Neural network knowledge distillation in tensor networks

Dereck Piché

March 23, 2023

Abstract

1 Introduction

TODO ctrl f R -¿ mathbbR TODO fix spelling

2 Knowledge Distillation

Knowledge Distillation is a machine learning practice which involves taking a trained model and using it's parameters to train another one. The already trained model is referred to as the "teacher", and the model in which his "knowledge" is to be distilled is referred to as the "student". While this is a relatively novel technique, there are already several distinct approaches introduced by researchers. We used two of these approaches. Our inspiration for the distillation methodology was found in a a 2021 survey which resumed the emerging practice [1].

2.1 Response-Based Knowledge Distillation

The first approach we used to look exclusively at the outputs of the student and teacher. Now, we apply the logits of our functions element-wise to the outputs and the softmax.

$$\mathsf{softmax}(v_i) = \frac{e^{v_i}}{\sum_j e^{v_i}} \tag{1}$$

The softmax function's objective is to transform the logits into probability distributions for the different classes. We will now apply a loss function L to these two functions. Since we are trying to reduce the divergence between two distributions, we will use the Kullback-Leibler divergence loss

$$\mathsf{KL}(\mathsf{P},\mathsf{Q}) = \frac{1}{n}\sum_{i}^{n} \mathsf{Q}\frac{\log_e(\mathsf{Q})}{\log_e(\mathsf{P})} \tag{2}$$

Here, Q is the distribution we are aiming for and P is the one we have.

2.2 Layer-based approach

The survey on Knowledge Distillation coined the term *Layer-based approach*. Deep learning models work in layers of feature maps. It is hypothesized that these different layers represent different layers of abstraction in the internal representation of their input.

3 Tensor Networks

Tensor Networks come from the study of quantum phenomena. They started being used recently as machine learning models. Tensor Networks can be thought out as two things: a visual notation system and a set of methods for tensor manipulation.

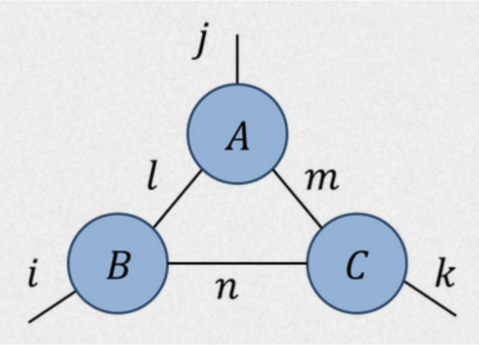
Tensors Before explaining these two, we shall disambiguate the meaning of "tensor". In this report (and very often in the context of machine learning), we use the word "tensor" to refer to the mathematical arrays of arbitrary indices. Here, the number of indices is called the "order" of the tensor, meaning that vectors are simply tensors of order 1 and matrices tensors of order 2.

Contraction At the heart of tensor networks is the *contraction* operation. Tensor Networks are used to compute a larger network by "contracting" several smaller tensors over chosen indices. A "contraction" is simply an operation where we sum over indices. For example, the contraction of A_{ijk} and B_{ijk} on index j will produce the tensor $C_{ik} = \sum_j A_{ijk}B_{ijk}$. Evidently, the two indices present in a contraction must be of the same size.

Notational system The principal motivation behind the creation of Tensor Networks to compute or approximate large tensors *contracting* several smaller tensors over chosen indices. This is all well and good until our conventional summation notation begins overclocking our primal brains. So, in order to make the manipulation of these network of tensors, mathematicians created a notational system. Tensors are represented as nodes where each vertex connected to the node represents one of the tensor's indices. If a vertex connects two nodes, it means that the indices shall be contracted in order to produce the post-contraction tensor. It should be evident that by the definitions above, no node (tensor) is completely isolated in a tensor network, as it would be completely purposeless. The shape of the post-contraction tensor can be easily visually identified, since it is found in the unconnected vertices. A simple tensor network can be found in figure 1.

Tensor Network Methods The term *Tensor Network Methods* is used to refer to, you guessed it, the methods. There are several architectures of Tensor Networks that are frequently used, such as the *Matrix Product State (MPS)*, the *Tenso*

Figure 1: Simple tensor network illustrating the notation.



4 Kernel

Consider an input vector $x \in R^d$. Let $\phi(x) : R \mapsto R^{d_\phi}$. Then, let us take the tensor product of apply $\phi(x)$ applied to every element of the vector x.

$$\Phi(x) = \phi(x_1) \otimes \phi(x_2) \otimes (\dots)\phi(x_d) \tag{3}$$

We obtain a tensor $\Phi(x)$, of which the sum of it's element contain the basis of a space of products of the elements of the transformed elements of $\phi(x)$.

In order to reduce the abstractness of this statement, we can take say that $\otimes$ refers to the Kronecker Product.

4.1 Multilinear feature map

In this project, we attributed a particular importance to the local feature map

$$\phi^*(x) = \begin{bmatrix} 1 \\ x \end{bmatrix} \tag{4}$$

With, this particular transformation, the Kronecker Product gives us a global feature map $\Phi(x)$ which is a tensor where each element is a basis of the space of multilinear functions on the elements of the vector x. Here, the order or shape of the tensor is superfluous and gives no theoretical advantage.

4.2 Linear Combinations

Now, the question is how do we use $\Phi(x)$, how does it become useful? Well, it becomes useful when we take a linear map of it's elements. Let our $\Phi(x)$ be an arbitrarily shaped tensor of d_Φ dimensions. The final form of our function $f(x)$ will be

$$f(x) = \sum_{i}^{d_\Phi} \theta_i [\Phi(x)]_i \tag{5}$$

Now, we have a function that can, under some choice of transformation ϕ, become extremely expressive. For our particular local feature map ϕ^*, it can represent any multililinear function of the input vector x. We shall represent linear combinations of $\Phi(x)$ by $T\Phi(x)$.

5 Using The Matrix Product State Tensor Network

In this project, we used the *Matrix Product State* (MPS) tensor network to approximate the function $T\Phi(x)$. Here, this function is computed by contracting a Tensor Network. Before the contraction, some of the elements of the tensors in the tensor network are set by applying the mapping $\phi^*(x)$.

5.1 Single MPS

TODO add info to this! How does the Matrix Product State decomposition work? What does it look like?. Suppose we have a Tensor of the form T.

5.2 Composition of MPS

When we use the $\begin{bmatrix}1\\x\end{bmatrix}$ local feature map, a single MPS can only express multilinear functions of x. It would be practical for our model to be able to express multipolynomial functions up to a certain degree z. In order to do this, we will have to perform a combination of functions of the form:

$$H \tag{6}$$

5.2.1 Proof of expressivity

Here, we shall prove an interesting statement regarding the expressivity of a composition. Here is a commutative diagram to help us visualise:

$$\begin{tikzcd}
X \in R^d \arrow[r, "f"] & Y_1 \in R^{2^d} \arrow[r, "g"] & Y_2 \in R^{(z \cdot d)} \arrow[r, "f"] & Y_3 \in R^{2^{z \cdot d}} \arrow[r, "g'"] & Y_4 \in R
\end{tikzcd} \tag{7}$$

Definition 5.1 (Multilinear polynomial)**.** A multilinear polynomial is a function $f : R^n \mapsto R$ of the form

$$f(x) = T \cdot \left(\begin{bmatrix}1\\x_1\end{bmatrix} \otimes \begin{bmatrix}1\\x_2\end{bmatrix} \otimes \cdots \otimes \begin{bmatrix}1\\x_d\end{bmatrix} \right) \tag{8}$$

Where $T \in R^{p \times R^n}$.

In other words, a multilinear polynomial is a polynomial that is linear if $\forall x_i$ then the polynomial is linear if we fix every variable except x_i. TODO use arrow to show the flow of functions

5.3 Patching of MPS (experimental)

5.4 Expressivity of MPS combinations with transformation $[1, x]^t$

Let $x \in R^d$. Let $f(x)$ be a function that returns a vector v, which contains every element required to form a basis of the space of x_i-variate multilinear functions.

Let

$$\Theta = \{\theta_1, \theta_2, (\dots), \theta_{d_{theta}}\} \tag{9}$$

in

$$M(f(x),\Theta) = \begin{bmatrix} m(f(x),\theta_1) \\ m(f(x),\theta_2) \\ (\dots) \\ m(f(x),\theta_{d_{theta}}) \end{bmatrix} \tag{10}$$

Where $m(f(x),\theta_i):\ R^{2^d} \mapsto (R^d \mapsto R)$ is of the form

$$m(f(x),\theta_i) = m(\nu,\theta_i) = \sum_j \theta_{i,j} \cdot \nu_j \tag{11}$$

We can trivially choose $\Theta^* \in \Theta$ such that

$$M(f(x),\Theta^*) = \begin{bmatrix} \textit{repeated } z \textit{ times} \begin{cases} x_1 \\ (\dots) \\ x_1 \end{cases} \\ \textit{repeated } z \textit{ times} \begin{cases} x_2 \\ (\dots) \\ x_2 \end{cases} \\ (\dots) \\ \textit{repeated } z \textit{ times} \begin{cases} x_n \\ (\dots) \\ x_n \end{cases} \end{bmatrix} = \lambda \tag{12}$$

We can rewrite λ as

$$\lambda = \begin{bmatrix} \varepsilon_{1,1} \\ (\dots) \\ \varepsilon_{1,z} \\ \varepsilon_{2,1} \\ (\dots) \\ \varepsilon_{2,z} \\ (\dots) \\ \varepsilon_{d,z} \end{bmatrix} = \lambda^* \tag{13}$$

Let Z be the space of x_i-variate polynomial functions of degree $\leq z$. Then every monomial of any x_i-variate polynomial function $\zeta \in Z$ is of the form.

$$x_1^{k_1} x_2^{k_2} (\dots) x_d^{k_d} \tag{14}$$

under the condition $\sum_i k_i \leq z$. However, for every $K = \{k_1, k_2, (\dots), k_d\}$ meeting this condition, we can rewrite the monomial as

$$\left(\prod_{i_1=1}^{k_1} \varepsilon_{1,i_1}\right)\left(\prod_{i_2=1}^{k_2} \varepsilon_{2,i_2}\right)(\dots)\left(\prod_{i_d=1}^{k_d} \varepsilon_{d,i_d}\right) \tag{15}$$

by using the elements of λ^* from (13). However, we clearly see that this term is a multilinear monomial of the variables in λ^*. This implies that $f(\lambda*)$ returns a basis-vector of x_i-variate polynomial function of degree $\leq z$.

In other words,

$$\xi(x, \Theta) = M\left(f\left(M\left(f\left(x\right), \Theta^*\right)\right), \Theta\right) \tag{16}$$

can express any x_i-variate polynomial function of degree $\leq z$ under fixed Θ.

6 Methodology

The experiments done for the projet were programmed using Python. Now, evidently, using Python alone was not possible. The big deep learning library we used was Pytorch, as is common in machine learning today. However, it does not provide many tools to train and build Tensor Networks. We thus used TorchMPS [2], a library built using Pytorch for the creation and training of learning Matrix Product State tensor networks.

Learning rate
We used the very standard learning rate of 0.01. This is the learning rate used by *Keras*.

Model size

Approach to the results As a matter of scientific integrity, we have chosen to show the results even if they are heavily disappointing. Not doing so can result in certain statistical biases which can be avoided.

7 Results

7.1 Training the single MPS

TODO: train batch loss MPS valid loss MPS

TODO: horizontal bar for teacher train batch loss MPS valid loss MPS

TODO: valid loss distill MPS valid loss base MPS

7.2 Training the hidden-layered MPS

TODO: train batch loss 2-MPS valid loss 2-MPS

TODO: horizontal bar for teacher train batch loss 2-MPS valid loss 2-MPS

TODO: valid loss distill MPS valid loss base MPS

7.3 Training the patch-MPS

TODO: Explain why the training failed.

8 Analysis

9 Conclusion

9.1 Further Exploration

TODO talk about capturing locality in the mappings TODO talk about capturing locality in general with tensors

References

[1] Jianping Gou et al. "Knowledge Distillation: A Survey". In: *International Journal of Computer Vision* 129.6 (Mar. 2021), pp. 1789–1819. DOI: `10.1007/s11263-021-01453-z`. URL: `https://doi.org/10.1007%2Fs11263-021-01453-z`.

[2] Jacob Miller. *TorchMPS*. `https://github.com/jemisjoky/torchmps`. 2019.