Self-Attention: A Better Building Block for Sentiment Analysis Neural Network Classifiers

Artaches Ambartsoumian
School of Computing Science
Simon Fraser University
Burnaby, BC, CANADA
aambarts@sfu.ca

Fred Popowich
School of Computing Science
Simon Fraser University
Burnaby, BC, CANADA
popowich@sfu.ca

Abstract

Sentiment Analysis has seen much progress in the past two decades. For the past few years, neural network approaches, primarily RNNs and CNNs, have been the most successful for this task. Recently, a new category of neural networks, self-attention networks (SANs), have been created which utilizes the attention mechanism as the basic building block. Self-attention networks have been shown to be effective for sequence modeling tasks, while having no recurrence or convolutions. In this work we explore the effectiveness of the SANs for sentiment analysis. We demonstrate that SANs are superior in performance to their RNN and CNN counterparts by comparing their classification accuracy on six datasets as well as their model characteristics such as training speed and memory consumption. Finally, we explore the effects of various SAN modifications such as multi-head attention as well as two methods of incorporating sequence position information into SANs.

1 Introduction

Sentiment analysis, also know as opinion mining, deals with determining the opinion classification of a piece of text. Most commonly the classification is whether the writer of a piece of text is expressing a position or negative attitude towards a product or a topic of interest. Having more than two sentiment classes is called fine-grained sentiment analysis with the extra classes representing intensities of positive/negative sentiment (e.g. very-positive) and/or the neutral class. This field has seen much growth for the past two decades, with many applications and multiple classifiers proposed [Mäntylä et al., 2018]. Sentiment analysis has been applied in areas such as social media [Jansen et al., 2009], movie reviews [Pang et al., 2002], commerce [Jansen et al., 2009], and health care [Greaves et al., 2013b] [Greaves et al., 2013a].

In the past few years, neural network approaches have consistently advanced the state-of-the-art technologies for sentiment analysis and other natural language processing (NLP) tasks. For sentiment analysis, the neural network approaches typically use pre-trained word embeddings such as word2vec [Mikolov et al., 2013] or GloVe[Pennington et al., 2014] for input, which get processed by the model to create a sentence representation that is finally used for a softmax classification output layer. The main neural network architectures that have been applied for sentiment analysis are recurrent neural networks(RNNs) [Tai et al., 2015] and convolutional neural networks (CNNs) [Kim, 2014], with RNNs being more popular of the two. For RNNs, typically gated cell variants such as long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997], Bi-Directional LSTM (BiLSTM) [Schuster and Paliwal, 1997], or gated recurrent unit (GRU) [Cho et al., 2014] are used.

Most recently, Vaswani et al. [Vaswani et al., 2017] introduced the first fully-attentional architecture, called Transformer, which utilizes only the self-attention mechanism and demonstrated its effectiveness on neural machine translation (NMT). The Transformer model achieved state-of-the-art performance on multiple machine translation datasets, without having recurrence or convolution components. Since then, self-attention networks have been successfully applied to a variety of tasks, including: image classification [Parmar et al., 2018], generative adversarial networks [Zhang et al., 2018], automatic speech recognition [Povey et al., 2017], text summarization [Liu et al., 2018], semantic role labeling [Strubell et al., 2018], as well as natural language inference and sentiment analysis [Shen et al., 2018].

In this paper we demonstrate that self-attention is a better building block compared to recurrence or convolutions for sentiment analysis classifiers. We

Proceedings of the 9th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, pages 130–139
Brussels, Belgium, October 31, 2018. ©2018 Association for Computational Linguistics
https://doi.org/10.18653/v1/P17

extend the work of [Barnes et al., 2017] by exploring the behaviour of various self-attention architectures on six different datasets and making direct comparisons to their work. We set our baselines to be their results for *LSTM*, *BiLSTM*, and *CNN* models, and used the same code for dataset preprocessing, word embedding imports, and batch construction. Finally, we explore the effectiveness of SAN architecture variations such as different techniques of incorporating positional information into the network, using multi-head attention, and stacking self-attention layers. Our results suggest that relative position representations is superior to positional encodings, as well as highlight the efficiency of the stacking self-attention layers.

Source code is publicly available[1].

2 Background

The attention mechanism was introduced by [Bahdanau et al., 2014] to improve the RNN encoder-decoder sequence-to-sequence architecture for NMT [Sutskever et al., 2014]. Since then, it has been extensively used to improve various RNN and CNN architectures ([Cheng et al., 2016]; [Kokkinos and Potamianos, 2017]; [Lu et al., 2016]). The attention mechanism has been an especially popular modification for RNN-based architectures due to its ability to improve the modeling of long range dependencies ([Daniluk et al., 2017]; [Zhou et al., 2018]).

2.1 Attention

Originally [Bahdanau et al., 2014] described attention as the process of computing a context vector for the next decoder step that contains the most relevant information from all of the encoder hidden states by performing a weighted average on the encoder hidden states. How much each encoder state contributes to the weighted average is determined by an alignment score between that encoder state and previous hidden state of the decoder.

More generally, we can consider the previous decoder state as the query vector, and the encoder hidden states as key and value vectors. The output is a weighted average of the value vectors, where the weights are determined by the compatibility function between the query and the keys. Note that the keys and values can be different sets of vectors [Vaswani et al., 2017].

[1]`https://github.com/Artaches/SSAN-self-attention-sentiment-analysis-classification`

The above can be summarized by the following equations. Given a query q, values $(v_1, ..., v_n)$, and keys $(k_1, ..., k_n)$ we compute output z:

$$z = \sum_{j=1}^{n} \alpha_j(v_j) \tag{1}$$

$$\alpha_j = \frac{\exp f(k_j, q)}{\sum_{i=1}^{n} \exp f(k_i, q)} \tag{2}$$

α_j is computed using the softmax function where $f(k_i, q)$ is the compatibility score between k_i and q,

For the compatibility function, we will be using using the scaled dot-product function from [Vaswani et al., 2017]:

$$f(k, q) = \frac{(k)(q)^T}{\sqrt{d_k}} \tag{3}$$

where d_k is the dimension of the key vectors. This scaling is done to improve numerical stability as the dimension of keys, values, and queries grows.

2.2 Self-Attention

Self-attention is the process of applying the attention mechanism outlined above to every position of the source sequence. This is done by creating three vectors (query, key, value) for each sequence position, and then applying the attention mechanism for each position x_i, using the x_i query vector and key and value vectors for all other positions. As a result, an input sequence $X = (x_1, x_2, ..., x_n)$ of words is transformed into a sequence $Y = (y_1, y_2, ..., y_n)$ where y_i incorporates the information of x_i as well as how x_i relates to all other positions in X. The (query, key, value) vectors can be created by applying learned linear projections [Vaswani et al., 2017], or using feed-forward layers.

This computation can be done for the entire source sequence in parallel by grouping the queries, keys, and values in Q, K, V matrices[Vaswani et al., 2017].

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{4}$$

Furthermore, instead of performing self-attention once for (Q,K,V) of dimension d_{model}, [Vaswani et al., 2017] proposed multi-head attention, which performs attention h times on projected (Q,K,V) matrices of dimension d_{model}/h. For each head, the (Q,K,V) matrices are uniquely projected to dimension d_{model}/h and self-attetnion is performed to yield an output of dimension d_{model}/h. The outputs of each head are then concatenated, and once again a linear projection layer is applied, resulting in an output of same dimensionality as performing self-attention once on the original (Q,K,V) matrices. This process is described by the following formulas:

$$\begin{aligned}\text{MultiHead}(Q, K, V) = \\ \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O\end{aligned} \tag{5}$$

$$\begin{aligned}\text{where head}_\text{i} = \\ \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned} \tag{6}$$

Where the projections are parameter matrices $W_i^Q \in R^{d_{\text{model}} \times d_k}$, $W_i^K \in R^{d_{\text{model}} \times d_k}$, $W_i^V \in R^{d_{\text{model}} \times d_v}$ and $W^O \in R^{hd_v \times d_{\text{model}}}$.

2.3 Position Information Techniques

The attention mechanism is completely invariant to sequence ordering, thus self-attention networks need to incorporate positional information. Three main techniques have been proposed to solve this problem: adding sinusoidal positional encodings or learned positional encoding to input embeddings, or using relative positional representations in the self-attention mechanism.

2.3.1 Sinusoidal Position Encoding

This method was proposed by [Vaswani et al., 2017] to be used for the Transformer model. Here, positional encoding (PE) vectors are created using sine and cosine functions of difference frequencies and then are added to the input embeddings. Thus, the input embeddings and positional encodings must have the same dimensionality of d_{model}. The following sine and cosine functions are used:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

where pos is the sentence position and i is the dimension. Using this approach, sentences longer than those seen during training can still have positional information added. We will be referring to this method as PE.

2.3.2 Learned Position Encoding

In a similar method, learned vectors of the same dimensionality, that are also unique to each position can be added to the input embeddings instead of sinusoidal position encodings[Gehring et al., 2017]. There are two downsides to this approach. First, this method cannot handle sentences that are longer than the ones in the training set as no vectors are trained for those positions. Second, the further position will likely not get trained as well if the training dataset has more short sentences than longer ones. Vaswani et al. [2017] also reported that these perform identically to the positional encoding approach.

2.3.3 Relative Position Representations

Relative Position Representations (RPR) was introduced by [Shaw et al., 2018] as a replacement of positional encodings for the Transformer. Using this approach, the Transformer was able to perform even better for NMT. Out of the three discussed, we have found this approach to work best and we will be referring to this method as RPR throughout the paper.

For this method, the self-attention mechanism is modified to explicitly learn the relative positional information between every two sequence positions. As a result, the input sequence is modeled as a labeled, directed, fully-connected graph, where the labels represent positional information. A tunable parameter k is also introduced that limits the maximum distance considered between two sequence positions. [Shaw et al., 2018] hypothesized that this will allow the model to generalize to longer sequences at test time.

3 Proposed Architectures

In this work we propose a simple self-attention (SSAN) model and test it in 1 as well as 2 layer stacked configurations. We designed the SSAN architecture to not have any extra components in order to compare specifically the self-attention component to the recurrence and convolution components of LSTM and CNN models. Our goal is to test the effectiveness of the main building blocks. We compare directly the results of two proposed architectures, *1-Layer-SSAN* and *2-Layer-SSAN*, to the LSTM, BiLSTM, and CNN architectures from [Barnes et al., 2017].

SSAN performs self-attention only once, which is identical to 1-head multi-head attention. SSAN

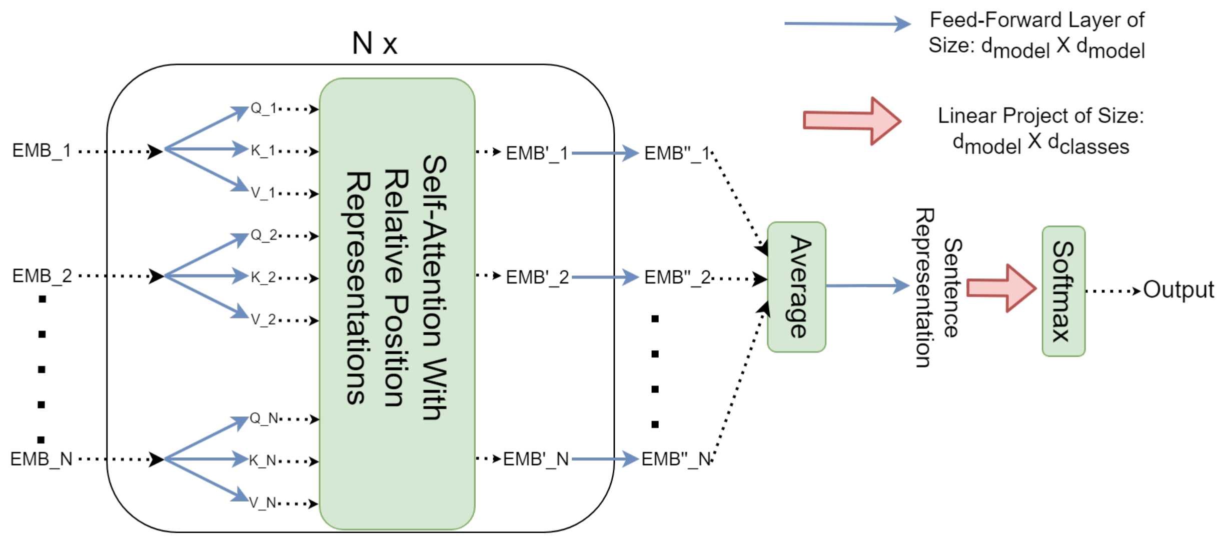
N x
Feed-Forward Layer of Size: dmodel X dmodel
Linear Project of Size: dmodel X dclasses
EMB_1
EMB_2
EMB_N
Q_1
K_1
V_1
Q_2
K_2
V_2
Q_N
K_N
V_N
Self-Attention With Relative Position Representations
EMB'_1
EMB'_2
EMB'_N
EMB"_1
EMB"_2
EMB"_N
Average
Sentence Representation
Softmax
Output

Figure 1: SSAN Model Architecture

takes in input word embeddings and applies 3 feed-forward layers to obtain Q,K,V representations on which self-attention is performed. The output of the self-attention layer is passed through another feed-forward layer. This process is done twice for *2-Layer-SSAN*, using the output of first layer as input for the second. The output of the last self-attention layer is averaged and a feed-forward layer is then applied to create a sentence representation vector of fixed dimension d_{model}. Finally, the sentence representation vector is passed through an output softmax layer that has an output dimension of d_{classes}. Dropout [Srivastava et al., 2014] is applied on input word embeddings, output of self-attention layers, on the sentence representation vector. The architecture is visualized in Figure 1. All feed-forward layers use ReLU [Nair and Hinton, 2010] activation functions. For relative positional representation, we set the parameter k=10, which is the maximum relative position considered for each input sequence position.

Finally, we also show results for other, more complex, self-attention architectures that are based on the Transformer. We take a 2 layer Transformer encoder as described by [Vaswani et al., 2017], then just like for SSAN, average the output of the second layer to create a sentence representation and apply a feed-forward layer followed by an output softmax layer. Dropout is applied as described in [Vaswani et al., 2017] as well as on the sentence representation vector.

4 Experiments

To reduce implementation deviations from previous work, we use the codebase from [Barnes et al., 2017] and only replace the model and training process. We re-use the code for batch pre-processing and batch construction for all datasets, accuracy evaluation, as well as use the same word embeddings[2]. All neural network models use cross-entropy for the training loss.

All experiments and benchmarks were run using a single GTX 1080 Ti with an i7 5820k @ 3.3Ghz and 32Gb of RAM. For model implementations: LSTM, BiLSTM, and CNN baselines are implemented in Keras_2.0.8 [Chollet et al., 2015] with Tensorflow_1.7 backend using cuDNN_5.1.5 and CUDA_9.1. All self-attention models are implemented in Tensorflow_1.7 and use the same CUDA libraries.

4.1 Datasets

In order to determine if certain neural network building blocks are superior, we test on six datasets from [Barnes et al., 2017] with different properties. The summary for dataset properties is in Table 1.

The Stanford Sentiment Treebank (*SST-fine*) [Socher et al., 2013] deals with movie reviews, containing five classes [very-negative, negative, neutral, positive, very-positive]. (*SST-binary*) is constructed from the same data, except the neutral class sentences are removed, all negative classes are

[2]`https://github.com/jbarnesspain/sota_sentiment`

	Train	Dev.	Test	# of Classes	Average Sent. Length	Max Sent. Length	Vocab. Size	Wiki Emb. Coverage	300D Emb. Coverage
SST-fine	8,544	1,101	2,210	5	19.53	57	19,500	94.4%	89.0%
SST-binary	6,920	872	1,821	2	19.67	57	17,539	95.0%	89.6%
OpeNER	2,780	186	743	4	4.28	23	2,447	94.2%	99.3%
SenTube-A	3,381	225	903	2	28.54	127	18,569	75.6%	74.5%
SenTube-T	4,997	333	1,334	2	28.73	121	20,276	70.4%	76.0%
SemEval	6,021	890	2,376	3	22.40	40	21,163	77.1%	99.8%

Table 1: Modified Table 2 from [Barnes et al., 2017]. Dataset statistics, embedding coverage of dataset vocabularies, as well as splits for Train, Dev (Development), and Test sets. The 'Wiki' embeddings are the 50, 100, 200, and 600 dimension used for experiments.

grouped, and all positive classes are grouped. The datasets are pre-processed to only contain sentence-level labels, and none of the models reported in this work utilize the phrase-level labels that are also provided.

The *OpeNER* dataset [y Montse Cuadros y Seán Gaines y German Rigau, 2013] is a dataset of hotel reviews with four sentiment classes: very negative, negative, positive, and very positive. This is the smallest dataset with the lowest average sentence length.

The SenTube datasets [Uryupina et al., 2014] consist of YouTube comments with two sentiment classes: positive and negative. These datasets contain the longest average sentence length as well as the longest maximum sentence length of all the datasets.

The SemEval Twitter dataset (*SemEval*) [Nakov et al., 2013] consists of tweets with three classes: positive, negative, and neutral.

4.2 Embeddings

We use the exact same word embeddings as [Barnes et al., 2017]. They trained the 50, 100, 200, and 600-dimensional word embeddings using the word2vec algorithm described in [Mikolov et al., 2013] on a 2016 Wikipedia dump. In order to compare to previous work, they also used the publicly available Google 300-dimensional word2vec embeddings, which are trained on a part of Google News dataset[3]. For all models, out-of-vocabulary words are initialized randomly from the uniform distribution on the interval [-0.25 , 0.25].

[3]`https://code.google.com/archive/p/word2vec/`

4.3 Baselines

We take 5 classifiers from [Barnes et al., 2017] and use their published results as baselines. Two of the methods are based on logistic regression, *Bow* and *Ave*, and 3 are neural network based, *LSTM*, *BiLSTM*, and *CNN*.

The (*Bow*) baseline is a L2-regularized logistic regression trained on bag-of-words representation. Each word is represented by a one-hot vectors of size $n = |V|$, where $|V|$ is the vocabulary size.

The (*Ave*) baseline is also a L2-regularized logistic regression classifier except trained on the average of the 300-dimension word embeddings for each sentence.

The *LSTM* baseline, input word embeddings are passed into an LSTM layer. Then a 50-dimensional feed-forward layer with ReLU activations is applied, followed by a softmax layer that produces that model classification outputs. Dropout [Srivastava et al., 2014] is applied to the input word embeddings for regularization.

The *BiLSTM* baseline is the same as *LSTM*, except that a second LSTM layer is used to process the input word embeddings in the reverse order. The outputs of the two LSTM layers are concatenated and passed a feed-forward layer, following by the output softmax layer. Dropout is applied identically as in *LSTM*. This modification improves the networks ability to capture long-range dependencies.

The final baseline is a simple *CNN* network. The input sequence of n embeddings is reshaped to an $n \times R$ dimensional matrix M, where R is the dimensionality of the embeddings. Convolutions with filter size of [2,3,4] are applied to M, following by a pooling layer of length 2. As for *LSTM* networks, a

	Model	Dim.	SST-fine	SST-binary	OpeNER	SenTube-A	SenTube-T	SemEval	Macro-Avg.
Baselines	*Bow*		40.3	80.7	77.1	60.6	66.0	65.5	65.0
	Ave	300	41.6	80.3	76.3	61.5	64.3	63.6	64.6
	LSTM	50	43.3 (1.0)	80.5 (0.4)	81.1 (0.4)	58.9 (0.8)	63.4 (3.1)	63.9 (1.7)	65.2 (1.2)
		100	44.1 (0.8)	79.5 (0.6)	82.4 (0.5)	58.9 (1.1)	63.1 (0.4)	67.3 (1.1)	65.9 (0.7)
		200	44.1 (1.6)	80.9 (0.6)	82.0 (0.6)	58.6 (0.6)	65.2 (1.6)	66.8 (1.3)	66.3 (1.1)
		300	45.3 (1.9)	81.7 (0.7)	82.3 (0.6)	57.4 (1.3)	63.6 (0.7)	67.6 (0.6)	66.3 (1.0)
		600	44.5 (1.4)	83.1 (0.9)	81.2 (0.8)	57.4 (1.1)	65.7 (1.2)	67.5 (0.7)	66.5 (1.0)
	BiLSTM	50	43.6 (1.2)	82.9 (0.7)	79.2 (0.8)	59.5 (1.1)	65.6 (1.2)	64.3 (1.2)	65.9 (1.0)
		100	43.8 (1.1)	79.8 (1.0)	82.4 (0.6)	58.6 (0.8)	66.4 (1.4)	65.2 (0.6)	66.0 (0.9)
		200	44.0 (0.9)	80.1 (0.6)	81.7 (0.5)	58.9 (0.3)	63.3 (1.0)	66.4 (0.3)	65.7 (0.6)
		300	45.6 (1.6)	82.6 (0.7)	82.5 (0.6)	59.3 (1.0)	66.2 (1.5)	65.1 (0.9)	66.9 (1.1)
		600	43.2 (1.1)	83.0 (0.4)	81.5 (0.5)	59.2 (1.6)	66.4 (1.1)	68.5 (0.7)	66.9 (0.9)
	CNN	50	39.9 (0.7)	81.7 (0.3)	80.0 (0.9)	55.2 (0.7)	57.4 (3.1)	65.7 (1.0)	63.3 (1.1)
		100	40.1 (1.0)	81.6 (0.5)	79.5 (0.9)	56.0 (2.2)	61.5 (1.1)	64.2 (0.8)	63.8 (1.1)
		200	39.1 (1.1)	80.7 (0.4)	79.8 (0.7)	56.3 (1.8)	64.1 (1.1)	65.3 (0.8)	64.2 (1.0)
		300	39.8 (0.7)	81.3 (1.1)	80.3 (0.9)	57.3 (0.5)	62.1 (1.0)	63.5 (1.3)	64.0 (0.9)
		600	40.7 (2.6)	82.7 (1.2)	79.2 (1.4)	56.6 (0.6)	61.3 (2.0)	65.9 (1.8)	64.4 (1.5)
Self-Attention Models	*1-Layer SSAN + RPR*	50	42.8 (0.8)	79.6 (0.3)	78.6 (0.5)	$\boxed{\textbf{64.1}}$ (0.4)	67.0 (1.0)	67.1 (0.5)	66.5 (0.6)
		100	44.6 (0.3)	82.3 (0.3)	81.6 (0.5)	61.6 (1.3)	68.6 (0.6)	68.6 (0.5)	67.9 (0.6)
		200	45.4 (0.4)	83.1 (0.5)	82.3 (0.4)	62.2 (0.6)	68.4 (0.8)	70.5 (0.4)	68.6 (0.5)
		300	$\boxed{\textbf{48.1}}$ (0.4)	$\boxed{\textbf{84.2}}$ (0.4)	83.8 (0.2)	62.5 (0.3)	68.4 (0.8)	$\boxed{\textbf{72.2}}$ (0.8)	$\boxed{\textbf{69.9}}$ (0.5)
		600	47.7 (0.7)	83.6 (0.4)	83.1 (0.4)	62.0 (0.4)	$\boxed{\textbf{68.8}}$ (0.7)	70.5 (0.8)	69.2 (0.5)
	2-Layer SSAN + RPR	50	43.2 (0.9)	79.8 (0.2)	79.2 (0.6)	63.0 (1.3)	66.6 (0.5)	67.5 (0.7)	66.5 (0.7)
		100	45.0 (0.4)	81.6 (0.9)	81.1 (0.4)	63.3 (0.7)	67.7 (0.5)	68.7 (0.4)	67.9 (0.5)
		200	46.5 (0.7)	82.8 (0.5)	82.3 (0.6)	61.9 (1.2)	68.0 (0.8)	69.6 (0.8)	68.5 (0.8)
		300	$\boxed{\textbf{48.1}}$ (0.8)	83.8 (0.9)	83.3 (0.9)	62.1 (0.8)	67.8 (1.0)	70.7 (0.5)	69.3 (0.8)
		600	47.6 (0.5)	83.7 (0.4)	82.9 (0.5)	60.7 (1.4)	68.2 (0.7)	70.3 (0.3)	68.9 (0.6)
	Transformer Encoder + RPR	300	47.3 (0.4)	83.8 (0.4)	$\boxed{\textbf{84.2}}$ (0.5)	62.0 (1.4)	68.2 (N1.6)	72.0 (0.5)	69.6 (0.8)
	Transformer Encoder + PE	300	45.0 (0.7)	82.0 (0.6)	83.3 (0.7)	62.3 (2.4)	66.9 (0.8)	68.4 (0.8)	68.0 (1.0)
	1-Layer SSAN	300	47.2 (0.5)	83.9 (0.7)	83.6 (0.6)	62.1 (2.5)	68.7 (1.0)	70.2 (1.2)	69.3 (1.1)
	1-Layer SSAN + PE	300	45.0 (0.3)	82.9 (0.2)	80.7 (0.6)	62.6 (2.3)	67.8 (0.4)	69.1 (0.3)	68.0 (0.7)

Table 2: Modified Table 3 from [Barnes et al., 2017]. Test accuracy averages and standard deviations (in brackets) of 5 runs. The baseline results are taken from [Barnes et al., 2017]; the self-attention models results are ours. Best model for each dataset is given in $\boxed{\textbf{bold}}$.

feed-forward layer is applied followed by an output softmax layer. Here, dropout is applied to input embeddings as well as after the convolution layers.

The *LSTM*, *BiLSTM*, and *CNN* baselines are trained using ADAM [Kingma and Ba, 2014] with cross-entropy loss and mini-batches of size 32. Hidden layer dimension, dropout amount, and the number of training epochs are tuned on the validation set for each (model, input embedding, dataset) combination.

4.4 Self-Attention Architectures

We use *1-Layer SSAN + RPR* and *2-Layer SSAN + RPR* to compare the self-attention mechanism to the recurrence and convolution mechanisms in *LSTM*, *BiLSTM*, and *CNN* models. We compare these models using all word embeddings sizes.

Next, we explore the performance of a modified *Transformer Encoder* described in 3. We do this to determine if a more complex architecture that utilized multi-head attention is more beneficial.

Finally, we compare the performance of using positional encodings (+*PE*) and relative positional

Model	# of Parameters	GPU VRAM Usage (Mb)	Training Time (s)	Inference Time (s)
LSTM	722,705	419Mb	235.9s	7.6s
BiLSTM	1,445,405	547Mb	416.0s	12.7s
CNN	83,714	986Mb	21.1s	0.85s
1-Layer SSAN + RPR	465,600	381Mb	64.6s	8.9s
1-Layer SSAN + PE	453,000	381Mb	58.1s	8.5s
2-Layer SSAN + RPR	839,400	509Mb	70.3s	9.3s
Transformer + RPR	1,177,920	510Mb	78.2s	9.7s

Table 3: Neural networks architecture characteristics. A comparison of number of learnable parameters, GPU VRAM usage (in megabytes) during training, as well as training and inference times (in seconds).

representations (+*RPR*) for the *Transformer Encoder* and *1-Layer-SSAN* architectures. We also test *1-Layer SSAN* without using any positional information techniques.

For the self-attention networks, we simplify the training process to only tune one parameter and apply the same process to all models. Only the learning rate is tuned for every (model, input embedding) pair. We fix the number of batches to train for to 100,000 and pick the model with highest validation accuracy. Each batch is constructed by randomly sampling the training set. Model dimensionality d_{model} is fixed to being the same as the input word embeddings. Learning rate is tuned based on the size of d_{model}. For d_{model} dimensions [50, 100, 200, 300, 600] we use learning rates of [0.15, 0.125, 0.1, 0.1, 0.05] respectively, because the larger d_{model} models tend to over-fit faster. Dropout of 0.7 is applied to all models, and the ADADELTA [Zeiler, 2012] optimizer is used with cross-entropy loss.

5 Analysis

Table 2 contains the summary of all the experimental results. For all neural network models we report mean test accuracy of five runs as well as the standard deviations. Macro-Avg results are the average accuracy of a model across all datasets. We focus our discussion on the Macro-Avg column as it demonstrates the models general performance for sentiment analysis.

Our results show general better performance for self-attention networks in comparison to *LSTM*, *BiLSTM* and *CNN* models. Using the same word embedding, all of the self-attention models receive higher Macro-Avg accuracy than all baseline models. *1-Layer-SSAN+RPR* models generally perform the best for all (input embeddings, dataset) combinations, and getting top scores for five out of six datasets. *Transformer Encoder+RPR* also performs comparatively well across all datasets, and achieves top accuracy for the *OpeNER* dataset.

Using *2-Layer-SSAN+RPR* does not yield better performance results compared to *1-Layer-SSAN+RPR*. We believe that one self-attention layer is sufficient as the datasets that we have tested on were relatively small. This is reinforced by the results we see from *Transformer Encoder + RPR* since it achieves similar accuracy as *2-Layer-SSAN+RPR* and *1-Layer-SSAN+RPR* while having greater architectural complexity and more trainable parameters, see Table 3.

Using relative positional representations for *1-Layer-SSAN+RPR* increases the Macro-Avg accuracy by 2.8% compared to using positional encodings for *1-Layer-SSAN+PE*, and by 0.9% compared to using no positional information at all (*1-Layer-SSAN*). Interestingly enough, we observe that using no positional information performs better than using positional encodings. This could be attributed once again to small dataset size, as [Vaswani et al., 2017] successfully used positional encodings for larger MT datasets.

Another observation is that SenTube dataset trials achieve a low accuracy despite having binary classes. This is unexpected as generally with a low number of classes it is easier to train on the dataset and achieve higher accuracy. We suspect that this is because SenTube contains longer sentences and very low word embedding coverage. Despite this, SSANs perform relatively well on the SenTube-A dataset, which suggests that they are superior at capturing long-range dependencies compared to other models.

Smaller d_{model} *SSAN* models perform worse for lower dimension input embeddings on *SST-fine*, *SST-binary* and *OpeNER* datasets while still performing well on *SenTube* and *SemEval*. This is caused by the limitations of our training process where we forced the network d_{model} to be same size as the input word embeddings and use the same learning rate for all datasets. We found that working with smaller dimensions of d_{model} the learning rate needed to be tuned individually for some datasets. For example, using a learning of 0.15 for 50D models would work well for *SenTube* and *SemEval*, but would under-fit for *SST-fine*, *SST-binary* and *OpeNER* datasets. We decided to not modify the training process for the smaller input embeddings in order to keep our training process simplified.

5.1 Model Characteristics

Here we compare training and test efficiency, memory consumption and number of trainable parameters for every model. For all models, we use the *SST-fine* dataset, hidden dimension size of 300, Google 300D embeddings, batch sizes of 32 for both training and inference, and the ADAM optimizer [Kingma and Ba, 2014]. The *Training Time* test is the average time it takes every model to train on 10 epochs of the SST-fine train set (2670 batches of size 32). The *Inference Time* test is the average time it takes a model to produce predictions for the validation set 10 times (344 batches of size 32). Table 3 contains the summary of model characteristics. The GPU VRAM usage is the amount of GPU video memory that is used during training.

CNN has the lowest number of parameters but consumes the most GPU memory. It also has the shortest training and inference time, which we attributed to the low number of parameters.

Using relative position representations compared to positional encoding for *1-Layer-SSAN* increases the number of trainable parameters by only 2.7%, training time by 11.2%, and inference time by 4.7%. These findings are similar to what [Shaw et al., 2018] reported.

BiLSTM has double the number of parameters as well as near double training and inference times compared to *LSTM*. This is reasonable due to the nature of the architecture being two *LSTM* layers. Much like *BiLSTM*, going from *1-Layer-SSAN* to *2-Layer-SSAN* doubles the number of trainable parameters. However, the training and inference times only increase by 20.1% and 9.4% respectively. This demonstrates the efficiency of the self-attention mechanism due to it utilizing only matrix multiply operations, for which GPUs are highly-optimized.

We also observe that self-attention models are faster to train than *LSTM* by about 3.4 times, and 5.9 times for *BiLSTM*. However, inference times are slower than *LSTM* by 15.5% and faster than *BiLSTM* by 41%.

6 Conclusion

In this paper we focused on demonstrating that self-attention networks achieve better accuracy than previous state-of-the-art techniques on six datasets. In our experiments, multiple *SSAN* networks performed better than *CNN* and *RNN* architectures; Self-attention architecture resulted in higher accuracy than *LSTMs* while having 35% fewer parameters and shorter training time by a factor of 3.5. Additionally, we showed that SSANs achieved higher accuracy on the *SenTube* datasets, which suggests they are also better at capturing long-term dependencies than *RNNs* and *CNNs*. Finally, we reported that using relative positional representation is superior to both using positional encodings, as well as not incorporating any positional information at all. Using relative positional representations for self-attention architectures resulted in higher accuracy with negligible impact on model training and inference efficiency.

For future work, we plan to extend the *SSAN* networks proposed to achieve state-of-the-art results on the complete *SST* dataset. We are also interested to see the behaviour of the models explored in this work on much larger datasets, we hypothesize that stacked multi-head self-attention architectures will perform significantly better than *RNN* and *CNN* counterparts, all while remaining more efficient at training and inference.

Acknowledgments

We thank the anonymous reviewers for their insightful suggestions.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv:1409.0473*, 2014.

Jeremy Barnes, Roman Klinger, and Sabine Schulte im Walde. Assessing state-of-the-art sentiment models on state-of-the-art sentiment datasets. In *Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, Copenhagen, Denmark, 2017.

Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *CoRR*, abs/1601.06733, 2016. URL http://arxiv.org/abs/1601.06733.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111. Association for Computational Linguistics, 2014. doi: 10.3115/v1/W14-4012. URL http://www.aclweb.org/anthology/W14-4012.

François Chollet et al. Keras. https://keras.io, 2015.

Michał Daniluk, Tim Rocktäschel, Johannes Welbl, and Sebastian Riedel. Frustratingly short attention spans in neural language modeling. *arXiv preprint arXiv:1702.04521*, 2017.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*, 2017.

Felix Greaves, Daniel Ramirez-Cano, Christopher Millett, Ara Darzi, and Liam Donaldson. Harnessing the cloud of patient experience: using social media to detect poor quality healthcare. *BMJ Qual Saf*, 22 (3):251–255, 2013a.

Felix Greaves, Daniel Ramirez-Cano, Christopher Millett, Ara Darzi, and Liam Donaldson. Use of sentiment analysis for capturing patient experience from free-text comments posted online. *Journal of medical Internet research*, 15(11), 2013b.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Bernard J Jansen, Mimi Zhang, Kate Sobel, and Abdur Chowdury. Twitter power: Tweets as electronic word of mouth. *Journal of the American society for information science and technology*, 60(11):2169–2188, 2009.

Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751. Association for Computational Linguistics, 2014. doi: 10.3115/v1/D14-1181. URL http://www.aclweb.org/anthology/D14-1181.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

Filippos Kokkinos and Alexandros Potamianos. Structural attention neural networks for improved sentiment analysis. *arXiv preprint arXiv:1701.01811*, 2017.

Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018.

Jiasen Lu, Jianwei Yang, Dhruv Batra, and Devi Parikh. Hierarchical question-image co-attention for visual question answering. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 289–297. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/6202-hierarchical-question-image-co-attention-for-visual-question-answering.pdf.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Mika V. Mäntylä, Daniel Graziotin, and Miikka Kuutila. The evolution of sentiment analysis—a review of research topics, venues, and top cited papers. *Computer Science Review*, 27:16 – 32, 2018. ISSN 1574-0137. doi: https://doi.org/10.1016/j.cosrev.2017.10.002. URL http://www.sciencedirect.com/science/article/pii/S1574013717300606.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

Preslav Nakov, Zornitsa Kozareva, Alan Ritter, Sara Rosenthal, Veselin Stoyanov, and Theresa Wilson. Semeval-2013 task 2: Sentiment analysis in twitter, 2013.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: Sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 79–86, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1118693.1118704. URL https://doi.org/10.3115/1118693.1118704.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, and Alexander Ku. Image transformer. *arXiv preprint arXiv:1802.05751*, 2018.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

Daniel Povey, Hossein Hadian, Pegah Ghahremani, Ke Li, and Sanjeev Khudanpur. A time-restricted self-attention layer for asr. 2017.

Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *CoRR*, abs/1803.02155, 2018.

Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. Disan: Directional self-attention network for rnn/cnn-free language understanding. In *AAAI Conference on Artificial Intelligence*, 2018.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. Linguistically-informed self-attention for semantic role labeling. *CoRR*, abs/1804.08199, 2018.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

Olga Uryupina, Barbara Plank, Aliaksei Severyn, Agata Rotondi, and Alessandro Moschitti. Sentube: A corpus for sentiment analysis on youtube social media. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation*. European Language Resources Association, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

Rodrigo Agerri y Montse Cuadros y Seán Gaines y German Rigau. Opener: Open polarity enhanced named entity recognition. *Procesamiento del Lenguaje Natural*, 51(0):215–218, 2013. ISSN 1989-7553. URL http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/4891.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. *arXiv preprint arXiv:1805.08318*, 2018.

Yi Zhou, Junying Zhou, Lu Liu, Jiangtao Feng, Haoyuan Peng, and Xiaoqing Zheng. Rnn-based sequence-preserved attention for dependency parsing. 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17176.