Abstract

Since the inception of computer science, data retrieval speed was a major bottleneck for numerous applications. The problem got only worse following the introduction of the Internet. To overcome this limitation the concept of caching was introduced. The solution is to store the data which is used the most closer to the location where it is used or store it in a storage with higher access speed. A large number of caching policies have been proposed but most of them require ad-hoc tuning of different parameters and none emerges as a clear winner across different request traces. For this reason, most of the practical caching systems adopt LRU (Least Recently Used) policy because of its simplicity and relatively good performance.

During the internship, we explore the possibility of the application of machine learning algorithms to solve the caching problem. We propose a caching policy which utilizes feedforward neural network and overperforms state of the art policies on both synthetic and real-world request traces. We also examine other attempts of application of machine learning techniques to handle the problem and compare their performance with the approach proposed by us.

Contents

1 Introduction

1.1 Caching problem

The invention of the computer allowed scientists to process vast amounts of data faster than ever before. However, soon a significant bottleneck was discovered - data retrieval speed. The introduction of the Internet only increased the influence of this problem. According to Cisco, annual global IP traffic is predicted to reach 3.3 zettabytes by 2021[1]. A massive increase in traffic volume naturally increases the load on the infrastructure. To improve the performance in various applications and to reduce the impact of the traffic growth the concept of caching was introduced. The idea behind this concept is to put the actively used data into storage from which it can be retrieved quicker. The goal is to reduce latency, shorten data access times and improve input/output. Since the workload of most of the applications is highly dependent upon I/O operations, caching positively influences applications performance. Previously described goals can be achieved by using a storage device which is physically closer to the data consumer or which has a higher data access speed. However, most of the time storage capacity is limited in such devices, or the use cost is higher. To maximize the utility of the storage devices various caching policies have been introduced.

1.2 Caching policies

Belady's min algorithm is proven to be optimal[2]. The general idea behind this algorithm is to evict from the cache objects which are requested furthest in the future compared to other objects in the cache. However, this information is not available in the real-time setting thus this algorithm cannot be deployed in a practical system.

First In First Out (FIFO) - one of the first proposed caching policies. Simple to implement and deploy but eventually has been replaced by more

sophisticated algorithms with better performance.

Least Recently Used (LRU) is the natural evolution of FIFO and the most commonly used caching replacement policy. It offers comparably good performance and does not require a lot of extra storage or CPU time.

Least Frequently Used (LFU) in some cases overperforms LRU but requires to track the number of requests for all of the objects observed. This disadvantage limits the number of applications of LFU.

Adaptive Replacement Cache (ARC)[3] is a caching policy introduced by IBM[4] in 2004. It offers better performance than LRU while keeping low computational resources requirements. Considered to be state of the art.

While a large number of caching policies has been introduced, there is still a room for improvement in comparison to the optimal algorithm. Moreover, since, as said before, the amount of web traffic is expected to rise, even a small improvement in caching policy performance could lead to significant cost savings in long-term. To compare the performance of caching policies we are going to use cache hit ratio[5] metric which is the most commonly used and effective metric for cache performance evaluation.

1.3 Neural networks

Following recent successful attempts of application of neural networks[6] for complex task solving[7, 8, 9] a question arises - is it possible to apply Neural Networks to learn close to optimal caching policy online? To tackle this problem, we will try to apply simple feedforward fully connected neural network with a goal to construct a new caching policy which would overperform existing methods. The primary challenge is to overcome the dependence on the future information by estimating it using neural networks.

1.4 Report organization

In the beginning, we will discuss related work in the area.

Then we continue by discussing what data is required to develop and test the proposed caching policy. For ease of development, a controlled and customizable environment is required. Thus we will discuss techniques to generate synthetic data which is good at representing the real world. We will continue by discussing what real-world data is used to test the performance of the proposed policy.

After that, we will discuss in more detail the concept of neural networks, intended use of neural networks for caching, the iterative process of tuning the architecture of the network.

In the last part, we will propose an architecture of a caching policy which is utilizing a neural network in the process of making a caching decision. We will compare the performance of the proposed policy with other approaches including the state of the art approaches.

2 Data preparation

Caching is intended to help with file retrieval from a distant server. A sequence of requests is called a request trace. Each entry to the request trace contains the time of the request, file ID, and optionally some metadata (size, type, etc.). To develop and test the algorithm the required data is split into two cases - the case of real-world data and the case of synthetic data. Real-world data is suitable for final algorithm evaluation since it represents real end-user request pattern. However, during the development process, it is better to use synthetic data, since it provides a controlled environment with a fixed number of unique items in which the behavior of the system is easier to understand.

2.1 Synthetic data

The primary challenge in the task of creation of the synthetic traces is to create them in such a way that they represent close to real-world data. A number of studies have been conducted to show that the popularity of files requested from web servers is distributed by Zipf's law[10]. At the same time, the arrival time of the requests can be modeled as a Poisson process[11]. This two facts will form the basis of synthetic trace generation.

While relying on previously described facts, we will be able to create synthetic traces, in the real world the popularity of the objects is not static with the passage of time since new content appears all the time and old content becomes less popular. That is why we have decided to represent synthetic traces in two cases. The first case is the case with the static popularity. The second case is the case with nonstatic popularity. In this case, the population is splitted in two equal sized parts. The first half of the population, as in the case one, has static Zipf distributed popularity. The popularity of the second half of the population is also distributed by Zipf's law but the popularity is randomly shuffled every predefined time frame t_0.

2.2 Real-world data

The real world data has been obtained from Akamai content delivery network[12]. The detailed information about the dataset you can find in the Table 1 below.

Total requests	$417 * 10^6$
Time span	5 days
Unique items	13271565
Request rate	966.97 requests/s
Min object size	3400 bytes
Max object size	1.15 gigabytes
Mean object size	$4.85 * 10^5$ bytes

Table 1: Akamai request trace information

As you can see in the Table 1, request trace contains not only the ID and the time of request arrival but also the size of the object. For now, we will consider that the size of all of the objects is equal and caching one object consumes one discreet place in the cache. The size of the object may later prove itself useful as a metadata feature for the neural network to process. This request trace is going to be used to evaluate the performance of the proposed algorithm and to compare it with other reviewed approaches.

3 Neural networks

3.1 Fully connected feedforward networks

The simplest example of a neural network is a fully connected feedforward neural network. It consists of an input layer, one or more hidden layers, and an output layer. All of the neurons in the previous layer are connected with all of the neurons in the next layer. Each connection has a weight. P^L is the matrix of weights between layers $(L-1)$ and L. The output o_L of the layer L is a column vector calculated as the product of the matrix P^L and the output of the previous layer o_{L-1}. Each layer can also have an activation function $f(x)$. Activation of the layer L is the $a_L = f(o_L)$. Typical activation functions used are:

Sigmoid: $f(x) = \frac{1}{(1-e^x)}$.

Rectified Linear Unit: $f(x) = max(0, x)$.

Hyperbolic Tangent: $f(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

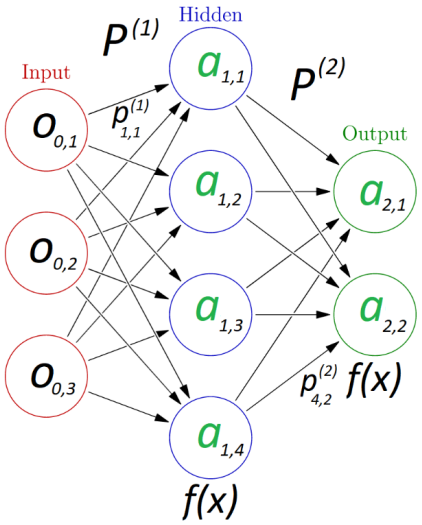

Figure 1: Fully connected feedforward network

The introduction of the activation functions ads nonlinearity to the input propagation through the neural network which should positively influence the accuracy of predictions.

Neural networks "learn" to make correct prediction through a process called error backpropagation[13]. It allows propagating the error from the output layer to the input layer while updating the weights between the layers using gradient descent. Even though many loss functions to calculate the error have been proposed, we are going to apply classical loss function - Mean Squared Error (MSE):

$$f(x) = \frac{\sum_{i=1}^{N}(y_{true} - y_{pred})^2}{N}$$

3.2 Chosen architecture

We propose an idea of predicting the popularity of objects in the future based on, mainly, the popularity in the past. To prepare a learning dataset, it is possible to split the request trace in time frames (or time windows) and calculate the popularity of each item in each time frame. Lets denote this popularity as $X_{i,j}$. Each row would consist of $K + 1$ popularities values, K values are input, and 1 is the output. To keep popularity independent of the number of requests in the time frame, the popularity is represented as the fraction of requests. Keeping popularity values in the "raw", unchanged state led to poor performance of the neural network since the large difference in popularity, which measured in a few orders of magnitude, caused the neural network to learn to make good predictions for the most popular objects sacrificing the accuracy of predictions of popularity for less popular objects. To fix this issue, we decided to apply a transformation for both input and output popularity values. All of the values are transformed by the next formula: $f(p) = -log(p + const)$. This transformation reduces the difference

between the smallest and the largest values processed by the neural network and proved to improve the accuracy of predictions greatly.

After some consideration, the next neural network architecture has been chosen. 4 neurons in the input layer, i. e. we are going to predict the popularity in the future based on popularity in 4 previous time frames. We will further experiment with this value discussing the performance of the proposed caching policy. Then the input is feedforwarded through 2 hidden layers with 128 neurons in each. We want to predict the popularity in the next time frame, thus only one neuron in the output layer. To every layer except the output, a bias neuron is added. A bias neuron always outputs 1 and is intended to improve the accuracy by allowing to shift the output of any layer in any dimension. As for the activation, we concluded that rectified linear unit performs the best. To overcome the "dying ReLU"[14] problem, a variation of ReLU is applied - Leaky ReLU:

$$f(x) = \begin{cases} x, & \text{if } x \geq 0; \\ a * x, a \ll 1 & \text{otherwise.} \end{cases}$$

References

[1] Cisco, "Cisco visual networking index: Forecast and methodology, 2016–2021." Available at `https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html`.

[2] L. A. Belady, "A study of replacement algorithms for virtual storage computers," *IBM Systems Journal*, pp. 78–101, January 1966.

[3] N. Megiddo and D. S. Modha, "Outperforming lru with an adaptive replacement cache algorithm," *Computer*, vol. 37, no. 4, pp. 58–65, April 2004.

[4] IBM. `https://www.ibm.com/`.

[5] B. Davison, "A survey of proxy cache evaluation techniques," *Proceedings of the Fourth International Web Caching Workshop*, pp. 67–77, April 1999.

[6] K. Hornik, M. Stincheombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[7] T. N. Sainath, B. Kingsbury, G. Saon, H. Soltau, A.-R. Mohamed, G. Dahl, and B. Ramabhadran, "Deep convolutional neural networks for large-scale speech tasks," *Neural Networks*, vol. 64, pp. 39–48, April 2015.

[8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis,

"Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 28 January 2016.

[9] I. Sutskever, O. Vinyals, and Q. V. Le, "Multilayer feedforward networks are universal approximators," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[10] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: evidence and implications," *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, 1999.

[11] C. Williamson, "Internet traffic measurement," *IEEE Internet Computing*, vol. 5, no. 6, pp. 70–74, Nov/Dec 2001.

[12] Akamai. `https://www.akamai.com/`.

[13] B. Widrow and M. A. Lehr, "30 years of adaptive neural networks: perceptron, madaline, and backpropagation," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415–1442, Sep 1990.

[14] M. M. Lau and K. H. Lim, "Investigation of activation functions in deep belief network," *2017 2nd International Conference on Control and Robotics Engineering (ICCRE)*, Apr 2017.