**Hands-on Meta Learning**

This document corresponds to summary of the book **Hands-On Meta Learning with Python** by author Sudharsan Ravichandiran.

1. **Introduction to Meta Learning**

the problem with deep neural networks is that we need to have a large training set to train our model and it will fail abruptly when we have very few data points. Let's say we trained a deep learning model to perform task A. Now, when we have a new task, B, that is closely related to A, we can't use the same model.

How do we humans learn? We generalize our learning to multiple concepts and learn from there.

Meta learning produces a versatile AI model that can learn to perform various tasks without having to train them from scratch. We train our meta learning model on various related tasks with few data points, so for a new related task, it can make use of the learning obtained from the previous tasks and we don't have to train them from scratch.

1. **Meta learning and few-shot**

Learning from fewer data points is called **few-shot learning** or **k-shot learning** where **k** denotes the number of data points in each of the classes in the dataset.

If we have exactly one dog and one cat image then it is called one-shot learning, that is, we are learning from just one data point per class. If we have, say 10 images of a dog and 10 images of a cat, then that is called 10-shot learning. So, **k** in k-shot learning implies a number of data points we have per class.

zero-shot learning where we don't have any data points per class.

two-way k-shot learning; so, n-way means the number of classes we have in our dataset.

we have a dataset, D, we sample a few data points from each of the classes present in our data set and we call it as **support set**.

we sample some different data points from each of the classes and call it as **query set.**

**train** our model with a **support** **set** and **test** with a **query** **set**. We train our model in an **episodic fashion**—that is, in each episode, we sample a few data points from our dataset, D, prepare our support set and query set, and train on the support set and test on the query set. Therefore, our model will learn how to learn from a smaller dataset.

1. **Types of meta learning**
2. Learning the metric space
3. Learning the initializations
4. Learning the optimizer
5. **Learning by metric space**

We want to learn the similarity between two images, for example. In the metric-based setting, we are a simple neural network that extracts the features from two images and finds the similarity by computing the distance between features of these two images. Common algorithms are Siamese Networks, Prototypical Networks and Relational Networks. This approach is widely used in few-shot learning where we do not have many data points.

1. **Learning the initializations**

In this method we try to learn optimal initial parameter values.

Instead of initializing the weights randomly, we try initializing the weights with optimal values or close to optimal values, then we can learn very quickly.

The algorithms based on learning initialization are MAML (Model Agnostic Meta Learning), Reptile, and Meta-SDG.

1. **Learning the optimizer**

In this method, we try to learn the optimizer.

In FSL settings, gradient descent fails as we will smaller dataset. So, in this case, we will learn the optimizer itself.

1. **Learning to learn gradient descent by gradient descent**

We will have two networks:

First, the base network that tries to learn and, second, a meta network that optimizer the base network.

This is one of the simplest meta learning algorithms.

We optimize our model using gradient descent, can we learn this optimization process automatically?

We replace our traditional optimizer (gradient descent) with Recurrent Neural Network (RNN)

What are we really doing in gradient descent? It is basically a sequence of updates from the output layer to the input layer and we store these updates in a state. So, we can use RNN and store the updates in an RNN cell.

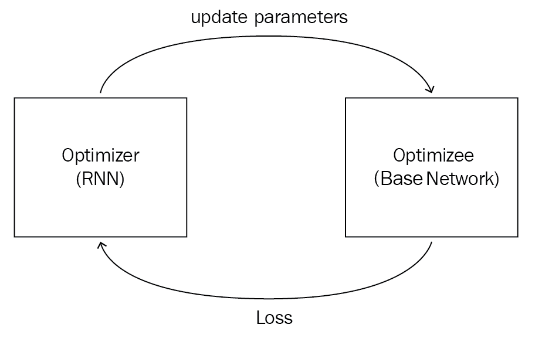
For optimizing on RNN we use gradient descent. So, we are learning to perform gradient descent through an RNN and that RNN is optimized by gradient descent. That is what is means by the name learning to learn gradient descent by gradient descent.

We call our RNN, an optimizer and out base network, an optimizee.

We use the RNN for finding this optimal parameter.

RNN optimizer finds the optimal parameter and sends it to the optimizee base network; the optimize uses it parameters, compute the loss, and sends the loss to the RNN.

Based on loss, the RNN optimizes itself through gradient descent and updated the model parameter θ.



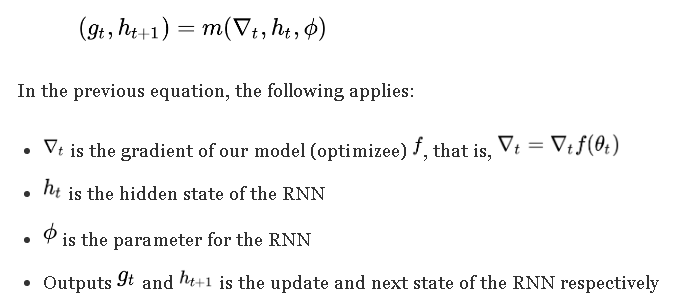
our optimizee (base network) is optimized through our optimizer (RNN). The optimizer sends the updated parameters—that is, weights—to the optimizee and the optimizee uses these weights, calculates the loss, and sends the loss to the optimizer; based on the loss, the optimizer improves itself through gradient descent.

base network (optimizee) is parameterized by θ and our RNN (optimizer) is parameterized by ϕ.

the loss of our optimizer is the average loss of the optimizee and it can be represented as follows:



RNN, takes as input the gradient of optimizee as well as its previous state and returns output, an update that can minimize the loss of our optimizee. Let's denote our RNN by a function :



So, we update our model parameter values using

A close up of a map

Description automatically generated

1. **Optimization as a model for few shot learning**

In few shot learning settings, gradient descent fails abruptly due to very few data points.

Gradient descent optimization requires more data points to reach the convergence and minimize the loss. So, we need a better technique.

1. **RNN Recurrent Neural Network**

RNN are a type of artificial neural network design to recognize patterns in sequence of data. These algorithms take time and sequence into account, they haver temporal dimension.

RNN are applicable even to images, which can be decomposed into a series of patches and treated as a sequence.

RNN possess a certain type of memory, and memory is also part of the human condition.

The information works in a cycle through a loop. It means, their input is not just the unique to take account but they have perceived previously in time.

A picture containing object

Description automatically generated

The decision a recurrent net reached at time step t-1 affects the decision it will reach one moment later at time step t. Therefore, RNN has two sources of input, the present and recent past.

Sequential information is preserved in the RNN hidden state, which manages to span many time steps as it cascades forward to affect the processing of each new example.

Long term dependencies are the correlations between events separated by many moments. An even downstream in time depends upon, and is a function of, more or one events that came before.

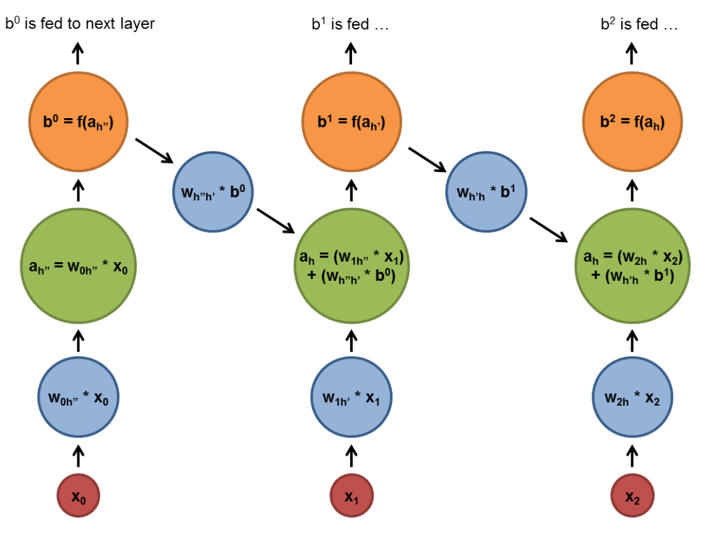
Process of carrying memory forward mathematically:

A close up of a clock

Description automatically generated

The hidden state at time step is . It is a function of the input at the same time step modified by weight matrix W. All is added to hidden state at the previous time multiplied by its own hidden-state-to-hidden-state matrix U.

U matrix are filters that determine how much importance according to both the present imput and past state hidden state. The error return via backpropagation ad be used to adjust their weights. Φ is the activation function.



Recurrent neural network relies on an extension of backpropagation called backpropagation through time (BPTT), however, BPTT cost per parameter update becomes wry higher over many time steps.

* Vanishing Gradients.

The vanishing gradient problem emerged as a major obstacle to recurrent net performance.

RNN seeking to establish connections between a final output and events many steps before were hobbed, because it is very difficult to know how much importance to accord to remote inputs. Moreover, because the layers and time steps of DNN relate to each other through multiplication, derivates are susceptible to vanishing or exploding.

A close up of a map

Description automatically generated

* **Long Short-Term Memory Units (LSTMs)**

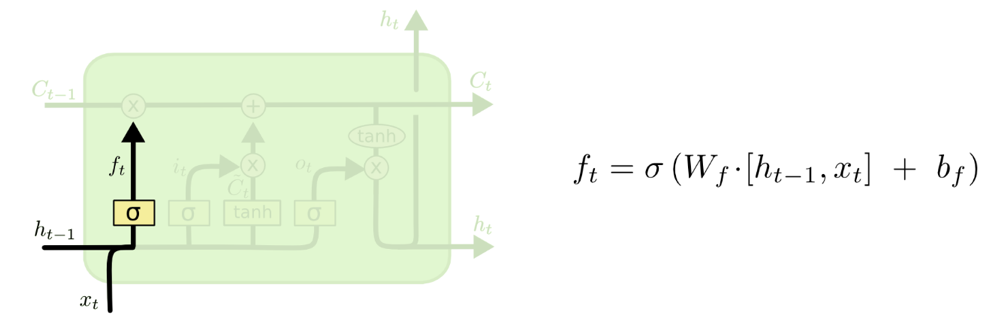
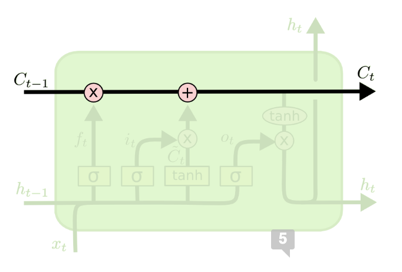
In direct response to the vanishing gradients problem of simple RNNs, the Long Short-Term Memory (LSTM) layer was invented. This layer performs much better at longer time series.

LSTM are a special kind of RNN, capable of learning long-term dependencies. They were introduced by [Hochreiter & Schmidhuber (1997)](http://www.bioinf.jku.at/publications/older/2604.pdf), and were refined and popularized by many people in following work.[1](https://colah.github.io/posts/2015-08-Understanding-LSTMs/#fn1) They work tremendously well on a large variety of problems, and are now widely used. LSTMs are explicitly designed to avoid the long-term dependency problem. The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. It’s very easy for information to just flow along it unchanged.

A picture containing text

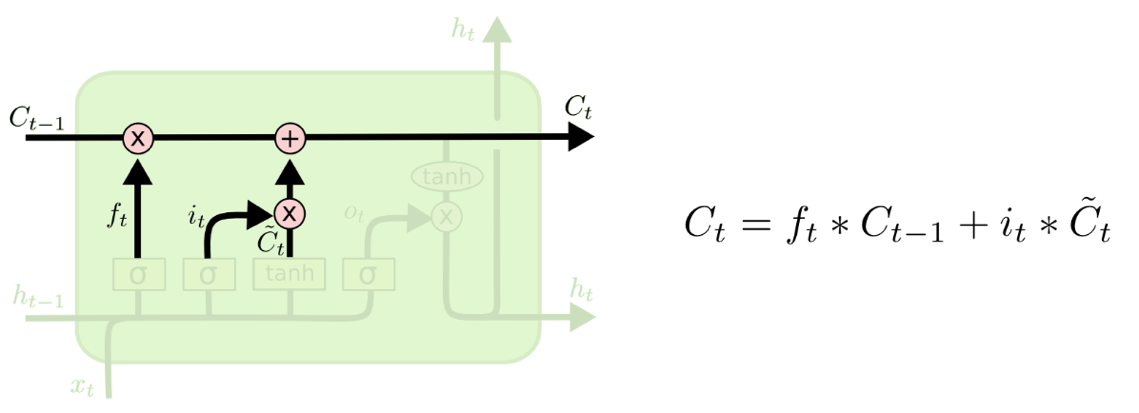
Description automatically generated

The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. An LSTM has three of these gates, to protect and control the cell state.



A close up of a piece of paper

Description automatically generated



A picture containing clock, object

Description automatically generated

* Gated Recurrent Unit (GRU)

