Alma Mater Studiorum · University of Bologna

SKA Data Challenge[1]

Deep Learning course final project

Alice Zandegiacomo (alice.zandegiacomo@studio.unibo.it)
Lorenzo Cellini (lorenzo.cellini3@studio.unibo.it)

September 4, 2021

[1]The code for the project is publicly available on GitHub

Contents

1 Summary

The tasks of object detection and classification have gained significant popularity over the past years within the deep learning and computer vision communities. Systems trained end-to-end now achieve great results on a variety of tasks in the video and image domains.

In this work, we address an object detection and classification problem on the Square Kilometer Array Dataset (SKADC1) [20]: given a large image, of about 32,000 pixels on each side and 4GB in size, the goal is to detect astronomycal sources and classify them among three possible classes.

This project focuses on the SKADC1 dataset, in particular on the 560MHz-1000h high S/N sky image, that contains more than 19 000 radio sources.

Among the state of the art types of network, we opted for a two-stage system, specifically a Faster R-CNN.

In this work we implemented and compared three different models:

1. B16: a Faster R-CNN with a naive feature extraction backbone with only 4 convolutional layers and with a receptive field of 16 on the last convolutional layer, that will act as our baseline model;
2. B44: a Faster R-CNN with a feature extraction backbone with 7 convolutional layers and a receptive field of 44 on the last convolutional layer;
3. A Faster R-CNN with a larger backbone, specifically we implemented a VGG16 backbone, COMPLETAQUI [19]

Each model consists of the same input and output (Region Proposal Network + Detector) structure, and what changes is the deepness of the feature extraction network (usually called backbone).

In training the listed models, we adopted a transfer learning technique, because it has been proven to be effective in speeding up the training phase. More specifically, we applied transfer learning only on the very first layers in order to inherit and retain the more basic feature, while letting the model learn deeper representation of them.

So each model is initialized with the VGG16 public weights, then we freezed the first four convolutional layers and let deeper ones trainable.

While this could seem an easy object detecion and classification problem, it turns out that it is an hard task on both the goals:

- the objects to be located are very small, as we will show later, and
- the dataset is extremely unbalanced with respect to class distribution.

For this two reasons we developed more than one model and we made some choices that will be discussed later.

Our experimental evaluations show that the best model is ... Such results are on par with those reported in the corresponding models' papers.

2 Background

– QUI DARE LE DEFINIZIONI DI YOLO SSD E FASTER RCNN + DESCRIZIONE DETTAGLIATA DEI MODULI DI FASTER RCNN –

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 [16] and PPMI [11], and the modern dense embeddings, such as Word2Vec [9] and GloVe [15]. In the latter case, embeddings for each word in the input vocabulary are usually computed with shallow encoders.

These embeddings are then used as inputs for models specialized in processing sequences. Nowadays, such models are mostly based on the Transformer architecture [21], 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 [7] and GRUs [4]. 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.

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).

While Transformers may always seem the way to go, since they throw away all the computational

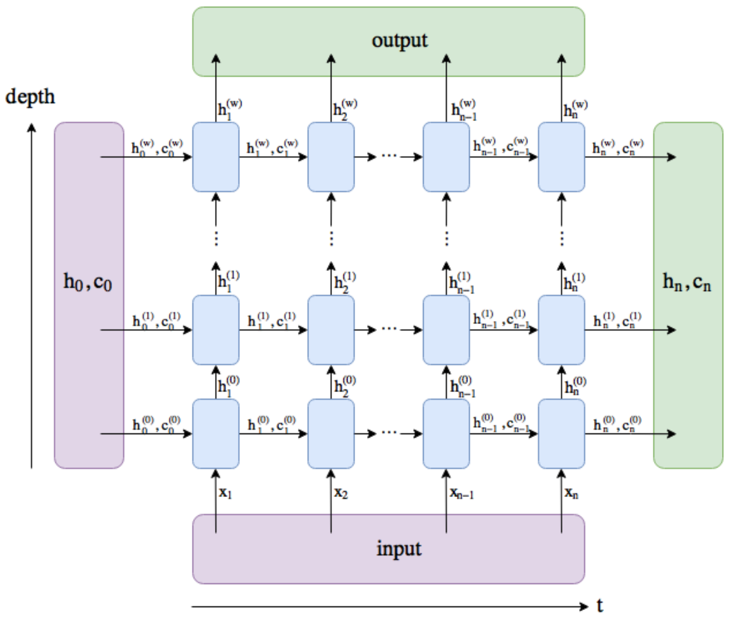

Figure 2.1: Recurrent module (image courtesy of [10])

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).

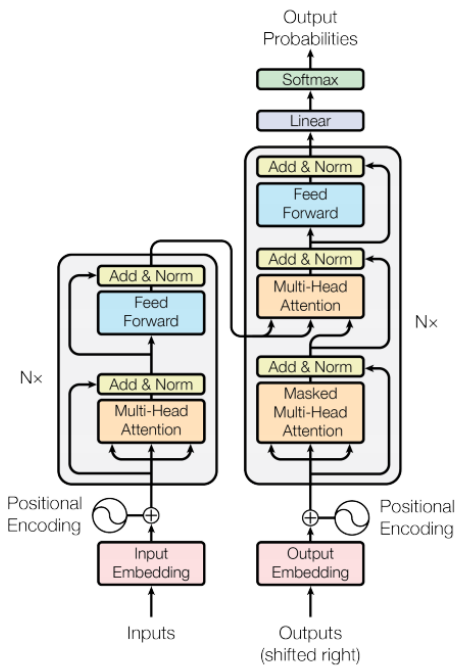

Figure 2.2: Transformers (image taken from [21])

3 Dataset description

– qui dire anche che le immagini vengono genrate on the fly e quindi anche l'augmentation, se accesa, viene fatta otf

4 System description

In this work, we implemented three different models. All of them are a Faster R-CNN network, but they differ on the backbone and hyperparameters.

In order to perform an effective comparison, the models share the same output architecture.

The following is a detailed description of the models. Beware that in each model section we report everything but what happens in the output module, which is described once in 4.3.

4.1 Feature Extraction Backbone

– aggiungere citazioni ovunque

4.1.1 B16 - Baseline 16

The first backbone we implemented has the following structure:

1. Block 1: consists of 2 convolutional layers with 3x3x64 filters, followed by a ReLU activation layer and a 2x2 MaxPooling layer with stride 2;
2. Block 2: consists of 2 convolutional layers with 3x3x128 filters, followed by a ReLU activation layer and a 2x2 MaxPooling layer with stride 2;

Both the blocks have been initialized with the public available VGG-16 weights and **freezed**.

It has been shown that, in computer vision tasks, the very first layers of a convolutional pipeline, learn the most basic features, as strokes or circles. So, given that the objects we have to detect have simple shapes, like circles or ellipses, and that we think these kind of basic feature are common across different domains, we tried to transfer learning from the VGG-16 and freezing the first layers, the ones that learn most basic features.

As mentioned before, the goal was to detect very small objects, with respect to the dimensions usually detected in other deep learning works. So we decided to crop the original 32kx32k pixels image into 20x20 pixels patches and rescale them to 100x100 pixels, before feeding the network. This resemble to zoom-in the image, so that the objects to detect become 5 times bigger.

As we've seen before, the bounding box shapes distribution shows that the 99% of the objects to detect is smaller than 18x18 pixels and the 90% is smaller then 8x8 pixels, before rescaling. After rescaling these values become: 99% smaller than 90x90 pixels and 90% smaller then 40x40 pixels.

So, with the B16 model, we wanted to test how a backbone with a receptive field smaller than the objects to detect performs on this task. Indeed the receptive field of B16 backbone (the feature extraction part) is 16, with a stride of 4. So, given an input image of size 100x100px, the last layer feature maps have a size of 25x25.

In this model we used the follwing values for anchor sizes and ratios:

- anchor size: [4, 8, 16, 24, 32, 64]
- anchor ratio: [2: 1, 1: 2, 1:1]

4.1.2 B44 - Baseline 44

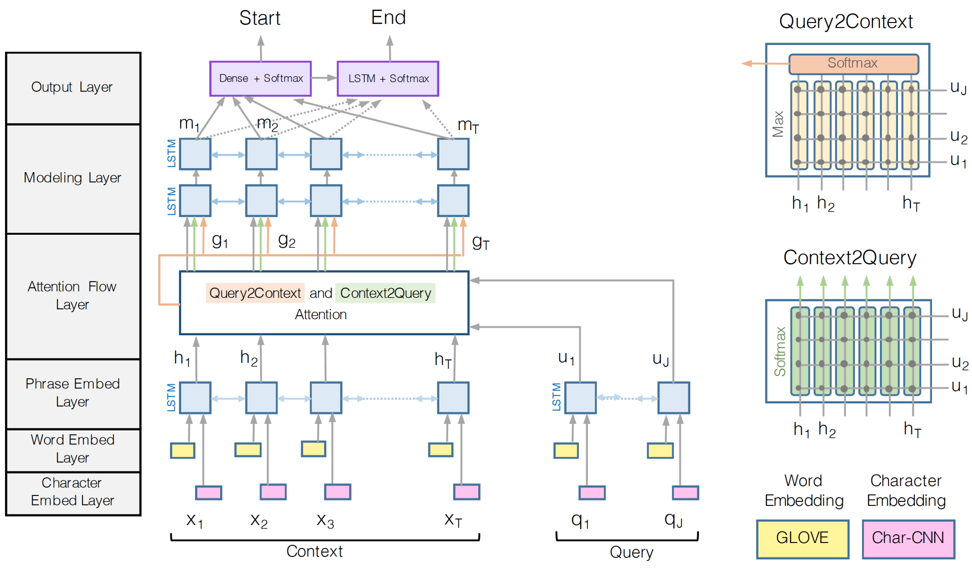

Figure 4.1: BiDAF model (image courtesy of [19])

4.2 Transformer-based models

4.2.1 BERT

All the Transformer models are based on BERT [3], which is a language model composed by a multi-layer bidirectional encoder-decoder structure [21]. Our experiments are all performed using the base, uncased implementation of BERT, which has 12 Transformer layers and 768 as its hidden size.

The peculiarity of its Transformer architecture allows the model to learn the context of a word based on all of its surroundings, at the same time, without having to perform a sequential scan of the input, as done in standard RNNs. This is reflected in its pre-training, which is composed of two tasks:

- Masked Language Modeling (MLM): at a high level, it is performed by masking 15% of the words in each sequence and replacing them with a [$MASK$] token, so that the model is required to predict the original value of the masked words, using the available ones
- Next Sentence Prediction (NSP): achieved giving as input to the model a pair of sentences (formatted as in 5.2), which are in 50% of the cases two subsequent sentences or a random picked pair otherwise, so that the model is asked to identify if the second sentence follows the first one in the pair

The first task, MLM, is used to gain a detailed understanding of the processed language, while the second task, NSP, is used to let the model have a broader comprehension of semantic connections.

BERT is trained using both tasks together, with the goal of minimizing the combined loss.

4.2.2 DistilBERT

BERT performances, as for the majority of large-scale language models, are inevitably intertwined with their number of parameters. DistilBERT [17, 18] tries to reduce this number using a technique called distillation, which consists in training a smaller "student" network to mimic the full output of the big "teacher" model [6]. Specifically, DistilBERT has about half the parameters of BERT and retains 95% of its performance on the GLUE benchmark [22].

4.2.3 ELECTRA

ELECTRA (Efficiently Learning an Encoder that Classifies Token Replacements Accurately) [2] has the same architecture as BERT, but uses a new pre-training approach which aims to outperform the MLM strategy with RTD (Replaced Token Detection). Its structure is composed of three steps, shown in figure 4.2:

1. Given an input sequence, randomly replace tokens with a [$MASK$] token
2. The generator (that performs 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), whose objective is to identify if each token is part of the original sequence or if it has been replaced by the generator

At the end of training the generator is not useful anymore, so that only the discriminator needs to be saved. This new pre-training task allows the model to compute the loss over each and every input token (not only on the masked ones, as done in BERT), which is what sets apart ELECTRA in terms of convergence speed and performance on downstream problems.

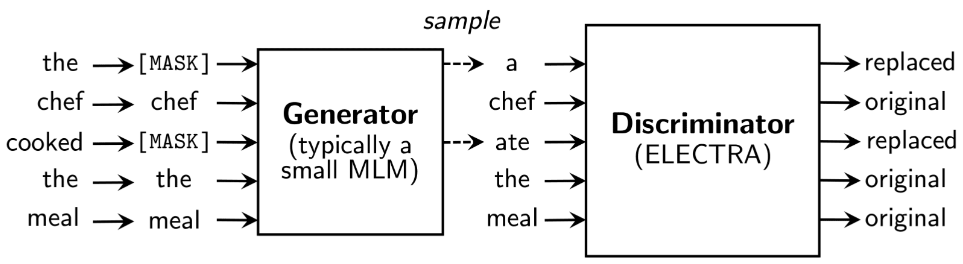

Figure 4.2: ELECTRA RTD pre-training task (image courtesy of [2])

4.3 Output module

The output module is composed by two fully-connected layers, used to identify the start and end token indexes of the answer in the given context. The start index classifier directly receives the output of the previous modules (e.g. in the BiDAF model that would correspond to the output of the modeling layer), while the same output vector is first passed into a single layer recurrent encoder before entering into the end index classifier. This additional RNN is used to implicitly enforce the constraint that the end index should follow the start one.

We then compute a softmax over both logits, so as to obtain a start and end index probability for each token in the context. To find the correct section of context that forms the answer, we compute the joint probability distribution (by multiplying each pair of start/end probabilities, for each token) and we take the start/end indexes corresponding to the maximum combined probability (respecting the assumption that the start index must precede the end one).

Finally, thanks to HuggingFace's tokenizers [8], we are able to simply convert start/end indexes over tokens to their corresponding values over original characters' positions.

The output layer is also responsible for the computation of the loss, which is a simple mean of cross-entropy losses of the start and end token probabilities.

5 Experimental setup and results

5.1 Data handling

Given the JSON training set, we decided to parse it into a suitable tabular data structure, which turned out 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, models are re-trained on the whole dataset (by merging training and validation splits), so as to simulate the cross-validation strategy. About re-training, results showed that this process only gives marginal improvements at the expense of a much more difficult way of choosing the best weights snapshots (since there are no validation metrics to observe, thus leaving only training loss available). Because of this, results reported in 5.6 do not show such training runs, but only their training/validation counterparts.

5.2 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 [12], 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]), normalizes unicode and foreign characters, 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 [3]) 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.

5.3 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, which is used to pad sequences to a fixed lenght, 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) [14]

   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.

5.4 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.

5.5 Environment

The main third-party libraries on which the project is based on are PyTorch [13], 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 [5], 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 [1], which is openly accessible here.

5.6 Results

Table 5.1 shows the best results obtained in the training, validation and testing phases of each model, with the hyperparameters listed in table 5.2. In particular, by "training"/"validation" we mean metrics computed on the training/validation splits described in 5.1, while "testing" represents metrics obtained on the official SQuAD v1.1 dev set.

	Training		**Validation**		**Test**	
	F1 (%)	**EM** (%)	**F1** (%)	**EM** (%)	**F1** (%)	**EM** (%)
Baseline	36.21	21.27	37.19	26.47	38.30	27.15
BiDAF	63.42	43.27	68.40	55.31	71.15	60.09
BERT	73.68	56.14	80.28	67.80	83.17	74.17
DistilBERT	73.72	55.95	79.26	66.97	82.44	73.64
ELECTRA	**76.42**	**58.89**	**84.45**	**71.83**	**88.27**	**80.60**

Table 5.1: Best results

The hyperparameters listed in table 5.2 were mainly taken from the corresponding models' papers, in order to have a well-estabilished benchmark, even though some of them were adjusted based on available resources (e.g. the batch size).

	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 5.2: Hyperparameters

6 Analysis of results

For the results analysis, we focused on misclassified examples on the SQuAD v1.1 dev set. In particular, table 6.1 shows some errors (defined as those answers whose EM is zero) that are common in both of our best architectures, i.e. BiDAF and ELECTRA.

The total number of errors with BiDAF is exactly 4222, while ELECTRA halves this metric to 2062. Instead, the amount of common errors, i.e. the questions whose predicted answer is wrong with both models, is equal to 1535 (almost covering all of ELECTRA's errors).

By analyzing the common errors, we can focus on the hardest examples in the dataset. In order to categorize such misclassifications, we manually observed a random subset of 50 elements and reported (in table 6.1) the most frequent and interesting patterns.

Error type	**Example**
Word boundaries (answers that are correct but are either too specific or too generic)	**Context**: In cases where the criminalized behavior is pure speech, civil disobedience can consist simply of engaging in the forbidden speech. An example would be WBAI's broadcasting the track "Filthy Words" from a George Carlin comedy album, which eventually led to the 1978 Supreme Court case of FCC v. Pacifica Foundation. Threatening government officials is another classic way of expressing defiance toward the government and unwillingness to stand for its policies. For example, Joseph Haas was arrested for allegedly sending an email to the Lebanon, New Hampshire city councilors stating, "Wise up or die." **Question**: What is one criminal behavior that is hard to stop by authorities? **Answers**: [pure speech, forbidden speech, engaging in forbidden speech] **Predictions**: [ELECTRA] speech [BiDAF] civil disobedience can consist simply of engaging in forbidden speech

Error type	**Example**
Question comprehension (answers that clearly show one of the models was not able to fully capture the meaning of the question)	**Context**: Although Kenya is the biggest and most advanced economy in east and central Africa, and has an affluent urban minority, it has a Human Development Index (HDI) of 0.519, ranked 145 out of 186 in the world. As of 2005, 17.7% of Kenyans lived on less than $1.25 a day. The important agricultural sector is one of the least developed and largely inefficient, employing 75% of the workforce compared to less than 3% in the food secure developed countries. Kenya is usually classified as a frontier market or occasionally an emerging market, but it is not one of the least developed countries. **Question**: What is Kenya's HDI? **Answers**: [0519 ranked 145 out of 186 in world, 0519] **Predictions**: [ELECTRA] human development index [BiDAF] human development index
Antonyms (answers for which one of the models predicted the exact opposite meaning of the correct one)	**Context**: Luther secretly returned to Wittenberg on 6 March 1522. He wrote to the Elector: "During my absence, Satan has entered my sheepfold, and committed ravages which I cannot repair by writing, but only by my personal presence and living word." For eight days in Lent, beginning on Invocavit Sunday, 9 March, Luther preached eight sermons, which became known as the "Invocavit Sermons". In these sermons, he hammered home the primacy of core Christian values such as love, patience, charity, and freedom, and reminded the citizens to trust God's word rather than violence to bring about necessary change. **Question**: How did Luther want people to bring about change? **Answers**: [trust gods word, love patience charity and freedom] **Predictions**: [ELECTRA] violence [BiDAF] reminded citizens to trust gods word rather than violence
Mathematical operations (answers that show models' limitations in producing sound outputs when math operations are expressed in natural language)	**Context**: In July 1888, Brown and Peck negotiated a licensing deal with George Westinghouse for Tesla's polyphase induction motor and transformer designs for $60,000 in cash and stock and a royalty of $2.50 per AC horsepower produced by each motor. Westinghouse also hired Tesla for one year for the large fee of $2,000 ($52,700 in today's dollars) per month to be a consultant at the Westinghouse Electric & Manufacturing Company's Pittsburgh labs. **Question**: How much did Westinghouse pay for Tesla's designs? **Answers**: [60000 in cash and stock and royalty of 250 per ac horsepower produced by each motor, 60000 in cash and stock and royalty] **Predictions**: [ELECTRA] 60000 [BiDAF] 60000

7 Discussion

In this work, we addressed the problem of question answering on the SQuAD v1.1 dataset, by leveraging both recurrent-based models and Transformer-based ones. Our Baseline is outperformed by every other architecture, as expected, and models based on BERT are able to almost reach human level performance, when evaluated on the F1 score metric (for reference, that is around 90%). It's worth to notice that current SoTA models are all based on ensembling techniques, even though the gap with good single models is getting smaller and smaller.

We mainly worked on implementing custom recurrent-based models, such as the BiDAF network, and on giving the same output interface to each and every model, so as to gain modularity and fairness in the evaluation metrics of the backbone.

Evaluation was done on the training and validation splits of the whole dataset and on a separate test set, which is the official SQuAD v1.1 dev set: our results are on par with those reported in the literature.

Further interesting analysis could be performed on the produced outputs, to reach a better understanding of the implemented models. Anyway, conducted experiments show that both recurrent-based and Transformer-based models perform similar errors, such as those on word boundaries and question comprehension. The total number of errors, in the SQuAD v1.1 dev set, that are common in every model is exactly 940.

Future improvements could be related to including character embeddings in the BiDAF architecture (as done in the original paper), using a better fine-tuning procedure for Transformer-based models (as described in 1) and handling OOV words with more sophisticated procedures for recurrent-based models. Other things to try are the following: stacking a BiDAF architecture on top of a BERT one or simply modifying the output layer to include other modules, like convolutional ones.

Bibliography

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

[2] 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].

[3] 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.

[4] 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.

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

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

[7] 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.

[8] HuggingFace. *HuggingFace tokenizers*. URL: https://huggingface.co/docs/tokenizers/python/latest/.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] Jeffrey Pennington. *Unknown word tag*. URL: https://groups.google.com/g/globalvectors/c/9w8ZADXJclA/m/hRdn4prm-XUJ?pli=1.

[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] "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.

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

[18] 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.

[19] 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.

[20] SKA. *SKA web site*. URL: https://www.skatelescope.org/news/ska-launches-science-data-challenge/.

[21] 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.

[22] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". In: *CoRR* abs/1804.07461 (2018). arXiv: 1804.07461. URL: http://arxiv.org/abs/1804.07461.

[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.