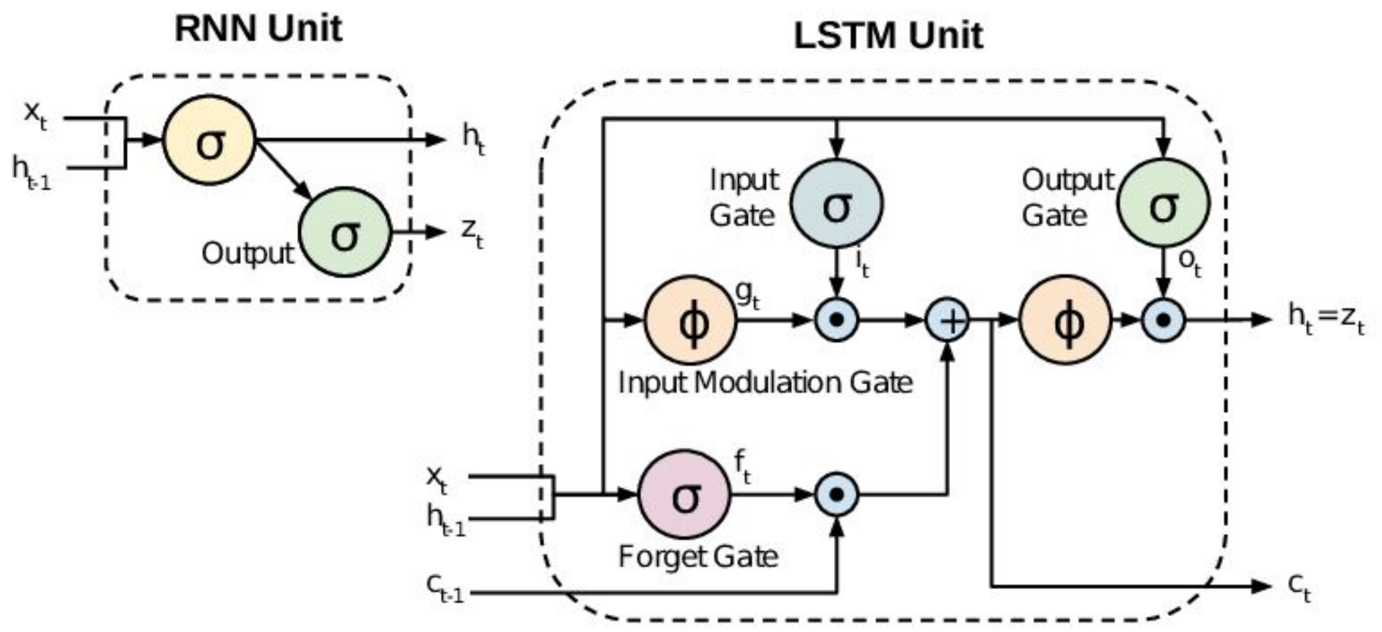
The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.



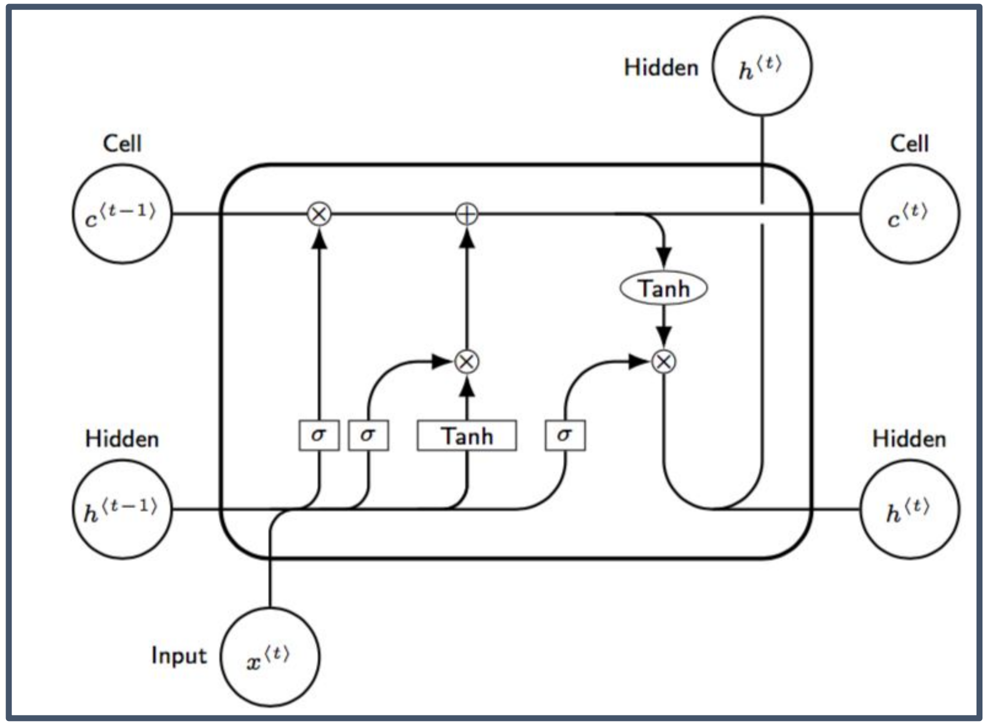
LSTM Gates

- The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.
- Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



Problems with LSTM

1. Not simple to Implement with 2013 architecture.
2. Does not address variable length inputs and outputs.
3. Does not scale well with lengthier inputs due to memory leak and lack of context.



1.3 Summary of Motivation

1. LSTM Networks show great progress in MT but leave room for improvement in key areas such as -
    a. Variable Length Input/Outputs.
    b. Not scalable due to complexity of structure.
    c. Does not take contextual semantic information into account during MT.
2. We propose an architecture which attempts to address these issues and improve the accuracy of Machine Translation.
3. Qualitatively, we show that the proposed model learns a semantically and syntactically meaningful representation of linguistic phrases.

2- Key Insights

2 - Key Insights

1. Sequences pose a challenge for DNNs because they require that the dimensionality of the inputs and outputs is known and fixed.
2. This transformation from input to output is called as a Sequence to Sequence class of problems.
3. Need to find a mapping from input to output which is flexible on length of both.
4. Could simplify LSTM gates.

Key Insights

1. There have been a number of related attempts to address the general sequence to sequence learning problem with neural networks. Our approach is closely related to Kalchbrenner and Blunsom who were the first to map the entire input sentence to vector, and is very similar to Cho et al.
2. Graves introduced a novel differentiable attention mechanism that allows neural networks to focus on different parts of their input, and an elegant variant of this idea was successfully applied to machine translation by Bahdanau et al.
3. The Connectionist Sequence Classification is another popular technique for mapping sequences to sequences with neural networks, although it assumes a monotonic alignment between the inputs and the outputs .

Key Insights

- Could use a generative approach to MT.
- Instead of translating directly from one lexicon into another., we could read the whole sentence, store the semantic meaning in a buffer middle storage and then generate the output sentence from this buffer.
- Using this buffer as an intermediary between the input and output language, the common semantic similarity that binds them together.

Key Insights - Summary

1. Allow for flexible length inputs and outputs
2. Solve Gradient Explosion and Vanishing by Updating the LSTM Unit. Preferably by simplifying it.
3. Incorporate contextual semantic information in translating phrases from one language to another.

3 - Proposed Approaches

3.1 Proposal Outline

Proposed Architectures -Outline

1. Encoder-Decoder Architecture.
2. Gated Recurrent Units to Solve E/V Gradients.
3. Alignment and Annotation mechanism.

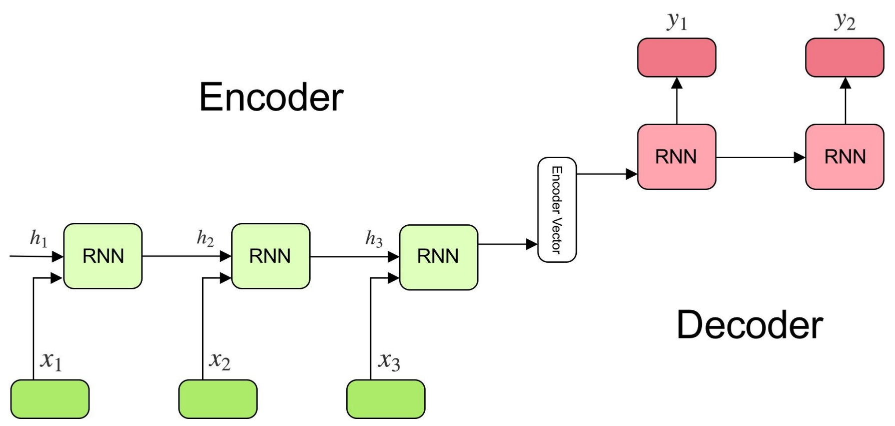
3.2 Encoder-Decoders

Encoder-Decoders

1. Despite their flexibility and power, LSTM-RNNs can only be applied to problems whose inputs and targets can be sensibly encoded with vectors of fixed dimensionality.
2. It is a significant limitation, since many important problems are best expressed with sequences whose lengths are not known a-priori. For example, speech recognition and machine translation are sequential problems.
3. Likewise, question answering can also be seen as mapping a sequence of words representing the question to a 1 sequence of words representing the answer. It is therefore clear that a domain-independent method that learns to map sequences to sequences would be useful.

Encoder-Decoders

1. In the general case, input sequences and output sequences have different lengths (e.g. machine translation) and the entire input sequence is required in order to start predicting the target.
2. This requires a more advanced setup, which is what people commonly refer to when mentioning "sequence to sequence models" with no further context.



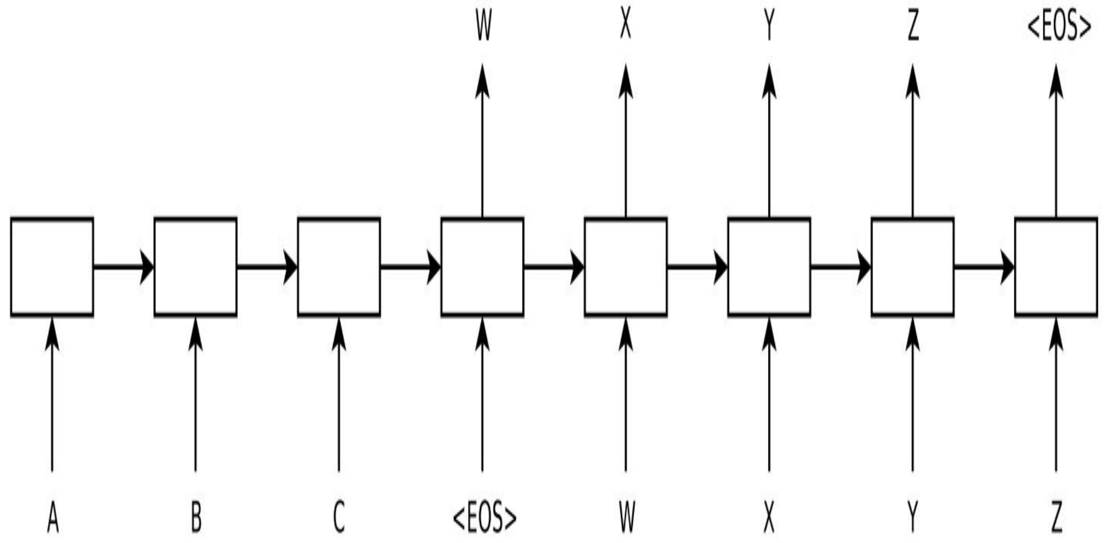
Encoder-Decoder

Here's how it works:

1. A RNN layer (or stack thereof) acts as "encoder": it processes the input sequence and returns its own internal state. Note that we discard the outputs of the encoder RNN, only recovering the state. This state will serve as the "context", or "conditioning", of the decoder in the next step.
2. Another RNN layer (or stack thereof) acts as "decoder": it is trained to predict the next characters of the target sequence, given previous characters of the target sequence. Specifically, it is trained to turn the target sequences into the same sequences but offset by one timestep in the future, a training process called "teacher forcing" in this context.

Encoder-Decoder

- Importantly, the encoder uses as initial state the state vectors from the encoder, which is how the decoder obtains information about what it is supposed to generate. Effectively, the decoder learns to generate targets[t+1...] given targets[...t], conditioned on the input sequence.



Encoder-Decoder

In inference mode, i.e. when we want to decode unknown input sequences, we go through a slightly different process:

1) Encode the input sequence into state vectors.

2) Start with a target sequence of size 1 (just the start-of-sequence character).

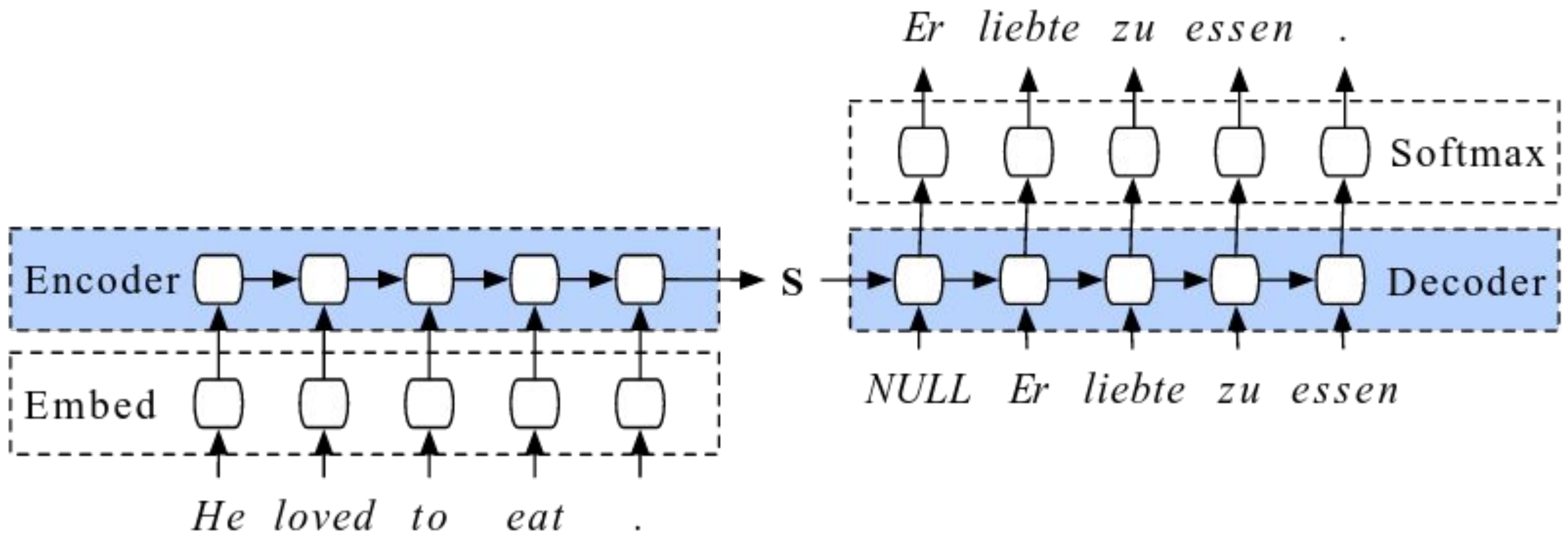
3) Feed the state vectors and 1-char target sequence to the decoder to produce predictions for the next character.

4) Sample the next character using these predictions (we simply use argmax).

5) Append the sampled character to the target sequence

6) Repeat until we generate the end-of-sequence character or we hit the character limit.

A High Level View of Encoder-Decoder



Encoder-Decoder

1. Encode a variable-length sequence into a fixed-length vector representation and to decode a given fixed-length vector representation back into a variable-length seque
2. After reading the end of the sequence (marked by an end-of-sequence symbol), the hidden state of the RNN is a summary c of the whole input sequence.
3. The two components of the proposed RNN Encoder–Decoder are jointly trained to maximize the conditional log-likelihood

   max θ 1 / N X Sigma(n=1..N){ log pθ(yn | xn)}

4. Once the RNN Encoder–Decoder is trained, the model can be used in two ways. One way is to use the model to generate a target sequence given an input sequence. On the other hand, the model can be used to score a given pair of input and output sequences, where the score is simply a probability pθ(y | x).

Encoder-Decoder

The hidden state of the decoder at time t is computed by,

$$h^{<t>} = f(h^{<t-1>}, y_{t-1}, c)$$

and similarly, the conditional distribution of the next symbol is

$$P(y_t \mid y_{t-1}, y_{t-2}, \ldots, y_1, c) = g(h^{<t>}, y^{<t-1>}, c) \; .$$

for given activation functions f and g (the latter must produce valid probabilities, e.g. with a soft

Encoder-Decoder Modularity

1. Because of the Nature of the Encoder-Decoder pair, it becomes trivial to replace each of them with a different Neural Network appropriate to the task.

2. For example, in Image Captioning, replace the Encoder RNN with an Encoder CNN.

3. Similarly, the Decoder is also replaceable according the task at hand.

Encoder-Decoder Summary

1. Encode a variable-length sequence into a fixed-length vector representation and to decode a given fixed-length vector representation back into a variable-length seque

2. After reading the end of the input sequence (marked by an end-of-sequence symbol), the hidden state of the input RNN is a summary c of the whole input sequence.

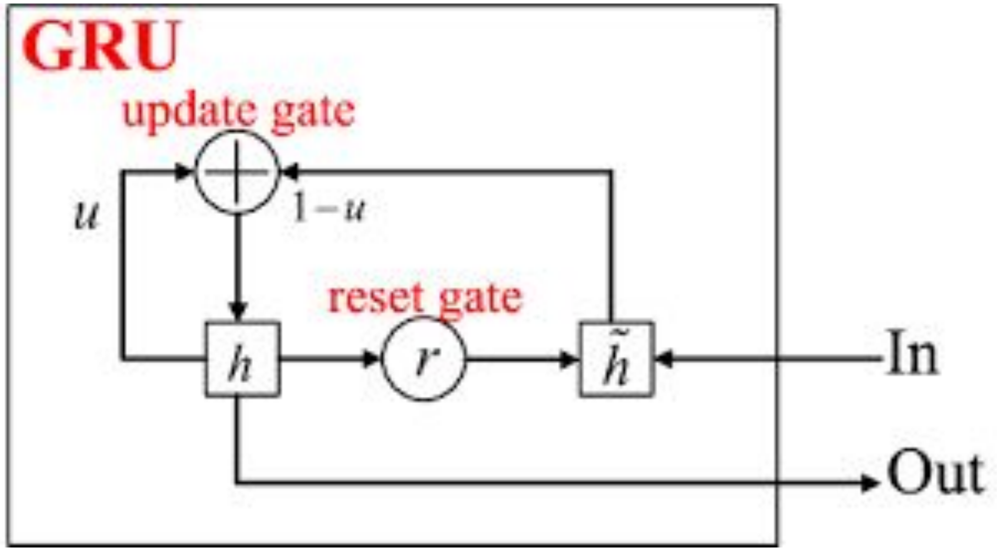
3. Hidden state is used to either generate or score a pair of input and output

3.3 Gated Recurrent Unit

GRU Motivation and Intuition

1. We introduce a new unit called GRU which is like the LSTM but LSTM unit but is much simpler to compute and implement.

2. The motivation is that simpler gates scale better enabling us to infer lengthier sequential information without encountering the exploding / vanishing gradient issue.

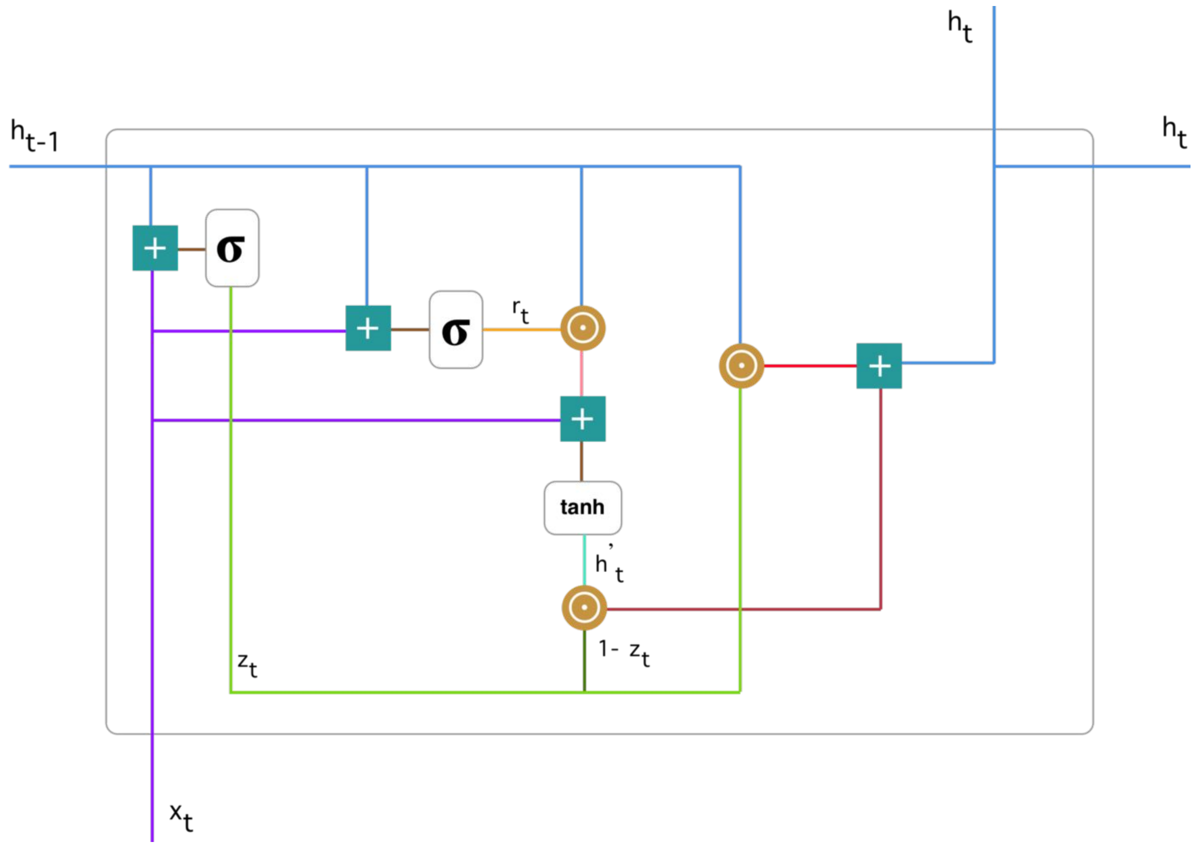


GRU Unit

1. GRU is related to LSTM as both are utilizing different way if gating information to prevent vanishing gradient problem. Here are some pin-points about GRU vs LSTM-
2. The GRU controls the flow of information like the LSTM unit, but without having to use a memory unit. It just exposes the full hidden content without any control.
3. GRU is ,according to our experiments, the performance is on par with LSTM, but computationally more efficient (less complex structure as pointed out).

GRU Unit

- To solve the vanishing gradient problem of a standard RNN, GRU uses, so-called, **update gate and reset gate**. Basically, these are two vectors which decide what information should be passed to the output.



GRU Gates - Update Gate

- We start with calculating the **update gate z_t for time step t** using the formula:

$$z_t = \sigma(W^{(z)} x_t + U^{(z)} h_{t-1})$$

- The update gate helps the model to determine how much of the past information (from previous time steps) needs to be passed along to the future.

- That is really powerful because the model can decide to copy all the information from the past and eliminate the risk of vanishing gradient problem.
- We plug in h_(t-1) and x_t ,multiply them with their corresponding weights, sum the results and apply the sigmoid function.

GRU Gates - Reset Gate

- This gate is used from the model to decide how much of the past information to forget.

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

- This formula is the same as the one for the update gate. The difference comes in the weights and the gate’s usage.

- As before, we plug in h_(t-1) — blue line and x_t — purple line, multiply them with their corresponding weights, sum the results and apply the sigmoid function.

GRU Unit - Current memory content

- We introduce a new memory content which will use the reset gate to store the relevant information from the past.

$$h_t' = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

- We do an element-wise multiplication of h_(t-1) and r_t and then sum the result with the input x_t. Finally, tanh is used to produce h’_t.

GRU Unit - Calculate current memory content

- Multiply the input x_t with a weight W and h_(t-1) with a weight U.
- Calculate the Hadamard (element-wise) product between the reset gate r_t and Uh_(t-1). That will determine what to remove from the previous time steps.
- Let’s say we have a sentiment analysis problem for determining one’s opinion about a book from a review he wrote.
  - The text starts with “This is a fantasy book which illustrates…” and after a couple paragraphs ends with “I didn’t quite enjoy the book because I think it captures too many details.”
  - To determine the overall level of satisfaction from the book we only need the last part of the review.
  - In that case as the neural network approaches to the end of the text it will learn to assign r_t vector close to 0, washing out the past and focusing only on the last sentences.
- Sum up the results of step 1 and 2.
- Apply the nonlinear activation function tanh.

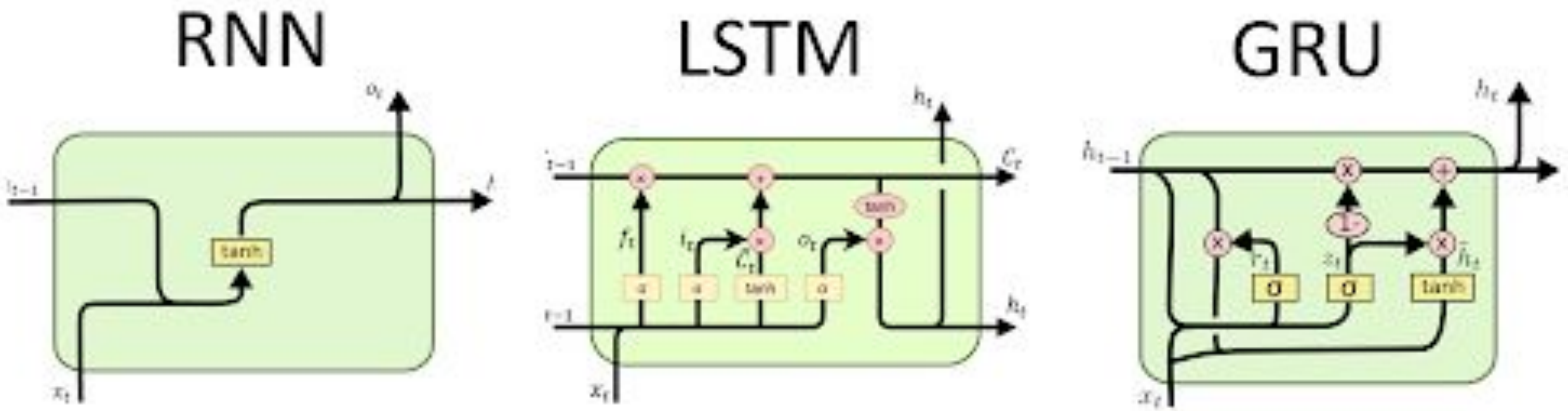
GRU Unit - Final memory at current time step

- As the last step, the network needs to calculate *h_t* — vector which holds information for the current unit and passes it down to the network. In order to do that the update gate is needed.
- It determines what to collect from the current memory content — *h'_t* and what from the previous steps — *h (t-1)*. That is done as follows:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h_t'$$

- Apply element-wise multiplication to the update gate z_t and h_(t-1).
- Apply element-wise multiplication to (1-z_t) and h'_t.
- Sum the results from step 1 and 2.

GRU vs LSTM


RNN
LSTM
GRU
tanh
σ
×
+
1-
x_t
h_t
h_{t-1}
C_t
C_{t-1}
f_t
i_t
$\tilde{C}_t$
o_t
r_t
z_t
$\tilde{h}_t$
o_t

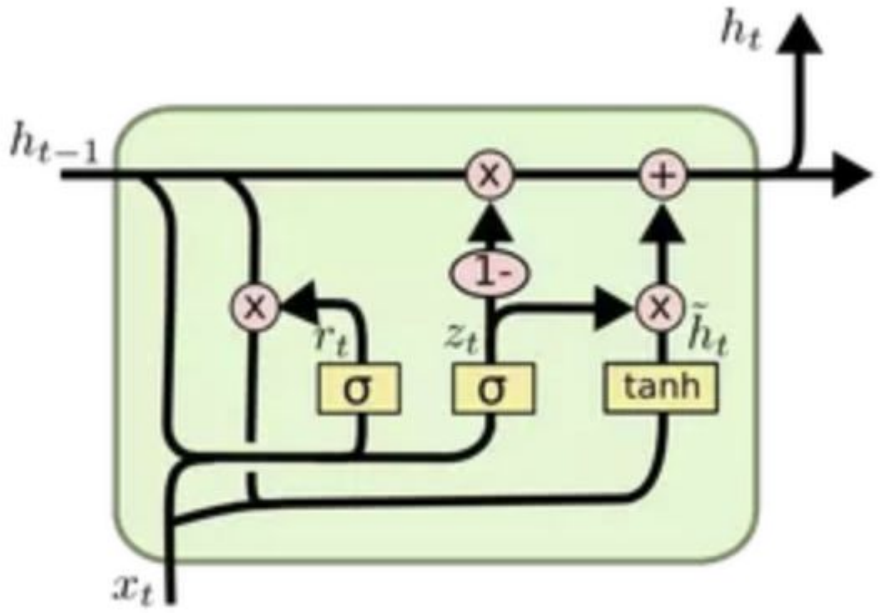
GRU vs LSTM

- LSTMs should in theory remember longer sequences than GRUs and outperform them in tasks requiring modeling long-distance relations. But in practice, we found them quite level.
- GRU exposes the complete memory unlike LSTM, so applications which that acts as advantage might be helpful. Also, adding onto why to use GRU - it is computationally easier than LSTM since it has only 2 gates and if it's performance is on par with LSTM, then it becomes a more feasible option.
- This research demonstrates excellently with graphs the superiority of gated networks over a simple RNN but cannot conclude which of the either are better.

GRU vs LSTM - Architecture Differences

- A GRU has two gates, an LSTM has three gates.
- GRUs don’t possess and internal memory (c_t) that is different from the exposed hidden state. They don’t have the output gate that is present in LSTMs.
- The input and forget gates are coupled by an update gate z and the reset gate r is applied directly to the previous hidden state. Thus, the responsibility of the reset gate in a LSTM is really split up into both r and z.
- We don’t apply a second nonlinearity when computing the output.

GRU Computation



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

GRU Summary

- GRUs are able to store and filter the information using their update and reset gates.
- This unit eliminates the vanishing gradient problem since the model is not washing out the new input every single time but keeps the relevant information and passes it down to the next time steps of the network.
- If carefully trained, they can perform extremely well even in complex scenarios.

3.4 - Alignment

Alignment Modelling

- Basic Encoder-Decoder , as shown by our experiments while allow us to build flexible input/output models in the first place, do not generate great results.
- They are limited in their ability to track long-term dependencies and even lose their ability to translate the end of long sentences correctly.
- The cause of this limitation is that this “basic encoder-decoder” architecture encodes everything about the input sentence in a single fixed-length vector (the encoder RNN’s final hidden state).

Alignment Modelling

- This paper addresses the single node bottleneck problem in two ways: first by using a bidirectional LSTM for input (this is not a new idea — the citation is from Schuster and Paliwal 1997), and second by introducing an alignment model, a matrix of weights connecting each input location to each output location.
- This can be thought of as an attention mechanism that allows the decoder to pull information from useful parts of the input rather than having to decode a single hidden state

Alignment Modelling

- The model, in a bit more detail: the biLSTM (actually two LSTMs, one running forward over the input sequence, the other running backward) outputs two hidden states for each input location j.

- The forward hidden state contains information from earlier in the sequence and the backward from later in the sequence; still, these states are primarily about the word at j because RNNs forget easily.

- The states are concatenated, forming an annotation. To sum up, the annotation hj is a representation of the word at j, with some context.

Alignment Modelling - Computation

- We start by computing the alignment given by figure below where *si*-1 is the previous decoder hidden state, *hj* is the annotation, *a* is the feedforward alignment model, and *eij* is interpreted as an energy (roughly, the importance of the annotation for generating the current output)

$$e_{ij} = a(s_{i-1}, h_j)$$

- Then we have the softmax and weighted sum, resulting in the context vector *ci*:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \qquad c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

Alignment Modelling - Computation

- Then we have the hidden state and a probability distribution over the output

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

$$p(y_i | y_1, \ldots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i)$$

Alignment Modelling - Summary

- Encoder-Decoder Models are limited in their ability to track long-term dependencies.
- There needs to be a flexible hidden state buffer as well which allows the Encoder focus on specific words for annotation.
- This can be thought of as an attention mechanism that allows the decoder to pull information from useful parts of the input rather than having to decode a single hidden state.
- Decoder takes both the hidden state, as well as the annotation into account while generating output.

Proposed Approaches - Summary

- Flexible length Inputs/Outputs can now be used for tasks by using a Encoder-Decoder mechanism.
- Using an alignment model also improves the scalability of the RNN model where we can expect better performance from increased length and more information.
- Introduced the GRU Unit which is cheaper to implement than the LSTM but gives comparable results , allowing us to scale even more efficiently

4 - Results

4.1 - Experiments

- The key experiments were performed on German/English translation on the Multi30k also known as The small-dataset WMT 2016 multimodal task, or Flickr30k.
- Some experiments were also performed on English/French SMT system in the framework of the WMT’14 dataset.
- To score the accuracy among the models , we used the BLEU Score as well as Perplexity abbreviated as PPL.

4.1 - Experiments

- To compare results, we solved the same task on 4 models
  a. Base RNN
  b. Base RNN + LSTM + Encoder + Decoder
  c. Base RNN + GRU + Encoder + Decoder
  d. Base RNN + GRU + Encoder + Decoder + Alignment Modelling

4.2 Hyperparameters - GRU Encoder-Decoder

- The first result observed is comparison between the following models on the ENG/FRE dataset.
- The RNN Encoder–Decoder used in the experiment had 1000 hidden units with the proposed gates at the encoder and at the decoder. The input matrix between each input symbol xhti and the hidden unit is approximated with two lower-rank matrices.
- We used rank-100 matrices, equivalent to learning an embedding of dimension 100 for each word.
- The activation function used for h˜ is a hyperbolic tangent function.

4.2 Hyperparameters - GRU Encoder-Decoder

- All the weight parameters in the RNN Encoder– Decoder were initialized by sampling from an isotropic zero-mean (white) Gaussian distribution with its standard deviation fixed to 0.01, except for the recurrent weight parameters.
- We used Adadelta and stochastic gradient descent to train the RNN Encoder–Decoder with hyperparameters = 10−6 and ρ = 0.95.
- Each input word was projected into the embedding space R 512 , and they were concatenated to form a 3072- dimensional vector. The concatenated vector was fed through two rectified layers (of size 1536 and 1024)

4.2 Hyperparameters - Alignment Model

- The encoder of the RNN search consists of forward and backward recurrent neural networks (RNN) each having 1000 hidden units. Its decoder has 1000 hidden units.
- In this case, we use a multilayer network with a single maxout hidden layer to compute the conditional probability of each target word.
- We use a mini-batch stochastic gradient descent (SGD) algorithm together with Adadelta to train each model. Each SGD update direction is computed using a minibatch of 80 sentences.

4.3 Results - Encoder-Decoder (ENG/FRE)

- We obtained the following results without alignment modelling on the ENG/FRE dataset.

Model	BLEU - dev	BLEU - Test
Baseline - Moses Model	30.64	33.30
RNN + LSTM	31.20	33.87
Encoder-Decoder + RNN	31.48	34.64

- This clearly shows the increased performance due to Encoder-Decoder Mechanisms.

4.3 Results - All Proposals (DE/ENG)

- The following results are a comparison of PPL and test loss on the same data between all the proposed models.

Model	Val Loss	Val PPL	Test Loss	Test PPL
Enc-Dec + LSTM	3.949	51.862	3.928	50.826
Enc-Dec + GRU	3.602	36.660	3.551	34.853
Enc-Dec +Alignment	3.318	27.609	3.154	23.418

- There is a clear trend of improvement with each addition of a proposal.

4.3 Results - Analysis

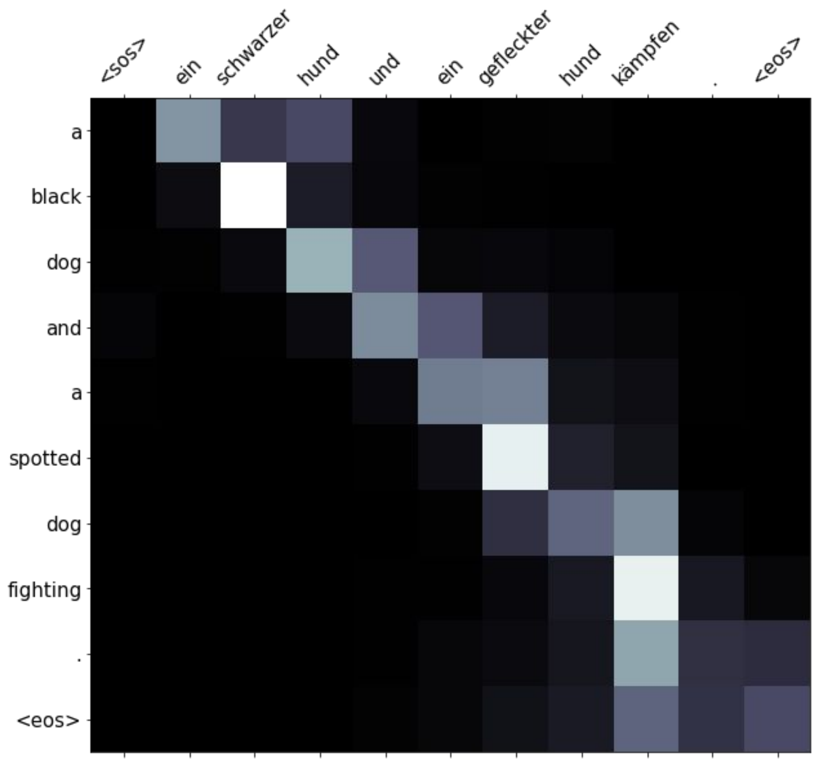
- Positives
    - Each proposal drastically improves all metrics of accuracy thereby signalling good design and architecture.
    - Encoder-Decoder can be used across all Deep Learning not only on Machine Translation due to
    - GRU improves over LSTMs by Cost but not by accuracy.
- Negatives
    - The model still misses a lot of contextual information and is prone to overfitting.
    - Model complexity and Variance could be too high.

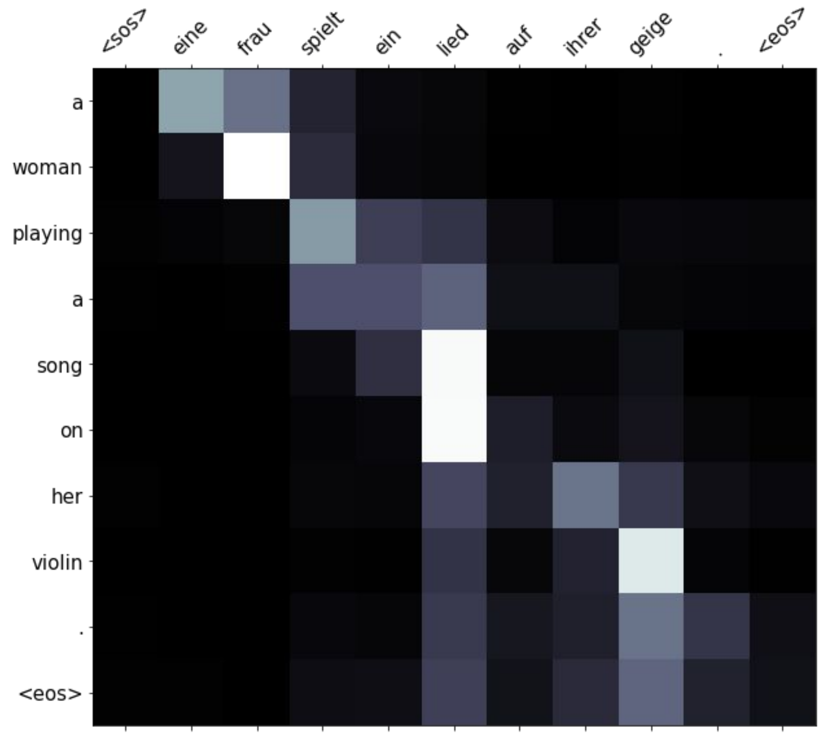
4.4 Example Translations

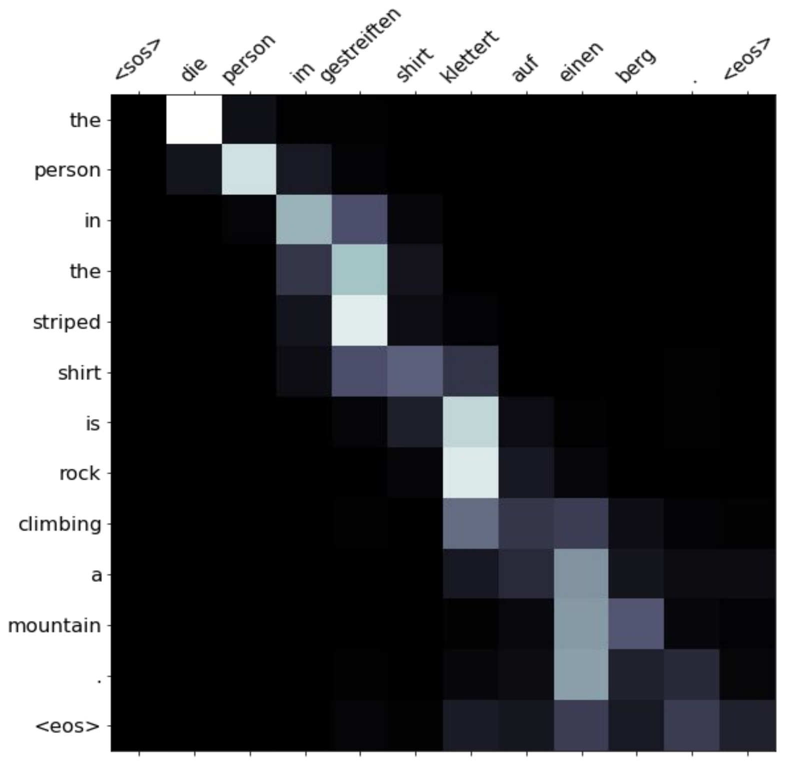
German Sentences	Target English Sentences	Translated English Sentences
`ein', 'schwarzer', 'hund', 'und', 'ein', 'gefleckter', 'hund', 'kämpfen', '.'`	`'a', 'black', 'dog', 'and', 'a', 'spotted', 'dog', 'are', 'fighting'`	`'a', 'black', 'dog', 'and', 'a', 'spotted', 'dog', 'fighting', '.', '<eos>'`
`'eine', 'frau', 'spielt', 'ein', 'lied', 'auf', 'ihrer', 'geige', '.'`	`'a', 'female', 'playing', 'a', 'song', 'on', 'her', 'violin', '.'`	`'a', 'woman', 'playing', 'a', 'song', 'on', 'her', 'violin', '.', '<eos>'`
`'die', 'person', 'im', 'gestreiften', 'shirt', 'klettert', 'auf', 'einen', 'berg', '.'`	`'the', 'person', 'in', 'the', 'striped', 'shirt', 'is', 'mountain', 'climbing', '.'`	`'the', 'person', 'in', 'the', 'striped', 'shirt', 'is', 'rock', 'climbing', 'a', 'mountain', '.', '<eos>'`

As you can see, the Translations are very accurate.

4.5 Visualization of Annotation Attention on Examples







The Lighter the box between the two letters, the more attention that has been computed between those two.

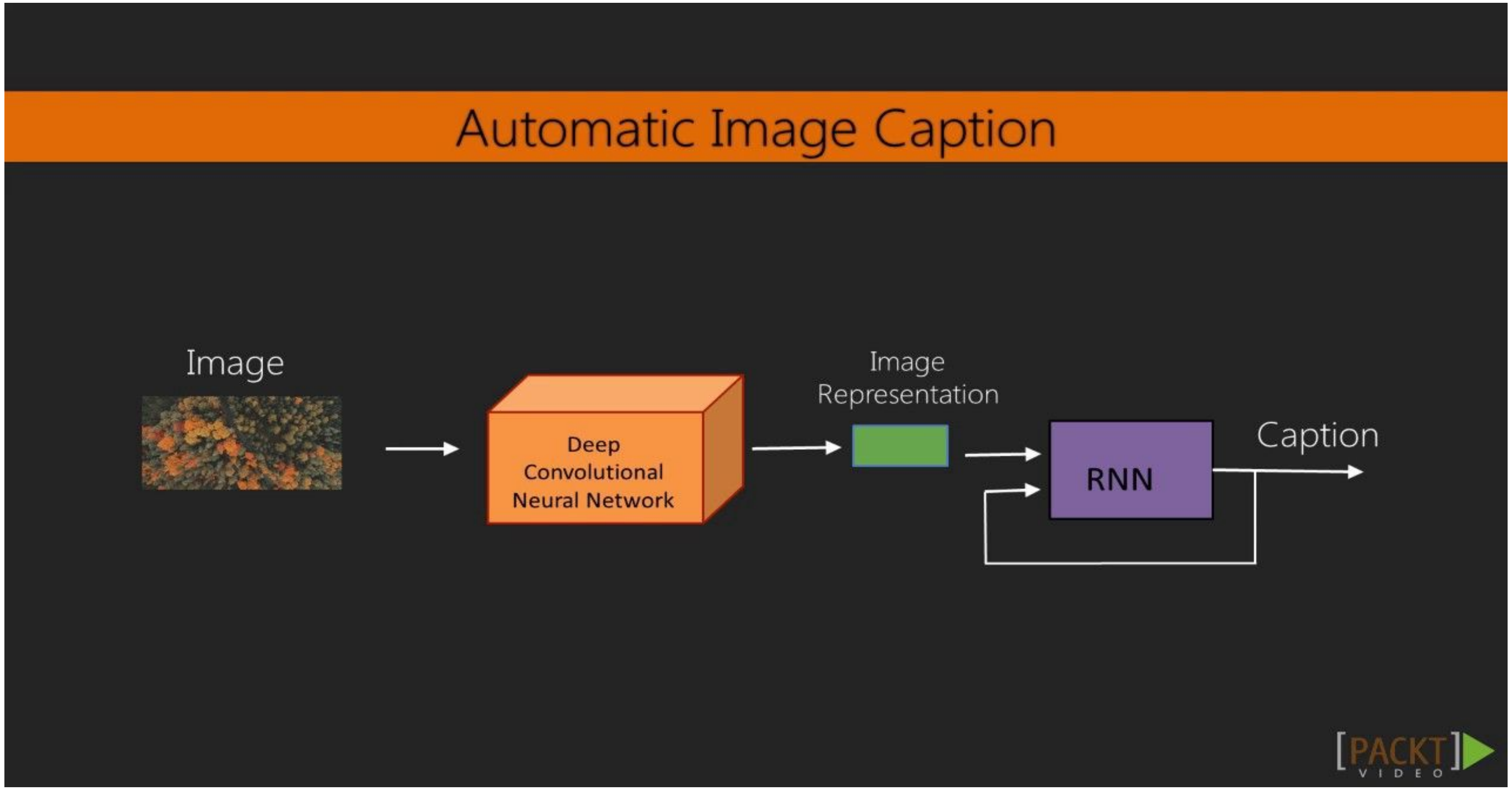
4.6 Results - Reproducible Code

[Reproducible Code for the Research]

Next Steps

- Encoder-Decoder Modularity could imply applications all over DNN including Image Captioning where Encoder would be a CNN instead of an RNN and Decoder would be a RNN.
- Similarly each problem could be a approached with plugin Enc-Dec pairs with each problem using a different set of Encoders and Decoders. Applications there could generate good results.
- Attention over word-word pairing in MT could be explored. Once a clear picture of context can be calculated by using an advanced Attention mechanism, NLP could achieve significant boosts in accuracy.

Next Steps


Automatic Image Caption
Image
Deep Convolutional Neural Network
Image Representation
RNN
Caption
[PACKT] VIDEO

Thanks to everyone who has contributed to and has inspired this research.

Disclaimer: This presentation is a work of fiction written from the perspective of a 2020 researcher traveling back in time to mid 2013 to share some 2020 xNN based application ideas; references to credit the actual inventors of the various ideas is provided at the end

References

- Note: As stated above, this presentation is a work of fiction; the following are the actual inventors of the ideas described in this presentation
- K.Cho, Y.Bengio , et. al., "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", arxiv:1406.1078, 2014.
- K.Cho, Y.Bengio , et. al., "Neural Machine Translation By Jointly Learning To Align And Translate", arxiv:1409.0473, 2014.
- K.Cho, Y.Bengio , et. al., "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling", arxiv:1412.3555,2014
- I. Sutskever, et. al.,"Sequence to Sequence Learning with Neural Networks", nips:5436, 2014.
- R. Staudemeyer, et. al., "Understanding LSTM", arxiv:1909.09586, 2019.
- Z. Lipton, et. al., "A Critical Review of Recurrent Neural Networks for Sequence Learning",arxiv:1506.00019,2015.

Disclaimer: This presentation is a work of fiction written from the perspective of a 2020 researcher traveling back in time to mid 2013 to share some 2020 xNN based application ideas; references to credit the actual inventors of the various ideas is provided at the end

Thank You!