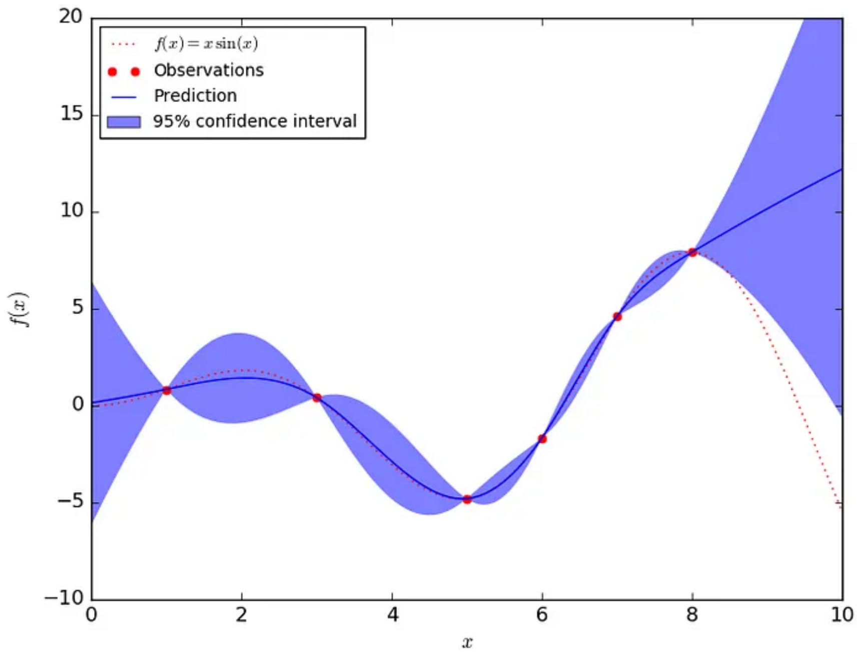
Introduction

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. With this infinite amount of functions, 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 the confidence interval analytically, without needing to generate an infinite amount of functions.

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 actually mean with a function is sampled points at regular intervals, like this:

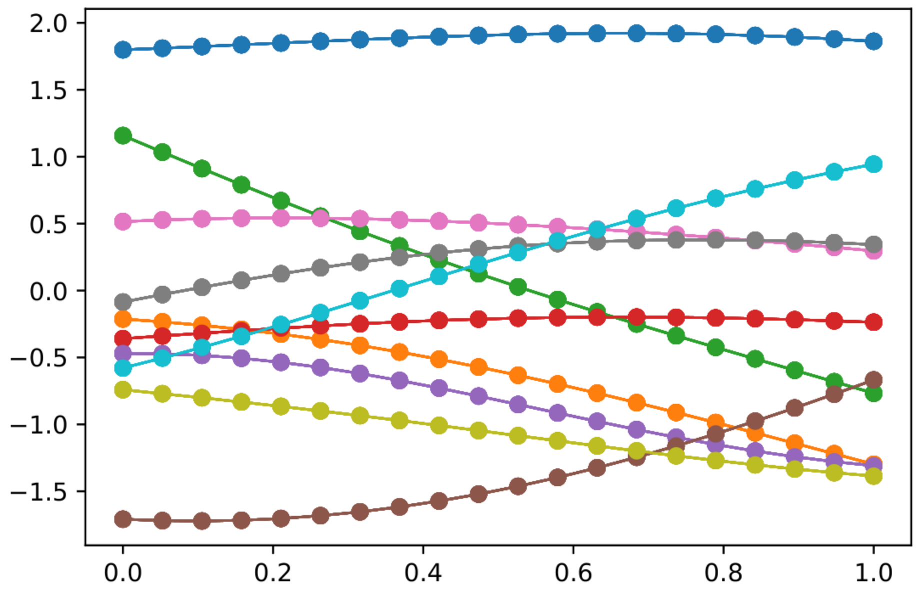


Figure 1: sampled functions

Note that we don't sample independent variables, because this would look like this; and we want similar inputs to have similar output

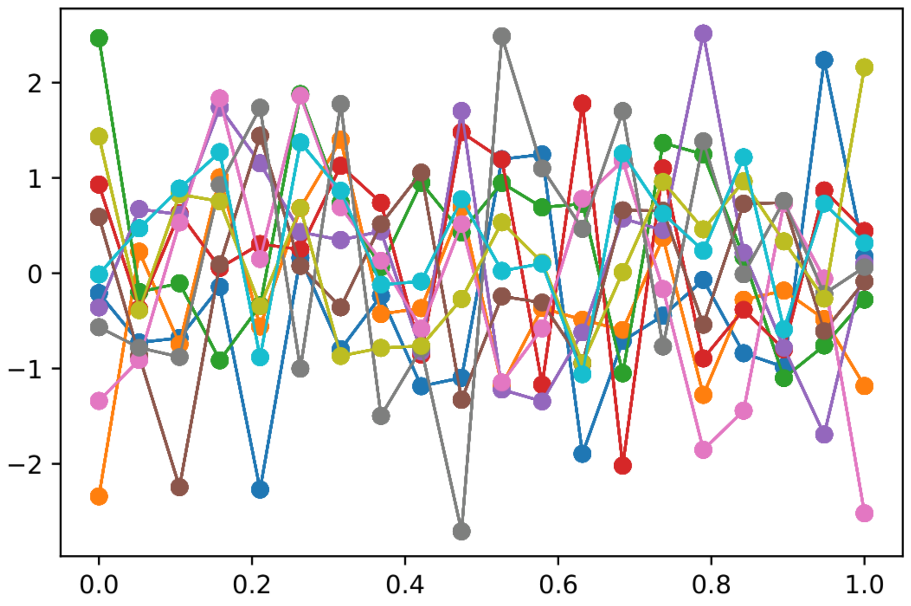
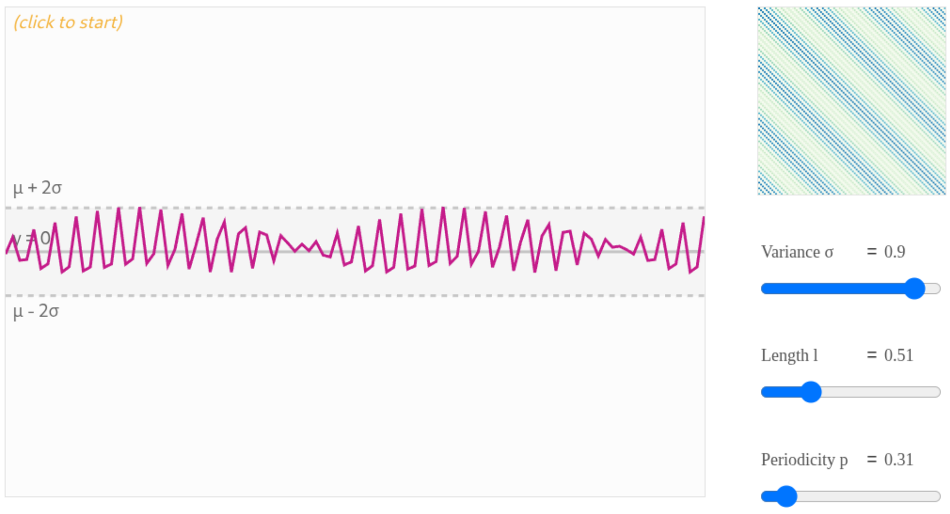


Figure 2: independent samples

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 Nintendo Switch sales, which released in March 2017, you would expect June 2017 and 2018 to be very different.

So what we do to ensure that the similar inputs have similar outputs, we can specify the relationship between the variables in the multivariate distribution. We do this with what is called a covariance matrix, or kernel how it is often called in GPRs. With this we can encode different patterns.

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

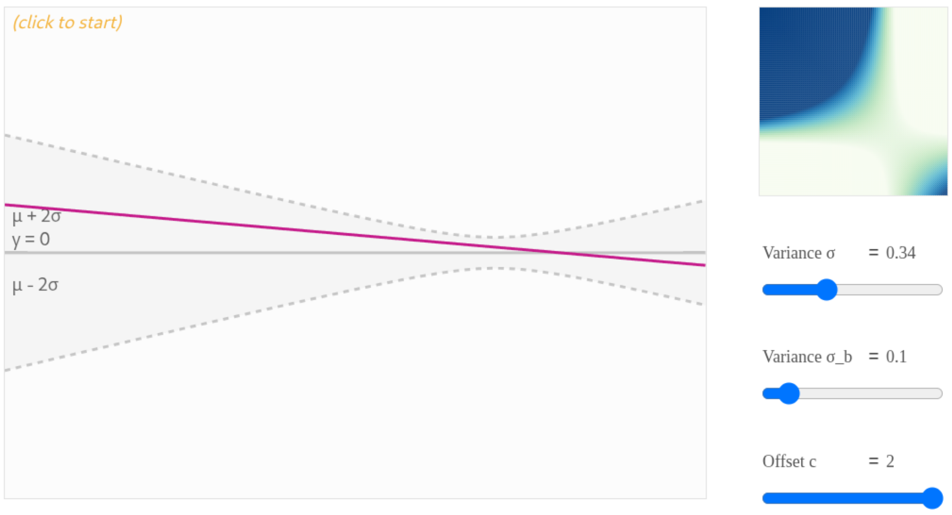


Variance σ = 0.9

Length l = 0.51

Periodicity p = 0.31

Figure 3: periodic kernel



Variance σ = 0.34

Variance σ_b = 0.1

Offset c = 2

Figure 4: linear kernel

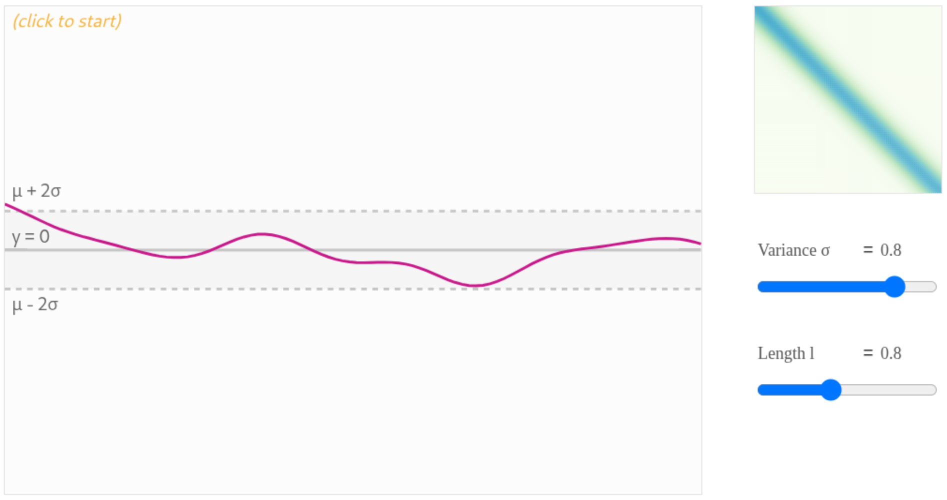


Figure 5: rbf kernel

One nice thing about those multivariate distributions, is that we don't need to sample. It's possible to analytically compute the mean function and the confidence interval resulting from a kernel and a few data-points to fit it to.

Why contrarian?

So why were long considered as opposite? 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 calculate 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. 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.

The Idea of the paper is to combine both methods, to get the best of both worlds. What they propose is to use a NN, which transforms the data into a latent representation. This latent representation is then feed to the GPR. The idea is that the NN is not only able to reduce dimensionality, but also to make the choice of the best kernel less crucial. The idea is that when choosing a bad kernel for the domain, using a NN beforehand would enable the input to be transformed so that the chosen kernel is not a bad fit anymore.