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) correlates a 1% loss in sales for Amazon to a 100ms increase in latency.

Latency can be reduced in many ways (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. Several algorithms can determine such memory partitions (e.g. Hill-climbing, LP-solver, and the algorithm proposed in (Abad et al, 2017)); Nevertheless, each outperforms the others on particular server workloads.

The goal of this project is to implement a neural network that can learn to pick the best[1] memory partitioning algorithm based on the workload of multiple applications.

[1] Best is the algorithm that minimizes latency (i.e. maximizes hit-rate). However, computation time and allocation costs are not considered when selecting.

The following sections detail an ANN that creates a fixed size embedding for multiple application's HRC. The embedding is later used to select the memory partitioning algorithm. The first section describes the architecture, the second section overviews the implementation and the last section proposes future work.

Architecture

The dataset consists of csv files. The left-most column is the memory partition, every other column is the hit-rate curve (HRC) of an application's workload (see figure 1 for an example). Files can contain multiple HRC curves. Accordingly, the model should account for all the curves when selecting the memory partitioning algorithm.

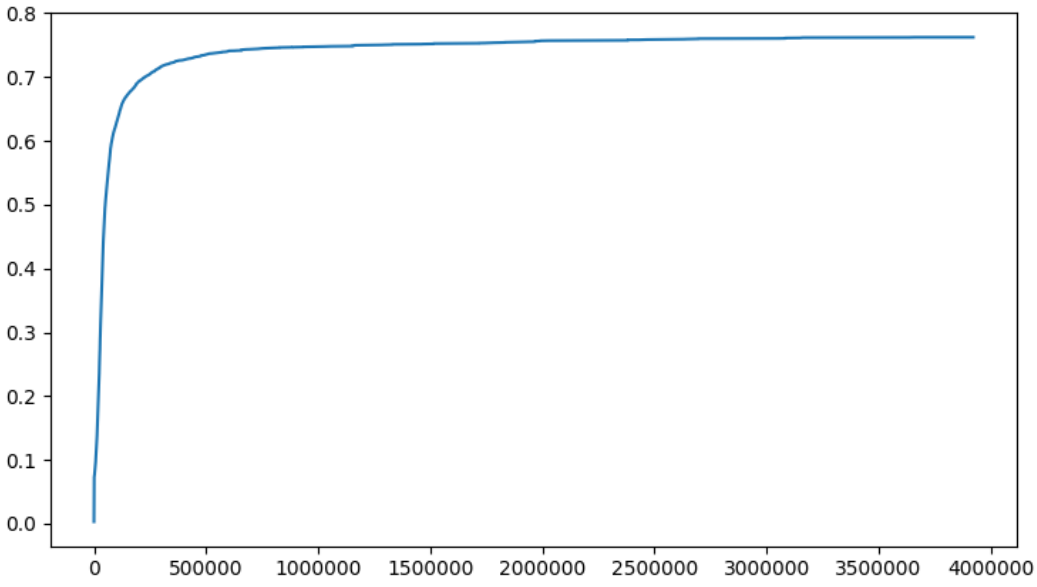

Figure 1: Sample HRC curve. The vertical axis represents hit-rate, the horizontal axis represents cache memory size.

The model is a feed-forward neural network. Each neuron's activation function is a sigmoid function, the activation of the last layer is a softmax function, and the loss is a cross-entropy cost function.

Figure 2 diagrams the neural network. Each HRC (x_i), from a file, is fed to the network independently. A max pooling layer then produces a fixed size embedding (h) from all the transformed HRCs (H). Lastly, the embedding is fed to a softmax layer to produce the final classification $(\hat{y})$.

p : Number of applications
m : Memory size
n : Hidden layer size
σ : Sigmoid activation function

$$X = \{(x_1, x_2, \ldots, x_p) / x_i \in \mathbb{R}^{m \times 1}\}$$
$$W \in \mathbb{R}^{n \times m}$$
$$V \in \mathbb{R}^{3 \times n}$$
$$b \in \mathbb{R}^{n}$$
$$c \in \mathbb{R}^{3}$$
$$y \in \mathbb{R}^{3}$$
$$o \in \mathbb{R}^{3}$$
$$\hat{y} \in \mathbb{R}^{3}$$
$$\text{softmax}(o_j) = \frac{e^{o_j}}{\sum_{k=1}^{3} e^{o_k}}$$

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

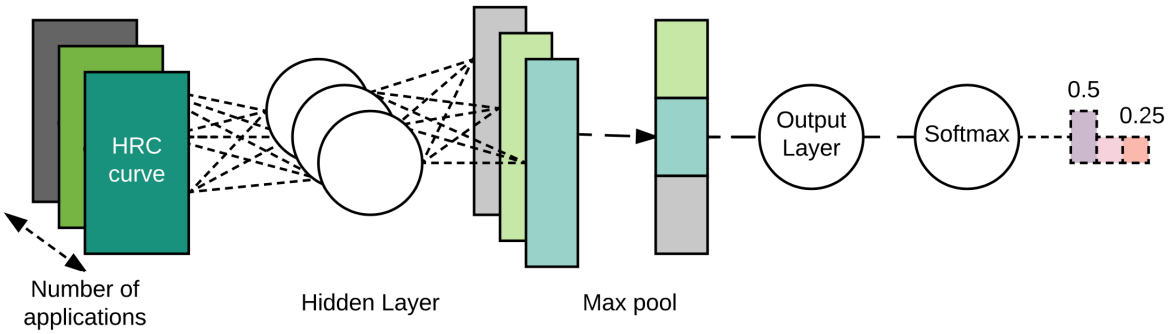

Figure 2: ANN architecture that picks a memory partitioning algorithm based on the application's workload. This corresponds to the processing of a single csv file with three HRCs.

Implementation

The model is implemented in python, using the Tensorflow API.

```
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. The trained model is saved to the tmp folder.

Train_ANN will train the model for number_of_files*number_of_epochs iterations. Afterwards, it will save the model to the /tmp folder. Figure 3 provides an overview of the computational graph created.

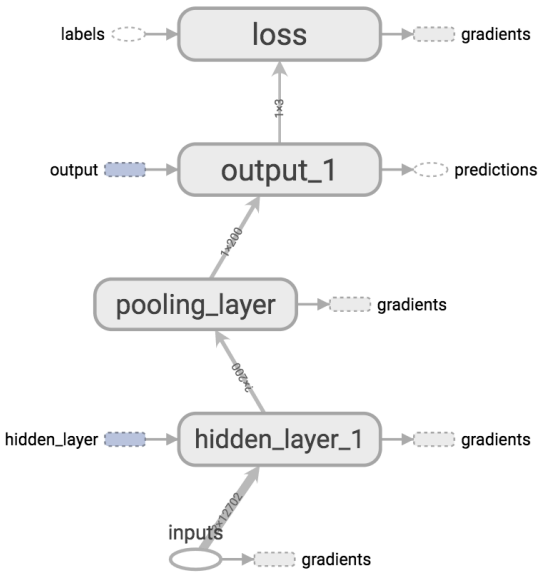

Figure 3: Computational graph produced by train_ANN. The visualization is done using Tensorboard. To turn Tensorboard on run the command: $ tensorboard —logdir ./tmp

Note that all HRCs must be of the same size. Accordingly, the parameter “ms” should be set to the traces size.

Future work

1. Define the csv input format (labels are missing from the files).
2. Implement mini-batch training.
3. Implement validation accuracy.
4. Use Tensorboard to visualize learning.
5. Replace the Queue based input pipeline with the Dataset API.
6. Use Tensorflow serving to serve the model.

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/