COMP 642 — Machine Learning Apr 07, 2022

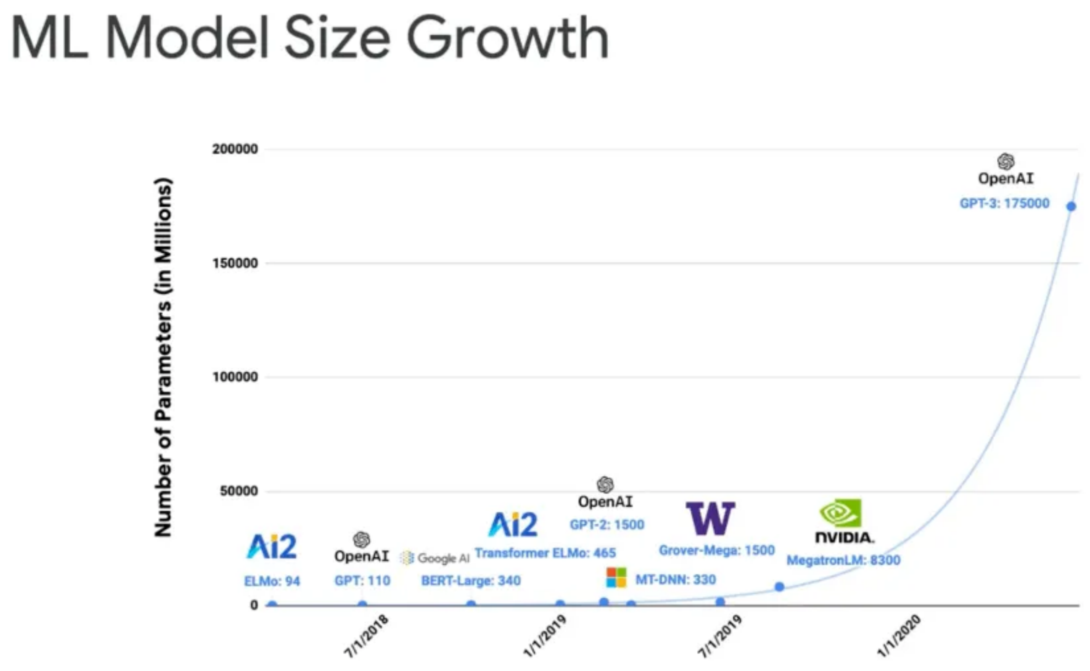
Lecture 22: Tiny ML

Lecturer: Anshumali Shrivastava
Scribe By: Aayush Chudgar, Ishtdeep Singh, Naman Mohindra, Nishant Satpathy

1 Tiny ML

TinyML is a field of study in Machine Learning and Embedded Systems that explores the types of models you can run on small, low-powered devices like micro-controllers. It enables low-latency, low power and low bandwidth model inference at edge devices. An edge device is any piece of hardware that controls data flow at the boundary between two networks. Edge devices fulfill a variety of roles, depending on what type of device they are, but they essentially serve as network entry – or exit – points. Some common functions of edge devices are the transmission, routing, processing, monitoring, filtering, translation and storage of data passing between networks.

In a world where machine learning algorithms are becoming larger and more resource-intensive, how does tinyML fit in? To put things in context, consider the chart below.



In recent years, machine learning models have grown exponentially—from less than 100 million parameters in 2018 to 530 billion parameters within the past five years to 2021! What's more, they're using more energy than ever before—according to one estimate, the carbon footprint of a single training of BERT is enough to fly round trip between New York and San Francisco. [1]

Training and deploying modern machine learning algorithms clearly has significant resource demands. Hence, most modern machine learning algorithms are deployed on relatively powerful devices and typically have access to the cloud, where the bulk of resource-intensive calculations take place. But bigger is not always better, and tinyML has an important role to play at the smaller end of things.

TinyML is basically machine learning on devices that have a small amount of storage/compute power. The need for TinyML arises from the fact that we need low latency, data privacy, low energy consumption and low bandwidth. Since the model runs on the edge, the data doesn't have to be sent to a server to run inference. This reduces the latency of the output. Since the model is running on the edge, your data is not stored in any servers. Finally, micro-controllers consume very little power. This enables them to run without being charged for a really long time.

2 Training Small Models

One of the most intuitive methods to implement tinyML is to train a small model and subsequently increase it's accuracy by retraining, hyper-parameter tuning, training on more data, using cross validation, etc. However, this method can only take you so far, you might not be able to meet the desired accuracy, model may not be maintainable(when you get more data), etc.

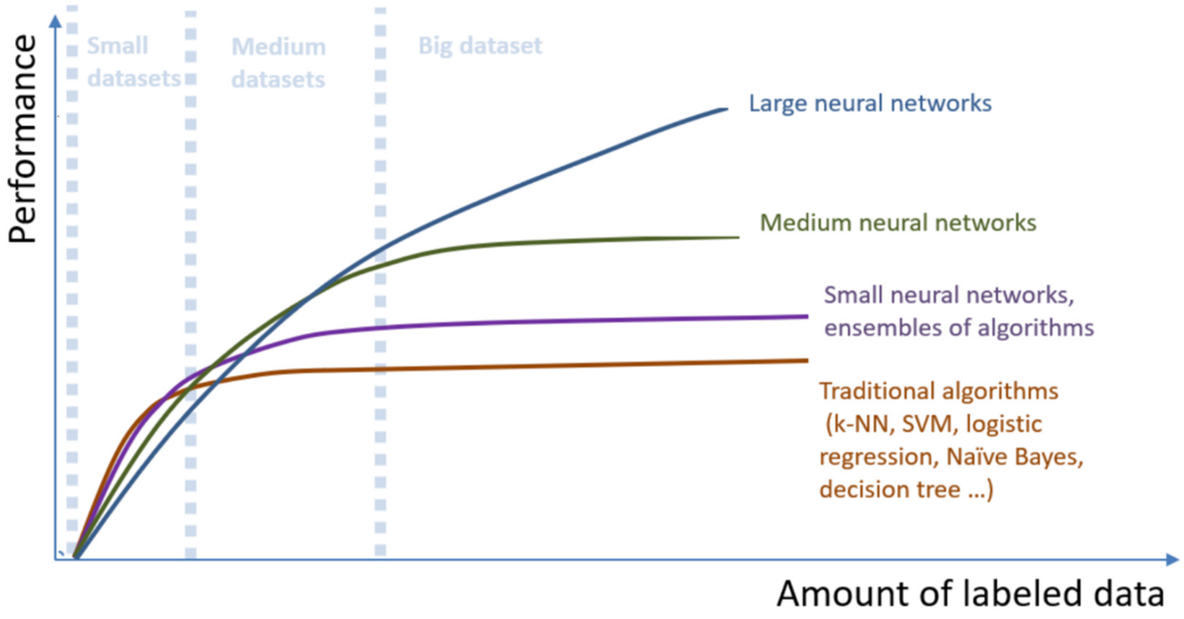
3 Pruning

It is well established that larger models are able to provide a better accuracy. Pruning is an idea that at first creates a large model that achieves the required accuracy. Then try and decrease the size (prune it) while trying to maintain the accuracy. Once the model has been pruned, this new pruned model is trained again on the dataset and this is known as iterative pruning. This is one of the most successful methods practiced in the industry. ML kind of works using the

greedy algorithm. The more data it has, the better it performs. The more complex the model is, the better it performs, usually. Because of this, big complex neural networks are giving the best accuracy in recent times. Smaller neural networks do achieve a decent accuracy but still fall short when compared to bigger models. Linear models give the lowest performance when there is a high amount of data available. On the other hand, when there is less data available, linear models give the best performance when compared to neural networks. As we had less data in the past, linear models worked best. They gave

- more accuracy
- faster speed

But now in the age of big data where datasets are in terabytes, we can use the power of neural networks and other complex models to boost the accuracy and increase performance.

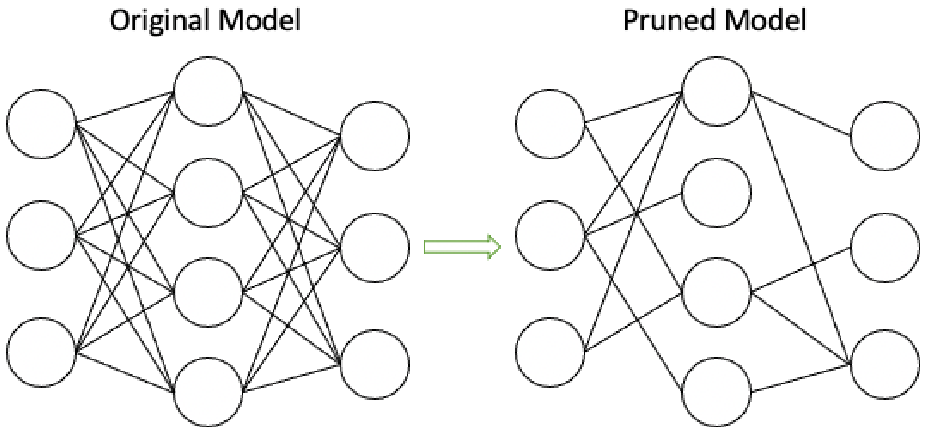


How to shrink the model?

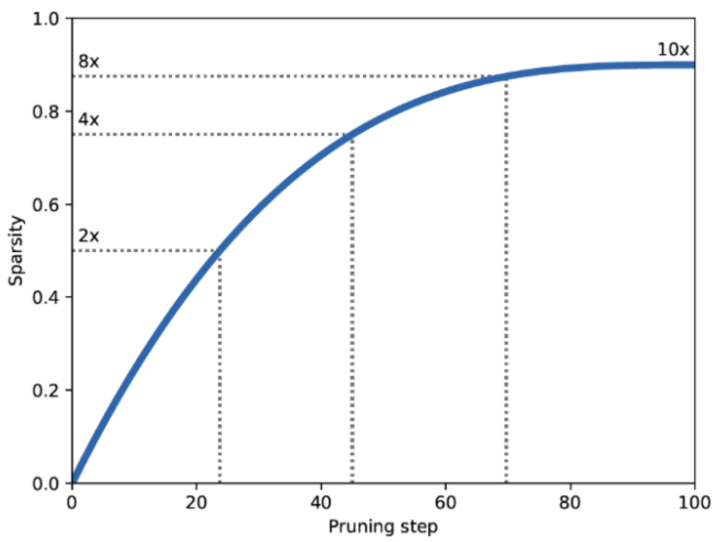
In a neural network, there are millions of parameters. We try to find the least significant parameters out of these. We then remove all of these parameters. For example, the parameter with value 0.00001 will be converted to 0 as it was not providing enough activation anyways. In this way, a small fraction of parameters are dropped from the model.

After this, the model is trained again. The remaining parameters will get retrained and change. Repeat until:

- you reach the memory budget, OR
- you reach the required accuracy.



The amount of sparsity for gradual pruning can be illustrated as a monotonic increasing function. The target sparsity can be achieved by increasing the pruning step from step 0 to 100 which achieves a 90% sparsity, as shown in the figure below.



In conclusion, pruning is a model compression technique that allows us to compress the model to a smaller size with zero or marginal loss of accuracy. In short, pruning eliminates the weights with low magnitude (That does not contribute much to the final model performance). Both original and pruned model has the same architecture, with the pruned model being sparser (weights with the low magnitude being set to zeros).[2]

4 Quantization

Quantization in Deep Learning refers to techniques which represents numbers with lower bit's than the standard 64 bit floating point precision, example 16 bit or even 8 bit numbers. This reduction in precision helps in saving memory as a 64 bit number requires 4 times the memory of a 16 bit number and 8 times that of an 8 bit number.

Quantization maps a floating point value x which has a range $[\alpha, \beta]$ (here α is the min value of x and β is the max value of x) to x_q which has a range $[\alpha_q, \beta_q]$ and usually

$$\beta - \alpha \leq \beta_q - \alpha_q$$

which means that the range of the new interval is smaller than the range of the initial interval. This is called quantization mapping [4]. Quantization has two operations, the first which is the quantization process which just converts a number into its quantized form

$$x_q = round(\frac{x}{c} - d)$$

where c and d are variables. The second process is the dequantization process which reverts a quantized number into its approximated full precision part.

$$x = c(x_q + d)$$

We can obtain c and d by solving a linear system of equations which then finally results in

$$c = \frac{\beta - \alpha}{\beta_q - \alpha_q}$$

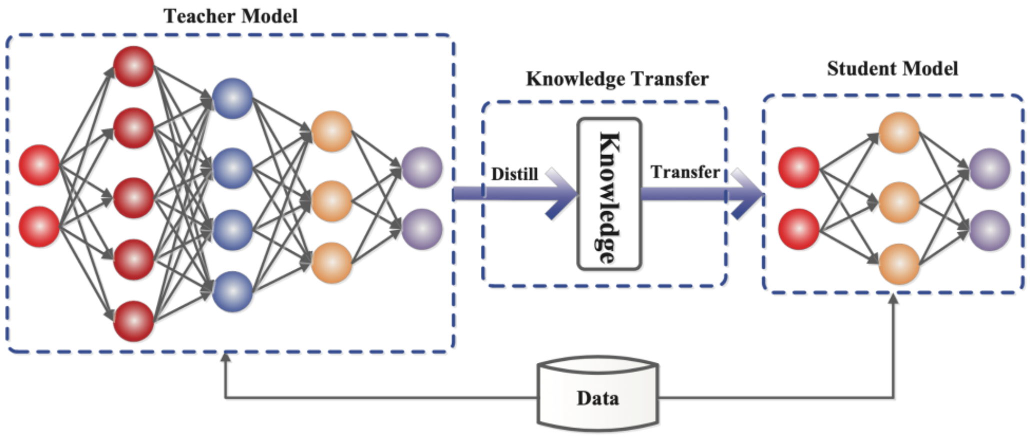
$$d = \frac{\alpha\beta_q - \beta\alpha_q}{\beta_q - \alpha_q}$$

One drawback of quantization is that there is loss of information as the precision is lost and only models can mostly do integer arithmetic due to the reduced range, but as we will see ahead that a lot of the results of training quantized neural networks suggest that the loss is negligible when compared to the degree of reduction in the model size. One good example of a quantized model would be I - BERT [3]. This is a quantized version of BERT which uses only integer based arithmetic. This technique approximates the non - linear operations such as GELU, Softmax etc. They show a speedup of up to 4 times using INT8 when compared to FP32.

5 Knowledge Distillation

This is a 'weird' technique based on the principle that if you have a lot of high quality data, a smaller machine learning model would be able to generate accurate results as well. This is an example of a student-teacher model.

Knowledge distillation works by effectively training a bigger ML model, or the TEACHER, with a given dataset. The model would generate predictions based on the given data. A new dataset would be constructed which would combine the old data and the information generated from the bigger model. This new dataset would be then fed to the smaller ML model, or the STUDENT.



[5]

So how does this work? We know that a bigger model will generate results with high accuracy. Hence, we will use this model to generate more information about the data, under the assumption that the information would be fairly accurate. The data coupled with this information would be of better quality. Hence we could make the trade off between quantity and quality and provide a smaller quantity of high quality data to the smaller model with poor accuracy and improve its performance.

A pseudo example could be as follows:
Let the original data be $\{x_i, y_i\}$ where y_i is basically 1 or 0. This data would be fed to the bigger model with higher accuracy. The model, as its outcome, would generate probability predictions for the two potential outcomes p_1 and p_2.

$\{x_i,\, y_i\} \rightarrow \{\mathrm{x}_i,\, [p_1,\, p_2]\}$

The new data generated would now be $\{x_i,\, [p_1,\, p_2]\}$ where instead of y_i being the two outcomes (0 and 1), it is the prediction probabilities of the two outcomes. This is data which is being generated from the point of view of the bigger model. This data encodes more information about the data itself and would increase the accuracy of the smaller model with lower accuracy.

References

[1] Giri. Why the big future of machine learning is tiny. https://highdemandskills.com/tinyml/, Feb 2022.

[2] Kelvin. Model compression via pruning. https://towardsdatascience.com/model-compression-via-pruning-ac9b730a7c7b, Nov 2020.

[3] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. I-bert: Integer-only bert quantization. In *International conference on machine learning*, pages 5506–5518. PMLR, 2021.

[4] Lei Mao. Quantization for neural networks https://leimao.github.io/article/neural-networks-quantization/, May 2020.

[5] Sundeep Teki. Knowledge distillation: Principles, algorithms, applications, Mar 2022.