Bilingual is At Least Monolingual (BALM):

A Novel Translation Algorithm that Encodes Monolingual Priors

Jeffrey Cheng

May 1, 2019

A THESIS
in
Copmuter Science

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Bachelor of Applied Science in Engineering

---

Professor Chris Callison-Burch
Supervisor of Thesis

Contents

1. Introduction

We motivate this research with a problem, an observation, and a technology.

1. **The problem domain**:

   Machine translation (MT) requires many pairs of parallel texts (also known as bilingual corpora): pairs of documents that are sentence-wise identical in meaning but written in different languages. Since translation is a variable-length learning problem in both input and output, it requires a large quantity of data. Machine translation on pairs of languages where parallel texts are scarce is an open problem (e.g. very few documents are written in both Haitian Creole and Sanskrit, so translation between the two is hard).

2. **A observation about current solutions**:

   It would be unreasonable – ludicrous, even – to ask a person who speaks neither English nor German to perform English-German translation. And yet most MT algorithms incorporate no prior knowledge of either its input or output language at any level (syntax or semantics); instead, these MT algorithms learn the translation problem directly. This goes against the conventional wisdom of building inductive biases based on domain expertise (e.g. translational invariance, rotational invariance, hierarchical encoding) into models.

3. **An underutilized technology**:

   It is hypothesized that the bidirectional encoder representations from transformers (BERT) algorithm allows practitioners to incorporate prior knowledge about a language by encoding sentences as fixed-length embeddings. If this hypothesis is true, then BERT can leverage the observation about how humans perform translation in order to address the MT problem posed. However, no work has proven whether BERT's mean-pooled fixed-length encodings can actually serve as a sentence embedding.

Suppose we want to translate between two languages A and B. Further suppose that there are large quantities of written text in both of these languages, but there are very few parallel texts in languages A and B. Current MT algorithms are unable to solve this problem, despite the fact that the vast majority of pairs of languages fall under this description.

Our premise is to develop a novel algorithm – the **Bilingual-is-At-Least-Monolingual model** (**BALM**) – which uses BERT to incorporate prior knowledge about language A and language B independently. This encoding of prior knowledge allows **BALM** to learn the translation problem as a fixed-length mapping problem (which is easier and more data-efficient). We now motivate our concept more deeply.

1.1 The Problem Domain: MT Restricts Choice of Model

MT is difficult in part because natural language is variable-length: the number of tokens in sentences varies greatly. Most classification models (logistic regression, decision trees, support vector machines, feedforward neural networks) have a rigid API that only allows for fixed-length inputs and fixed-length outputs. Thus, they are not only unable to learn MT; they are unable to compute even a single-forward pass over the data.

Certain architectures are designed for variable-length problems, notably recurrent neural nets (RNNs) and recursive neural nets. However, even RNNs cannot learn MT directly since their API is still not flexible enough.

- RNNs can map from variable-length inputs to fixed-length outputs.
- RNNs can map from fixed-length inputs to variable-length outputs.
- RNNs can map from variable-length inputs to variable-length outputs **iff** the input and output have the same length.

There are no atomic models that directly allow for arbitrarily variable-length inputs and outputs. Since MT has natural langauge as both input and output, there are therefore no atomic models that can directly solve MT.

MT practitioners are thus forced to use rigid compound models that either use recurrence or enforce an awkward sequence length maximum. We would like to relax the conditions of the MT problem to allow simpler models, such as feedforward neural networks.

1.2 Related Work in NLP and NMT

Neural machine translation (NMT) was popularized by encoder-decoder networks' impressive performance between English and French in 2015.[1]

Bengio et al's seq2seq architecture got around the variable-length problem by a simple composition of two attentioned RNNs: an encoder to embed sentences into a fixed-length **thought vector** and a decoder to interpret the thought vector as language. The attention mechanism allowed the models to recognize non-consecutive patterns in sequence data directly without sole reliance on memory gates. Seq2seq represented a clean improvement from prior efforts in statistical machine translation because of its compactness and because of its end-to-end differentiability.

Shortly after Bengio et al's success with seq2seq, NMT literature noted two fundamental challenges with using recurrent models for translation.

1. Hochreiter et al found that each variable-length touchpoint within recurrent models exacerbates their susceptibility to vanishing gradients since

the path along the computation graph from the end loss to early weights increases linearly in the sequence length. The chain rule is multiplicative along the gradient path; therefore, the gradient may decay exponentially, which prevents efficient training.[5] Since MT has two such variable-length touchpoints (variable-length inputs and variable-length outputs), recurrent models for MT are difficult to tune and train reliably.

2. Neural networks require a large quantity of data – even simple problems like MNIST require 60K examples and several epochs in order to converge. MT is particularly a noisy domain. Koehn and Knowles note that the combination of these two factors makes NMT is an extremely data-hungry problem setup. [7]

   Koehn and Knowles' observation is exacerbated by the fact that most pairs of languages are not like the English-French translation problem solved by Bengio et al. English and French are both extremely widely-used languages, and the pair has an enormous number of parallel texts. Most pairs of languages have few parallel texts, and the data requirement imposed by such a model for universal translation scales quadratically in the number of languages.

   For example, anyone can find a public corpus of Romanian text on the order of billions of words with a quick search.[9] However, the largest parallel text source between Romanian and another language is only about 1 million tokens. [12]

Further developments in attention led to Google Brain's development of the transformer, which currently holds state-of-the-art results in machine translation between English-German and English-French.[13]

Transformers utilize an encoder-decoder scheme using three kinds of multi-headed attention (encoder self-attention, decoder self-attention, and encoder-decoder attention). Transformers are a fixed-length attention-based model, which reduces the vanishing gradient problem. However, even the smallest pre-trained English transformer models have over 110 million parameters – the size of this model **worsens** the data-hungriness problem.

Several works have attempted to design algorithms that artificially augment the sizes of NLP and MT datasets.

- Sennrich et al developed backtranslation, a semi-supervised algorithm that concatenates monolingual utterances to bilingual corpora by using NMT to impute a translation. Backtranslation demonstrates modest increases in BLEU. However, neural nets under the backtranslation setup still do not learn the structure of any single language; the imputed examples are still used for directly learning the translation task.[10]

- Burlot et al found that variants of backtranslation and GAN-based data augmentation can increase the size of the dataset slightly without compromising the generalizability of the neural net. However, since learning curves are extremely sublinear with respect to duration, these data augmentation schemes provide small, diminishing returns.[2]

- Sriram, Jun, and Satheesh developed cold fusion (an NLP technique, contrasted with deep fusion), which performs simultaneous inference and language modeling in order to reduce dependence on dataset size. This is the exact inductive bias that humans use and is the jumping point for our work. However, cold fusion is unable to extend from monolingual inference to MT. This is likely because at the time of Sriram et al's writing (August 2017), no general sentence embedding algorithm existed. As explained later, we now bypass this constraint with Google's BERT algorithm.[11]

None of these algorithms adequeately addresses our first observation that MT given natural language understanding in a single language is a much easier learning task than directly learning MT.

Therefore, since data quantity is a performance bottleneck and and since bilingual training data for translation models is typically difficult to obtain, we would like to pre-train the models as much as possible by learning efficient representations using **monolingual data** before attempting translation. Kiros et al make progress towards pretrained monolingual language modeling for MT by learning multimodal embeddings on words.[6]

We will progress in the same vein but jump directly to pre-training sentence embeddings rather than word embeddings by using the encoding prowess of transformers. This approach could have the added advantages of context-based disamguation and better understanding of syntactical arrangement.

1.3 An Underutilized Technology: BERT for Translation

Deep bidirectional transformers for language understanding (BERT) is an extremely popular pre-trained transformer model that is supposedly able to encode sentences as fixed length vectors using only a monolingual corpus; this represents an improvement over previous language embedding technologies such as ELMO, which could only embed words into vectors.[3]

BERT uses the encoder stack of a transformer architecture to return an context-driven embedding for each word in a sequence – practitioners have found that taking the mean pool of the word embeddings in a sentence functions well as a sentence embedding. [14] However, BERT's sentence embedding property has never been verified with an autoencoder.

We note that if BERT is able to embed sentences into fixed-length vectors, the problem of translation no longer has the difficulty of being a variable-length

input problem. Similarly, if we can find an inversion of the BERT model, the MT problem will no longer be a variable-length output problem. Combining these two hypothetical successes would cast MT into a fixed-length learning problem, which can be solved with any arbitrary classification algorithm. We would then be able to use much simpler models than attentioned RNNs (such as feedforward neural nets or even logistic regression).

1.4 Research Goals

1. Create an English sentence autoencoder using a pre-trained English BERT as the encoder and a newly initialized recurrent decoder.

    - If the autoencoder can reconstruct sentences with high fidelity, we will have verified that mean-pooling over BERT's word embedding does create sentence embeddings. We will also have found a useful English thought-space that can be used for transfer learning.
    - It is still interesting if the autoencoder fails since this disproves the running assumption of many practitioners that a mean-pooled BERT output acts as a sentence embedding. We would then attribute BERT's positive performance as a transfer learning tool to its ability to learn domain-specific embedding features but an inability to compress all sentence features into a single vector.

2. Conditional on a successful autoencoder, create a German-English translation model using German BERT as an encoder, a translation model mapping between English and German thought-spaces, and the aforementioned BERT-inverting decoder.

3. Conditional on a successful translator, compare its learning curve with SOTA models and check for convergence with shorter duration (fewer minibatches / epochs). If **BALM** is successful here, then it shows promise as an algorithm for MT in pairs of languages with few bilingual corpora.

4. Conditional on a successful translator, check for reasonably good translations as evaluated by bilingual evaluation understudy (BLEU).[8] Similarly, conditional on a successful translator, check for qualitatively good translations on in-sample and out-of-sample sentence examples. If **BALM** is successful here, then it shows promise as an algorithm for general MT as a substitute for current SOTA methods.

2. Model Descriptions

We begin with a non-standard definition that will clarify the model premises.

Definition 2.1. (Thought-space) For a language L, a thought-space $S_{L,k} \subseteq \mathbb{R}^k$ is a k-dimensional embedding of sentences in language L for some fixed $k \in \mathbb{N}$.

The key insight of seq2seq learning is that by learning an intermediate thought-space, the overall sequence-to-sequence task becomes easier since the subproblems are learnable by RNNs. Our insight here is that we can make machine translation significantly easier by learning 2 intermediate thought-spaces.

Suppose we want to translate German $\rightarrow$ English, with L_{English} and L_{German} representing the formal languages. We learn the two intermediate thought-spaces $S_{\text{English},k}$ and $S_{\text{German},k}$ – each language has its own embedded space. We thus construct the **Bilingual-is-At-Least-Monolingual** (**BALM**) model for German-to-English translation with the following three submodules:

1. A German BERT encoder learns to embed German natural language into a fixed-length embedding. Its outputs are German "thoughts." **This is learnable by monolingual datasets**. In fact, one can easily download a pretrained BERT model that does this; no bilingual corpora is necessary. Formally,

$$B_{\text{German}} : L_{\text{German}} \rightarrow S_{\text{German},k} \qquad \text{(Encodes from German for fixed } k\text{)}$$

2. A feedforward neural net translates from fixed-length German thoughts into fixed-length English thoughts. Note that since this is a fixed-length problem, the learning task is significantly easier in this step and should require less data.

$$F_{\text{German}\rightarrow\text{English}} : S_{\text{German},k} \rightarrow S_{\text{English},k}$$
(Translates German thoughts into English thoughts)

3. A recurrent English decoder learns to reconstruct English natural language from fixed-length English thoughts. **This is learnable by monolingual datasets**.

$$B^{-1}_{\text{English}} : S_{\text{English},k} \rightarrow L_{\text{English}} \qquad \text{(Decodes into English for fixed } k\text{)}$$

Under this framework, translation has been stripped of its variable-length property – we only need to train one atomic model with the bilingual parallel texts!

We must first pre-train B_{German} and B^{-1}_{English}. We will take the former for granted since there are published pre-trained German BERT models. We will train B^{-1}_{English} with an autoencoder, then transfer-learn by using the pre-trained weights of B_{German} and B^{-1}_{English} in a full translation model.

2.1 Autoencoder Model

We utilize a pre-trained BERT embedding model B_{English}.

$$B_{\text{German}} : L_{\text{English}} \to S_{\text{English},k} \qquad \text{(Encodes from English for fixed } k\text{)}$$

In order to build an autoencoder with a BERT encoder, we need to be able to *invert* the function B_{German}.

$$B_{\text{English}}^{-1} : S_{\text{English},k} \to L_{\text{English}} \qquad \text{(Decodes into English for fixed } k\text{)}$$

We implement B_{English}^{-1} as an RNN with gated recurrent units (GRUs). We choose not to use vanilla RNNs due to concerns about vanishing gradients – natural language frequently reaches sequences of lengths greater than 50. We choose the GRU over the LSTM because its API is more compatible with the BERT encoder; BERT returns one hidden state; GRUs recur with one hidden state while LSTMs recur with two.

The autoencoder A is thus $\boxed{A = B_{\text{English}} \circ B_{\text{English}}^{-1}}$. Note that $A(x) \approx x$.

2.2 Translation Model

Once the autoencoder has converged, we copy B_{English}^{-1} as a pre-trained decoder and use a pre-trained German BERT model B_{German}. This allows us to transfer-learn from the language modeling task to the translation task. The full **BALM** translation model is:

$$\begin{aligned} B_{\text{German}} &: L_{\text{German}} \to S_{\text{German},k} \\ F_{\text{German}\to\text{English}} &: S_{\text{German},k} \to S_{\text{English},k} \\ B_{\text{English}}^{-1} &: S_{\text{English},k} \to L_{\text{English}} \end{aligned}$$

The translator T is thus the composition $\boxed{T = B_{\text{German}} \circ F \circ B_{\text{English}}^{-1}}$.

Note that while the translator is training, the feedforward network $F_{\text{German}\to\text{English}}$ is the only model being trained from initialization. We merely fine-tune the two transferred models B_{German} and B_{English}^{-1}.

Given the two compound models that need to be trained (autoencoder A and **BALM** translator T), we proceed to our MT methodology.

3. Methodology

3.1 Data source

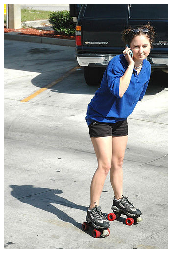
We utilize Elliott et al's Multi30k translation dataset[1], a multilingual extension of the Flickr30k image-captioning dataset.[4] Each image has an English-language caption and a human-generated German-language caption. We depict one such example below.

1. Brick layers constructing a wall.
2. Maurer bauen eine Wand.

1. The two men on the scaffolding are helping to build a red brick wall.
2. Zwei Mauerer mauern ein Haus zusammen.

1. Trendy girl talking on her cellphone while gliding slowly down the street
2. Ein schickes Mädchen spricht mit dem Handy während sie langsam die Straße entlangschwebt.

(a) Translations



1. There is a young girl on her cellphone while skating.
2. Eine Frau im blauen Shirt telefoniert beim Rollschuhfahren.

(b) Independent descriptions

Figure 1: *We see two images here. (Right) Each image has an independently written caption in both English and German. (Left) Each independent caption has a corresponding translation in the dataset. Image courtesy of Elliott et al.*

By the nature of image captioning, this dataset is biased towards static structures (e.g. "This is a white house.") and animals doing things ("The brown dog is drinking from the bowl."). Notably, there are no questions, commands, or abstract statements in the data; thus, we should expect that our models are only able to create good representations of descriptive text.

We load this dataset using the builtin function within torchtext, an NLP extension of Facebook's PyTorch framework. Natural language cannot be directly fed into an encoding model, so we must:

1. Tokenize the text.
2. Convert the text to categorical id values.
3. Initialize one-hot vectors.
4. Embed the vectors into a low-dimensional space for learning.

[1] `https://github.com/multi30k/dataset`

We utilize a hybrid ecosystem of standard *PyTorch*[2], the NLP extension *torchtext*[3], and *pytorch-pretrained-bert*[4] a library within the PyTorch ecosystem for pre-trained transformers developed by huggingface.ai. The diagram depicts the pipeline infrastructure and the dimensionality of the flow.

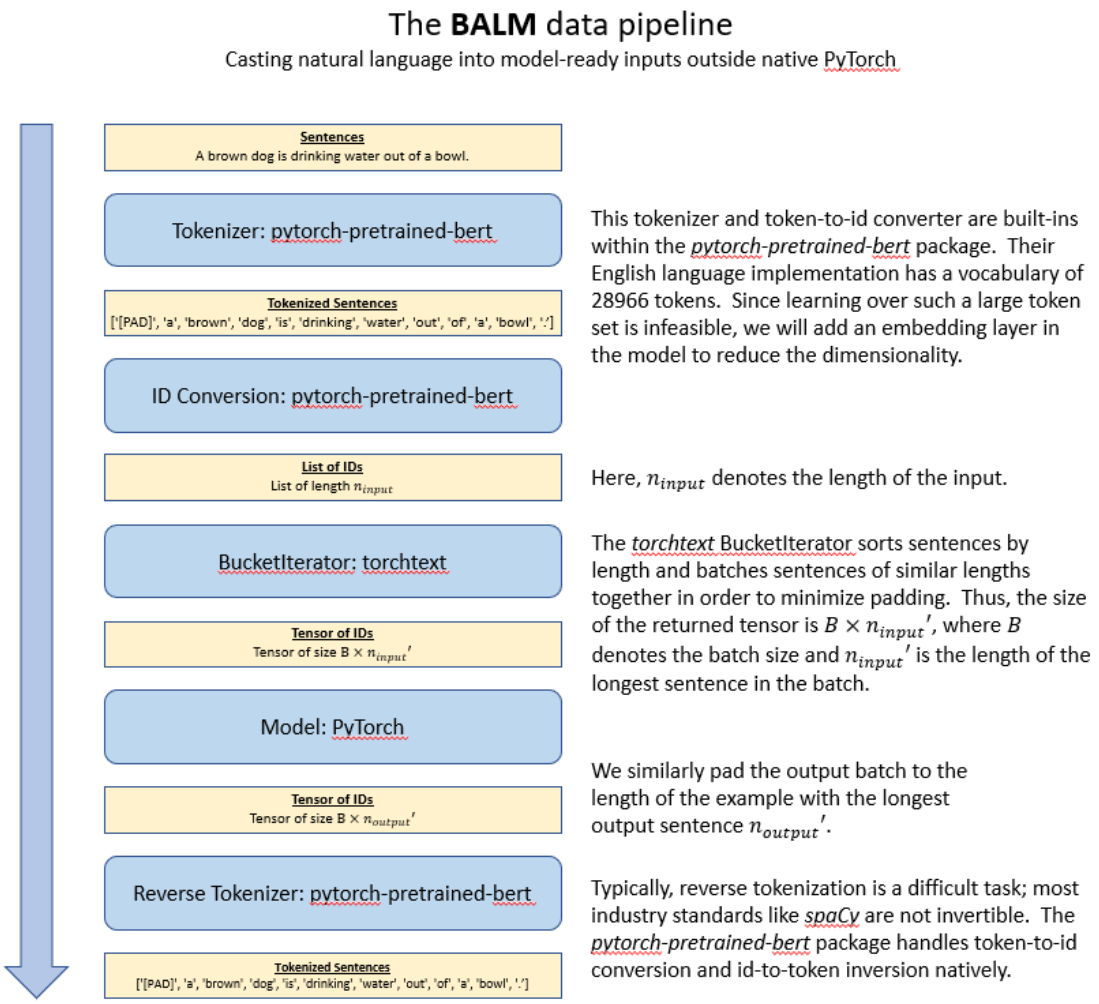


Figure 2: *Three libraries transform natural language data into model-ready batch-wise tensors. Original flow chart diagram and original descriptions.*

We now expand out the model and explain the forward passes of the autoencoder and the translator.

[2] `https://pytorch.org/docs/stable/index.html`
[3] `https://torchtext.readthedocs.io/en/latest/`
[4] `https://github.com/huggingface/pytorch-pretrained-BERT#usage`

3.2 Model implementation

Autoencoder Implementation

We implement the **BALM** autoencoder model in native PyTorch with the following modules:

- The pre-trained English BERT encoder. This is downloaded from the *pytorch-pretrained-bert* package and has 110 million parameters. We allow the gradient updates of the autoencoder to backpropagate through the pre-trained BERT model in order to fine-tune its embedding. Note that we are not using the encoder in the **BALM** translation model; the fine-tuning is solely to improve learning outcomes for the GRU decoder model.

- A mean pool layer, implemented with *torch.mean*. The BERT encoder has 12 self-attention layers, each of which outputs a hidden state. We could theoretically learn over all 12 hidden states since this would capture the entire set of low-level and high-level features embedded by the transformer. However, learning over such a rich feature space would present memory issues for the GPU; thus, we take only the last one to obtain an vector of size 768 for each token. We then mean-pool across the dimension of sentence length to get a sentence embedding of size 768; this operation is optimized by PyTorch's CUDA interface.

- The English GRU decoder layer. This is implemented as a single-layer RNN with gated recurrent units and hidden dimension equal to that of BERT's encoding. In order to produce an output at a given timestep t, the GRU takes the hidden state at t and passes it through a linear layer of output size equal to the vocabulary length (28996).

- A word embedding layer (not depicted below). We implement teacher-forcing on the GRU decoder in order to learn tail-end patterns early in training. In order to teacher-force correctly, the incoming target words have the same dimensionality as the GRU's hidden layers. Thus, we learn a custom word embedding, implemented with *nn.Embedding*.

The autoencoder's forward pass thus uses 3 trainable modules: the word embedding linear layer, the transformer encoder stack from BERT, and the GRU decoder. The embedding and the decoder are trained from initialization. The sequence of layers and rationale for architecture choices are depicted in the following flow diagram.

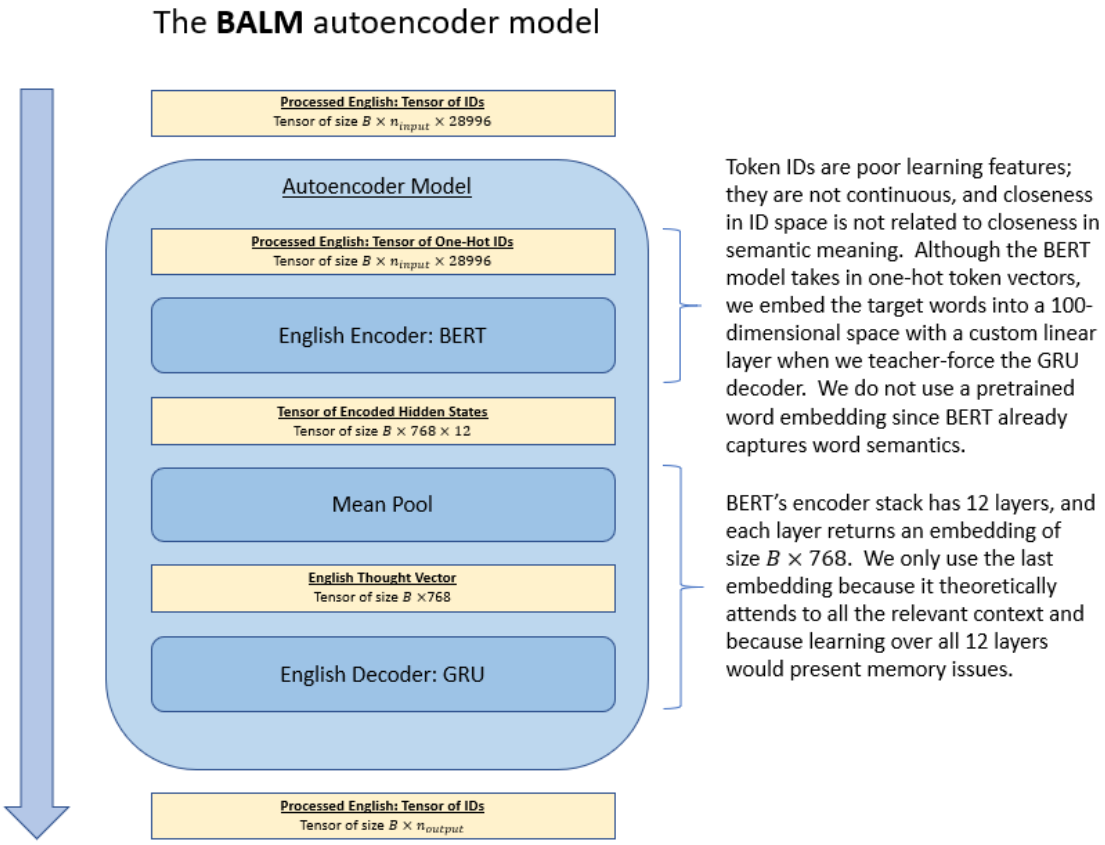


Figure 3: *Original flow chart diagram and original descriptions.*

Translator Implementation

We implement the translator with the same framework as the autoencoder, substituting the English BERT encoder for a German BERT encoder. We also add in an intermediate module to learn the desired function $F_{\text{German}\to\text{English}}$ mapping between the thought-spaces of the two languages.

- The pre-trained German BERT encoder. This is downloaded from the *pytorch-pretrained-bert* package and has 110 million parameters; the specific model used in this experiment is actually a multilingual BERT encoder that can taken in tokens in English, French, or German. We fine-tune the model by allowing gradient updates.

- A mean pool layer, implemented with *torch.mean*. The rationale for this module is the same as the rationale for the mean pool layer in the autoencoder.

- The feedforward network representing a thought translator (implemented with two *nn.Linear* units and ReLU activations). This neural net's purpose is to learn the fixed-length function $F_{\text{German}\to\text{English}}$. It has architecture

$768 \times 768 \times 768$; its input and output dimensions are fixed by the output dimension of the BERT encoder.

- The English GRU decoder layer. This is transfer-learned from the autoencoder. We allow the gradient updates of the autoencoder to backpropagate in order to fine-tune the model for translation.
- A word embedding layer (not depicted below) for teacher-forcing.

The translator's forward pass thus uses 4 trainable modules: the word embedding linear layer, the transformer encoder stack from BERT, the feedforward network, and the GRU decoder. The sequence of layers and rationale for architecture choices are depicted in the following flow diagram.

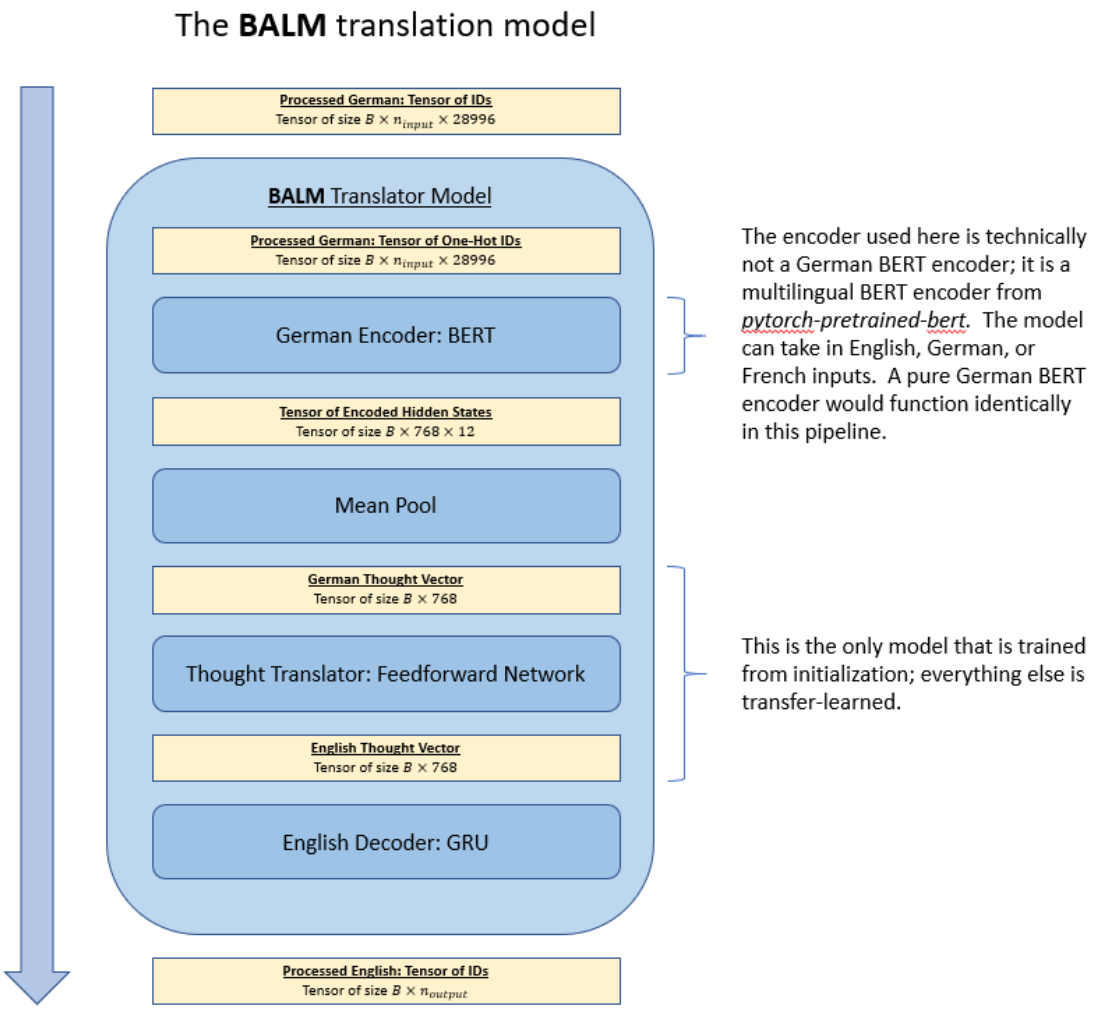


Figure 4: *Original flow chart diagram and original descriptions.*

We tuned the hyperparameters of the two models to the following constraints:

- Minimize loss on the validation set.
- Minimize iteration-to-iteration variability.
- Keep total memory allocation under 10GB in order to run the model on a standard Nvidia GeForce GTX 1080 GPU.

The final tuned hyperparameters are listed below. Surprisingly, the models seemed to be relatively insensitive to choices of hyperparameters. We attribute this to the large amount of pretraining in the submodules for both the autoencoding and the translating tasks; given a good "seed" embedding from a careful selection of hyperparameter during the pre-training phase, perhaps the actual task itself because relatively invariant to hyperparameter tuning.

Shared Hyperparameters for Autoencoder and Translator

Hyperparameter	Symbol	Tuned Value	Notes
Batch size	B	40 examples	Smaller batch sizes lead to catastrophic forgetting. Larger batch sizes seem to remove stochasticity and converge to bad local optima.
Dropout	p	0.8	This is a particularly high dropout rate to prevent overfitting; Multi30k has low variability compared to NLP benchmarks.
Learning rate	μ	0.0001	The learning curve is shockingly smooth for this pair of (model, task). Thus, the learning rate can be relatively high.
Number of epochs	T	200 epochs	Models tend to hit convergent loss around 100 epochs. Catastrophic forgetting occasionally happens beyond 500 epochs.

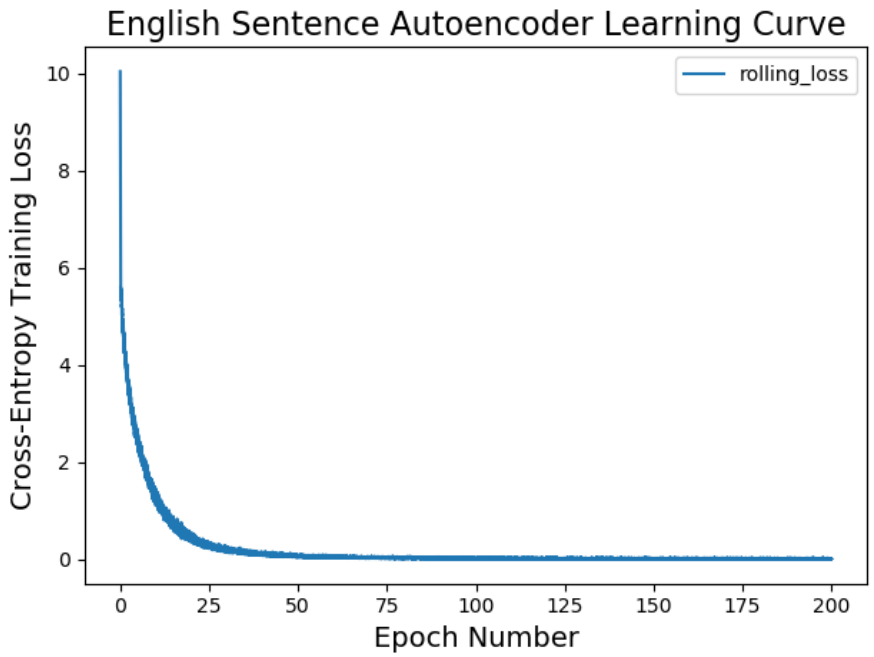
Figure 5: *The final tuned hyperparameters of both the autoencoder setup and the translator setup. They are the same for both pipelines for consistency.*

Given the above hyperparameters, we run the autoencoder model and the translator model in sequence. Each model's training run of 200 epochs took roughly 20 hours to run: a relatively quick convergence for an NLP task.

4. Results

4.1 Autoencoder Model

4.1.1 Learning Curve



The learning curve of the autoencoder has several striking features.

- The convergence is extremely fast. Many NLP tasks take several hundred epochs to converge; the elbow of this learning curve is around 15 epochs.
- The convergence is extremely smooth. Given how noisy natural language is as a domain, this is surprising.
- The convergence of the learning curve is right at the Bayes error. Random guessing over 28996 classes has an expected cross entropy loss of:

$$
\begin{aligned}
E[L(y,\hat{y})] &\geq L(y, E[\hat{y}]) && \text{(By Jensen's Inequality; } L \text{ is concave in } \hat{y}) \\
&= -\sum_{z=0}^{28995} \mathbb{I}(z=y) \ln E[\hat{y}_i] && \text{(By definition of cross-entropy)} \\
&= -\ln \frac{1}{28996} \approx \boxed{10.27} && \text{(Exactly one indicator is one)}
\end{aligned}
$$

We find that the cross-entropy loss converges to an average of 0.012 after epoch 180. This is the equivalent of the model always answering with the correct token with an assigned probability of $e^{-0.012} \approx 0.988$.

4.1.2 BLEU score

While a low cross-entropy loss is a signal of a good model and is conveniently differentiable, it doesn't directly give us the quality of reconstructions. We turn to the bilingual evaluation understudy (BLEU) score for a more holistic measure of the autoencoder's reconstruction. The BLEU score measures the "adequacy, fidelity, and fluency" of proposed translations by measuring the proportion of n-grams shared between the proposed translation and the ground-truth translation.[8]

The **BALM** autoencoder achieves a remarkably high BLEU score of $\boxed{0.605}$.

4.1.3 Qualitative Analysis of Reconstructions

Finally, in order to qualitatively interpret the output of a model, we convert the ouput IDs back into tokens. We utilize the reverse-tokenization builtins from *pytorch-pretrained-bert*. We observe the autoencoder's outputs for training examples, test examples, handwritten-caption-like examples, and non-caption-like examples.

Selected Autoencoder Reconstructions	
Type of example	**Text**
Training example	**Actual text:** ['a', 'brown', 'dog', 'is', 'drinking', 'water', 'out', 'of', 'a', 'bowl', '.'] **Predicted text:** ['a', 'brown', 'dog', 'is', 'drinking', 'water', 'out', 'of', 'a', 'bowl', '.']
Test example	**Actual text:** ['there', 'is', 'a', 'young', 'girl', 'on', 'her', 'cell', '##phone', 'while', 'skating', '.'] **Predicted text:** ['there', 'is', 'a', 'young', 'girl', 'on', 'her', 'cell', '##phone', 'while', 'skating', '.']
Like a caption	**Actual text:** ['a', 'student', 'is', 'shouting', 'at', 'his', 'computer', '.'] **Predicted text:** ['a', 'teenager', 'is', 'checking', 'at', 'his', 'computer', '.']
Not like a caption	**Actual text:** ['hey', 'ch', '##ris', ',', 'i', 'think', 'that', 'this', 'work', 'is', 'pretty', 'good', '.'] **Predicted text:** ['this', 'good', ',', ',', 'this', ',', ',', 'this', 'have', 'have', 'about', 'this', '.', '.']

Figure 6: *Selected examples of the **BALM** autoencoder's reconstructions.*

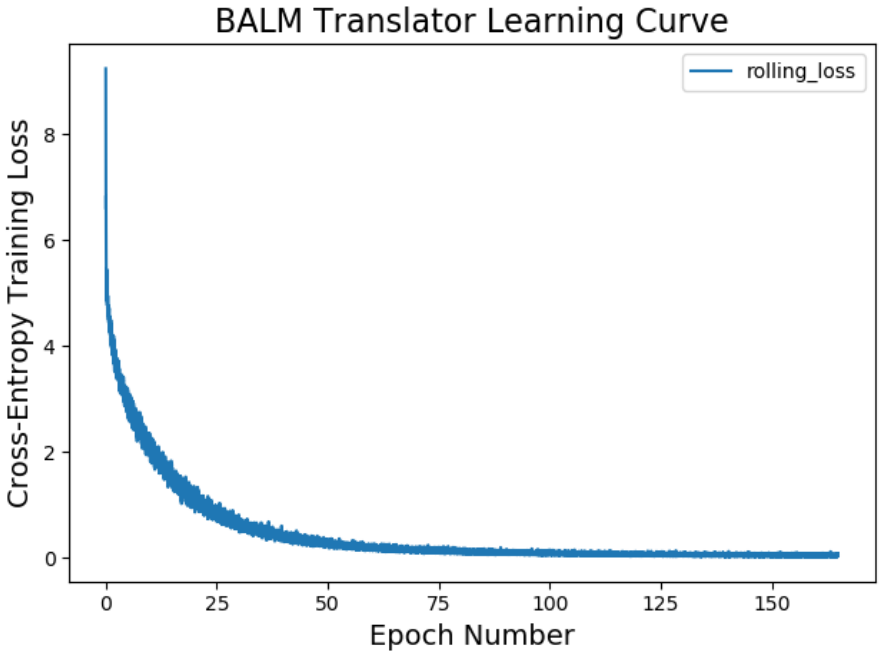
We note that on examples from the Multi30K dataset (train and test), the autoencoder has perfect reconstructions. For manually-written examples that

are somewhat similar to captions, the autoencoder gets the rough syntax and semantically similar words. On examples that are not like captions (not declarative sentences), the autoencoder is only able to capture the sentiment of the sentence.

Overall, this is exactly what we'd expect of a properly trained autoencoder on the Multi30K dataset. We take the favorable learning curve, the high BLEU score, and the selected reconstructions as strong evidence that BERT creates good sentence embeddings and that the **BALM** autoencoder is able to capture the English thought-space.

4.2 Translation Model

4.2.1 Learning Curve



We notice similar features in the translator learning curves as compared to the autoencoder's.

- The convergence is extremely fast. The translator converges just a hair slower than the autoencoder.
- The convergence is extremely smooth. Again, the translator's loss is slightly noisier than the autoencoder.

- The convergence of the learning curve is right at the Bayes error. Recall that random guessing over 28996 classes has an expected cross entropy loss of at least 10.27. After epoch 140, the translator model has an average loss of 0.014.

4.2.2 BLEU score

We again use the BLEU score for evaluation – this time, we use it as it was intended: for bilingual translation. We find that the score is $\boxed{0.248}$. Although this is not nearly as high as our autoencoder score and falls short of the state-of-the-art performance on Multi30K of 0.35, we take this BLEU score as a weak success. In general, MT practitioners consider a BLEU score above 15% to indicate that nontrivial learning is happening – and this is being achieved with a feedforward network!

4.2.3 Qualitative Reconstructions

Selected Translation Reconstructions	
Type of example	**Text**
Training example	**German text:** ['ein', 'brauner', 'hung', 'trink', 'wasser', 'aus', 'einer', 'schüssel', '.'] **Actual English text:** ['a', 'brown', 'dog', 'is', 'drinking', 'water', 'out', 'of', 'a', 'bowl', '.'] **Predicted English text:** ['a', 'brown', 'dog', 'is', 'drinking', 'water', 'from', 'his', 'bowl', '.', '.']
Test example	**German text:** ['es', 'ist', 'ein', 'junges', 'Mädchen', 'auf', 'dem', 'handy', 'beim', 'skaten', '.'] **Actual text:** ['there', 'is', 'a', 'young', 'girl', 'on', 'her', 'cell', '##phone', 'while', 'skating', '.'] **Predicted text:** ['a', 'young', 'girl', ',', 'is', 'walking', 'at', 'the', 'cell', '##phone', '.', '[PAD]']

Figure 7: *Selected examples of the **BALM** translator's reconstructions.*

We see that the translation model does relatively well on the training examples. In fact, it produces the strongest signal of natural language understanding: a correct translation that is synonymous but not identical to the given translation. However, the model struggles a bit on the test set. There is clear n-gram similarity, but the model misses the key verb and returns a poorly formed sentence.

5. Conclusion

We first conclude that NLP practitioners are correct to assume that a simple mean-pool over BERT's word embeddings serve as a rich sentence embedding. We take the extremely high performance of the BERT-driven sentence autoencoder – nearly zero cross-entropy loss, an extremely high BLEU score of 0.605, and impressive performance on out-of-dataset examples – as strong evidence that the English thought-space learned by BERT captures all salient features of the English natural language (at least within the subdomain of image captioning).

This work definitely contradicts Ray Mooney's famous exclamation at the Association of Computational Linguistics (ACL) 2014: "You can't cram the meaning of a whole %&!$# sentence into a single $&!#* vector!"

Next, we conclude that BERT embeddings allow extremely complex sequence-to-sequence NLP problems like MT to be solved by extremely simple models like feedforward networks. A simple transfer learning from an autoencoder and a shallow feedforward network trained on the German-to-English translation task achieves similar training curves, a sub-SOTA but respectable BLEU score of 0.248, and reasonable in-sample reconstructions.

Finally, we conclude that the **BALM** algorithm does seem to converge faster than both seq2seq and transformer-based MT systems, although its final performance is not as strong. This bodes well for the initial premise of this research, which was for translating between pairs of languages that have very few parallel bilingual texts.

6. Impact

Consider the Haiti earthquake. Many organizations, such as the United Nations, UNICEF, and the Red Cross, immediately reached out to offer humanitarian aid; however, there are few translators between, say, English and Haitian creole. Before this crisis, automated translation between frequently spoken languages and Haitian creole did not exist because of a lack of parallel texts.

Microsoft's best efforts reflect positively on the company but poorly on the state-of-the-art of NLP at the time. Their fast-tracked model involved crowdsourcing parallel texts from universities (e.g. Carnegie Mellon University), websites (e.g. haitisurf.com), and the online Haitian community. It took Microsoft Research's NLP group 5 days to support Haitian creole on Bing's translator service.[5]

There are roughly 4500 languages with more than 1000 speakers.[6] Now consider a model that could translate between any of the $\binom{4500}{2} = 10122750$ pairs of languages with minimal assistance from bilingual corpora. Translation

[5] https://www.microsoft.com/en-us/research/blog/translator-fast-tracks-haitian-creole/

[6] https://www.infoplease.com/askeds/how-many-spoken-languages

would no longer depend on a data bottleneck (especially one that's difficult to solve with crowdsourcing given the rarity of speakers in some lagnauges). Key stakeholders would include humanitarian organizations, any internationally-deployed branch of the military, scholars of dead languages, and language-sensitive content creators.

The value of this research is that by better utilizing more plentiful data resources – monolingual rather than bilingual texts – we open up the possibility of high-quality machine translation beyond the few pairs of frequently used languages.

7. Future Work

This work has opened up many doors to follow-up analyses and use cases.

1. **Interpretability**

   A major issue with neural methods in general is that they are considered opaque: as a semiparametric family of models, it is difficult to analyze what is happening in parameter space.[7] However, a MT algorithm under the **BALM** framework has two usable thought-spaces that can be used to visualize any mistakes. Suppose that a mistake occurs in translation due to the German encoder creating a faulty embedding; we can check this by using a German thought-decoder. Suppose that a mistake occurs in translation due to the English decoder reconstructing poorly; we can check this by using an English encoder on the poor reconstructor to see if we reproduce the output of the thought translator. And suppose that a mistake occurs in the thought translator; we can use the English decoder to pinpoint the source of the error. It would be interesting to see whether certain submodules are more susceptible to certain kinds of errors and whether it's possible to create algorithmic tools for identifying or preventing such mistakes.

2. **Regularization and Learnability**

   We see that a shallow feedforward neural network can function as a reasonable thought translator. Would a more complex fixed-length model be able to increase the performance of **BLAM** to SOTA? A deeper neural network would not be more expressive but could improve the computational learning properties of the system.

   Or perhaps the feedforward neural network is too complex and overfits on the data: we saw in the translator that it faithfully reproduced the training examples but struggled with test examples. Perhaps an extremely simple model like logistic regression would serve as a regularization scheme.

   Other forms of regularization could include changing the optimizer (we used Adam by default); some works seem to demonstrate that SGD acts as

better regularizer because of its increased stochasticity. Perhaps combining SGD with **BALM** would reduce the amount of overfitting.

3. **Transfer Learning on Different MT Tasks**

   We test here only one one dataset: Multi30K. We do not know if this algorithm will even generalize to other German-to-English translation tasks outside of image captioning. Furthermore, it would be interesting to see if the target language decoder for one task can be lifted onto a different task altogether; in theory, they should function the same if given the same BERT embedding.

8. Appendix

8.1 Explicit Model Architectures as Code

```
class GRUDecoder(nn.Module):
    def __init__(self, emb_dim, vocab, hid_dim, n_layers, dropout):
        super().__init__()

        self.emb_dim = emb_dim  # should be 1 if we don't embed
        self.vocab_size = len(vocab)
        self.embed = nn.Embedding(self.vocab_size, self.emb_dim)
        self.hid_dim = hid_dim
        self.n_layers = n_layers  # should be 1
        self.rnn = nn.GRU(emb_dim, hid_dim, n_layers, dropout=dropout)
        self.out = nn.Linear(self.hid_dim, self.vocab_size)

    def forward(self, last_word, last_hidden):
        output, new_hidden = self.rnn(last_word.float(), last_hidden.float())
        prediction = self.out(output.squeeze(0))
        return prediction, new_hidden
```

```
class Autoencoder(nn.Module):
    def __init__(self, encoder, decoder, device):
        super().__init__()

        self.encoder = encoder
        # self.shrink = shrink_net
        self.decoder = decoder
        self.device = device
        self.number_of_batches_seen = 0

    def forward(self, src, trg, teacher_forcing_ratio=0.5):

        batch_size = trg.shape[1]
        if trg is None:
            max_len = 100
        else:
            max_len = trg.shape[0]

        outputs = torch.zeros(max_len,
                              batch_size,
                              self.decoder.vocab_size).to(self.device)

        src = src.permute(1, 0)
        hidden = self.encoder(src)

        hidden = hidden[0]  # ignore pooled output
        hidden = hidden[-1]  # only grab last layer's output
        hidden = torch.mean(hidden, dim=1)
        hidden = hidden.unsqueeze(dim=0)

        # first input to the decoder is the <sos> tokens
        curr_token = trg[0, :]

        for t in range(1, max_len):
            curr_token = self.decoder.embed(curr_token)
            curr_token = curr_token.unsqueeze(dim=0)
            new_output, hidden = self.decoder(curr_token, hidden)
            outputs[t] = new_output
            teacher_force = random.random() < teacher_forcing_ratio
            top1 = new_output.max(1)[1]
            curr_token = (trg[t, :] if teacher_force else top1)

        self.number_of_batches_seen += 1
        return outputs
```

```
class Translator(nn.Module):
    def __init__(self, encoder, decoder, device):
        super().__init__()

        self.encoder = encoder
        self.fc1 = nn.Linear(768, 768)
        self.fc2 = nn.Linear(768, 768)
        self.decoder = decoder
        self.device = device
        self.number_of_batches_seen = 0
        self.relu = nn.ReLU(True)

    def forward(self, src, trg, teacher_forcing_ratio=0.5):
        batch_size = trg.shape[1]
        if trg is None:
            max_len = 100
        else:
            max_len = trg.shape[0]
        outputs = torch.zeros(max_len,
                              batch_size,
                              self.decoder.vocab_size).to(self.device)
        src = src.permute(1, 0)
        german_thought = self.encoder(src)
        german_thought = german_thought[0][-1]
        german_thought = torch.mean(german_thought, dim=1)
        german_thought = self.relu(self.fc2(self.relu(self.fc1(german_thought))))

        english_thought = english_thought.unsqueeze(dim=0)
        hidden = english_thought

        # first input to the decoder is the <sos> tokens
        curr_token = trg[0, :]

        for t in range(1, max_len):
            curr_token = self.decoder.embed(curr_token)
            curr_token = curr_token.unsqueeze(dim=0)
            new_output, hidden = self.decoder(curr_token, hidden)
            outputs[t] = new_output
            teacher_force = random.random() < teacher_forcing_ratio
            top1 = new_output.max(1)[1]
            curr_token = (trg[t, :] if teacher_force else top1)

        self.number_of_batches_seen += 1
        return outputs
```

8.2 Links to Research Materials

Github repository: `https://github.com/jeffreyscheng/senior-thesis-translation`

Pre-trained **BALM** models: `https://drive.google.com/drive/folders/1bt8hU24U_Uwn9j7gque2IjGzv4dphz3M?usp=sharing`

9. Works Cited

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014.

[2] Franck Burlot and FranÃğois Yvon. Using monolingual data in neural machine translation: a systematic study. *Proceedings of the Third Conference on Machine Translation*.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[4] Desmond Elliott, Stella Frank, Khalil Sima'an, and Lucia Specia. Multi30k: Multilingual english-german image descriptions. *CoRR*, abs/1605.00459, 2016.

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

[6] Ryan Kiros, Ruslan Salakhutdinov, and Richard S. Zemel. Unifying visual-semantic embeddings with multimodal neural language models. *CoRR*, abs/1411.2539, 2014.

[7] Philipp Koehn and Rebecca Knowles. Six challenges for neural machine translation. *CoRR*, abs/1706.03872, 2017.

[8] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[9] Michael Rundell. Romanian text corpora. https://www.sketchengine.eu/user-guide/user-manual/corpora/by-language/romanian-text-corpora/.

[10] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. *CoRR*, abs/1511.06709, 2015.

[11] Anuroop Sriram, Heewoo Jun, Sanjeev Satheesh, and Adam Coates. Cold fusion: Training seq2seq models together with language models. *CoRR*, abs/1708.06426, 2017.

[12] Clarin Eric: Utrecht University. Parallel corpora in the clarin infrastructure. https://www.clarin.eu/resource-families/parallel-corpora.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[14] Han Xiao. Bert-as-a-service. https://github.com/hanxiao/bert-as-service.