Deep learning statistical models

D. Nychka, F. Gerber, M. Bailey and S. Sain*

Colorado School of Mines and *Jupiter

September 10, 2021

Outline

- The Gaussian CDF
- Climate model output
- Training Deep Nets on Gaussian Processes
- Comparison to Maximum likelihood
- Back to climate
- Hydrological models and the Generalized Pareto

Finding the normal CDF

$$\text{erf(x)} = F(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-u^2/2} du$$

and the tail probability

$$\text{erfc(x)} = 1 - \text{erf(x)}$$

Compute $F(.25)$

```
# in R
> pnorm( .25)
[1] 0.59870632568292
```

Finding the normal CDF

How is this really computed?
(try to find `pnorm.c` within the R source code)

$$\text{erf}\,(x) \simeq x R_{lm}(x^2)\,, \qquad |x| \leqq .5\,,$$
$$\text{erfc}\,(x) \simeq e^{-x^2} R_{lm}(x)\,, \qquad .46875 \leqq x \leqq 4.0\,,$$
$$\text{erfc}\,(x) \simeq \frac{e^{-x^2}}{x}\left\{\frac{1}{\sqrt{\pi}} + \frac{1}{x^2}\, R_{lm}(1/x^2)\right\}, \qquad x \geqq 4\,,$$

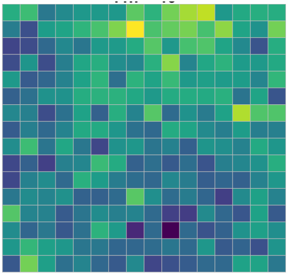
Cody, W. D. (1993)

$R_{lm}(x) = P_l(x)/Q_m(x)$ a ratio of a 5^{th} and a 4^{th} degree polynomial.
Accurate to 14 digits!

Approximation works – but it is mysterious!

Another computation

A 16×16 spatial field



$\rightarrow$ MLEs for covariance parameters

The MLE estimator in this case is the function

$$\mathcal{F}(\text{field}) \rightarrow \hat{\theta}, \hat{\lambda}$$

$$\Re^{256} \rightarrow \Re^{2}$$

In the `fields` package and for a Matern

```
fit <- spatialProcess( x, y, smoothness=.5)
```

Example of a statistical approximation

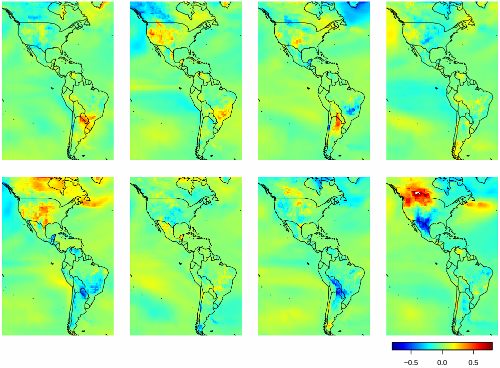
Use a deep network (CNN or dense network) to approximate "maximum likelihood estimates".

The $\mathcal{F}$!

- Covariance parameters for a Gaussian spatial process.
- Parameters of the extreme value distribution

Approximation works – but it is mysterious!

Local temperature sensitivity to global temperature
First 8 out of 30 *centered* ensemble members



Goal: Simulate additional fields efficiently that match the spatial dependence in this 30 member ensemble.

A Statistical Approach

- $\approx$ 13K grid boxes over N and S America
- Estimate a spatially varying covariance function by fitting stationary covariances to small windows. (16×16)
- Range and variance parameter for each window.
- Encode local estimates into a global model to simulate Gaussian random fields.
  This is Ashton Wiens Ph D work.

Train a convolution neural net (CNN) on the “image” to estimate covariance parameters.

Or train a dense neural net on the variogram

We found a speedup by a factor of 100!

Matérn covariance function

Covariance function:

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sigma^2 \text{Matern function }_{\nu}(d)$$

with $d = ||\boldsymbol{x}_1 - \boldsymbol{x}_2||/\theta$

- Matérn function is a modified Bessel function.
- Smoothness ν measures number of mean square derivatives and is equivalent to the polynomial tail behavior of the spectral density.
- θ is the range parameter.
- For $\nu = .5$: the classic exponential

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sigma^2 e^{-||\boldsymbol{x}_1 - \boldsymbol{x}_2||/\theta}$$

Observational model

$$Y(\boldsymbol{x}) = g(\boldsymbol{x}) + \boldsymbol{e}(\boldsymbol{x})$$

with g following a Gaussian process with Matérn covariance, variance σ^2 and $\boldsymbol{e}$ white noise, variance τ^2.

We are interested in maximum likelihood estimates for θ (range), σ^2 and τ^2.

- A useful short cut is to focus on θ and $\lambda = \sigma^2/\tau^2$

Can convert λ to equivalent degrees of freedom of the smoother for g.

- Analytical expresssions for MLEs of σ^2 and τ^2 based on MLE of λ.

Finding the MLEs

log Likelihood for covariance parameters.

$$= -\frac{\boldsymbol{y}^T(\sigma^2 C(\theta) + \tau^2 I)^{-1}\boldsymbol{y}}{2} - (1/2)ln|\sigma^2 C(\theta) + \tau^2 I| - (n/2)\ln(\pi)$$

or concentrating onto λ and θ

$$= -(n/2) - (1/2)ln|\hat{\sigma}^2(\theta, \lambda)(C(\theta) + \lambda)I| - (n/2)\ln(\pi)$$

- $C(\theta)$ correlation matrix for observations.
- No closed form for maximum.
- Often hard to find good starting values for optimization.
- Evaluating inverse and determinant can be time consuming.

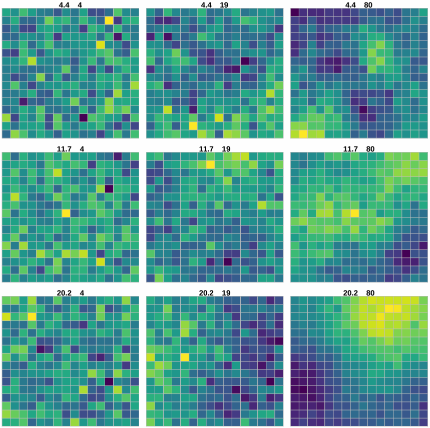
Examples of training fields

$$Y(\boldsymbol{x}) = g(\boldsymbol{x}) + \boldsymbol{e}$$

$\boldsymbol{x}$ on a 16×16 grid, $Var(Y) = 1$,

$\rightarrow$ decreasing noise (τ^2)

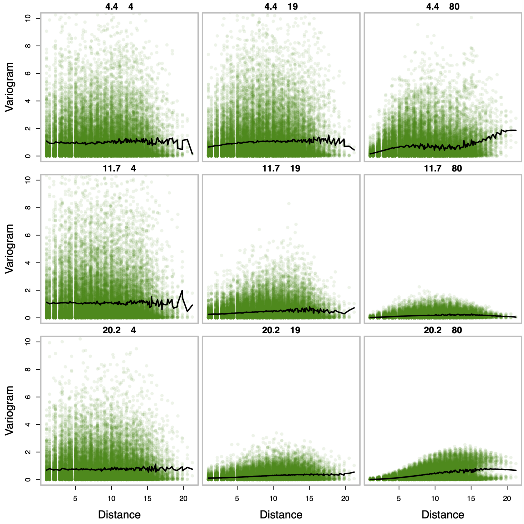
increasing range (θ) $\downarrow$



Variograms of training fields

$\rightarrow$ decreasing noise (τ^2)

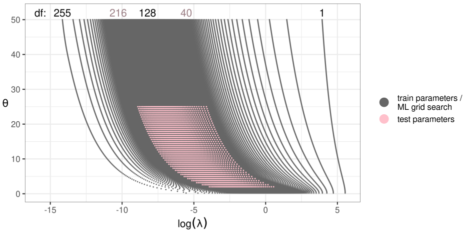
increasing range (θ) $\downarrow$



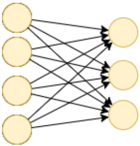
Neural net setup

- A neural network is a composition of many simple functions to approximate arbitrary functions.
- It depends on estimating many parameters (weights) based on a large training sample. Training / testing sets

  - Input are 16×16 Gaussian fields or their variograms
  - $200 \times 201 = 40200$ values in covariance parameter space
  - $\approx$ 1M fields generated for training.
  - Tested on 10K fields from 2000 parameter combinations.

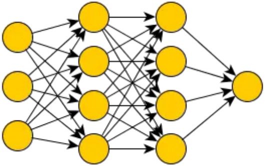


Inputs and artificial neurons



Three neurons with four available inputs:
Each neuron creates a linear combination of the inputs followed by a nonlinear transformation.

- Outputs from one layer become the inputs for another layer.
- Linear transformation is estimated (learned) for every neuron
- "Deep Learning" considers many neurons and mulitple layers.



Two hidden layers:

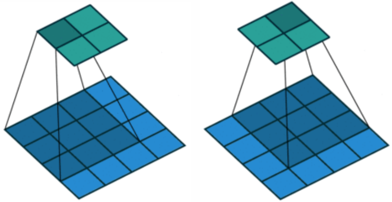
- $4 \times (3+1) = 16$ parameters in second layer
- $4 \times (4+1) = 20$ parameters in third layer
- $4 + 1 = 5$ parameters in output layer

See Neural Networks and Deep Learning, Michael Nielsen, for a good introduction

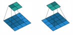
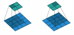
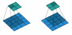
A convolution version CNN

Designed to work on images

A linear filter is applied to every 3×3 block of the input field followed by a nonlinear transformation.

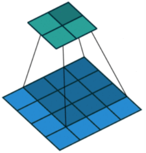


In this case the 5×5 image is reduced to a 2×2 output image.

- These filter results are then filtered again ... and again !
- Many filters (128) are considered. [] [] ... []
- Filter weights found by training (of course!).

Basic functional step

Where is the neuron?



$$y_j = \phi(b + w_1 x_1 + \ldots + w_9 x_9)$$

- $\{x_1, x_2, \ldots, x_9\}$ pixel values from 3×3 "image" (aka a tensor)

- y_j – the j^{th} pixel value for "image" (tensor) at next layer
- w the weights and b offset to be estimated/optimized
- $\phi(u) = u_+$ Rectified linear unit

Net architecture

Form for finding MLEs based on an image

Layer	Operation	Size
1	Input	16$\times$16 image
2	2D convolution	128 7$\times$7 filtered images
3	2D convolution	128 3 $\times$ 3 filtered images
4	2D convolution	128 outputs
5	dense	500 outputs
6	dense	2 outputs (θ and λ)

636K parameters!

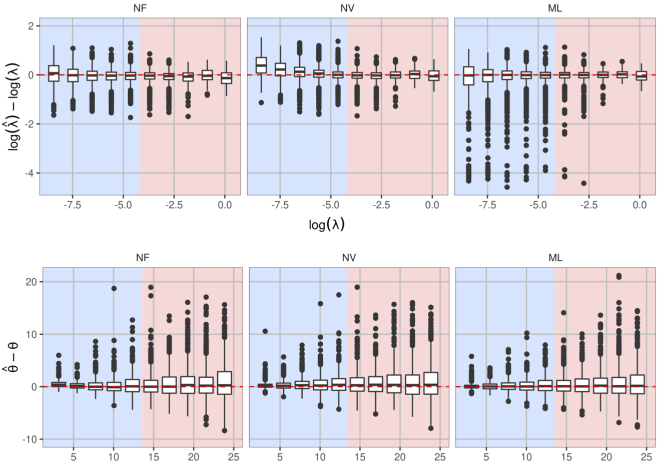
E.g at layer 5 there are 500 neurons each with 128 inputs, (128 $+$1)X 500 parameters

- Simpler 2 layer dense network used for finding MLEs based on a variogram

Parameter estimates on 10K test samples

- Recall that the MLEs are “optimal” for large n based on Cramer-Rao lower bound.

CNN Image NN Variogram MLEs



Results continued

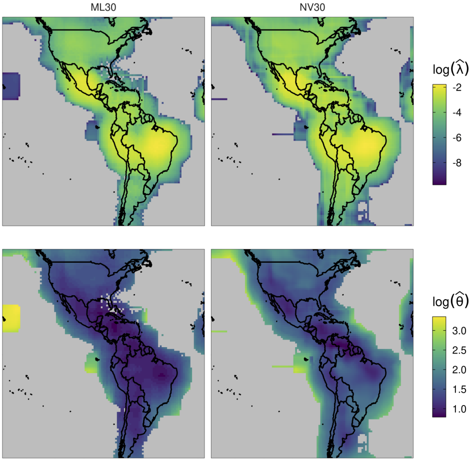
Training:

- CNN , NN, and MLE estimates tend to track the red lines (truth)
- CNN and NN overall has comparable accuracy to the MLEs
- Potential tradeoff between bias in CNN estimates and variance in MLEs

Climate model emulation

Estimated log Variance and range

MLE NN variogram



Timing

Training the network $2 - 6$ hours using cloud computing
(but this can be shortened for a slightly less accuracy)
For the climate model output

- 12,769 windowed estimates
- 10s of seconds using the CNN , 2 minutes using the NN variogram
- $\approx$ 1.5 hours using standard MLE fitting in R (`fields` package)

In general we find a factor of 100 or more speedup.

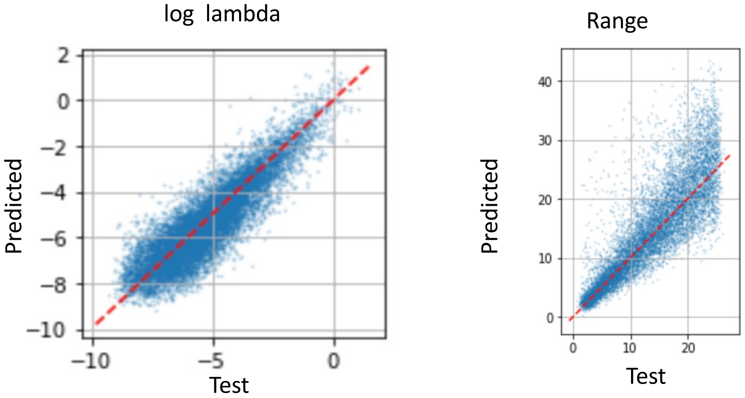
Part of this may be due to the efficiency of the tensor flow libraries and low level coding.

More about irregular spatial data

Build a deep net based on the variogram statistic.

Can the variogram serve as an approximate "sufficient statistic" for a stationary covariance function?

log parameter estimates for λ and the range



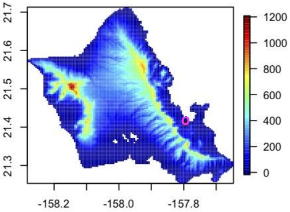
Loose ends

- How to adjust variogram NN for different bin counts.?
- Train on a SAR model (LatticeKrig, SPDE) directly instead of Matérn .
- Train for the likelihood *surface* instead of just the estimates.

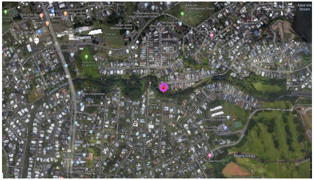
Hydrologic models and extremes

Steve Sain, *Jupiter* , Maggie Bailey *Mines*
Simulating flooding from extreme precipitation

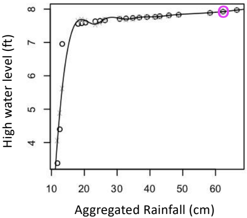
Elevations Oahu, Hawaii



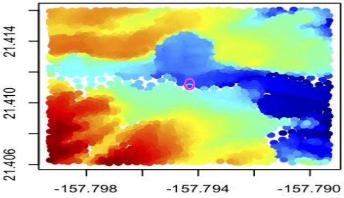
Aerial view of study region and grid cell



Grid cell high water response



Simulated high water field for 60 cm



Hydrologic models and extremes

Goal is to estimate the shape ($\boldsymbol{\xi}$) and scale parameters ($\boldsymbol{\sigma}$) of a generalized Pareto distribution at many (millions) of pixels.

$$f(x) = \begin{cases} (1/\boldsymbol{\sigma})(1 + \xi \frac{(x-\mu)}{\boldsymbol{\sigma}})^{-\frac{\boldsymbol{\xi}+1}{\boldsymbol{\xi}}} & \boldsymbol{\xi} \neq 0 \\ (1/\boldsymbol{\sigma}) e^{-\frac{(x-\mu)}{\boldsymbol{\sigma}}} & \boldsymbol{\xi} = 0 \end{cases}$$

and

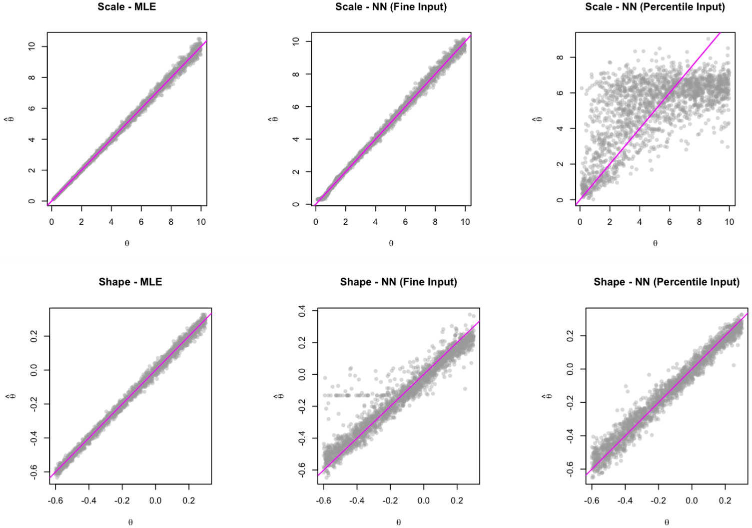
$$\begin{array}{ll} x \geq \mu & \boldsymbol{\xi} \geq 0 \\ \mu \leq x \leq \mu - (\boldsymbol{\sigma}/\boldsymbol{\xi}) & \boldsymbol{\xi} \leq 0 \end{array}$$

- Train a neural net on finely binned histograms to obtain estimates comparable to the MLEs.

- Potential speedup will allow for data analysis on high resolution model output.

Prelimnary results

Sample size of 2500, 8000/2000 cases for training/testing, $\mu = 1$ i.e. threshold is fixed.



Summary

- Exploit fast simulation of statistical samples to train a neural net.

- Neural nets can accurately reproduce statistical computations but evaluate much more quickly.

- Training and test samples provide a rigorous way to insure neural net approximations.

Thank you

An example of a Keras/R specification

- NxN image initial image, and 3X3 filters interspersed with max pooling reductions.
- Final step takes the last "image" and feeds to a dense neural network with 2 outputs

```
modelMatern11 %>%
  layer_conv_2d( 32, kernel_size=3, activation='relu',
                 input_shape=c(N,N,1) ) %>%
  layer_max_pooling_2d() %>%
  layer_conv_2d( 32, kernel_size=3, activation='relu') %>%
  layer_max_pooling_2d() %>%
  layer_flatten() %>% layer_dense(2)
```

Keras model summary

```
  > modelMatern11
Model
Model: "sequential"
________________________________________________________________________________________________
Layer (type)                             Output Shape                           Param #
================================================================================================
conv2d (Conv2D)                          (None, 14, 14, 32)                     320
________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)             (None, 7, 7, 32)                       0
________________________________________________________________________________________________
conv2d_1 (Conv2D)                        (None, 5, 5, 32)                       9248
________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)           (None, 2, 2, 32)                       0
________________________________________________________________________________________________
flatten (Flatten)                        (None, 128)                            0
________________________________________________________________________________________________
dense (Dense)                            (None, 2)                              258
================================================================================================
Total params: 9,826
Trainable params: 9,826
Non-trainable params: 0
________________________________________________________________________________________________
```