Università
della
Svizzera
italiana

Facoltà
di
scienze
informatiche

Edge Computing in the IoT

Machine Learning on the Edge

Alberto Ferrante
TA: Luca Butera

Models and Weights

Machine Learning is a broad topic that covers many approaches and models.

In recent years **Neural Networks** trained via **Gradient Descent** have gained popularity as a flexible set of techniques to solve a broad range of tasks.

Neural Networks are, in simple words, **huge parametric functions** whose parameters we can tune until we find the right spot.

State of the Art models for language and vision tasks reach the **billions of parameters**.

On most IoT devices **we don't have enough memory nor computational power** to run such models.

How can we try to **reduce these models' size** without losing too much performance?

Quantization

The aim of **quantization** is to train models that occupy less space in memory by using **smaller numeric types** to encode the model's weights.

This may end in a **degradation of the performance**, as the quantized model may not have the required expressive power for the task.

Quantization can be carried out **after training**, but this results in the need of **fine-tuning** to return to the original performance.

Quantization-aware training aims at removing the need for fine-tuning by **directly training the quantized weights**.

Go to:
https://www.tensorflow.org/model_optimization/guide/quantization/training_example
To see an example of quantization-aware training in TensorFlow

Pruning

The aim of **pruning** is to reduce the number of parameters of a model by removing the ones that are not necessary.
Magnitude-based pruning puts to zero parameters that are really small compared to others, this allows for **better model compression** and reduces both memory occupation and latency.

Also pruning may **degrade the performance** of the model, and **fine-tuning** is usually required as a final step.

Pruning is carried out **after training**.

Go to: https://www.tensorflow.org/model_optimization/guide/pruning/pruning_with_keras
To see an example of magnitude-based pruning in TensorFlow

Other techniques

Other model optimization techniques exist:

- **Weight Clustering:** consists in finding clusters of similar valued weights and putting their value to the value of the cluster's centroid. Helps with model compression and memory footprint. https://www.tensorflow.org/model_optimization/guide/clustering

- **Knowledge Distillation:** consists in training a small student model to emulate the behavior of a bigger teacher model. More advanced and complex technique; can reduce both latency and memory footprint. https://keras.io/examples/vision/knowledge_distillation/

Inference on Arduino

To run our model on arduino we use the EloquentTinyML library available at:
https://github.com/eloquentarduino/EloquentTinyML
You can install it from the Arduino IDE Library Manager searching **EloquentTinyML.**

Under **File -> Examples -> EloqeuntTinyML -> 2.4 -> SineExample** you can find an example of how to use this library to load a model and perform inference directly on an Arduino board.

To use a model with this library we first need to convert it to a C-array, you will find the code for a function to do so on iCorsi.

```
# this will convert the model and save it to “name.h”
c_model = model_dump(tf_model, ‘name’)
```

Conversion to Micro

TensorFlow Lite for Microcontrollers does not support mixed type operations, hence, you need to specify the conversion from Keras to TensorFlow Lite in the following manner:

```
def representative_dataset():
 for i in range(100):
   yield [x[i].astype(np.float32)]
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_dataset
tfl_model = converter.convert()
```

This is a little different than what is used in the examples, and is needed to ensure all operations are carried out with integer types instead of a mix of floats and integers.

The full story

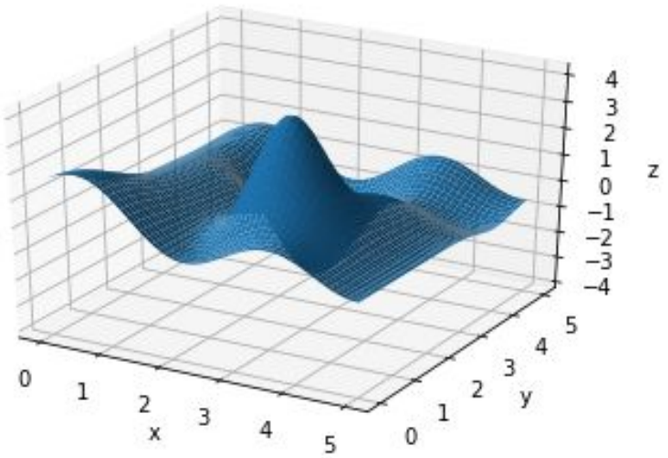
Eloquent's TinyML library **simplifies** model's running on Arduino.

Under the hood it does a series of operations that you should otherwise do manually, and may still have to perform if you need finer control over your model.

If you wish to see the full story you can check out this tutorial by Tensorflow:
https://www.tensorflow.org/lite/microcontrollers/get_started_low_level

Excercise

Try and use what we saw to train a model in Google Colab that learns to predict the *z* value, given *x* and *y*, from samples of the following function:



You can find the code to create the data and build the model on iCorsi.

- Asses the number of parameters of the model and its size in memory.
- Does it fit on the Portenta? If yes, what is the latency?
- Apply Quantization and repeat the assessment.
- Do the same for Pruning.
- Did the answers change? If yes, how?