Juan García
Distributed Systems / Cristina Abad
August 15, 2017

ANN to select memory partitioning algorithm

Companies correlate sales with the latency customers experience, for example (Shalom, 2008) correlate a 1% loss in sales for Amazon to a 100ms increase in latency. There are many ways to reduce latency (e.g. multiple layer caching (Huang et al, 2013)) but this project focuses on a server side solution: Appropriate memory allocation.

Appropriate memory allocation consists on partitioning cache memory amongst applications, in a way that maximizes hit rate and in turn minimizes latency. There exists several algorithms to determine such memory partition, e.g., Hill-climbing, LP-solver, and the algorithm proposed in (Abad et al, 2017); nonetheless their performance is dependent on the application's workload and hence there isn't an apparent best for all situations. The goal of this project is then to implement a neural network that can learn to pick the best[1] memory partition algorithm based on the workload of multiple applications.

The following sections detail an ANN that creates a fixed size embedding for the HRC of multiple applications, and uses the embedding to select the memory partition algorithm. The first section describes the architecture, the second section explains how to train it, and the last section details the next steps to take in implementation.

[1] Best is the algorithm that produces the minimum latency (maximizes hit-rate). However constraints such as computation time and allocation costs are not considered; this could change what best means.

Architecture

The dataset consists on csv files where each column corresponds to the hit-rate curve (HRC) of one application workload (see figure 1). This csv files can contain multiple HRC curves, which implies the model should account for all the curves when selecting the best algorithm.

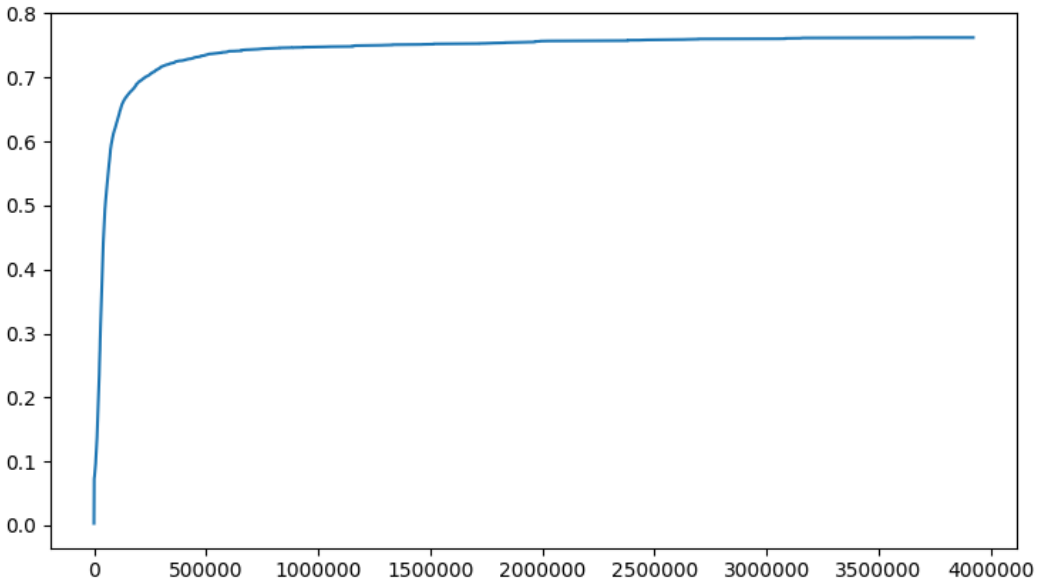

Figure 1: Sample HRC curve.

The model is a feed-forward neural network with a softmax layer connected to the output layer (This is expressed as a single “softmax layer” in the code). The neuron activation functions are sigmoid functions and the loss is a cross-entropy function. Each HRC is fed to the network independently. A max pooling layer is employed to create a fixed sized embedding for the HRCs in the same file. The embedding is then fed to the softmax layer and the classification is produced.

Note that this implementation compromise gradient correctness by not considering the stochastic loss (sum of the losses in a mini-batch); however (Keskar, 2017) and (Goodfellow, 2016) suggest this doesn't hinder the capabilities of the model to generalize.

p : Number of applications
m : Memory size
n : Hidden layer size

$$X = \{(x_1, x_2, \ldots, x_p) x_i \in \mathbb{R}^{m \times 1}\}$$
$$W \in \mathbb{R}^{n \times m}$$
$$V \in \mathbb{R}^{3 \times n}$$
$$b \in \mathbb{R}^{n}$$
$$c \in \mathbb{R}^{3}$$
$$y \in \mathbb{R}^{3}$$

$$h_i = \{\sigma(W x_i + b) / x_i \in X\}$$
$$H = (h_1, h_2, \ldots, h_t)$$
$$h = \text{max\_pool}(H)$$
$$o = \sigma(Vh + c)$$
$$\hat{y} = \text{softmax}(o)$$
$$L = -\, y^T \ln(\hat{y})$$

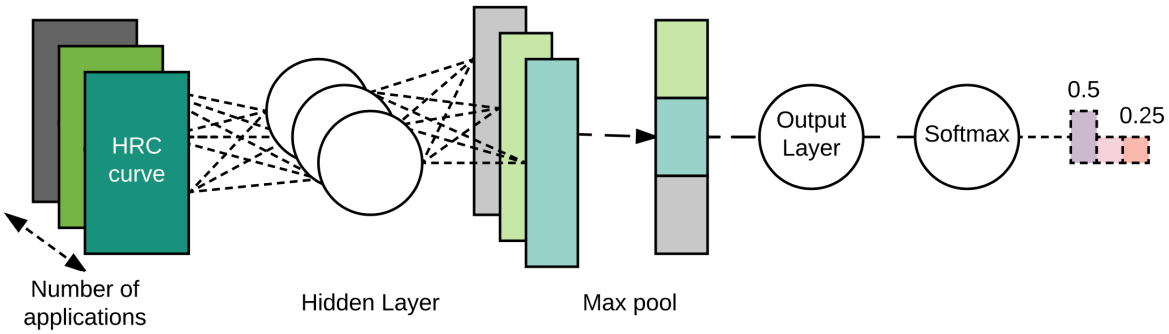

Figure 2: ANN architecture that picks a memory partitioning algorithm based on the application's workload.

Implementation

```
train_ANN(
mem_size=12702,
hidden_size=200,
learning_rate=0.01,
epochs=5
)
```

Defined in https://github.com/DeadlyBunny24/ANN-DistributedSystems/blob/master/model.py

Args:

- mem_size: Integer value. Number of partitions in memory.
- hidden_size: Integer value. Size of the hidden layer.
- learning_rate: Floating point value. Learning rate.
- epochs: Integer value. Number of epochs to train for.

Returns:

None. Produces the trained model.

Future work

1. Define the csv input format (labels are missing from the files).
2. Implement mini-batch training.
3. Implement validation accuracy.
4. Visualize learning using Tensorboard.
5. Serve and train the model using Tensorflow serving.

References

Abad, C. L., Abad, A. G., & Lucio, L. E. (2017). Dynamic Memory Partitioning for Cloud Caches with Heterogeneous Backends (pp. 87–90). ACM Press. https://doi.org/10.1145/3030207.3030237

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. Cambridge, Massachusetts: The MIT Press.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv Preprint arXiv: 1609.04836*. Retrieved from https://arxiv.org/abs/1609.04836

Huang, Q., Birman, K., van Renesse, R., Lloyd, W., Kumar, S., & Li, H. C. (2013). An analysis of Facebook photo caching (pp. 167–181). ACM Press. https://doi.org/10.1145/2517349.2522722

Shalom, N. (2008, August 13). Amazon found every 100ms of latency cost them 1% in sales. Retrieved from https://blog.gigaspaces.com/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/