Introduction

In this talk we will introduce the concept of deep kernel learning, which combines two approaches that were long held for contrary: gaussian process regressors, and neural networks.

You've been to this lecture, so we assume some prior knowledge in neural networks, but we will shortly introduce the concept of gaussian process regressors, and why they were thought of as opposite to neural networks. We will then show how the paper "deep kernel learning" proposes to combine both methods to achive some interesting results

Gaussian Process

So what is gaussian process regression? Well, it's a machine learning method, which, instead of outputting a single prediction value like neural networks do, returns a mean and a confidence interval for it's predicted value. Let's look at a picture to get a sense of it. Say we want to model a function, and we have a few data-points. You can see the GPR for this function would look like this: There is a mean function, and a shaded area which shows the confidence interval.

You can also see the data-points, and what is interesting to see, is that the confidence interval shrinks as it gets closer to the data-points. This is intended, because closer to the data points, we can assume, that the value would also be closer to the observation.

The way this works under the hood is somewhat like this: We're given a set of data points at which we know the value of the function. And we are given a distribution of functions which we call prior. We then generate an infinite amount of functions from this distribution, which pass through those points. This is called the posterior. From this posterior, we can generate a confidence interval.

Obviously generating an infinite amount of functions is not feasible, so thankfully, there is some nice math that allows us to calculate it analytically.

Math

Let's quickly take a look at the math that makes this possible. When I am talking about generating functions this might be misleading, because I am not talking about functions like x^3 or 2x -1. What I mean with function is a set of sampled points at regular intervals. like this:

Another thing to note is that similar input might mean different things in different scenarios. For instance when modeling ecosystems, you might expect the first of June in 2017, and 2018 to be very similar. On the other hand, when modeling the cumulative sales, you would expect June 2017 and 2018 to be very different.

So to ensure that similar inputs have similar outputs we specify the relationship between variables with a multivariate distribution. We do this with what is called a covariance matrix. As a quick background, you can see a multivariate distribution here with two variables. This distribution can be entirely described with the covariance matrix on the left. Since the functions are sampled from a multivariate distribution, we can change the covariance matrix, to change the types of functions we generate. With it we can create different patterns.

Let me quickly introduce the term kernel functions. Those functions are used to create the covariance matrix. There is some important distinction between the two when it comes to implementation, but for now just think of covariance matrix and kernel function as interchangable.

With different kernels it is possible to model different types of patterns for instance: There are kernels which can model periodic patterns, smoothed out patterns, linear patterns, and much more. It is also possible to combine two kernels. For instance to have a periodic function with a linear trend.

One nice thing about those multivariate distributions, is that we don't need to sample from them. It's possible to calculate the mean and confidence interval just with the kernel and a few data points. But here you just need to believe me, because Im running out of time.

Why contrarian?

Earlier I mentioned that they were long considered as contrarian, but why? Well first because it represents uncertainty, but also because they are non-parametric. What that means, is that neural networks learn to model the data, by adjusting millions of parameters, so that some error function is minimized. GPRs in contrast don't adjust parameters, they take in the data, and some kernel function which limits the functions and analytically calculates the mean and confidence intervals.

This creates some fundamental differences in how a model is created. For neural networks, no prior knowledge on the domain is needed to guide the structure of the network. Or at least, not as much prior knowledge as gprs need. that is because NNs are much more flexible in the kinds of functions they can model. For instance in previous example of ecosystems and or Nintendo sales data, you could use the same structure. In contrast, GPRs need a kernel which is guided by prior knowledge. Unlike NNs, GPRs can also predict uncertainty. However, they are computationally heavy and work best with low dimensionality.