Alma Mater Studiorum · University of Bologna

Question answering on the SQuAD dataset

NLP course final project

Leonardo Calbi (leonardo.calbi@studio.unibo.it)
Lorenzo Cellini (lorenzo.cellini3@studio.unibo.it)
Alessio Falai (alessio.falai@studio.unibo.it)

February 2, 2021

Contents

1 Summary

The tasks of Machine Comprehension (MC) and Question Answering (QA) have gained significant popularity over the past few years within the natural language processing and computer vision communities. Systems trained end-to-end now achieve great results on a variety of tasks in the text and image domains.

In this work we address a question answering problem and, in particular, the Stanford Question Answer Dataset (SQuAD) problem: given a large collection of Wikipedia articles with associated questions, the goal is to identify the span of characters that contains the answer to the question.

This project focuses on the SQuAD v1.1 dataset, that contains more than 100,000 question-answer pairs on more than 500 articles. Here each question has at least one associated answer, whereas in the SQuAD v2.0 dataset there are also questions with no answers at all.

In this work we implement and compare five different models:

1. A naïve LSTM encoder-decoder that will act as our baseline
2. The Bi-Directional Attention Flow (BIDAF) network [20]
3. A model that wraps a pretrained Bidirectional Encoder Representations from Transformers (BERT) [4], as language model, coupled with a custom output layer on top of it
4. A model that wraps DistilBERT [19], a smaller general-purpose language representation model, 40% smaller then BERT but with 97% of its language understanding capabilities and 60% faster
5. A model that wraps ELECTRA [3], a more sample-efficient pre-training task called "replaced token detection"

For the last three models, we developed a modular reusable output layer, allowing us to implement different models by only change the language model levels (input, embedding and modelling).

Our experimental evaluations show that our models achieve ....

2 Background

As already described above, the question answering task is based on the idea of identifying one possible answer to the given question as a subset of the given context.

Since the input data is of textual form and the latest models for the task are all based on neural architectures, there is the need to encode such text into a numeric representation. As of today, there are two main approaches to embed words in numerical format: sparse embeddings, like TF-IDF [17] and PPMI [12], and the modern dense embeddings, such as Word2Vec [10] and GloVe [15]. In the latter case, embeddings for each word in the input vocabulary are usually computed with shallow encoders.

These emebddings are then used as inputs for models specialized in processing sequential inputs. Nowadays, such models are mostly based on the Transformer architecture [22], which has the attention mechanism at its core. Instead, before this revolution, NLP competition's leaderboards were mostly populated by models based on recurrent and convolutional modules.

The most influential recurrent networks are LSTMs [8] and GRUs [5]. They are both based on processing sequential inputs and they keep an evolving series of hidden states, such that the output $h_t^{(0)}$ is a function of $h_{t-1}^{(0)}$ and the input x_t at position t. Recurrent layers can also be stacked to increase the capacity of the network: in that case, $h_t^{(0)}$ would act as an input to $h_t^{(1)}$, i.e. the first hidden state of the next depth-wise layer, and the same goes for the successive layers, as shown in figure 2.1.

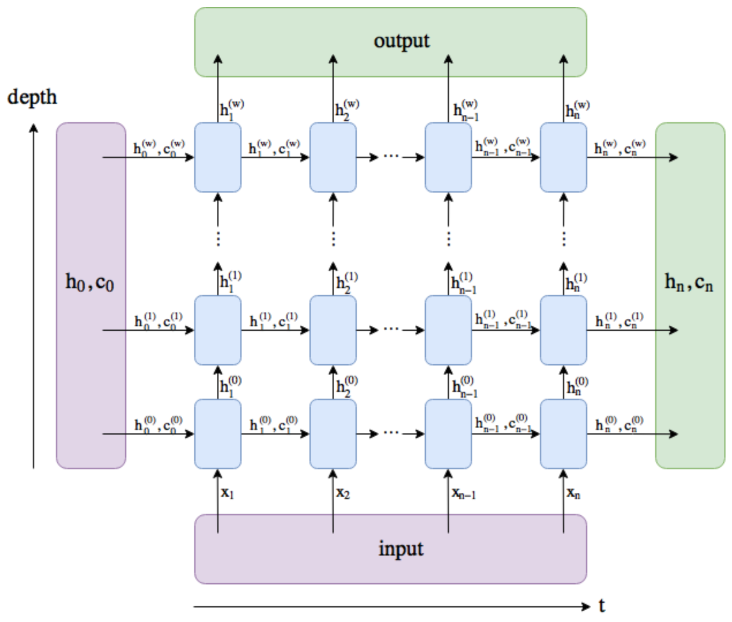


Figure 2.1: Recurrent module (image courtesy of [11])

In natural language, meaning does not usually simply flow from left to right, so that in RNNs other techniques may be necessary to reach a higher level of understanding. One example of such improvement is bi-directionality: the input is processed in two ways, from left to right (the "forward" pass) and from right to left (the "backward" pass); then, outputs of the two computations are merged together (usually by concatenating them over their last dimension). The intuition behind bi-directional RNNs is that they tend to understand the overall context better than their uni-directional counterparts, since they are able to aggregate information from the "past" and from the "future".

About Transformers, at a high-level, they comprise a stack of encoder-decoder modules, where each encoder features a multi-head attention block (to focus on different subsets of the input sequence when encoding one specific token) and a point-wise fully-connected layer, while each decoder features the same architecture as the encoder, but with an additional multi-head attention block in the middle (that helps the decoder focus on relevant parts of the input sequence).

The multi-head attention modules in the Transformer architecture are the ones responsible for gathering information from other tokens in the input sequence, thus resembling the task of hidden states in RNNs (i.e. incorporate the representation of previous inputs with the currently processed one).

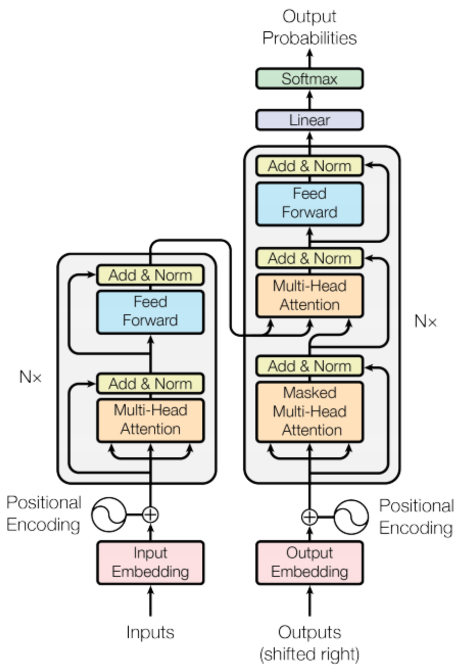


Figure 2.2: Transformers (image taken from [22])

While Transformers may always seem the way to go, since they throw away all the computational burden of training recurrent modules (which are sequential in nature and, thus, scarcely parallelizable), they also suffer from limitations due to the quadratic complexity of attention and the inherent maximum number of input tokens (which is implicitly linked to architecture choices themselves).

3 System description

In this work we implement three distinct models: a naïve LSTM encoder-decoder that will act as our baseline, the Bi-Directional Attention Flow (BIDAF) network [20] and some models that wraps a pretrained Bidirectional Encoder Representations from Transformers (BERT) [4] or variations of it, specifically DistilBert [19] and Electra [3], as language models, coupled with a custom output layer on top of it.

Each model is trained on the SQuAD v1.1 training set and evaluated on the SQuAD v1.1 dev set, available on the github repository of the SQuAD project.

The models' evaluation is carried out on the same preprocessed data: indeed, we defined a preprocessing pipeline, applied it to the training and dev set once and the resulting dataset is fed to each model. (CONTROLLARE QUI – mi pare che per bert abbiamo tokenizzato diversamente)

In order to perform an effective models' comparison, each model shares the same input and output layer: what changes is the language model (embedding + attention layers) and the modeling layer.

The following is a detailed description of the models.

3.1 Recurrent-based models

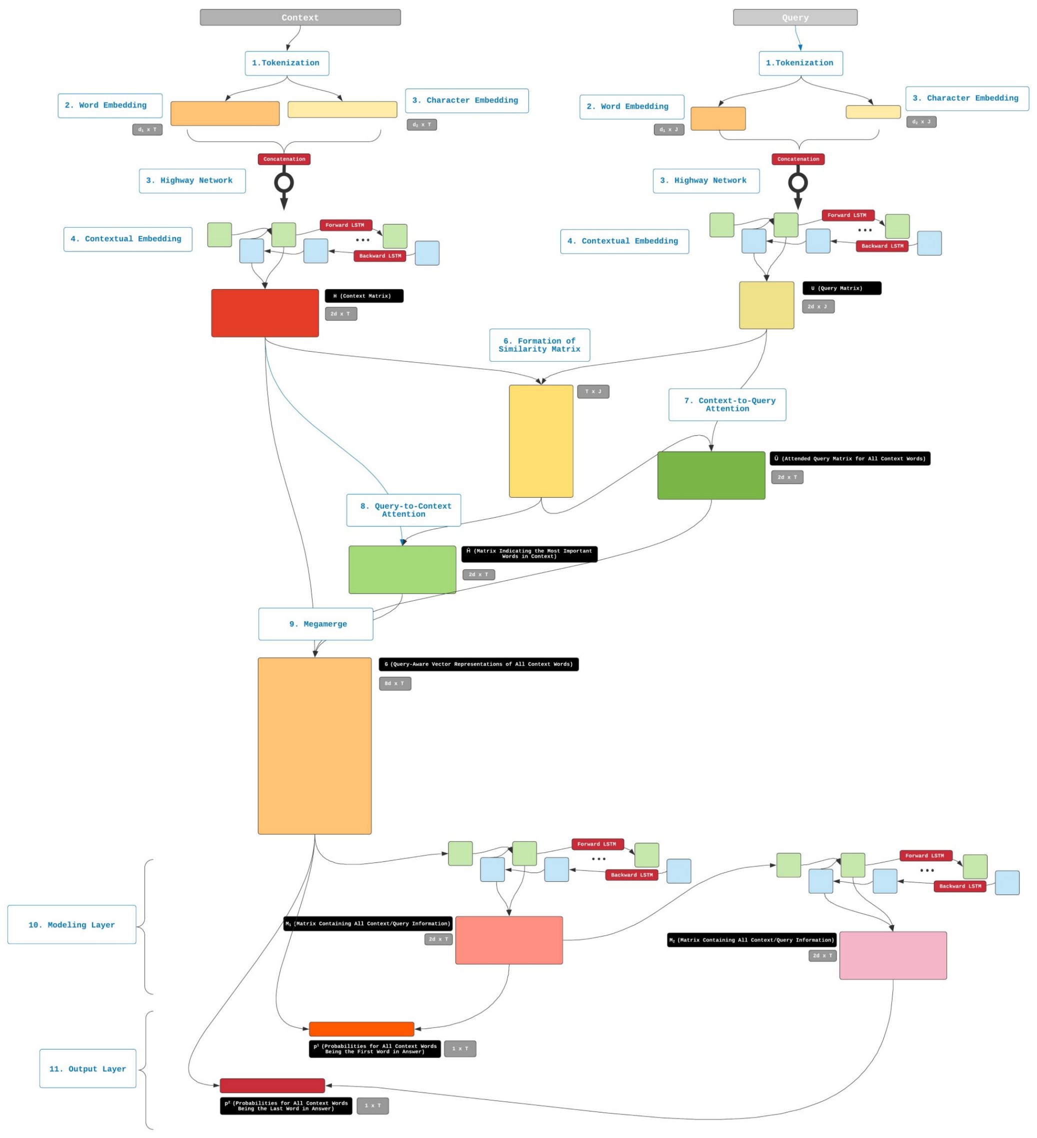
3.1.1 Baseline

The baseline model embedding layer performs a standard word level embedding on both questions, and context, than proceeds projection this embeddings into a fixed hidden size dimesion. Regarding the modeling it is composed by a single recurrent encoder, which is given both questions and contexts as two separate inputs. Then, all the hidden states of a single question are averaged together (over the embedding dimension) so as to obtain a single vector which should encode the semantic information of the question at the sentence level. This aggregated question vector is then element-wise multiplied to each context token latent representation, so as to perform some kind of query-aware context encoding. Finally, the query-aware context vectors are passed onto another recurrent module and used as inputs for the end token classifier, while the query-aware context vectors are directly used as input for the start token classifier.

3.1.2 BiDAF

The BiDAF model is composed of four major layers, each one defines a more in depth representation of the question, context input couple:

1. Embedding layers
2. Attention layers
3. Modeling layer
4. Output layer.



Context
Query
1.Tokenization
2. Word Embedding
3. Character Embedding
d_w x T
d_c x T
d_w x J
d_c x J
Concatenation
3. Highway Network
4. Contextual Embedding
Forward LSTM
Backward LSTM
H (Context Matrix)
2d x T
U (Query Matrix)
2d x J
6. Formation of Similarity Matrix
T x J
7. Context-to-Query Attention
Ũ (Attended Query Matrix for All Context Words)
2d x T
8. Query-to-Context Attention
H̃ (Matrix Indicating the Most Important Words in Context)
2d x T
9. Megamerge
G (Query-Aware Vector Representations of All Context Words)
8d x T
10. Modeling Layer
M1 (Matrix Containing All Context/Query Information)
2d x T
M2 (Matrix Containing All Context/Query Information)
2d x T
11. Output Layer
p1 (Probabilities for All Context Words Being the First Word in Answer)
1 x T
p2 (Probabilities for All Context Words Being the Last Word in Answer)
1 x T

Figure 3.1: General BiDAF model workflow (image courtesy of [1])

We tried to follow the original paper [20] as closely as possible as presented in Figure 3.1, but in the end we made some minor changes. The Embedding layers of our model perform word level embedding on both question and context tokens using a sligthly modified GloVe embedding, see 4.2 and 4.3, then the achieved embedding is projected into a fixed size hidden dimesion using a linear layer without bias, similarly to the baseline model. The last step is performed by a highway network [21] which is usually used to balance the contributions of words and character embedding, but in our case we used it to ease the training process, lacking the character embedding part, we removed it because it seem to increase the metrics by only a couple of percentage points, while requiring a lot of additional computation and structure to work optimally.

After the highway network we delve into the heart of BiDAF which takes as input the result of a contextual embedding layer, shared by contexts and query, formed by a single layer bidirectional LSTM. With this two outputs we can compute the similarity matrix of context and queries, as the product of weights of a linear layer, without bias, and the concatenation of question,context and element-wise multiplication of the formers. This similarity matrix represents how much each question and context words are grammatically and semantically alike, it is used as base for the evaluation of both BiDAF peculiar attentions, Context-To-Query (C2Q) and Query-To-Context (Q2C). This attentions represent which query words are most relevant to each context words and vicerversa. The last step in the attention layers is called "megamerge" which is the concatenation of four embeddings:

- Context embeddings of the contextual layer
- Context-To-Query values
- Element-wise multiplication of Context-To-Query values and context embeddings
- Element-wise multiplication of Query-To-Context values and context embeddings

The modeling layer is structured as two bidirectional LSTMs, the first one has two layers and takes as input the "megamerge",the second one has one layer and runs on the output of the first, this specific result is used to create a differentiation in the modeling to aid the search of the start token's index and end token's index of the answer.

The output layer receives as inputs the concatenation of "megamerge" and the corresponding modeling outputs, see 3.3

3.2 Transformer-based models

3.2.1 BERT model

All the transformer models are based on BERT [4], which is a language model, composed of a multi-layer bidirectional transformer encoder [22] (we used it's base implementation which has 12 transformer layers and 768 has hidden size).

The peculiarity of it's transformer's architecture allows the model to learn the context of a word based on all of its surrounding at the same time, not just as a sequential read. This reflects in it's pre-training, which is composed of two tasks:

- Masked LM (MLM)
- Next Sentence Prediction (NSP)

Simplistically Masked LM is performed masking 15% of the words in each sequence replacing them whit a $[MASK]$ token, the model is required to predict the original value of the masked words, using the available ones. Next Sentence Prediction is achieved giving as input to the model a pair of sentences formatted as in 4.2 in the case of transformer tokenization, which are in 50% of the cases two subsequent sentences, and a random picked couple otherwise, the model is asked to identify if the following sentence is subsequent to the first. BERT model are trained using both task together with the goal of minimize the combined loss [9].

3.2.2 DistilBERT model

BERT performance, as for the majority of large-scale language models, are inevitably intertwined with their number of parameters, DistilBERT [19] tries to reduce this number using a technique called `distillation` which consist in training a smaller "student" network to mimic the full output of the big "teacher" model [7]. Specifically DistilBERT has about half the parameters of BERT and retains 95% of its performance on GLUE, its distillation uses `Kullback-Leibler loss` as explained in [18].

3.2.3 ELECTRA model

ELECTRA (Efficiently Learning an Encoder that Classifies Token Replacements Accurately) [3] as an architecture equal to BERT, it's only difference is using a new pre-training approach, which aims to outperform MLM strategy. It's structure is composed of three steps:

1. Given an input sequence randomly replace tokens with a $[MASK]$ token
2. The `generator` (a small MLM) predicts the original token for the masked one
3. The new sequence with the replaced token is given to the `discriminator` (ELECTRA) which objective is to identify for each token if it is original or replaced

At the end of the training the `discriminator` is saved and the `generator` is thrown away. This approach allows to compute the loss over all input tokens and not only on the masked ones, which is the key difference between the two approaches [16].

3.3 Output layer

The output layer is composed of two linear layers, one for the identification of the start index and one for the end one. They receive two different inputs, the end token one being the result of the computation of a single layer recurrent encoder on the start input. We then compute the softmax, s.t. we obtain a start and an end probability for each token in the context. The final part is the search of the correct section of context that forms the answer, this is made finding the feasible intervall with maximum combined probabilities (respecting the assumption that the start index must precede the end one).

4 Experimental setup and results

4.1 Data handling

Given the JSON training set, we decided to parse it into a suitable tabular data structure, which turned to be useful even for subsequent pre-processing tasks. In particular, since each context has multiple question/answer pairs and each question can have multiple answers, we allocated one row for each context/question/answer triple, thus replicating the same context and question multiple times, in order to consider it as a separate example.

Regarding the splitting strategy for the dataset, we went for a simple holdout, thus setting aside 20% of the whole dataset for validation purposes. The way the validation set is built guarantees that the entire set of rows referencing to the same context is kept into the same split, thus avoiding unjustifiable boosting of results.

As an additional test, we evaluated all of our models on the official SQuAD v1.1 dev set. The main difference between this dataset and the training one is that the same question can have multiple correct answers.

Moreover, final models are re-trained on the whole dataset (by merging training and validation splits), so as to simulate the cross-validation strategy.

4.2 Tokenization

Tokenization has the following meaning: given a string of text, its task is to split it into substrings (called tokens), depending on the chosen algorithm. Different models call for different tokenization pipelines. In our case, we have two macro-categories:

1. Recurrent-based models tokenizer: splits words by whitespaces and punctuations, removes accents and applies a lowercasing function to all the tokens. Moreover, questions are padded (not truncated), while contexts are truncated and padded. Taking inspiration from [13], the maximum number of tokens for truncating contexts was fixed to 300.

2. Transformer-based models tokenizer: splits words using the WordPiece algorithm (introduced in [24]), removes accents and applies a lowercasing function to all the tokens, while also merging together questions and contexts as $[CLS]q_1q_2 \dots q_n[SEP]c_1c_2 \dots c_m[SEP]$ (it leverages the special tokens $[CLS]$ and $[SEP]$). Moreover, the combined question/context sentence is truncated to a maximum number of tokens (512, as in [4]) and padded to the right.

Tokenization happens on the fly at the batch level, thus enabling us to perform dynamic padding (based on the longest sequence in a batch) and avoiding the pre-tokenization overhead.

The tokenizer is also used as some kind of pre-processor, to remove from the training dataset those rows whose contexts would not contain the relative answer, after truncation.

4.3 Embeddings

In recurrent-based models, tokens are numerically encoded using the standard GloVe embeddings. In particular, we tested different types of GloVe models, with different embedding dimensions:

- Wikipedia 2014 and Gigaword 5 (6B tokens, 400K vocab, uncased, 300d vectors)
- Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 200d vectors)

We additionally embedded two more tokens (not present in the standard GloVe vocabulary):

1. $[PAD]$: the corresponding vector contains all zeros
2. $[UNK]$: the corresponding vector, which is used to handle OOV (Out Of Vocabulary) words, is the mean of all the GloVe vectors, as reported by Pennington (one of GloVe's authors)

> *I've found that just taking an average of all or a subset of the word vectors produces a good unknown vector*

In Transformer-based models, we instead rely on the wrapped language models of the HuggingFace library [23], so that there is no need to manually handle token embeddings.

4.4 Metrics

Models are evaluated at the end of each epoch and metrics are computed for both the training and validation set. In our case, the validation set was mainly used to track models' performance on unseen data, in order to avoid overfitting problems.

Metrics used for evaluating the models are F1 and EM (Exact Match) and they were taken from the official SQuAD evaluation script. The F1 score ($f1 = \frac{2 \cdot p \cdot r}{p+r}$), defined as the harmonic mean between precision ($p = \frac{TP}{TP+FP}$) and recall ($r = \frac{TP}{TP+FN}$), is computed per question and then averaged across all questions. The EM score is instead computed as the number of questions that are answered in exact same words as the ground truth, divided by the total number of questions.

Since the same question can have multiple ground truths (specifically, in the dev set), there is the need to select just one for each question. To fulfill this requirement, only the nearest ground truth for each prediction is considered: given a prediction $p = [p_s, p_e]$, where p_s is the start index of the predicted answer and p_e is the corresponding end index, and a set of associated ground truths $G = \{[g_s^{(0)}, g_e^{(0)}], \dots, [g_s^{(n)}, g_e^{(n)}]\}$, the selected ground truth is $\min_{g \in G} \|p - g\|_2$.

Moreover, we noticed that the training set does not have multiple answers for the same question, so this "nearest answer" approach comes into play only at inference time: this is actually the desired behavior, since at training time the described technique could lead to an unstable learning process, given by the non-stationarity of the target distribution.

4.5 Environment

The main third-party libraries on which the project is based on are PyTorch [14], an open source machine learning framework with automatic differentiation and eager execution, and

HuggingFace [23], a library to build, train and deploy state of the art NLP models.

All of the training and validation processes were executed on Google Colaboratory [6], a platform that gives the possibility to exploit some computational resources for free. In particular, Colab allows you to select a GPU runtime, that boosts the training time of neural models and most of the times we were assigned an NVIDIA Tesla T4 GPU, with 16GB of RAM.

Training and evaluation metrics, along with model checkpoints and results, are directly logged into a W&B project [2], which is openly accessible here.

4.6 Results

Table 4.1 shows the best results obtained in the training, validation and testing phases of each model, with the hyperparameters listed in table 4.2. In particular, by "training" we mean a learning run over the whole dataset (training and validation splits combined together), by "validation" we mean the evaluation of a model on the validation split described in 4.1, while "testing" represents metrics obtained on the official SQuAD v1.1 dev set with models trained on the whole dataset.

	Training		**Validation**		**Test**	
	F1 (%)	**EM** (%)	**F1** (%)	**EM** (%)	**F1** (%)	**EM** (%)
Baseline			37.28	26.33		
BiDAF	65.23	45.61	68.05	55.31	73.54	63.14
BERT	70.96	54.21	79.82	68.03	83.46	74.36
DistilBERT			78.82	66.17		
ELECTRA			84.19	72.23		

Table 4.1: Best results

The hyperparameters listed in table 4.2 were mainly taken from the corresponding models' paper, in order to have a well-estabilished benchmark.

	Epochs	**Batch size**	**Optimizer**	**Learning rate**
Baseline	30	128	Adam	$1e-3$
BiDAF	12	60	Adadelta	0.5
BERT	3	8	Adam	$5e-5$
DistilBERT	3	16	Adam	$5e-5$
ELECTRA	3	8	Adam	$5e-5$

Table 4.2: Hyperparameters

5 Analysis of results

6 Discussion

Bibliography

[1] Meraldo Antonio. *An Illustrated Guide to Bi-Directional Attention Flow (BiDAF)*. URL: https://towardsdatascience.com/the-definitive-guide-to-bi-directional-attention-flow-d0e96e9e666b.

[2] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from the site wandb.com. 2020. URL: https://www.wandb.com/.

[3] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators*. 2020. arXiv: 2003.10555 [cs.CL].

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. cite arxiv:1810.04805Comment: 13 pages. 2018. URL: http://arxiv.org/abs/1810.04805.

[5] Yuan Gao and Dorota Glowacka. "Deep Gate Recurrent Neural Network". In: *Proceedings of The 8th Asian Conference on Machine Learning*. Ed. by Robert J. Durrant and Kee-Eung Kim. Vol. 63. Proceedings of Machine Learning Research. The University of Waikato, Hamilton, New Zealand: PMLR, 2016, pp. 350–365. URL: http://proceedings.mlr.press/v63/gao30.html.

[6] Google. *Colaboratory*. URL: https://colab.research.google.com/.

[7] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [stat.ML].

[8] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[9] Rani Horev. *BERT Explained: State of the art language model for NLP*. URL: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270.

[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. "Distributed Representations of Words and Phrases and their Compositionality". In: *NIPS*. Curran Associates, Inc., 2013, pp. 3111–3119. URL: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[11] Nikolai Morin. *What's the difference between "hidden" and "output" in PyTorch LSTM?* URL: https://stackoverflow.com/questions/48302810/whats-the-difference-between-hidden-and-output-in-pytorch-lstm.

[12] Yoshiki Niwa and Yoshihiko Nitta. "Co-Occurrence Vectors from Corpora vs. Distance Vectors from Dictionaries". In: *Proceedings of the 15th Conference on Computational Linguistics - Volume 1*. COLING '94. USA: Association for Computational Linguistics, 1994, 304–309. DOI: 10.3115/991886.991938. URL: https://doi.org/10.3115/991886.991938.

[13] Do-Hyoung Park and Vihan Lakshman. *Question Answering on the SQuAD Dataset*. Tech. rep. Stanford University.

[14] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[15] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://www.aclweb.org/anthology/D14-1162.

[16] Thilina Rajapakse. *Understanding ELECTRA and Training an ELECTRA Language Model*. URL: https://towardsdatascience.com/understanding-electra-and-training-an-electra-language-model-3d33e3a9660d.

[17] "TF–IDF". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 986–987. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_832. URL: https://doi.org/10.1007/978-0-387-30164-8_832.

[18] Victor Sanh. *Smaller, faster, cheaper, lighter: Introducing DistilBERT, a distilled version of BERT*. URL: https://medium.com/huggingface/distilbert-8cf3380435b5.

[19] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *CoRR* abs/1910.01108 (2019). arXiv: 1910.01108. URL: http://arxiv.org/abs/1910.01108.

[20] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. "Bidirectional Attention Flow for Machine Comprehension". In: *CoRR* abs/1611.01603 (2016). arXiv: 1611.01603. URL: http://arxiv.org/abs/1611.01603.

[21] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. "Highway Networks". In: *CoRR* abs/1505.00387 (2015). arXiv: 1505.00387. URL: http://arxiv.org/abs/1505.00387.

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[23] Thomas Wolf et al. "HuggingFace's Transformers: State-of-the-art Natural Language Processing". In: *CoRR* abs/1910.03771 (2019). arXiv: 1910.03771. URL: http://arxiv.org/abs/1910.03771.

[24] Mike Schuster Yonghui Wu et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *CoRR* abs/1609.08144 (2016). arXiv: 1609.08144. URL: http://arxiv.org/abs/1609.08144.