Range Map Machine

# Abstract

Consider a much quicker form of learning in which there are a set of inputs, each input has a range which is updated. Possibly a box/whisker/diagram. Then transforms are variationaly/randomly defined between subsequent nodes that try to match the N point distribution of the output.

Essentially take a basic histogram of the inputs and outputs, define random transforms which match the endpoints to the output distribution. Assimilate the usefulness of each transform across the data and pick the best few.

Randomly select a set of percentiles. Fit the function to these. (The percentile drawing function is important, flat or focus on the middle of the data set).

Consider domains of applicability.

Try the whole softmax cross entropy minimising thing for distribution matching?

To add hidden layers

# Prescription

For input variables make histograms of with some resolution based on quartiles etc. No scaling required. For the outputs, also do this. Find numerically the distribution which recreates the output distribution by various convolutions (addition) and (product). Find the real statistical distribution which is closest to this (after fitting). Apply mathematical forms, given parameters in each case. Do some sanity checking? Repeat process.  
  
The network will then create an extremely generalised mathematical function which is traceable. The convolutions represent products or sums with random variables drawn from those distributions. When the network is trained, we can explicitly evaluate all of these.

We run off there being all kinds of possibilities and running analytic distributions for all of those cases?

Assume all inputs and outputs are Gaussian distributed:

Assume a Gaussian distributed random variable exists such that (for each pair in general a different variable).

Then

Now assume that the random variable is a weighted sum of random inputs for unknown weights

Then

We can solve this? (Two equations for two unknowns). There will be a few solutions

For products we assume normal distributed inputs and then

Then

Method looping and saying if and were all distributed like … then if y was a weighted sum the weights would be …, if y was a product the weights would be … try pairs and pick the most promising. Then reiterate but with the sets of pairs instead.

We can also assume

Then

In both cases,

We could also consider the Q-Q plot. Which plots the quantiles of the data. Apparently if the distributions are linearly related, then the quantiles will form a line.

If we did fit these two and there was some kind curve which could be expressed, we would have a mapping

That being said, the function may have to be some kind of parametric function. Or could be a parametric function. The x’s must increase so there should be some sense of progression.

# Width Based learning

We can think of the function as a transform for some input dataset into an output dataset.

The transform will correctly convert p.d.f’s for any input data set.

We can also transform the input x’s into a kind of inverse space where we measure the possible widths occurring at different probabilities. There may be multiple functions created as inputs, alongside the overall sum of all functions version. We may want to also try and learn the corresponding widths of the output function as a function on the inputs.

In a sense this is two networks. For two problems but we know the data are related and the transforms are related?

Say we were learning , very simple. If we looked at the training histogram of and and we might see that the convolution for addition of random variables. This is a binary transform over probability distributions. From this we can infer that but it will be quite hard to notice such a thing. If we applied the widths transform to the P(x) distributions, there should also be some kind of transform.

# Splitting of the data

If we have and , we could assume something like the target is a sum of two things

This is an artificial splitting, because we could merge the two sums into a single sum with new weights . However, we can see what happens for now, later we can make the sum of two non-linear things. There will be distributions and , the distribution for can be written with

How does one train coefficient vectors such that this relationship is true? The loss associated with this truth is then

We can attempt to evaluate

If we assume some kind of kernel function and our training set is we can write

These kernel functions might be Gaussians for example.

Then

In the more complicated even that which is fundamentally more interesting, we can also write by using the inverse distribution for one of the variables.

It is best to alternate with products and sums. In this way there is no reduplication

Whether it is possible and useful to split the sum into three of four sub sums is a point for discussion.

In order to perform the multiplicative convolution, the two values must be centered on . Is there a way to deal with this?

For the summation convolution change the basis function to a Gaussian.

For the multiplicative convolution,

## Considerations:

The multiplicative convolution of two Gaussians is a Bessel function

Three Gaussians is a Meijer-G function and this continues for n Gaussians. The additive convolution of two Gaussians is another Gaussian.

# Unit Triangle

We can consider the convolutions of different types using the unit triangle basis function as .

A summation convolution gives a piecewise function, which in general can be expressed as

# Not the same basis functions

Another possibility is to pick a complementary pair of basis functions, such that the integral is defined for the two. If they are both reasonably complete, then it shouldn’t matter much that each probability distribution is represented by one or the other basis set. You could even swap the roles of the basis functions to see if it makes a difference to the answer.

This was supposed to be a very quick operation, but it is becoming clouded.

The idea was grab the quantiles of in the output distribution and use those to grab two other distributions.

We should then formulate this in a piecewise way. We want the two types of convolution for two general piecewise distributions.

A more generalised version:

# Moment matching machine

Is it true that for then and if then if they are uncorrelated???

If we postulate that an output is a mixture of sums and products of some inputs. Then for that mixture to be true, then the mixture should hold well across a range of moment predictions of those quantities?

Then we have nodes which sum and nodes which multiply.

# Mellin-Meijer Network AI

Have a positive model variable , assume that can be written using a suitably complicated function taken from the space of -G functions, in the restricted domain that are also probability distributions. For this to be the case, then the integral over all space is , which is normalisation. Then the Mellin transform of obeys where means the Mellin transform.

The Mellin transform of the Meijer-G function is

If we assume that the random variable is the product of two others and then we know

Or

If we assume that A and B are different linear sums of the inputs, and their probability distributions also follow complicated Meijer-G functions, then we know that

For any moment exponent . We can train the weights in the linear sum, and we can train the parameters in the Meijer-G functions. Obviously these products look like they will explode or shrink to nothing, so we take logs.

Then the left is a sum of log-gamma functions and the right is a sum of log-gamma functions. These can be handled in a nice way. They are anharmonic bowls between 0 and infinity. The argument should be kept away from 0.

We try to find a set of linear sums of the inputs, and try to find sets of non-linear functions, which together under a product make the output.

If there are 10 inputs, and three functions each with 4 parameters, then there are 30+12 parameters in the model. However, we need to first train a function to be the output probability distribution. The problem is the **function parameters depend on the sample**. But the linear sum parameters depend on the descriptors and the underlying model. Can we do a cheeky trick where we make up data to make the observation distribution exactly fit a well known statistical distribution?

We essentially take random moments of the values. These can be sampled using some special distribution. We need to find parameters such that the moments match the Mellin transforms at appropriate values. We must use the condition that the zeroth moment gives , to keep the probability distribution.

These give the coefficients of the model Meijer-G function. There is a nice relationship in <http://functions.wolfram.com/HypergeometricFunctions/MeijerG/22/04/> which gives the coefficients if the original distribution is a product of two others. This may not be much use in this case.

Once we have these coefficients

We can also consider the case where the output model is a KD estimation of Meijer functions, (potentially in a very generalise sense), this may be more complicated.

We can also consider the application of Ramanujan’s Master theorem. Which says the Mellin transform of a function that obeys certain criteria is related to the coefficient of the function’s series expansion

Then

Because we know the Mellin transform for the Meijer G function, then we also know it’s series expansion to an extent by reversal of the sign in the .

Derivatives of Log-Gamma, supposing we can come up with a distribution for y, possibly by fitting a polynomial times and converting to a Meijer G function, then to find the other function parameters we will need a variational algorithm which trains the parameters. We will then need to consider the loss function

We may also need to include a number of regularisation parameters to help enforce that all of the functions are probability distributions

If we take derivatives of this with respect to parameters, we find the derivative of log gamma is given by

For the digamma function

We will want to find good/quick approximations for this quantity. It is now relatively straightforward to find the parameters needed. It will require a descent method on a set of special functions.

The other required steps are how to vary the linear combinations of parameters used as inputs. Or whether this is required at all? Can we assume a separable model? What kind of inputs might we have. Assays? Consider the datasets that have been run with other ML methods?

We know the Mellin transforms are the moments, we can approximate these moments from the data.

There will need to be some kind of self consistent loop here? Start with as random, then work out some functions, then update alpha beta, then update functions? Does that work?

We need the functions, and we need the embeddings. We can extract the moments from the functions. We could kde approximate the functions? Sounds expensive. We will have a very complex manifold to train on.

I think to begin with, we should not assume a linear sum of variables.