Wikipedia Sentence Simplification

Alexander Wang
aswang96@stanford.edu

Ryan Kurohara
rkuro6@stanford.edu

David Liang
dwliang@stanford.edu

Abstract

Sentence simplification is a task that produces a simplified sentence from any input sentence. Sentence simplification systems can be useful tools for people whose first language is not English, children, or the elderly to help with understanding. Initial approaches to this task were primarily borrowed from neural machine translation systems with transformations at the sentence level or with architectures based on recurrent neural networks (RNNs). In our project, we implemented the state-of-the-art transformer architecture built with multi-head attention blocks, integrated it with a paraphrase database, and introduced a novel loss function.

1 Introduction

Our task is to a create a model that, given an input sentence, can return a simplified version of that sentence. This NLP task, called sentence simplification, can be a useful tool for people whose first language is not English, children, or the elderly. Initial approaches to this task were primarily borrowed from neural machine translation systems with transformations at the sentence level. Transformers and multi-head self-attention have revolutionized NLP progress. In this project, we utilize transformer architecture in the sentence simplification task, integrating into it a paraphrase database and incorporate a custom loss function to improve upon the state of the art model.

2 Related Work

The architectures, approaches, and evaluation metrics that we used are at the forefront of sentence simplification research today.

2.1 Sentence Simplification with Deep Reinforcement Learning

Recently, approaches to this task have been quite similar to those of translation tasks. The model would be given a pairs of sentences, one complex and one simplified. "Sentence Simplification with deep reinforcement learning" used advanced learning techniques to simplify sentences and achieved a respectable SARI score of 37.08. [1] The problem with many of these models is that they do not incorporate hand-curated rules which, although are expensive to create, are extremely effective.

2.2 Optimizing Statistical Machine Translation for Text Simplification

The paper "Optimizing Statistical Machine Translation for Text Simplification" was crucial for this NLP task because it introduced the idea of using large external paraphrase databases and also introduced the SARI metric, which has become the standard for evaluating sentence simplification. [2]

2.3 Integrating Transformer and Paraphrase Rules for Sentence Simplification

Finally, we decided to build off of the work of Zhao et al. because this paper achieves the highest SARI score recorded at 40.45. [3] Their model is build using multi-head attention blocks in a transformer architecture. They introduce two new approaches called DCSS and DMASS that incorporate an external paraphrase database of rules. Since this paper is the primarly focus of our project, we expand on the details much more in the “Main Approach” section.

3 Approach

3.1 Baselines and Oracle

We implemented five baselines and an oracle. The performance of these models are reported in the Results section.

We call the first baseline “identity” and it simply returns the original sentence as the “simplification” i.e. it does not modify the sentence. This is a natural baseline, as this provides no simplification, so we would hope our model can beat the performance of this baseline.

Our second baseline “skip” randomly removes words with some fixed probability. This was a natural baseline, because sentences can be made simpler by removing some words: however, this model doesn’t know anything about which words to remove, so we should also be able to beat it with any reasonable model.

For our third baseline, we used our code from HW5 to implement a “neural machine translation” baseline. It treats the unsimplified original sentences as one language and the simplified sentences as another language.

For our fourth baseline, we took the paraphrase database PPDB used in the paper we are following and created a model that performs any substitution it can find in the database [4]. For our fifth and final baseline, we took the paraphrase model and stacked it on top of the NMT model i.e. we apply the paraphrase model after the NMT translation. This sounded reasonable to us, as we would be combining the power of both models.

Finally, we implemented an oracle which “cheats” and returns a reference translation. It achieves very high BLEU and SARI scores but does not actually provide an upperbound to the SARI score, as proven by state of the art results, but is a nice target for us to aim for regardless.

3.2 Main Approach

The architecture of the model in our main approach is the transformer architecture, following the seminal work by Vaswani et al. [5]. It is built using many multi-head attention blocks. In this section, we delve into the details of multi-head attention, and then its use in the transformer architecture

3.2.1 Multi-Head Attention

An attention function maps a query and a set of key-value pairs to an output that is some weighted sum of the values.

Multi-head attention uses a scaled dot-product attention function. Scaled dot-product attention works as follows: we assume each query, key, and value is a vector where the query and keys are of dimension d_k and the values are of dimension d_v. Consider packing the queries into a matrix Q where each row corresponds to one query and packing the keys into a matrix K and the values into a matrix V in a similar fashion. Then we compute

$$\text{ScaledDotProductAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

In other words, we compute the dot products between all combinations of query and key vectors, scale by $\frac{1}{\sqrt{d_k}}$, and then multiply this by the values, producing a reweighted matrix of the original values. The scaling helps prevent the softmax from reaching regions where the gradient is extremely small and the learning is extremely slow.

Multi-head attention can be computed for d_{model}-dimensional keys, values, and queries as

$$\text{MultiHeadAttention}(Q, K, V) = \begin{bmatrix}\text{head}_1 \ldots \text{head}_h\end{bmatrix} W^O$$

where

$$\text{head}_i = \text{ScaledDotProductAttention}(QW_i^Q, KW_i^K, VW_i^V)$$

for some matrices $W_i^Q, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$. h represents the number of parallel attention layers, or "heads."

Essentially, multi-head attention first projects the keys, values, and queries into some higher dimensional space, then computes the attention function on these higher-dimensional projections. Vaswani et al. found that this allowed the model to "jointly attend to information from different representation subspaces at different positions." [5] We provide a schematic of the architecture in figure 1 of appendix for reference.

3.2.2 Transformer Architecture

The transformer architecture consists of an encoder stack and decoder stack. The encoder consists of 6 identical layers. Each layer consists of two sublayers: a multi-head self-attention mechanism followed by a simple 2 layer fully connected feed-forward network with a relu activation. There are residual connections surrounding each of these two sublayers, and then layer normalization is applied afterwards. The decoder also consists of 6 identical layers. Here, each layer consists of the same two sublayers as the encoder layers, but with an additional sublayer which performs multi-head attention over the output of the encoder stack with residual connections and layer normalization. Furthermore, the first Multi-Head Attention layer in the decoder masks out future tokens, else the model can simply look ahead and see the target output. The architecture can be seen in figure 2 of the appendix.

3.2.3 Paraphrase Database

The novel contribution of Zhao et al. is the integration of a paraphrase database into their model. The paraphrase database used is PPDB [4], a publicly available set of paraphrase rules. While PPDB includes phrasal and syntactic rules, Zhao et al. only incorporate lexical (e.g.) rules into their model. Lexical rules are used in this model to map from complex words to similar and simpler synonyms.

Zhao et al. have used the concept of a simplification memory store to modify the transformer architecture in two ways.

Deep Critic Sentence Simplification (DCSS)

The first is the Deep Critic Sentence Simplification (DCSS) model, and is designed to apply rules identified by the Simple PPDB by introducing a new loss function L_{critic} to train their model against. If the input sentence I contains a word that is the input or the output of a paraphrase rule, we calculate L_{critic} for the simplification as in figure 5 in the appendix, where I is the input sentence, θ are the model parameters, and w_{rule} is the simplification weight of the paraphrase rule, which is provided by PPDB. The higher the weight, the more simplifying power the rule holds.

Because L_{critic} only only focuses on words in PPDB, while L_{seq} focuses on the entire vocabulary trained upon, the model is trained by minimizing L_{critic} and L_{seq} alternately. Zhao et al. do not specify at what level this alternation occurs, but one can assume it is at a batch or sentence level.

We note that the paper does not specify what value L_{critic} takes in the case that the model produces neither the word 'winner' nor 'recipient' (see figure 5). We will assume it takes a value of 0.

We plan create a holistic loss function that is a simple or weighted sum of L_{seq} and L_{critic} and train consistently against this loss instead, which would provide consistency during the training process (as each loss function would be evaluated against every training example, instead of alternating).

Deep Memory Augmented Sentence Simplification (DMASS)

Zhao et al. introduced a novel way to incorporate the PPDB directly into the transformer architecture called Deep Memory Augmented Sentence Simplification or DMASS. The general idea of DMASS is to remember the contexts in which the correct simplifications occurred, so that when the input word is seen again, the model can more accurately produce the simplified output.

There are two main steps: memory update and memory read. In this section, we will walk through both in detail. Starting with memory update, given an input word, the model predicts an output word and checks if that word pair is represented in the PPDB. As one can recall, the Lexical PPDB is a map from a input word like “recipient” to a list of possible simplifications such as “receiver”, “winner”, and “host”. So, if the model is given “recipient” and predicts “winner”, we would like to remember that state for the future. We define the “pre_context” as the output of the second multi-head attention layer and the “post_context” as the output of the feed forward layer, both of which are in each decoder layer. So, when the model finds a match like “recipient” and “winner”, the memory module saves both the “pre_context” and “post_context”. If the pair is found again, the old contexts are overwritten.

Next, we consider how the model reads the memory. Imagine the input word “recipient” is encountered again, the pre_context is $c_{current}$, and the memory module for “recipient” is as follows.

output_word	pre_context	post_context
receiver	$c_{receiver}$	$r_{receiver}$
winner	c_{winner}	r_{winner}
host	c_{host}	r_{host}

We take the dot product between the current pre_context and each of the translation pre_context’s and then take the softmax of this output.

$$\alpha_i^r = \frac{exp(c_i \cdot c_{current})}{\sum exp(c_c \cdot c_{current})} \quad c_c = [c_{receiver}, c_{winner}, c_{host}]$$

Then we calculate $r_{context}$ with the following equation.

$$r_{context} = \sum \alpha_i^r r_r \quad r_r = [r_{receiver}, r_{winner}, r_{host}]$$

Finally, we add $r_{context}$ to $c_{current}$, and feed this new value into feed forward layer of the decoder. The result is that the model will be able to translate the word “recipient” better because it remembered the contexts in which different simplifications took place and can thus choose the correct one for the current context.

4 Experiments

4.1 Data

We trained our models using the WikiLarge, which is the largest Wikipedia corpus and contains over 296,402 normal-simple sentence pairs. For validation and testing, we used the Turk data set which has 2000 samples for validation and 356 samples for testing. Each sample has 8 simplification translations, in which multiple reference translations are necessary for calculating and effective SARI score [1]. We did a little extra pre-processing to put the words into lowercase and delete special characters. We chose not to substitute names for people and organizations and locations with special tokens, because of time constraints.

4.2 Evaluation Method

We used both BLUE and SARI in order to evaluate our models’ sentence simplification. As we learned in class, BLEU, or Bilingual Evaluation Understudy, is a measure of the number of matching Ngrams between the machine’s output and a reference translation. BLEU has been the traditional standard for MT systems, but it is not the best metric for sentence simplification because it compares all the Ngrams between the output and reference equally, instead of weighing the specific changes, such as insertions, more. To address these shortcomings, a new type of evaluation metric called SARI was introduced. SARI stands for ”System output Against References and against the normal sentence”, and it is the arithmetic mean of the Ngram (1-gram through 4-gram) F1-scores of the addition, deletion, and include operations. This means that it rewards specific operations, like addition operations for simplified words and deletion operations for unnecessary words. For further evidence of SARI’s validity, the paper Xu et al. (2016) revealed that SARI “correlates reasonably with human judgments”. [2]

4.3 Experimental Details

4.3.1 Baseline Models

For each of our baseline models and our oracle, we trained them over a dataset consisting of pairs of unsimplified and simplified sentences and then had the models simplify a set of test sentences, and used these outputs to compute BLEU and SARI scores.

The identity model required no training or tuning.

The skip model randomly discards words in the input, so again no training is needed. We played around with the skip probability, and noticed that as the skip probability decreases, the SARI and BLEU scores increase, which makes sense because with skip probability of 0, the skip model and identity models are the same. When the skip probability increases, more and more words are dropped, so the simplification becomes very different from the original sentence, eventually becoming a meaningless sequence of words.

The paraphrase model simply looks up in the dictionary which simplification to make, so again no training is needed. We tested the use of only phrasal substitutions (phrasal rules apply to sequences of words instead of single words), only word substitutions, and both phrasal and word substitutions at once.

Only the NMT model required training. We trained the model using HW5 code, and it terminated early after roughly 6 hours of GPU time. We ran it with all the default hyperparameters: an embedding size of 256, hidden state size of 256, dropout rate of 0.3, and we did include the character decoder. For decoding, our beam search used width of 5, and our max number of decoding time steps was 70. We used the trained NMT model in our paraphase NMT model as well, and tested the paraphrase NMT model with only phrasal substitutions, only word substitutions, and both phrasal word substitutions.

4.3.2 Transformer Models

For training the transformer models, we ran the models on VMs for usually 30 epochs (10-15 hours depending on model and hardware) or until stagnation in SARI scores. We provide a more detailed discussion of hyperparameters in the analysis section. We tuned the hyperparameters of the transformer in a number of ways, but we didn't have much time to tune them thoroughly. We found that limiting the vocabulary size to 10,000 words was a good balance between GPU memory use and SARI scores. Ideally, we would run with a larger vocabulary size, but it was taking too long. We also tested with a smaller vocabulary size (5,000 words) and found the results to be poorer. We also found that 4 layers for both the decoder and encoder stacks was the best for SARI score, as compared to the original paper's architecture that used 6 layers. The reason may be that more layers requires more data and computational resource and time to see good results, and we just didn't have more data and enough time to train larger models. We also found that using the 300d GloVe embeddings was better for SARI score than the 100d and 50d glove embeddings, and that projecting the query, key, and value vectors into a 64 dimensional space in multi-head attention ($d_v = 64$) was best. We also found that a dropout rate of .3 was best for achieving high training accuracy as well as high scores on our validation metrics. We found that the optimizer suggested in Vashwani et al. was extremely useful for accelerating early training. They essentially increase the learning rate for n_{warmup} steps before exponentially decaying the learning rate.

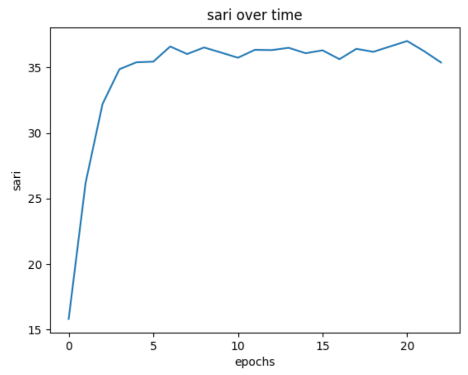
4.4 Results

We present our results in this section. Table 1 describes the SARI and BLEU scores of our baselines, our transformer models, and the state of the art models.

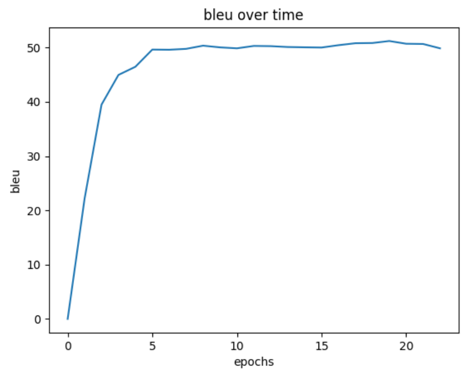
5 Analysis

5.1 Baselines

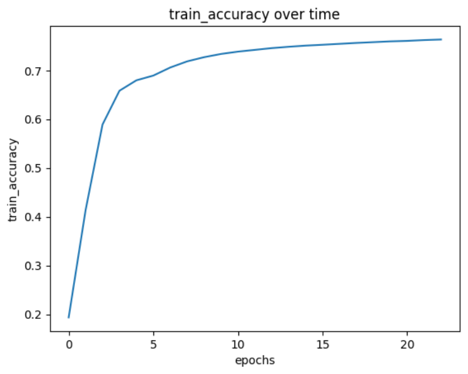
Our baseline models achieved modest results. In terms of SARI, the paraphrase model performed fairly well for both words and phrases. This is expected because the substitution operations are highly accurate: words and phrases are directly simplified using a database of a human-curated features. Of



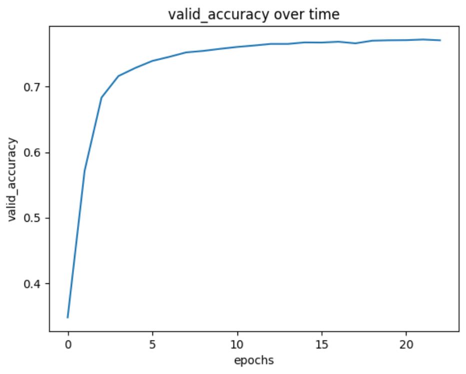
(a) SARI Scores



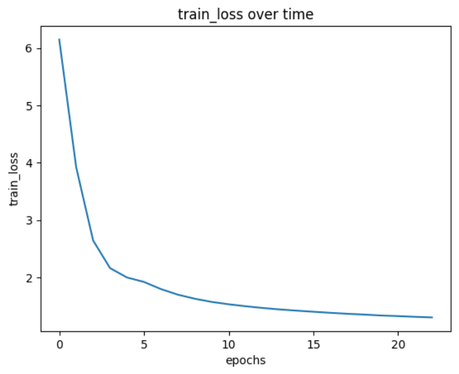
(b) BLEU Scores



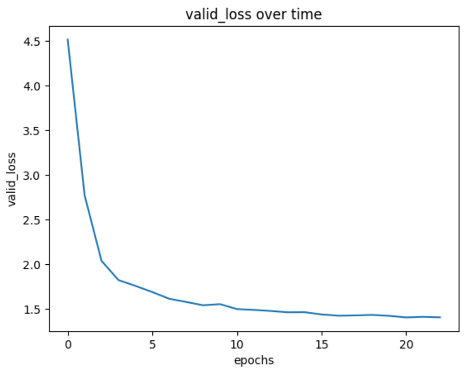
(c) Train Accuracy



(d) Valid Accuracy



(e) Train Loss



(f) Validation Loss

Figure 1: Training plots from the best transformer model we had.

Table 1: Results

Model	SARI	BLEU
baseline models		
identity	25.95	61.47
skip (prob=.5)	21.62	8.97
paraphrase (word)	**32.02**	56.29
paraphrase (phrase)	31.94	55.85
paraphrase (both)	33.91	51.74
nmt	25.61	16.13
paraphrase nmt (word)	24.99	14.76
paraphrase nmt (phrase)	24.75	14.14
paraphrase nmt (both)	24.23	13.05
transformer models		
base transformer	37.02	50.71
transformer + DMASS	**37.04**	47.51
transformer + critic loss	9.71	0
state of the art / oracle		
state of the art (SBSMT + PPDB + SARI)	**39.96**	73.08
oracle	37.92	91.148

Table 2: Good translation

Input	adrastea was discovered by david c. jewitt and g. edward danielson in voyager probe .
Model output	adrastea was found by david c. jewitt and g. edward <unk> in voyager probe .
Gold	it was found by david c. jewitt and g. edward danielson in voyager probe .

course, our skip was fairly ineffective. Our baseline NMT model produced values slightly below the identity model. In terms of BLEU, the paraphrase model performed worse than the identity model and the NMT model preformed much worse. One explanation for the low scores of the NMT score is that, for sentence simplification, models with hand-curated features have historically performed much better than NMT systems without it [3]. Because it takes a huge of amount of training data in order to learn all the simplification rules, our NMT model didn't perform well with only six hours of training.

5.2 Transformer

However, the transformer and its augmentations performed exceptionally well, reaching SARI scores close to state-of-the-art models. It is interesting to note that the state of the art model outperforms the oracle in terms of the SARI score, which is possible because SARI score is computed from 8 different simplifications of the same sentence from mechanical turk workers, so it is possible that the given simplification differs from the 8 other simplifications, while the model made simplifications more similar to those of the turk workers.

5.3 DMASS

The addition of the DMASS, or memory module, barely affected the scores of our transformer model, if at all. According to the paper that introduced DMASS, the SARI score of the base transformer was 38.84, and the SARI score of DMASS was only 39.81 [3]. The small increase in scores in their paper is important to consider because it reveals how difficult it is for DMASS to learn. The reason is that updating the augmented memory map happens very infrequently, only when with the output word is

Table 3: Bad translation

Input	frances was later absorbed by an extratropical cyclone on november .
Model output	emily was later absorbed by an extratropical cyclone on november .
Gold	frances was later destroyed by an extratropical low on november .

a valid simplification of the input word. On top of that, the model needs to learn these simplifications for every word in the PPDB, and the knowledge of certain rules doesn't transfer to other rules. As a result, it is possible that we need to train are model more in order to see substantial results. Secondly, another reason why our DMASS didn't work as well is that the paraphrase database we used was only 230,000 rules while they used a more advanced version of PPDB which has 4.5 million lines. Since the most common rules are included in the smaller databases, we thought that using the smaller database would be not only convenient but also just as effective. But, it's still possible that our model couldn't learn as much as theirs because we were limited to a smaller amount of rules.

5.4 Critic Loss

The addition of critic loss was detrimental to our model's performance, resulting in a SARI score of 9.71 and a BLEU score of 0 (recall that our worst baseline achieves 21.62 and 8.97 respectively). We believe that this performance is due to the fact that critic loss is calculated independently of the target translation. Instead, it relies only on the PPDB paraphrase rules. More specifically, the model is rewarded when it simplifies a word according to the PPDB, and is penalized if it outputs a word that can be simplified. This incentivizes the model to simplify by blindly applying the PPDB paraphrase rules, irregardless of context and correctness. We observed that, by doing this, the model was able to achieve negative loss, at the cost of accuracy.

To alleviate this problem, we attempted weighting the critic loss contribution by a constant fraction, and dividing it by the length of the output sequence to get an average critic loss per word. Neither of these methods raised the evaluation metrics.

5.5 Critic Loss Modification

Given more time, we would have tightened our conditions for when critic loss should reward or penalize a translation by cross-referencing it with the target sequence. If the model outputs a simple version of a word *and* it matches the target sequence, then we reward it. Conversely, if the model outputs a complex version of a word *and* the simple version matches the target sequence, then we penalize it. This modification would only allow the critic loss to affect the model when appropriate.

5.6 Output Analysis

We show some resulting simplifications in Tables 2 and 3. We found it hard to even find a good output, because our model seems to copy many words from the input sentence into the output sentence, and infrequently does any simplification. On occasion, as shown in Table 3, the model substitutes a complex word for a simpler word. In this example, "discovered" was replaced by "found" which is a great simplification. For a worse translation, we can consider Table 3. "frances" was replaced by "emily" as a "simplification" but this should not have happened as it changes the meaning of the sentence. If we had masked out all nouns referring to people, we could have prevented this. In both examples, not that much simplification was done, while the SARI score for the model is quite decent, so we postulate that the state of the art models also see similar results of little simplification being done while achieving high SARI scores.

6 Conclusion

In conclusion, our group integrated a transformer model and implemented two new approaches to sentence simplification, Deep Critic Sentence Simplification (DCSS) and Deep Memory Augmented Sentence Simplification (DMASS). The transformer model proved well, but the results of DCSS and DMASS were not as ideal. We found that working on the combination of the transformer architecture with hand-curated rules through PPDB to be fascinating. In the future, it will be interesting to find novel ways at integrating human rules into advanced neural networks to see if the combination is stronger than each part individually.

Bibliography

[1] Xingxing Zhang and Mirella Lapata. Sentence simplification with deep reinforcement learning. pages 595–605, 2017.

[2] Wei Xu, Courtney Napoles, Ellie Pavlick, Quanze Chen, and Chris Callison-Burch. Optimizing statistical machine translation for text simplification. 2016.

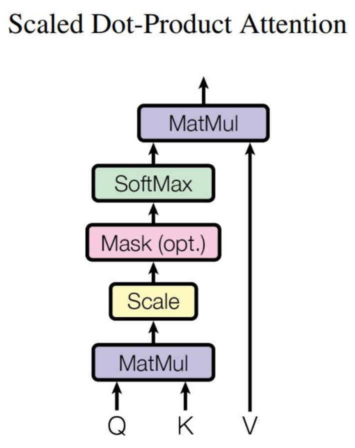
[3] Sanqiang Zhao, Rui Meng, Daqing He, Saptono Andi, and Parmanto Bambang. Integrating transformer and paraphrase rules for sentence simplification. 2018.

[4] Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-burch. Ppdb: The paraphrase database. 2013.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv e-prints*, page arXiv:1706.03762, Jun 2017.

Appendices

Figure 2: (left) Scaled dot-product attention. (left) Multi-head attention. [5]



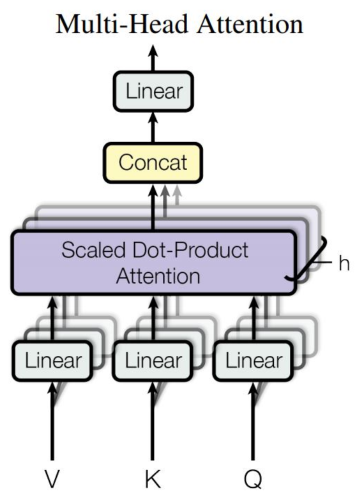


Figure 3: Transformer architecture. [5]

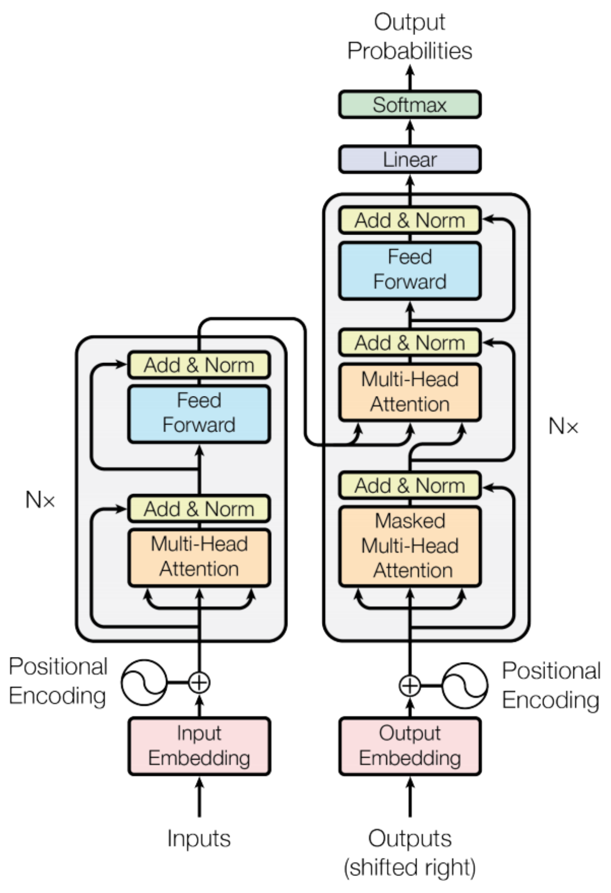


Figure 4: Deep Memory Augmented Sentence Architecture. [3]

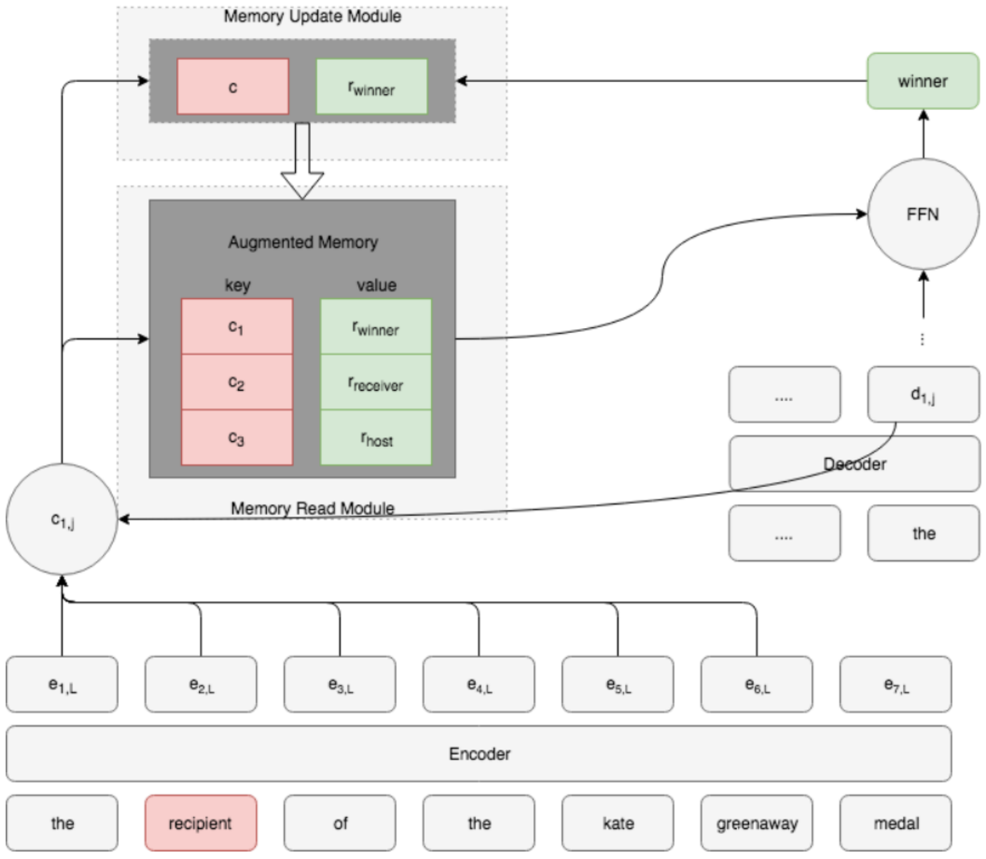


Figure 5: Critic Loss. [3]

$$L_{critic} = \begin{cases} -w_{rule} log P(winner|I, \theta) \\ \qquad\qquad\qquad \text{if model generates recipient} \\ w_{rule} log P(recipient|I, \theta) \\ \qquad\qquad\qquad \text{if model generates winner} \end{cases} \tag{4}$$