Introduction to Deep Learning

Perceptrons, Datasets, and Neural Nets, if you think you've stumbled across a biology article you have been mistaken. We're going to run through the fundamental principles of deep learning at a very high level to give you a smooth introduction to the field, without the mathematical baggage that comes with it.

AI FUNDAMENTALS?

The emergence of libraries/services such as FastAI, PyTorch, and AWS brings the focus of deep learning away from nitty-gritty mathematical details, opening up the field to the general practitioner instead of the privileged few with their enormous in-house clusters. To put this into relatable terms, we have now been thrust into the Wild West era of AI. The ease-of-use that comes with FastAI v2, in particular, allows anyone to train state-of-the-art models easily within five lines of code to allow for image classification of dogs and cats.

Within thirty minutes of consuming documentation, anybody with a python script, a unique dataset, and a decent computer can create a highly accurate classification model that could outperform some of the state-of-the-art models from a couple of years ago. If it's so easy though, why bother with the fundamentals of Deep Learning?

To put it simply - it has become *easy to write deep learning code* (you no longer have to write low-level NumPy code), getting it to *do exactly what you want well* still requires insight into the core fundamental principles behind how neural networks function.

Writing and designing deep learning algorithms strays vastly from traditional software engineering. One cannot simply expect to rip some code from an online forum and expect their model to fit a given dataset. You as the deep learning practitioner know your model best, and it is only with a solid grasp of the fundamentals that you can expect to solve most deep learning problems.

The Method

Knowledge surrounding the development and diagnosing of issues that are associated with a deep learning-based model are relatively much newer, and there is more mystery surrounding all the parts that go into, and how a neural network actually works. In this article we will be diving into the four main ingredients that come with the development of training a model:

Data: When it comes to any data-science problem, what we start by looking at is what we're loading into our model and using - data. If we take care to collect, maintain and process our data, we'll have provided our model with solid information to learn from.

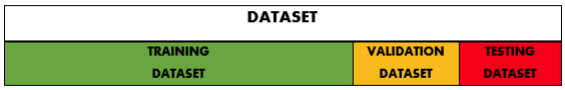
Model: Artificial neural networks have been greatly inspired by research into how our human brains work. We use artificial neural networks in Deep Learning to give our models the ability to learn from the dataset by tweaking the network slowly over time during training. There's an art to constructing a great model for a given dataset that can only be learnt through practice.

Training: The details in how our network learns over time to identify patterns within the dataset. Each model learns differently, and although there is a basic methodology at play, there is no gold standard procedure for how to train every given network. No matter how good a model is, a poor training setup could lead it nowhere.

Evaluation: Having a model doesn't mean that we've got a useful one. Hence, the crux of deep learning is being able to visually and quantitatively analyse how well our model is doing.

Data and the Dataset

Traditionally speaking, the process of 'Deep Learning' requires huge amounts of data. The more data you have, the better you can train your model to perform a task. Classic examples of datasets include cats and dogs, MNIST (images of handwritten numbers), and imagenet (hundreds of thousands of labeled images). Datasets can be of images, text, or anything else relating to the task at hand.



As deep learning practitioners, we have to cherish every bit of data we are given. Not every dataset is infinite and may be constrained by size, which may hinder the model's ability to fit the data depending on how small the dataset

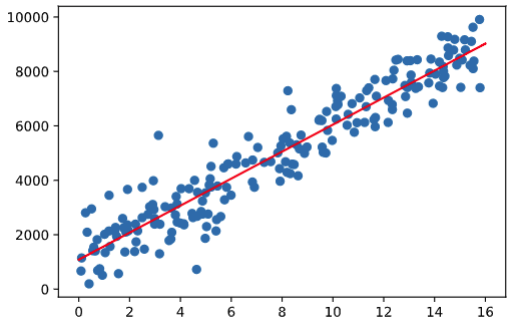
Training Dataset: The dataset we have pre-allocated for the model to learn from. This dataset typically acquires the majority of the initial dataset, and the proportion has taken increases as the size of the dataset increases. This is due to one simple rule, more data means more diversity. It allows our model to learn from more examples, allowing for an improved fit.

Validation Dataset: A separate set of data that is used to determine the accuracy of the model throughout the training process. As the model learns from the data, it is important to be able to *validate* its accuracy continually. This helps in selecting *parameters* for the network and determining whether the model is *overfitting* (i.e. memorising the data, discussed later in this article). Bypassing a new dataset into the model, separate from the data it is training on, provides an objective view of how the model is currently performing.

Note: parameters and overfitting are discussed here later on

Test Dataset: Established as a final evaluation of a model after we've modified our network by analysing our performance on the validation dataset. It is used as an independent evaluation of the model's performance **after** training. The results from this evaluation can be used to create metrics and visualisations to help understand where the network was struggling in particular. We will discuss the evaluation procedure in greater detail later on.

The Model



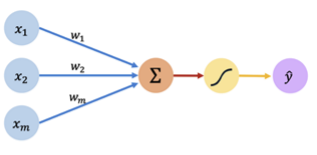
source: omnisci.com/technical-glossary/statistical-modeling

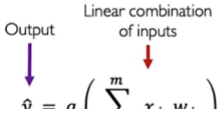
The aim of our models is to *fit* to a dataset, or in other words approximate an unknown, imperfect function which represents our data (i.e. to create accurate predictions).

Models can be used to differentiate between cats and dogs, to colourise black and white images or even to play games like chess and go. The primary difference between these models is their design. Both simple and complex models use the same building blocks, however the variation in model design occurs in the way they are combined (like lego). One can be relatively simple, while the other can require careful design and attention to detail of the artificial neural network.

Having an understanding of the fundamentals of neural networks is crucial to designing these kinds of models. We'll begin by diving into the building blocks of a neural network; the perceptron (modelled off of neurons within the human brain), and slowly work our way up to creating a model.

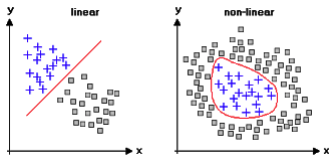
The Perceptron





Source: MIT 6.S191

The purpose of an individual perceptron is as simple as performing a non-linear transformation on a set of inputs, and providing an output. To break it down, we apply a unique weight to each individual input into the perceptron and sum the linear combination of these inputs together. Then we are able to pass this sum through a non-linear activation function to produce our output. Over time, these weights will change due to the network's training, and eventually be tuned to accurately approximate some output.



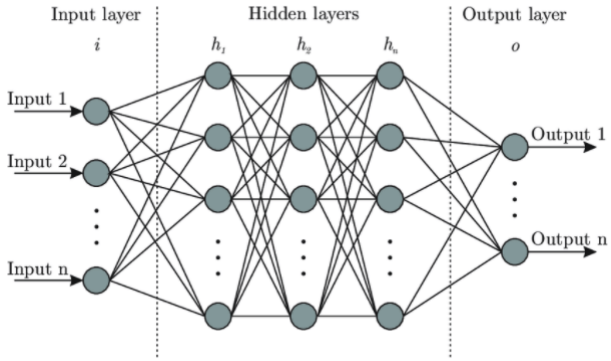
Source:
http://www.statistics4u.com/fundstat_eng/cc_linvsnonlin.html

Why do we pass this sum of inputs into a non-linear activation function? Simple - not every set of data lies in a straight line! Without our network incorporating some sort of non-linearity, we would only ever be able to model phenomena which can be represented by straight linear lines. With each proceeding layer, it slowly builds upon the results of the preceding non-linear perceptrons which leads to a more complex nonlinear function (our neural network) that is able to approximate the *unknown function*.

Assembling the Neural Net

As could be gathered from the previous section, multiple perceptrons can be added together to create something special, a dense layer. The term 'dense' comes from the imponderable amount of connections between inputs of the network and perceptrons. With each added perceptron comes an additional set of weights and a new output, allowing us to increase the complexity of our model.

Once the network of perceptrons gets to the point where a hidden layer (the layers in the middle, where nonlinear transformations occur) exists between inputs and outputs, we can classify it as a neural network. A network is classified by the number (and type, but this is more complicated) of hidden layers within its architecture, i.e.: a neural network with a single hidden layer is called a single-layer neural network.



Source: https://www.kdnuggets.com/2019/11/designing-neural-networks.html

THE TRAINING

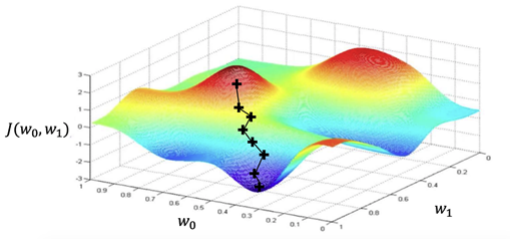
The Loss Function

The loss itself is essentially how wrong our network is on a given prediction, and it is what we train the network against to ensure our model can fit the dataset. We naturally aim to reduce this numerical loss by training the network. The process of minimising the loss function is known as **backpropagation**.

output, and travelling backwards from the output layer through the hidden layers, manipulating the weights to minimise the loss.

Our **loss function** is what we use to calculate the loss, and it is the function we seek to minimize (or maximise) in order to improve our model. It outputs some sort of loss by estimating the amount of error within a set of weights. An example of a very basic loss function could be a mean squared error loss between the actual and predicted values.

Below is a very simplified visualisation of what we call the "loss landscape", in this example it can be seen that our loss function 'J(w0, w1)' is a function of only two weights. As we have seen above, it would be very hard to have a network involving only two weights; however, it works as a demonstration to show how we aim to find a point of global minimum within the landscape. With such a simple function, there are plenty of algorithms out there that could determine the global minimum with nanoseconds, due to our network architectures generally being much larger in complexity, we need to find other methods for locating this global minimum.



Gradient Descent

One of the more vanilla approaches for locating the global minimum is an optimisation algorithm known as gradient descent. To understand how it works, it's best to liken it to hiking down a mountain. From any given starting point on the mountain, you are trying to find the next step to take to ensure you are on a trajectory to make it to the global minimum point. Ideally, you aim to go down the steepest slope of the mountain to reach the bottom. If you take really small steps, you can make it, but it may take forever. If you take really large steps, you may end up going down the wrong path, and ending up in a local minimum. The amount you step down is known as the learning rate. It's very complex and problem specific to determine the ideal learning rate, and generally an estimate is established, and it is slowly tuned from there.

If you were to examine the loss landscape image shown above, imagine the plus symbols on there to be the model learning from the weights it currently has, so if you have two points and calculate the gradient, if it is positive, you know you have traversed up the landscape, which is not ideal for minimising loss. However, if the gradient is negative, it means you are traversing down the landscape, which is ideal. These gradients can then be used to update the weights of the model, which is how we use gradient descent to assist us in the backpropagation process.

Epochs

Before we begin to train the network, we must also decide to specify how many epochs will occur. An epoch is defined as a single training iteration, which includes an entire forward pass and backward pass of all the input data. The number of epochs you choose to train with impacts how accurate the model is and how to fit the model is to the data.

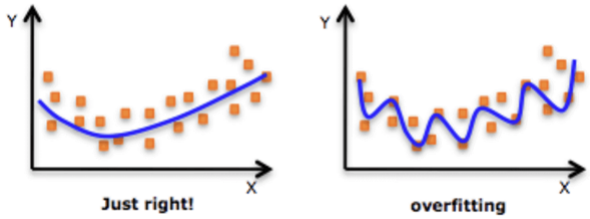
EVALUATION

Training and Validation Loss

Although both are experienced during training, it is sometimes necessary to step in and stop unnecessary computation. Training loss is a measure of how accurate the model is with the training set of data and validation loss is how accurate the model is with the validation set of data after each epoch. Generally speaking, the validation loss converging is a good indicator that your model is also converging to its optimal performance.

Overfitting

When the accuracy ot the model becomes really high on the training dataset and pertorms poorly on the validation and test datasets, it is a good indication that the model is overfitting to the training data. Overfitting is a modelling error that can occur when a function is too closely fitted to a set of data, as can be seen below.



Source: https://intellipaat.com/community/368/how-to-interpret-loss-and-accuracy-for-a-machine-learning-model

Here we can see that a function has been fit to the data points on the left to accurately model the data, whereas the function on the right has adapted to almost every single point/idiosyncrasy within the data.

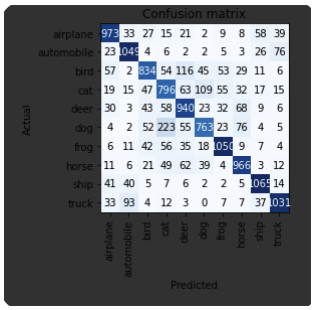
The deep learning practitioner must recognise when a minimum of loss is occurring and attempt to stop training before the data becomes overfit to the model. Overfitting can also be reduced by training the network on more data, and by also changing the complexity of the neural network architecture.

Remember, the model should be learning from the data, it should not be learning the data itself!

Source: Convolutional Memes

Metrics and Visualising Error

Seeing where the model went wrong is really important as it can help to identify the problems that may be occurring. Usually, we can identify problems with the dataset such as incorrect labels or low-quality data. An example of visualising the error can be seen in a confusion matrix like the one shown below, in which it identifies the predicted vs actual of a model after it has been trained to identify certain objects within images.



Source: FastAI v2.0

To Sum Up!

It really all boils down to how good your dataset is, the field itself is incredibly technical, but if you are lacking a large quantity of good quality data, as a deep learning practitioner you are limited from the get-go.

under your belt.

How can we improve? Pop your feedback in this form.

Want to join our help desk? Sign up here. It will be from 7-8pm, every Sunday, during September 2021.

Ryan Hartshorne (he/him)

Reinforcement Learning Project Manager

Comments (0)

Newest First

Preview POST COMMENT...

PREVIOUS
Introduction to Pytorch

ATTRIBUTIONS

Main banner gif: http://maximschoemaker.com/