Neural network knowledge distillation in tensor networks

Dereck Piché

January 27, 2023

Abstract

1 Introduction

In the last few decades, the use of artificial neural networks has gained a lot of popularity in several application areas. Because of their usefulness, we would like to be able to use them in systems that have fewer computational resources (nested systems, for example). Thus, we would like to find methods to reduce their temporal and spatial complexity while keeping their capabilities. This is where knowledge distillation comes in. Suppose we have a trained neural network that performs well on a certain task T. Then we can say that this neural network has knowledge about this task T. We would like to be able to transfer the knowledge of an already trained neural network into a system that takes less space. The technical term given to the knowledge transfer process is knowledge distillation. Despite the youth of this avenue of study, it remains broad in scope. The project will therefore focus more specifically on knowledge distillation of artificial neural networks in tensor networks.

We now need to briefly explain what a tensor network is. Tensor networks are mathematical objects originating from the modeling of quantum phenomena (see Richard Penrose Feynman). Here we see tensors as a generalization of vectors and matrices where we have an arbitrary number of indices. It is important to mention that tensor networks are intimately linked to a graphical notation which allows us to manipulate them much more easily than if we were limited to a purely analytical notation. From this notation, researchers have created different types (or patterns or categories) of tensor networks that possess certain distinct properties. Some of

these patterns (such as the MPS https://tensornetwork.org/mps/) possess properties that are crucial to our task. In particular, several factorization and optimization methods for temporal and spatial complexity are known and studied for MPS (Matrix Product State / Tensor Train). Thus, we could use these optimizations after distillation to reduce the computational costs of neural networks. The project will consist of analyses and experiments that will aim to clarify/advance this topic (in the form of a report). We have not determined how theoretical or practical the project will be.

2 Layer-by-layer approach

It has been common to each layer of a neural network as a certain abstracted representation of the previous information and their for generalisibility. Thus, we propose to change the cost, we use a tensor mapping and train each mapping individualy.

$$x \longrightarrow H_1 \longrightarrow H_2 \longrightarrow (\dots) \xrightarrow{g} \hat{y}$$

Each hidden layer is of the form;

$$a^l = \sigma^l\big(W^l a^{l-1}\big)$$

In tensor layer form, it will be defined as

$$a^l = T^l \cdot \Phi\big(a^{l-1}\big)$$

b The main difference is that in the tensor approach, the "heavy" part is done by the non-linear transformation while a little work is done with the linear mapping. The opposite is true with neural networks. Here, $\Phi(X)$ (X begin the input vector) is a tensor product of several identical non-linear mappings of each element x_i. Thus, we have

$$\Phi(X) = \phi(x_1)\phi(x_2)\dots\phi(x_n)$$

Were each $\phi : \mathbb{R} \to \mathbb{R}^d$, and each $d > 1$. Thus, our $\Phi(X) : \mathbb{R}^n \to \mathbb{R}^{(d\times)^{n-1}d}$. In other words, our Φ returns a tensor of order n, where each indices run from 1 to d.

3 Going further with local transformations

In previous works, the non-linear mappings were independant and equal for each (x_i). It would be interesting if these linear mappings were codependant.

Here is an example of transformation to desambiguate what we mean

$$\phi_i(x_i, x_{i+1}) = \begin{bmatrix} x_i \\ x_i \cdot x_{i+1} \end{bmatrix}$$

It would be interesting to apply this transformation for a set of randomised pairs before the training.

4 On transformations

This is it!

$$\begin{gathered} T_1\phi_1(x) + T_2\phi_2(x) + k_1 \\ \equiv \\ T_3\phi_1(\ T_5\phi_1(x) + T_6\phi_2(x) + k_3\ ) + T_4\phi_2(\ T_{75}\phi_1(x) + T_8\phi_2(x) + k_4\ ) + k_2 \\ \equiv T_3\phi_1(\ \lambda_1(x)\ ) + T_4\phi_2(\ \lambda_2(x)\ ) + k_2 \end{gathered}$$

Then we have with derivatives

$$\begin{gathered} \frac{d}{dx}[\ T_1\phi_1(x) + T_2\phi_2(x) + k_1\ ] \\ \equiv \\ \frac{d}{dx}[\ T_3\phi_1(\ \lambda_1(x)\ ) + T_4\phi_2(\ \lambda_2(x)\ ) + k_2\ ] \end{gathered}$$

Which is saying that

$$
\begin{aligned}
& T_1 \frac{d}{dx}\ \phi_1(x) + T_2 \frac{d}{dx}\phi_2(x) \\
& \equiv \\
& \frac{d}{dx} T_3 \phi_1(\ \lambda_1(x)\ ) + \frac{d}{dx} T_4 \phi_2(\ \lambda_2(x)\ ) \\
& \equiv T_3 \frac{d\phi_1}{d\lambda_1}\frac{\lambda_1(x)}{dx} + \frac{d}{dx} T_4 \frac{d\phi_2}{d\lambda_2}\frac{\lambda_2(x)}{dx} \\
& \equiv T_3 \frac{d\phi_1}{d\lambda_1}\frac{\lambda_1}{dx} + T_4 \frac{d\phi_2}{d\lambda_2}\frac{\lambda_2}{dx} \\
\equiv T_3 \frac{d\phi_1}{d\lambda_1} & \left[T_5 \frac{d}{dx}\phi_1(x) + T_6 \frac{d}{dx}\phi_2(x) \right] + T_4 \frac{d\phi_2}{d\lambda_2} \left[T_7 \frac{d}{dx}\phi_1(x) + T_8 \frac{d}{dx}\phi_2(x) \right] \\
& \equiv T_3 T_5 T_4 T_7 \frac{d\phi_1}{d\lambda_1}\frac{d\phi_2}{d\lambda_2}\frac{d}{dx}\phi_1(x) + T_3 T_6 T_4 T_8 \frac{d\phi_1}{d\lambda_1}\frac{d\phi_2}{d\lambda_2}\frac{d}{dx}\phi_2(x) \\
& \equiv\ T_1' \frac{d\phi_1}{d\lambda_1}\frac{d\phi_2}{d\lambda_2}\frac{d}{dx}\phi_1(x) + T_2' \frac{d\phi_1}{d\lambda_1}\frac{d\phi_2}{d\lambda_2}\frac{d}{dx}\phi_2(x)
\end{aligned}
$$

Thus, we can see that it will only possible in this case if the set of do not form a constant! In other words, they must not be be cancellable!. We can generalise this to higher order transformations! Their derivatives must not be cancellable! It is a requirement!