NL1 PROJECT

Fake News Detection on FPGA

Matteo Santelmo

2021

1 Introduction

After the global pandemic of COVID-19 the world has experienced a massive growth of fake news spread, initially about the virus, and reaching the vaccines in the later period. This brief introduction wanted to show a real life scenario in which a tool like the one this project aimed to create would have been useful. The project can be seen as divided in two different steps: creating the neural network that does the actual detection, and then implementing the model on a FPGA.

In the first place, a method to detect fake news was needed, and the choice was to build a deep learning model able to analyze the text of the news considered. The machine learning approach is perfect for our purpose, since it would be pretty difficult to create an algorithm to realize the detector, while a machine learning model would learn how to solve this problem by leveraging data. A problem to be solved before choosing a specific machine learning model was to understand how to represent the data as well as which features of the articles we wanted to analyze. As you will see in the next section the choice was to create a style-based fake news detection, easier to realize compared to propagation-based detectors, which examines the way fake news spreads, for example analysing aspects like the number of shares and how this value increases in time. In the second place the model would be deployed onto an FPGA (Field-Programmable Gate Array), an integrated circuit designed to be programmable in its logic and elaboration features. The benefits of using an FPGA device, especially when working with machine learning models, are the significantly higher speed of computation, thanks to the AI acceleration, and a better energy efficiency.

2 Machine Learning Model

2.1 Data pre-processing

The dataset used to train the neural network described in section 2.2 is a table that collects around twenty thousand articles, each with its title, author and text, with an additional column for the boolean label that tells us whether that news is reliable or not. This dataset can be found on Kaggle [1]. For efficiency related reasons, the model will analyse only the titles, so all the following steps will refer implicitly to the title and not to the entire article. A more accurate final result could probably be obtained by considering the entire text, but this would significantly increase the time required for both data preprocessing and model training. The first step required to build a neural network is to find a way to represent the data in a way that the network can understand. In this case, since we have to analyze natural language data, certain techniques are required in order to process the data and make it more understandable to our machine; such techniques fall under the definition of NLP (Natural Language Processing). The particular one chosen in this case resorts to the stemming procedure, i.e. the process of reducing words to their word stem or root form. A simple example can give a better understanding of how the stemming process works: let's suppose we give a stemmer algorithm the words fishing, fish, fisher and fishes, then our result would be the stem fish. Once the stemmed data is obtained, another step is to filter out words that are very recurrent and useless to the analysis, the so called stop words, like "a", "is", "in" and many others included in the list StopWords of NLTK (Natural Language Toolkit), the toolkit used in the code to complete this operation. Filtering out stop words from natural language is a very common technique, used for example by Internet search engines to simplify the research process. Now, the next step is to transform the titles, that still are lists of words, into numerical arrays. Firstly the words are passed to the so-called One Hot function, that creates a vector of a given size, which will contain only zeros, except for a specific position, distinctive for that particular word, where we will find a 1. The index of the position of the only 1 in the vector is an integer number, specific and univocal for every single word. The final step consists of creating for every title the corresponding array of integer values, derived by the One Hot encoding just discussed. Now the data has been processed and is ready to be passed through the model.

2.2 Model description and training

Once the data analysis and processing has been completed, it is necessary to figure out what kind of model to build, in order to achieve a better result. In this project, a simple neural network has been chosen, despite the fact that an LSTM-based network, where LSTM stands for Long Short-Term Memory, which is a particular kind of RNN i.e. Recurrent Neural Network, would have performed slightly better. This choice has been influenced by the second step of the work, that is the implementation on FPGA. Indeed, the tool that will be used to realize this second part, Vitis AI, does not support the LSTM layer yet. In any case, in the repository of this project [2] you can find in the form of a python notebook, both the code of the model discussed and the LSTM-based one, which does perform slightly better than the one we will talk about, in terms of accuracy. Finally, we can get to the model, realized using TensorFlow, which is developed as a sequence of layers, where every layer realizes a specific mathematical function.

The first layer used is an embedding layer, often fundamental in the development of natural language related applications. What this kind of layer does is create a representation for text, where words with similar meanings will have similar representations. Every word is represented as a vector where every value is a float one, ranging from -1 to +1, and indicates how close that word is to a specific characteristic. For a better understanding of word embedding operation, an example is given in figure 1. This process is fundamental for our kind of application because it makes it possible to detect schemes in sentences, linking different words that have similar meaning or refer to similar subjects. It is important to note that, since the embedding is part of the model, the analysed features are subject to the learning process, that means they are not defined and static. After the model has learned how to represent words through embedding, the data is passed to the next layers. Firstly, a max pooling layer is needed to reduce the dimension of data and consequently the computational load, but also to filter data by taking maximum values over certain areas. The following layer is a dense one, which means that neurons of this layer are fully connected to neurons of the previous and the next layer. The last layer is composed by a single neuron which is actually a float number, ranging from 0 to 1, that represents the actual output.

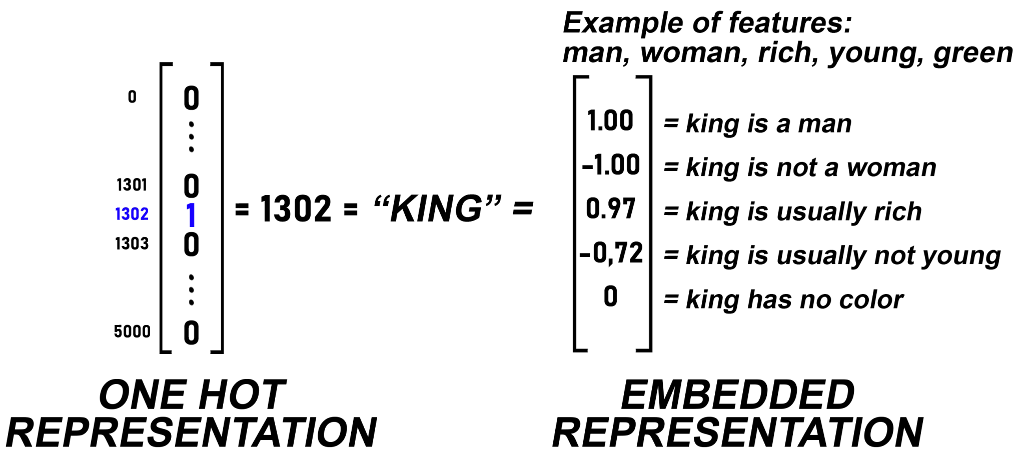

Figure 1: Convertion of the word "King" to its embedded representation. Every float value in the output vector express on a -1 to 1 range, how much the word analysed true to that feature. If the word were "Queen" the final result would have been different only for the first two features.

Once defined, the model is compiled and then trained using 67% of the data previously prepared. The training process means that the data passes through the model so that, comparing the output to the label given in the dataset, the machine can recalibrate the parameters of all the layers, in order to get a result that is closer to the desired one. This particular process, reiterated in this case only 10 times, is the core of machine learning functioning. Using the remaining 33% of the data, the accuracy of the model is measured, comparing the predictions made by the model to the original dataset label.

3 FPGA implementation

3.1 Vitis AI workflow

Xilinx, market leader of reconfigurable devices, provides an open-source development environment for AI inference on Xilinx hardware platforms, named Vitis-AI. Thanks to this tool it is possible to realize the complete workflow of AI acceleration on Xilinx boards, starting from the machine learning model. Generally, Vitis AI foresees three steps:

- Model development: the Vitis AI tools allows to quantize and compile the chosen model to convert it into DPU instruction files, where DPU stands for Deep Learning Processing Unit, which is a programmable engine dedicated to Deep Learning applications. These steps will be further investigated in the following section.

- Hardware development: through Vitis tool, integrating DPU with platform, it is possible to generate board boot files.

- Software development: to finish software application development and generate executable running on board.

As far as this application is concerned, we will only need to run through the first and last step, since a hardware build is provided by Xilinx. To realize this part of the project the Vitis AI user guide [3] has been essential, thanks to its in depth description of all the procedures.

3.2 Quantizing and Compiling

As mentioned in the introduction, FPGA platforms are capable of running deep learning applications and achieving high performance with a good energy efficiency, at the expense of a small accuracy degradation. In order to reach this goal it is necessary to simplify all the calculation load of the model and this is done by the quantization. Quantization allows the use of 8-bit integer computing units for weights and activations instead of 32-bit floating point ones, where weights and activations are no less than the key linear parameters used inside neural network layers. Another Vitis AI tool that helps in this regard is the pruning tool, which reduces the overall number of operations of the net. To help visualizing the quantization functioning a scheme is proposed in figure 2

Once the Vitis AI environment is set up it is possible to launch quantization from the command line or through a script, providing as argument a floating-point model, saved after the training process seen before, and a subset of the training data that is used for calibration. In the repository of this project [2] you can find the python script that includes the machine learning application and the quantization commands. Once the quantized model is saved it can be evaluated through the same steps seen before for the original model: model training and accuracy calculation.

The last step is converting the model to a DPU instruction sequence. This is realized by the Vitis AI compiler, a single interface to multiple compilers, designed to optimize neural network computations for different DPUs. Firstly the compiler parses the quantized model's topology, generating an intermediate computation graph. Subsequently this graph is broken into subgraphs and the compiler applies more optimizations. Finally, the optimized and compiled model is generated as a .xmodel file, taking into consideration the DPU microarchitecture chosen, in our case the architecture of a Xilinx ZCU104 board.

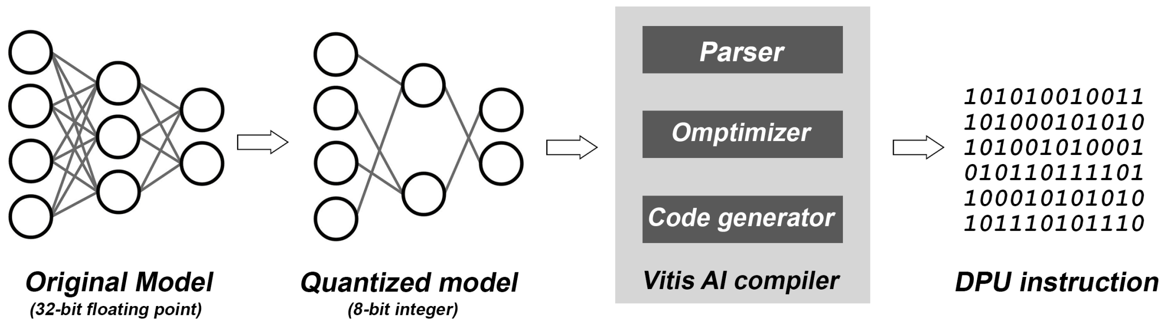

Figure 2: An overview scheme to visualize the process, starting from the original model, through quantization and compilation, up to the DPU instruction sequence.

Since FPGA are designed mainly for the implementation of convolutional network application, the embedding layer used in our model and fundamental in natural language related applications, is not supported by the Vitis-AI compiler. As a result of this incompatibility, in order to complete the process and generate the executable file for the DPU, the model has been changed and the embedding layer has been removed. The new model, that you can find in the repository [2], is composed of three dense layers, where the neurons are fully connected.

4 Results and conclusion

The final results of this project should be considered keeping in mind the incompatibility discussed in the previous section, since the two neural networks, one with word embedding and the other without this feature, are very different in terms of final accuracy. Numerically speaking, the first model, that is the one described accurately in section 2.2, has reached an accuracy outcome, measured on a different bunch of data from the one used during model training, around 92% in only 10 epochs of training that required no more than 1 minute on a mid-range laptop. The quantization of the model then drops this result to 88Moving to the model without word embedding, the results are less acceptable: both the accuracy values before and after the quantization, that lowers the precision mainly on complex systems so not in this case, are approximately around 75%. Comparing the two results it becomes clear the importance and the potential of word embedding in this type of task. Talking about future possible developments of this project, an interesting task could be redesigning the model, in view of FPGA implementation, as a convolutional neural network. These types of networks, frequently used for image recognition and classification, are designed for a better analysis and detection of certain schemes and patterns in the data, so that it would be interesting to see how well a network like this could perform in style-based recognition of fake news. A model like this would be interesting also in terms of its implementation on FPGA since these architectures are ideal for CNN models, as can be seen in the plenty examples available on Vitis AI repository [4]. Talking about real life application of the tool created in this project, this fake news classifier could be used in social media applications to detect possible unreliable news and warn the user to be wary of what he or she is reading, and maybe check it through a different source.

References

[1] Kaggle dataset

[2] GitHub repository of this project

[3] Xilinx Vitis AI user guide

[4] Xilinx Vitis AI GitHub repository