Learned or Lost?
Reproducing *Learned in Translation: Contextualized Word Vectors*

Report by Adam Hare
Project completed by Adam Hare, Lisa Schut, Hans Hanley, and Aditya Agarwal

1 Introduction & Motivation

In this project we attempt to reproduce *Learned in Translation: Contextualized Word Vectors* [McCann et al., 2017]. This paper is interesting because transfer learning is a field with a lot of promise and potential applications. If useful, CoVe could save countless training time and become a standard part of many machine learning models, much like GloVe is today [Pennington et al., 2014]. This paper includes many elements from other papers on the list and topics covered in class, such as attention, question answering, and co-attention networks. This provided us with an opportunity to learn about these other techniques and research. Finally, we believed this paper would be feasible in terms of computational resources. Many of the datasets are somewhat small and did not take long to train on. Choosing this paper and these experiments allowed us to spend more time doing expansions in addition to recreating their original work.

2 CoVe Summary

The idea behind contextualized word vectors (CoVe) is straightforward. Recognizing the advantages that transfer learning provides to a variety of applications, [McCann et al., 2017] decided to use a machine translation LSTM-based model (MT-LSTM) to gather contextualized representations of words. This follows the traditional assumption used by machine translation models [Sutskever et al., 2014] that information about the meaning of a word can be at partially inferred by the meaning of the words around it. Providing these embeddings alongside global embeddings (i.e. GloVe) should boost performance in a variety of NLP tasks such as sentiment analysis, question classification, entailment, and question answering.

For this paper, we focused on reproducing the original CoVe embeddings using the MT-LSTM and the sentiment classification task using the biattentive classification network (BCN) as outlined in [McCann et al., 2017]. I will now briefly summarize both the MT-LSTM and BCN models using the equations and diagrams provided in [McCann et al., 2017].

2.1 MT-LSTM

The MT-LSTM is the machine translation model used to generate the CoVe embeddings. This model consists of an encoder and a decoder. Once the model is trained on a dataset the weights of the encoder are saved. CoVe embeddings are then generated by passing the sentence to embed as input to the encoder and using the encoder's output as the embedding.

The MT-LSTM model [Klein et al., 2017] takes as input a pair of sentences, one in the source language (w^s) and one in the target language (w^t). w^s is converted into a sequence of GloVe and used as input to the MT-LSTM encoder. This encoder is a two-layer BiLSTM [Graves and Schmidhuber, 2005] which outputs a sequence of hidden states h.

This output is then passed to the decoder, consisting of a unidirectional, two-layer LSTM. For our implementation, at the first time step we initialized these layers with the hidden states of the final layers of the encoder. In subsequent steps, they are initialized with their hidden state in the previous time step.

The decoder is given as input the previous target word, embedded as an initially random word vector, concatenated with a "context-adjusted hidden state $\tilde{h}_{t-1}$" [McCann et al., 2017]. The output of this, called h_t^{dec}, is then used as part of the attention steps. Specifically we create a matrix H which is just h "stacked

along the time dimension" [McCann et al., 2017]. We get the attention weights from:

$$\alpha_t = \text{softmax}(H(W_1 h_t^{dec} + b_1)) \tag{1}$$

where W_1 and b_1 are learned parameters. This is then given as input along with h_t^{dec} to a tanh layer:

$$\tilde{h}_t = \tanh(W_2[H^T \alpha_t; h_t^{dec}] + b_2) \tag{2}$$

Where $[H^T \alpha_t;\ h_t^{dec}]$ denotes the concatenation of $H^T \alpha_t$ and h_t^{dec}. Using $\tilde{h}_t$ as input to a dense layer with softmax activation yields the output distribution.

After training, a CoVe embedding is generated by representing each word in sentence w with its GloVe embedding, passing this as input to the encoder, and taking the result as CoVe(w). Each word in CoVe(w) is 600 dimensions as all LSTMs have a hidden size of 300 and so the output of a Bi-LSTM is 600. This is then concatenated with the original GloVe embedding to form:

$$\tilde{w} = [\text{GloVe}(w); \text{CoVe}(w)] \tag{3}$$

$\tilde{w}$ is a 900 dimensional encoding for each word in w.[1]

2.2 BCN

The biattentive classification network (BCN) is a variant of the dynamic coattention network [Xiong et al., 2016] and is used to test the effectiveness of CoVe in downstream tasks. As it requires two sentences as input, for sentiment classification the input sequence is duplicated and passed twice. The diagram provided in [McCann et al., 2017] is below:

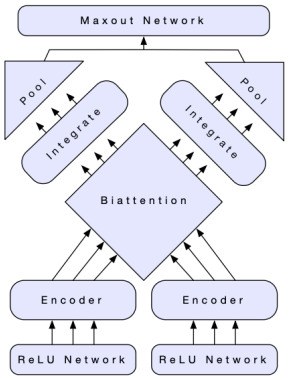

Figure 1: BCN Model

Two input sequences w^x, w^y are transformed according to Equation 3 into sequences $\tilde{w}^x, \tilde{w}^y$.[2] These are then passed to the 'ReLU Network' blocks consisting of a feed-forward network with ReLU activation. The results from this network are then passed to the 'Encoder' block. This is not the same encoder as the MT-LSTM; it is a single BiLSTM layer with sigmoid activation. Together these two blocks transform $\tilde{w}^x, \tilde{w}^y$ to "task specific representations" x, y [McCann et al., 2017]. These blocks transform the general GloVe/CoVe embeddings to embeddings that are more useful for the task of the BCN and are similar to the task specific RNNs in [Peters et al., 2017] and ELMo [Peters et al., 2018].

There is a discrepancy in this part of the paper. Based on the wording, it seems to imply that the same feed-forward network and BiLSTM were used for both $\tilde{w}^x$ and $\tilde{w}^y$. However, the diagram seems to indicate that there

[1] 300 for GloVe and 600 for CoVe

[2] for sentiment classification, $\tilde{w}^x = \tilde{w}^y$

are two separate feed-forward networks and two BiLSTMs, one for each of $\tilde{w}^x$ and $\tilde{w}^y$. As [McCann et al., 2017] did not provide their BCN code, we were unable to resolve this and it also appears to be a concern in other implementations.[3] Empirically we found that in most cases a single feed-forward network and a single BiLSTM worked better, although conceptually this should be a subset of the models that can be learned using two separate networks and BiLSTMs and so is strictly less powerful. Our implementation uses a single network and BCN, which means that $x = y$ in the above equations and the Biattention step is effectively a form of self-attention [McCann et al., 2017].

After obtaining the task specific representations, x and y are transformed into matrices X and Y by stacking along the time axis [McCann et al., 2017]. These are then used in the 'Biattention' block [Seo et al., 2016] [Xiong et al., 2016]. This block first calculates $A = XY^T$ (the affinity matrix) and then $A_x = \text{softmax}(A), A_y = \text{softmax}(A^T)$.[4] These serve as attention weights. The biattention block then outputs two context summaries $C_x = A_x^T X, C_y = A_y^T Y$ (Equation 10 in [McCann et al., 2017]). This applies the appropriate attention weights to the matrices X and Y.

After the biattention block, we concatenate $X, X - C_y$, and the element-wise product of X and C_y. X is included to ensure that the biattention representation did not cause any loss of information. $X - C_y$ conveys an explicit difference between the original representation and the context summary [McCann et al., 2017]. The element-wise product weights the entries of the original representation based on their importance in the context summary. An analogous concatenation is performed with Y and C_x and these are given as inputs to the two "Integrate" blocks, each consisting of a one-layer BiLSTM.[5] This is compressed to the following equations:[6]

$$X_{|y} = \text{biLSTM}([X; X - C_y; X \odot C_y]) \qquad Y_{|x} = \text{biLSTM}([Y; Y - C_x; Y \odot C_x]) \tag{4}$$

We now have a matrix $X_{|y}$ representing the original representation of X conditioned on the representation of Y and the analogous $Y_{|x}$. These are both given as input to the pooling blocks. [McCann et al., 2017] performs four types of pooling on $X_{|y}$ and $Y_{|x}$: max pooling, mean pooling, min pooling, and self-attentive pooling. The first three are common ways of generating summaries, whereas the final gets weighted summations of each sequence through the following equations:[7]

$$\beta_x = \text{softmax}(X_{|y} v_1 + d_1) \qquad \beta_y = \text{softmax}(Y_{|x} v_2 + d_2) \tag{5}$$

$$x_{self} = X_{|y}^T \beta_x \qquad y_{self} = Y_{|x}^T \beta_y \tag{6}$$

β_x and β_y serve as attention weights for each column in $X_{|y}$ and $Y_{|x}$ and x_{self} and y_{self} as summaries of each word in the sequence. We then concatenate all eight pooling results and pass them as input to the maxout network block [Goodfellow et al., 2013]. This network is batch-normalized with three layers. Its final layer is a dense layer with a softmax activation, which outputs a probability distribution over the classes [McCann et al., 2017].

In addition to the CoVe encodings proposed in Equation 3, [McCann et al., 2017] also tested on random word embeddings, GloVe embeddings, character n-gram embeddings [Hashimoto et al., 2016], and a combination of GloVe, CoVe, and Char embeddings. They found that generally the embeddings from the CoVe model trained on the most data (CoVe-L) combined with GloVe and Char performed the best [McCann et al., 2017].

[3] `https://github.com/markmenezes11/COMPM091/blob/master/CoVe-BCN/model.py#L77`

[4] Equation 9 in [McCann et al., 2017]

[5] Note that the left integrate block only gets the results of X and C_y and the right integrate block gets only the results of Y and C_x

[6] Equations 11 and 12 in [McCann et al., 2017]

[7] Equations 13 and 14 in [McCann et al., 2017]

3 Reproducibility

The authors of [McCann et al., 2017] propose many different applications of CoVe and implement several of them in the paper. We decided to reproduce their results of training the MT-LSTM on the WMT 2016 dataset [Specia et al., 2016] and testing the BCN on SST-2 and SST-5.[8] This dataset was chosen because it is a manageable size to run on Google Colab. This allowed us to maximize compute time, perform more experiments, and try more extensions than if we had chosen one of the larger sets. The trade-off, of course, was performance, although from their results training CoVe on a larger dataset did not appear to make much of a difference for sentiment classification. We chose to do sentiment analysis with SST-2 and SST-5 for similar reasons. They both make use of the BCN architecture, rather than SQuAD, which requires a DCN instead. These are commonly used datasets and small enough to make our analysis quick (Tables 1 and 2).

While all general results were performed with the small dataset, we also ran our MT-LSTM with the IWLST dataset [Cettolo et al., 2016] which is a subset of CoVe-M in [McCann et al., 2017]. I will refer to this as MT-LSTM-M. We unfortunately did not have the resources to reproduce CoVe-L, which uses seven million sentence pairs.

Name	Training	Validation	Testing	Source
WMT 2016 (EN/DE)	29,000	1014	1000	[Specia et al., 2016]
Europarl v7 (FR/EN)	100,000	2000	2000	[Popel and Bojar, 2018a]
IWLST (DE/EN)	98,132	887	1565	[Cettolo et al., 2016]

Table 1: Encoder Datasets

Name	Training	Validation	Testing	Source
SST-2	6228	692	1821	[Socher et al., 2013]
SST-5	9017	1127	1128	[Socher et al., 2013]

Table 2: Sentiment Analysis Datasets

Once we chose which parts to reproduce, we decided to code as a group the original MT-LSTM encoder and decoder as well as the BCN. After that, we split up to work on different extensions. We decided to implement both of these using `keras` and `tensorflow`. We used nmt_with_attention.ipynb[9] [Luong et al., 2017] from `tensorflow` as inspiration and consulted the code provided by [McCann et al., 2017].[10] We majorly overhauled the NMT code to use our more complicated encoders. We also updated to `tensorflow v2` to prevent using eager execution, as this was causing out of memory errors. We share roughly the same high-level layout (i.e. having an encoder model and decoder model as well as how we train and evaluate) with [Luong et al., 2017], but implemented our own encoder and decoder. We also had to change the loss function and how we passed information from the encoder to the decoder because we have more hidden states than in that paper.

For the BCN, we used a `keras Serial` model. We mainly followed the conceptual outline for the BCN provided in [McCann et al., 2017] and occasionally consulted a Cove implementation[11] and AllenNLP [Gardner et al., 2017]. These were used mostly to check parameter sizes and to confirm some implementation details, but overall the BCN code is our own and not reliant on any of those implementations. Our code can be found at `https://github.com/adi2103/AML-CoVe`.

We used the weights provided by [McCann et al., 2017] to recreate the exact CoVe encodings used in the paper. We called this result the "BASELINE" throughout. This baseline model will allow us to control somewhat for problems with the BCN - if our MT-LSTM results are close to those in the CoVe paper, then they should give a similar performance to the BASELINE weights regardless of the BCN. We were unable to

[8] Training the MT-LSTM on WMT 2016 generates CoVe-S in [McCann et al., 2017]

[9] `https://colab.research.google.com/github/tensorflow/tensorflow/blob/master/tensorflow/contrib/eager/python/examples/nmt_with_attention/nmt_with_attention.ipynb`

[10] `https://github.com/salesforce/cove`

[11] `https://github.com/markmenezes11/COMPM091`; McCann pointed us towards this repository in an email

	SST-2			SST-5		
Name of Model	CoVe Val	Our Val	Our Test	CoVe Val	Our Val	Our Test
Random	84.2	77.5	76.3	48.6	34.4	34.0
GloVe	88.4	81.5	78.0	53.5	**46.8**	45.9
GloVe + Char	90.1	**82.8**	**83.5**	52.2	43.8	45.6
GloVe + BASELINE	**91.1**	81.1	80.3	**54.5**	45.2	**47.1**
GloVe + MT-LSTM	89.0	81.8	82.7	54.0	45.9	46.4
GloVe + MT-LSTM + Char	-	81.2	83.2	-	44.7	42.1
GloVe + MT-LSTM-M	-	80.8	81.9	-	37.8	38.2

Table 4: BCN validation and test accuracy. CoVe only reports test accuracy for GloVe + CoVe-L + Char which is not comparable to any of our models, so CoVe test accuracy is omitted. - represents results not reported by [McCann et al., 2017]

find out which dataset this was trained with but assume it is CoVe-L [McCann et al., 2017]. This means it was trained on a dataset of 7 million sentence pairs compared to just 29,000 we are using and so should outperform our implementation.

We chose the same hyper-parameters for our MT-LSTM as are mentioned in [McCann et al., 2017] (Table 3). The maximum sentence length of 35 corresponds to the maximum sentence length we found in the dataset. The size of the maxout layers depend on the size of the encoding used. The sizes in the table correspond to the CoVe encoding. We chose a dropout of 0.3 for the BCN because it was one of those mentioned in [McCann et al., 2017] and it performed the best in our initial trials. We also experimented with using SGD and different learning rates, but found that Adam [Kingma and Ba, 2014] with the default parameters performed best.[12]

Part of Model	Name of Hyper-parameter	Value
MT-LSTM	Dropout	0.2
	Encoder Hidden Units	300
	Decoder Hidden Units	600
	Batch Size	128
BCN	Dropout	0.3
	Dense Layer Units	300
	LSTM Hidden Units	300
	Maxout Layer 1	1200
	Maxout Layer 2	600
	Batch Size	256
Both	Maximum Sentence Length	35
	Recurrent Dropout	0
	Optimizer	Adam
	Learning Rate	0.001

Table 3: Reproducibility Hyper-parameters

We ran our MT-LSTM three times, each with a different seed, and chose the one with the best BLEU as our final model. The results of this can be found in the extensions section in Table 7. We used early stopping on validation loss, where we stopped training the model if the validation loss did not improve after two epochs.

After choosing the best MT-LSTM, we used it as an encoder to generate CoVe weights and then performed sentiment analysis on the SST-2 and SST-5 datasets. The results are in Table 4 and are reported as accuracy on the validation and test sets. BASELINE, MT-LSTM, and MT-LSTM-M all correspond to the CoVe embeddings generated by these models, as in Equation 3.

In general, we were not able to replicate the high validation accuracy reported in [McCann et al., 2017]; even our random model is 7% below the reported validation accuracy. The overall trend is somewhat similar, in that adding GloVe has the highest performance boost. The BASELINE results under-performed our own on

[12]These parameters were agreed upon by the group for all tests with the BCN but discussions after results came in indicated that how the BCN was trained varied somewhat from member to member.

SST-2 and only slightly over-performed it on SST-5. This may indicate that the BCN was unable to extract the additional information in the BASELINE results or that our MT-LSTM results are over-performing relative to CoVe-S in [McCann et al., 2017]. GloVe + Char seems to perform the best on SST-2, which may indicate that CoVe is not having an effect here.

It is clear that GloVe and Char both help to boost performance significantly above Random, but adding CoVe does not result in significant improvements beyond this. This seems more true for SST-2 than for SST-5, as BASELINE and our MT-LSTM both achieved better test accuracy on this dataset than the implementations without CoVe. Adding Char boosts test performance on SST-2 but hurts accuracy on SST-5, which is consistent with [McCann et al., 2017].

4 Extensions

After completing the main MT-LSTM and BCN models, we broke into smaller groups to complete extensions. Here I will summarize the experiments and results of the two candidates I did not do extensions with and then go more in depth on the extensions that I implemented with Lisa Schut.

4.1 Other Group Member Extensions

One extension was to train the MT-LSTM on a different dataset and test it against SST-2 and SST-5. The candidate used [Popel and Bojar, 2018a], a French to English dataset (Table 1). All hyper-parameters are the same as in the standard MT-LSTM. The results of this experiment can be seen in Tables 7 and 8 under the model name MT-LSTM-FR. It appears that this dataset results in significant gains for SST-2 validation and test accuracy over the MT-LSTM models trained on the WMT dataset. Part of this is likely due to its larger size (100K vs. 29K), although this would not explain why it outperformed the MT-LSTM-M. It is also possible that the encodings learned in translating English to French are somehow more useful in sentiment classification than those learned in translating English to German. This would be somewhat supported by [Vaswani et al., 2017], which also found significantly higher performance on English to French than on English to German, but more testing would be required to draw a firm conclusion.

Another extension tested different dropout rates for the MT-LSTM on the WMT dataset and experimented with adding recurrent dropout [Gal and Ghahramani, 2016]. This was performed via a grid search over (0.1, 0.2, 0.3, 0.4, 0.5) trained on 3000 sentence pairs from the SST-2 training set for both dropout and recurrent dropout; the optimal parameters were found to be 0.2 for dropout and 0.1 for recurrent dropout. The results of training this on the BCN can be found in Tables 7 and 8 under the model Tuned MT-LSTM. It appeared to assist slightly in validation accuracy but to harm test accuracy on both SST-2 and SST-5.

ELMo embeddings [Peters et al., 2018] were also tested. ELMo is an expansion upon CoVe as it generates deep contextualized word vectors. Rather than use only the final output layer of the BiLSTM for the encoding, as in [McCann et al., 2017], ELMo uses a linear combination of all internal layers of a bidirectional language model.[13] The exact weights of the linear combination are learned specific to the task, allowing the downstream model to choose the parts of the language model that are most useful to it. ELMo also benefits from using character-level convolutions. ELMo has been shown to outperform CoVe, see [Peters et al., 2018]. Here, ELMo encodings were used in place of CoVe and passed directly to the same BCN as before. As we can see in Table 8, ELMo generally performed worse than random on SST-2 and significantly worse than the MT-LSTM on SST-5. This is not consistent with [Peters et al., 2018], which reported a 1% absolute increase in accuracy when they replaced CoVe with ELMo. I this is because the implementation used here learns the task-specific weightings in the same way as in [McCann et al., 2017]. However, Equation 1 in [Peters et al., 2018] specifies a different way of learning-task specific weighting and encourages layer normalization. I believe this discrepancy could have a significant effect on the results.

[13] Each language model consists of a forward and backward LSTM

Other group members also experimented with using a CNN instead of the BCN for sentiment classification. Two variants of this were tried: the first with one convolutional layer [Kim, 2014] and the second with three [Ouyang et al., 2015]. As in lecture, the convolutional layers are then followed by max pooling and a fully-connected layer with dropout and softmax activation. All hyper-parameters, including dropout, kernel size, and number of filters followed exactly from the provided papers. The results are in Table 5.

It appears that in general the results here are comparable to those from the BCN. Some models saw an increase of one or two points in accuracy, but not much substantial. It appears that three layers boost performance in general over only one, but this is to be expected as it is a larger model.

	SST-2		SST-5	
Name of Model	Validation Loss	Test Loss	Validation Loss	Test Loss
1 Layer CNN				
Random	66.46	63.29	34.31	31.77
GloVe	80.64	80.20	43.61	39.66
GloVe + Char	81.94	78.76	44.06	40.37
GloVe + BASELINE	80.06	80.35	44.50	41.53
GloVe + MT-LSTM	79.91	**81.79**	**44.86**	43.03
GloVe + MT-LSTM + Char	80.92	79.19	44.41	**43.48**
ELMO	77.02	73.94	43.44	41.88
GloVe + MT-LSTM-M	74.24	78.90	39.04	37.94
GloVe + MT-LSTM-FR	**82.51**	79.34	32.28	33.61
3 Layer CNN				
Random	62.86	65.32	31.29	30.61
GloVe	80.78	80.49	42.20	40.20
GloVe + Char	81.50	81.65	43.09	43.48
GloVe + BASELINE	82.36	82.36	43.79	44.90
GloVe + MT-LSTM	80.49	82.08	42.20	44.54
GloVe + MT-LSTM + Char	80.92	80.92	44.24	**46.68**
ELMO	79.48	76.30	45.92	44.63
GloVe + MT-LSTM-M	80.06	80.35	40.73	42.64
GloVe + MT-LSTM-FR	**83.24**	**85.12**	**49.02**	44.51

Table 5: Sentiment CNN

4.2 Our Extensions

Lisa and I first did some exploratory data analysis. We found that GloVe and CoVe entries are roughly normally distributed with mean 0. We saw GloVe had a larger range whereas CoVe entries mostly ranged from -0.25 to 0.25. No CoVe entries were exactly 0. The standard deviations across embedding dimensions tend to follow a chi-squared distribution. We did TSNE plots for some sentences, which showed that words with many meanings often did not have consistent CoVe encodings, while those that are not context-dependent tended to have similar CoVe encodings. See Appendix C for more details and plots.

Lisa and I decided to expand the [McCann et al., 2017] translation model by experimenting with new encoders. We were especially curious about these results as non-sequential models can be trained more efficiently than recurrent models. We also suspected that different models could learn different things in their encodings, which could affect downstream performance. I will briefly outline the concepts behind both of our new encoders, then comment on implementation, and finally compare all results. We also did some exploratory data analysis, see Appendix C.

A general note about our encoders: we do not pass any hidden states from the encoder to the decoder. Originally we had passed created proxies for the encoder hidden states using pooling at different points to create output of the appropriate sizes. We saw that this improved performance around 5 to 10% during training. We ultimately decided not to use these as they have no intuitive theoretical justification. Additionally, we wanted our encoder-decoder to learn mostly from the encodings, not from the passed hidden states. We corresponded with McCann over email who said that although passing the hidden states from the encoder was common

practice for the MT-LSTM, he did not see a significant difference between doing that, passing zeroes, or passing an average of encoder hidden states in his experiments. We initialize the decoder hidden states with all zeroes.

4.2.1 Convolutional Encoder (CNN)

Our first new encoder uses a CNN instead of an LSTM. Our implementation does not follow directly from any specific paper, but was inspired by several such as [Kalchbrenner et al., 2016], which uses convolutional layers for both the encoder and decoder, and [Gehring et al., 2016], which uses a CNN encoder and an LSTM decoder.

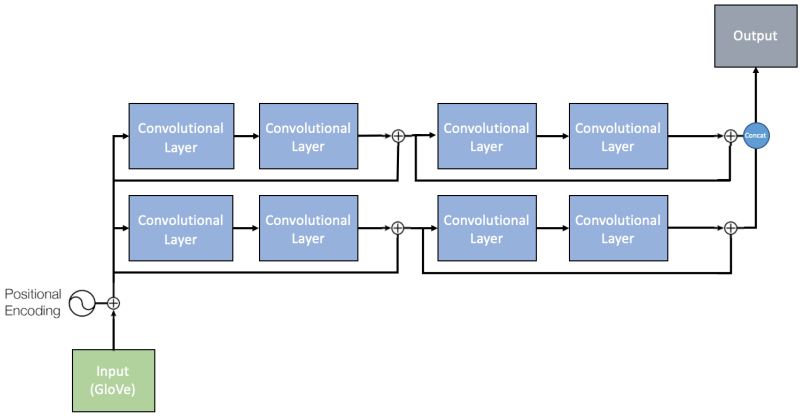

Figure 2: CNN Encoder Model

See Figure 2 for a diagram. We also used masking and skip connections to ease training. Inputs are re-scaled immediately before and after the skip connections. We also employ dilation starting with 1 and doubling in each subsequent convolutional layer as in [Van Den Oord et al., 2016]. Dilation allows the filters of later convolutional layers to consider non-adjacent words. This greatly increases the effective window size without costing much additional computation.

The CNN encoder makes use of positional encodings [Gehring et al., 2017] [Gehring et al., 2016]. These encodings contain information about where in the sentence a word is and gives the CNN a sense of what part of the sentence it is seeing. Positional encodings were not needed in the MT-LSTM due to its recurrent nature.

4.2.2 Transformer (TRNS)

The second new encoder uses two encoder layers from the Transformer model [Vaswani et al., 2017] and the MT-LSTM decoder from [McCann et al., 2017]. We decided not to use a Transformer decoder to make our results more comparable to those from the MT-LSTM. You can see a diagram of the encoder, a modified version of the one from [Vaswani et al., 2017] in Figure 3. We followed directly the equations from this paper, differing only in the number of heads and other hyper-parameters (Section 4.2.4). We decided to use two encoder layers so that the overall TRNS model with the decoder would be comparable in size to the CNN and MT-LSTM (Appendix A).

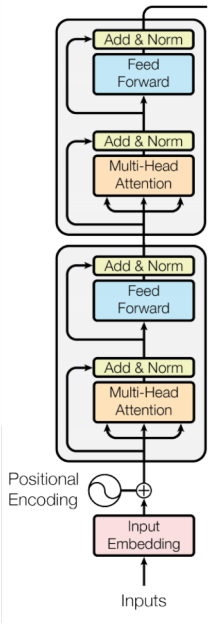

Figure 3: TRNS Encoder Model

The basic Transformer model is an encoder-decoder, with the encoder comprised of several identical layers, each with a multi-headed self attention step and a fully connected feed-forward network [Vaswani et al., 2017]. The attention mechanism maps a query and a key-value pair to an output, weighting the value vector based on the "compatibility" of the key and query vectors [Vaswani et al., 2017]. The Transformer model extends this to multi-headed attention, where several attention mechanisms are run in parallel and concatenated at the end. This improves resolution as the model is able to learn several representations rather than a single average result.

[Vaswani et al., 2017] also make use of skip connections and positional embeddings, for the same reasons as above. Transformers regularize using dropout [Srivastava et al., 2014] and employ layer normalization [Lei Ba et al., 2016] to reduce training time.

4.2.3 Combinations

We experimented with combining different encodings. Specifically, we ran experiments where one input sentence to the BCN was GloVe + BASELINE and the other was GloVe + CNN or GloVe + TRNS. For these experiments, we used separate feed-forward networks and Bi-LSTMs on each side to learn task specific representations for each type of encoding. We hoped to see from these results if TRNS or CNN learn something *different* from CoVe in such a way that combining the two can boost performance.

4.2.4 Implementation Details

For the Transformer, we set the number of heads to six and d_k[14] to 100 in order to obtain a 600-dimensional encoding as in [McCann et al., 2017]. We used the Adam optimizer [Kingma and Ba, 2014] with the default learning rate and other parameters given in [Vaswani et al., 2017]. We experimented with annealing, gradient clipping, learning rate, and Stochastic Gradient Descent but none of these improved training. The hyper-parameters in Table 6 yielded the best results. Some tips for training were derived from [Popel and Bojar, 2018b].

For the CNN, we chose 300 hidden units for each convolutional layer to get a 600-dimensional encoding. We found that using two 300-unit channels as in Figure 2 performed better than one with 600 units because different non-linearities[15] could be learned. We found that using sums for skip connections performed better than concatenation. We experimented with dropout, the number of filters, batch size, kernel size, and the activation function. We also found that we got the best results by using the Adam optimizer until the loss

[14] the dimension of the key, value, and query vectors
[15] the tanh activations in each convolutional layer

Part of Model	Name of Hyper-parameter	Value
CNN	Dropout	0.3
	Filter Size	128, 256, **300**, 600
	Kernel Size	1, 2, 3, **4**
	Activation Function	ReLU, Softmax, Hard Softmax, **tanh**
	Batch Size	128
TRNS	Dropout	0.1, **0.2**, 0.3, 0.4, 0.5
	Number of Heads	6
	Key Dimension (d_k)	100
	Dense Layer Units	600
	Batch Size	128, 256, **512**
Both	Maximum Sentence Length	35
	Positional Encoding Size	300
	Final Encoding Dimension	600

Table 6: Extension Hyper-parameters. Comma separated values indicate hyper-parameters tried, the ones used are bolded.

Name of Model	Loss	Perplexity	BLEU
MT-LSTM	29.76	11.35	**32.17**
Tuned MT-LSTM	29.36	14.56	29.75
MT-LSTM-M	91.86	206.39	0.29
MT-LSTM-FR	68.88	18.06	9.57
CNN	39.98	38.31	0
TRNS	39.24	35.83	0

Table 7: Encoder Results

did not improve for two epochs, then switching to SGD with annealing and gradient clipping. We divided the learning rate by 10 each time validation loss did not improve until it reached 10^{-5}.

4.2.5 Results

See Table 7 for the training results across all tested encoders. The standard MT-LSTM appears to perform the best in terms of BLEU[16] score.[17] The CNN and TRNS models appeared to struggle with translation, which may have been a result of using the MT-LSTM decoder or an over-reliance on teacher forcing in the training phase. Specifically, they appear to suffer from mode collapse based on the first word they predict in the sentence; perhaps incorporating teacher forcing into testing would have improved these results. For non-MT-LSTM models, we generated the encodings from the model with the lowest test loss (categorical cross entropy).

[16]BLEU calculated using mult-bleu.perl script from Moses (`https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl`)

[17]If you pick the model with the lowest validation loss, the BLEU score drops to 29.75

Name of Model	SST-2 Validation	SST-2 Test	SST-5 Validation	SST-5 Test
Random	77.46	76.30	34.43	34.04
BASELINE	81.07	80.29	45.16	47.07
MT-LSTM	81.79	82.66	45.92	46.41
MT-LSTM-FR	**83.96**	**83.67**	47.20	46.45
Tuned MT-LSTM	82.65	76.64	45.31	44.61
ELMO	77.31	74.52	41.61	42.73
CNN	81.50	80.67	43.09	42.15
TRNS	80.35	67.98	**48.54**	45.48
BASELINE + CNN	82.08	82.54	44.19	42.82
BASELINE + TRNS	81.65	74.35	46.10	**48.80**

Table 8: BCN Accuracy: All Models Are Concatenated With GloVe

The results from the BCN on each of these encodings can be found in Table 8. In general, we found that the Transformer outperformed all other models on the SST-5 dataset. This suggests that it is learning something somewhat different from the MT-LSTM and CNN and is not unsurprising as Transformers are the basis for models such as BERT [Devlin et al., 2018]. It also points to a lack of a monotonic relation between translation performance and sentiment classification performance for various encoders in our results. We also saw that combining BASELINE with CNN and TRNS generally boosted results over each separately, which suggests that combining different encodings in this way could prove fruitful. The encoder trained on the French dataset seemed to perform better, which is consistent with the results from the Sentiment CNN.

5 Conclusions

Overall, we were not able to reproduce the CoVe results [McCann et al., 2017]. The most important factor in this was the BCN. This code was not provided directly in [McCann et al., 2017], and when we emailed the authors, they directed us to a repository[18] which also expressed confusion with regards to some of the details. [Peters et al., 2018] also notes that their BCN implementation "reached significantly lower validation accuracies in their experiments" than those reported by [McCann et al., 2017]. We found the BCN to be either unstable or slow to train and because we lacked information on how the models were trained in [McCann et al., 2017], for instance test loss and number of epochs, we were not able to better calibrate our model. Our implementation may also have differed, specifically with respect to learning task-specific weights and the size of the maxout network.[19] We also used a relatively large batch size (256 vs 100 in [Gardner et al., 2017][20]).

It is also possible that our MT-LSTM results did not align with those reported by [McCann et al., 2017]. Although they compared well to the BASELINE results, this could have been a result of a bad BCN. The BCN could have been able to learn somewhat better than random on CoVe embeddings, but not able to extract more information from different encodings. Because the BCN was so unstable for most of the training, it is hard to say that the differences in accuracy we saw are definitively a result of stronger encodings. This concern is somewhat alleviated by the BLEU score of our MT-LSTM: [McCann et al., 2017] reports 38.5 compared to our best 32.17. While different, these results are somewhat comparable, especially considering implementation differences[21] and so suggest our MT-LSTM is not the most serious problem. The performance of our MT-LSTM likely suffered because we did not use beam search[22] or masking,[23] as they were not mentioned in [McCann et al., 2017], but were used in the code they provided.

There were also a few problems with our data. We used the GloVe [Pennington et al., 2014] embeddings trained on 6 billion tokens (see dataprep.py on our GitHub), whereas [McCann et al., 2017] used embeddings trained on 840 billion tokens. These results were too big to run on Colab, but could have caused pervasive performance problems. GloVe is not only concatenated with CoVe for downstream tasks: the encoder-decoder learn encodings based on GloVe. If our GloVe embeddings contained less information, it may be that our encoder-decoders were just not able to learn as well. This would not explain why our Random result on the BCN performed about 7 percentage points worse on SST-2 than is reported in [McCann et al., 2017], as this result did not use GloVe at all. We also had issues using the proper train/test/validation split for SST-2 and SST-5. We had the correct training and test sets for SST-2, but had to take a subset of the training set for validation. For SST-5, we drew a new, random train/validation/test split for each seed. This limits the comparability of our results to those in [McCann et al., 2017]. We also use the `keras` tokenizer instead of Moses as in [McCann et al., 2017], although this is unlikely to have a major impact.

[18] `https://github.com/markmenezes11/COMPM091/blob/master/CoVe-BCN/model.py`

[19] [McCann et al., 2017] call for a "three-layer, batch-normalized maxout network." There was some confusion as to if this meant two maxout layers and one softmax layer or three maxout layers and a softmax. We went with the former, as did the BCN code provided to us, but we are uncertain about this. When [Peters et al., 2018] reproduced the CoVe paper, they dispensed with the maxout network altogether.

[20] `https://github.com/allenai/allennlp/blob/master/training_config/biattentive_classification_network.jsonnet`

[21] [McCann et al., 2017] used uncased, tokenized BLEU results whereas ours are not tokenized; papers such as [Post, 2018] argue that such differences can make BLEU scores nearly incomparable

[22] `https://github.com/salesforce/cove/blob/master/OpenNMT-py/onmt/Beam.py`

[23] `https://github.com/salesforce/cove/blob/master/OpenNMT-py/train.py#L149`

Problems aside, our results somewhat suggest that transfer learning aides performance in downstream sentiment classification tasks. CoVe results always performed better than random, although not always better than just GloVe or GloVe + Char. This suggests CoVe embeddings are beneficial, although it is unclear that they are superior to simpler models. Our results from switching languages suggest that what language the MT-LSTM is trained on may matter significantly, although more testing would be needed to show this. Our results show that the Transformer encoder performs significantly and consistently better on SST-5 than the CoVe results, and that these were further improved by combining them with BASELINE CoVe. This suggests that such combinations have merit and may be an interesting area of future work. They may also indicate that the MT-LSTM and the Transformer are learning somewhat different things in their encoders.

Our results also show that CoVe works well as an "add-on" to GloVe or GloVe and Char embeddings. In our reproducibility results, we saw the major gain over random embeddings when we introduced GloVe rather than we added CoVe; indeed, [McCann et al., 2017] do not even include results for CoVe without GloVe. Our results suggest that CoVe is rather fragile and depends on the specifics of training. While [McCann et al., 2017] did not share details of the training time or hardware they trained on, it is safe to assume that they used significant resources. Of course, the beauty of transfer learning is that the MT-LSTM need not be trained again for CoVe to benefit other models. This is the real advantage of CoVe: it may, in conjunction with other tools, moderately boost performance at little cost in terms of generating encodings. Even so, it is not clear that CoVe has any advantages over newer models such as ELMo [Peters et al., 2018] and BERT [Devlin et al., 2018].

6 Further Extensions

There are several extensions and fixes I would like to conduct with more time and resources, such as switching from GloVe 6B to GloVe 840B and altering the task-specific learning for the BCN and ELMo. I would try to make the BCN train better, possibly by replacing the maxout network as in [Peters et al., 2018]. I would change the decoder for the Transformer encoder to that used in [Vaswani et al., 2017]. I would continue to add more datasets, specifically ones in different languages, and test CoVe on other downstream tasks. I would want to add more layers to the MT-LSTM and try to combine results from each layer, essentially building an NMT version of ELMo [Peters et al., 2018]. I would test CoVe without GloVe at all and test GloVe + Char + Random, where random is a 600-dimensional, randomly initialized matrix. I would test if CoVe can be outperformed by simpler models, such as a part of speech and positional encoding, and continue to combine results from different encoders, for instance by concatenating. Lastly, I would try sub-word CoVe encodings, as adding Char encodings greatly boosted performance and these encodings would not suffer from out-of-vocabulary problems. [Devlin et al., 2018] use WordPiece embeddings [Wu et al., 2016], suggesting that performance could be boosted by using smaller units of contextualization.

Appendix A Model Sizes

Name of Model	Number of Trainable Parameters
Encoder Models	
MT-LSTM	22,377,647
CNN	21,655,247
TRNS	21,838,847
Downstream Models	
BCN	18,287,107
1-Layer CNN	8,070,530
3-Layer CNN	18,055,682

Table 9: Model sizes - the number of variables used in calculating the gradient.

We can see from Table 9 that all of the encoder-decoder models are roughly the same size, with the CNN and Transformer both being slightly smaller than the MT-LSTM proposed in [McCann et al., 2017]. For the downstream models, the BCN and 3-Layer CNN are about the same size, although the 1-Layer CNN is significantly smaller. Note that the size of the downstream models depends somewhat on the embedding; the numbers included are all for a 600-dimensional CoVe embedding.

Appendix B Encoder Training Times

Name of Model	Seconds Per Epoch
MT-LSTM	926
Tuned MT-LSTM	868
MT-LSTM-M	3,925
MT-LSTM-FR	3,418
CNN	450
TRNS	220

Table 10: Encoder Training Time Per Epoch

Table 10 includes the average training time per epoch for each of the encoders. Note that the Transformer model uses a batch size of 512 whereas the rest use a batch size of 128. We can clearly see that using non-recurrent networks can cut training time in half or better.

Appendix C Exploratory Data Analysis

Lisa and I also performed some exploratory data analysis on GloVe and the CoVe embeddings from BASELINE[24]. First, we made a histogram of the values for every entry in GloVe (Figure 4) and CoVe (Figure 5).

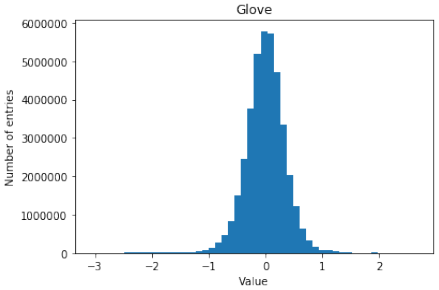

Figure 4: Histogram of GloVe entries on WMT training set

[24]The CoVe embeddings provided by [McCann et al., 2017]

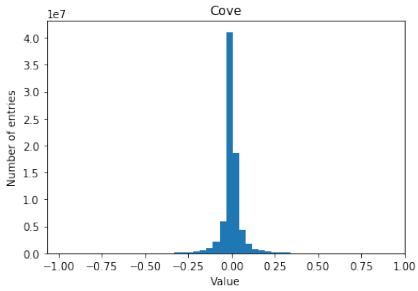

Figure 5: Histogram of CoVe entries on WMT training set

We can see from these results that the values in both GloVe and CoVe embeddings tend to roughly follow a normal distribution with mean 0. We can also see that the range for values appears to be larger for GloVe, with a significant amount of data on [-1, -0.5] and [0.5, 1] whereas almost all CoVe entries are on [-0.25, 0.25]. We also tested and saw that no entries for CoVe were exactly 0.0.

We also plotted the standard deviation across each embedding axis[25] (see Figures 6 and 7).

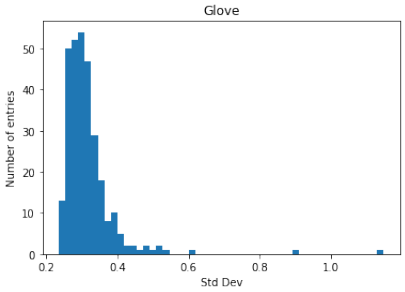

Figure 6: Histogram of standard deviations across embedding dimension for GloVe entries on WMT training set

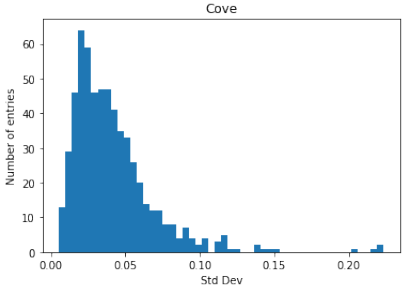

Figure 7: Histogram of standard deviations across embedding dimension for CoVe entries on WMT training set

Both of these results seem to roughly follow a chi-squared distribution, although they again differ greatly in magnitude. This result tells us that the dimensions of each embedding tend to take similar values across words, and that there are a few dimensions in each that tend to vary a lot more. For CoVe, values in a given dimension may be similar across words if they represent, say, that word's position in the sentence.

After this, we looked at several different words and compared their representations in CoVe using `scikit learn`'s [Pedregosa et al., 2011] TSNE plot [Maaten and Hinton, 2008]. These plots should group together words with similar CoVe embeddings. For each plot, we chose a word and plotted all words in sentences containing that word. The word itself is labelled on the plot. We didn't see many words with the exact same encoding, but we first plotted "set," a word with many meanings in English (Figure 8).

[25] 300 for GloVe, 600 for CoVe

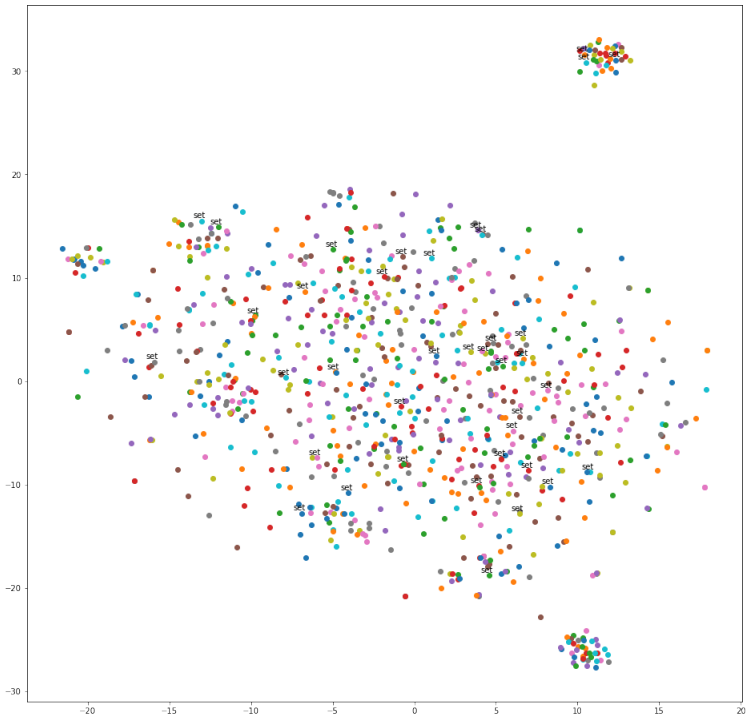

Figure 8: TSNE: set

We can see here that although there are a few clusters, "set" appears to have many different and somewhat dissimilar CoVe representations. This is to be expected, as the word's meaning should change significantly based on context. We then plotted "sad," a word with few meanings in English (Figure 9).

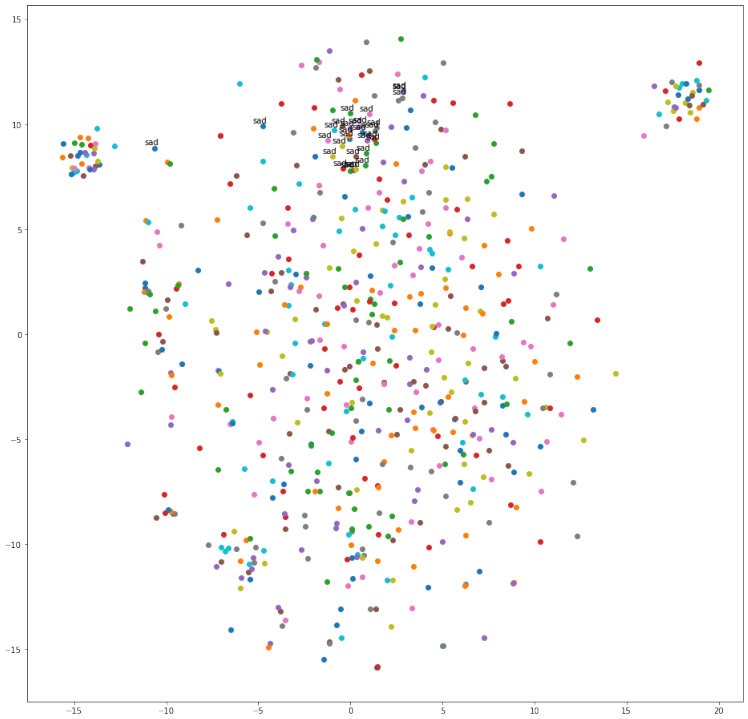

Figure 9: TSNE: sad

Here we see all of the results are clustered relatively close together, which makes sense as the meaning of "sad" is not very context dependent. We see the same thing for another word with few English meanings: "brilliant" (Figure 10).

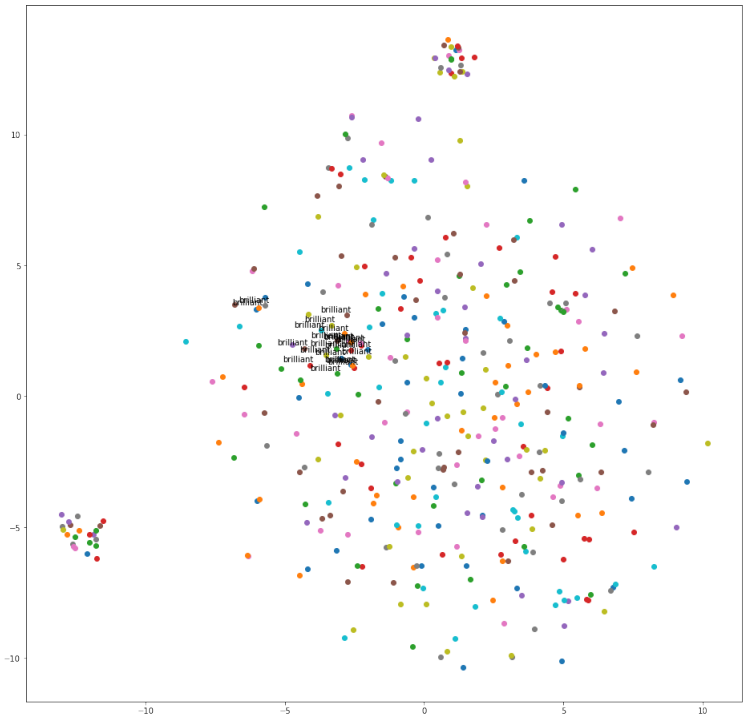

Figure 10: TSNE: brilliant

Lastly, we consider an English word with a few distinct meanings: “intelligence” (Figure 11).

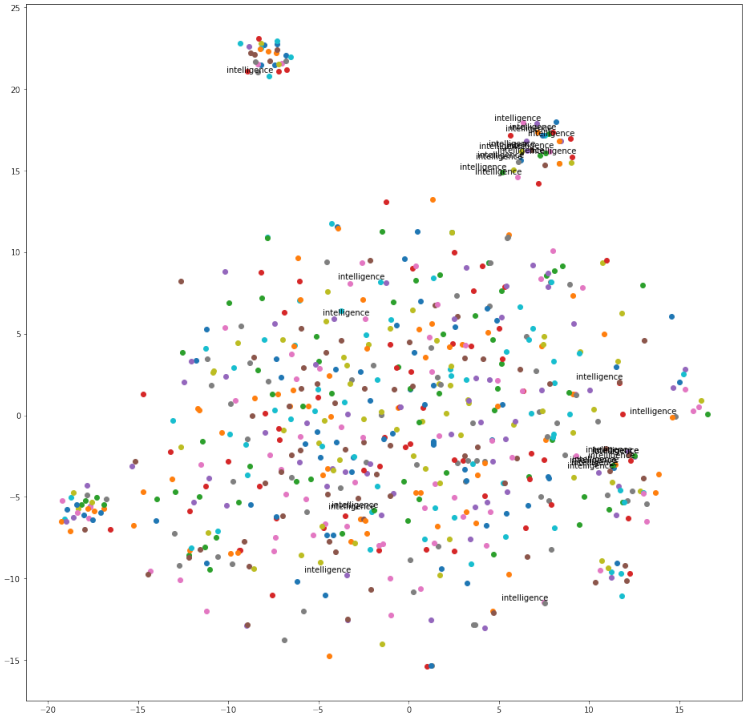

Figure 11: TSNE: intelligence

Here we see at least two distinct clusters. From looking at the source sentences, it seems that one may correspond to a positive description of intelligence (“[the film] is to be viewed and treasured for its extraordinary intelligence and originality”) and another to a negative one (“carried out by men of marginal intelligence”). If correct, this insight reinforces the idea that such embeddings can be quite useful for tasks such as sentiment classification.

References

[Cettolo et al., 2016] Cettolo, M., Jan, N., Sebastian, S., Bentivogli, L., Cattoni, R., and Federico, M. (2016). The iwslt 2016 evaluation campaign. In *International Workshop on Spoken Language Translation*.

[Devlin et al., 2018] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[Gal and Ghahramani, 2016] Gal, Y. and Ghahramani, Z. (2016). A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027.

[Gardner et al., 2017] Gardner, M., Grus, J., Neumann, M., Tafjord, O., Dasigi, P., Liu, N. F., Peters, M., Schmitz, M., and Zettlemoyer, L. S. (2017). Allennlp: A deep semantic natural language processing platform.

[Gehring et al., 2016] Gehring, J., Auli, M., Grangier, D., and Dauphin, Y. N. (2016). A convolutional encoder model for neural machine translation. *arXiv preprint arXiv:1611.02344*.

[Gehring et al., 2017] Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252. JMLR. org.

[Goodfellow et al., 2013] Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout networks. *arXiv preprint arXiv:1302.4389*.

[Graves and Schmidhuber, 2005] Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6):602–610.

[Hashimoto et al., 2016] Hashimoto, K., Xiong, C., Tsuruoka, Y., and Socher, R. (2016). A joint many-task model: Growing a neural network for multiple nlp tasks. *arXiv preprint arXiv:1611.01587*.

[Kalchbrenner et al., 2016] Kalchbrenner, N., Espeholt, L., Simonyan, K., Oord, A. v. d., Graves, A., and Kavukcuoglu, K. (2016). Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*.

[Kim, 2014] Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Klein et al., 2017] Klein, G., Kim, Y., Deng, Y., Senellart, J., and Rush, A. M. (2017). Opennmt: Open-source toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810*.

[Lei Ba et al., 2016] Lei Ba, J., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

[Luong et al., 2017] Luong, M., Brevdo, E., and Zhao, R. (2017). Neural machine translation (seq2seq) tutorial. *https://github.com/tensorflow/nmt*.

[Maaten and Hinton, 2008] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

[McCann et al., 2017] McCann, B., Bradbury, J., Xiong, C., and Socher, R. (2017). Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pages 6294–6305.

[Ouyang et al., 2015] Ouyang, X., Zhou, P., Li, C. H., and Liu, L. (2015). Sentiment analysis using convolutional neural network. In *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, pages 2359–2364. IEEE.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

[Peters et al., 2017] Peters, M. E., Ammar, W., Bhagavatula, C., and Power, R. (2017). Semi-supervised sequence tagging with bidirectional language models. *arXiv preprint arXiv:1705.00108*.

[Peters et al., 2018] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

[Popel and Bojar, 2018a] Popel, M. and Bojar, O. (2018a). Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110(1):43–70.

[Popel and Bojar, 2018b] Popel, M. and Bojar, O. (2018b). Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110(1):43–70.

[Post, 2018] Post, M. (2018). A call for clarity in reporting bleu scores. *arXiv preprint arXiv:1804.08771*.

[Seo et al., 2016] Seo, M., Kembhavi, A., Farhadi, A., and Hajishirzi, H. (2016). Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*.

[Socher et al., 2013] Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. (2013). 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.

[Specia et al., 2016] Specia, L., Frank, S., Sima'an, K., and Elliott, D. (2016). A shared task on multimodal machine translation and crosslingual image description. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, volume 2, pages 543–553.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

[Van Den Oord et al., 2016] Van Den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *SSW*, 125.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

[Wu et al., 2016] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

[Xiong et al., 2016] Xiong, C., Zhong, V., and Socher, R. (2016). Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604*.