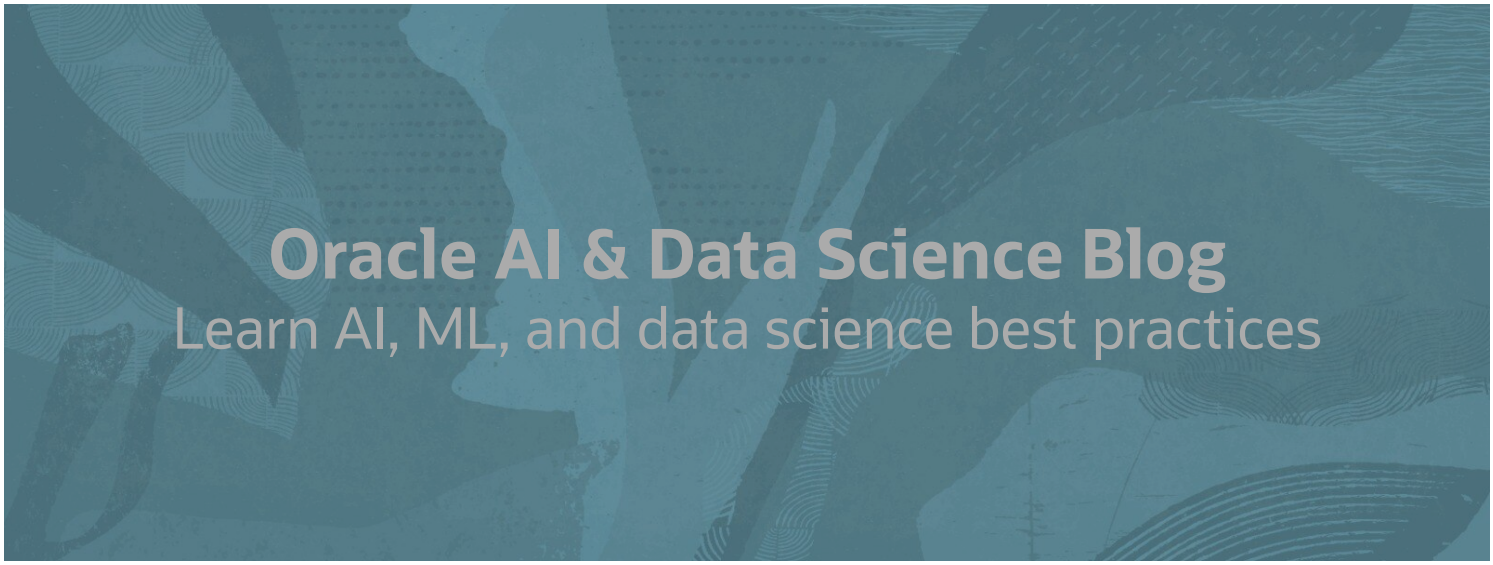


AI IN BUSINESS, DATA SCIENCE

July 15, 2020

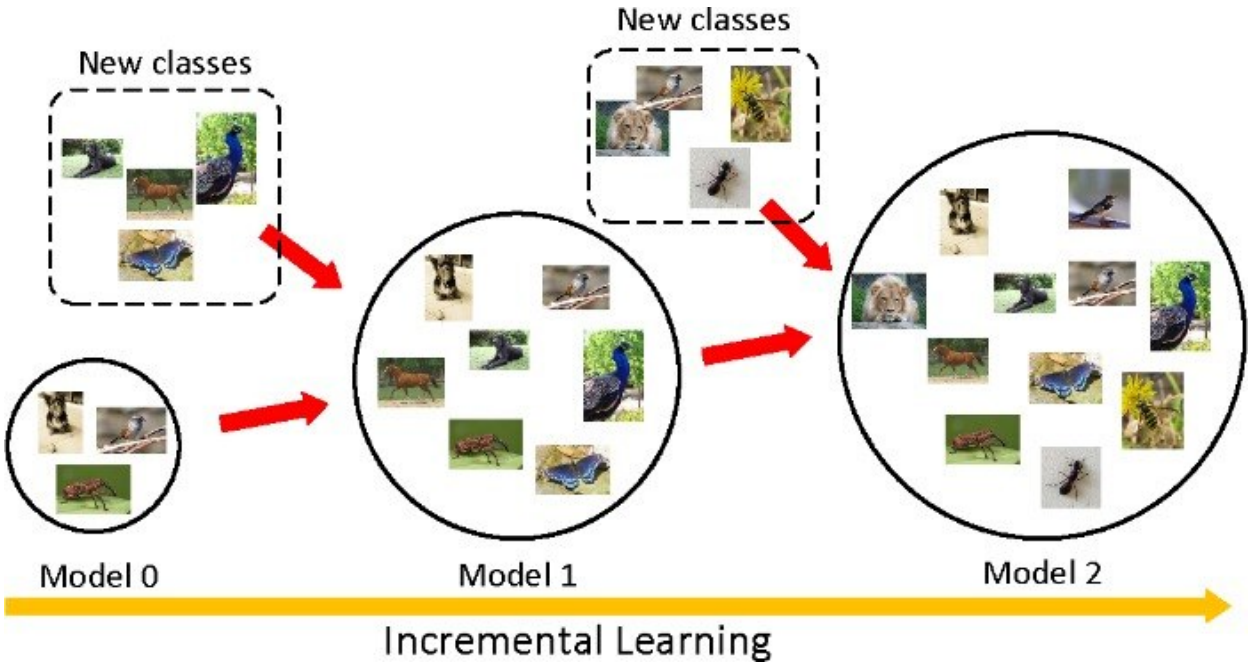
Knowledge Distillation for Incremental Learning

Praphul Singh

One of the major areas of concern in deep learning is the generalisation problem. This has been a hot topic for research for the past few years. Generally what happens is that we get a use case, we build a model for that and we push it in production.

But what if we have a slight change in our problem statement? Do we need to solve it once again from scratch? What if we don't have the dataset we had previously?

We seek for a way to preserve the previous learning of the system and work on just the evolution part. And we will be talking about one of the similar aspects of the problem in this blog.



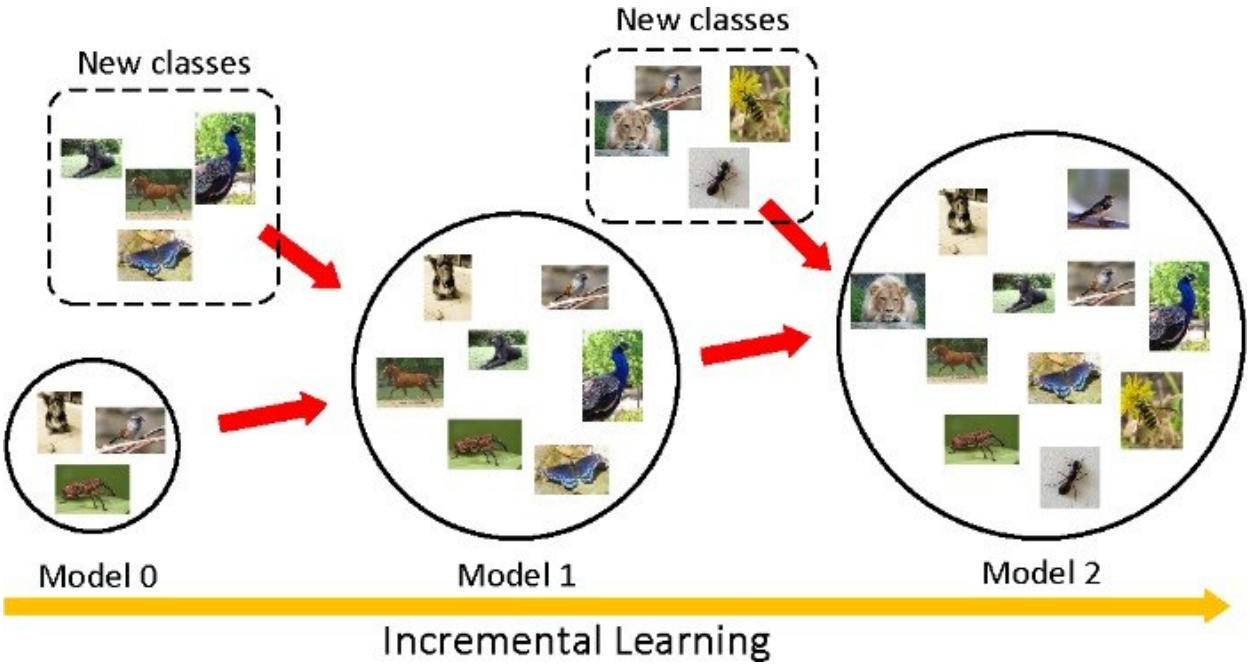
So, welcome to the new blog of **Learn & Share** thread, and bear with me for some of the next few paragraphs on knowledge distillation for incremental learning.

Subscribe to the Oracle AI & Data Science Newsletter to get the latest AI, machine learning, and data science content sent straight to your inbox!

Incremental Learning

Let us define the problem statement first, and then we shall discuss the various solutions for that.

Suppose you have a dataset for 5 classes and you built a deep learning network for the classification problem. Now, let us imagine you have the model, but you lost the dataset and you need to add an extra class for the existing problem statement. Well, let us not get to the solution that quickly. Instead, let us derive the solution from the history itself.

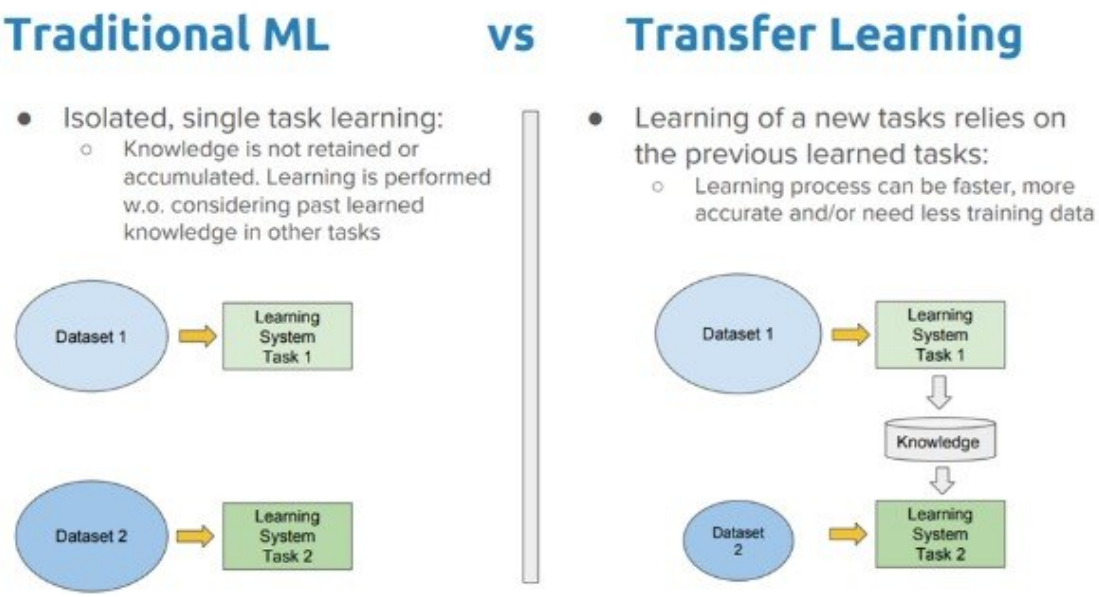


One of the famous starts in the approaches of generalisation is t**ransfer learning,** which has in fact proved to be very successful.

So, in this approach, we already have a pre-trained model for a dataset1. Now, with a new dataset2, which comes from a similar kind of domain, we can use the knowledge of the previous pre-trained model instead of training it again from scratch.

What happens is that we initialise the new model with the same weights as of the pre-trained model and add a new softmax layer, removing the last layer of the previous model (considering it is a classification problem). And we train the model, starting from the point where the pre-trained model had reached already. This results in fast and better convergence.

If the previous model is very large, we do not have to update the weights for its previous layers (you can decide how many) by freezing them (making them untrainable). We update only the weights of the extra layers (some previous layers too if you want) we have added. This strategy is called **Fine Tuning**.



Check out Praphul's reinforcement learning series, which includes posts on Q-learning, building a custom environment, deep Q networks, actor-critic networks, and proximal policy optimization (PPO).

Limitations of Transfer Learning

The limitation of transfer learning is that it cannot be used for incremental learning if we do not have the dataset for the previous classes.

Confused? 🤔 Let us understand it by an example: Suppose the pre-trained model used the dataset1 which had classes -> {A, B, C, D}. Now we need to add some extra classes, let us say {E, F, G} to model. The obvious approach would be to remove the previous softmax layer having 4 output node, add a new one with 7 output nodes, and fine-tune the model with a new dataset consisting of classes -> {E, F, G}. Well, if you do that, you will get very bad accuracy for the previous classes (maybe for the new classes too).

This is the major problem we will be talking about from here now, and it is called **Catastrophic Forgetting** by neural networks.

Revise and Learn

We, humans, are very good at generalisation because we have some sort of memory network in our brain which stores the previous knowledge and fine-tunes it according to the new tasks we get exposed to. The important thing is that we do not deviate much from our previous learning. Neural networks are very basic and simple prototypes of real neurons, and it still is challenging to reach to that level.

In our previous problem, if we have the previous dataset1 and we combine it with the new dataset2 to reach our final objective, we surely get a good result. That is because the dataset1 lets the neural network revise its knowledge and avoids it from deviating too much. The dataset2, on the other hand, allows the neural network to fine-tune itself for learning new classes.

So, is having the entire dataset the only solution? **IT IS NOT** 😀. Let us discuss the possible solutions one by one:

A Necessary Naive One 👇🏻

As the name suggests, it does not go well with the performance, but still, it is good to know the naive ones for achieving a better solution.

Remember, we talked about not letting our neural network deviate too much. So one obvious thing for that would be to add a regularizing constraint on the weights of the pre-trained model to prevent it from going too far.

The reason it does not work is because, in neural networks, generally we have more number of weights than the entire data points, and thus they learn functions which are highly complicated and far away from being linear. Even a small amount of change in the weights could allow the new final objective function to go farther from the old one.

No worries, have patience. We have a better solution coming next 👇🏻

Pseudo-Revision 🙄

If you recall, we talked about revising old things before learning the new ones. So, the idea here is to obtain something which can be used for the revision for our pre-trained model. And this is what a research proposed by **Zhizhong Li et al** does.

The revision part is done by the dataset2, but with a different strategy. Even if a data-point does not belong to the trained classes, a neural network will predict some probabilities for each of those classes. The predicted probabilities for the old classes act like the old labels in the new-training now. The aim is to obtain a similar kind of probability distribution which the old model would have predicted for the new dataset2.

Knowledge Distillation-Loss

The loss function used for the old task will be a modified **cross-entropy** with the probabilities altered according to the definition given by **Hinton et al.**

The idea here is to increase the weights for the low probabilities for the better encoding of similarities among the classes. The loss function can be mathematically represented as follows:

$$\begin{aligned}\mathcal{L}_{old}(\mathbf{y}_o, \hat{\mathbf{y}}_o) &= -H(\mathbf{y}'_o, \hat{\mathbf{y}}'_o) \\ &= -\sum_{i=1}^{l} y_o'^{(i)} \log \hat{y}_o'^{(i)}\end{aligned}$$

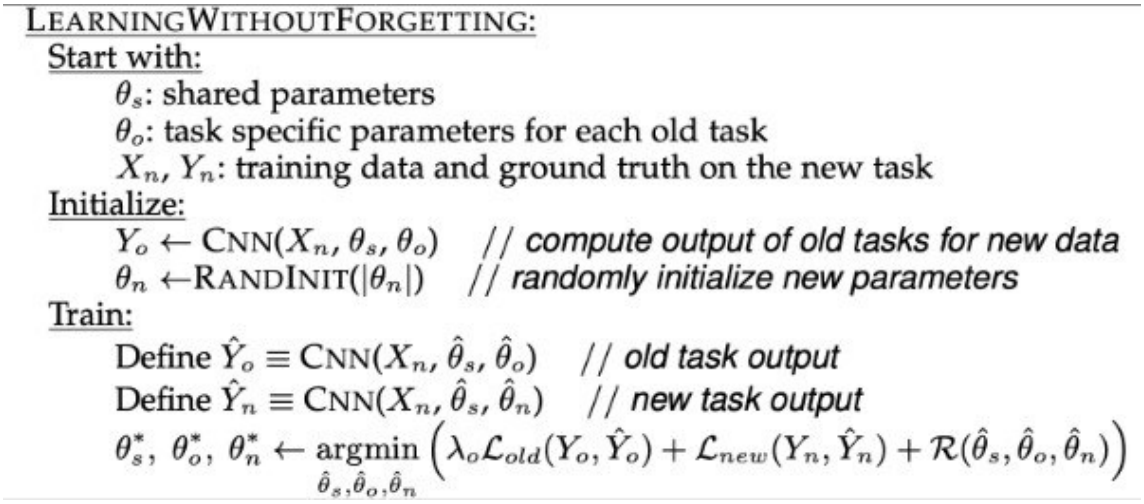
where,

$$y_o'^{(i)} = \frac{(y_o^{(i)})^{1/T}}{\sum_j (y_o^{(j)})^{1/T}}, \quad \hat{y}_o'^{(i)} = \frac{(\hat{y}_o^{(i)})^{1/T}}{\sum_j (\hat{y}_o^{(j)})^{1/T}}.$$

T is a constant greater than or equal to 1. For the new tasks, we use the usual **categorical cross-entropy loss**.

Algorithm

Now that we understand the loss functions and the basic idea behind the research, let us discuss the end-to-end algorithm:



Step 1: Obtain probabilities for the old task by feeding the dataset2 to the old model Step 2: Randomly initialise the weights of new layers that you have added.

During training we have three losses:

1. The loss for old task: $\mathbf{L_{old}}$
2. The loss for new task: $\mathbf{L_{new}}$
3. Regularization for the weights

To control the importance for the losses, the total loss is a weighted sum of the three losses mentioned above with $\boldsymbol{\lambda_{old}}$ as the weight for the old task loss, $\boldsymbol{\lambda_{new}}$ as the weight for the new task loss, and $\boldsymbol{\lambda_r}$ as the regularizer constant.

The above algorithm gives us good performance, but the multi-task transfer learning approach with data for every class stands above when we consider performance.

References:

1. https://arxiv.org/pdf/1606.09282.pdf

2. https://arxiv.org/pdf/1503.02531.pdf

I hope the blog gives an informative insight into incremental learning. I will keep adding more algorithms as I keep learning about them.

Till then, **Keep Learning, Keep Sharing**

To learn more about AI and machine learning, visit the Oracle AI page, and follow us on Twitter @OracleAI.

Be the first to comment

Comments (0)

Recent Content

DATA SCIENCE

A Simple Guide to Connect OCI Data Science with ADB

DATA SCIENCE

These five data science tips help you find valuable insights faster

It's no surprise that tech startups depend on data science. At the Cloud Native Computing

Foundation's KubeCon conference on November 18,...